

PUBLIC
2025-06-20

SAP HANA Automated Predictive Library Developer Guide

© 2025 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 SAP HANA Automated Predictive Library Developer Guide. 8

2 What's New in SAP HANA APL 2514. 9

3 Administrator and Developer Tasks. 10

4 Getting Started as a Developer. 12

5 SAP HANA APL: Basic Concepts. 13

5.1 Model Types. 13

Time Series Modeling. 17

Data Comparison. 20

5.2 Functions. 22

5.3 Parallelization. 24

Parallel Apply. 24

Segmented Modeling. 26

Configuration Settings in the SAP HANA Cloud, SAP HANA Database. 30

5.4 Calling Functions. 32

Choosing a Technique. 33

Use Case: Training a Model. 34

Use Case: Debriefing a Model with a Testing Partition. 36

Function Availability. 38

5.5 Architecture Overview. 40

5.6 Compatibility Information. 41

6 Working with SAP HANA APL Functions. 42

6.1 Supported Database Objects. 42

6.2 Supported SQL Data Types. 43

6.3 Table Types. 45

Table Type Variants. 46

6.4 Calling a Function with the Direct Technique. 47

Connecting to the Database Schema. 48

Creating Table Types. 48

Generating an AFL Wrapper. 49

Creating Input and Output Tables. 51

Calling the AFL Wrapper. 51

6.5 Calling a Function with the Procedure Technique. 52

Connecting to the Database Schema. 53

Creating Input and Output Tables. . . . 53
Executing the Stored Procedure. . . . 54
Performance Considerations. . . . 55
Supporting SAP HANA Cloud Multitenancy. . . . 55
6.6 Calling a Function as an ANY Procedure. . . . 55
Connecting to the Database Schema. . . . 56
Creating Input and Ouput Tables. . . . 56
Running the Function. . . . 57
6.7 Getting Progress Information. . . . 58
6.8 Canceling the Execution of a Function. . . . 60
Cancellable Functions. . . . 61

7 Analyzing Data with SAP HANA APL Functions. . . . 62
7.1 Creating the Model. . . . 62
7.2 Training the Model. . . . 62
7.3 Applying the Model. . . . 63
7.4 Partition Strategies. . . . 64
7.5 Variable Interactions. . . . 67
7.6 Quantiles of Predicted Values. . . . 68
7.7 Interoperability between SAP HANA APL and Automated Analytics. . . . 69

8 Function Reference. . . . 72
8.1 Predictive Model Functions. . . . 72
APPLY_MODEL. . . . 72
APPLY_MODEL_AND_TEST. . . . 98
APPLY_MODEL_DEBRIEF. . . . 102
APPLY_RECO_MODEL. . . . 108
APPLY_RECOMMENDER_TO_BASKET 113
APPLY_RECOMMENDER_TO_USERS. . . . 117
APPLY_SOCIAL_MODEL. . . . 120
COMPUTE_CONFUSION_MATRIX. . . . 127
CREATE_MODEL. . . . 132
CREATE_MODEL_AND_TRAIN. . . . 137
CREATE_MODEL_AND_TRAIN (3 Datasets). . . . 142
CREATE_MODEL_AND_TRAIN (with Debrief Tables). . . . 148
CREATE_MODEL_AND_TRAIN_APPLY. . . . 153
CREATE_RECO_MODEL_AND_TRAIN. . . . 157
CREATE_RECOMMENDER. . . . 169
CREATE_SOCIAL_MODEL_AND_TRAIN. . . . 173
EXPORT_APPLY_CODE. . . . 183
EXPORT_APPLY_CODE_FOR_RECO. . . . 191
EXPORT_PROFITCURVES. . . . 196

EXPORT_VARIABLEDESCRIPTIONS. 199
GET_MODEL_DATASET_TYPES. 201
GET_MODEL_DEBRIEF. 203
GET_MODEL_INFO. 206
GET_TABLE_TYPE_FOR_APPLY. 213
GET_TABLE_TYPE_FOR_SOCIAL_APPLY. 216
GET_VARIABLE_INDICATORS. 222
IMPORT_MODEL. 224
IMPORT_VARIABLEDESCRIPTIONS. 226
PARALLEL_APPLY_MODEL. 228
PARALLEL_APPLY_RECO_MODEL. 231
PARALLEL_APPLY_SOCIAL_MODEL. 234
PUBLISH_MODEL. 239
RETRAIN_MODEL. 242
RETRAIN_MODEL (3 Datasets). 247
TEST_MODEL. 251
TEST_MODEL_DEBRIEF. 255
TRAIN_MODEL. 258
TRAIN_MODEL (3 Datasets). 264
UPDATE_MODEL. 270
8.2 Predictive Business Functions. 273
COMPARE_DATA. 273
DEBRIEF_APPLY_RESULT. 277
FORECAST. 281
FORECAST_AND_DEBRIEF. 291
GET_KEY_INFLUENCERS_FROM_MODEL. 300
KEY_INFLUENCERS 304
PROFILE_DATA. 307
PROFILE_DATA_AND_GET_ASSOCIATIONRULES. 314
RECOMMEND. 320
SCORING_EQUATION. 326
8.3 Technical Functions. 334
PING. 334
PROGRESS_CLEANUP. 335
8.4 Utility Stored Procedures. 336
8.5 Common APL Aliases for Model Training. 336

9 Table Type Reference. 353
9.1 ADMIN. 353
9.2 APPLY_OUT_TYPE. 354
9.3 ASSOCIATION_RULES. 361
9.4 CONTINUOUS_GROUPS. 362

9.5 DATASET. . . . 362
9.6 DEBRIEF_METRIC. . . . 364
9.7 DEBRIEF_PROPERTY . . . 365
9.8 FUNC_HEADER. . . . 365
9.9 INDICATORS. . . . 370
9.10 INFLUENCERS. . . . 380
9.11 INFO. . . . 380
9.12 LOG. . . . 381
9.13 MODEL. . . . 382
9.14 OPERATION_CONFIG. . . . 384
9.15 OTHER_GROUPS. . . . 385
9.16 OUTPUT_TABLE_TYPE. . . . 385
9.17 PROFITCURVES. . . . 386
9.18 PUBLISHING_SUMMARY. . . . 387
9.19 RESULT. . . . 387
9.20 RULE_ITEMS. . . . 388
9.21 SUMMARY. . . . 388
9.22 USER. . . . 396
9.23 VARIABLE_DESC. . . . 396
9.24 VARIABLE_ROLES. . . . 398
9.25 Table Types for Social Modeling. . . . 400
Social Models Serialization. . . . 400
ATTRIBUTES. . . . 405
GRAPH_FILTERS. . . . 406
GRAPH_INDICATORS. . . . 406
GRAPH_POST_PROCESSING. . . . 407
LINKS. . . . 410
NODES. . . . 411
RECO_SCORE. . . . 411
TRANSACTION. . . . 412
APPLY MODE for Social Modeling. . . . 412
Graph Types. . . . 413

10 Statistical Reports. . . . 414
10.1 Generic. . . . 417
List of Partitions. . . . 418
List of Variables. . . . 419
Category Frequencies of Variables and Partitions. . . . 420
Group Frequencies of Variables and Partitions. . . . 421
List of Continuous Variables. . . . 422
10.2 Target Types. . . . 423
Binary Target. . . . 424

Multiclass Target. . . . 431
Continuous Target. . . . 433
10.3 Time Series Models. . . . 436
Model Overview. . . . 437
Model Components. . . . 438
Model Outliers. . . . 439
Model Performance by Horizon. . . . 440
Model Performance. . . . 441
Model Change Points. . . . 443
Model Decomposition. . . . 444
Forecast Breakdown. . . . 445
Impact of Cycles. . . . 447
Impact of Numerical Influencers. . . . 448
Impact of Categorical Influencers. . . . 449
Variables Ranked by Contribution to Trend. . . . 451
Cyclic Variable Analysis on the Detrended Signal 452
10.4 Classification and Regression Models. . . . 453
List of Correlated Variable Pairs. . . . 453
Variables Ranked by Contribution to Explaining the Target. . . . 454
Variable Interaction Effect. . . . 456
Variable Interaction Matrix. . . . 457
List of Excluded Variables. . . . 458
Variable Selection Summary. . . . 460
Variable Selection Details. . . . 461
Variable Selection Performance. . . . 462
Binary Classification Confusion Matrix. . . . 463
Multinomial Classification Confusion Matrix. . . . 465
Multinomial Classification Performance by Class. . . . 466
Model Performance. . . . 467
10.5 Deviations. . . . 469
Deviations By Variable. . . . 470
Deviations By Category. . . . 471
Deviations By Group. . . . 472
Deviations Category Frequencies. . . . 473
Deviations Group Frequencies. . . . 474
Target Based Deviations By Category. . . . 476
Target Based Deviations By Group. . . . 477

11 Runtimes. . . . 479
11.1 C++ Runtime. . . . 479
11.2 Java Runtime. . . . 480
11.3 JavaScript Runtime. . . . 480

Load the JavaScript Runtime. . . . 481
Using the JavaScript Runtime. . . . 482

12 Sample Use Case: Predicting Auto Insurance Fraud 487
12.1 The Business Data. . . . 487
12.2 Overall Workflow. . . . 488
12.3 The Learning Phase. . . . 489
Training Inputs. . . . 489
Training Outputs. . . . 490
12.4 The Prediction Phase. . . . 492
Apply Inputs. . . . 492
Apply Outputs. . . . 493

13 Troubleshooting. . . . 495
13.1 Check if SAP HANA APL is Correctly Loaded. . . . 495
13.2 Script Error During Stored Procedure Creation. . . . 495
13.3 Error During Stored Procedure Execution. . . . 496
13.4 Supporting SAP HANA Cloud Multitenancy. . . . 497

1 SAP HANA Automated Predictive Library Developer Guide

This guide describes how to use the SAP HANA APL function library to build and apply predictive models on SAP HANA.

It includes the following:

- A *What's New in This Release* section
- Information about how to use the APL functions
- Which techniques are supported to call which functions
- A list of functions, stored procedures, and their parameters
- Troubleshooting information

Related Information

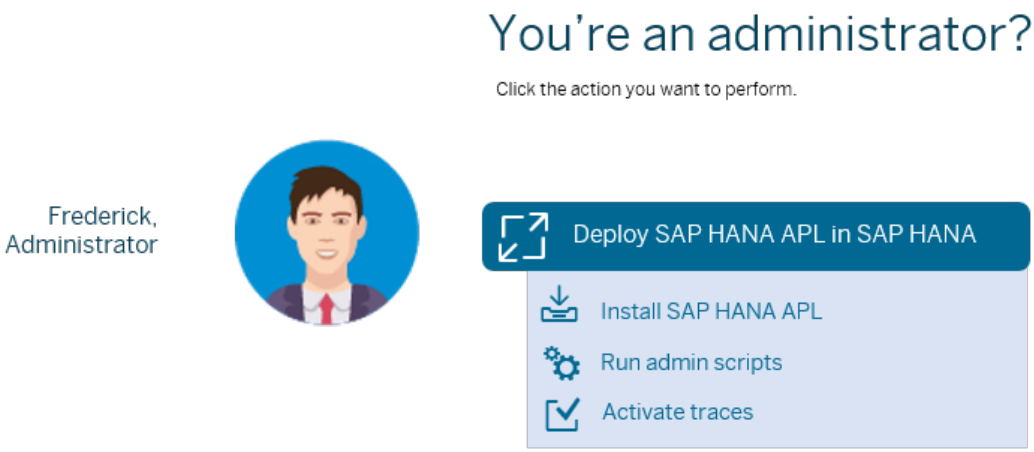
SAP HANA Academy - SAP HANA APL Videos

2 What's New in SAP HANA APL 2514

There are no new features or enhancements in SAP HANA Automated Predictive Library (APL) 2514.

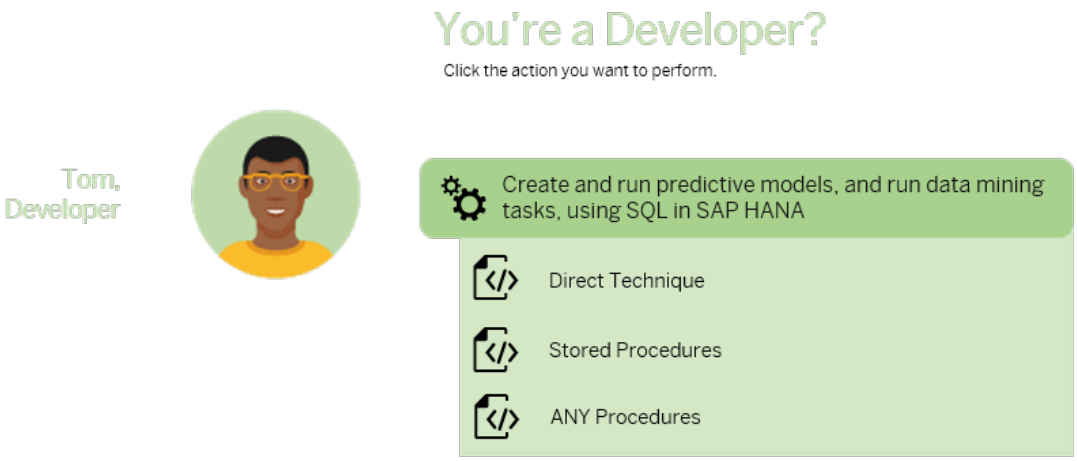
3 Administrator and Developer Tasks

Administrator Role



- SAP HANA Automated Predictive Library Installation Guide
- Installing SAP HANA APL
- Running Admin Scripts
- Activate the Traces

Developer Role



- Function Reference [page 72]
- Calling a Function with the Direct Technique [page 47]
- Calling a Function with the Procedure Technique [page 52]
- Calling a Function as an ANY Procedure [page 55]

4 Getting Started as a Developer

You're a developer and you want to use SAP HANA APL, but you don't know where to start? Watch this video to get advice.

Open this video in a new window

Related Information

SAP HANA APL GitHub Repository - Samples and Runtimes

5 SAP HANA APL: Basic Concepts

The goal of SAP HANA Automated Predictive Library (APL) is to expose the data mining capabilities in SAP HANA and to allow their usage through SQL.

SAP HANA APL is an application function library (AFL) which exposes the data mining capabilities of the Automated Analytics engine in SAP HANA through a set of functions. As a developer, you use these functions to develop a predictive modeling process that allows business analysts to answer simple questions about their customer datasets stored in SAP HANA. You can also use these functions through a set of stored procedures delivered as a part of SAP HANA APL. Any application that can use SAP HANA functions, be it on premise or in the cloud, can also use SAP HANA APL.

5.1 Model Types

SAP HANA APL lets you build and apply different types of predictive models, such as classification, regression, and time series forecasting models.

The following model types are supported:

- Binary classification and regression models
- Binary classification, multinomial classification, and regression models based on the gradient boosting algorithm
- Clustering models
- Time series analysis models
- Data Comparison
- Recommendation models
- Social network analysis models

Regression and Classification Models (Legacy)

SAP HANA APL builds a model (model type: `regression/classification`) that implements a mapping between a set of descriptive attributes (model inputs) and a target attribute (model output). This type of model belongs to the family of regression algorithms used for building predictive models. The embedded Automated Analytics engine uses a proprietary algorithm that is a derivation of a principle described by V. Vapnik as "Structural Risk Minimization". It builds predictive models from a training data set based on a business question. The returned models are expressed as a polynomial expression of the input numbers. The output model can be analyzed in terms of the attributes' contribution, which shows the relative importance of the inputs and is characterized by two indicators:

- Predictive Power (formerly known as KI): This is the quality indicator of the models generated by the Automated Analytics engine. It corresponds to the proportion of information contained in the target variable that the explanatory variables are able to explain.

- Predictive Confidence (formerly known as KR): This is the robustness indicator of the models generated by the Automated Analytics engine. It indicates the capacity of the model to achieve the same performance when it is applied to a new data set exhibiting the same characteristics as the training dataset.

Regression Models (Gradient Boosting)

SAP HANA APL builds a model (model type: `regression`) that implements a mapping between a set of descriptive attributes (model inputs) and a continuous target attribute (model output). The type of model used for regression relies on the gradient boosting technique, which combines base learners sequentially. At each iteration step, the last learner attempts to fix the residual difference between the actual and predicted values left by the preceding learners. The prediction performance is mainly measured by the Root Mean Square Error, Mean Absolute Error, and Mean Absolute Percentage Error metrics, which quantify the gap between two numerical series (predictive and actual). The regression model type provides a more accurate modeling result than the legacy `regression/classification` model type, typically when input variable interaction is significant in terms of explaining the target values.

Binary Classification Models (Gradient Boosting)

SAP HANA APL builds a model (model type: `binary classification`) that implements a mapping between a set of descriptive attributes (model inputs) and a nominal target attribute (model output) with two classes. The type of model used for binary classification relies on the gradient boosting technique, which combines base learners sequentially. At each iteration step, the last learner attempts to fix the residual difference between the actual and predicted values left by the preceding learners. The prediction performance is mainly measured by the Predictive Power (formerly known as KI) and the Prediction Confidence (formerly known as KR). The binary classification model type provides a more accurate modeling result than the legacy `regression/classification` model type, typically when input variable interaction is significant in terms of explaining the target values.

Multinomial Classification Models (Gradient Boosting)

SAP HANA APL builds a model (model type: `multiclass`) that implements a mapping between a set of descriptive attributes (model inputs) and a target attribute (model output) with more than two classes. The type of model used for multinomial classification relies on the gradient boosting technique, which combines base learners sequentially. At each iteration step, the last learner attempts to fix the residual misclassification error left by the preceding learners. This ensemble model produces probabilities for each target class. The predicted class corresponds to the class with the highest probability.

The classification performance is mainly measured by standard precision, recall, and F1-score metrics that quantify the classification correctness. The number of classes has an influence on model size and thus on training time. If the target variable has N classes, the model training will be N-2 times slower than the training on a binary classification model.

The maximum number of distinct classes supported for the target of a multinomial model is 100.

Clustering Models

APL builds a model (model type: `clustering`) that implements a mapping between a set of descriptive attributes (model inputs) and the id (model output) of one of several clusters computed by the system. It belongs to the family of clustering algorithms used for building descriptive models. The goal of these models is to gather similar data in the same cluster. The Automated Analytics engine uses a K-Means engine to compute the cluster index output variable. K-Means is a method for finding clusters of similar individuals within a population.

The K-Means method proceeds as follows: Starting from an initial position of would-be centers of clusters, the method associates the closest individual with each center, which leads to a first definition of the clusters. The positions of the centers are then adjusted to the true central positions within the clusters. The new positions are then used to recompute the closest individuals, and the process is restarted. This is repeated iteratively until the centers land in a stable position. In practice, the process converges very quickly to a stable configuration.

Time Series Models

SAP HANA APL lets you build predictive models (model type: `timeseries`) from data that represents a time series. Time series models allow you to identify and understand the nature of the phenomenon represented by a sequence of measures (that is, a time series). They also allow you to forecast the evolution of the time series in the short and medium term and, therefore, predict its future values.

In a time series, a signal is considered as the combination of at least one of the three following basic components:

- One trend: The trend represents the evolution of a time series over the period analyzed. The trend is represented either by a function of time or by signal differentiation, which is calculated using the principle that a value can be predicted well enough based on the previous known value. Calculating the trend allows a stationary representation of the time series to be built (that is, the time series does not increase or decrease any more). This stationary representation is essential for the analysis of the two other components.
- One or more cycles: The cyclicity describes the recurrence of a variation in the signal. It is important to distinguish calendar time from natural time. These two time representations are often out of phase. The former - which is referred to as seasonality - represents dates (day, month, year, and so on), while the latter - which is referred to as periodicity - represents a continuous time (1, 2, 3, and so on).
- One fluctuation: Fluctuations represent disturbances that affect a time series. In other words, a time series does not only depend on external factors but also on its last states (memory phenomena). Parts of the fluctuations are explained by modeling them on past values of the time series (ARMA or GARCH models).

Another important aspect of time series modeling is to make forecasts. A time series model uses its own prediction in order to predict the next value.

For more information about the basic components of a signal, see Time Series Modeling [page 17].

Data Comparison

SAP HANA APL data comparison is about detecting potential deviations of a given dataset from a reference dataset.

For more informaiton, see Data Comparison [page 20].

Recommendation Models

For recommendation models (model type: `recommendation`), the recommendation engine aims to produce from transactional data (containing typically quantified or weighted relations from a user entity to an item or product entity) a list of suggested items or products for each submitted user. The underlying training model is based on the well-known "assocation rule" principles, consisting of determining the set of most signficant inference rules from an item set (namely antecedents) to another item. Each rule ({A,B} => C) has a confidence measure defined in percentage. This indicator corresponds to the conditional probability that C occurs when {A,B} are realized and is evaluated merely on the quantification of the item set occurrence. The SAP HANA APL RECOMMEND function performs in a single call both the model training and a subsequent apply on the passed-in list of users without intermediate model serialization.

> ### Note
>
> The list of users to score should be part of the transactional data used to train the model.

> ### Note
>
> The procedure EXPORT_APPLY_CODE_FOR_RECO requires that the link table input is named KXLINKS1 in the SAP HANA system.

Statistic Builder Models

Statistic builder models (model type: `statbuilder`) are specific to SAP Predictive Analytics and allow you to generate statistics from input datasets. You can use this model to compute the performance of a classification/ regression model from its apply result.

Related Information

Functions [page 22]

5.1.1 Time Series Modeling

SAP HANA APL time series modeling is about finding the internal structure within a sequence of data points ordered by time. When that structure is known, the temporal measurement can be extrapolated in the forecast horizon.

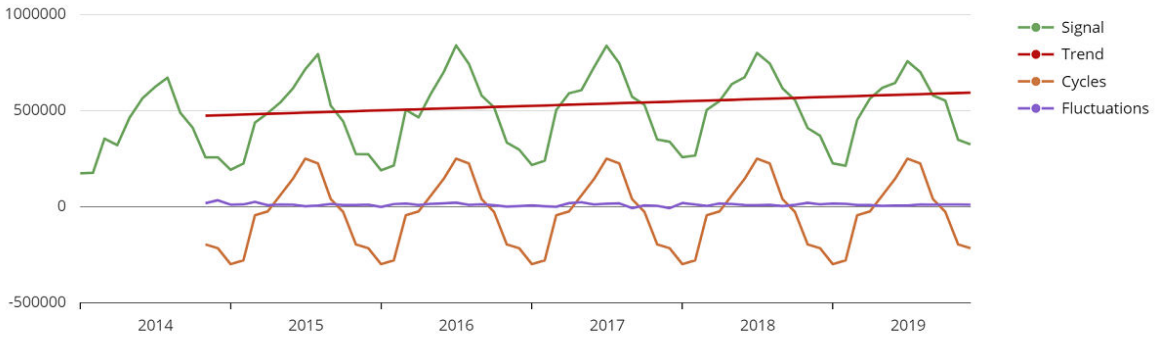
To build a time series model, SAP HANA APL decomposes the signal (that is, the observed series) into additive components in the following way:

Signal = Trend + Cycles + Fluctuations + Residuals

The components are described below:

Component	Mandatory	Description
Trend	Yes	The trend is the general orientation of the signal.
Cycles	No	The model can detect fixed-length cycles or seasonal cycles (based on a calendar). Up to 3 periodic elements can be found in the signal.
Fluctuations	No	Fluctuations represent the part of the time series detected by the model that is dependent on past values of the signal.
Residuals	Yes	Residuals refer to what is left when the trend, the cycles, and the fluctuations have been extracted from the signal. They reflect the part of the time series that SAP HANA APL cannot explain or model.

The example below shows a time series signal together with all its components, except the residuals:



For each component, several candidates are tested. The best combination of candidates, measured by Horizon Wide MAE (Mean Absolute Error) criteria, is kept as the final decomposition of the signal. When several combinations are found with similar MAE values (within the range of 5%), a criterion of simplicity is used to elect the best one.

The trend and cycle components support predictors other than the date if a dataset contains additional information, such as holiday, temperature, sporting event, policy change, and so on. These predictors are referred to as influencers or as extra predictable variables. Their values must be provided for the past periods but also for the future periods of the forecast horizon. Extra predictable variables may help make the forecast more accurate. Which influencers are kept by the model can be seen in the statistical report `TimeSeries_Decomposition`.

During the modeling process, SAP HANA APL identifies potential outliers. A value in the signal qualifies as an outlier if its corresponding forecasting error is considered to be abnormal relative to the average of the

forecasting errors on the estimation partition (the first 75% of the time-ordered dataset). The forecasting error is the absolute difference between the actual and predicted values. The forecasting error is said to be abnormal when its value is over three standard deviations. The outliers found by the model are listed in the statistical report `TimeSeries_Outliers`.

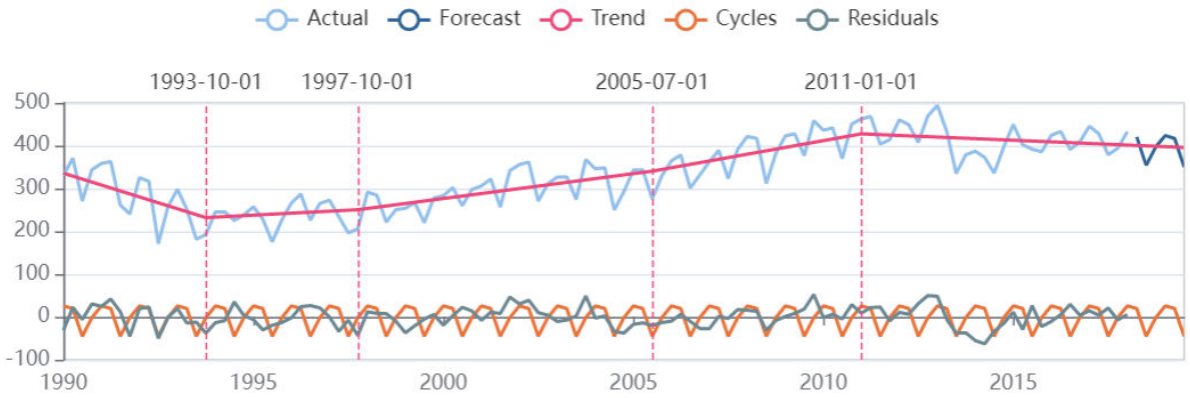
Basic Components

The candidates for the basic components are described briefly below.

Trend

Trend Type	Description
Polynom(ExtraPredictables)	A curve corresponding to the detected polynomial. A polynomial is modeled using Classification/Regression.
Linear(Time), Linear(Time, ExtraPredictables)	A straight line. It is a special case of `Polynom`.
Piecewise Linear Trend(Time), Piecewise Linear Trend(Time, ExtraPredictables)	Change points, that is, abrupt changes in the global trend, are detected and a piecewise linear trend is extracted as the resulting curve. See the example below the table.
Simple Exponential Smoothing	The curve is the smoothed signal value with the newest data weighted more than the oldest data, with its influence decreasing exponentially.
Double Exponential Smoothing	In addition to the smoothed signal value from simple exponential smoothing, the smoothed slope value is considered with the newest data weighted more than the oldest data, with its influence decreasing exponentially.
Triple Exponential Smoothing	In addition to the smoothed signal value and slope from double exponential smoothing, the smoothed seasonal values are considered with the newest data weighted more than the oldest data, with its influence decreasing exponentially. Triple exponential smoothing leverages the calendar to model series with irregular time intervals (for example, daily series without weekend days, monthly series without a particular month) as well as series with seasons of variable size, like day of year, week of year, week of month. For other cases, triple exponential smoothing uses fixed-length cycles.
L1 / Lag 1	The signal moved one step backward. This is the basic forecast where the predicted observation equals the latest signal observation.

The following example shows a piecewise linear trend with four change points (represented as vertical dotted lines):



Cycle

Cycle Type	Description
Periodic(n)	A cycle not depending on the date, where n is the size of the cycle in number of periods (default maximum value: 200).
Periodic(ExtraPredictables)	A cycle from an additional predictor.
Seasonal	The following seasonal variables are automatically detected: semesterOfYear quarterOfYear monthOfQuarter monthOfSemester halfMonthOfYear dayOfYear dayOfMonth dayOfWeek monthOfYear weekOfYear weekOfMonth hourOfDay minuteOfHour secondOfMinute

Fluctuation

Fluctuation Type	Description
AutoRegression(2)	Fluctuation uses an autoregressive technique to model changes not captured by the previous components.

Related Information

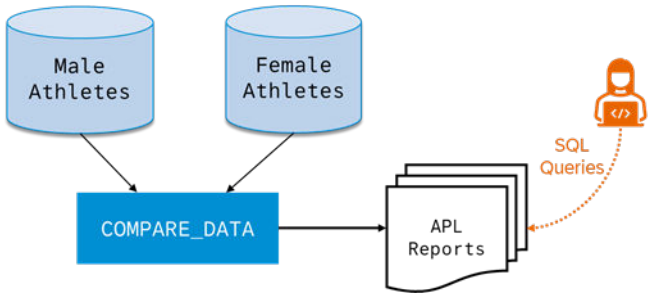
Model Decomposition [page 444]
Model Outliers [page 439]
Standard deviation (Wikipedia)

5.1.2 Data Comparison

SAP HANA APL data comparison detects potential deviations of a given dataset from a reference dataset.

There are couple of ways to compare datasets in APL. The first is to run the COMPARE_DATA function. For example, you could compare male Olympics athletes to female Olympics athletes to get results in the deviation report for each of the following input variable: age, height, medal, weight, and so on.

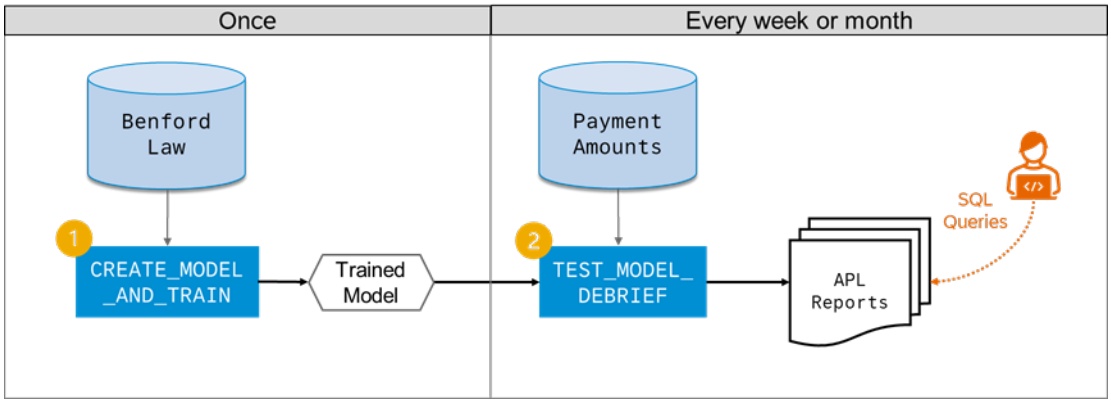
For more information, see COMPARE_DATA [page 273].



To do a comparison by Country or by Season (Winter/Summer), specify Country or Season as a segment with the parameter `APL/SegmentColumnName`.

To compare, on a regular basis, a changing dataset to a fixed reference, use the CREATE_MODEL_AND_TRAIN function instead of COMPARE_DATA to set the parameter `APL/ModelType` to `statbuilder`. A model will be persisted with the statistics computed on the reference dataset.

If your scenario is to detect financial fraud and errors, the reference can be the Benford's Law for leading digits. Then run the function TEST_MODEL_DEBRIEF with a second dataset. For example, the first digits of payment amounts.



Deviating Variables

The variables likely to deviate are listed in the Deviations By Variable [page 470] report.

This report has the following built-in filter:

```
Deviation_ Threshold > 0.95
```

The threshold value can be changed when calling the function as shown below:

```
select * from SAP_PA_APL."sap.pa.apl.debrief.report::Deviation_ByVariable"
(:out_property, :out_metric, Deviation_Threshold => 0.90);
```

The deviation indicator ranges from 0 to 1. The value 1 means that the distributions of the two datasets are very likely to be different. APL uses a chi-squared test to compute the deviation indicator.

Pair-wise comparison

Take a card games with 26 black clubs, 26 black spades, 26 red diamonds, 26 red hearts, and compare it to a non-standard deck with 26 red clubs, 26 red spades, 26 black diamonds, 26 black hearts.

None of the variables, Suit (Club/Spade/Diamond/Heart) and Color (Black/Red), appear as deviating because each variable is tested separately by APL. For a pair-wise comparison, specify the variable Color as a target, and check the Target Based Deviations By Category [page 476] report.

As an alternative, set Color as the segment. Suit will appear in the Deviations By Variable [page 470] report.

Frequency Reports

To view the two datasets side-by-side use the Deviations Category Frequencies [page 473] report. This report computes by category (categories results from a binning operation done by APL) the percentage change between the Reference dataset and the other dataset. The most changing categories (positive or negative change) are presented first. This report does not use the deviation indicator and has no built-in filter.

A more compact view using grouped categories is available in the Deviations By Group [page 472] report. The group report is empty when using the `statbuilder` model type. To get groups, set the model type to `variable-encoder`.

Related Information

Deviations [page 469]

5.2 Functions

SAP HANA APL functions are divided into three sets of functions: predictive model functions, predictive business functions, and technical functions.

- Predictive model functions
  The predictive model functions focus on predictive modeling activities. They are used to manipulate a predictive model, which can be used both as an input and an output parameter of these functions. Depending on its configuration and state, a predictive model only allows a subset of actions. For example, you cannot apply a model that has not yet been trained.
  You can use the predictive model functions to do the following:
  - Create and train models to obtain the best model for analyzing new data
  - Apply models to datasets to analyze the content and make decisions
  - Query models, that is, retrieve variable descriptions and statistics, model indicators, profit curves, and reporting information
- Predictive business functions
  The predictive business functions focus on specific data-mining workflows and do not need to materialize a predictive model, although one is used internally. These functions are usually executed in one single call, that is, they work on a dataset and produce results, without the need to materialize the underlying predictive model.
  You can use the predictive business functions to apply higher-level and simpler predictive functions to data, without materializing a predictive model.
- Technical functions
  You can use the technical functions to retrieve technical information about the APL component.

Predictive Model Functions

The predictive model functions allow you to implement a predictive modeling process, which involves various activities that change the model:

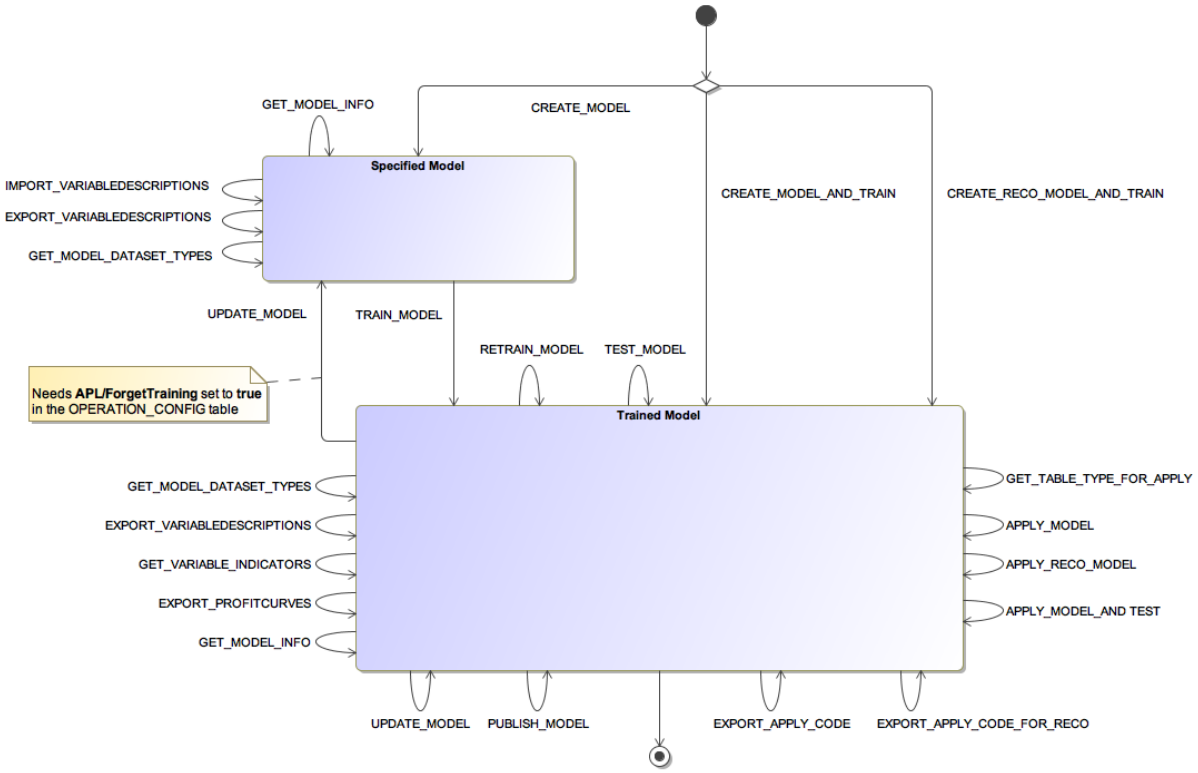
- Creating a model and describing its datasets
- Training and generating a model
- Running or applying the model

The predictive model functions are stateless, which means that the following applies:

- Any function that works on a model requires this model to be passed as an input parameter.
- Any function that updates a model produces a new model as an output parameter.
- In a full modeling process, the model needs to be passed from function to function.

The predictive model has a state, which can be updated by each function. For example, a model that has just been created is different from a model that has already been trained. This also means that you cannot run any function on any model. For example, a model that has just been created cannot be applied; it needs to be trained first.

The figure below depicts the different model states in SAP HANA APL and shows which functions can be called to transition between these states. A function called on a model that is not in the expected state will fail:



Predictive Business Functions

The predictive business functions are more straightforward. You generally only need to provide a dataset and some configuration parameters, and you can then call the appropriate function to get the expected results, without worrying about a predictive model. As a consequence, the state diagram for these functions is extremely simple, since there is no state at all. With just one single call, you can trigger the data-mining operation you need and get the expected results.

Related Information

Predictive Model Functions [page 72]
Predictive Business Functions [page 273]
Technical Functions [page 334]
Model Types [page 13]

5.3 Parallelization

SAP HANA APL supports parallelism based on SAP HANA tasks and the SAP HANA workload mechanism. This allows you to benefit from the parallel features and infrastructure of the SAP HANA database.

Parallelism is supported in the following ways:

- You can use parallelism when applying a single model on a large table, where several chunks of the apply table are computed in parallel.
- You can use segmented modeling, where several segments are processsed in parallel.

The use of the parallel apply process and the segmented modeling process is fully integrated in the SAP HANA infrastructure. The maximum number of tasks to be used in the process is defined in the function header parameter `MaxTasks`, which indicates the expected maximum number of tasks. The number of tasks that is actually used is always under the control of the SAP HANA workload manager:

- The actual number of tasks is limited by the number of cores of the current SAP HANA instances.
- If defined, the current workload class for the current user or the current user session will limit the actual value of the `MaxTasks` parameter.
- Depending on the current global workload and priorities of other jobs, the SAP HANA workload manager can limit the actual value of the `MaxTasks` parameter.

> **Note**
>
> The SAP HANA Cloud, SAP HANA database default configuration limits the number of concurrent tasks for an SQL statement to 25% of the number of CPUs, with a minimum of 4. This has an impact on the level of parallelism that can be used in the parallel apply process and in segmented modeling. For more information about strategies for managing this configuration, see *Configuration Settings in the SAP HANA Cloud, SAP HANA Database*.

For more information about SAP HANA workload classes, see *Managing Workload with Workload Classes*.

Related Information

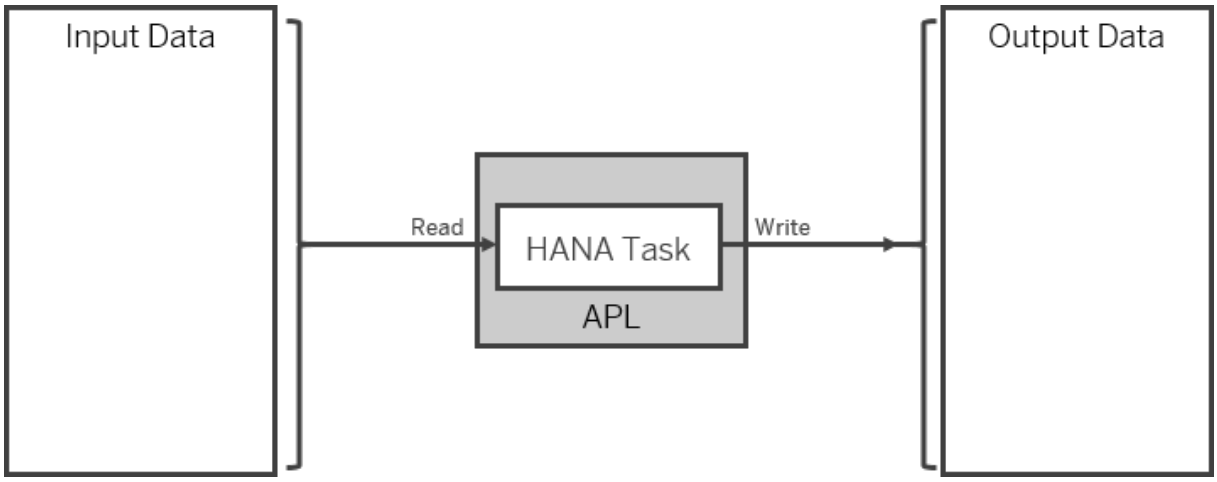
Managing Workload with Workload Classes

5.3.1 Parallel Apply

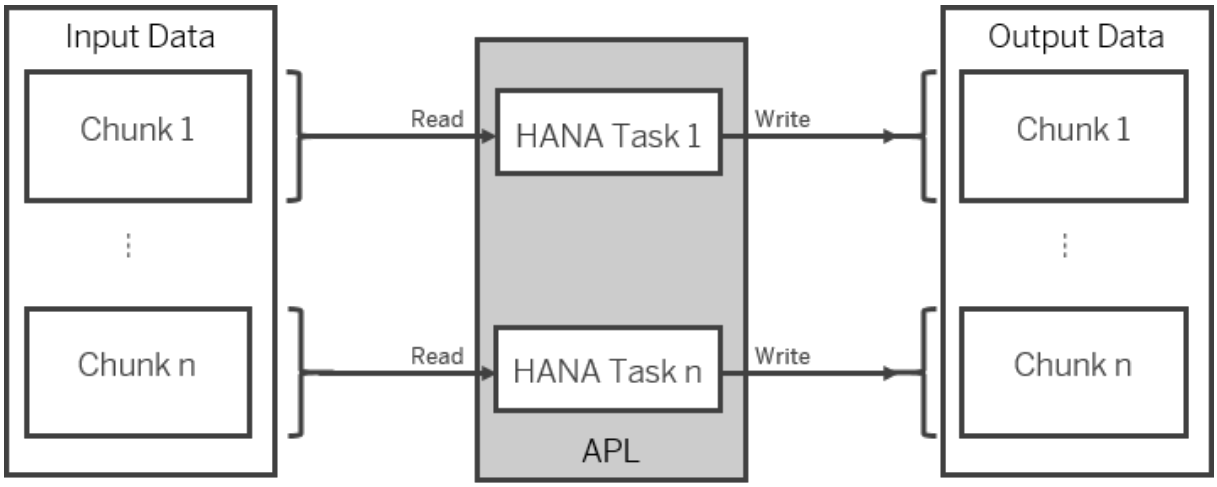
The parallel apply mode allows you to score datasets in parallel. This can be beneficial because when a trained model is used to output predictions ('scoring'), this is typically done on a much larger input dataset than the one used for training.

In parallel mode, several tasks under the control of SAP HANA are created by SAP HANA APL. Each task processes a chunk of the input dataset and all tasks are run in parallel. As a result, the output dataset can be produced more quickly.

In default apply mode, one task is used as illustrated below:



In parallel apply mode, n tasks are used in parallel as shown below:



Parallel mode is supported for the APPLY_MODEL function and is available for classification, regression, and multinomial models, as well as clustering models.

The maximum number of tasks to be used in the apply process can be defined in the function header parameter `MaxTasks`. For more information about the meaning of `MaxTasks` and the level of parallelism applied, see *Parallelization*.

When available, the summary output table provides information about the CPU time used by each task as well as the global time for the apply process.

Related Information

APPLY_MODEL [page 72]
FUNC_HEADER [page 365]
SUMMARY [page 388]
Example: Parallel Apply PROCEDURE [page 96]

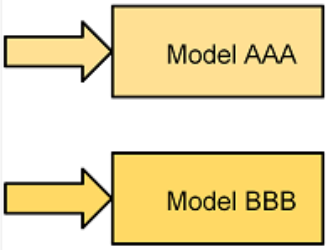
Parallelization [page 24]
Configuration Settings in the SAP HANA Cloud, SAP HANA Database [page 30]

5.3.2 Segmented Modeling

Generally, an input dataset contains the data for one model, so one model is trained. Segmented modeling, however, partitions the input dataset into different segments, where each segment produces its own distinct model.

An example is shown below:

RegionID	Education	Income	Age	Customer
BBB	2	$300	35	T
AAA	2	$100	22	F
BBB	5	$400	30	T
BBB	3	$600	60	T
AAA	9	$500	21	F
AAA	3	$300	32	F



Segmented modeling is supported for the following models only:

- Time series
- Regression or binary classification with gradient boosting
- Multinomial classification
- Regression or binary classification
- Clustering

The segmentation process is an automated process that requires only a minimal setup in order to manage (train, apply, debrief, and so on) multiple models with standard APL calls. Importantly, since the number of segments is typically high, learning or applying on all segments can be done in parallel using the parallel features and infrastructure of the SAP HANA database.

Input Dataset

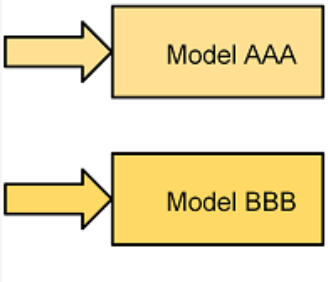
For segmented modeling, the input dataset must meet the following requirements:

- It must have a column to store a segment ID, which is a unique ID related to each segment.
  - The column name can be freely defined but must comply with the limitations described under *SAP HANA Table and Column Names* in the restrictions section of the SAP Note of the SAP HANA APL release (see https://help.sap.com/docs/apl).
  - The column can have any type.
  - The segment ID must be stored in one column. Multiple columns are not supported.

    - The column can have any position in the table.
- The input dataset must be ordered by segment ID so that all related records are contiguous in the input dataset.

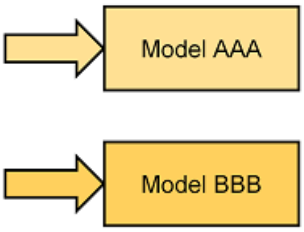
An input dataset ordered by segment ID is shown in the example below:

RegionID	Education	Income	Age	Customer
AAA	2	$100	22	F
AAA	9	$500	21	F
AAA	3	$300	32	F
BBB	2	$300	35	T
BBB	5	$400	30	T
BBB	3	$600	60	T



> **Note**
>
> For time series, the standard constraint on time series input data is still valid, which means that the input dataset must be ordered by segment ID and timestamp so that all related records are contiguous in the input dataset.
>
> An input dataset ordered by segment ID and timestamp is shown in the example below:

StoreID	Date	Revenue
AAA	03 Apr 2018	$100
AAA	04 Apr 2018	$500
AAA	05 Apr 2018	$300
BBB	03 Apr 2018	$300
BBB	04 Apr 2018	$400
BBB	05 Apr 2018	$600



APL Functions

The following SAP HANA APL functions support segmented modeling:

- FORECAST and FORECAST_AND_DEBRIEF
- CREATE_MODEL_AND_TRAIN and CREATE_MODEL_AND_TRAIN (with Debrief Tables)
- GET_TABLE_TYPE_FOR_APPLY
- GET_MODEL_DEBRIEF
- APPLY_MODEL
- TEST_MODEL
- GET_MODEL_INFO
- EXPORT_APPLY_CODE

The alias `APL/SegmentColumnName` is used to specify the name of the column that stores the segment ID.

The column that stores the segment ID information is a technical column that identifies the current segment but is never processed and is not even seen by the training process. This column must not be added to the variable descriptions or to the variable roles. If variable descriptions and variable roles are provided, they must be the same as the ones used in standard scoring.

The maximum number of tasks to be used to process the set of segments in parallel can be defined in the function header parameter `MaxTasks`. For more information about the meaning of `MaxTasks` and the level of parallelism applied, see *Parallelization*.

Output Dataset

The scoring functions that support segmented mode (FORECAST, FORECAST_AND_DEBRIEF, APPLY_MODEL, and GET_TABLE_TYPE_FOR_APPLY) produce an output dataset containing the predicted values with the same set of columns as in regular calls. In addition, the output dataset contains a segment column:

- The name of the segment column is the name specified with the alias `APL/SegmentColumnName`.
- The type must be the same type as the segment column of the input dataset. For (N)VARCHAR types, the length can be greater but must not be smaller.
- Its values are the segment IDs of all the segments.

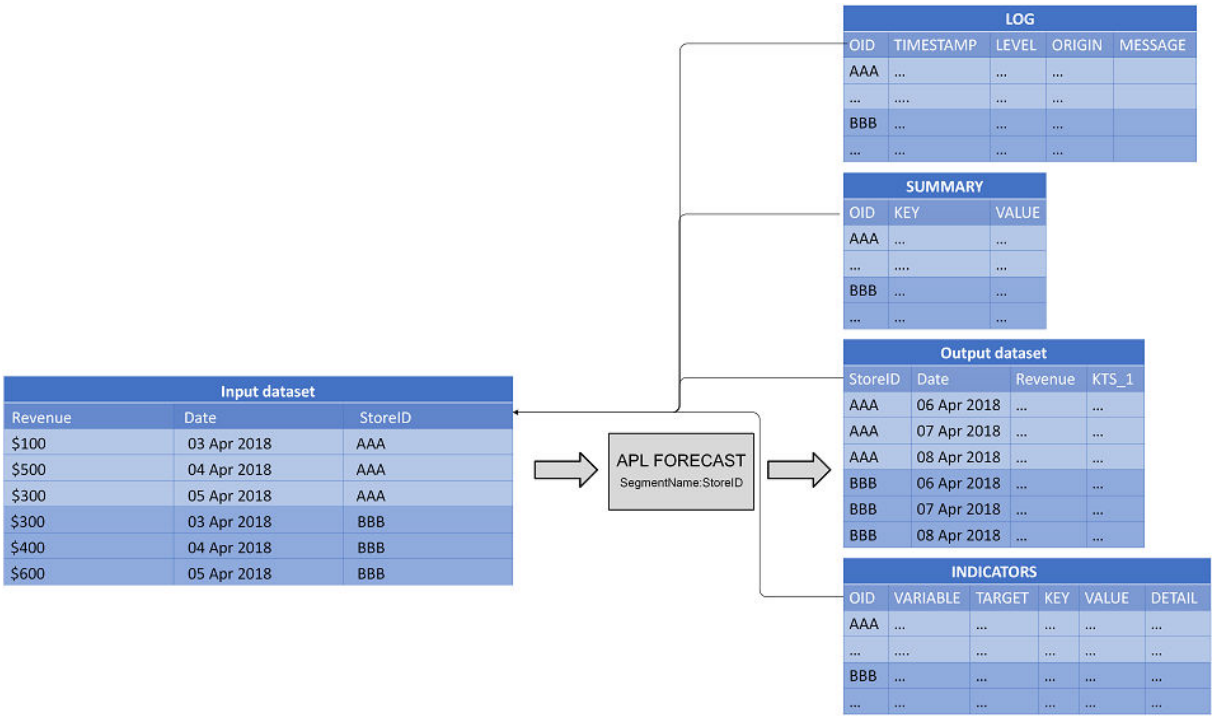
The output dataset is common to all models. Therefore, its layout (number, name, and type of columns) must be the same for all segments. The `APL/ApplyExtraMode` options, which dynamically generate fields depending on the actual values found in the input dataset, cannot be prechecked before a segmented apply process. If used, an unsupported `APL/ApplyExtraMode` value triggers an error before the scoring is done. For more information about the supported values, see *APL/ApplyExtraMode with Segmented Modeling*.

Other Output Information

With segmented modeling, all other tables produced by the scoring functions (FORECAST, FORECAST_AND_DEBRIEF, APPLY_MODEL, and GET_TABLE_TYPE_FOR_APPLY), that is, SUMMARY, LOGS, INDICATORS, DEBRIEF_METRIC, and DEBRIEF_PROPERTY, have the same structure as in standard modeling. Only the content of the OID (operational ID) field of these tables is different:

- Standard modeling: The OID field is a user-defined value set with the `Oid` parameter in the function header and is transported to all output tables other than the output dataset.
- Segmented modeling: For generic information valid for all segments, the OID field is the user-defined value set with the `Oid` parameter of the function header. For information related to a specific segment, the OID field is the value of the current segment ID. It overrides the global user-defined value coming from the `Oid` parameter in the function header.
  For example, in the summary table:
  - The global time of the scoring process is global information and is tagged with the global `Oid` parameter.
  - The succeeded or failed status provided for each segment is tagged with the segment ID.
  - The CPU time used for each segment is tagged with the segment ID.

An output dataset and output tables are shown in the FORECAST example below:



When available, the summary output table provides information about the CPU time used for each segment as well as the global time for the scoring process.

Limitations

- The actual values of the segment IDs are used in logs as an NVARCHAR(50) data type to identify the logs related to each segment. This means that the textual representation of the segment ID must not exceed 50 characters in length.
- Segmented modeling is supported only with the model formats `bin` (default value) and `zbin`. For more information, see *FUNC_HEADER* (`ModelFormat` parameter).

Related Information

FUNC_HEADER [page 365]
SUMMARY [page 388]
APL/ApplyExtraMode with Segmented Modeling [page 93]
Example: Segmented Forecast PROCEDURE [page 298]

5.3.3 Configuration Settings in the SAP HANA Cloud, SAP HANA Database

The SAP HANA Cloud, SAP HANA database default configuration limits the number of concurrent tasks for an SQL statement to 25% of the number of CPUs, with a minimum of 4.

With the default configuration of the SAP HANA Cloud, SAP HANA database, this means, for example, that on a server with 16 CPUs, segmented modeling can only process a maximum of 4 segments in parallel.

There are several strategies to manage this specific configuration parameter in SAP HANA Cloud:

- Alter the global configuration of the instance
- Alter the configuration for a given user
- Use the workload mechanism

Altering the Global Configuration

You can use the parameter `default_statement_concurrency_limit` in the `[execution]` section of the `global.ini` configuration file to manage parallel processing. The possible values are:

Parameter Value	Description
-1 (default)	Maximum number of concurrent tasks is 25% of the number of CPUs, with a minimum of 4.
0	There is no explicit concurrency limitation.
1	Number of CPUs

The SAP HANA cockpit is the preferred tool for changing the configuration, but you can also change it using the SQL statement ALTER SYSTEM ALTER CONFIGURATION.

For example, for all users, allow any SQL statement to use all CPUs:

```
ALTER SYSTEM ALTER CONFIGURATION ('global.ini','SYSTEM')
SET ('execution' , 'default_statement_concurrency_limit') = '0' WITH RECONFIGURE;
```

Note the following:

- There is no need to restart the SAP HANA database to activate the new setup.
- You must have the INIFILE ADMIN system privilege.
- The actual maximum is always capped by the actual number of CPUs.

For more information about the ALTER SYSTEM ALTER CONFIGURATION statement, see *ALTER SYSTEM ALTER CONFIGURATION Statement (System Management)*.

Setting User Parameters for Workload

You can set the maximum number of concurrent tasks for an SQL statement for a given user. You can do this using the SAP HANA cockpit or the following SQL statement:

```
ALTER USER <user_name> SET PARAMETER STATEMENT THREAD LIMIT = '<new_value>'
```

For example, for the user named USER_APL, allow any SQL statement to use all CPUs:

```
ALTER USER USER_APL SET PARAMETER STATEMENT THREAD LIMIT = '0';
```

Note the following:

- There is no need to restart the SAP HANA database to activate the new setup.
- You can reset the parameter to its default value with the following SQL statement:

  ```
  ALTER USER <user_name> CLEAR PARAMETER STATEMENT THREAD LIMIT;
  ```

- The actual maximum is always capped by the actual number of CPUs.

For more information, see *Setting User Parameters for Workload*.

Managing Workload with Workload Classes

A workload class defines resource limitations. You can apply workload classes based on user and application context information via a workload mapping.

Example

Define a workload class ALL_CPUS with no limit on the number of CPUs used by an SQL statement:

```
CREATE WORKLOAD CLASS "ALL_CPUS" set 'STATEMENT THREAD LIMIT'='0';
```

Define a workload mapping to activate the ALL_CPUS workload class for any user session related to the application FORECAST NEXT QUARTER:

```
CREATE WORKLOAD MAPPING "ACTIVATE_ALL_CPUS" WORKLOAD CLASS "ALL_CPUS"
SET 'APPLICATION NAME' = 'FORECAST NEXT QUARTER';
```

Declare that the current session is part of the FORECAST NEXT QUARTER application so that the ALL_CPUS workload class is applied for this session:

```
SET 'APPLICATION' = 'FORECAST NEXT QUARTER';
```

For more information about workload classes, see *Managing Workload with Workload Classes*.

Related Information

ALTER SYSTEM ALTER CONFIGURATION Statement (System Management)

Setting User Parameters for Workload
Managing Workload with Workload Classes
Database Configuration (SAP HANA Cockpit)
Create User-Specific Parameters (SAP HANA Cockpit)

5.4 Calling Functions

There are three ways to call SAP HANA APL functions. They can be called using the stored procedure technique, the direct technique, or the ANY procedure technique.

The three techniques are described below:

Technique	Description
Procedure	As an SAP HANA developer, you call stored procedures that take care of the lifecycle of the AFL wrappers and all their database objects. Procedures are part of the HCO_PA_APL delivery unit, which is automatically deployed when the function library is installed. This technique is not only much simpler than the direct one but is also more efficient and scalable. Use the `samples/sql/procedure` scripts to learn how to use the procedures.
Direct	As an SAP HANA developer, you explicitly generate an AFL wrapper for the function to be executed. The generation of the wrapper requires the explicit creation of the table types, signature table, and input and output tables. Once the AFL wrapper is generated, it can be invoked through a call statement. This direct technique is always available. Use the `samples/sql/direct` scripts to learn how to use this method.
ANY procedure	As an SAP HANA developer, you call APL functions as ANY AFLLANG procedures directly from the _SYS_AFL schema deployed on SAP HANA. You do not need to generate wrappers or call the stored procedures of the HCO_PA_APL delivery unit. Use the `samples/sql/any` scripts to learn how to use the ANY procedure technique. Note that the syntax used in SAP HANA Cloud is slightly different from the syntax used in SAP HANA on-premise systems. This technique is only available for deployments of SAP HANA APL on SAP HANA 2.0 SPS03 and higher.

> **Note**
>
> - The direct technique allows you to build read-only procedures that wrap the SAP HANA APL functions. The stored procedures of the delivery unit cannot be read-only since they make modifications in the SAP HANA database.
> - SAP recommends you use the direct technique to call the SAP HANA APL functions through ABAP Managed Database Procedures (AMDPs).
> - Contrary to the direct and stored procedure techniques, the ANY procedure technique does not convert the SQL data types of the dataset columns implicitly when you call an APL function. Therefore,

the datasets must contain data types that comply with the SAP HANA AFL data type support and restrictions. See *Supported SQL Data Types*.

Related Information

Calling a Function with the Direct Technique [page 47]
Calling a Function with the Procedure Technique [page 52]
Calling a Function as an ANY Procedure [page 55]
Supported SQL Data Types [page 43]

5.4.1 Choosing a Technique

The three techniques for running SAP HANA APL functions differ in the functionality they support and in their usage. An overview is shown below.

Feature	Direct Technique	Stored Procedure	ANY Procedure
No need to create the table types	x	√	√
Automated creation of the ApplyOut dataset table	x	√	x
Implicit data type conversion	√	√	x
Support of SAP HANA `HINT` for parallelization	√	x	√
Model debriefing by partition (training/validation/testing)	√	x	√
Getting the confusion matrix of a binary classification model	x	√	x
Training and applying a recommendation model on a large volume of data rapidly	x	√	x

You can combine the techniques in your SAP HANA APL end-to-end workflows. Here are some examples.

Example 1: Training and applying a model

1. Train a model with the direct or ANY procedure technique.
2. Prepare the ApplyOut table with the stored procedure technique:
    1. Call the `GET_TABLE_TYPE_FOR_APPLY` function (available in all techniques).
    2. Call the `CREATE_TABLE_FROM_TABLE_TYPE` function (available only in the stored procedure technique).
3. Apply the model with the direct or ANY procedure technique.

Example 2: Getting the confusion matrix

1. Train a binary classification model with the direct or ANY procedure technique.

2. Compute the confusion matrix with the stored procedure technique.

Example 3: Debriefing a model with a partition

1. Train a model with the stored procedure technique.
2. Debrief the model by partition with the direct or ANY procedure technique.

Related Information

Function Availability [page 38]

5.4.2 Use Case: Training a Model

This example compares the different techniques for calling SAP HANA APL functions. Sample scripts are used to run some of the functions. The use case is to create and train a model.

Direct Technique

You first create the table types for the IN and OUT parameters of the APL functions by running the `apl_create_table_types.sql` sample script that is located in the `\samples\sql\direct` folder.

You then run the `apl_createmodel_and_train.sql` sample script contained in the `\samples\sql\direct\apl_samples\classification_regression` folder. This script does the following:

1. Creates the table type needed for the training dataset:

```
create type ADULT01_T as table (
"age" INTEGER,
"workclass" NVARCHAR(32),
"fnlwgt" INTEGER,
"education" NVARCHAR(32),
"education-num" INTEGER,
"marital-status" NVARCHAR(32),
"occupation" NVARCHAR(32),
"relationship" NVARCHAR(32),
"race" NVARCHAR(32),
"sex" NVARCHAR(16),
"capital-gain" INTEGER,
"capital-loss" INTEGER,
"hours-per-week" INTEGER,
"native-country" NVARCHAR(32),
"class" INTEGER
);
```

2. Inserts it into the signature:

```
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (5, 'USER_APL','ADULT01_T',
'IN');
```

3. Creates the wrapper procedure:

```
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('APL_AREA',
'CREATE_MODEL_AND_TRAIN', 'USER_APL', 'APLWRAPPER_CREATE_MODEL_AND_TRAIN',
CREATE_MODEL_AND_TRAIN_SIGNATURE );
```

4. Calls the procedure.

```
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_AND_TRAIN_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;

APLWRAPPER_CREATE_MODEL_AND_TRAIN(:header, :config, :var_desc,:var_role, :dataset
, out_model,  out_log, out_sum, out_indic);
    insert into  "USER_APL"."MODEL_TRAIN_BIN" select * from :out_model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :out_indic;
END;
```

Stored Procedure Technique

You don't need to create the tables types, since they are provided by the SAP_PA_APL schema. You also don't need to create the procedures since they are also provided as stored procedures by the same schema.

You run the `apl_createmodel_and_train_ex.sql` sample script that is located in the `\samples\sql\procedure\apl_samples\classification_regression` folder. The script calls the following procedure:

```
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_AND_TRAIN_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;

"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN"(:header, :config, :var_des
c,:var_role, 'APL_SAMPLES','ADULT01', model,  train_log, train_sum,
train_indic);
    insert into  "USER_APL"."MODEL_TRAIN_BIN" select * from :model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :train_log;
    insert into  "USER_APL"."SUMMARY"         select * from :train_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :train_indic;
END;
```

ANY Procedure Technique

You use the tables types provided by the SAP_PA_APL schema and the procedures provided by the _SYS_AFL schema.

You run the `apl_createmodel_and_train_any.sql` sample script that is located in the `\samples\sql\any\apl_samples\classification_regression` folder. The script calls the following procedure:

```
call "_SYS_AFL"."APL_CREATE_MODEL_AND_TRAIN" (
FUNC_HEADER, CREATE_AND_TRAIN_CONFIG, VARIABLE_DESC, VARIABLE_ROLES,
"APL_SAMPLES"."ADULT01", --- training dataset
MODEL_TRAIN_BIN, OPERATION_LOG, SUMMARY, INDICATORS) with overview;
```

For SAP HANA Cloud, see the sample script `apl_createmodel_and_train_any_hce.sql`.

5.4.3 Use Case: Debriefing a Model with a Testing Partition

You want to know the predictive power from the testing partition of the dataset. You can debrief a model with the testing partition of the dataset by using the `INDICATORS_DATASET_T` table type in the CREATE_MODEL_AND_TRAIN function signature. This is only supported by the direct technique and the ANY procedure.

Direct Technique

1. The `apl_create_table_types.sql` sample script allows you to create the INDICATORS by partition table type:

```
create type INDICATORS_DATASET_T as table (
"OID" VARCHAR(50),
"VARIABLE" VARCHAR(255),
"TARGET" VARCHAR(255),
"KEY" VARCHAR(100),
"VALUE" NCLOB,
"DETAIL" NCLOB,
"DATASET" VARCHAR(255)
);
```

2. You insert this table type into the function signature:

```
insert into GET_MODEL_INFO_SIGNATURE values (7,
'USER_APL','INDICATORS_DATASET_T', 'OUT');
```

3. You create the output table from that table type:

```
create table INDICATORS_DATASET like INDICATORS_DATASET_T;
```

4. You then call the generated wrapper:

```
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config    = select * from OPERATION_CONFIG;
    APLWRAPPER_GET_MODEL_INFO(:header, :model_in, :config, out_summary,
out_variable_roles, out_variable_desc, out_indicators,
out_profitcurves ,out_schema_training);
```

```
    -- store result into table
    insert into  "USER_APL"."SUMMARY" select * from :out_summary;
    insert into  "USER_APL"."VARIABLE_ROLES" select * from :out_variable_roles;
    insert into  "USER_APL"."VARIABLE_DESC" select * from :out_variable_desc;
    insert into  "USER_APL"."INDICATORS" select * from :out_indicators;
    insert into  "USER_APL"."PROFITCURVES" select * from :out_profitcurves;
    insert into  "USER_APL"."SCHEMA_TRAINNING" select *
from :out_schema_training;
END;
```

5. You query the predictive power for the test partition as well as the validation partition:

```
select * from "USER_APL"."INDICATORS_DATASET"
where variable = 'IS_FRAUD' and key = 'PredictivePower' and dataset in
('Validation','Test')
```

OID	VARIABLE	TARGET	KEY	VALUE	DETAIL	DATASET
#42	IS_FRAUD	IS_FRAUD	PredictivePower	0.66459999999999997	rr_IS_FRAUD	Validation
#42	IS_FRAUD	IS_FRAUD	PredictivePower	0.60399999999999998	rr_IS_FRAUD	Test

Stored Procedure Technique

This function is not implemented for the stored procedure technique.

ANY Procedure Technique

1. You create the table:

```
create table INDICATORS_DATASET like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS_DATASET";
```

2. You call the ANY procedure:

```
call "_SYS_AFL"."APL_GET_MODEL_INFO" (FUNC_HEADER, MODEL_TRAIN_BIN,
OPERATION_CONFIG, SUMMARY, VARIABLE_ROLES, VARIABLE_DESC, INDICATORS_DATASET,
PROFITCURVES
) with overview;
```

The result is the same as for the direct technique.

Related Information

CREATE_MODEL_AND_TRAIN [page 137]

5.4.4 Function Availability

The following overview shows which techniques are supported for each of the SAP HANA APL functions.

Predictive Model Functions

Function	Direct or ANY procedure	Stored Procedure
APPLY_MODEL [page 72]	√	√
APPLY_MODEL_AND_TEST [page 98]	√	√
APPLY_RECO_MODEL [page 108]	√	√
APPLY_RECOMMENDER_TO_BASKET [page 113]	x	√
APPLY_RECOMMENDER_TO_USERS [page 117]	x	√
APPLY_SOCIAL_MODEL [page 120]	√	√
COMPUTE_CONFUSION_MATRIX [page 127]	x	√
CREATE_MODEL [page 132]	√	√
CREATE_MODEL_AND_TRAIN [page 137]	√	√
CREATE_MODEL_AND_TRAIN (3 Datasets) [page 142]	√	√
CREATE_RECO_MODEL_AND_TRAIN [page 157]	√	√
CREATE_RECOMMENDER [page 169]	x	√
CREATE_SOCIAL_MODEL_AND_TRAIN [page 173]	√	√
EXPORT_APPLY_CODE [page 183]	√	√
EXPORT_APPLY_CODE_FOR_RECO [page 191]	√	√
EXPORT_PROFITCURVES [page 196]	√	√
EXPORT_VARIABLEDESCRIPTIONS [page 199]	√	√
GET_MODEL_DATASET_TYPES [page 201]	√	√
GET_MODEL_INFO [page 206]	√	√
GET_TABLE_TYPE_FOR_APPLY [page 213]	√	√
GET_TABLE_TYPE_FOR_SOCIAL_APPLY [page 216]	√	√
GET_VARIABLE_INDICATORS [page 222]	√	√
IMPORT_VARIABLEDESCRIPTIONS [page 226]	√	√
PARALLEL_APPLY_MODEL [page 228]	√	x
PARALLEL_APPLY_RECO_MODEL [page 231]	√	x
PARALLEL_APPLY_SOCIAL_MODEL [page 234]	√	x
PUBLISH_MODEL [page 239]	√	√
RETRAIN_MODEL [page 242]	√	√
RETRAIN_MODEL (3 Datasets) [page 247]	√	√
TEST_MODEL [page 251]	√	√
TRAIN_MODEL [page 258]	√	√

Function	Direct or ANY procedure	Stored Procedure
TRAIN_MODEL (3 Datasets) [page 264]	√	√
UPDATE_MODEL [page 270]	√	√

Predictive Business Functions

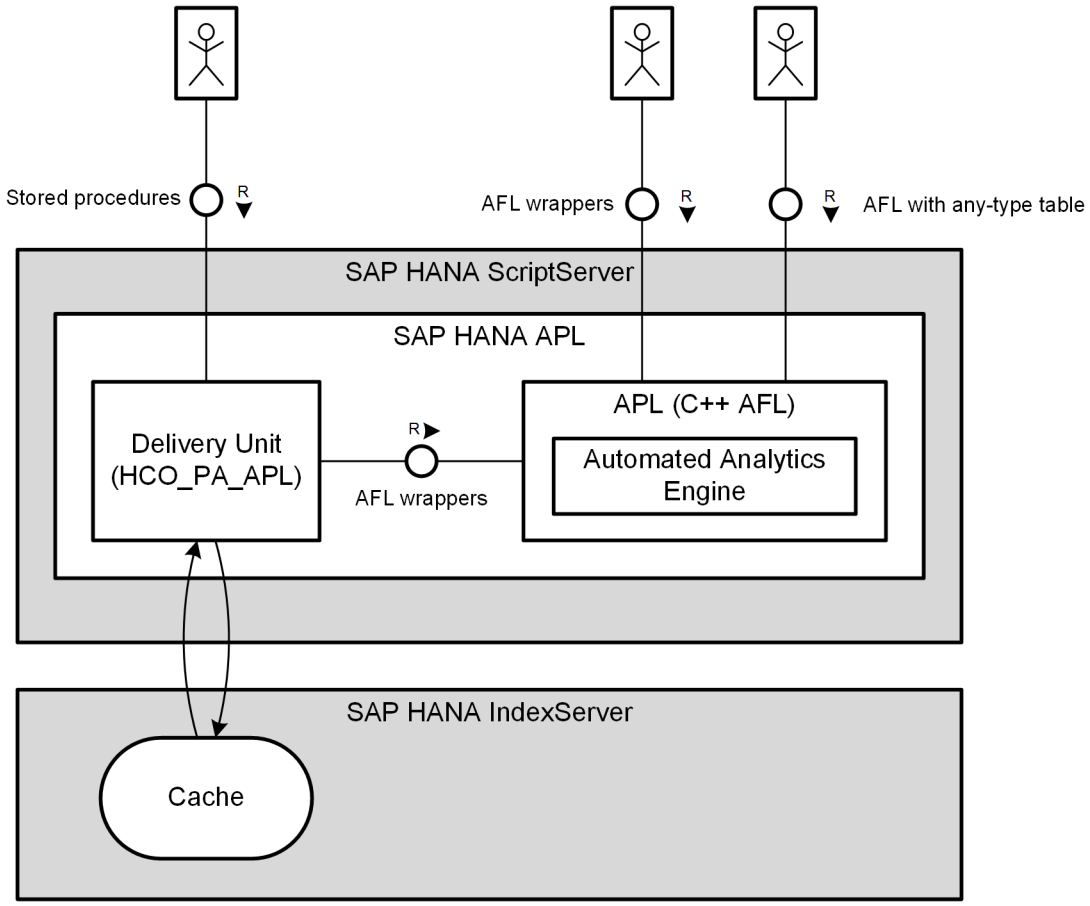
	Direct or ANY procedure	Stored Procedure
DEBRIEF_APPLY_RESULT [page 277]	√	√
FORECAST [page 281]	√	√
GET_KEY_INFLUENCERS_FROM_MODEL [page 300]	√	√
KEY_INFLUENCERS [page 304]	√	√
PROFILE_DATA [page 307]	√	√
PROFILE_DATA_AND_GET_ASSOCIATIONRULES [page 314]	√	√
RECOMMEND [page 320]	√	√
SCORING_EQUATION [page 326]	√	√

Technical Functions

	Direct or ANY procedure	Stored Procedure
CLEANUP [page 336]	x	√
CREATE_TABLE_FROM_TABLE_TYPE [page 336]	x	√
CREATE_TYPE_FROM_TABLE_TYPE [page 336]	x	√
IMPORT_MODEL [page 336]	x	√
PING [page 334]	√	√
PROGRESS_CLEANUP [page 335]	√	x

5.5 Architecture Overview

SAP HANA APL embeds the Automated Analytics engine within its APL component. SAP HANA APL is an application function library (AFL) and, therefore, runs by default in the SAP HANA script server.



> ### Note
>
> Three techniques are available to call the SAP HANA APL functions: AFL wrappers (direct technique), stored procedures, and ANY procedures. SAP recommends to use the simpler method of the stored procedures. Stored procedures are part of the `HCO_PA_APL` delivery unit.

> ### Note
>
> In the procedure technique, a cache that is used to store and reuse transient database objects improves function call performance.

Related Information

Working with SAP HANA APL Functions [page 42]

5.6 Compatibility Information

You can reuse current versions of models in earlier SAP HANA APL versions provided they are compatible. For gradient boosting models, there is an interoperability issue between environments with different endianness.

Compatibility with Earlier Versions

You can reuse the models created in this version in earlier SAP HANA APL versions. If a model created in version N of APL is saved in version N-1, then the model becames a model of this version N-1. For example, a classification model created in APL 1807 can be applied as a 3.3 model using the `APPLY_MODEL` function of APL 3.3.

The following table tells you more about the specificities of this compatibility according to the type of model:

Model Type	Untrained Model	Trained Model
Classification and Regression	Compatible with APL 3.x and APL 2.5	Compatible with APL 3.3
Clustering	Compatible with APL 3.x and APL 2.5	Compatible with APL 3.3
Time Series	Compatible with APL 3.x and APL 2.5	Compatible with APL 3.3
Time Series with exponential smoothing and linear regression	Compatible with APL 3.3 and APL 3.2	Compatible with APL 3.3

Interoperability Between Environments with Different Endianness

For gradient boosting models, there is an interoperability issue between environments with different endianness. If you create a gradient boosting model in an IBM Power Systems (big-endian) environment, you can't reload it in an Intel or IBM Power Systems (little-endian) environment. Conversely, a gradient boosting model created in a little-endian environment can't be reloaded in a big endian environment.

6 Working with SAP HANA APL Functions

You can call SAP HANA APL functions using the stored procedure technique, the direct technique, or the ANY procedure technique. You can also get progress information about SAP HANA APL functions while they are running. Some SAP HANA APL functions can be canceled, if needed.

6.1 Supported Database Objects

Various database objects can be used as the input or output parameters of the SAP HANA APL functions.

You can use the following types of database tables as the input and output parameters of an SAP HANA APL function:

- Tables that store data row-wise
- Tables that store data column-wise
- Virtual tables

You can use the following SAP HANA database objects as input parameters only:

- SQL views
- Synonyms

For more information about table types, see the *SAP HANA Administration Guide*.

You can also use a calculation view by creating an SQL view on top of it. You can use a calculation view as follows:

1. As the system user, give the database user access to the calculation view.
2. As the database user, create the SQL view on top of the calculation view.
3. Call the APL function with the SQL view as the input dataset.

> **Note**
>
> You can only use calculation views as input datasets.

Example

You want to run a forecast for a given market by using the `CV_TIME_SERIES` calculation view. As shown below, this calculation view has a variable that must be assigned a value:

Name	Label	Aggregation	Variable	Semantic Type	Label Column	Hidden
MARKET	MARKET		MARKET_VAR			☐
KPI_VALUE	KPI_VALUE	Sum				☐
ABAP_DAY	ABAP_DAY					☑
THE_DATE	THE_DATE					☐

You call the FORECAST function with the TS_SORTED view as the input dataset:

```
-------- as user: SYSTEM
GRANT EXECUTE ON sys.repository_rest TO USER_APL;
GRANT MODELING ON USER_APL;
-------- as user: USER_APL
CREATE VIEW TS_SORTED as
(
SELECT "THE_DATE", SUM("KPI_VALUE") AS "KPI_VALUE"
FROM   "_SYS_BIC"."ADV_SAMPLES/CV_TIME_SERIES"
WHERE  "MARKET" =  'Domestic'  -- Answering the mandatory variable
GROUP BY "THE_DATE"
ORDER BY "THE_DATE")
DO BEGIN
    header   = select * from FUNC_HEADER;
    config    = select * from FORECAST_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;


"SAP_PA_APL"."sap.pa.apl.base::FORECAST"(:header,  :config, :var_desc, :var_role,
 'USER_APL','TS_SORTED', 'USER_APL','FORECAST_OUT', out_log, out_summary,
out_indicators);

    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_summary;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indicators;
END;
```

6.2 Supported SQL Data Types

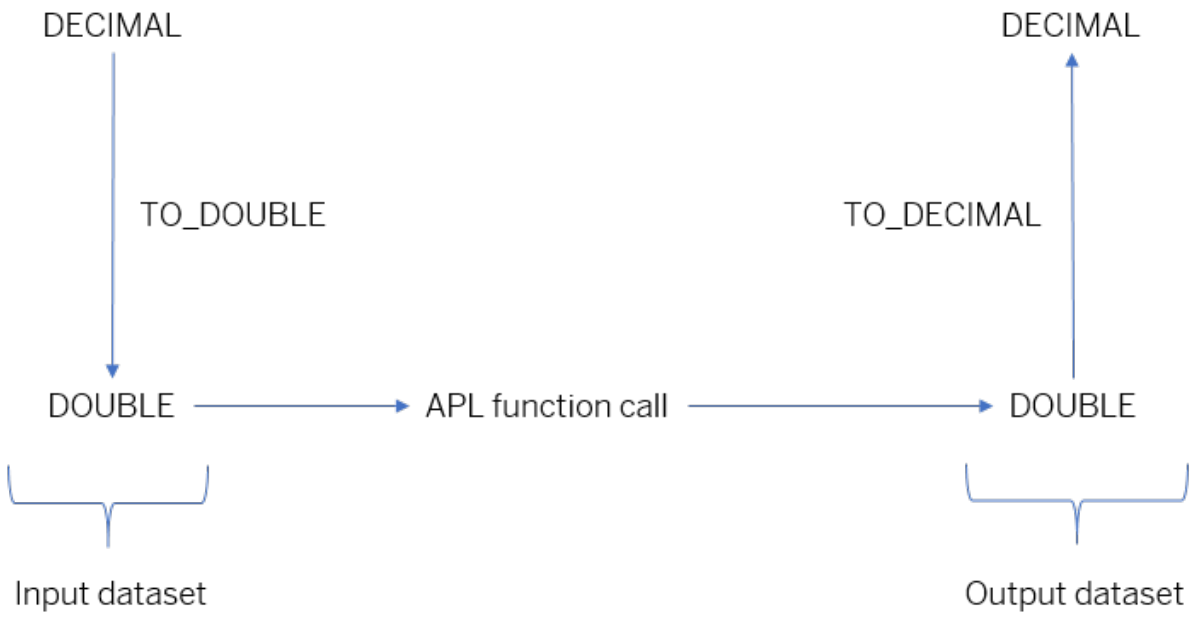
The following table lists the SQL data types that you can use in the input or output dataset tables of the SAP HANA APL functions.

Supported SQL Data Type
"INTEGER"
"BIGINT"
"DOUBLE"
"BINARY" (not with the ANY procedure)
"VARBINARY" (not with the ANY procedure)
"CLOB"
"NCLOB"
"VARCHAR(n)"
"NVARCHAR(n)"
"DATE"
"TIME"
"TIMESTAMP"

Supported SQL Data Type
`"SECONDDATE"`
`"DECIMAL"`
`"DECIMAL(precision,scale)"` where 0 < precision <= 38 and 0 <= scale <= precision

DECIMAL Data Type

The `"DECIMAL"` data type is fully supported on SAP HANA 2.0 SPS03 and beyond but not on previous versions of SAP HANA 1.0 and 2.0. However, you can use this type in the input dataset by first converting it to `"DOUBLE"` before calling an APL function. You do this conversion using the `TO_DOUBLE` SQL function. To use it in the output dataset, you must convert it back from `"DOUBLE"` to `"DECIMAL"` by using the `TO_DECIMAL` SQL function. For more information about the functions, see *SAP HANA SQL and System Views Reference*.

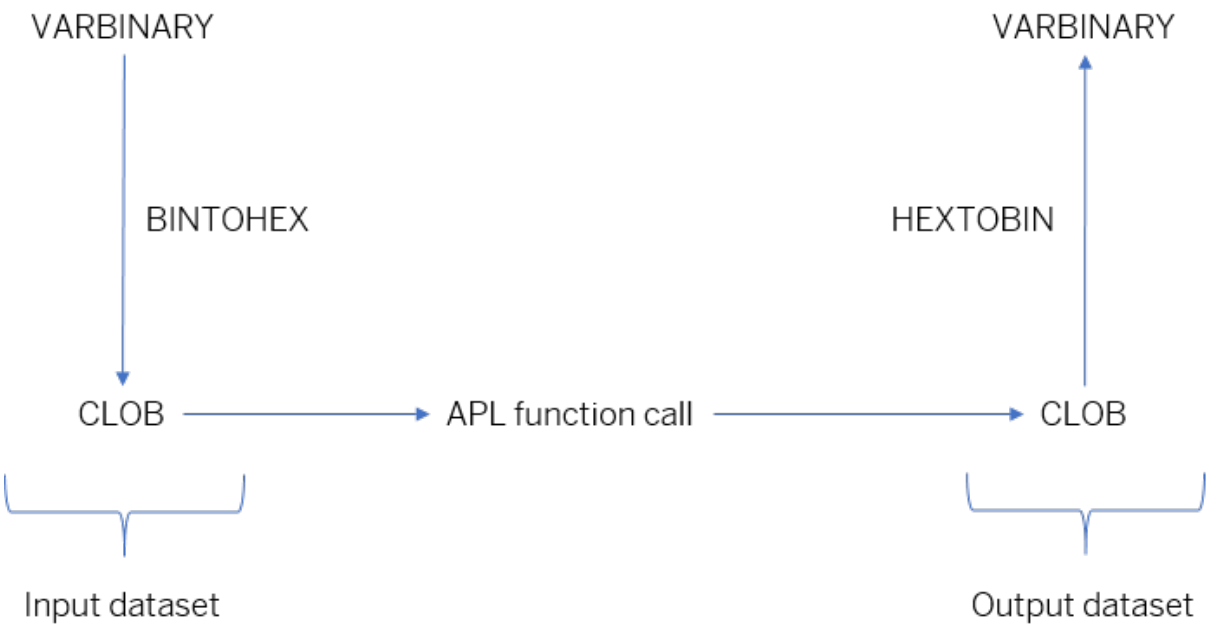


BINARY and VARBINARY Data Types

You can use the `“BINARY”`, `“VARBINARY”`, and all variations of binary data type (for example, `ST_GEOMETRY`, `ST_POINT`, and `BLOB`) if you first convert them to `CLOB` before any call to an APL function. You can do this conversion by using the `BINTOHEX('<binary value>')` SQL function.

To use these types in the output dataset, you must convert them back from `"CLOB"` with the help of the `HEXTOBIN('<clob value>')` SQL function.

For more information about the functions, see the *SAP HANA SQL and System Views Reference*.



6.3 Table Types

The SAP HANA APL functions accept tables and SQL views as input parameters, and return tables. The tables must comply with a certain structure, which is in the form of a list of [column name, column SQL type] pairs.

When using the SAP HANA APL functions, you need to be able to rely on table types that define all table structures used by the functions. The table types must be compatible with the table structures expected by the APL API.

The table types are used as follows:

- Either explicitly through SQL code or automatically through stored procedures when the AFL wrappers are generated
- As a way to easily create new tables to hold the input and output parameters when the functions or stored procedures are used

You can explicitly create table types or they can be created automatically:

- Explicitly created table types
  This is the default technique. It is fully customizable but error-prone. The APL table types may have structure variants and datasets may require their own table types. The `apl_create_table_types.sql` script is provided as part of the samples to help you create all the table types required by SAP HANA APL. Since all APL users need to have access to the table types, you must make sure they are created for every user or shared with appropriate database privileges.
- Automatically created table types
  The table types are automatically created when the HCO_PA_APL delivery unit is deployed and are then available for use. Users must be granted the privilege `sap.pa.apl.base.roles::APL_EXECUTE` to be able to use them.

> **Note**
>
> Table types are transient objects. Technically, they need to exist only for the duration of the function call and can be dropped when the call has completed. They can be reused, however, as long as the same user is using the same table structure. The advantage of using stored procedures is that they manage the life cycle of the table types for you.

6.3.1 Table Type Variants

Different table type variants are supported for some parameters of some of the SAP HANA APL functions. For example, the `OPERATION_CONFIG_T` table type can have a 2-column structure or a 3-column structure, depending on the use case and function.

When using SAP HANA APL directly through its functions and through the AFL wrappers, you are responsible for creating the appropriate table types and providing the appropriate input tables to the function. If the function supports different variants for the table structures, then no error is reported.

The stored procedures do not support any table type variants. As a result, only a subset of all the supported table types is automatically deployed and actually used within the stored procedures. The following table shows the mapping between the table types that can be explicitly created (for example, using the `apl_create_table_types.sql` sample script) and those that are automatically deployed with the stored procedures:

User-Created Table Type	Stored Procedure Table Type
`PROCEDURE_SIGNATURE_T`	`sap.pa.apl.base::BASE.T.PROCEDURE_SIGNATURE`
`CONFIGURATION_T`	None
`CONFIGURATION_OUTPUT_T`	None
`INTERFACE_CATEGORIES_T`	None
`INTERFACE_FUNCTIONS_T`	None
`INTERFACE_SIGNATURES_T`	None
`INTERFACE_TTYPES_T`	None
`INTERFACE_TDESCS_T`	None
`PING_OUTPUT_T`	`sap.pa.apl.base::BASE.T.PING_OUTPUT`
`FUNCTION_HEADER_T`	`sap.pa.apl.base::BASE.T.FUNCTION_HEADER`
`TABLE_TYPE_T`	`sap.pa.apl.base::BASE.T.TABLE_TYPE`
`MODEL_BIN_T`	None
`MODEL_BIN_OID_T`	`sap.pa.apl.base::BASE.T.MODEL_BIN_OID`
`MODEL_TXT_T`	None
`MODEL_TXT_OID_T`	None

User-Created Table Type	Stored Procedure Table Type
MODEL_NATIVE_T	sap.pa.apl.base::BASE.T.MODEL_NATIVE
VARIABLE_DESC_T	None
VARIABLE_DESC_OID_T	sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID
VARIABLE_ROLES_T	None
VARIABLE_ROLES_WITH_COMPOSITES_T	None
VARIABLE_ROLES_WITH_COMPOSITES_OID_T	sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID
OPERATION_CONFIG_T	None
OPERATION_CONFIG_DETAILED_T	sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED
ADMIN_T	sap.pa.apl.base::BASE.T.ADMIN
OPERATION_LOG_T	sap.pa.apl.base::BASE.T.OPERATION_LOG
SUMMARY_T	sap.pa.apl.base::BASE.T.SUMMARY
RESULT_T	sap.pa.apl.base::BASE.T.RESULT
INDICATORS_T	sap.pa.apl.base::BASE.T.INDICATORS
INDICATORS_DATASET_T	sap.pa.apl.base::BASE.T.INDICATORS_DATASET
PROFITCURVE_T	sap.pa.apl.base::BASE.T.PROFITCURVE
INFLUENCERS_T	sap.pa.apl.base::BASE.T.INFLUENCERS
CONTINUOUS_GROUPS_T	sap.pa.apl.base::BASE.T.CONTINUOUS_GROUPS
OTHER_GROUPS_T	sap.pa.apl.base::BASE.T.OTHER_GROUPS
ASSOCIATION_RULES_T	sap.pa.apl.base::BASE.T.ASSOCIATION_RULES
RULE_ITEMS_T	sap.pa.apl.base::BASE.T.RULE_ITEMS
None	sap.pa.apl.base::BASE.T.DROPPED_OBJECTS_LIST

6.4 Calling a Function with the Direct Technique

SAP HANA APL is based on the SAP HANA application function library (AFL) technology. When you use the direct technique, you create an AFL wrapper and insert the APL function into it. You then invoke the wrapper to call the function.

Your sequence of SQL statements should do the following:

1. Connect to the database schema
2. Create the input and output table types
3. Generate the AFL wrapper
4. Create the input and output tables
5. Run the function

6.4.1 Connecting to the Database Schema

Connecting to the schema is the first step in the SQL script that calls your APL function. You will create wrappers and tables in this schema.

Run the following command to connect to the SAP HANA database schema using the user (`<APL_user>`) and password (`<password>`) defined by the administrator.

```
connect <APL_user> password <password>;
```

> **Note**
>
> The user must have the appropriate privileges to be able to trigger the AFL/APL call mechanism.

6.4.2 Creating Table Types

> **→ Remember**
>
> This procedure is no longer necessary if you are using the stored procedures from the `HCO_PA_APL` delivery unit.

Use the SQL script `apl_create_table_types.sql` to create table types that comply with the SAP HANA APL function signatures.

Ensure that the table types you create are compatible with the table types used with the functions. "Compatible" means that the table types are not strictly defined. For instance, when the function expects a string parameter, you can provide a table which contains a VARCHAR(10), VARCHAR(30), or CLOB.

> **Note**
>
> Using blank spaces or single quotes (`'`) in column names of table types causes an error when generating the AFL wrapper. Be careful when defining the table types corresponding to the training dataset and the "apply out" dataset. For all the other table types, SAP recommends you use the provided APL table types.

1. Make sure the appropriate user privileges have been created and granted using script `apl_admin.sql`.
2. Run the script `samples/sql/direct/apl_create_table_types.sql`.

   Excerpt of the script content:

   ```
   connect USER_APL password Password1;
   drop type PROCEDURE_SIGNATURE_T;
   create type PROCEDURE_SIGNATURE_T as table (
   ```

```
    "POSITION" INT CS_INT,
    "SCHEMA_NAME" NVARCHAR(256) CS_STRING,
    "TYPE_NAME" NVARCHAR(256) CS_STRING,
    "PARAMETER_TYPE" VARCHAR(7) CS_STRING
);
drop type INTERFACE_CATEGORIES_T;
create type INTERFACE_CATEGORIES_T as table (
    "CATEGORY_ID" INTEGER,
    "NAME" VARCHAR(255),
    "DESCRIPTION" VARCHAR(255)
);
drop type INTERFACE_FUNCTIONS_T;
create type INTERFACE_FUNCTIONS_T as table (
    "FUNC_ID" INTEGER,
    "NAME" VARCHAR(255),
    "DESCRIPTION" VARCHAR(255),
    "CATEGORY_ID" INTEGER
);
...
;
```

6.4.3 Generating an AFL Wrapper

- The users and privileges have been created and granted.
- The table types have been created.

> **Note**
>
> You must manage the lifecycle of your created SQL objects. They should be dropped at some point, usually at the end of the APL function call. Dispose of all the transient database objects that have just been created: the AFL wrapper, the signature table, the table types, and so on.

1. Create a signature table.

   By convention, the names of the signature tables in this document are `<FUNCTION_NAME>_SIGNATURE`.
2. Insert a row for each input and output parameter into the signature table.

   Specify the following:

   - The position of the parameter in the table (an index value)
   - The schema name of the parameter table type
   - The name of the parameter table type
   - The direction of the parameter (`IN` or `OUT`)
3. Call the `AFLLANG_WRAPPER_PROCEDURE_DROP` procedure to drop the wrapper previously generated.
4. Call the `SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE` procedure to generate the wrapper.

   Specify the following:

   - `APL_AREA`, which is the APL component area name
   - The actual function name to be executed
   - The target schema for the created AFL wrapper
   - The user schema for the generated wrapper
   - The name of the APL wrapper you want to create

- The name of the signature table used for the function

> **Note**
>
> Refer to SAP note 2046767 for specific information about creating, dropping, and executing `AFLLANG` wrapper procedures.

Example

This example shows how to create a wrapper for the `CREATE_MODEL` function. Before generating the wrapper, you create a table type for the dataset.

```
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CREATE_MODEL_SIGNATURE;
create column table CREATE_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into CREATE_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into CREATE_MODEL_SIGNATURE values (2, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into CREATE_MODEL_SIGNATURE values (3, 'USER_APL','ADULT01_T', 'IN');
insert into CREATE_MODEL_SIGNATURE values (4, 'USER_APL','MODEL_BIN_OID_T',
'OUT');
insert into CREATE_MODEL_SIGNATURE values (5, 'USER_APL','VARIABLE_DESC_OID_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('<APL_user>','APLWRAPPER_CREATE_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_MODEL','USER_APL',
'APLWRAPPER_CREATE_MODEL', CREATE_MODEL_SIGNATURE);
```

Related Information

https://launchpad.support.sap.com/#/notes/2046767

Running Admin Scripts

6.4.4 Creating Input and Output Tables

You can now create the tables for the input and output parameters of the APL function and insert values into them. The input tables are the arguments used for the function. The output tables will contain the results of the operation.

1. Drop any existing tables before creating new ones.
2. Create tables.
3. Insert rows with values into the required tables.

Example

```
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');

drop table CREATE_CONFIG;
create table CREATE_CONFIG like OPERATION_CONFIG_T;
insert into CREATE_CONFIG values ('APL/ModelType', 'regression/
classification');

drop table MODEL_BIN;
create table MODEL_BIN like MODEL_BIN_OID_T;

drop table VARIABLE_DESC_OUT;
create table VARIABLE_DESC_OUT like VARIABLE_DESC_OID_T;
```

6.4.5 Calling the AFL Wrapper

You can now execute your function through a call to the AFL wrapper.

Call the AFL wrapper with the appropriate input and output tables.

The input and output dataset tables are two strings {<schema_name>, <table or view name>} concatenated with a comma.

Example

CREATE_MODEL

The input dataset is `APL_SAMPLES.ADULT01`.

```
DO BEGIN
```

```
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_CONFIG;
    dataset  = select * from APL_SAMPLES.ADULT01;

    APLWRAPPER_CREATE_MODEL(:header, :config,:dataset,out_model,out_var_desc);
    -- store result into table
    insert into  "USER_APL"."MODEL_BIN"          select * from :out_model;
    insert into  "USER_APL"."VARIABLE_DESC_OUT"  select * from :out_var_desc;
    -- show result
    select * from "USER_APL"."MODEL_BIN";
    select * from "USER_APL"."VARIABLE_DESC_OUT";
END;
```

APPLY_MODEL

The input dataset is `APL_SAMPLES.ADULT01`. The output dataset is `ADULT01_APPLY`.

```
DO BEGIN
    header       = select * from FUNC_HEADER;
    model_in     = select * from MODEL_TRAIN_BIN;
    apply_config = select * from APPLY_CONFIG;
    dataset      = select * from APL_SAMPLES.ADULT01;
    APLWRAPPER_APPLY_MODEL(:header, :model_in, :apply_config,  :dataset,
out_apply , out_apply_log);
    -- store result into table
    insert into  "USER_APL"."ADULT01_APPLY"   select * from :out_apply;
    insert into  "USER_APL"."APPLY_LOG"       select * from :out_apply_log;
    -- show result
    select * from "USER_APL"."ADULT01_APPLY";
    select * from "USER_APL"."APPLY_LOG";
END;
```

6.5 Calling a Function with the Procedure Technique

The procedure technique relies on the stored procedures and table types that are provided with the `sap.pa.apl` package.

A stored procedure is available for every function with the same name. Their signatures are equivalent except that every time an input or output dataset is expected as a parameter, the stored procedure signature expects a pair of strings, {schema name, table or view name}, as opposed to a table or view that is passed directly.

All table types required by the functions have a corresponding counterpart with a similar name in the `sap.pa.apl.base` package. For more information about the mapping between the table types and their counterparts, see *Table Type Variants*.

The table types are automatically created when the delivery unit is deployed, so you only need to create the input and output tables. You can then call the stored procedure corresponding to the APL function you want to execute.

Your sequence of SQL statements should do the following:

1. Connect to the database schema
2. Create the input and output tables
3. Run the stored procedure

Related Information

Table Type Variants [page 46]

6.5.1 Connecting to the Database Schema

Connecting to the schema is the first step in the SQL script that calls your APL function. You will create wrappers and tables in this schema.

Run the following command to connect to the SAP HANA database schema using the user (`<APL_user>`) and password (`<password>`) defined by the administrator.

```
connect <APL_user> password <password>;
```

> **Note**
>
> The user must have the appropriate privileges to be able to trigger the AFL/APL call mechanism.

6.5.2 Creating Input and Output Tables

You can now create the tables for the input and output parameters of the stored procedure and insert values into them. The input tables are the arguments used for the function. The output tables will contain the results of the operation.

1. Drop any existing tables before creating new ones.
2. Create tables using stored procedure table types.
3. Insert rows with values into the required tables.

Example

```
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#69');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');

drop table CREATE_CONFIG;
create table CREATE_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into CREATE_CONFIG values ('APL/ModelType', 'regression/
classification',null);

drop table MODEL_BIN;
create table MODEL_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
```

```
drop table VARIABLE_DESC_OUT;
create table VARIABLE_DESC_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
```

6.5.3 Executing the Stored Procedure

Call the stored procedure with the appropriate parameters.

The input and output dataset tables expected are pairs of strings {'<schema_name>', '<table or view name>'} between single quotes and separated by a comma.

Example

CREATE_MODEL

The input dataset is 'APL_SAMPLES','ADULT01'.

```
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_CONFIG;


"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL"(:header, :config,'APL_SAMPLES','ADUL
T01',out_model,out_var_desc);
    -- store result into table
    insert into  "USER_APL"."MODEL_BIN"          select * from :out_model;
    insert into  "USER_APL"."VARIABLE_DESC_OUT"  select * from :out_var_desc;
    -- show result
    select * from "USER_APL"."MODEL_BIN";
    select * from "USER_APL"."VARIABLE_DESC_OUT";
END;
```

APPLY_MODEL

The input dataset is 'APL_SAMPLES','ADULT01'. The output dataset is 'USER_APL','ADULT01_APPLY'.

```
DO BEGIN
    header       = select * from FUNC_HEADER;
    model_in     = select * from MODEL_TRAIN_BIN;
    apply_config = select * from APPLY_CONFIG;

"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL"(:header, :model_in, :apply_config,
'APL_SAMPLES','ADULT01', 'USER_APL','ADULT01_APPLY' , out_apply_log, out_sum);
    -- store result into table
    insert into  "USER_APL"."APPLY_LOG"       select * from :out_apply_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    -- show result
    select * from "USER_APL"."ADULT01_APPLY";
    select * from "USER_APL"."APPLY_LOG";
END;
```

6.5.4 Performance Considerations

When you use your own user schema to save the stored procedures, performance is impacted due to writing the stored procedure and updating the rights so that other roles can access the new object (stored procedure). This update is implicit and impacts the performance.

6.5.5 Supporting SAP HANA Cloud Multitenancy

When you use the procedure method to call APL in an SAP HANA Cloud multitenant scenario, you must use the APL_CACHE_SCHEMA session variable.

Using the procedure method to call APL in an SAP HANA Cloud tenant triggers an error message:

```
Object is out of boundary. Tenant user cannot create an object in a non-tenant schema.
```

A tenant user cannot write outside the tenant database. The procedure mode tries to create objects that are stored in a cache (technically, a shared schema SAP_PA_APL) outside the tenant database, and this fails.

The workaround uses the session variable APL_CACHE_SCHEMA, which allows changing the schema where APL stores its cache. Within an SAP HANA Cloud tenant, two possible values for this session variable exist:

- Use the current schema of the tenant user to store APL cache:
  `SET 'APL_CACHE_SCHEMA' = CURRENT_USER;`
- Create a schema in the tenant and use it to store APL cache:
  `SET 'APL_CACHE_SCHEMA' = '<schema_name>';`

6.6 Calling a Function as an ANY Procedure

You can call any APL function as an ANY AFLLANG procedure from the `_SYS_AFL` schema (SAP HANA 2.0 SPS03 or higher).

Using this method means the following:

- You do not need to use the cache mechanism, so you do not need to set the custom cache for the `HCO_PA_APL` stored procedures.
- You call the function directly without calling `SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE` because the SAP HANA application function library (AFL) generates the wrapper.

Sample scripts are available in the `samples/sql/any` folder for demonstration purposes:

- SAP HANA on-premise: `apl_createmodel_and_train_any.sql`
- SAP HANA Cloud: `apl_createmodel_and_train_any_hce.sql`

Your sequence of SQL statements should do the following:

1. Connect to the database schema
2. Create the input and output tables

3. Run the APL function

> **Note**
>
> Data type casting is not allowed when you call a function as an ANY procedure. The SQL data types in the input and output tables must comply with the SAP HANA AFL restrictions.

Related Information

Supported SQL Data Types [page 43]

6.6.1 Connecting to the Database Schema

Connecting to the schema is the first step in the SQL script that calls your APL function. You will create wrappers and tables in this schema.

Run the following command to connect to the SAP HANA database schema using the user (`<APL_user>`) and password (`<password>`) defined by the administrator.

```
connect <APL_user> password <password>;
```

> **Note**
>
> The user must have the appropriate privileges to be able to trigger the AFL/APL call mechanism.

6.6.2 Creating Input and Ouput Tables

You can now create the tables for the input and output parameters of the function and insert values into them. The input tables are the arguments used for the function. The output tables will contain the results of the operation.

1. Drop any existing tables before creating new ones.
2. Create tables using stored procedure table types.
3. Insert rows with values into the required tables.

Example

```
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
```

```
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_AND_TRAIN_CONFIG;
create table CREATE_AND_TRAIN_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into CREATE_AND_TRAIN_CONFIG values ('APL/ModelType', 'regression/
classification',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/VariableAutoSelection',
'true',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionBestIteration', 'true',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMaxNbOfFinalVariables', '5',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMinNbOfFinalVariables', '1',null);
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to use guess variables
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
-- variable roles are optional, hence the empty table
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
```

6.6.3 Running the Function

You do not need to call `SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE` like in the direct technique, because the AFL SDK that comes with SAP HANA 2.0 SPS03 now generates the wrapper.

Call the function as follows:

- The syntax must state the actual function name:

  ```
  call "_SYS_AFL"."APL_<Function_Name>();
  ```

  > **Note**
  >
  > For example, use `CREATE_MODEL__OVERLOAD_3_1` if you want to use this function, not `CREATE_MODEL`.

- You enter an input or output dataset as one parameter by using the concatenation of the schema and table names with double quotes, for example `"APL_SAMPLES"."ADULT01"`. You do not enter schema and table as two different parameters like in the stored procedure technique.

Example

On SAP HANA on-premise systems:

```
call "_SYS_AFL"."APL_CREATE_MODEL_AND_TRAIN"(FUNC_HEADER,
CREATE_AND_TRAIN_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, "APL_SAMPLES"."ADULT01",
MODEL_TRAIN_BIN, OPERATION_LOG, SUMMARY, INDICATORS) with overview;
select * from "USER_APL"."MODEL_TRAIN_BIN";
select * from "USER_APL"."OPERATION_LOG";
select * from "USER_APL"."SUMMARY";
select * from "USER_APL"."INDICATORS";
```

In SAP HANA Cloud:

```
DO BEGIN
    DECLARE model       "SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
    DECLARE train_log   "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
    DECLARE train_sum   "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
    DECLARE train_indic "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";

    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_AND_TRAIN_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = SELECT * FROM  "USER_APL"."ADULT01_ANY";

    CALL
"_SYS_AFL"."APL_CREATE_MODEL_AND_TRAIN"(:header, :config, :var_desc,:var_role,:da
taset, model,  train_log, train_sum, train_indic);
    insert into  "USER_APL"."MODEL_TRAIN_BIN" select * from :model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :train_log;
    insert into  "USER_APL"."SUMMARY"         select * from :train_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :train_indic;

    select * from "USER_APL"."MODEL_TRAIN_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

6.7 Getting Progress Information

All SAP HANA APL functions are able to report progress information while they are running. This is particularly useful for situations where you expect feedback during long-running tasks, such as training or apply operations.

The progress information comprises all the messages issued by the Automated Analytics engine. The message amount is defined by the `LogLevel` parameter, which can be customised in the function header (`FUNC_HEADER` parameter) of the function.

Note the following:

- This option is only available if an operation identifier (`Oid`) has been set in the function header (`FUNC_HEADER`) parameter of the function call.
- It incurs a performance hit, so it is an opt-in capability.

- The visibility of the progress information is user-based and not session-based or task-based. In other words, the progress information is visible to the SAP HANA database user who triggered the function execution. As a result, if the same user is used to concurrently run different functions (for example, when a single SAP HANA technical user is used for all APL workflows), then this user has access to all the progress information of all the different APL function executions. A filter based on the connection ID or execution ID can be used in such cases.

Enable the progress information capability on a per-call basis by setting the `ProgressLog` parameter of the function header (`FUNC_HEADER` parameter).

The progress information is only available while the function is running, which means that it vanishes when the function completes.

The progress information is written to the following SAP HANA database table:

- Name: `FUNCTION_PROGRESS_IN_APL_AREA`
- Schema: `_SYS_AFL`
- Owner: `_SYS_AFL`

The table has the following structure:

Column	SQL Data Type	Description
`EXECUTION_ID`	VARCHAR(64)	Application-specific identifier of the task
`FUNCTION_NAME`	NVARCHAR(256)	Name of the involved application function
`HOST`	VARCHAR(64)	Host name
`PORT`	INTEGER	Internal port
`CONNECTION_ID`	INTEGER	Connection that triggered the task
`PROGRESS_TIMESTAMP`	TIMESTAMP	Absolute timestamp of the task progress event
`PROGRESS_ELAPSEDTIME`	INTEGER	Relative timestamp (in seconds) of the task progress event
`PROGRESS_CURRENT`	INTEGER	Steps of the task that are currently finished
`PROGRESS_MAX`	INTEGER	Maximum steps of the task
`PROGRESS_LEVEL`	VARCHAR(16)	Application-specific description of the task progress event level
`PROGRESS_MESSAGE`	NVARCHAR(256)	Formatted message for this progress event

Related Information

FUNC_HEADER [page 365]

6.8 Canceling the Execution of a Function

Some model training operations can take quite a long time, so you may decide at some point to just cancel the operation. By canceling a function, you terminate its execution. Functions that include a train, apply, or test operation support cancellation. The cancellation capability is disabled by default.

1. Enable the cancellation capability on a per-call basis by setting the `Cancelable` and `CancelCheckInterval` parameters in the function header (`FUNC_HEADER` table) of the function.

   If you don't enable the cancellation capability, you can still use the `ALTER SYSTEM CANCEL WORK IN SESSION` statement as described below, but the following applies:

   - The target operation to be canceled will actually run until its completion.
   - The target operation will end with an error.
   - No output will be generated.
2. Do one of the following to cancel the function:

Option	Description
In SAP HANA Studio	1. Go to the SQL Editor property dialog box to retrieve the session ID. 2. Go to the Performance/Sessions tab to cancel the function.

Via SQL	Run the following statements:
	1. To retrieve the session ID:
	`SELECT SESSION_CONTEXT('CONN_ID') FROM DUMMY`
	2. To cancel the function:
	`ALTER SYSTEM CANCEL [WORK IN] SESSION '<id>'`

   The following occurs:

   - The transaction is automatically rolled back when the query is canceled.
   - The operation that receives the cancellation request does not terminate immediately. Internally, the operation checks for cancel requests on a regular basis (every 10 seconds by default) and then forwards the request to the Automated Analytics engine, which might also cause some delay in the processing of the cancel request.

Example

```
-- the next select statement retrieves the current connection id, which can used
later on to cancel an AFL function execution
SELECT SESSION_CONTEXT('CONN_ID') FROM DUMMY;
-- execute the APL wrapper, which in turn executes the APL function
-- ...
-- some model training operations can be quite lengthy, so the user may decide
at some point to just cancel the current operation
-- assuming the current connection id is '400401', the operation can be
cancelled with the following SQL statement
ALTER SYSTEM CANCEL WORK IN SESSION '400401';
-- at this point, the operation that was running is going to terminate with an
error message.
```

Related Information

FUNC_HEADER [page 365]

6.8.1 Cancellable Functions

The cancellation capability allows a function to be canceled while it's running. Only functions that include a train, apply, or test operation support cancellation.

The following functions can be canceled:

Function Type	
Predictive Model Functions	CREATE_MODEL_AND_TRAIN [page 137]
	CREATE_MODEL_AND_TRAIN (3 Datasets) [page 142]
	TRAIN_MODEL [page 258]
	RETRAIN_MODEL [page 242]
	TEST_MODEL [page 251]
	APPLY_MODEL [page 72]
	APPLY_MODEL_AND_TEST [page 98]
	CREATE_RECO_MODEL_AND_TRAIN [page 157]
	APPLY_RECO_MODEL [page 108]
	CREATE_SOCIAL_MODEL_AND_TRAIN [page 173]
	GET_TABLE_TYPE_FOR_SOCIAL_APPLY [page 216]
	APPLY_SOCIAL_MODEL [page 120]
Predictive Business Functions	FORECAST [page 281]
	KEY_INFLUENCERS [page 304]
	SCORING_EQUATION [page 326]
	RECOMMEND [page 320]
	PROFILE_DATA [page 307]
	PROFILE_DATA_AND_GET_ASSOCIATIONRULES [page 314]
	COMPARE_DATA [page 273]

7 Analyzing Data with SAP HANA APL Functions

To analyze data with respect to a business question, you perform the following high-level tasks:

1. Generate an analysis model
2. Train the model on an input dataset
3. Apply the trained model to an application dataset

> **Note**
>
> For more information on the Automated Analytics models supported, see *Automated Analytics User Guides and Scenarios* on the SAP Help portal.

7.1 Creating the Model

Use the `CREATE_MODEL` function and provide the following input tables:

- The function header
- The operation configuration, which defines the type of analysis you want to perform (regression/classification, clustering, or timeseries).
- The training dataset

You can also use the `CREATE_MODEL_AND_TRAIN` function. You provide it with additional input tables containing variable roles and variable descriptions that show the relationships between the data in the training dataset.

> **Note**
>
> You must have a sufficiently large volume of data to be able to build a relevant and robust model. An analysis model that is generated from a dataset of 50 lines may have low generalization capacity, and contain low informative value.

7.2 Training the Model

Use the `TRAIN_MODEL` function and provide the following input tables:

- The function header
- The dataset, which includes the known target variables
- The operation configuration, which defines the type of analysis you want to perform

- The variable roles, which describe the relationships between the data and specify which column is the target variables
- The model description

The function returns the following output tables:

- The updated model, that takes into account the relationships between the data and the target variable in your training dataset
- The indicators table, which contains performance indicators such as the quality indicator (predictive power) and the robustness indicator (predictive confidence) of the model
- The training summary, which contains performance indicators provided by SAP HANA APL and an overview of the training operation provided by the Automated Analytics engine
- The log, which contains any status, warning and error messages returned by the function

The Training Dataset

The input dataset, also known as the training dataset, has a column that already contains the results to your business question. That column contains the business question results is the target variable, and must not have any empty cells. When you train the model, the model is updated to take into account the relationships between the data and the target variable of your training dataset. It is known as a trained model. You can retrain the model if you want to use another training dataset in order to create a more powerful model.

The training dataset must comply with the following rules:

- It must be presented in the form of a single table of data.
- It must have a sufficiently large volume of data to be able to build a valid model.
- It must contain a target variable that expresses your business issue. The target variable must be known for each observation of the training dataset. No target variable values may be missing over the range of the entire training dataset.
- It must be in a format that is supported by the package.

To allow validation of the model generated, the training dataset is cut into three subsets using a partition strategy.

The training dataset may correspond to either a complete population section of your database or a sample extracted from this population. The choice depends on the type of study to be performed, the tools used and the budget allocated to the study.

7.3 Applying the Model

You use the `APPLY_MODEL` function to get the target variables for your application dataset. You provide the following input tables:

- The function header
- The operation configuration, which defines the type of analysis you want to perform
- The model description

- The application dataset, which includes the known target variables

The `APPLY_MODEL` provides the output `DATASET` with target variable values, and an operation `LOG` table containing any status/warning/error messages.

The Application Dataset

An application dataset is a dataset to which you apply a trained model. This dataset contains an unknown target variable column for which you want to know the value. This column must be empty for all rows. The results returned include the output table that now has values for the target variable.

The model applied to the application dataset must have been previously generated from a training dataset. The application dataset must contain exactly the same information structure as the corresponding training dataset, it must have the following:

- The same number of variables
- The same types of variables
- The variables must be presented in the same order.
- The column for the target variable must be completely empty.

Getting the Output Table Type

You can use the `GET_TABLE_TYPE_FOR_APPLY` function to get the description of the output table that you should use for the `APPLY_MODEL` function. Provide the following as input:

- The function header
- The operation configuration, which defines the type of analysis you want to perform
- The model description
- The application dataset, which includes the known target variables

The function returns the expected table type definition of the output dataset for an apply operation.

7.4 Partition Strategies

The partition strategy defines how a training dataset is cut into three subsets: estimation, validation and testing datasets.

Depending on the model characteristics (model type, number of targets, and so on), not all cutting strategies can be used.

For regression/classification and clustering, you can use the following values for the cutting strategy:

- `'random'`
- `'periodic'`

- `'sequential'`
- `'periodic with test at end'`
- `'random with test at end'`
- `'sequential with no test'`
- `'periodic with no test'`
- `'random with no test'` (default value)

The following values are supported for setting the cutting strategy for time series:

- `'sequential'`
- `'sequential with no test'` (default value)

Partition Strategy 'random'

This cutting strategy distributes the data of the initial dataset in a random manner between the three subsets (estimation, validation and testing).

Partition Strategy 'periodic'

This strategy is implemented based on the following distribution cycle:

1. Three lines of the initial dataset are distributed to the estimation subset.
2. Three lines of the initial dataset are distributed to the validation subset.
3. One line is distributed to the testing subset.
4. Distribution begins again at step 1.

Partition Strategy 'sequential'

This strategy cuts the initial dataset into three blocks, corresponding to the following cutting proportions:

- The lines corresponding to the first 3/5 of the initial dataset are distributed as a block to the estimation dataset.
- The lines corresponding to the next 1/5 of the initial dataset are distributed as a block to the validation dataset.
- The lines corresponding to the final 1/5 of the initial dataset are distributed as a block to the testing dataset.

Partition Strategy 'periodic with test at end'

This strategy distributes:

- 4/5 of the initial dataset in a periodic manner to the two subsets estimation and validation, 3/5 being distributed to the estimation data subset and 1/5 to the validation data subset.
- The final 1/5 of the initial dataset is sent as a block of data to the test subset.

In other words, this strategy is based on the following distribution cycle:

1. Three lines of the first 4/5 of the initial dataset are distributed to the estimation subset.
2. One line of the first 4/5 of the initial dataset is distributed to the validation subset.
3. - If the entire 4/5 of the initial dataset is not yet distributed, distribution operations begin again at step 1.
   - If the entire 4/5 of the initial dataset has been distributed, distribution operations go to step 4
4. The final 1/5 of the initial dataset is sent as a block of data to the testing subset.

Partition Strategy 'random with test at end'

This strategy distributes:

- 4/5 of the initial dataset in a random manner to the two subsets estimation and validation, 3/5 being distributed to the estimation data subset and 1/5 to the validation data subset.
- The final 1/5 of the initial dataset is sent directly into the test subset.

This is a useful strategy in the following cases:

- Your database corresponds to a well-defined evolution because of the way it was built, which may mean, for example, that the data is in chronological order.
- You may wish to take this order into account when generating your model.

For example, imagine that the following applies:

- New customers are added every month to your database.
- You know that the datasets to which you apply the model will, once generated, have a better chance of resembling the most recent section of your database, that is, the section that contains the most recent customers entered.

Using the `random with test at the end` cutting strategy, you decide to test the model generated on that section of your database that is most likely to resemble the state of your future application datasets.

Partition Strategy 'sequential with no test'

This strategy cuts the initial dataset into two blocks as follows:

- The lines corresponding to the first 3/4 of the initial dataset are distributed as a block to the estimation dataset.
- The lines corresponding to the next 1/4 of the initial dataset are distributed as a block to the validation dataset.

As no test subset is used, all the data from your training dataset can be used for the estimation and validation subsets. This can lead to a model with better quality and robustness.

Partition Strategy 'periodic with no test'

This strategy distributes the whole initial dataset in a periodic manner to the two subsets estimation and validation as follows:

- 3/4 of the initial dataset are distributed to the estimation subset.
- 1/4 to the initial dataset is distributed to the validation subset.

In other words, this strategy is based on the following distribution cycle:

1. Three lines of the initial dataset are distributed to the estimation subset.
2. One line is distributed to the validation subset.
3. Distribution begins again at step 1.

As no test subset is used, all the data from your training dataset can be used for the estimation and validation subsets. This can lead to a model with better quality and robustness.

Partition Strategy 'random with no test'

This test strategy cuts the initial dataset into two blocks as follows:

- The lines corresponding to the first 3/4 of the initial dataset are distributed as a block to the estimation dataset.
- The lines corresponding to the next 1/4 of the initial dataset are distributed as a block to the validation dataset.

As no testing subset is used, all the data from your training dataset can be used for the estimation and validation subsets. This can lead to a model with better quality and robustness.

7.5 Variable Interactions

Information about the interactions of variables is part of the global explanation of the predictive model. It can help you understand how much a variable interacts with all the other variables in the model in comparison to its own main effect. Interaction values are calculated when a model is trained.

The computation method used by SAP HANA APL for interactions is based on the Shapley Taylor index (see *The Shapley Taylor Interaction Index*). Two statistical reports are provided:

- Variable Interaction Effect
- Variable Interaction Matrix

By default, the first report is filled and the second report is empty. To fill the interaction matrix report, you must set the `APL/Interactions` alias to true in the model training configuration. This functionality is only available for gradient boosting models.

Related Information

The Shapley Taylor Interaction Index
Variable Interaction Effect [page 456]
Variable Interaction Matrix [page 457]

7.6 Quantiles of Predicted Values

A quantile in SAP HANA APL is an integer value ranging from 1 to n. Each quantile value can be seen as a bucket number. When n=100 (that is, quantile type percentiles), each bucket contains roughly 1% of the population in the apply output dataset. When n=10 (that is, quantile type deciles), each bucket contains roughly 10% of the population.

Use Cases

You can use the 100 quantile (that is, percentiles) to filter the individual values in the apply output dataset for further analysis or action. For example, using a churn classification model, the marketing department can filter the individuals with a percentile less than or equal to 15 (around 15% of the population) to identify those individuals most likely to churn (according to the predictive model) in order to follow up with retention offers.

Approximate Quantiles Versus Exact Quantiles

The quantiles generated by SAP HANA APL are not exact quantiles but approximate quantiles.

Exact quantiles are equally sized buckets generated from the predicted values found in the SAP HANA APL apply output dataset. Exact quantiles can be obtained by running the SAP HANA SQL function NTILE on the apply output table.

Approximate quantiles in SAP HANA APL are based on the distribution of the predicted values as observed in the training dataset, not in the apply output dataset. Therefore, the content of the apply output dataset does not impact the definition of the approximate quantiles.

Output Dataset

Quantiles appear in the output dataset as generated columns named as follows:

Alias	Model Type	Output Column Name	Description
`APL/ApplyPredictedQuantile`	Regression/Classification (Legacy)	`quantile_rr_<target_name>_<bucket_number>`	The approximate quantile of the predicted value. The column is in the table if the value of `APL/ApplyPredictedQuantile` is greater than 0.
	Regression (Gradient Boosting)	`gb_quantile_<target_name>_<bucket_number>`	
	Binary Classification (Gradient Boosting)	`gb_quantile_<target_name>_<bucket_number>`	
`APL/ApplyPredictedAscendingQuantile`	Regression/Classification (Legacy)	`asc_quantile_rr_<target_name>_<bucket_number>`	The approximate quantile of the predicted value. The column is in the table if the value of `APL/ApplyPredictedAscendingQuantile` is greater than 0.
	Regression (Gradient Boosting)	`gb_asc_quantile_<target_name>_<bucket_number>`	
	Binary Classification (Gradient Boosting)	`gb_asc_quantile_<target_name>_<bucket_number>`	

Related Information

APL/ApplyExtraMode [page 85]
APPLY_OUT_TYPE [page 354]
NTILE Function (Window) (SAP HANA Cloud, SAP HANA Database SQL Reference Guide)
NTILE Function (Window) (SAP HANA SQL Reference Guide for SAP HANA Platform)

7.7 Interoperability between SAP HANA APL and Automated Analytics

The SAP Automated Analytics Modeler can read, save, and manage predictive models in SAP HANA tables, using an ODBC connection and the definition of so-called ODBC Stores. Meanwhile, SAP HANA APL reads and writes models using its own serialization format in tables that comply with a predefined table structure. The function library is fully interoperable with Automated Analytics:

- It can consume a model that's been produced by the Automated Analytics Modeler and saved into SAP HANA through an ODBC Store. This capability is transparently supported by all the functions that take a model as an input parameter.
  To correctly read an Automated Analytics model from an HANA ODBC Store, the table which contains the model needs to be passed through a SQL view that:
  - Filters out any information that doesn't belong to the target model. This can be done by filtering on the `NAME` and `VERSION` column of the model table.
  - Sorts the table rows by ID

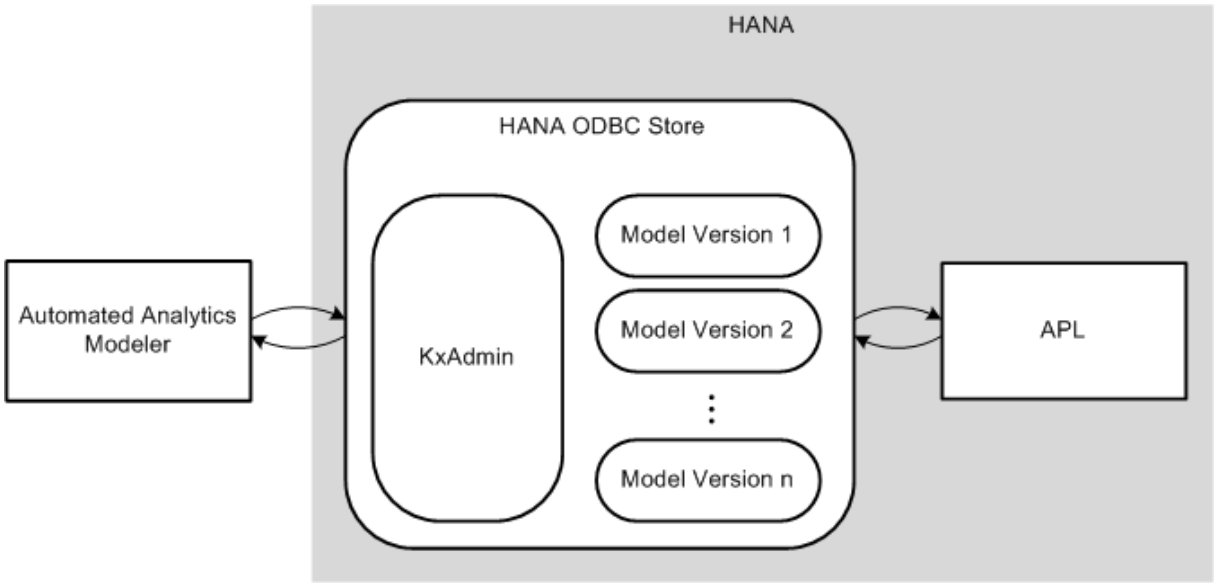
  For example:

  ```
  CREATE VIEW AA_MODEL_FOR_APL AS SELECT * FROM AA_MODELS WHERE NAME =
  'MyModel' AND VERSION = 1 ORDER BY "ID" ASC
  ```

- It can publish a model into an existing HANA ODBC Store managed by Automated Analytics. This is achieved with the `PUBLISH_MODEL` function. This function basically allows pushing a new model version into the Automated-Analytics-managed HANA ODBC Store, as if the Modeler had been used to save a new model version.
  To correctly publish a model compliant with the Automated Analytics Modeler, the `PUBLISH_MODEL` function needs to be used with some constraints:
  - The same `KXADMIN` table name should be used both as an input and as an output parameter of `PUBLISH_MODEL`. In fact, the `KXADMIN` table is first read by SAP HANA APL to extract metadata (especially the last model version), then it's updated with new information for the same model.
  - The `KXADMIN` table should be locked while `PUBLISH_MODEL` is running. This will garantee that there's no integrity issue regarding the model version information.
  - The APL alias `APL/ModelSpaceName` should designate the fully qualified name of the table that's going to contain the serialized model.

This interoperability allows implementing use cases where parts of the data-mining process are managed in the Automated Analytics Modeler, whereas other parts are deleguated to SAP HANA APL. For instance:

- The rich and complex model authoring worklow could be handled in the Automated Analytics Modeler.
- The training and execution of this authored model could be run using APL functions.
- The report-based debriefing of the model training could also be handled in the Automated Analytics Modeler.


HANA
HANA ODBC Store
Automated Analytics Modeler
KxAdmin
Model Version 1
Model Version 2
Model Version n
APL

8 Function Reference

The SAP HANA APL functions are listed by function type.

Each function reference provides the following information:

- What the function does
- Function SQL syntax that mention the names of the table types and parameters for each technique supported (direct technique, stored and ANY procedures)
- Description of tables types and parameters accepted in input and output
- List of operation configuration parameters supported if any
- Examples of SQL scripts that use the `APL_SAMPLES` schema

Related Information

Function Availability [page 38]

8.1 Predictive Model Functions

This group of functions focus on the predictive modeling activities. They allow you to create, train, update, query, and apply predictive models.

8.1.1 APPLY_MODEL

Applies a new model to a dataset.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, DATASET, LOG, SUMMARY)
```

The function signature can contain:

- `DATASET` as single output.
- `DATASET` and `LOG` as outputs only.

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
INPUT_DATASET_SCHEMA, INPUT_DATASET_NAME, OUTPUT_DATASET_SCHEMA, OUTPUT_DATASET_NAME,
LOG, SUMMARY)
```

The parameters `INPUT_DATASET_SCHEMA`, `INPUT_DATASET_NAME`, `OUTPUT_DATASET_SCHEMA`, and `OUTPUT_DATASET_NAME` are strings.

The stored procedure can accept the dataset as a single output.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_APPLY_MODEL__OVERLOAD_4_1"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
DATASET, DATASET)
```

Syntax

```
"_SYS_AFL"."APL_APPLY_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, DATASET,
LOG)
```

Syntax

```
"_SYS_AFL"."APL_APPLY_MODEL__OVERLOAD_4_3"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
DATASET, DATASET, LOG, SUMMARY)
```

Before trying to apply a model, you should make sure that the model is indeed available by checking the summary indicator of the training operation (that is, `'ModelAvailable'` is `true`). When applying a model on new data, an output dataset is produced with the predicted values and other indicators. The structure of this output dataset depends on the selected targets and on the selected extra-mode values (if any). In order to get the table type of the output dataset and correctly create the output dataset for the apply operation, you can do one of the following:

- Infer the table type, with simple rules
- Use the `GET_TABLE_TYPE_FOR_APPLY` function

Related Information

Parallel Apply [page 24]

8.1.1.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER [page 365]`	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL [page 382]`	Yes	The trained model to be applied. Most APL functions create, read, write, or update a predictive model. This model is specific to Automated Analytics.
IN	`OPERATION_CONFIG [page 384]`	No (The table must be provided but it can be empty.)	The configuration of the apply operation. For this function, you must declare the type of Automated Analytics model and you can declare the optional Apply Extra Mode in the `OPERATION_CONFIG` table as described in the next section.
IN	`DATASET [page 362]`	Yes	Your input dataset.
OUT	`DATASET [page 362]`	Yes	Your output dataset. This "ApplyOut" dataset must have the correct table structure. See GET_TABLE_TYPE_FOR_APPLY [page 213].
OUT	`LOG [page 381]`	No	The apply log. This output is optional.
OUT	`SUMMARY [page 388]`	No	The APPLY summary. This output is optional. The summary table is not generated if the table is not declared.

Related Information

GET_TABLE_TYPE_FOR_APPLY [page 213]

8.1.1.2 OPERATION_CONFIG Parameters

All Models

Key	Required	Supported Values	Description
`APL/ApplyExtraMode`	No	For regression/binary classification: • `'Min Extra'` • `'Individual Contributions'` • `'Decision'` • `'Quantiles'` • `'Advanced Apply Settings'` For binary classification (gradient boosting): • `'Score'` (default value) • `'Probability'` • `'Decision'` • `'BestProbabilityAndDecision'` • `'Individual Contributions'` • `'Advanced Apply Settings'` For regression (gradient boosting): • `'Score'` (default value) • `'Individual Contributions'` • `'Advanced Apply Settings'` For multinomial classification: • `'Score'` (default value) • `'Probability'` • `'Decision'` • `'BestProbabilityAndDecision'` • `'Individual Contributions'` • `'Advanced Apply Settings'` For clustering: • `'No Extra'` (default value)	Defines the information to be output in the ApplyOut result table. The result table has different columns depending on the value chosen. For table structure examples, see APPLY_OUT_TYPE [page 354]. Depending on the model, not all extra modes can be used. For more information about `APL/ApplyExtraMode`, see APL/ApplyExtraMode [page 85] and *Automated Analytics User Guides and Scenarios* on the SAP Help portal.

Key	Required	Supported Values	Description
		• 'For K2R' • 'For K2R copy data' • 'Target Mean' • 'Cluster copy data' • 'Advanced Apply Settings' For time series: • 'Forecasts and Error Bars' • 'Signal Components' • 'Component Residues' • 'Stable Components and Error Bars' • 'First Forecast with Stable Components and Residues and Error Bars'	
APL/ LastApplyRow	No	Integer	The last valid row that the system takes as part of the ApplyIn dataset. Default is to use all dataset rows.

Key	Required	Supported Values	Description
`APL/VariableConverter`	No	• `'abap_converter'` • `'default'`	`'abap_converter'` indicates that the following variables with an ABAP data type are converted into variables with a standard ISO 8601 format: • ABAP date (DATS) • ABAP time (TIMS) • ABAP short timestamp (TIMESTAMP/TZNTSTMPS) • ABAP long timestamp (TIMESTAMPL/TZNTSTMPL) Conversion is done during training or apply. `'default'` indicates that the guessed variable storage is kept as is in the model. **Note** To override the default behavior, you can set this alias to a specific variable by using the 3-column structure of the OPERATION_CONFIG [page 384] table type, for example: `insert into CREATE_AND_TRAIN_CONFIG values ('APL/VariableConverter','abap_converter', '<VariableName>');`
`APL/EnableMappingByPosForApply`	No	• `'true'` • `'false'` (default value)	Enables mapping based on the position of the columns in the ApplyIn dataset when the names of the the ApplyIn columns do not match the expected column names.

Binary Classification Models (Legacy)

Key	Required	Supported Values	Default	Description
`APL/TargetKey`	No	Binary target	The least frequent category of the Target variable	The category of the Target variable that will be used as the positive target in the model

Binary Classification Models (Legacy and Gradient Boosting)

Key	Required	Supported Values	Default	Description
APL/ApplyProbability	No	• 'true' • 'false'	'false'	Allows the probability of the row being a positive target to be generated in the output

Classification (Legacy), Binary Classification (Gradient Boosting), and Multinomial Classification Models

Key	Required	Supported Values	Default	Description
APL/ApplyDecision	No	• 'true' • 'false'	'false'	Allows the generation of the best decisions in the output
APL/ApplyProba	No	• 'true' • 'false'	'false'	Allows the generation of the probability of all target classes in the output. For binary classification models, only the probability of the positive class is provided.
APL/ApplyProbaDecision	No	• 'true' • 'false'	'false'	Allows the probability value associated with the classification decision to be generated in the output

Classification/Regression (Legacy), Regression (Gradient Boosting), Binary Classification (Gradient Boosting), and Multinomial Classification Models

Key	Required	Supported Values	Default	Description
APL/ApplyContribution	No	• 'none' • 'all' • '<var_name>;<var_name>'	'none'	Specifies whether contributions of variables are produced. '<var_name>;<var_name>' is not allowed with gradient boosting-based models (regression, binary classification, and multinomial classification). For gradient boosting models, the value 'all' returns the Shapley values that give the local explanation of each predicted value or score.
APL/ApplyPredictedValue	No	• 'true' • 'false'	'false'	Indicates whether the score value is in the output
APL/ApplyCopyVariables	No	• 'none' • 'all' • '<var_name>;<var_name>'	'none'	Allows all variables in the input dataset to be copied into the ApplyOut dataset

Key	Required	Supported Values	Default	Description
APL/UnpivotedExplanations	No	• 'true' • 'false'	'false'	Activates the unpivoted explanation columns
APL/UnpivotedExplanations/AbsoluteExplanationsCount	No	Integer	10	Defines the number of explanations sorted by absolute strength
APL/UnpivotedExplanations/PositiveExplanationsCount	No	Integer	0	Defines the maximum number of most positive explanations
APL/UnpivotedExplanations/NegativeExplanationsCount	No	Integer	0	Defines the maximum number of most negative explanations
APL/UnpivotedExplanations/PositiveOthers	No	• 'true' • 'false'	'true'	If 'true', aggregates the positive explanations to respect the constraints defined by ExplanationsCount
APL/UnpivotedExplanations/NegativeOthers	No	• 'true' • 'false'	'true'	If 'true', aggregates the negative explanations to respect the constraints defined by ExplanationsCount
APL/UnpivotedExplanations/ExplanationName	No	• 'true' • 'false'	'true'	If 'true', exports the name of the influencing column in the *Explanation_Influencer* column

Key	Required	Supported Values	Default	Description
`APL/UnpivotedExplanations/ExplanationValue`	No	• `'true'` • `'false'`	`'true'`	If `'true'`, exports the value of the influencing column in the *Explanation_Influencer_Value* column
`APL/UnpivotedExplanations/ExplanationStrength`	No	• `'true'` • `'false'`	`'true'`	If `'true'`, exports the strength of the influencing column in the *Explanation_Influencer_Strength* column
`APL/UnpivotedExplanations/ExplanationContribution`	No	• `'true'` • `'false'`	`'true'`	If `'true'`, exports the contribution of the influencing column in the *Explanation_Influencer_Contribution* column

Classification and Regression Models (Legacy)

Key	Required	Supported Values	Default	Description
APL/ApplyReasonCode	No	Concatenated string: "[Nb Reason];[Threshold];[Criterion];[Quality]" or "[Nb Reason];[Threshold];[Criterion]"	"[3;Mean;Below; false]"	• Nb Reason: The number of reason codes you want to generate • Threshold: Threshold used for computing the most important reason codes (Maximum, Minimum, Mean) • Criterion: Indicates whether you want to generate the reason codes when the customer variable contribution is above or below the threshold (Below, Above) • Quality: Generates the quality of the reason code (true, false). Quality is optional.

Quality Value	Impact of the Reason Code
>= 3	Strong, positive
Between 1 and 3	Meaningful, positive
Between 0 and 1	Small, positive
Between 0 and -1	Small, negative
Between -1 and -3	Meaningful, negative
<= 3	Strong, negative

Classification/Regression (Legacy) and Regression (Gradient Boosting) Models

Key	Required	Supported Values	Default	Description
APL/ApplyBar	No	• 'true' • 'false'	'false'	Allows the generation of the error bar in the output

Classification/Regression (Legacy), Regression (Gradient Boosting), and Binary Classification (Gradient Boosting) Models

Key	Required	Supported Values	Default	Description
`APL/ApplyOutlierFlag`	No	• `'true'` • `'false'`	`'false'`	Indicates whether the outlier flag associated with the score value is produced
`APL/ApplyPredictedAscendingQuantile`	No	Integer	`0`	Allows the quantiles of the predicted values to be generated in the output, sorted in ascending order. The lowest predicted values belong to the first quantile while the highest belong to the last quantile. Use 10 to obtain deciles or 100 to obtain percentiles, for example. A value of 0 means no quantiles will be generated in the output.
`APL/ApplyPredictedQuantile`	No	Integer	`0`	Allows the quantiles of the predicted values to be generated in the output, sorted in descending order. The highest predicted values belong to the first quantile while the lowest belong to the last quantile. Use 10 to obtain deciles or 100 to obtain percentiles, for example. A value of 0 means no quantiles will be generated in the output.

Binary Classification Models (Gradient Boosting)

Key	Required	Supported Values	Default	Description
`APL/ApplyDecisionThreshold`	No	Between 0 and 1	The default decision threshold is computed based on the distribution of the target class in the training data set.	Determines the level beyond which the model predicts the positive class. For example, a value of 0.5 represents a probability threshold of 50%.
`APL/ApplyBestProbability`	No	• `'true'` • `'false'`	`'false'`	Allows the probability associated with the predicted value to be generated in the output

Key	Required	Supported Values	Default	Description
`APL/OutlierPercentage`	No	Integer	1	Defines the desired percentage of detected outliers. Based on the cumulated number of prediction errors for all score categories, a score value is extracted that allows the targeted percentage of the worst prediction errors to be identified. These represent the outliers.
`APL/OutlierProbabilityErrorThreshold`	No	Between 0 and 1	0.9	Defines the probability error threshold for outlier detection. An outlier is detected when the probability error is greater than the specified probability error threshold.
`APL/OutlierStrategy`	No	• `'FromPredictedProbability'` • `'FromOutlierPercentage'`	`'FromOutlierPercentage'`	Defines the outlier detection strategy. The outlier detection strategy can be based on the predicted probability or a specified percentage of outliers.

Multinomial Classification Models

Key	Required	Supported Values	Default	Description
`APL/ApplyBestProbability`	No	• `'true'` • `'false'`	`'false'`	Allows the probability associated with the predicted value to be generated in the output
`APL/ApplyProbability`	No	• `'true'` • `'false'`	`'false'`	Allows the probability of all target classes to be generated in the output

Regression (Gradient Boosting), Binary Classification (Gradient Boosting), and Multinomial Classification Models

Key	Required	Supported Values	Default	Description
`APL/ApplyReasonCode/BottomCount`	No	Integer	`0`	Number of bottom explanatory variables (ranked by their respective strength) to output
`APL/ApplyReasonCode/RankOnAbsoluteValues`	No	• `'true'` • `'false'`	`'true'`	If `'true'`, rank the explanatory variables based on the absolute value of their respective strength

Key	Required	Supported Values	Default	Description
APL/ApplyReasonCode/ShowOtherStrength	No	• 'true' • 'false'	'false'	Output the aggregated strength value or/and indicator for remaining explanatory variables which are not part of top and bottom selection
APL/ApplyReasonCode/ShowStrengthIndicator	No	• 'true' • 'false'	'true'	Output a qualitative strength indicator for each top/bottom explanatory variable (strong, meaningful, and small positive or negative)
APL/ApplyReasonCode/ShowStrengthValue	No	• 'true' • 'false'	'false'	Output the strength value of each top/bottom explanatory variable
APL/ApplyReasonCode/TopCount	No	Integer	0	Number of top explanatory variables (ranked by their respective strength) to output

Clustering, Classification/Regression (Legacy), Regression (Gradient Boosting), Binary Classification (Gradient Boosting), and Multinomial Classification Models

Key	Required	Supported Values	Default	Description
APL/ApplyConstant/ApplyDate	No	• 'export' • 'skip'	'skip'	Adds the apply date to the output. Requires 'Advanced Apply Settings' as APL/ApplyExtraMode.
APL/ApplyConstant/BuildDate	No	• 'export' • 'skip'	'skip'	Adds the training date to the output. Requires 'Advanced Apply Settings' as APL/ApplyExtraMode.
APL/ApplyConstant/ModelName	No	• 'export' • 'skip'	'skip'	Adds the model name to the output. Requires 'Advanced Apply Settings' as APL/ApplyExtraMode.
APL/ApplyConstant/ModelVersion	No	• 'export' • 'skip'	'skip'	Adds the model version to the output. Requires 'Advanced Apply Settings' as APL/ApplyExtraMode.

Key	Required	Supported Values	Default	Description
`APL/SegmentColumnName`	No	String	No default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

Time Series Models

Key	Required	Supported Values	Default	Description
`APL/AppliedHorizon`	No	Integer		Indicates the number of forecasts you get after applying a time-series model to your dataset
`APL/ApplyLastTimePoint`	No	A date of which format is YYYY:MM:DD HH:mm:ss		Value in the time point column which represents the last date before the Apply of the model. For example, if the forecast period is from Jan 2018 to Dec 2018 and historical values correspond to the period from Jan 2005 to Dec 2017, this date will be Dec 2017. This alias is required if the dataset has extra predictable variables.
`APL/SegmentColumnName`	No	String	No default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

8.1.1.2.1 APL/ApplyExtraMode

SAP HANA APL provides a preconfigured set of fields that are output when models are applied. You can change the fields that are included in the output by using the `APL/ApplyExtraMode` alias. Different values are supported for the different types of models.

One of the values available for the `APL/ApplyExtraMode` alias is `'Advanced Apply Settings'`. The Advanced Apply Settings mode allows you to choose from a selection of options (aliases) to determine the combination of fields that should be included in the ApplyOut result table.

An overview of the `APL/ApplyExtraMode` values is provided below:

APL/ApplyExtraMode: Supported Values

Value	Description	Model
`'No Extra'`	Adds the cluster index to the output Default value This mode supports segmented modeling.	Clustering
`'Min Extra'`	Regression: Outputs the predicted value and the error bar if the target is a continuous target Classification: Outputs the score, probability, and the error bar on the probability if the target is a nominal target This mode supports segmented modeling.	Regression/binary classification
`'Individual Contributions'`	Allows the contribution of each variable to the score to be generated in the output. The sum of the contributions is approximately equal to the score value. In the case of a multinomial classification model, the individual contributions returned are the ones associated with the predicted class.	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
`'Decision'`	Allows the classification decision based on the class with the highest prediction probability to be generated in the output This mode supports segmented modeling.	Regression/binary classification Binary classification (gradient boosting) Multinomial classification
`'Quantiles'`	Cuts the output dataset into quantiles and assigns to each row the number of the quantile containing it Approximate quantiles are constructed based on the sorted distribution and the boundaries of predicted scores from the validation sample. The score boundaries are used to determine approximate quantiles on the apply dataset. This mode supports segmented modeling.	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting)

Value	Description	Model
`'Score'`	Adds the score of the target class to the output Default value This mode supports segmented modeling.	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
`'Probability'`	Allows the probability that the row is a positive target to be generated in the output	Binary classification (gradient boosting) Multinomial classification
`'BestProbabilityAndDecision'`	Allows the probability value associated with the classification decision to be generated in the output This mode supports segmented modeling.	Multinomial classification
`'For K2R'`	Adds the cluster index and its disjunctive coding to the output This mode supports segmented modeling.	Clustering
`'For K2R copy data'`	Adds the cluster index, its disjunctive coding, and all input data to the output This mode supports segmented modeling.	Clustering
`'Target Mean'`	Adds the cluster index, target mean, and target error for the cluster to the output This mode supports segmented modeling.	Clustering
`'Cluster copy data'`	Adds the cluster index and all input data to the output This mode supports segmented modeling.	Clustering

Value	Description	Model
`'Forecasts and Error Bars'`	Includes error bars with the forecast values in the output The error bar at a given horizon H is an interval centered around the forecasted value, whose width is calculated as 1.96 * 2* RMSE (Root Mean Square Error). RMSE is calculated on the actual vs predicted value observed in a validation set when predicting at horizon H. The weighted value of 1.96 corresponds to a confidence level of 95%. The value of 1.64 corresponds to a confidence level of 90%. The value of 2.58 corresponds to a confidence level of 99%. The error bar columns are `'kts_1_lowerlimit_95%'` and `'kts_1_upperlimit_95%'`. This mode supports segmented modeling.	Time series
`'Signal Components'`	Allows the component value (trend, cycles, fluctuation) for each forecast to be generated in the output	Time series
`'Component Residues'`	Allows the remaining values (residue) obtained after extracting each component from each forecast to be generated in the output	Time series
`'Stable Components and Error Bars'`	Includes forecast values, error bars, component values (trend, cycles, fluctuations) and their corresponding residuals in the output This mode supports segmented modeling.	Time series
`'First Forecast with Stable Components and Residues and Error Bars'`	Generates the same output as `'Stable Components and Error Bars'` but includes only the first horizon This mode supports segmented modeling.	Time series

Value	Description	Model
`'Advanced Apply Settings'`	Provides an additional selection of options for configuring the output. See Advanced Apply Settings [page 89]. This mode supports segmented modeling.	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification Clustering

Related Information

APPLY_OUT_TYPE [page 354]

8.1.1.2.1.1 Advanced Apply Settings

One of the values available for the `APL/ApplyExtraMode` alias is `'Advanced Apply Settings'`. The Advanced Apply Settings mode allows you to choose from a selection of options (aliases) to determine the combination of fields that should be included in the ApplyOut result table.

An overview of the `'Advanced Apply Settings'` options is provided below:

Advanced Apply Settings: Supported Aliases

Parameter	Model
`APL/ApplyBar`	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
`APL/ApplyBestProbability`	Binary classification (gradient boosting) Multinomial classification
`APL/ApplyConstant/ApplyDate`	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification Clustering
`APL/ApplyConstant/BuildDate`	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification Clustering
`APL/ApplyConstant/ModelName`	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification Clustering

Parameter	Model
APL/ApplyConstant/ModelVersion	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification Clustering
APL/ApplyContribution	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/ApplyCopyVariables	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial Classification
APL/ApplyDecision	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/ApplyDecisionThreshold	Binary classification (gradient boosting)
APL/ApplyOutlierFlag	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting)
APL/ApplyPredictedAscendingQuantile	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting)
APL/ApplyPredictedQuantile	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting)
APL/ApplyPredictedValue	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/ApplyProba	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/ApplyProbability	Binary classification Binary classification (gradient boosting) Multinomial classification
APL/ApplyProbaDecision	Binary classification (gradient boosting) Multinomial classification
APL/ApplyReasonCode	Regression/binary classification
APL/ApplyReasonCode/BottomCount	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification

Parameter	Model
APL/ApplyReasonCode/RankOnAbsoluteValues	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/ApplyReasonCode/ShowOtherStrength	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/ApplyReasonCode/ ShowStrengthIndicator	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/ApplyReasonCode/ShowStrengthValue	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/ApplyReasonCode/TopCount	Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/UnpivotedExplanations	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/UnpivotedExplanations/ AbsoluteExplanationsCount	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/UnpivotedExplanations/ ExplanationContribution	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/UnpivotedExplanations/ ExplanationName	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/UnpivotedExplanations/ ExplanationStrength	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
APL/UnpivotedExplanations/ ExplanationValue	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification

Parameter	Model
`APL/UnpivotedExplanations/NegativeExplanationsCount`	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
`APL/UnpivotedExplanations/NegativeOthers`	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
`APL/UnpivotedExplanations/PositiveExplanationsCount`	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification
`APL/UnpivotedExplanations/PositiveOthers`	Regression/binary classification Binary classification (gradient boosting) Regression (gradient boosting) Multinomial classification

For more information about these parameters, see OPERATION_CONFIG Parameters [page 75].

You can use the Advanced Apply Settings mode as shown in the example below:

Sample Code

Code sample `samples/sql/procedure/apl_samples/classification_regression/apl_apply_with_adv_setting.sql`

```
insert into APPLY_CONFIG values ('APL/ApplyExtraMode', 'Advanced Apply
Settings',null);
insert into APPLY_CONFIG values ('APL/ApplyProba','true',null);
insert into APPLY_CONFIG values ('APL/ApplyBar','true',null);
insert into APPLY_CONFIG values ('APL/ApplyDecision','true',null);
insert into APPLY_CONFIG values ('APL/ApplyProbaDecision','true',null);
…
```

Related Information

APPLY_OUT_TYPE [page 354]

8.1.1.2.1.2 APL/ApplyExtraMode with Segmented Modeling

The table below lists the `APL/ApplyExtraMode` values supported in segmented modeling.

APL/ApplyExtraMode: Supported Values in Segmented Modeling

Model	Value
Time Series	`'Forecasts and Error Bars'`
	`'No Extra'`
	`'Stable Components and Error Bars'` `'First Forecast with Stable Components and Residues and Error Bars'`
Regression (Gradient Boosting)	`'Score'`
	`'Advanced Apply Settings'`
Binary Classification (Gradient Boosting)	`'Decision'`
	`'Score'`
	`'BestProbabilityAndDecision'`
	`'Advanced Apply Settings'`
Multinomial Classification	`'Decision'`
	`'Score'`
	`'BestProbabilityAndDecision'`
	`'Advanced Apply Settings'`
Regression/Binary Classification	`'Min Extra'`
	`'Decision'`
	`'Quantiles'`
	`'Advanced Apply Settings'`
Clustering	`'No Extra'`
	`'For K2R'`
	`'For K2R copy data'`
	`'Target Mean'`
	`'Cluster copy data'`
	`'Advanced Apply Settings'`

The `Advanced Apply Settings` option allows a fine-grained specification of the output columns. The `Advanced Apply Settings` value of the `APL/ApplyExtraMode` alias is not prechecked before scoring, so output columns can be generated depending on the actual values found in each segment's input dataset. In such a case, a technical mapping error is triggered for each model's output that does not match the current output table.

Related Information

Segmented Modeling [page 26]

8.1.1.3 Example: DIRECT

```
-- ============================================================
-- APL_AREA, APPLY_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train.sql)
-- ----------------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- Ouput table type: dataset
drop type ADULT01_T_OUT;
create type ADULT01_T_OUT as table (
    "KxIndex" INTEGER,
    "class" INTEGER,
    "rr_class" DOUBLE
);
-- ----------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ----------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table APPLY_MODEL_SIGNATURE;
create column table APPLY_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into APPLY_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (4, 'USER_APL','ADULT01_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (5, 'USER_APL','ADULT01_T_OUT',
'OUT');
```

```
insert into APPLY_MODEL_SIGNATURE values (6, 'USER_APL','OPERATION_LOG_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_APPLY_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','APPLY_MODEL','USER_APL',
'APLWRAPPER_APPLY_MODEL', APPLY_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like OPERATION_CONFIG_T;
-- TODO: insert training configuration parameters (to be defined)
drop table ADULT01_APPLY;
create column table ADULT01_APPLY like ADULT01_T_OUT;
drop table APPLY_LOG;
create column table APPLY_LOG like OPERATION_LOG_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header       = select * from FUNC_HEADER;
    model_in     = select * from MODEL_TRAIN_BIN;
    apply_config = select * from APPLY_CONFIG;
    dataset      = select * from APL_SAMPLES.ADULT01;
    APLWRAPPER_APPLY_MODEL(:header, :model_in, :apply_config,  :dataset,
out_apply , out_apply_log);
    -- store result into table
    insert into  "USER_APL"."ADULT01_APPLY"   select * from :out_apply;
    insert into  "USER_APL"."APPLY_LOG"       select * from :out_apply_log;
    -- show result
    select * from "USER_APL"."ADULT01_APPLY";
    select * from "USER_APL"."APPLY_LOG";
END;
```

8.1.1.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, APPLY_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train_ex.sql)
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
-- Ouput table type: dataset
drop type ADULT01_T_OUT;
create type ADULT01_T_OUT as table (
    "KxIndex" INTEGER,
    "class" INTEGER,
    "rr_class" DOUBLE
);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
```

```
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
-- TODO: insert training configuration parameters (to be defined)
drop table ADULT01_APPLY;
create column table ADULT01_APPLY like ADULT01_T_OUT;
drop table APPLY_LOG;
create column table APPLY_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create column table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header       = select * from FUNC_HEADER;
    model_in     = select * from MODEL_TRAIN_BIN;
    apply_config = select * from APPLY_CONFIG;

"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL"(:header, :model_in, :apply_config,
'APL_SAMPLES','ADULT01', 'USER_APL','ADULT01_APPLY' , out_apply_log, out_sum);
    -- store result into table
    insert into  "USER_APL"."APPLY_LOG"       select * from :out_apply_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    -- show result
    select * from "USER_APL"."ADULT01_APPLY";
    select * from "USER_APL"."APPLY_LOG";
END;
```

8.1.1.5 Example: Parallel Apply PROCEDURE

```
-- ================================================================
-- APL_AREA, APPLY_MODEL, using a binary format for the model
--
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--  @depend(apl_createmodel_and_train_ex.sql)

connect USER_APL password Password1;

-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
-- Ouput table type: dataset
drop type ADULT01_T_OUT;
create type ADULT01_T_OUT as table (
    "KxIndex" INTEGER,
    "class" INTEGER,
    "rr_class" DOUBLE
);


-- ------------------------------------------------------------------------
```

```
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
-- ask 4 tasks
insert into FUNC_HEADER values ('MaxTasks', '4');


drop table APPLY_CONFIG;
create table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";

drop table ADULT01_APPLY;
create column table ADULT01_APPLY like ADULT01_T_OUT;

drop table APPLY_LOG;
create column table APPLY_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";

drop table SUMMARY;
create column table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";

-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header       = select * from FUNC_HEADER;
    model_in     = select * from MODEL_TRAIN_BIN;
    apply_config = select * from APPLY_CONFIG;


"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL"(:header, :model_in, :apply_config,
'APL_SAMPLES','ADULT01', 'USER_APL','ADULT01_APPLY' , out_apply_log, out_sum);

    -- store result into table
    insert into  "USER_APL"."APPLY_LOG"       select * from :out_apply_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;

    -- show result
    select * from "USER_APL"."ADULT01_APPLY";
    select * from "USER_APL"."APPLY_LOG";
END;

select 'Average time',AVG(to_double("VALUE")) from USER_APL.SUMMARY where "KEY"
= 'AplTaskElapsedTime'
UNION ALL
select 'Task time',"VALUE" from USER_APL.SUMMARY where "KEY" =
'AplTaskElapsedTime';
```

Related Information

Parallel Apply [page 24]

8.1.2 APPLY_MODEL_AND_TEST

Applies a model to an existing model and tests the new version of the model.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, DATASET, MODEL, LOG,
INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL_AND_TEST"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, INPUT_DATASET_SCHEMA, INPUT_DATASET_NAME, OUTPUT_DATASET_SCHEMA,
OUTPUT_DATASET_NAME, MODEL, LOG, INDICATORS)
```

The parameters `INPUT_DATASET_SCHEMA`, `INPUT_DATASET_NAME`, `OUTPUT_DATASET_SCHEMA`, and `OUTPUT_DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_APPLY_MODEL_AND_TEST"(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET,
DATASET, MODEL, LOG, INDICATORS)
```

Before trying to apply a model, you should make sure that the model is indeed available by checking the summary indicator of the training operation (that is, `'ModelAvailable'` is `true`). When applying a model on new data, an output dataset is produced with the predicted values and other indicators. The structure of this output dataset depends on the selected targets and on the selected extra-mode values (if any). In order to get the table type of the output dataset and correctly create the output dataset for the apply operation, you can do one of the following:

- Infer the table type, with simple rules
- Use the `GET_TABLE_TYPE_FOR_APPLY` function

8.1.2.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The trained model to be applied
IN	OPERATION_CONFIG [page 384]	No (The table must be provided but it can be empty.)	The configuration of the apply operation.
IN	DATASET [page 362]	Yes	The input dataset
OUT	DATASET [page 362]		The output dataset. This "ApplyOut" dataset needs to have the correct table structure. See GET_TABLE_TYPE_FOR_APPLY [page 213].
OUT	MODEL [page 382]		The updated model, with additional statistics
OUT	LOG [page 381]		The apply log
OUT	INDICATORS [page 370]		The variable statistics and indicators

8.1.2.2 OPERATION_CONFIG Parameters

See APPLY_MODEL [page 72].

8.1.2.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, APPLY_MODEL_AND_TEST, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train.sql)
-- ----------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------
connect USER_APL password Password1;
```

```
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- Ouput table type: dataset
drop type ADULT01_T_OUT;
create type ADULT01_T_OUT as table (
    "KxIndex" INTEGER,
    "class" INTEGER,
    "rr_class" DOUBLE
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table APPLY_MODEL_AND_TEST_SIGNATURE;
create column table APPLY_MODEL_AND_TEST_SIGNATURE like PROCEDURE_SIGNATURE_T;
insert into APPLY_MODEL_AND_TEST_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T',  'IN');
insert into APPLY_MODEL_AND_TEST_SIGNATURE values (2,
'USER_APL','MODEL_BIN_OID_T',     'IN');
insert into APPLY_MODEL_AND_TEST_SIGNATURE values (3,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into APPLY_MODEL_AND_TEST_SIGNATURE values (4,
'USER_APL','ADULT01_T',          'IN');
insert into APPLY_MODEL_AND_TEST_SIGNATURE values (5,
'USER_APL','ADULT01_T_OUT',      'OUT');
insert into APPLY_MODEL_AND_TEST_SIGNATURE values (6,
'USER_APL','MODEL_BIN_OID_T',    'OUT');
insert into APPLY_MODEL_AND_TEST_SIGNATURE values (7,
'USER_APL','OPERATION_LOG_T',    'OUT');
insert into APPLY_MODEL_AND_TEST_SIGNATURE values (8,
'USER_APL','INDICATORS_T',       'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_APPLY_MODEL_AND_TEST');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','APPLY_MODEL_AND_TEST',
'USER_APL', 'APLWRAPPER_APPLY_MODEL_AND_TEST', APPLY_MODEL_AND_TEST_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like OPERATION_CONFIG_T;
-- TODO: insert training configuration parameters (to be defined)
drop table ADULT01_APPLY_TEST;
create column table ADULT01_APPLY_TEST like ADULT01_T_OUT;
drop table APPLY_TEST_LOG;
create column table APPLY_TEST_LOG like OPERATION_LOG_T;
drop table APPLY_MODEL_AND_TEST_BIN;
```

```
create column table APPLY_MODEL_AND_TEST_BIN like MODEL_BIN_OID_T;
drop table APPLY_TEST_INDICATORS;
create table APPLY_TEST_INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from APPLY_CONFIG;
    model_in = select * from MODEL_TRAIN_BIN;
    dataset  = select * from APL_SAMPLES.ADULT01;
    APLWRAPPER_APPLY_MODEL_AND_TEST(:header, :model_in, :config, :dataset,
apply_out, out_model,  out_log,out_indic);
    insert into  "USER_APL"."ADULT01_APPLY_TEST"       select * from :apply_out;
    insert into  "USER_APL"."APPLY_MODEL_AND_TEST_BIN" select * from :out_model;
    insert into  "USER_APL"."APPLY_TEST_LOG"           select * from :out_log;
    insert into  "USER_APL"."APPLY_TEST_INDICATORS"    select * from :out_indic;
    select * from "USER_APL"."ADULT01_APPLY_TEST";
    select * from "USER_APL"."APPLY_MODEL_AND_TEST_BIN";
    select * from "USER_APL"."APPLY_TEST_LOG";
    select * from "USER_APL"."APPLY_TEST_INDICATORS";
END;
```

8.1.2.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, APPLY_MODEL_AND_TEST, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train_ex.sql)
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Ouput table type: dataset
drop type ADULT01_T_OUT;
create type ADULT01_T_OUT as table (
    "KxIndex" INTEGER,
    "class" INTEGER,
    "rr_class" DOUBLE
);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
-- TODO: insert training configuration parameters (to be defined)
drop table ADULT01_APPLY_TEST;
create column table ADULT01_APPLY_TEST like ADULT01_T_OUT;
drop table APPLY_TEST_LOG;
create column table APPLY_TEST_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
```

```
drop table APPLY_MODEL_AND_TEST_BIN;
create column table APPLY_MODEL_AND_TEST_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table APPLY_TEST_INDICATORS;
create table APPLY_TEST_INDICATORS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
select * from "USER_APL"."ADULT01_APPLY_TEST";
select * from "USER_APL"."APPLY_MODEL_AND_TEST_BIN";
select * from "USER_APL"."APPLY_TEST_LOG";
select * from "USER_APL"."APPLY_TEST_INDICATORS";
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from APPLY_CONFIG;
    model_in = select * from MODEL_TRAIN_BIN;

"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL_AND_TEST"(:header, :model_in, :config,
'APL_SAMPLES','ADULT01','USER_APL','ADULT01_APPLY_TEST', out_model,
out_log,out_indic);
    insert into  "USER_APL"."APPLY_MODEL_AND_TEST_BIN" select * from :out_model;
    insert into  "USER_APL"."APPLY_TEST_LOG"           select * from :out_log;
    insert into  "USER_APL"."APPLY_TEST_INDICATORS"    select * from :out_indic;
    select * from "USER_APL"."ADULT01_APPLY_TEST";
    select * from "USER_APL"."APPLY_MODEL_AND_TEST_BIN";
    select * from "USER_APL"."APPLY_TEST_LOG";
    select * from "USER_APL"."APPLY_TEST_INDICATORS";
END;
```

8.1.3 APPLY_MODEL_DEBRIEF

Applies a model on a new dataset and also generates its debrief tables.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, DATASET, LOG, SUMMARY,
DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL_DEBRIEF"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, INPUT_DATASET_SCHEMA, INPUT_DATASET_NAME, OUTPUT_DATASET_SCHEMA,
OUTPUT_DATASET_NAME, LOG, SUMMARY, DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

The parameters `INPUT_DATASET_SCHEMA`, `INPUT_DATASET_NAME`, `OUTPUT_DATASET_SCHEMA`, and `OUTPUT_DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APPLY_MODEL__OVERLOAD_4_5"(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET,
DATASET, LOG, SUMMARY, DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

Before trying to apply a model, you should make sure that the model is indeed available by checking the summary indicator of the training operation (that is, `'ModelAvailable'` is `true`). When applying a model on new data, an output dataset is produced with the predicted values and other indicators. The structure of this output dataset depends on the selected targets and on the selected extra-mode values (if any). In order to get the table type of the output dataset and correctly create the output dataset for the apply operation, you can do one of the following:

- Infer the table type, with simple rules
- Use the `GET_TABLE_TYPE_FOR_APPLY` function

8.1.3.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL` [page 382]	Yes	The trained model to be applied
IN	`OPERATION_CONFIG` [page 384]	No (The table must be provided but it can be empty.)	The configuration of the apply operation.
IN	`DATASET` [page 362]	Yes	The input dataset
OUT	`DATASET` [page 362]		The output dataset. This "ApplyOut" dataset needs to have the correct table structure. See GET_TABLE_TYPE_FOR_APPLY [page 213].
OUT	`LOG` [page 381]		The apply log
OUT	`SUMMARY` [page 388]		The apply summary
OUT	DEBRIEF_METRIC [page 364]	Yes	The debrief metrics
OUT	DEBRIEF_PROPERTY [page 365]	Yes	The debrief properties

8.1.3.2 OPERATION_CONFIG Parameters

See APPLY_MODEL [page 72].

In addition, the following alias is also supported.

Model	Parameter Name	Required	Supported Values	Description
Time Series	`APL/LocalExplanations/Activate`	No	• `'true'` • `'false'` (default value)	Generates information about the decomposition of the forecasted value for each forecast period for use in the statistical report `TimeSeries_ForecastBreakdown`

8.1.3.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, call multiple functions for a complete use case
--
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- CREATE_MODEL
-- TRAIN_MODEL
-- APPLY_MODEL
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
drop view CENSUS_SORTED;
create view CENSUS_SORTED as select "age", "workclass", "fnlwgt", "education",
"education-num", "marital-status", "occupation", "relationship", "race", "sex",
"capital-gain", "capital-loss", "hours-per-week", "native-country", "class" from
APL_SAMPLES.CENSUS order by "id";
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create COLUMN table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', 'foobar');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_CONFIG;
create COLUMN table CREATE_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into CREATE_CONFIG (KEY, VALUE) values ('APL/ModelType',
'classification');
drop table TRAIN_CONFIG;
create COLUMN table TRAIN_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into TRAIN_CONFIG (KEY, VALUE) values ('APL/CuttingStrategy', 'random');
drop table VARIABLE_ROLES;
create COLUMN table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
insert into VARIABLE_ROLES (NAME, ROLE) values ('age', 'target');
insert into VARIABLE_ROLES (NAME, ROLE) values ('class', 'input');
drop table TRAIN_LOG;
create COLUMN table TRAIN_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create COLUMN table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
```

```
create COLUMN table INDICATORS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table MODEL_CREATE_BIN_OUT;
create COLUMN table MODEL_CREATE_BIN_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table MODEL_GUESS_BIN_OUT;
create COLUMN table MODEL_GUESS_BIN_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table MODEL_TRAIN_BIN_OUT;
create COLUMN table MODEL_TRAIN_BIN_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table VARDESC_OUT;
create COLUMN table VARDESC_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header = select * from FUNC_HEADER;
    config = select * from CREATE_CONFIG;
    train_config = select * from TRAIN_CONFIG;
    var_roles = select * from VARIABLE_ROLES;
    call "SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL"(:header, :config,
'USER_APL', 'CENSUS_SORTED', out_model,out_var_desc);
    call
"SAP_PA_APL"."sap.pa.apl.base::TRAIN_MODEL"(:header, :out_model, :train_config, :
var_roles, 'USER_APL', 'CENSUS_SORTED', out_trained_model, out_logs,
out_summary, out_indicators);
    -- store result into table
    insert into  "USER_APL"."MODEL_TRAIN_BIN_OUT" select *
from :out_trained_model;
END;
-- Ouput table type: dataset
drop type CENSUS_T_OUT;
create type CENSUS_T_OUT as table (
    "KxIndex" INTEGER,
    "age" INTEGER,
    "rr_age" INTEGER
);
drop table APPLY_OUT;
create column table APPLY_OUT like CENSUS_T_OUT;
drop table APPLY_LOG;
create COLUMN table APPLY_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table APPLY_CONFIG;
create COLUMN table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
drop table DEBRIEF_METRIC;
create table DEBRIEF_METRIC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_METRIC_OID";
drop table DEBRIEF_PROPERTY;
create table DEBRIEF_PROPERTY like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_PROPERTY_OID";
DO BEGIN
    header = select * from FUNC_HEADER;
    model = select * from MODEL_TRAIN_BIN_OUT;
    config = select * from APPLY_CONFIG;
    call
"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL_DEBRIEF"(:header,  :model, :config,
'USER_APL', 'CENSUS_SORTED', 'USER_APL', 'APPLY_OUT', out_apply_log,
out_apply_sum, out_debrief_metric, out_debrief_property);
    -- store result into table
    insert into  "USER_APL"."DEBRIEF_METRIC" select * from :out_debrief_metric;
    insert into  "USER_APL"."DEBRIEF_PROPERTY" select *
from :out_debrief_property;
END;
```

```
select "Oid", "Partitioning Method", "Partition", "Weight", "% Weight" from
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_Partition"(DEBRIEF_PROPERTY,
DEBRIEF_METRIC) order by "Oid", "Partition";
```

8.1.3.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, call multiple functions for a complete use case
--
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- CREATE_MODEL
-- TRAIN_MODEL
-- APPLY_MODEL
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
drop view CENSUS_SORTED;
create view CENSUS_SORTED as select "age", "workclass", "fnlwgt", "education",
"education-num", "marital-status", "occupation", "relationship", "race", "sex",
"capital-gain", "capital-loss", "hours-per-week", "native-country", "class" from
APL_SAMPLES.CENSUS order by "id";
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create COLUMN table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', 'foobar');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_CONFIG;
create COLUMN table CREATE_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into CREATE_CONFIG (KEY, VALUE) values ('APL/ModelType',
'classification');
drop table TRAIN_CONFIG;
create COLUMN table TRAIN_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into TRAIN_CONFIG (KEY, VALUE) values ('APL/CuttingStrategy', 'random');
drop table VARIABLE_ROLES;
create COLUMN table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
insert into VARIABLE_ROLES (NAME, ROLE) values ('age', 'target');
insert into VARIABLE_ROLES (NAME, ROLE) values ('class', 'input');
drop table TRAIN_LOG;
create COLUMN table TRAIN_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create COLUMN table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create COLUMN table INDICATORS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table MODEL_CREATE_BIN_OUT;
create COLUMN table MODEL_CREATE_BIN_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table MODEL_GUESS_BIN_OUT;
create COLUMN table MODEL_GUESS_BIN_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table MODEL_TRAIN_BIN_OUT;
```

```
create COLUMN table MODEL_TRAIN_BIN_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table VARDESC_OUT;
create COLUMN table VARDESC_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header = select * from FUNC_HEADER;
    config = select * from CREATE_CONFIG;
    train_config = select * from TRAIN_CONFIG;
    var_roles = select * from VARIABLE_ROLES;

    call "SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL"(:header, :config,
'USER_APL', 'CENSUS_SORTED', out_model,out_var_desc);
    call
"SAP_PA_APL"."sap.pa.apl.base::TRAIN_MODEL"(:header, :out_model, :train_config, :
var_roles, 'USER_APL', 'CENSUS_SORTED', out_trained_model, out_logs,
out_summary, out_indicators);

    -- store result into table
    insert into  "USER_APL"."MODEL_TRAIN_BIN_OUT" select *
from :out_trained_model;
END;
-- Ouput table type: dataset
drop type CENSUS_T_OUT;
create type CENSUS_T_OUT as table (
    "KxIndex" INTEGER,
    "age" INTEGER,
    "rr_age" INTEGER
);
drop table APPLY_OUT;
create column table APPLY_OUT like CENSUS_T_OUT;
drop table APPLY_LOG;
create COLUMN table APPLY_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table APPLY_CONFIG;
create COLUMN table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
drop table DEBRIEF_METRIC;
create table DEBRIEF_METRIC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_METRIC_OID";
drop table DEBRIEF_PROPERTY;
create table DEBRIEF_PROPERTY like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_PROPERTY_OID";
DO BEGIN
    header = select * from FUNC_HEADER;
    model = select * from MODEL_TRAIN_BIN_OUT;
    config = select * from APPLY_CONFIG;

    call
"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL_DEBRIEF"(:header,  :model, :config,
'USER_APL', 'CENSUS_SORTED', 'USER_APL', 'APPLY_OUT', out_apply_log,
out_apply_sum, out_debrief_metric, out_debrief_property);

    -- store result into table
    insert into  "USER_APL"."DEBRIEF_METRIC" select * from :out_debrief_metric;
    insert into  "USER_APL"."DEBRIEF_PROPERTY" select *
from :out_debrief_property;
END;
select "Oid", "Partitioning Method", "Partition", "Weight", "% Weight" from
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_Partition"(DEBRIEF_PROPERTY,
DEBRIEF_METRIC) order by "Oid", "Partition";
```

8.1.4 APPLY_RECO_MODEL

Applies a recommendation model to the dataset.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, NODES1, NODES2,LINKS, OPERATION_CONFIG, DATASET, RECO_CODE, LOG)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::APPLY_RECO_MODEL"(FUNC_HEADER, MODEL, NODES1_SCHEMA, NODES1_NAME, NODES2_SCHEMA, NODES2_NAME, LINKS_SCHEMA, LINKS_NAME, OPERATION_CONFIG, DATASET_SCHEMA, DATASET_NAME, RECO_CODE_SCHEMA, RECO_CODE_NAME, LOG)
```

The parameters `NODES1_SCHEMA`, `NODES1_NAME`, `NODES2_SCHEMA`, `NODES2_NAME`, `LINKS_SCHEMA`, `LINKS_NAME`, `DATASET_SCHEMA`, `DATASET_NAME`, `RECO_CODE_SCHEMA`, and `RECO_CODE_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_APPLY_RECO_MODEL"(FUNC_HEADER, MODEL, NODES1, NODES2,LINKS, OPERATION_CONFIG, DATASET, RECO_CODE, LOG)
```

8.1.4.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL` [page 382]		The model
IN	`NODES` [page 411]		Graph node storage (repository 1)
IN	`NODES` [page 411]		Graph node storage (repository 2)
IN	`LINKS` [page 410]		Graph link storage
IN	`OPERATION_CONFIG` [page 384]	Yes	The configuration of the training operation

Direction	Type	Required	Description
IN	DATASET [page 362]	Yes	The name of your training dataset
OUT	RECO_SCORE [page 411]		The resulting recommendation score
OUT	LOG [page 381]		The training log produced by the AutomatedAnalytics engine

8.1.4.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Default	Description
APL/IncludeBestSellers	No		'false'	Includes de-facto best seller items into the suggestion list
APL/SkipAlreadyOwned	No		'true'	Excludes the items already owned from the suggestion list
APL/Top	No		max	Keeps the top N of recommended items ranked by confidence

Related Information

APPLY_MODEL [page 72]

8.1.4.3 Example: DIRECT

```
-- ====================================================================
-- APL_AREA, APPLY_RECO_MODEL, using a native format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_SORTED, MODEL_NODES1, MODEL_NODES2, MODEL_LINKS tables.
--               For instance you have used apl_create_reco_model_and_train.sql
before.
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- KxNodes1
drop type MODEL_RECO_NODES1_T;
```

```
create type MODEL_RECO_NODES1_T as table (
    "node" INT -- must be of the same SQL type as the User column (UserID here)
);
-- KxNodes2
drop type MODEL_RECO_NODES2_T;
create type MODEL_RECO_NODES2_T as table (
    "node" NVARCHAR(255) -- must be of the same SQL type as the Item column
(ItemPurchased here)
);
-- KxLinks
drop type MODEL_RECO_LINKS_T;
create type MODEL_RECO_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" INT, -- must be of the same SQL type as the User column
(UserID here)
    "KXNODESECOND" NVARCHAR(255), -- must be of the same SQL type as the Item
column (ItemPurchased here)
    "KXNODESECOND_2" NVARCHAR(255) -- must be of the same SQL type as the Item
column (ItemPurchased here)
);
drop type RECO_SCORE_T;
create type RECO_SCORE_T as table (
    "UserID" INTEGER,
    "ItemPurchased" NVARCHAR(128),
    "sn_rec_rule_id" INTEGER,
    "sn_rec_kxReco" NVARCHAR(128),
    "sn_rec_source"  NVARCHAR(128),
    "sn_rec_score" DOUBLE
);
drop type USER_T;
create type USER_T as table (
    "UserID" INTEGER -- must be of the same SQL type as the User column (UserID
here)
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table APPLY_MODEL_SIGNATURE;
create column table APPLY_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into APPLY_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_NATIVE_T', 'IN');
insert into APPLY_MODEL_SIGNATURE values (3, 'USER_APL','MODEL_RECO_NODES1_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (4, 'USER_APL','MODEL_RECO_NODES2_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (5, 'USER_APL','MODEL_RECO_LINKS_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (6, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (7, 'USER_APL','USER_T',          'IN');
insert into APPLY_MODEL_SIGNATURE values (8, 'USER_APL','RECO_SCORE_T',
'OUT');
insert into APPLY_MODEL_SIGNATURE values (9, 'USER_APL','OPERATION_LOG_T',
'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_APPLY_RECO_MODEL');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','APPLY_RECO_MODEL','USER_APL',
'APLWRAPPER_APPLY_RECO_MODEL', APPLY_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
```

```
insert into FUNC_HEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like OPERATION_CONFIG_T;
-- TODO: insert training configuration parameters (to be defined)
insert into APPLY_CONFIG values ('APL/Top', '5');                          --
optional (default: max)
insert into APPLY_CONFIG values ('APL/IncludeBestSellers', 'false');       --
optional (default: false)
insert into APPLY_CONFIG values ('APL/SkipAlreadyOwned', 'true');          --
optional (default: true)
drop table USER_APPLYIN;
create column table USER_APPLYIN like USER_T;
insert into USER_APPLYIN values ('23');
drop table USER_APPLYOUT;
create column table USER_APPLYOUT like RECO_SCORE_T;
drop table APPLY_LOG;
create column table APPLY_LOG like OPERATION_LOG_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
drop view MODEL_SORTED;
create view MODEL_SORTED AS SELECT *  from MODEL_RECO order by "ID" asc  ;
DO BEGIN
    header       = select * from FUNC_HEADER;
    model        = select * from MODEL_SORTED;
    model_nodes1 = select * from MODEL_NODES1;
    model_nodes2 = select * from MODEL_NODES2;
    model_links  = select * from MODEL_LINKS;
    config       = select * from APPLY_CONFIG;
    apply_in     = select * from USER_APPLYIN;

APLWRAPPER_APPLY_RECO_MODEL(:header, :model, :model_nodes1, :model_nodes2, :model
_links, :config, :apply_in, out_apply, out_log);
    -- store result into table
    insert into  "USER_APL"."USER_APPLYOUT"   select * from :out_apply;
    insert into  "USER_APL"."APPLY_LOG"       select * from :out_log;
    -- show result
    select * from "USER_APL"."USER_APPLYOUT";
    select * from "USER_APL"."APPLY_LOG";
END;
```

8.1.4.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, APPLY_RECO_MODEL, using a native format for the model
--
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid trained model (created by APL) in the
MODEL_SORTED, MODEL_NODES1, MODEL_NODES2, MODEL_LINKS tables.
--               For instance you have used
apl_create_reco_model_and_train_ex.sql before.
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- KxNodes1
drop type MODEL_RECO_NODES1_T;
create type MODEL_RECO_NODES1_T as table (
    "node" INT -- must be of the same SQL type as the User column (UserID here)
);
```

```
-- KxNodes2
drop type MODEL_RECO_NODES2_T;
create type MODEL_RECO_NODES2_T as table (
    "node" NVARCHAR(255) -- must be of the same SQL type as the Item column
(ItemPurchased here)
);
-- KxLinks
drop type MODEL_RECO_LINKS_T;
create type MODEL_RECO_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" INT, -- must be of the same SQL type as the User column
(UserID here)
    "KXNODESECOND" NVARCHAR(255), -- must be of the same SQL type as the Item
column (ItemPurchased here)
    "KXNODESECOND_2" NVARCHAR(255) -- must be of the same SQL type as the Item
column (ItemPurchased here)
);
drop type RECO_SCORE_T;
create type RECO_SCORE_T as table (
    "UserID" INTEGER,
    "ItemPurchased" NVARCHAR(128),
    "sn_rec_rule_id" INTEGER,
    "sn_rec_kxReco" NVARCHAR(128),
    "sn_rec_source"  NVARCHAR(128),
    "sn_rec_score" DOUBLE
);
drop type USER_T;
create type USER_T as table (
    "UserID" INTEGER -- must be of the same SQL type as the User column (UserID
here)
);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
-- TODO: insert training configuration parameters (to be defined)
insert into APPLY_CONFIG values ('APL/Top', '5',null);                          --
optional (default: max)
insert into APPLY_CONFIG values ('APL/IncludeBestSellers', 'false',null);
-- optional (default: false)
insert into APPLY_CONFIG values ('APL/SkipAlreadyOwned', 'true',null);       --
optional (default: true)
drop table USER_APPLYIN;
create column table USER_APPLYIN like USER_T;
insert into USER_APPLYIN values ('23');
drop table USER_APPLYOUT;
create column table USER_APPLYOUT like RECO_SCORE_T;
drop table APPLY_LOG;
create column table APPLY_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header       = select * from FUNC_HEADER;
    model        = select * from MODEL_RECO;
    config       = select * from APPLY_CONFIG;
    "SAP_PA_APL"."sap.pa.apl.base::APPLY_RECO_MODEL"(:header, :model,
'USER_APL','MODEL_NODES1', 'USER_APL', 'MODEL_NODES2',
```

```
'USER_APL','MODEL_LINKS', :config,'USER_APL','USER_APPLYIN',
'USER_APL','USER_APPLYOUT', out_log);
    -- store result into table
    insert into  "USER_APL"."APPLY_LOG"       select * from :out_log;
    -- show result
    select * from "USER_APL"."USER_APPLYOUT";
    select * from "USER_APL"."APPLY_LOG";
END;
```

8.1.5 APPLY_RECOMMENDER_TO_BASKET

Generates recommendations from a recommendation model for a user of a dataset and a series of items.

PROCEDURE

Syntax

`"SAP_PA_APL"."sap.pa.apl.base::APPLY_RECOMMENDER_TO_BASKET"` (`FUNC_HEADER`, '`model_schema`', '`model_name`', `OPERATION_CONFIG`, '`basketItems_schema`', '`basketItems_name`', '`transactions_schema`', '`transactions_name`', '`recommendations_schema`', '`recommendations_name`', `SUMMARY`)

→ Remember

The direct technique and the ANY procedure technique do not support this function.

8.1.5.1 Input/Output Parameters

Direction	Parameter	Data Type	Required	Description
IN	`model_schema`	NVARCHAR(127)	No	The name of the schema of the table that contains the recommendation model. If null, the current schema is used.
IN	`model_name`	NVARCHAR(127)	Yes	The reference to the name of the table that contains the recommendation model used
IN	`basketItems_schema`	NVARCHAR(127)	No	The name of the schema of the table that contains the items to which the model is applied. If null, the current schema is used.
IN	`basketItems_name`	NVARCHAR(127)	Yes	The reference to the name of the table that contains the items to which the model is applied. The table must have an `ItemID` column.
IN	`transactions_schema`	NVARCHAR(127)	No	The name of the schema used for the table that contains the dataset. If null, the current schema is used.

Direction	Parameter	Data Type	Required	Description
IN	`transactions_name`	NVARCHAR(127)	No	The reference to the name of the table that contains the dataset. Use it with `APL/UserID` and `APL/SkipAlreadyOwned` set to `true` to filter the recommendations according to the user history.
IN	`recommendations_schema`	NVARCHAR(127)	No	The name of the schema of the table that contains the recommendations. If null, the current shema is used.
IN	`recommendations_name`	NVARCHAR(127)	Yes	The reference to the name of the table that contains the recommendations. Will be created if it does not exist.

8.1.5.2 Input/Output Tables

Direction	Table	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	`sap.pa.apl.base::BASE.T.FUNCTION_HEADER`	No	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`OPERATION_CONFIG` [page 384]	`sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED`	No (The table must be provided but it can be empty.)	The operation configuration parameters. See table below.
OUT	`SUMMARY` [page 388]	`sap.pa.apl.base::BASE.T.SUMMARY`		The summary of the results

8.1.5.3 OPERATION_CONFIG Parameters

Key	Required	Description	Default
`APL/Date`	No	The name of the column containing the date in the transaction table	

Key	Required	Description	Default
APL/EndDate	No	The end date of the time period considered in the transaction table. APL/Date must be specified.	
APL/FillWithBestSellers	No	A flag indicating whether a recommendation list is completed with bestsellers in order to reach the maximum number of recommendations per user. If true or 1 with APL/Top specified, the function adds bestsellers to the list.	false
APL/Item	No	The name of the column containing the item ID in the transaction table	
APL/Metric	No	The metrics used as score to sort the recommendations. Possible values are: • CONFIDENCE • SUPPORT • KI • LIFT • ADDED_VALUE • COSINE	CONFIDENCE
APL/SkipAlreadyOwned	No	A flag indicating whether an item which is already part of a user transaction history is removed from the recommendation list. If true or 1 with APL/UserID specified, the item cannot be recommended.	true
APL/StartDate	No	The start date of the time period considered in the transaction table. APL/Date must be specified.	
APL/Top	No	The maximum number of recommendations per user. Only best recommendations according to APL/metric are kept. There is no maximum if not specified.	
APL/User	No	The name of the column containing the user ID in the transaction table	
APL/UserID	No	The user ID to specify along with transactions_name et APL/SkipAlreadyOwned set to true, to filter recommendations	

8.1.5.4 Example

```
-- ================================================================
-- APL_AREA, APPLY_RECOMMENDER_TO_BASKET, there is no model generated
-- This script uses a recommender to generate recommendations for a basket of
items
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid recommender (created by APL) in the
RECOMMENDER_CUST_TRANSACTIONS table.
```

```
--               For instance you have used apl_create_recommender_ex.sql before.
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create COLUMN table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
-- Prepare
DROP TABLE RECO_CONFIG;
CREATE TABLE RECO_CONFIG LIKE
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into RECO_CONFIG values ('APL/Top', '5',
NULL);                                -- optional (default: NULL)
insert into RECO_CONFIG values ('APL/SkipAlreadyOwned', 'true',
NULL);              -- optional (default: true)
insert into RECO_CONFIG values ('APL/Metric', 'CONFIDENCE',
NULL);                   -- optional (default: CONFIDENCE) (values: CONFIDENCE,
SUPPORT, KI, LIFT, COSINE, ADDED_VALUE)
insert into RECO_CONFIG values ('APL/FillWithBestSellers', 'false',
NULL);          -- optional (default: false)
insert into RECO_CONFIG values ('APL/UserID', '23',
NULL);                           -- optional (needed if we want to skip already
owned items)
insert into RECO_CONFIG values ('APL/User', 'UserID',
NULL);                         -- optional (needed if we want to skip already
owned items)
insert into RECO_CONFIG values ('APL/Item', 'ItemPurchased',
NULL);                 -- optional (needed if we want to skip already owned
items)
--insert into RECO_CONFIG values ('APL/Date', 'Date_PutInCaddy',
NULL);               -- optional (default: NULL)
--insert into RECO_CONFIG values ('APL/StartDate', '2000-12-20 17:32:24',
NULL);      -- optional (default: NULL)
--insert into RECO_CONFIG values ('APL/EndDate', '2001-03-26 17:32:24',
NULL);        -- optional (default: NULL)
-- The items in the basket MUST be available in a column named "ItemID"
DROP TABLE BASKET_CUST;
CREATE TABLE BASKET_CUST ("ItemID" nvarchar(18));
INSERT INTO BASKET_CUST values ('Pullovers');
INSERT INTO BASKET_CUST values ('Polo');
INSERT INTO BASKET_CUST values ('Shirts');
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
DROP TABLE RECOMMENDATIONS_FOR_BASKET;
-- The table will be created directly by the procedure if it does not already
exist
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header       = select * from FUNC_HEADER;
    config       = select * from RECO_CONFIG;
   "SAP_PA_APL"."sap.pa.apl.base::APPLY_RECOMMENDER_TO_BASKET"(:header,
'USER_APL', 'RECOMMENDER_CUST_TRANSACTIONS', :config, 'USER_APL', 'BASKET_CUST',
'APL_SAMPLES', 'CUST_TRANSACTIONS', 'USER_APL', 'RECOMMENDATIONS_FOR_BASKET',
out_summary);
    -- store result into table
    insert into  "USER_APL"."SUMMARY"       select * from :out_summary;
    -- show result
    --select * from "USER_APL"."RECOMMENDATIONS_FOR_BASKET";
    select * from "USER_APL"."SUMMARY";
END;
```

8.1.6 APPLY_RECOMMENDER_TO_USERS

Generates recommendations from a recommendation model for all users or a subset of users of a dataset.

PROCEDURE

> Syntax
>
> `"SAP_PA_APL"."sap.pa.apl.base::APPLY_RECOMMENDER_TO_USERS"(FUNC_HEADER, 'model_schema', 'model_name', OPERATION_CONFIG, 'transactions_schema', 'transactions_name', 'userIDs_schema', 'userIDs_name', 'recommendations_schema', 'recommendations_name', SUMMARY)`

> → Remember
>
> The direct technique and the ANY procedure technique do not support this function.

8.1.6.1 Input/Output Parameters

Direction	Parameter	Data Type	Required	Description
IN	`model_schema`	NVARCHAR(127)	No	The name of the schema used for the table that contains the recommendation model. If null, the current schema is used.
IN	`model_name`	NVARCHAR(127)	Yes	The reference to the name of the table that contains the recommendation model used
IN	`transactions_schema`	NVARCHAR(127)	No	The name of the schema used for the table that contains the dataset. If null, the current schema is used.
IN	`transactions_name`	NVARCHAR(127)	Yes	The reference to the name of the table that contains the dataset
IN	`userIDs_schema`	NVARCHAR(127)	No	The name of the schema used for the table that contains the list of users. If null, the current schema is used.
IN	`userIDs_name`	NVARCHAR(127)	No	The reference to the name of the table that contains the list of users. If null, all users are considered. The table must have a `UserID` column.
IN	`recommendations_schema`	NVARCHAR(127)	No	The name of the schema used for the table that contains the recommendations. If null, the current shema is used.
IN	`recommendations_name`	NVARCHAR(127)	Yes	The reference to the name of the table that contains the recommendations. Will be created if it does not exist.

8.1.6.2 Input/Output Tables

Direction	Table	Type	Required	Description
IN	FUNC_HEADER [page 365]	sap.pa.apl.base::BASE.T.FUNCTION_HEADER	No	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	OPERATION_CONFIG [page 384]	sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED	Yes	The operation configuration parameters. See table below.
OUT	SUMMARY [page 388]	sap.pa.apl.base::BASE.T.SUMMARY		The summary of the results

8.1.6.3 OPERATION_CONFIG Parameters

Key	Required	Description	Default
APL/Date	No	The name of the column containing the date in the transaction table	
APL/EndDate	No	The end date of the time period considered in the transaction table. APL/Date must be specified.	
APL/FillWithBestSellers	No	A flag indicating whether a recommendation list is completed with bestsellers in order to reach the maximum number of recommendations per user. If true or 1 with APL/Top specified, the function adds bestsellers to the list.	false
APL/Item	Yes	The name of the column containing the item ID in the transaction table	

Key	Required	Description	Default
APL/Metric	No	The metrics used as score to sort the recommendations. Possible values are: • CONFIDENCE • SUPPORT • KI • LIFT • ADDED_VALUE • COSINE	CONFIDENCE
APL/SkipAlreadyOwned	No	A flag indicating whether an item which is already part of a user transaction history is removed from the recommendation list. By default, the item cannot be recommended.	true or 1
APL/StartDate	No	The start date of the time period considered in the transaction table. APL/Date must be specified.	
APL/Top	No	The maximum number of recommendations per user. Only best recommendations according to APL/metric are kept. There is no maximum if not specified.	
APL/User	Yes	The name of the column containing the user ID in the transaction table	

8.1.6.4 Example

```
-- ================================================================
-- APL_AREA, APPLY_RECOMMENDER_TO_USERS, there is no model generated
-- This script uses a recommender to generate recommendations for users
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid recommender (created by APL) in the
RECOMMENDER_CUST_TRANSACTIONS table.
--               For instance you have used apl_create_recommender_ex.sql before.
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create COLUMN table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
DROP TABLE RECO_CONFIG;
CREATE  TABLE RECO_CONFIG LIKE
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into RECO_CONFIG values ('APL/Top', '5',
NULL);                                -- optional (default: NULL)
insert into RECO_CONFIG values ('APL/SkipAlreadyOwned', 'true',
NULL);              -- optional (default: true)
insert into RECO_CONFIG values ('APL/Metric', 'CONFIDENCE',
NULL);                  -- optional (default: CONFIDENCE) (values: CONFIDENCE,
SUPPORT, KI, LIFT, COSINE, ADDED_VALUE)
insert into RECO_CONFIG values ('APL/FillWithBestSellers', 'false',
NULL);          -- optional (default: false)
```

```
insert into RECO_CONFIG values ('APL/User', 'UserID',
NULL);                          -- mandatory
insert into RECO_CONFIG values ('APL/Item', 'ItemPurchased',
NULL);                  -- mandatory
--insert into RECO_CONFIG values ('APL/Date', 'Date_PutInCaddy',
NULL);               -- optional (default: NULL)
--insert into RECO_CONFIG values ('APL/StartDate', '2000-12-20 17:32:24',
NULL);      -- optional (default: NULL)
--insert into RECO_CONFIG values ('APL/EndDate', '2001-03-26 17:32:24',
NULL);        -- optional (default: NULL)
-- The user list MUST be available in a column named "UserID"
DROP TABLE USERS_CUST;
CREATE COLUMN TABLE USERS_CUST ("UserID" Integer);
INSERT INTO USERS_CUST VALUES (728);
INSERT INTO USERS_CUST VALUES (65);
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
-- Result will be available in the table "RECOMMENDATIONS_FOR_BASKET" of the
current schema
-- The table will be created directly by the procedure if it does not already
exist
DROP TABLE RECOMMENDATIONS_FOR_USERS;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from RECO_CONFIG;

"SAP_PA_APL"."sap.pa.apl.base::APPLY_RECOMMENDER_TO_USERS"(:header,'USER_APL',
'RECOMMENDER_CUST_TRANSACTIONS', :config,'APL_SAMPLES', 'CUST_TRANSACTIONS',
'USER_APL', 'USERS_CUST', 'USER_APL', 'RECOMMENDATIONS_FOR_USERS',
out_summary);
    -- store result into table
    insert into  "USER_APL"."SUMMARY"         select * from :out_summary;
    -- show result
    -- select * from "USER_APL"."RECOMMENDATIONS_FOR_USERS";
    select * from "USER_APL"."SUMMARY";
END;
```

8.1.7 APPLY_SOCIAL_MODEL

Applies a social model on a new dataset.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, NODES1, NODES2, LINKS, COMMUNITIES, ATTRIBUTES1,
ATTRIBUTES2, OPERATION_CONFIG, DATASET, DATASET, LOG)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::APPLY_SOCIAL_MODEL"(FUNC_HEADER, MODEL,
NODES1_SCHEMA, NODES1_NAME, NODES2_SCHEMA, NODES2_NAME, LINKS_SCHEMA, LINKS_NAME,
COMMUNITIES_SCHEMA, COMMUNITIES_NAME, ATTRIBUTES1_SCHEMA, ATTRIBUTES1_NAME,
```

```
ATTRIBUTES2_SCHEMA, ATTRIBUTES2_NAME, OPERATION_CONFIG, INPUT_DATASET_SCHEMA,
INPUT_DATASET_NAME, OUTPUT_DATASET_SCHEMA, OUTPUT_DATASET_NAME, LOG)
```

The parameters NODES1_SCHEMA, NODES1_NAME, NODES2_SCHEMA, NODES2_NAME, LINKS_SCHEMA, LINKS_NAME, COMMUNITIES_SCHEMA, COMMUNITIES_NAME, ATTRIBUTES1_SCHEMA, ATTRIBUTES1_NAME, ATTRIBUTES2_SCHEMA, ATTRIBUTES2_NAME, INPUT_DATASET_SCHEMA, INPUT_DATASET_NAME, OUTPUT_DATASET_SCHEMA, and OUTPUT_DATASET_NAME are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_APPLY_SOCIAL_MODEL"(FUNC_HEADER, MODEL, NODES1, NODES2, LINKS,
COMMUNITIES, ATTRIBUTES1, ATTRIBUTES2, OPERATION_CONFIG, DATASET, DATASET, LOG)
```

Related Information

APPLY_MODEL [page 72]
GET_TABLE_TYPE_FOR_SOCIAL_APPLY [page 216]

8.1.7.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The model to be applied
IN	NODES [page 411]	No	The graph node storage (repository1). Social Models Serialization [page 400]
IN	NODES [page 411]	No	The graph node storage (repository1)
IN	LINKS	No	Graph link storage
IN	COMMUNITIES	No	Communities storage
IN	ATTRIBUTES [page 405]	No	Attributes (repository 1) storage
IN	ATTRIBUTES [page 405]	No	Attributes (repository 2) storage

Direction	Type	Required	Description
IN	OPERATION_CONFIG [page 384]	No	The configuration of the apply operation. See also the Configuration table below.
IN	DATASET [page 362]	Yes	The input dataset The input dataset must contain the following variables: • One variable for each population • The KXCOMINDEX variable, which contains communities identifiers. It should be a nominal integer.
OUT	DATASET		The output dataset The structure of the output dataset depends on the selected targets and on the selected mode values (if any). In order to get the table type of the output dataset and correctly create the output dataset for the apply operation, you can: • Infer the table type with simple rules • Use the GET_TABLE_TYPE_FOR_SOCIAL_APPLY function
OUT	LOG [page 381]		The apply log

See Social Models Serialization [page 400] for COMMUNITIES and LINKS table types.

8.1.7.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
APL/ApplyMode	Yes	See APPLY MODE for Social Modeling [page 412]	Apply mode for the model.

8.1.7.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, APPLY_SOCIAL_MODEL, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid and trained model (created by APL) in the
MODEL_SOCIAL table.
```

```
--               For instance, you have used
apl_create_social_model_and_train_links_only.sql before
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type APPLY_IN_CONTACT_EVENTS_T;
create type APPLY_IN_CONTACT_EVENTS_T as table (
    "CALLER" NVARCHAR(10),
    "KXCOMINDEX" INTEGER
);
-- KxNodes1
drop type MODEL_SOCIAL_NODES1_T;
create type MODEL_SOCIAL_NODES1_T as table (
    "node" NVARCHAR(10)
);
-- KxNodes2
-- not used here
drop type MODEL_SOCIAL_NODES2_T;
create type MODEL_SOCIAL_NODES2_T as table (
    "node" NVARCHAR(255)
);
-- KxLinks
drop type MODEL_SOCIAL_LINKS_T;
create type MODEL_SOCIAL_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" NVARCHAR(10),
    "KXNODEFIRST_2" NVARCHAR(10)
);
--KxCommunities1
drop type MODEL_SOCIAL_COMMUNITIES_T;
create type MODEL_SOCIAL_COMMUNITIES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "KXNODEFIRST" NVARCHAR(10),
    "COM_INDEX" INTEGER,
    "COM_INTRA" INTEGER,
    "COM_EXTRA" INTEGER
);
drop type MODEL_SOCIAL_ATTRIBUTES_T;
create type MODEL_SOCIAL_ATTRIBUTES_T as table (
    "DUMMY" NVARCHAR(1)
);
drop type SOCIAL_SCORE_T;
create type SOCIAL_SCORE_T as table (
    "CALLER" NVARCHAR(10),
    "kxComIndex" INT,
    "sn_graph01_cm_idx" INT,
    "sn_graph01_cm_idx_1" INT,
    "sn_graph01_cm_idx_2" INT,
    "sn_graph01_cm_idx_3" INT,
    "sn_graph01_cm_c_off" INT,
    "sn_graph01_cm_c_off_1" INT,
    "sn_graph01_cm_c_off_2" INT,
    "sn_graph01_cm_c_off_3" INT,
    "sn_graph01_cm_r_off" DOUBLE,
    "sn_graph01_cm_r_off_1" DOUBLE,
    "sn_graph01_cm_r_off_2" DOUBLE,
    "sn_graph01_cm_r_off_3" DOUBLE,
    "sn_graph01_cm_rl" NVARCHAR(5000),
    "sn_graph01_cm_sz" INT,"sn_graph01_cm_sz_1" INT,
    "sn_graph01_cm_sz_2" INT,"sn_graph01_cm_sz_3" INT,
    "sn_graph01_i_dg" INT,"sn_graph01_o_dg" INT,
    "sn_graph01_i_w_dg" DOUBLE,"sn_graph01_o_w_dg" DOUBLE,
    "sn_graph01_sg" NVARCHAR(5000),"sn_graph01_i_c_off" DOUBLE,
    "sn_graph01_o_c_off" DOUBLE,"sn_graph01_i_r_off" DOUBLE,
    "sn_graph01_o_r_off" DOUBLE,"sn_graph01_tc" INT);
```

```
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CALL_SIGNATURE;
create column table CALL_SIGNATURE   like PROCEDURE_SIGNATURE_T;
insert into CALL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',  'IN');
insert into CALL_SIGNATURE values (2, 'USER_APL','MODEL_NATIVE_T',     'IN');
insert into CALL_SIGNATURE values (3, 'USER_APL','MODEL_SOCIAL_NODES1_T',
'IN');
insert into CALL_SIGNATURE values (4, 'USER_APL','MODEL_SOCIAL_NODES2_T',
'IN');
insert into CALL_SIGNATURE values (5, 'USER_APL','MODEL_SOCIAL_LINKS_T',
'IN');
insert into CALL_SIGNATURE values (6,
'USER_APL','MODEL_SOCIAL_COMMUNITIES_T',     'IN');
insert into CALL_SIGNATURE values (7,
'USER_APL','MODEL_SOCIAL_ATTRIBUTES_T',     'IN');
insert into CALL_SIGNATURE values (8,
'USER_APL','MODEL_SOCIAL_ATTRIBUTES_T',     'IN');
insert into CALL_SIGNATURE values (9, 'USER_APL','OPERATION_CONFIG_DETAILED_T',
'IN');
insert into CALL_SIGNATURE values (10,
'USER_APL','APPLY_IN_CONTACT_EVENTS_T',          'IN');
insert into CALL_SIGNATURE values (11, 'USER_APL','SOCIAL_SCORE_T',     'OUT');
insert into CALL_SIGNATURE values (12, 'USER_APL','OPERATION_LOG_T',    'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_APPLY_SOCIAL_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA', 'APPLY_SOCIAL_MODEL',
'USER_APL', 'APLWRAPPER_APPLY_SOCIAL_MODEL', CALL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like OPERATION_CONFIG_DETAILED_T;
insert into APPLY_CONFIG values ('APL/ApplyMode', 'Default_Mode', null);
-- TODO: insert apply configuration parameters (to be defined)
drop table SCHEMA_OUT;
create table SCHEMA_OUT like TABLE_TYPE_T;
drop table APPLY_LOG;
create table APPLY_LOG like OPERATION_LOG_T;
drop table INPUT_DATA;
create table INPUT_DATA like APPLY_IN_CONTACT_EVENTS_T;
insert into INPUT_DATA values('6479988873',0);
drop view SOCIAL_MODEL_SORTED;
create view SOCIAL_MODEL_SORTED AS SELECT *  from MODEL_SOCIAL order by "ID"
asc  ;
-- select * from SOCIAL_MODEL_SORTED;
drop table APPLY_OUT;
create table APPLY_OUT like SOCIAL_SCORE_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header             = select * from FUNC_HEADER;
    config             = select * from APPLY_CONFIG;
    model              = select * from SOCIAL_MODEL_SORTED;
    model_nodes1       = select * from MODEL_SOCIAL_NODES1;
    model_nodes2       = select * from MODEL_SOCIAL_NODES2;
    model_links        = select * from MODEL_SOCIAL_LINKS;
    model_communities  = select * from MODEL_SOCIAL_COMMUNITIES;
    model_attributes1  = select * from MODEL_SOCIAL_ATTRIBUTES1;
    model_attributes2  = select * from MODEL_SOCIAL_ATTRIBUTES2;
```

```
    apply_in           = select * from INPUT_DATA;

    APLWRAPPER_APPLY_SOCIAL_MODEL(
        :header,
        :model,
        :model_nodes1,
        :model_nodes2,
        :model_links,
        :model_communities,
        :model_attributes1,
        :model_attributes2,
        :config,
        :apply_in,
        out_apply,
        out_log);
    -- store result into table
    insert into  "USER_APL"."APPLY_OUT"  select * from :out_apply;
    insert into  "USER_APL"."APPLY_LOG"  select * from :out_log;
    -- show result
    select * from "USER_APL"."APPLY_LOG";
    select * from "USER_APL"."APPLY_OUT";
END;
```

8.1.7.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, APPLY_SOCIAL_MODEL, using a native format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained model (created by APL) in the
MODEL_SOCIAL table.
--               For instance, you have used
apl_create_social_model_and_train_links_only_ex.sql before
-- ----------------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type APPLY_IN_CONTACT_EVENTS_T;
create type APPLY_IN_CONTACT_EVENTS_T as table (
    "CALLER" NVARCHAR(10),
    "KXCOMINDEX" INTEGER
);
-- This type depends on the model type and the apply settings used, use
GET_SOCIAL_TABLE_TYPE_FOR_APPLY to get its format
drop type SOCIAL_SCORE_T;
create type SOCIAL_SCORE_T as table (
    "CALLER" NVARCHAR(10),
    "kxComIndex" BIGINT,
    "sn_graph01_cm_idx" BIGINT,
    "sn_graph01_cm_idx_1" BIGINT,
    "sn_graph01_cm_idx_2" BIGINT,
    "sn_graph01_cm_idx_3" BIGINT,
    "sn_graph01_cm_c_off" BIGINT,
    "sn_graph01_cm_c_off_1" BIGINT,
    "sn_graph01_cm_c_off_2" BIGINT,
    "sn_graph01_cm_c_off_3" BIGINT,
    "sn_graph01_cm_r_off" DOUBLE,
    "sn_graph01_cm_r_off_1" DOUBLE,
    "sn_graph01_cm_r_off_2" DOUBLE,
    "sn_graph01_cm_r_off_3" DOUBLE,
    "sn_graph01_cm_rl" NVARCHAR(5000),
```

```
    "sn_graph01_cm_sz" BIGINT,
    "sn_graph01_cm_sz_1" BIGINT,
    "sn_graph01_cm_sz_2" BIGINT,
    "sn_graph01_cm_sz_3" BIGINT,
    "sn_graph01_i_dg" BIGINT,
    "sn_graph01_o_dg" BIGINT,
    "sn_graph01_i_w_dg" DOUBLE,
    "sn_graph01_o_w_dg" DOUBLE,
    "sn_graph01_sg" NVARCHAR(5000),
    "sn_graph01_i_c_off" DOUBLE,
    "sn_graph01_o_c_off" DOUBLE,
    "sn_graph01_i_r_off" DOUBLE,
    "sn_graph01_o_r_off" DOUBLE,
    "sn_graph01_tc" BIGINT
);
-- KxNodes1
drop type MODEL_SOCIAL_NODES1_T;
create type MODEL_SOCIAL_NODES1_T as table (
    "node" NVARCHAR(10)
);
-- KxNodes2
-- not used here
drop type MODEL_SOCIAL_NODES2_T;
create type MODEL_SOCIAL_NODES2_T as table (
    "node" NVARCHAR(255)
);
-- KxLinks
drop type MODEL_SOCIAL_LINKS_T;
create type MODEL_SOCIAL_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" NVARCHAR(10),
    "KXNODEFIRST_2" NVARCHAR(10)
);
--KxCommunities1
drop type MODEL_SOCIAL_COMMUNITIES_T;
create type MODEL_SOCIAL_COMMUNITIES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "KXNODEFIRST" NVARCHAR(10),
    "COM_INDEX" INTEGER,
    "COM_INTRA" INTEGER,
    "COM_EXTRA" INTEGER
);
drop type MODEL_SOCIAL_ATTRIBUTES_T;
create type MODEL_SOCIAL_ATTRIBUTES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "UserID" INTEGER,
    "generation_id" INTEGER,
    "SOURCE_NODES" NVARCHAR(255),  -- same type than ItemPurchased
    "HEAD_NODE_ID" INTEGER,
    "TAIL_NODE_ID" INTEGER
);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNCHEADER;
create table FUNCHEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNCHEADER values ('Oid', '#42');
insert into FUNCHEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into APPLY_CONFIG values ('APL/ApplyMode', 'Default_Mode', null);
-- TODO: insert apply configuration parameters (to be defined)
drop table INPUT_DATA;
create table INPUT_DATA like APPLY_IN_CONTACT_EVENTS_T;
insert into INPUT_DATA values ('6479988873', 0);
```

```
drop table APPLY_LOG;
create table APPLY_LOG like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table APPLY_OUT;
create table APPLY_OUT like SOCIAL_SCORE_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header             = select * from FUNC_HEADER;
    config             = select * from APPLY_CONFIG;
    model              = select * from MODEL_SOCIAL;
    apply_in           = select * from INPUT_DATA;

    "SAP_PA_APL"."sap.pa.apl.base::APPLY_SOCIAL_MODEL"(
        :header,
        :model,
        'USER_APL','MODEL_SOCIAL_NODES1',
        'USER_APL','MODEL_SOCIAL_NODES2',
        'USER_APL','MODEL_SOCIAL_LINKS',
        'USER_APL','MODEL_SOCIAL_COMMUNITIES',
        'USER_APL','MODEL_SOCIAL_ATTRIBUTES1',
        'USER_APL','MODEL_SOCIAL_ATTRIBUTES2',
        :config,
        'USER_APL','INPUT_DATA',
        'USER_APL','APPLY_OUT',
       out_log);
    -- store result into table
    insert into  "USER_APL"."APPLY_LOG"  select * from :out_log;
    -- show result
    select * from "USER_APL"."APPLY_LOG";
    select * from "USER_APL"."APPLY_OUT";
END;
```

8.1.8 COMPUTE_CONFUSION_MATRIX

Computes the confusion matrix and returns the confusion matrix numbers and other derived indicators for a trained, binary classification model.

> ⓘ Note
>
> This function does not support multinomial classification models and regression models based on the gradient boosting algorithm. It does support binary classification models based on the same algorithm.

PROCEDURE

> {·} Syntax
>
> `"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(FUNC_HEADER, OPERATION_CONFIG, INDICATORS, OUTPUT)`

> → Remember
>
> The direct technique and the ANY procedure technique do not support this function.

8.1.8.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER [page 365]`	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`OPERATION_CONFIG [page 384]`	No	The configuration of the operation
IN	`INDICATORS [page 370]`	Yes	The input variable statistics and indicators computed for a trained, binary classification model. This input table has been filled in with either CREATE_MODEL_AND_TRAIN [page 137], TRAIN_MODEL [page 258], GET_MODEL_INFO [page 206], or GET_VARIABLE_INDICATORS [page 222].
OUT	`INDICATORS [page 370]`		The table that contains the computed confusion matrix numbers and the derived indicators. You can use the input table as output table.

8.1.8.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Default	Description
`APL/ConfusionMatrixFromPercentPopulation`	No	In the range [0,1]		Specifies the threshold as a percentage of the population
`APL/ConfusionMatrixFromPercentTargetKey`	No	In the range [0,1]		Specifies the threshold as a percentage of the target key
`APL/ConfusionMatrixThreshold`	No		The model threshold	Specifies the score threshold

Key	Required	Supported Values	Default	Description
APL/ConfusionMatrixTarget	Yes if multiple target variables are used. No, if only one target variable is used.			Specifies the name of the target variable in the case of multiple target variables
APL/ConfusionMatrixFromMaxProfit	No	• 'true' • 'false'	'false'	Computes the threshold for maximizing profit based on the costs of true positive, true negative, false positive, and false negative
APL/TruePositiveCost	No	DOUBLE	0	Defines the cost of a true positive (correctly predicted positive observation)
APL/TrueNegativeCost	No	DOUBLE	0	Defines the cost of a true negative (correctly predicted negative observation)
APL/FalsePositiveCost	No	DOUBLE	0	Defines the cost of a false positive (actual negative observation that has been predicted positive)
APL/FalseNegativeCost	No	DOUBLE	0	Defines the cost of a false negative (actual positive observation that has been predicted negative)

8.1.8.3 List of Returned Indicators

Confusion matrix numbers with the same expressed as percentages	• True_Positive and Percent_True_Positive • True_Negative and Percent_True_Negative • False_Positive and Percent_False_Positive • False_Negative and Percent_False_Negative
Computed totals with the same expressed as percentages	• Actual_Positive and Percent_Actual_Positive • Actual_Negative and Percent_Actual_Negative • Predicted_Positive and Percent_Predicted_Positive • Predicted_Negative and Percent_Predicted _Negative

The derived indicators	• `Accuracy` • `Sensitivity` • `Specificity` • `Precision` • `F1_Score`
Cost matrix totals	• `Profit` • `Random_Profit` • `Gain`
Other indicators	• `Dataset_Size` • `Threshold`

> **Note**
>
> For more information about the definitions of these numbers, see INDICATORS [page 370].

8.1.8.4 Example

In this example you do the following:

- You create the necessary input and output tables and fill them.
- You create and train a binary classification model by calling the stored procedure `"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN"`. You get a table of indicators.
- You also get the indicators of the variables by calling the stored procedure `"SAP_PA_APL"."sap.pa.apl.base::GET_VARIABLE_INDICATORS"`.
- You compute the confusion matrix from the indicators you got previously by calling the stored procedure `"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"`.
- You request the confusion matrix numbers.

```
-- ================================================================
-- COMPUTE_CONFUSION_MATRIX, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid trained model with target 'class' (created by
APL procedure call) an in the MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train_ex.sql)
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table OPERATION_CONFIG;
create table OPERATION_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into OPERATION_CONFIG values ('APL/IndicatorDataset', 'Validation',null);
-- Ask to export Indicator from  dataset 'Validation'
```

```
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table PROFITCURVES;
create table PROFITCURVES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.PROFITCURVE";
-- ------------------------------------------------------------------------
-- Extract Indicators from trained model
-- ------------------------------------------------------------------------
DO BEGIN
    declare config_matrix
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
    header    = select * from FUNC_HEADER;
    model_in  = select * from MODEL_TRAIN_BIN;
    config    = select * from OPERATION_CONFIG;
    "SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_INFO"(:header, :model_in, :config,
out_summary, out_variable_roles, out_variable_desc, out_indicators,
out_profitcurves);

    -- store result into table
    insert into  "USER_APL"."INDICATORS" select * from :out_indicators;
    -- show result
    select * from "USER_APL"."INDICATORS";

    -- test default config

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;
    -- test with 0% Population
    :config_matrix.insert(('APL/ConfusionMatrixFromPercentPopulation',
'0',null));

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;

    -- test with 50% Population
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixFromPercentPopulation',
'0.5',null));

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;

    -- test with 100% Population
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixFromPercentPopulation',
'1',null));

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;
    -- test with 0% Target Key  Population
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixFromPercentTargetKey', '0',null));

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;
```

```
    -- test with 50% Target Key Population
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixFromPercentTargetKey',
'0.5',null));

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;
    -- test with 100% Target Key Population
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixFromPercentTargetKey', '1',null));

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;
    -- test with threshold 0.05
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixThreshold', '0.05',null));

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;
    -- test with threshold 0.10
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixThreshold', '0.10',null));

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;
    -- test with target class
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixTarget', 'class',null));

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;
    -- test with over max  threshold
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixThreshold', '2',null)); -- over
max  threshold

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;
    -- test with  below min  threshold
    :config_matrix.delete();
    :config_matrix.insert(('APL/ConfusionMatrixThreshold', '-1',null)); -- below
min  threshold

"SAP_PA_APL"."sap.pa.apl.base::COMPUTE_CONFUSION_MATRIX"(:header,:config_matrix,:
out_indicators, out_result);
    select * from :out_result;


END;
```

8.1.9 CREATE_MODEL

Creates a new analysis model and lets the engine guess the variable descriptions of the training dataset.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, DATASET, MODEL, VARIABLE_DESC)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL"(FUNC_HEADER, OPERATION_CONFIG,
DATASET_SCHEMA, DATASET_NAME, MODEL, VARIABLE_DESC)
```

The parameters `DATASET_SCHEMA` and `DATASET_NAME` are strings.

`VARIABLE_DESC` is optional. If you use the direct or stored procedure technique, simply remove it from the signature.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_CREATE_MODEL"(FUNC_HEADER, OPERATION_CONFIG, DATASET, MODEL,
VARIABLE_DESC)
```

Syntax

```
"_SYS_AFL"."APL_CREATE_MODEL__OVERLOAD_3_1"(FUNC_HEADER, OPERATION_CONFIG, DATASET,
MODEL)
```

8.1.9.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`OPERATION_CONFIG` [page 384]	Yes	The type of Automated Analytics model

Direction	Type	Required	Description
IN	`DATASET` [page 362]	Yes, if variable descriptions are not provided. The variable descriptions will be guessed. No, if the variable descriptions are provided. (The table must be provided but it can be empty.)	The training dataset, which includes a column for the target variable. These column values are the answer to the business question for your existing known data and must not have any empty cells.
OUT	`MODEL` [page 382]		The created model
OUT	`VARIABLE_DESC` [page 396]		The guessed variable descriptions of the created model

8.1.9.2 OPERATION_CONFIG Parameters

In addition, all Common APL Aliases for Model Training [page 336] are also supported.

Key	Required	Supported Values	Description
`APL/ModelType`	Yes	• `'regression/classification'` (legacy) • `'regression'` (gradient boosting) • `'binary classification'` (gradient boosting) • `'multiclass'` for multinomial classification models • `'clustering'` • `'timeseries'` • `'statbuilder'`	The type of Automated Analytics model. See Model Types [page 13].
`APL/UseCustomCuttingStrategy`	No	• `'true'` • `'false'`	The created model will use 3 input datasets (training, validation, and testing) for the training phase

8.1.9.3 Example: DIRECT

```
-- ============================================================
-- APL_AREA, CREATE_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
```

```
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CREATE_MODEL_SIGNATURE;
create column table CREATE_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into CREATE_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into CREATE_MODEL_SIGNATURE values (2, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into CREATE_MODEL_SIGNATURE values (3, 'USER_APL','ADULT01_T', 'IN');
insert into CREATE_MODEL_SIGNATURE values (4, 'USER_APL','MODEL_BIN_OID_T',
'OUT');
insert into CREATE_MODEL_SIGNATURE values (5, 'USER_APL','VARIABLE_DESC_OID_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_CREATE_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_MODEL','USER_APL',
'APLWRAPPER_CREATE_MODEL', CREATE_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_CONFIG;
create table CREATE_CONFIG like OPERATION_CONFIG_T;
insert into CREATE_CONFIG values ('APL/ModelType', 'regression/classification');
insert into CREATE_CONFIG values ('APL/UseCustomCuttingStrategy', 'true');
drop table MODEL_BIN;
create table MODEL_BIN like MODEL_BIN_OID_T;
drop table VARIABLE_DESC_OUT;
create table VARIABLE_DESC_OUT like VARIABLE_DESC_OID_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_CONFIG;
    dataset  = select * from APL_SAMPLES.ADULT01;
```

```
    APLWRAPPER_CREATE_MODEL(:header, :config,:dataset,out_model,out_var_desc);
    -- store result into table
    insert into  "USER_APL"."MODEL_BIN"          select * from :out_model;
    insert into  "USER_APL"."VARIABLE_DESC_OUT"  select * from :out_var_desc;
    -- show result
    select * from "USER_APL"."MODEL_BIN";
    select * from "USER_APL"."VARIABLE_DESC_OUT";
END;
```

8.1.9.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, CREATE_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin_ex.sql).
connect USER_APL password Password1;
-- ----------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ----------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#69');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_CONFIG;
create table CREATE_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into CREATE_CONFIG values ('APL/ModelType', 'regression/classification',
null);
-- enable the custom cutting strategy
insert into CREATE_CONFIG values ('APL/UseCustomCuttingStrategy', 'true', null);
drop table MODEL_BIN;
create table MODEL_BIN like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table VARIABLE_DESC_OUT;
create table VARIABLE_DESC_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- ----------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ----------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_CONFIG;

    "SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL"(:header, :config,
'APL_SAMPLES','ADULT01',out_model,out_var_desc);
    -- store result into table
    insert into  "USER_APL"."MODEL_BIN"          select * from :out_model;
    insert into  "USER_APL"."VARIABLE_DESC_OUT"  select * from :out_var_desc;
    -- show result
    select * from "USER_APL"."MODEL_BIN";
    select * from "USER_APL"."VARIABLE_DESC_OUT";
END;
```

8.1.10 CREATE_MODEL_AND_TRAIN

Creates a model and trains it on a known dataset.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET, MODEL, LOG, SUMMARY, INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET_SCHEMA, DATASET_NAME, MODEL, LOG, SUMMARY, INDICATORS)
```

The parameters `DATASET_SCHEMA` and `DATASET_NAME` are strings.

You can have `MODEL` as a single OUT parameter. If you use the direct or stored procedure technique, simply remove `LOG`, `SUMMARY` and `INDICATORS` from the signature.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_CREATE_MODEL_AND_TRAIN"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET, MODEL, LOG, SUMMARY, INDICATORS)
```

Syntax

```
"_SYS_AFL"."APL_CREATE_MODEL_AND_TRAIN__OVERLOAD_5_1"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET, MODEL)
```

8.1.10.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.

Direction	Type	Required	Description
IN	`OPERATION_CONFIG` [page 384]	Yes	The configuration of the training operation. In addition, all Common APL Aliases for Model Training [page 336] are also supported, depending on the type of model.
IN	`VARIABLE_DESC` [page 396]	No (The table must be provided but it can be empty.)	The description of the variables used in the training data source. If no variable description is provided, then the variable descriptions are automatically guessed by the AutomatedAnalytics engine.
IN	`VARIABLE_ROLES` [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training.
IN	`DATASET` [page 362]	Yes	The name of your training dataset
OUT	`MODEL` [page 382]		The created and trained model
OUT	`LOG` [page 381]		The training log produced by the AutomatedAnalytics engine
OUT	`SUMMARY` [page 388]		The training summary
OUT	`INDICATORS` [page 370]		The variable statistics and indicators. You can use the `INDICATORS_DATASET_T` table type to debrief the model from a testing partition dataset. This is supported by the direct technique and ANY procedure only.

8.1.10.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
`APL/ModelType`	Yes	• `'regression/classification'` (legacy) • `'regression'` (gradient boosting) • `'binary classification'` (gradient boosting) • `'multiclass'` for multinomial classification models • `'clustering'` • `'timeseries'` • `'statbuilder'`	The type of Automated Analytics model. See Model Types [page 13].
`APL/VariableAutoSelection`	No	• `'true'` • `'false'` (default value)	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.
`APL/ExcludeLeakVariables`	No	• `true` • `false`	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.
`APL/SegmentColumnName`	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

8.1.10.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, CREATE_MODEL_AND_TRAIN, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
```

```
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CREATE_MODEL_AND_TRAIN_SIGNATURE;
create column table CREATE_MODEL_AND_TRAIN_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (4,
'USER_APL','VARIABLE_ROLES_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (5, 'USER_APL','ADULT01_T',
'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (6,
'USER_APL','MODEL_BIN_OID_T', 'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (7,
'USER_APL','OPERATION_LOG_T', 'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (8, 'USER_APL','SUMMARY_T',
'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (9,
'USER_APL','INDICATORS_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_CREATE_MODEL_AND_TRAIN'
);
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_MODEL_AND_TRAIN','USER_AP
L', 'APLWRAPPER_CREATE_MODEL_AND_TRAIN', CREATE_MODEL_AND_TRAIN_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_AND_TRAIN_CONFIG;
create table CREATE_AND_TRAIN_CONFIG like OPERATION_CONFIG_T;
insert into CREATE_AND_TRAIN_CONFIG values ('APL/ModelType', 'clustering');
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to use guess variables
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
insert into VARIABLE_ROLES values ('class', 'skip');
insert into VARIABLE_ROLES values ('fnlwgt', 'target');
drop table CLUST_TRAIN_BIN;
create table CLUST_TRAIN_BIN like MODEL_BIN_OID_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
```

```
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_AND_TRAIN_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;

APLWRAPPER_CREATE_MODEL_AND_TRAIN(:header, :config, :var_desc,:var_role, :dataset
,out_model,out_log,out_sum,out_indic);

    -- store result into table
    insert into  "USER_APL"."CLUST_TRAIN_BIN" select * from :out_model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :out_indic;
    -- show result
    select * from "USER_APL"."CLUST_TRAIN_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    SELECT *,
    SUBSTR_REGEXPR('[^\x1F]+' IN "DETAIL" OCCURRENCE 1) as "ClusterId",
    SUBSTR_REGEXPR('[^\x1F]+' IN "DETAIL" OCCURRENCE 2) as "Categorie"
    FROM "USER_APL"."INDICATORS"
    where "KEY" = 'ClusterCategoryFrequency';
END;
```

8.1.10.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, CREATE_MODEL_AND_TRAIN, using a binary format for the model
-- This script creates a model, guesses its description and trains it
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_AND_TRAIN_CONFIG;
create table CREATE_AND_TRAIN_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into CREATE_AND_TRAIN_CONFIG values ('APL/ModelType', 'regression/
classification',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/VariableAutoSelection',
'true',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionBestIteration', 'true',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMaxNbOfFinalVariables', '5',null);
```

```
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMinNbOfFinalVariables', '1',null);
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to use guess variables
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
-- variable roles are optional, hence the empty table
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_AND_TRAIN_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;

"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN"(:header, :config, :var_des
c,:var_role, 'APL_SAMPLES','ADULT01', model,  train_log, train_sum,
train_indic);
    insert into  "USER_APL"."MODEL_TRAIN_BIN" select * from :model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :train_log;
    insert into  "USER_APL"."SUMMARY"         select * from :train_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :train_indic;
    select * from "USER_APL"."MODEL_TRAIN_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.11 CREATE_MODEL_AND_TRAIN (3 Datasets)

Creates a model and trains it on three datasets (training, validation, and testing).

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES,
TRAINING_DATASET, VALIDATION_DATASET, TESTING_DATASET, MODEL, LOG, SUMMARY, INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN_MULTI_DATASET"(FUNC_HEADER,
OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, TRAINING_DATASET_SCHEMA,
```

```
TRAINING_DATASET_NAME, VALIDATION_DATASET_SCHEMA, VALIDATION_DATASET_NAME,
TESTING_DATASET_SCHEMA, TESTING_DATASET_NAME, MODEL, LOG, SUMMARY, INDICATORS)
```

The parameters `TRAINING_DATASET_SCHEMA`, `TRAINING_DATASET_NAME`, `VALIDATION_DATASET_SCHEMA`, `VALIDATION_DATASET_NAME`, `TESTING_DATASET_SCHEMA`, and `TESTING_DATASET_NAME` are strings.

You can have `MODEL` as a single OUT parameter. If you use the direct or stored procedure technique, simply remove `LOG`, `SUMMARY` and `INDICATORS` from the signature.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_CREATE_MODEL_AND_TRAIN__OVERLOAD_7_4"(FUNC_HEADER,
OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, TRAINING_DATASET, VALIDATION_DATASET,
TESTING_DATASET, MODEL, LOG, SUMMARY, INDICATORS)
```

Syntax

```
"_SYS_AFL"."APL_CREATE_MODEL_AND_TRAIN__OVERLOAD_7_1"(FUNC_HEADER,
OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, TRAINING_DATASET, VALIDATION_DATASET,
TESTING_DATASET, MODEL)
```

8.1.11.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`OPERATION_CONFIG` [page 384]	Yes	The configuration of the training operation. In addition, all Common APL Aliases for Model Training [page 336] are also supported.
IN	`VARIABLE_DESC` [page 396]	No (The table must be provided but it can be empty.)	The description of the variables used in the training data source. If no variable description is provided, then the variable descriptions are automatically guessed by the AutomatedAnalytics engine.
IN	`VARIABLE_ROLES` [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training.

Direction	Type	Required	Description
IN	`TRAINING_DATASET` [page 362]	Yes	The name of your training dataset
IN	`VALIDATION_DATASET` [page 362]	Yes	The name of your validation dataset
IN	`TESTING_DATASET` [page 362]	No (The table must be provided but it can be empty.)	The name of your testing dataset
OUT	`MODEL` [page 382]		The created and trained model
OUT	`LOG` [page 381]		The training log produced by the AutomatedAnalytics engine
OUT	`SUMMARY` [page 388]		The training summary
OUT	`INDICATORS` [page 370]		The variable statistics and indicators

8.1.11.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
`APL/ModelType`	Yes	• `'regression/classification'` (legacy) • `'regression'` (gradient boosting) • `'binary classification'` (gradient boosting) • `'multiclass'` for multinomial classification models • `'clustering'` • `'timeseries'` • `'statbuilder'`	The type of Automated Analytics model. See Model Types [page 13].
`APL/VariableAutoSelection`	No	• `'true'` • `'false'` (default value)	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.

Key	Required	Supported Values	Description
`APL/ExcludeLeakVariables`	No	• `true` • `false`	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.

8.1.11.3 Example: DIRECT

```
-- ====================================================================
-- APL_AREA, CREATE_MODEL_AND_TRAIN, using a binary format for the model
-- This script creates a model, guesses its description and trains it
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CREATE_MODEL_AND_TRAIN_SIGNATURE;
create column table CREATE_MODEL_AND_TRAIN_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_T', 'IN');
```

```
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (4,
'USER_APL','VARIABLE_ROLES_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (5, 'USER_APL','ADULT01_T',
'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (6, 'USER_APL','ADULT01_T',
'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (7, 'USER_APL','ADULT01_T',
'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (8,
'USER_APL','MODEL_BIN_OID_T', 'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (9,
'USER_APL','OPERATION_LOG_T', 'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (10, 'USER_APL','SUMMARY_T',
'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (11,
'USER_APL','INDICATORS_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_CREATE_MODEL_AND_TRAIN'
);
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_MODEL_AND_TRAIN','USER_AP
L', 'APLWRAPPER_CREATE_MODEL_AND_TRAIN', CREATE_MODEL_AND_TRAIN_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_AND_TRAIN_CONFIG;
create table CREATE_AND_TRAIN_CONFIG like OPERATION_CONFIG_T;
insert into CREATE_AND_TRAIN_CONFIG values ('APL/ModelType', 'regression/
classification');
insert into CREATE_AND_TRAIN_CONFIG values ('APL/VariableAutoSelection', 'true');
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionBestIteration', 'true');
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMaxNbOfFinalVariables', '5');
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMinNbOfFinalVariables', '1');
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to use guess variables
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- variable roles are optional, hence the empty table
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like MODEL_BIN_OID_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
drop view  TEST;
create view  TEST as (select * from APL_SAMPLES.ADULT01);
drop view ESTIMATION;
create view ESTIMATION as (select * from APL_SAMPLES.ADULT01);
drop view  VALIDATION;
create view VALIDATION as (select * from APL_SAMPLES.ADULT01);
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_AND_TRAIN_CONFIG;
```

```
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    estimation  = select * from ESTIMATION;
    validation  = select * from VALIDATION;
    test  = select * from TEST;

APLWRAPPER_CREATE_MODEL_AND_TRAIN(:header, :config, :var_desc,:var_role, :estimat
ion,:validation,:test,out_model,out_log,out_sum,out_indic);
    insert into  "USER_APL"."MODEL_TRAIN_BIN" select * from :out_model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"    select * from :out_indic;
    select * from "USER_APL"."MODEL_TRAIN_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.11.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, CREATE_MODEL_AND_TRAIN, using a binary format for the model
-- This script creates a model, guesses its description and trains it
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_AND_TRAIN_CONFIG;
create table CREATE_AND_TRAIN_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into CREATE_AND_TRAIN_CONFIG values ('APL/ModelType', 'regression/
classification',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/VariableAutoSelection',
'true',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionBestIteration', 'true',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMaxNbOfFinalVariables', '5',null);
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMinNbOfFinalVariables', '1',null);
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to use guess variables
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
-- variable roles are optional, hence the empty table
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
```

```
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop view  TEST;
create view  TEST as (select * from APL_SAMPLES.ADULT01);
drop view ESTIMATION;
create view ESTIMATION as (select * from APL_SAMPLES.ADULT01);
drop view  VALIDATION;
create view VALIDATION as (select * from APL_SAMPLES.ADULT01);
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_AND_TRAIN_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;

"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN_MULTI_DATASET"(:header, :co
nfig, :var_desc,:var_role, 'USER_APL','ESTIMATION', 'USER_APL','VALIDATION',
'USER_APL','TEST',out_model,out_log,out_sum,out_indic);
    insert into  "USER_APL"."MODEL_TRAIN_BIN" select * from :out_model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"    select * from :out_indic;
    select * from "USER_APL"."MODEL_TRAIN_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.12 CREATE_MODEL_AND_TRAIN (with Debrief Tables)

Creates a model and trains it on a known dataset. This function outputs a model and its debrief tables in one call.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET,
MODEL, LOG, SUMMARY, DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN_DEBRIEF"(FUNC_HEADER,
OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET_SCHEMA, DATASET_NAME, MODEL,
LOG, SUMMARY, DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

The parameters `DATASET_SCHEMA` and `DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_CREATE_MODEL_AND_TRAIN__OVERLOAD_5_5"(FUNC_HEADER,
OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET, MODEL, LOG, SUMMARY,
DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

8.1.12.1 Input/Output Tables

The following table details the IN and OUT tables for the CREATE_MODEL_AND_TRAIN (with Debrief tables) function.

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	OPERATION_CONFIG [page 384]	Yes	The configuration of the training operation. In addition, all Common APL Aliases for Model Training [page 336] are also supported, depending on the type of model.
IN	VARIABLE_DESC [page 396]	No (The table must be provided but it can be empty.)	The description of the variables used in the training data source. If no variable description is provided, then the variable descriptions are automatically guessed by the AutomatedAnalytics engine.
IN	VARIABLE_ROLES [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training.
IN	DATASET [page 362]	Yes	The name of your training dataset
OUT	MODEL [page 382]		The created and trained model
OUT	LOG [page 381]		The training log produced by the AutomatedAnalytics engine
OUT	SUMMARY [page 388]		The training summary
OUT	DEBRIEF_METRIC [page 364]	Yes	The debrief metrics
OUT	DEBRIEF_PROPERTY [page 365]	Yes	The debrief properties

8.1.12.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
`APL/ModelType`	Yes	• `'regression/classification'` (legacy) • `'regression'` (gradient boosting) • `'binary classification'` (gradient boosting) • `'multiclass'` for multinomial classification models • `'clustering'` • `'timeseries'` • `'statbuilder'`	The type of Automated Analytics model. See Model Types [page 13].
`APL/VariableAutoSelection`	No	• `'true'` • `'false'` (default value)	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.
`APL/ExcludeLeakVariables`	No	• `'true'` • `'false'` (default value)	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.
`APL/SegmentColumnName`	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

8.1.12.3 Example: DIRECT

The following code shows an example of the Direct technique for the CREATE_MODEL_AND_TRAIN (with Debrief tables) function.

```
-- ------------------------------------------------------------------
-- APL_AREA, CREATE_MODEL_AND_TRAIN, using a binary format for the model
-- This script creates a model, guesses its description and trains it
--
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
```

```
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CREATE_MODEL_AND_TRAIN_SIGNATURE;
create column table CREATE_MODEL_AND_TRAIN_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (4,
'USER_APL','VARIABLE_ROLES_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (5, 'USER_APL','ADULT01_T',
'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (6,
'USER_APL','MODEL_BIN_OID_T', 'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (7,
'USER_APL','OPERATION_LOG_T', 'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (8, 'USER_APL','SUMMARY_T',
'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (9,
'USER_APL','DEBRIEF_METRIC_OID_T', 'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (10,
'USER_APL','DEBRIEF_PROPERTY_OID_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_CREATE_MODEL_AND_TRAIN'
);
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_MODEL_AND_TRAIN','USER_AP
L', 'APLWRAPPER_CREATE_MODEL_AND_TRAIN', CREATE_MODEL_AND_TRAIN_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table CREATE_AND_TRAIN_CONFIG;
create table CREATE_AND_TRAIN_CONFIG like OPERATION_CONFIG_T;
insert into CREATE_AND_TRAIN_CONFIG values ('APL/ModelType', 'regression/
classification');
insert into CREATE_AND_TRAIN_CONFIG values ('APL/VariableAutoSelection', 'true');
```

```
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionBestIteration', 'true');
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMaxNbOfFinalVariables', '5');
insert into CREATE_AND_TRAIN_CONFIG values ('APL/
VariableSelectionMinNbOfFinalVariables', '1');
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to use guess variables
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- variable roles are optional, hence the empty table
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like MODEL_BIN_OID_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table DEBRIEF_METRIC;
create table DEBRIEF_METRIC like DEBRIEF_METRIC_OID_T;
drop table DEBRIEF_PROPERTY;
create table DEBRIEF_PROPERTY like DEBRIEF_PROPERTY_OID_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_AND_TRAIN_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;

APLWRAPPER_CREATE_MODEL_AND_TRAIN(:header, :config, :var_desc,:var_role, :dataset
, out_model,  out_log, out_sum, out_debrief_metric,
out_debrief_property);
    insert into  "USER_APL"."MODEL_TRAIN_BIN" select * from :out_model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."DEBRIEF_METRIC"   select * from :out_debrief_metric;
    insert into  "USER_APL"."DEBRIEF_PROPERTY" select *
from :out_debrief_property;
    select * from "USER_APL"."MODEL_TRAIN_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."DEBRIEF_METRIC";
    select * from "USER_APL"."DEBRIEF_PROPERTY";
END;
```

8.1.13 CREATE_MODEL_AND_TRAIN_APPLY

This function creates a model, trains the model on a known dataset, and applies the trained model on a new dataset in one call. In the same call, this function also outputs a model and the model's debrief tables.

Direct

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER,OPERATION_CONFIG,VARIABLE_DESCS,VARIABLE_ROLES,DATASET,DATASET,DATASET,DATASET,LOG,SUMMARY,DEBRIEF_METRIC,DEBRIEF_PROPERTY)
```

Procedure

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN_APPLY"(FUNC_HEADER,OPERATION_CONFIG,VARIABLE_DESCS,VARIABLE_ROLES,TRAINING_DATASET_SCHEMA,TRAINING_DATASET_NAME,APPLYIN_DATASET_SCHEMA,APPLYIN_DATASET_NAME,OUTPUT_DATASET_SCHEMA,OUTPUT_DATASET_NAME,MODEL,LOG,SUMMARY,DEBRIEF_METRIC,DEBRIEF_PROPERTY)
```

The parameters `TRAINING_DATASET_SCHEMA`,`TRAINING_DATASET_NAME`,`APPLYIN_DATASET_SCHEMA`,`APPLYIN_DATASET_NAME`,`OUTPUT_DATASET_SCHEMA`, and `OUTPUT_DATASET_NAME` are strings.

Any Procedure

Syntax

```
"_SYS_AFL"."APL_CREATE_MODEL_AND_TRAIN_APPLY"(FUNC_HEADER,OPERATION_CONFIG,VARIABLE_DESCS,VARIABLE_ROLES,TRAINING_DATASET,APPLYIN_DATASET,OUTPUT_DATASET,MODEL,LOG,SUMMARY,DEBRIEF_METRIC,DEBRIEF_PROPERTY)
```

8.1.13.1 Input/Output Tables

The following table details the IN and OUT tables for the CREATE_MODEL_AND_TRAIN_APPLY function.

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	OPERATION_CONFIG [page 384]	Yes	The configuration of the training operation. In addition, all Common APL Aliases for Model Training [page 336] are also supported, depending on the type of model.
IN	VARIABLE_DESC [page 396]	No (The table must be provided but it can be empty.)	The description of the variables used in the training data source. If no variable description is provided, then the variable descriptions are automatically guessed by the AutomatedAnalytics engine.
IN	VARIABLE_ROLES [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training.
IN	DATASET [page 362]	Yes	The name of your training dataset.
IN	DATASET [page 362]	Yes	The input dataset for apply.
OUT	DATASET [page 362]	Yes	The output dataset. This “ApplyOut” dataset needs to have the correct table structure. For more information, see GET_TABLE_TYPE_FOR_APPLY [page 213].
OUT	MODEL [page 382]		The created and trained model.
OUT	LOG [page 381]		The training log produced by the AutomatedAnalytics engine.
OUT	SUMMARY [page 388]		The training summary.
OUT	DEBRIEF_METRIC [page 364]	Yes	The debrief metrics.
OUT	DEBRIEF_PROPERTY [page 365]	Yes	The debrief properties.

8.1.13.2 OPERATION_CONFIG Parameters

The `CREATE_MODEL_AND _TRAIN_APPLY` function chains together the train and apply parameters.

Train Parameters

The following table details the supported train parameters, which are the same as the OPERATION_CONFIG Parameters [page 150] for the CREATE_MODEL_AND_TRAIN (with Debrief Tables) function.

Key	Required	Supported Values	Description
`APL/ModelType`	Yes	• `'regression/classification'` (legacy) • `'regression'` (gradient boosting) • `'binary classification'` (gradient boosting) • `'multiclass'` for multinomial classification models • `'clustering'` • `'timeseries'` • `'statbuilder'`	The type of Automated Analytics model. See Model Types [page 13].
`APL/VariableAutoSelection`	No	• `'true'` • `'false'` (default value)	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.
`APL/ExcludeLeakVariables`	No	• `true` • `false`	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.
`APL/SegmentColumnName`	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

Apply Parameters

The CREATE_MODEL_AND_TRAIN_APPLY function supports all OPERATION_CONFIG parameters for the APPLY_MODEL. For a list of those parameters, and for additional supported parameters from the `APPLY_MODEL` function, see the following topics:

- OPERATION_CONFIG Parameters [page 75]
- APL/ApplyExtraMode [page 85]
- Advanced Apply Settings [page 89]
- APL/ApplyExtraMode with Segmented Modeling [page 93]

8.1.13.3 Example: PROCEDURE

The following code shows an example of the Procedure technique for the CREATE_MODEL_AND_TRAIN_APPLY.

```
-- @required(hanaMinimumVersion,2.0.32)
-- ================================================================
connect USER_APL
password Password1;
-- Uncomment to modify the cache location to schema APL_CACHE

-- SET SESSION 'APL_CACHE_SCHEMA' = 'APL_CACHE';
-- input dataset for training
DROP
VIEW TRAINING;
CREATE
VIEW TRAINING
AS
SELECT *
FROM
"APL_SAMPLES"."ADULT01"
LIMIT 10000;
-- input dataset for apply
DROP
VIEW APPLYIN;
CREATE
VIEW APPLYIN
AS
SELECT *
FROM
"APL_SAMPLES"."ADULT01"
LIMIT 20000 OFFSET 10000;
DROP
TABLE ADULT01_APPLY;
CREATE
TABLE ADULT01_APPLY ("KxIndex"
INTEGER,"class"
INTEGER,"rr_class"
DOUBLE);
DO
BEGIN

   DECLARE
 header   "SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";

   DECLARE
 config "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
```

```
   DECLARE
 var_desc "SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
   DECLARE
 var_role
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";

   :header.insert(('Oid',
'#42'));
   -- activate checking of all parameters
   :header.insert(('CheckOperationConfig',
'true'));
-- setup the type of model
:config.insert(('APL/ModelType',
'regression/classification',null));

TRUNCATE
TABLE
"ADULT01_APPLY";
   "SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN_APPLY"(:header,
 :config, :var_desc,:var_role, 'USER_APL','TRAINING',
'USER_APL','APPLYIN',
'USER_APL','ADULT01_APPLY',
 out_model, out_log, out_sum, out_debrief_metric, out_debrief_property);

   -- display all outputs
   select
 * from :out_model;
   select
 * from :out_log;
   select
 * from :out_sum;
   select
 * from :out_debrief_metric;
   select
 * from :out_debrief_property;
   select
 * from
"ADULT01_APPLY";
END;
```

8.1.14 CREATE_RECO_MODEL_AND_TRAIN

Creates a recommendation model and trains it on a dataset.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, DATASET, MODEL, NODES1,
NODES2, LINKS, LOG, SUMMARY, INDICATORS, RECO_CODE)
```

In the direct technique, this function can run without `RECO_CODE` as output.

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::CREATE_RECO_MODEL_AND_TRAIN"(FUNC_HEADER,
OPERATION_CONFIG, VARIABLE_DESC, DATASET_SCHEMA, DATASET_NAME, MODEL, NODES1_SCHEMA,
```

```
NODES1_NAME, NODES2_SCHEMA, NODES2_NAME, LINKS_SCHEMA, LINKS_NAME, LOG, SUMMARY,
INDICATORS, RECO_CODE)
```

The parameters DATASET_SCHEMA, DATASET_NAME, NODES1_SCHEMA, NODES1_NAME, NODES2_SCHEMA, NODES2_NAME, LINKS_SCHEMA, and LINKS_NAME are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_CREATE_RECO_MODEL_AND_TRAIN"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESC, DATASET, MODEL, NODES1, NODES2, LINKS, LOG, SUMMARY, INDICATORS)
```

Syntax

```
"_SYS_AFL"."APL_CREATE_RECO_MODEL_AND_TRAIN__OVERLOAD_4_8"(FUNC_HEADER,
OPERATION_CONFIG, VARIABLE_DESC, DATASET, MODEL, NODES1, NODES2, LINKS, LOG, SUMMARY,
INDICATORS, RECO_CODE)
```

Note

This function is similar to the CREATE_MODEL_AND_TRAIN function, but this function has additional tables for model persistence. You can also use this function to create a recommendation model that returns similar objects. See Using the Function to Obtain Recommendations for Similar Items [page 165].

8.1.14.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	OPERATION_CONFIG [page 384]	Yes	The configuration of the training operation
IN	VARIABLE_DESC [page 396]	No (The table must be provided but it can be empty.)	The description of the variables used in the training data source. If no variable description is provided (i.e. empty table), then the variable descriptions are automatically guessed by the Automated Analytics engine.
IN	DATASET [page 362]	Yes	The name of your training dataset

Direction	Type	Required	Description
OUT	MODEL [page 382]		The new model model. Composite model storage
OUT	NODES [page 411]		Graph node storage (repository 1)
OUT	NODES [page 411]		Graph node storage (repository 2)
OUT	LINKS [page 410]		Graph link storage
OUT	LOG [page 381]		The training log produced by the Automated Analytics engine.
OUT	SUMMARY [page 388]		The training summary.
OUT	INDICATORS [page 370]		The variable statistics and indicators.
OUT	RECO_CODE		The generated code for items recommendation In addition all the aliases mentioned below are supported.

8.1.14.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Default	Description
APL/BestSeller	No		50000	Bestseller threshold
APL/CodeKey	No			Key name. This configuration key is not compatible with the APL/RecommendationType 'similarity' mode
APL/CodeSpace	No			Table name. This configuration key is not compatible with the APL/RecommendationType 'similarity' mode
APL/CodeType	Yes in 'Standard' recommendation type	'SQL'		Type of code to export. This configuration key is not compatible with the APL/RecommendationType 'similarity' mode.
APL/Date	No			Transaction date
APL/EndDate	No			End date filter
APL/IncludeBestSellers	No		false	Include de-facto best seller items into the suggestion list. This configuration key is not compatible with the APL/RecommendationType 'similarity' mode
APL/Item	Yes			Item/product entity: bipartite destination node

Key	Required	Supported Values	Default	Description
APL/MaxTopNodes	No		100000	Max top nodes
APL/MinimumConfidence	No		0.05	Minimum confidence to consider the association rule
APL/MinimumPredictivePower	No		disable	Minimum KI to consider the association rule
APL/MinimumSupport	No		2	Minimum support to consider the association rule
APL/RecoLinksTable	Yes		KXLINKS1	Graph links storage table name. The provided value should match the `links` parameter. This configuration key is not compatible with the standard APL/RecommendationType 'standard' mode usage.
APL/RecommendationType	No	• 'standard' • 'similarity'		Specifies the recommendation type.
APL/RecommendProcName	No		RECOMMEND_SIMILAR_ITEM	Used only in the 'similarity' code type context, this configuration key allows specifying the name of the generated store procedure
APL/RuleWeight	No	• Support • Independence Probability	Independence Probability	Rules graph edge weight
APL/SkipAlreadyOwned	No		true	Exclude items already owned from the suggestion list. This configuration key is not compatible with the APL/RecommendationType 'similarity' mode
APL/StartDate	No			Start date filter
APL/Top	No		max	Keep top N of recommended items ranked by confidence. This configuration key is not compatible with the APL/RecommendationType 'similarity' mode
APL/User	Yes			User entity : bipartite source node

Key	Required	Supported Values	Default	Description
APL/Weight	No			Graph edge weight

8.1.14.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, CREATE_RECO_MODEL_AND_TRAIN, using a native format for the model
-- This script creates a model, guesses its description and trains it
--
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type CUST_TRANSACTIONS_T;
create type CUST_TRANSACTIONS_T as table (
    "UserID" INTEGER,
    "ItemPurchased" NVARCHAR(18),
    "Date_PutInCaddy" DATETIME,
    "Quantity" INTEGER,
    "TransactionID" INTEGER
);
-- KxNodes1
drop type MODEL_RECO_NODES1_T;
create type MODEL_RECO_NODES1_T as table (
    "node" INT -- must be of the same SQL type as the User column (UserID here)
);
-- KxNodes2
drop type MODEL_RECO_NODES2_T;
create type MODEL_RECO_NODES2_T as table (
    "node" NVARCHAR(255) -- must be of the same SQL type as the Item column
(ItemPurchased here)
);
-- KxLinks
drop type MODEL_RECO_LINKS_T;
create type MODEL_RECO_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" INT, -- must be of the same SQL type as the User column
(UserID here)
    "KXNODESECOND" NVARCHAR(255), -- must be of the same SQL type as the Item
column (ItemPurchased here)
    "KXNODESECOND_2" NVARCHAR(255) -- must be of the same SQL type as the Item
column (ItemPurchased here)
);

-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE;
create column table CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE  like
PROCEDURE_SIGNATURE_T;
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
```

```
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_T', 'IN');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (4,
'USER_APL','CUST_TRANSACTIONS_T', 'IN');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (5,
'USER_APL','MODEL_NATIVE_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (6,
'USER_APL','MODEL_RECO_NODES1_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (7,
'USER_APL','MODEL_RECO_NODES2_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (8,
'USER_APL','MODEL_RECO_LINKS_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (9,
'USER_APL','OPERATION_LOG_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (10,
'USER_APL','SUMMARY_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (11,
'USER_APL','INDICATORS_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_CREATE_RECO_MODEL_AND_T
RAIN');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_RECO_MODEL_AND_TRAIN','US
ER_APL', 'APLWRAPPER_CREATE_RECO_MODEL_AND_TRAIN',
CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table RECO_CONFIG;
create table RECO_CONFIG like OPERATION_CONFIG_T;
insert into RECO_CONFIG values ('APL/ModelType', 'recommendation');
insert into RECO_CONFIG values ('APL/User', 'UserID');                  --
mandatory
insert into RECO_CONFIG values ('APL/Item', 'ItemPurchased');           --
mandatory
insert into RECO_CONFIG values ('APL/Weight', 'Quantity');              --
optional
insert into RECO_CONFIG values ('APL/Date', 'Date_PutInCaddy');         --
optional
insert into RECO_CONFIG values ('APL/BestSeller', '11111000000');           --
optional (default: 50000)
insert into RECO_CONFIG values ('APL/MinimumSupport', '4');             --
optional (default: 2)
insert into RECO_CONFIG values ('APL/MinimumConfidence', '0.1');        --
optional (default: 0.05)
insert into RECO_CONFIG values ('APL/MinimumPredictivePower', '0.01');  --
optional (default: disable)
insert into RECO_CONFIG values ('APL/MaxTopNodes', '1000');             --
optional (default: 100000)
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to use guess variables
drop table MODEL_RECO;
create table MODEL_RECO like MODEL_NATIVE_T;
drop table MODEL_NODES1;
create table MODEL_NODES1 like MODEL_RECO_NODES1_T;
drop table MODEL_NODES2;
create table MODEL_NODES2 like MODEL_RECO_NODES2_T;
drop table MODEL_LINKS;
create table MODEL_LINKS like MODEL_RECO_LINKS_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
```

```
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from RECO_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    dataset  = select * from APL_SAMPLES.CUST_TRANSACTIONS;

APLWRAPPER_CREATE_RECO_MODEL_AND_TRAIN(:header, :config, :var_desc, :dataset,
out_model,  out_model_nodes1, out_model_nodes2, out_model_links, out_log,
out_sum, out_indic);
    -- store result into table
    insert into  "USER_APL"."MODEL_RECO"      select * from :out_model;
    insert into  "USER_APL"."MODEL_NODES1"    select * from :out_model_nodes1;
    insert into  "USER_APL"."MODEL_NODES2"    select * from :out_model_nodes2;
    insert into  "USER_APL"."MODEL_LINKS"     select * from :out_model_links;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_NODES1";
    select * from "USER_APL"."MODEL_NODES2";
    select * from "USER_APL"."MODEL_LINKS";
END;
```

8.1.14.4 Example: PROCEDURE

```
-- ====================================================================
-- APL_AREA, CREATE_RECO_MODEL_AND_TRAIN, using a native format for the model
-- This script creates a model, guesses its description and trains it
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type CUST_TRANSACTIONS_T;
create type CUST_TRANSACTIONS_T as table (
    "UserID" INTEGER,
    "ItemPurchased" NVARCHAR(128),
    "Date_PutInCaddy" DATETIME,
    "Quantity" INTEGER,
    "TransactionID" INTEGER
);
-- KxNodes1
drop type MODEL_RECO_NODES1_T;
create type MODEL_RECO_NODES1_T as table (
    "node" INT -- must be of the same SQL type as the User column (UserID here)
);
-- KxNodes2
drop type MODEL_RECO_NODES2_T;
create type MODEL_RECO_NODES2_T as table (
    "node" NVARCHAR(255) -- must be of the same SQL type as the Item column
(ItemPurchased here)
);
-- KxLinks
drop type MODEL_RECO_LINKS_T;
```

```
create type MODEL_RECO_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" INT, -- must be of the same SQL type as the User column
(UserID here)
    "KXNODESECOND" NVARCHAR(255), -- must be of the same SQL type as the Item
column (ItemPurchased here)
    "KXNODESECOND_2" NVARCHAR(255) -- must be of the same SQL type as the Item
column (ItemPurchased here)
);

-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table RECO_CONFIG;
create table RECO_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into RECO_CONFIG values ('APL/ModelType', 'recommendation',null);
insert into RECO_CONFIG values ('APL/User', 'UserID',null);                  --
mandatory
insert into RECO_CONFIG values ('APL/Item', 'ItemPurchased',null);           --
mandatory
insert into RECO_CONFIG values ('APL/Weight', 'Quantity',null);              --
optional
insert into RECO_CONFIG values ('APL/Date', 'Date_PutInCaddy',null);         --
optional
insert into RECO_CONFIG values ('APL/BestSeller', '11111000000',null);
-- optional (default: 50000)
insert into RECO_CONFIG values ('APL/MinimumSupport', '4',null);             --
optional (default: 2)
insert into RECO_CONFIG values ('APL/MinimumConfidence', '0.1',null);        --
optional (default: 0.05)
insert into RECO_CONFIG values ('APL/MinimumPredictivePower', '0.01',null);  --
optional (default: disable)
insert into RECO_CONFIG values ('APL/MaxTopNodes', '1000',null);             --
optional (default: 100000)
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to use guess variables
drop table MODEL_RECO;
create table MODEL_RECO like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_NATIVE";
drop table MODEL_NODES1;
create table MODEL_NODES1 like MODEL_RECO_NODES1_T;
drop table MODEL_NODES2;
create table MODEL_NODES2 like MODEL_RECO_NODES2_T;
drop table MODEL_LINKS;
create table MODEL_LINKS like MODEL_RECO_LINKS_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from RECO_CONFIG;
    var_desc = select * from VARIABLE_DESC;
```

```
"SAP_PA_APL"."sap.pa.apl.base::CREATE_RECO_MODEL_AND_TRAIN"(:header, :config, :va
r_desc, 'APL_SAMPLES','CUST_TRANSACTIONS', out_model,
'USER_APL','MODEL_NODES1','USER_APL', 'MODEL_NODES2', 'USER_APL','MODEL_LINKS',
out_log, out_sum, out_indic,out_export_sql);
    -- store result into table
    insert into  "USER_APL"."MODEL_RECO"      select * from :out_model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_RECO";
    select * from "USER_APL"."MODEL_NODES1";
    select * from "USER_APL"."MODEL_NODES2";
    select * from "USER_APL"."MODEL_LINKS";
END;
```

8.1.14.5 Using the Function to Obtain Recommendations for Similar Items

Use the `CREATE_RECO_MODEL_AND_TRAIN` function to generate a list of similar items. You must set the recommendation type to `'Similarity'`.

When you write the SQL, you include the output table declaration `RECO_CODE`. This generates a stored procedure for the model that you invoke and provides the productID and an attribute. The stored procedure is saved with the name defined by the key `APL/RecommendProcName`, whose default value is `RECOMMEND_SIMILAR_ITEM`.

You call the stored procedure using the following command:

```
CALL RECOMMEND_SIMILAR_ITEM(681, 'frontMaterial-Plastic', ?)
```

The stored procedure will return a table of items that are similar to the provided attribute.

In addition to the input/output tables listed above, you include the following output table declaration:

Additional Output Table for Obtaining the Similar Items Recommendation

Direction	Type	Description
OUT	`RECO_CODE`	The generated SQL code (stored procedure) for items recommendation.

8.1.14.5.1 Example: DIRECT

```
-- ================================================================
-- APL_AREA, CREATE_RECO_MODEL_AND_TRAIN, using a native format for the model
-- This script creates a recommendation model, guesses its description, trains
it and generates the code for similar products recommendation.
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
```

```
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type MOVIES_ATTRIBUTES_T;
create type MOVIES_ATTRIBUTES_T as table (
    "FILM" NVARCHAR(50),
    "ATTRIBUTE" NVARCHAR(80)
);
-- KxNodes1
drop type MODEL_RECO_NODES1_T;
create type MODEL_RECO_NODES1_T as table (
    "node" NVARCHAR(80) -- must be of the same SQL type as the attribute column
(ATTRIBUTE here)
);
-- KxNodes2
drop type MODEL_RECO_NODES2_T;
create type MODEL_RECO_NODES2_T as table (
    "node" NVARCHAR(50) -- must be of the same SQL type as the Item column (FILM
here)
);
-- KxLinks
drop type MODEL_RECO_LINKS_T;
create type MODEL_RECO_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" NVARCHAR(80), -- must be of the same SQL type as the attribute
column (ATTRIBUTE here)
    "KXNODESECOND" NVARCHAR(50), -- must be of the same SQL type as the Item
column (FILM here)
    "KXNODESECOND_2" NVARCHAR(50) -- must be of the same SQL type as the Item
column (FILM here)
);

-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE;
create column table CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE  like
PROCEDURE_SIGNATURE_T;
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_T', 'IN');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (4,
'USER_APL','MOVIES_ATTRIBUTES_T', 'IN');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (5,
'USER_APL','MODEL_NATIVE_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (6,
'USER_APL','MODEL_RECO_NODES1_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (7,
'USER_APL','MODEL_RECO_NODES2_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (8,
'USER_APL','MODEL_RECO_LINKS_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (9,
'USER_APL','OPERATION_LOG_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (10,
'USER_APL','SUMMARY_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (11,
'USER_APL','INDICATORS_T', 'OUT');
insert into CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE values (12,
'USER_APL','RESULT_T', 'OUT');
```

```
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_CREATE_RECO_MODEL_AND_T
RAIN');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_RECO_MODEL_AND_TRAIN','US
ER_APL', 'APLWRAPPER_CREATE_RECO_MODEL_AND_TRAIN',
CREATE_RECO_MODEL_AND_TRAIN_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table RECO_CONFIG;
create table RECO_CONFIG like OPERATION_CONFIG_T;
insert into RECO_CONFIG values ('APL/ModelType', 'recommendation');
insert into RECO_CONFIG values ('APL/User', 'ATTRIBUTE');      -- mandatory
insert into RECO_CONFIG values ('APL/Item', 'FILM');           -- mandatory
insert into RECO_CONFIG values ('APL/RecommendationType', 'similarity');  --
optional (default: standard)
insert into RECO_CONFIG values ('APL/RecommendProcName',
'USER_APL.RECOMMEND_SIMILAR_MOVIES');  -- optional (default:
RECOMMEND_SIMILAR_ITEM)
insert into RECO_CONFIG values ('APL/RecoLinksTable',
'"USER_APL"."MODEL_LINKS"');  -- optional (default: KxLinks1)
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to use guess variables
drop table MODEL_RECO;
create table MODEL_RECO like MODEL_NATIVE_T;
drop table MODEL_NODES1;
create table MODEL_NODES1 like MODEL_RECO_NODES1_T;
drop table MODEL_NODES2;
create table MODEL_NODES2 like MODEL_RECO_NODES2_T;
drop table MODEL_LINKS;
create table MODEL_LINKS like MODEL_RECO_LINKS_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
drop table EXPORT_RECO_SQL;
create table EXPORT_RECO_SQL like RESULT_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from RECO_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    dataset  = select * from APL_SAMPLES.MOVIES_ATTRIBUTES;

APLWRAPPER_CREATE_RECO_MODEL_AND_TRAIN(:header, :config, :var_desc, :dataset,
out_model,  out_model_nodes1, out_model_nodes2, out_model_links, out_log,
out_sum, out_indic, out_export_sql);
    -- store result into table
    insert into  "USER_APL"."MODEL_RECO"      select * from :out_model;
    insert into  "USER_APL"."MODEL_NODES1"    select * from :out_model_nodes1;
    insert into  "USER_APL"."MODEL_NODES2"    select * from :out_model_nodes2;
    insert into  "USER_APL"."MODEL_LINKS"     select * from :out_model_links;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :out_indic;
    insert into  "USER_APL"."EXPORT_RECO_SQL" select * from :out_export_sql;
    -- show result
```

```
    select * from "USER_APL"."MODEL_NODES1";
    select * from "USER_APL"."MODEL_NODES2";
    select * from "USER_APL"."MODEL_LINKS";
    select * from "USER_APL"."EXPORT_RECO_SQL";
END;
```

8.1.14.5.2 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, CREATE_RECO_MODEL_AND_TRAIN, using a native format for the model
-- This script creates a recommendation model, guesses its description, trains
it and generates the code for similar products recommendation.
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type MOVIES_ATTRIBUTES_T;
create type MOVIES_ATTRIBUTES_T as table (
    "FILM" NVARCHAR(50),
    "ATTRIBUTE" NVARCHAR(80)
);
-- KxNodes1
drop type MODEL_RECO_NODES1_T;
create type MODEL_RECO_NODES1_T as table (
    "node" NVARCHAR(80) -- must be of the same SQL type as the attribute column
(ATTRIBUTE here)
);
-- KxNodes2
drop type MODEL_RECO_NODES2_T;
create type MODEL_RECO_NODES2_T as table (
    "node" NVARCHAR(50) -- must be of the same SQL type as the Item column (FILM
here)
);
-- KxLinks
drop type MODEL_RECO_LINKS_T;
create type MODEL_RECO_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" NVARCHAR(80), -- must be of the same SQL type as the attribute
column (ATTRIBUTE here)
    "KXNODESECOND" NVARCHAR(50), -- must be of the same SQL type as the Item
column (FILM here)
    "KXNODESECOND_2" NVARCHAR(50) -- must be of the same SQL type as the Item
column (FILM here)
);

-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table RECO_CONFIG;
create table RECO_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into RECO_CONFIG values ('APL/ModelType', 'recommendation',null);
insert into RECO_CONFIG values ('APL/User', 'ATTRIBUTE',null);      -- mandatory
```

```
insert into RECO_CONFIG values ('APL/Item', 'FILM',null);           -- mandatory
insert into RECO_CONFIG values ('APL/RecommendationType', 'similarity',null);
-- optional (default: standard)
insert into RECO_CONFIG values ('APL/RecommendProcName',
'USER_APL.RECOMMEND_SIMILAR_MOVIES',null);  -- optional (default:
RECOMMEND_SIMILAR_ITEM)
insert into RECO_CONFIG values ('APL/RecoLinksTable',
'"USER_APL"."MODEL_LINKS"',null);  -- optional (default: KxLinks1)
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to use guess variables
drop table MODEL_RECO;
create table MODEL_RECO like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_NATIVE";
drop table MODEL_NODES1;
create table MODEL_NODES1 like MODEL_RECO_NODES1_T;
drop table MODEL_NODES2;
create table MODEL_NODES2 like MODEL_RECO_NODES2_T;
drop table MODEL_LINKS;
create table MODEL_LINKS like MODEL_RECO_LINKS_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table EXPORT_RECO_SQL;
create table EXPORT_RECO_SQL like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.RESULT";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from RECO_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    dataset  = select * from APL_SAMPLES.MOVIES_ATTRIBUTES;

"SAP_PA_APL"."sap.pa.apl.base::CREATE_RECO_MODEL_AND_TRAIN"(:header, :config, :va
r_desc, 'APL_SAMPLES','MOVIES_ATTRIBUTES', out_model,
'USER_APL','MODEL_NODES1','USER_APL', 'MODEL_NODES2', 'USER_APL','MODEL_LINKS',
out_log, out_sum, out_indic, out_export_sql);
    -- store result into table
    insert into  "USER_APL"."MODEL_RECO"      select * from :out_model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :out_indic;
    insert into  "USER_APL"."EXPORT_RECO_SQL" select * from :out_export_sql;
    -- show result
    select * from "USER_APL"."MODEL_NODES1";
    select * from "USER_APL"."MODEL_NODES2";
    select * from "USER_APL"."MODEL_LINKS";
    select * from "USER_APL"."EXPORT_RECO_SQL";
END;
```

8.1.15 CREATE_RECOMMENDER

Creates a recommendation model.

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::CREATE_RECOMMENDER"(FUNC_HEADER,
'Transactions_schema', 'Transactions_name', OPERATION_CONFIG, 'Recommender_schema',
'Recommender_name', SUMMARY)
```

Note

The direct technique and the ANY procedure technique do not support this function.

8.1.15.1 Input/Output Parameters

Direction	Parameter	Data Type	Required	Description
IN	`Transactions_schema`	NVARCHAR(127)	No	The name of the schema used for the table that contains the dataset. If null, the current schema is used.
IN	`Transactions_name`	NVARCHAR(127)	Yes	The reference to the name of the table that contains the dataset
IN	`Recommender_schema`	NVARCHAR(127)	No	The name of the schema used for the table that contains the recommendation model. If null, the current schema is used.
IN	`Recommender_name`	NVARCHAR(127)	Yes	The reference to the name of the table that contains the recommendation model. Will be created if it does not exist.

8.1.15.2 Input/Output Tables

Direction	Table	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	`sap.pa.apl.base::BASE.T.FUNCTION_HEADER`	No	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`OPERATION_CONFIG` [page 384]	`sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED`	Yes	The operation configuration parameters. See table below.

Direction	Table	Type	Required	Description
OUT	SUMMARY [page 388]	sap.pa.apl.base::BASE.T.SUMMARY		The summary of the results

8.1.15.3 OPERATION_CONFIG Parameters

Key	Required	Description	Default
APL/BestSeller	No	The threshold above which an item is considered as a bestseller. If the number of occurrences of an item in the dataset is higher than this value, then this item is considered as a bestseller. Only items below this value are considered in recommendations.	50000
APL/Date	No	The name of the column containing the date in the dataset	
APL/EndDate	No	The end date of the time period considered in the transaction table. APL/Date must be specified.	
APL/Item	Yes	The name of the column containing the item ID in the dataset	
APL/MaxRules	No	The maximum number of recommandation rules allowed in the results. There is no maximum if it is not specified.	
APL/MinimumConfidence	No	The minimum confidence of a recommendation rule. Recommendation rules whose confidence is strictly below this value are discarded.	0.05
APL/MinimumPredictivePower	No	The minimum predictive power of a recommendation rule. Recommendation rules whose predictive power is strictly below this value are discarded.	0.0
APL/MinimumSupport	No	The minimum number of occurrences of an association to be considered as a recommendation rule	2
APL/StartDate	No	The start date of the time period considered in the transaction table. APL/Date must be specified.	
APL/User	Yes	The name of the column containing the user ID in the dataset	

8.1.15.4 Example

```
-- ================================================================
-- APL_AREA, CREATE_RECOMMENDER, there is no model generated
-- This script creates a recommender, which can be then used through
APPLY_RECOMMENDER_TO_BASKET, APPLY_RECOMMENDER_TO_USER, RECO_APPLY_SQL and
RECO_EXCLUDE_HISTORY_ADD_TOP
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create COLUMN table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
drop table RECO_CONFIG;
create table RECO_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into RECO_CONFIG values ('APL/User', 'UserID',
NULL);                      -- mandatory
insert into RECO_CONFIG values ('APL/Item', 'ItemPurchased',
NULL);               -- mandatory
--insert into RECO_CONFIG values ('APL/Date', 'Date_PutInCaddy',
NULL);             -- optional (default: NULL)
--insert into RECO_CONFIG values ('APL/StartDate', '2000-12-20 17:32:24',
NULL);    -- optional (default: NULL)
--insert into RECO_CONFIG values ('APL/EndDate', '2001-03-26 17:32:24',
NULL);      -- optional (default: NULL)
insert into RECO_CONFIG values ('APL/BestSeller', '50000',
NULL);                -- optional (default: 50000)
insert into RECO_CONFIG values ('APL/MinimumSupport', '2',
NULL);                -- optional (default: 2)
insert into RECO_CONFIG values ('APL/MinimumConfidence', '0.05',
NULL);           -- optional (default: 0.05)
insert into RECO_CONFIG values ('APL/MinimumPredictivePower', '0.01',
NULL);      -- optional (default: 0.0)
--insert into RECO_CONFIG values ('APL/MaxRules', '200',
NULL);                    -- optional (default: NULL)
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
DROP TABLE RECOMMENDER_CUST_TRANSACTIONS;
-- The table will be created directly by the procedure if it does not already
exist
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from RECO_CONFIG;
    "SAP_PA_APL"."sap.pa.apl.base::CREATE_RECOMMENDER"(:header,
'APL_SAMPLES','CUST_TRANSACTIONS', :config,'USER_APL',
'RECOMMENDER_CUST_TRANSACTIONS', out_summary);
    -- store result into table
    insert into  "USER_APL"."SUMMARY"         select * from :out_summary;
    -- show result
    --select * from "USER_APL"."RECOMMENDER_CUST_TRANSACTIONS";
    select * from "USER_APL"."SUMMARY";
END;
```

8.1.16 CREATE_SOCIAL_MODEL_AND_TRAIN

Creates a new social model and trains it immediately.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES,
GRAPH_FILTERS, GRAPH_POST_PROCESSING, DATASET, MODEL, NODES1, NODES2, LINKS, COMMUNITIES,
ATTRIBUTES1, ATTRIBUTES2, LOG, SUMMARY, GRAPH_INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::CREATE_SOCIAL_MODEL_AND_TRAIN"(FUNC_HEADER,
OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, GRAPH_FILTERS, GRAPH_POST_PROCESSING,
DATASET_SCHEMA, DATASET_NAME, MODEL, NODES1_SCHEMA, NODES1_NAME, NODES2_SCHEMA,
NODES2_NAME, LINKS_SCHEMA, LINKS_NAME, COMMUNITIES_SCHEMA, COMMUNITIES_NAME,
ATTRIBUTES1_SCHEMA, ATTRIBUTES1_NAME, ATTRIBUTES2_SCHEMA, ATTRIBUTES2_NAME, LOG,
SUMMARY, GRAPH_INDICATORS)
```

The parameters DATASET_SCHEMA, DATASET_NAME, NODES1_SCHEMA, NODES1_NAME, NODES2_SCHEMA, NODES2_NAME, LINKS_SCHEMA, LINKS_NAME, COMMUNITIES_SCHEMA, COMMUNITIES_NAME, ATTRIBUTES1_SCHEMA, ATTRIBUTES1_NAME, ATTRIBUTES2_SCHEMA, and ATTRIBUTES2_NAME, are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_CREATE_SOCIAL_MODEL_AND_TRAIN"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESC, VARIABLE_ROLES, GRAPH_FILTERS, GRAPH_POST_PROCESSING, DATASET, MODEL,
NODES1, NODES2, LINKS, COMMUNITIES, ATTRIBUTES1, ATTRIBUTES2, LOG, SUMMARY,
GRAPH_INDICATORS)
```

Related Information

Social Models Serialization [page 400]

8.1.16.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	OPERATION_CONFIG [page 384]	Yes	The configuration of the create-social-model-and-train operation. Refer to the Compound Parameters table in the OPERATION_CONFIG page. Also refer to the CONFIGURATION table below.
IN	VARIABLE_DESC [page 396]	No (The table must be provided but it can be empty.)	The variable descriptions for the input dataset, as expected by the Automated Analytics engine. If no variable description is provided (i.e. empty table), then the variable descriptions are automatically guessed by the Automated Analytics engine.
IN	VARIABLE_ROLES [page 398]	No (The table must be provided but it can be empty.)	Reserved for future usage
IN	GRAPH_FILTERS [page 406]	No (The table must be provided but it can be empty.)	Filters definitions
IN	GRAPH_POST_PROCESSING [page 407]	No (The table must be provided but it can be empty.)	Post-processing definitions
IN	DATASET [page 362]	Yes	
OUT	MODEL [page 382]		The new model
OUT	NODES [page 411]		The graph node storage (repository1)
OUT	NODES [page 411]		The graph node storage (repository2)
OUT	LINKS		Graph link storage
OUT	COMMUNITIES		Communities storage
OUT	ATTRIBUTES [page 405]		Attributes (repository 1) storage
OUT	ATTRIBUTES [page 405]		Attributes (repository 2) storage
OUT	LOG [page 381]		The apply log
OUT	SUMMARY [page 388]		The training summary

Direction	Type	Required	Description
OUT	GRAPH_INDICATORS [page 406]		The variable statistics and indicators

See Social Models Serialization [page 400] for `COMMUNITIES` and `LINKS` table types.

8.1.16.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Default	Applicable to Graph Type	Description
`APL/ModelType`	Yes	`Social`,			
`APL/Graph`	Yes	`link`, `contact`, `transaction`, `nearestneighbors`		`links`, `contact`	Type of request graph. Graph Types [page 413]
`APL/LinkType`	Yes	`directed`, `undirected`			Define whether the direction of the links is relevant
`APL/SourceNode`	Yes				Column containing the identifier of the source node when creating a directed graph, or the first population when creating a transaction type graph
`APL/TargetNode`	Yes				Column containing the identifier of the target node when creating a directed graph, or the second population when creating a transaction type graph
`APL/Date`	No				Column containing the date information
`APL/StartDate`	No				Allow you to filter the events on a specific period. (Start date, included in the defined period)
`APL/EndDate`	No				Allow you to filter the events on a specific period. (End date, excluded from the defined period)
`APL/MaxConnections`	No		2		Maximum number of connections to create for one node. Once threshold is reached, the node is discarded.

Key	Required	Supported Values	Default	Applicable to Graph Type	Description
APL/Weight	No				Column containing the weight to be attributed to each link. In case of a contact graph type, the link weight in conjunction with the minimum weight can be used to determine which links are going to be created.
APL/MinWeight	No				The minimum weight value needed for a link to be created.
APL/ContactsThreshold	No				The minimum number of occurrences of the couple source node-target node needed in the dataset to create a link.
APL/Distance	No				Column containing the relative distance between nodes
APL/MaxNeighbors	No				The maximum number of nearest neighbors that should be kept in the graph.
APL/Target	No				You can filter your results depending on the value of a specific variable in your dataset. This is the column where the filter is applied.
APL/TargetKey	No		The least frequent category of the Target variable		The category of the Target variable that will be used as the positive target in the model

> **Note**
>
> - APL/SourceNode and APL/TargetNode values must be different.
> - APL/ContactsThreshold is exclusive with APL/Weight and APL/MinimumWeight.

8.1.16.3 Example: DIRECT

```
-- ------------------------------------------------------------
-- APL_AREA, CREATE_SOCIAL_MODEL_AND_TRAIN, using a native format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
```

```
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type CONTACT_EVENTS_T;
create type CONTACT_EVENTS_T as table (
    "CALLER" NVARCHAR(10),
    "CALLEE" NVARCHAR(10),
    "CLASS" NVARCHAR(5),
    "DATE" LONGDATE,
    "DURATION" INTEGER
);
-- KxNodes1
drop type MODEL_SOCIAL_NODES1_T;
create type MODEL_SOCIAL_NODES1_T as table (
    "node" NVARCHAR(10)
);
-- KxNodes2
-- not used here
drop type MODEL_SOCIAL_NODES2_T;
create type MODEL_SOCIAL_NODES2_T as table (
    "node" NVARCHAR(255)
);
-- KxLinks
drop type MODEL_SOCIAL_LINKS_T;
create type MODEL_SOCIAL_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" NVARCHAR(10),
    "KXNODEFIRST_2" NVARCHAR(10)
);
--KxCommunities1
drop type MODEL_SOCIAL_COMMUNITIES_T;
create type MODEL_SOCIAL_COMMUNITIES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "KXNODEFIRST" NVARCHAR(10),
    "COM_INDEX" INTEGER,
    "COM_INTRA" INTEGER,
    "COM_EXTRA" INTEGER
);
-- Only used for transaction graphs with projection, type is not important for
other graphs
drop type MODEL_SOCIAL_ATTRIBUTES_T;
create type MODEL_SOCIAL_ATTRIBUTES_T as table (
    "DUMMY" NVARCHAR(1)
);
drop table CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE;
create column table CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE  like
PROCEDURE_SIGNATURE_T;
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_DETAILED_T', 'IN');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_OID_T', 'IN');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (4,
'USER_APL','VARIABLE_ROLES_WITH_COMPOSITES_OID_T', 'IN');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (5,
'USER_APL','GRAPH_FILTERS_T', 'IN');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (6,
'USER_APL','GRAPH_POST_PROCESSINGS_T', 'IN');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (7,
'USER_APL','CONTACT_EVENTS_T', 'IN');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (8,
'USER_APL','MODEL_NATIVE_T', 'OUT');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (9,
'USER_APL','MODEL_SOCIAL_NODES1_T', 'OUT');
```

```
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (10,
'USER_APL','MODEL_SOCIAL_NODES2_T', 'OUT');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (11,
'USER_APL','MODEL_SOCIAL_LINKS_T', 'OUT');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (12,
'USER_APL','MODEL_SOCIAL_COMMUNITIES_T', 'OUT');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (13,
'USER_APL','MODEL_SOCIAL_ATTRIBUTES_T', 'OUT');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (14,
'USER_APL','MODEL_SOCIAL_ATTRIBUTES_T', 'OUT');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (15,
'USER_APL','OPERATION_LOG_T', 'OUT');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (16,
'USER_APL','SUMMARY_T', 'OUT');
insert into CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE values (17,
'USER_APL','GRAPH_INDICATORS_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_CREATE_SOCIAL_MODEL_AND
_TRAIN');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_SOCIAL_MODEL_AND_TRAIN','
USER_APL', 'APLWRAPPER_CREATE_SOCIAL_MODEL_AND_TRAIN',
CREATE_SOCIAL_MODEL_AND_TRAIN_SIGNATURE);
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table SOCIAL_CONFIG;
create table SOCIAL_CONFIG like OPERATION_CONFIG_DETAILED_T;
insert into SOCIAL_CONFIG values ('APL/ModelType', 'social',null);
insert into SOCIAL_CONFIG values ('APL/Graph', 'contact','graph01');
/** Begin Common **/
insert into SOCIAL_CONFIG values ('APL/LinkType', 'directed','graph01');
insert into SOCIAL_CONFIG values ('APL/SourceNode', 'CALLER','graph01');
insert into SOCIAL_CONFIG values ('APL/TargetNode', 'CALLEE','graph01');
insert into SOCIAL_CONFIG values ('APL/MaxConnections', '50000','graph01');
insert into SOCIAL_CONFIG values ('APL/Date', 'DATE','graph01');
--insert into SOCIAL_CONFIG values ('APL/StartDate', '2007-04-26
00:00:00','graph01');
--insert into SOCIAL_CONFIG values ('APL/EndDate', '2007-06-20
00:00:00','graph01');
/** End Common **/
/** Begin Specifics Contacts **/
--insert into SOCIAL_CONFIG values ('APL/Weight', 'DURATION','graph01');
--insert into SOCIAL_CONFIG values ('APL/MinWeight', '1000','graph01');
insert into SOCIAL_CONFIG values ('APL/ContactsThreshold', '1','graph01');
/** End Specifics Contacts **/
/** Begin configure filters **/
drop table SOCIAL_GRAPH_FILTERS;
create table SOCIAL_GRAPH_FILTERS like GRAPH_FILTERS_T;
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','CALLER','accept','7945321222');
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','CALLER','accept','4169042981');
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','CALLEE','discard','6134491824');
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','CALLEE','discard','6047659299');
-- insert into SOCIAL_GRAPH_FILTERS values ('graph01','DURATION','minimum','0');
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','DURATION','maximum','1000');
/** End configure filters **/
/** Begin Post Processing **/
drop table SOCIAL_POST_PROCESSINGS;
create table SOCIAL_POST_PROCESSINGS like GRAPH_POST_PROCESSINGS_T;
-- Communities detection
insert into SOCIAL_POST_PROCESSINGS values ('CM_1',
'communities','GraphName','graph01');
```

```
insert into SOCIAL_POST_PROCESSINGS values ('CM_1',
'communities','MaxIterations','11');
insert into SOCIAL_POST_PROCESSINGS values ('CM_1',
'communities','Epsilon','4.2E-5');
/*
-- Megahub detection
insert into SOCIAL_POST_PROCESSINGS values ('MEGA_1',
'megahub','GraphName','graph01');
--insert into SOCIAL_POST_PROCESSINGS values ('MEGA_1',
'megahub','DeviationFactor','12');
insert into SOCIAL_POST_PROCESSINGS values ('MEGA_1',
'megahub','Threshold','40000');
insert into SOCIAL_POST_PROCESSINGS values ('MEGA_1',
'megahub','Population','First');
*/
-- Nodes pairing
/*
insert into SOCIAL_POST_PROCESSINGS values ('N_1',
'pairing','FirstGraph','graph01');
insert into SOCIAL_POST_PROCESSINGS values ('N_1',
'pairing','SecondGraph','graph01');
insert into SOCIAL_POST_PROCESSINGS values ('N_1', 'pairing','TopN','69');
insert into SOCIAL_POST_PROCESSINGS values ('N_1',
'pairing','CountThreshold','7');
insert into SOCIAL_POST_PROCESSINGS values ('N_1',
'pairing','PairingGraph','xxx');
insert into SOCIAL_POST_PROCESSINGS values ('N_1',
'pairing','PairingType','Independence_Ratio');
insert into SOCIAL_POST_PROCESSINGS values ('N_1',
'pairing','RatioThreshold','0.7');
insert into SOCIAL_POST_PROCESSINGS values ('N_1',
'pairing','WeightedRatio','false');
insert into SOCIAL_POST_PROCESSINGS values ('N_1',
'pairing','IncludeNeighbors','false');
*/
/** End Post Processing **/
drop table MODEL_SOCIAL;
create table MODEL_SOCIAL like MODEL_NATIVE_T;
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_OID_T;
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_WITH_COMPOSITES_OID_T;
drop table MODEL_SOCIAL_NODES1;
create table MODEL_SOCIAL_NODES1 like MODEL_SOCIAL_NODES1_T;
drop table MODEL_SOCIAL_NODES2;
create table MODEL_SOCIAL_NODES2 like MODEL_SOCIAL_NODES2_T;
drop table MODEL_SOCIAL_LINKS;
create table MODEL_SOCIAL_LINKS like MODEL_SOCIAL_LINKS_T;
drop table MODEL_SOCIAL_COMMUNITIES;
create table MODEL_SOCIAL_COMMUNITIES like MODEL_SOCIAL_COMMUNITIES_T;
drop table MODEL_SOCIAL_ATTRIBUTES1;
create table MODEL_SOCIAL_ATTRIBUTES1 like MODEL_SOCIAL_ATTRIBUTES_T;
drop table MODEL_SOCIAL_ATTRIBUTES2;
create table MODEL_SOCIAL_ATTRIBUTES2 like MODEL_SOCIAL_ATTRIBUTES_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like GRAPH_INDICATORS_T;
DO BEGIN
    header           = select * from FUNC_HEADER;
    config           = select * from SOCIAL_CONFIG;
    var_desc         = select * from VARIABLE_DESC;
    var_roles        = select * from VARIABLE_ROLES;
    graph_filters    = select * from SOCIAL_GRAPH_FILTERS;
    post_processings = select * from SOCIAL_POST_PROCESSINGS;
    dataset          = select * from APL_SAMPLES.CONTACT_EVENTS;
```

```
    APLWRAPPER_CREATE_SOCIAL_MODEL_AND_TRAIN(
        :header,
        :config,
        :var_desc,
        :var_roles,
        :graph_filters,
        :post_processings,
        :dataset,
        out_model,
        out_model_nodes1,
        out_model_nodes2,
        out_model_links,
        out_model_communities,
        out_model_attributes1,
        out_model_attributes2,
        out_log,
        out_summary,
        out_indicators);
    -- store result into table
    insert into  "USER_APL"."MODEL_SOCIAL"               select *
from :out_model;
    insert into  "USER_APL"."MODEL_SOCIAL_NODES1"        select *
from :out_model_nodes1;
    insert into  "USER_APL"."MODEL_SOCIAL_NODES2"        select *
from :out_model_nodes2;
    insert into  "USER_APL"."MODEL_SOCIAL_LINKS"         select *
from :out_model_links;
    insert into  "USER_APL"."MODEL_SOCIAL_COMMUNITIES"   select *
from :out_model_communities;
    insert into  "USER_APL"."MODEL_SOCIAL_ATTRIBUTES1"   select *
from :out_model_attributes1;
    insert into  "USER_APL"."MODEL_SOCIAL_ATTRIBUTES2"   select *
from :out_model_attributes2;
    insert into  "USER_APL"."OPERATION_LOG"              select * from :out_log;
    insert into  "USER_APL"."SUMMARY"                    select *
from :out_summary;
    insert into  "USER_APL"."INDICATORS"                 select *
from :out_indicators;
    -- show result
    select * from MODEL_SOCIAL_NODES1;
    select * from MODEL_SOCIAL_LINKS;
    select * from MODEL_SOCIAL_COMMUNITIES;
    select * from MODEL_SOCIAL;
    select * from SUMMARY;
    select * from INDICATORS;
    select * from OPERATION_LOG;
END;
```

8.1.16.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, CREATE_SOCIAL_MODEL_AND_TRAIN, using a native format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- KxNodes1
drop type MODEL_SOCIAL_NODES1_T;
create type MODEL_SOCIAL_NODES1_T as table (
```

```
    "node" NVARCHAR(10)
);
-- KxNodes2
-- not used here
drop type MODEL_SOCIAL_NODES2_T;
create type MODEL_SOCIAL_NODES2_T as table (
    "node" NVARCHAR(255)
);
-- KxLinks
drop type MODEL_SOCIAL_LINKS_T;
create type MODEL_SOCIAL_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" NVARCHAR(10),
    "KXNODEFIRST_2" NVARCHAR(10)
);
--KxCommunities1
drop type MODEL_SOCIAL_COMMUNITIES_T;
create type MODEL_SOCIAL_COMMUNITIES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "KXNODEFIRST" NVARCHAR(10),
    "COM_INDEX" INTEGER,
    "COM_INTRA" INTEGER,
    "COM_EXTRA" INTEGER
);
drop type MODEL_SOCIAL_ATTRIBUTES_T;
create type MODEL_SOCIAL_ATTRIBUTES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "UserID" INTEGER,
    "generation_id" INTEGER,
    "SOURCE_NODES" NVARCHAR(255),  -- same type than ItemPurchased
    "HEAD_NODE_ID" INTEGER,
    "TAIL_NODE_ID" INTEGER
);
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table SOCIAL_CONFIG;
create table SOCIAL_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into SOCIAL_CONFIG values ('APL/ModelType', 'social',null);
insert into SOCIAL_CONFIG values ('APL/Graph', 'link','graph01');
/** Begin Common **/
insert into SOCIAL_CONFIG values ('APL/LinkType', 'directed','graph01');
insert into SOCIAL_CONFIG values ('APL/SourceNode', 'CALLER','graph01');
insert into SOCIAL_CONFIG values ('APL/TargetNode', 'CALLEE','graph01');
insert into SOCIAL_CONFIG values ('APL/MaxConnections', '50000','graph01');
insert into SOCIAL_CONFIG values ('APL/Date', 'DATE','graph01');
--insert into SOCIAL_CONFIG values ('APL/StartDate', '2007-04-26
00:00:00','graph01');
--insert into SOCIAL_CONFIG values ('APL/EndDate', '2007-06-20
00:00:00','graph01');
/** End Common **/
/** Begin Specifics (Link-Only)**/
insert into SOCIAL_CONFIG values ('APL/Weight', 'DURATION','graph01');
/** End Specifics (Link-Only)**/
drop table SOCIAL_GRAPH_FILTERS;
create table SOCIAL_GRAPH_FILTERS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.GRAPH_FILTERS";
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','CALLER','accept','7945321222');
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','CALLER','accept','4169042981');
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','CALLEE','discard','6134491824');
```

```
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','CALLEE','discard','6047659299');
-- insert into SOCIAL_GRAPH_FILTERS values ('graph01','DURATION','minimum','0');
-- insert into SOCIAL_GRAPH_FILTERS values
('graph01','DURATION','maximum','1000');
drop table SOCIAL_POST_PROCESSINGS;
create table SOCIAL_POST_PROCESSINGS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.GRAPH_POST_PROCESSINGS";
-- Communities detection
insert into SOCIAL_POST_PROCESSINGS values ('CM_1',
'communities','GraphName','graph01');
insert into SOCIAL_POST_PROCESSINGS values ('CM_1',
'communities','MaxIterations','11');
insert into SOCIAL_POST_PROCESSINGS values ('CM_1',
'communities','Epsilon','4.2E-5');
drop table MODEL_SOCIAL;
create table MODEL_SOCIAL like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_NATIVE";
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
drop table MODEL_SOCIAL_NODES1;
create table MODEL_SOCIAL_NODES1 like MODEL_SOCIAL_NODES1_T;
drop table MODEL_SOCIAL_NODES2;
create table MODEL_SOCIAL_NODES2 like MODEL_SOCIAL_NODES2_T;
drop table MODEL_SOCIAL_LINKS;
create table MODEL_SOCIAL_LINKS like MODEL_SOCIAL_LINKS_T;
drop table MODEL_SOCIAL_COMMUNITIES;
create table MODEL_SOCIAL_COMMUNITIES like MODEL_SOCIAL_COMMUNITIES_T;
drop table MODEL_SOCIAL_ATTRIBUTES1;
create table MODEL_SOCIAL_ATTRIBUTES1 like MODEL_SOCIAL_ATTRIBUTES_T;
drop table MODEL_SOCIAL_ATTRIBUTES2;
create table MODEL_SOCIAL_ATTRIBUTES2 like MODEL_SOCIAL_ATTRIBUTES_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.GRAPH_INDICATORS";
DO BEGIN
    header           = select * from FUNC_HEADER;
    config           = select * from SOCIAL_CONFIG;
    var_desc         = select * from VARIABLE_DESC;
    var_roles        = select * from VARIABLE_ROLES;
    graph_filters    = select * from SOCIAL_GRAPH_FILTERS;
    post_processings = select * from SOCIAL_POST_PROCESSINGS;
    dataset          = select * from APL_SAMPLES.CONTACT_EVENTS;

    "SAP_PA_APL"."sap.pa.apl.base::CREATE_SOCIAL_MODEL_AND_TRAIN"(
        :header,
        :config,
        :var_desc,
        :var_roles,
        :graph_filters,
        :post_processings,
        'APL_SAMPLES', 'CONTACT_EVENTS',
        out_model,
        'USER_APL','MODEL_SOCIAL_NODES1',
        'USER_APL','MODEL_SOCIAL_NODES2',
        'USER_APL','MODEL_SOCIAL_LINKS',
        'USER_APL','MODEL_SOCIAL_COMMUNITIES',
        'USER_APL','MODEL_SOCIAL_ATTRIBUTES1',
        'USER_APL','MODEL_SOCIAL_ATTRIBUTES2',
```

```
        out_log,
        out_summary,
        out_indicators);
    -- store result into table
    insert into  "USER_APL"."MODEL_SOCIAL"                select *
from :out_model;
    insert into  "USER_APL"."OPERATION_LOG"               select * from :out_log;
    insert into  "USER_APL"."SUMMARY"                     select *
from :out_summary;
    insert into  "USER_APL"."INDICATORS"                  select *
from :out_indicators;
    -- show result
    select * from MODEL_SOCIAL_NODES1;
    select * from MODEL_SOCIAL_LINKS;
    select * from MODEL_SOCIAL_COMMUNITIES;
    select * from MODEL_SOCIAL_ATTRIBUTES1;
    select * from MODEL_SOCIAL_ATTRIBUTES2;
    select * from MODEL_SOCIAL;
    select * from SUMMARY;
    select * from INDICATORS;
    select * from OPERATION_LOG;
END;
```

8.1.17 EXPORT_APPLY_CODE

Exports code (SQL, Java, and so on) so that you can apply a trained model outside SAP HANA APL.

This function does not support time series and social network analysis models. For regression models (gradient boosting), binary classification models (gradient boosting), and multinomial classification models (gradient boosting), only JSON export is supported.

Use EXPORT_APPLY_CODE_FOR_RECO [page 191] for recommendation models.

> **Note**
>
> SQL code cannot be exported for the Apply of a classification model of polynomial degree greater than 1.

DIRECT

> **Syntax**
>
> `<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, RESULT)`

PROCEDURE

> **Syntax**
>
> `"SAP_PA_APL"."sap.pa.apl.base::EXPORT_APPLY_CODE"(FUNC_HEADER, MODEL, OPERATION_CONFIG, RESULT)`

ANY PROCEDURE

> **Syntax**
>
> `"_SYS_AFL"."APL_EXPORT_APPLY_CODE"(FUNC_HEADER, MODEL, OPERATION_CONFIG, RESULT)`

8.1.17.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL` [page 382]	Yes	The model to be exported
IN	`OPERATION_CONFIG` [page 384]	Yes	The configuration of the training operation. For this function, you must declare the type of Automated Analytics model and you can declare the optional partition strategy in the `OPERATION_CONFIG` table as shown in the next section.
OUT	`RESULT` [page 387]		The exported code

8.1.17.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
`APL/CodeType`	Yes	• `'HANA'` • `'HANAUDF'` • `'HTML'` • `'CCL'` • `'JAVA'` • `'C'` • `'C++'` • `'JSON'`	The type of code exported. Gradient boosting algorithms only support JSON code export. For more information, see the table below.
`APL/CodeTarget`	Yes		A valid target name
`APL/CodeSpace`	Yes		A valid table name
`APL/CodeKey`	Yes		A valid key name

Key	Required	Supported Values	Description
`APL/CodeClassName`	No	• None: The default value of the generated class name is 'JavaKxModel' for Java code and 'CppKxModel' for C++ code. • A string: A valid Java or C++ identifier of the class to be generated	The name of the Java or C++ class to be generated
`APL/ApplyExtraMode`	No	For regression/binary classification: • `'No Extra'` (default value) • `'Min Extra'` • `'Advanced Apply Settings'` For clustering: • `'No Extra'` (default value)	Defines the information you want to get in your ApplyOut result table. The resulting table will have different columns depending on the value chosen. Depending on the model (model type, number of targets ...) not all extra modes can be used. See Automated Analytics documentation for details.
`APL/ApplyBar`	No	• `'true'` • `'false'`	Allows the confidence interval on the score value to be generated in the output. Requires `'Advanced Apply Settings'` as `APL/ApplyExtraMode`. **Note** Binary classification models and regression models only.
`APL/ApplyContribution`	No	• `'none'` (default value) • `'all'` • `'<var_name>;<var_name>'`	Specifies whether contributions of variables should be produced. Requires `'Advanced Apply Settings'` as `APL/ApplyExtraMode`. A new {key, value} row is added to the `RESULT` table. It contains the key `pivotForContribution` and a SQL code as value that produces contribution variables in rows instead of columns in the output table. The three new columns `Id`, `ContribName` and `ContribValue` contain the contribution variable IDs, names, and values. **Note** Binary classification models only.

Key	Required	Supported Values	Description
`APL/ApplyDecision`	No	• `'true'` • `'false'` (default value)	Allows the best decisions to be generated in the output. Requires `'Advanced Apply Settings'` as `APL/ApplyExtraMode`. **Note** Binary classification models only.
`APL/ApplyPredictedValue`	No	• `'true'` • `'false'` (default value)	Allows the score value to be generated in the output. Requires `'Advanced Apply Settings'` as `APL/ApplyExtraMode`. **Note** Binary classification models and regression models only.
`APL/ApplyProba`	No	• `'true'` • `'false'`	Allows the probability to be generated in the output. Requires `'Advanced Apply Settings'` as `APL/ApplyExtraMode`. **Note** Binary classification models only.
`APL/ApplyProbaDecision`	No	• `'true'` • `'false'` (default value)	Allows the probability for one or more target variable categories to be generated in the output. Requires `'Advanced Apply Settings'` as `APL/ApplyExtraMode`. **Note** Binary classification models and multinomial classification models only.
`APL/CodeUseVarNameAlias`	No	• `'true'` • `'false'` (default value)	Uses the variable name as an alias
`APL/SegmentColumnName`	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

8.1.17.3 Supported Code Export Types

Code Type	Regression (Gradient Boosting)	Classification (Gradient Boosting)	Multiclass (Gradient Boosting)	Regression	Classification	Clustering	Time Series	Social Network
HANA	x	x	x	√	√	√	x	x
HANAUDF	x	x	x	√	√	√	x	x
HTML	x	x	x	√	√	√	x	x
CCL	x	x	x	√	√	√	x	x
JAVA	x	x	x	√	√	√	x	x
C	x	x	x	√	√	√	x	x
C++	x	x	x	√	√	√	x	x
JSON	√	√	√	√	x	x	x	x

For some exported code types (C++, Java, and Javascript), runtime files are needed to execute the exported code. Reference implementations of these runtimes are provided.

Related Information

Runtimes [page 479]

8.1.17.4 Example: DIRECT

```
-- ================================================================
-- APL_AREA, EXPORT_APPLY_CODE
--
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid and trained model published by Automatedin in
the MODELS table.
--  @depend(apl_createmodel_and_train.sql)
connect USER_APL password Password1;
-- ----------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ----------------------------------------------------------------------------
drop table EXPORT_APPLY_CODE_SIGNATURE;
create column table EXPORT_APPLY_CODE_SIGNATURE like PROCEDURE_SIGNATURE_T;
insert into EXPORT_APPLY_CODE_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into EXPORT_APPLY_CODE_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',
'IN');
insert into EXPORT_APPLY_CODE_SIGNATURE values (3,
'USER_APL','OPERATION_CONFIG_T', 'IN');
```

```
insert into EXPORT_APPLY_CODE_SIGNATURE values (4, 'USER_APL','RESULT_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_EXPORT_APPLY_CODE');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','EXPORT_APPLY_CODE','USER_APL',
'APLWRAPPER_EXPORT_APPLY_CODE', EXPORT_APPLY_CODE_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
drop table EXPORT_SQL;
create table EXPORT_SQL like RESULT_T;
drop table EXPORT_SQL_WITH_EXTRA;
create table EXPORT_SQL_WITH_EXTRA like RESULT_T;
drop table EXPORT_JAVA;
create table EXPORT_JAVA like RESULT_T;
drop table EXPORT_CPP;
create table EXPORT_CPP like RESULT_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
-- Export SQL for HANA
drop table EXPORT_CODE_CONFIG_1;
create table EXPORT_CODE_CONFIG_1 like OPERATION_CONFIG_T;
insert into EXPORT_CODE_CONFIG_1 values ('APL/CodeType', 'HANA');
insert into EXPORT_CODE_CONFIG_1 values ('APL/CodeTarget', 'class');
insert into EXPORT_CODE_CONFIG_1 values ('APL/CodeKey', '"id"');
insert into EXPORT_CODE_CONFIG_1 values ('APL/CodeSpace', 'APL_SAMPLES.CENSUS');
insert into EXPORT_CODE_CONFIG_1 values ('APL/ApplyExtraMode', 'No Extra');
-- Export score SQL and stats for HANA
drop table EXPORT_CODE_CONFIG_2;
create table EXPORT_CODE_CONFIG_2 like OPERATION_CONFIG_T;
insert into EXPORT_CODE_CONFIG_2 values ('APL/CodeType', 'HANA');
insert into EXPORT_CODE_CONFIG_2 values ('APL/CodeTarget', 'class');
insert into EXPORT_CODE_CONFIG_2 values ('APL/CodeKey', '"id"');
insert into EXPORT_CODE_CONFIG_2 values ('APL/CodeSpace', 'APL_SAMPLES.CENSUS');
insert into EXPORT_CODE_CONFIG_2 values ('APL/ApplyExtraMode', 'Min Extra');
-- Export JAVA
drop table EXPORT_CODE_CONFIG_3;
create table EXPORT_CODE_CONFIG_3 like OPERATION_CONFIG_T;
insert into EXPORT_CODE_CONFIG_3 values ('APL/CodeType', 'JAVA');
insert into EXPORT_CODE_CONFIG_3 values ('APL/CodeTarget', 'class');
insert into EXPORT_CODE_CONFIG_3 values ('APL/CodeClassName',
'ExportedModelInJava');
insert into EXPORT_CODE_CONFIG_3 values ('APL/ApplyExtraMode', 'No Extra');
-- Export C++
drop table EXPORT_CODE_CONFIG_4;
create table EXPORT_CODE_CONFIG_4 like OPERATION_CONFIG_T;
insert into EXPORT_CODE_CONFIG_4 values ('APL/CodeType', 'C++');
insert into EXPORT_CODE_CONFIG_4 values ('APL/CodeTarget', 'class');
insert into EXPORT_CODE_CONFIG_4 values ('APL/CodeClassName',
'ExportedModelInCPP');
insert into EXPORT_CODE_CONFIG_4 values ('APL/ApplyExtraMode', 'No Extra');
DO BEGIN
    header           = select * from FUNC_HEADER;
    model_in         = select * from MODEL_TRAIN_BIN;
    export_1_config  = select * from EXPORT_CODE_CONFIG_1;
    export_2_config  = select * from EXPORT_CODE_CONFIG_2;
    export_3_config  = select * from EXPORT_CODE_CONFIG_3;
    export_4_config  = select * from EXPORT_CODE_CONFIG_4;

APLWRAPPER_EXPORT_APPLY_CODE(:header, :model_in, :export_1_config, :out_code1);

APLWRAPPER_EXPORT_APPLY_CODE(:header, :model_in, :export_2_config, :out_code2);

APLWRAPPER_EXPORT_APPLY_CODE(:header, :model_in, :export_3_config, :out_code3);
```

```
APLWRAPPER_EXPORT_APPLY_CODE(:header, :model_in, :export_4_config, :out_code4);
    -- store result into table
    insert into  "USER_APL"."EXPORT_SQL"            select * from :out_code1;
    insert into  "USER_APL"."EXPORT_SQL_WITH_EXTRA" select * from :out_code2;
    insert into  "USER_APL"."EXPORT_JAVA"           select * from :out_code3;
    insert into  "USER_APL"."EXPORT_CPP"            select * from :out_code4;
    -- show result
    select * from "USER_APL"."EXPORT_SQL";
    select * from "USER_APL"."EXPORT_SQL_WITH_EXTRA";
    select * from "USER_APL"."EXPORT_JAVA";
    select * from "USER_APL"."EXPORT_CPP";
END;
```

8.1.17.5 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, EXPORT_APPLY_CODE
--
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 3: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train_ex.sql
before
--  @depend(apl_createmodel_and_train_ex.sql)
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
drop table EXPORT_SQL;
create table EXPORT_SQL like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.RESULT";
drop table EXPORT_SQL_WITH_EXTRA;
create table EXPORT_SQL_WITH_EXTRA like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.RESULT";
drop table EXPORT_JAVA;
create table EXPORT_JAVA like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.RESULT";
drop table EXPORT_CPP;
create table EXPORT_CPP like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.RESULT";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
-- Export SQL for HANA
drop table EXPORT_CODE_CONFIG_1;
create table EXPORT_CODE_CONFIG_1 like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into EXPORT_CODE_CONFIG_1 values ('APL/CodeType', 'HANA',null);
insert into EXPORT_CODE_CONFIG_1 values ('APL/CodeTarget', 'class',null);
insert into EXPORT_CODE_CONFIG_1 values ('APL/CodeKey', '"id"',null);
insert into EXPORT_CODE_CONFIG_1 values ('APL/CodeSpace',
'APL_SAMPLES.CENSUS',null);
insert into EXPORT_CODE_CONFIG_1 values ('APL/ApplyExtraMode', 'No Extra',null);
-- Export score SQL and stats for HANA
drop table EXPORT_CODE_CONFIG_2;
create table EXPORT_CODE_CONFIG_2 like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into EXPORT_CODE_CONFIG_2 values ('APL/CodeType', 'HANA',null);
insert into EXPORT_CODE_CONFIG_2 values ('APL/CodeTarget', 'class',null);
```

```
insert into EXPORT_CODE_CONFIG_2 values ('APL/CodeKey', '"id"',null);
insert into EXPORT_CODE_CONFIG_2 values ('APL/CodeSpace',
'APL_SAMPLES.CENSUS',null);
insert into EXPORT_CODE_CONFIG_2 values ('APL/ApplyExtraMode', 'Min Extra',null);
-- Export JAVA
drop table EXPORT_CODE_CONFIG_3;
create table EXPORT_CODE_CONFIG_3 like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into EXPORT_CODE_CONFIG_3 values ('APL/CodeType', 'JAVA',null);
insert into EXPORT_CODE_CONFIG_3 values ('APL/CodeTarget', 'class',null);
insert into EXPORT_CODE_CONFIG_3 values ('APL/CodeClassName',
'ExportedModelInJava',null);
insert into EXPORT_CODE_CONFIG_3 values ('APL/ApplyExtraMode', 'No Extra',null);
-- Export C++
drop table EXPORT_CODE_CONFIG_4;
create table EXPORT_CODE_CONFIG_4 like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into EXPORT_CODE_CONFIG_4 values ('APL/CodeType', 'C++',null);
insert into EXPORT_CODE_CONFIG_4 values ('APL/CodeTarget', 'class',null);
insert into EXPORT_CODE_CONFIG_4 values ('APL/CodeClassName',
'ExportedModelInCPP',null);
insert into EXPORT_CODE_CONFIG_4 values ('APL/ApplyExtraMode', 'No Extra',null);
DO BEGIN
    header           = select * from FUNC_HEADER;
    model_in         = select * from MODEL_TRAIN_BIN;
    export_1_config  = select * from EXPORT_CODE_CONFIG_1;
    export_2_config  = select * from EXPORT_CODE_CONFIG_2;
    export_3_config  = select * from EXPORT_CODE_CONFIG_3;
    export_4_config  = select * from EXPORT_CODE_CONFIG_4;

"SAP_PA_APL"."sap.pa.apl.base::EXPORT_APPLY_CODE"(:header, :model_in, :export_1_c
onfig, :out_code1);

"SAP_PA_APL"."sap.pa.apl.base::EXPORT_APPLY_CODE"(:header, :model_in, :export_2_c
onfig, :out_code2);

"SAP_PA_APL"."sap.pa.apl.base::EXPORT_APPLY_CODE"(:header, :model_in, :export_3_c
onfig, :out_code3);

"SAP_PA_APL"."sap.pa.apl.base::EXPORT_APPLY_CODE"(:header, :model_in, :export_4_c
onfig, :out_code4);
    -- store result into table
    insert into  "USER_APL"."EXPORT_SQL"            select * from :out_code1;
    insert into  "USER_APL"."EXPORT_SQL_WITH_EXTRA" select * from :out_code2;
    insert into  "USER_APL"."EXPORT_JAVA"           select * from :out_code3;
    insert into  "USER_APL"."EXPORT_CPP"            select * from :out_code4;
    -- show result
    select * from "USER_APL"."EXPORT_SQL";
    select * from "USER_APL"."EXPORT_SQL_WITH_EXTRA";
    select * from "USER_APL"."EXPORT_JAVA";
    select * from "USER_APL"."EXPORT_CPP";
END;
```

8.1.17.6 Example: PROCEDURE with Advanced Settings

```
-- ================================================================
-- APL_AREA, EXPORT_APPLY_CODE, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained classification model (created by
APL) in the MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train_ex.sql
before
```

```
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
-- ------------------------------------------------------------------------
-- Export SQL for HANA
-- ------------------------------------------------------------------------
drop table EXPORT_SQL;
create table EXPORT_SQL like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.RESULT";
drop table EXPORT_CONFIG;
create table EXPORT_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into EXPORT_CONFIG values ('APL/ApplyExtraMode', 'Advanced Apply
Settings',null);
insert into EXPORT_CONFIG values ('APL/ApplyProba','true',null);
insert into EXPORT_CONFIG values ('APL/ApplyBar','true',null);
insert into EXPORT_CONFIG values ('APL/ApplyDecision','true',null);
insert into EXPORT_CONFIG values ('APL/ApplyProbaDecision','true',null);
insert into EXPORT_CONFIG values ('APL/
ApplyContribution','education;occupation',null);
insert into EXPORT_CONFIG values ('APL/CodeType', 'HANA',null);
insert into EXPORT_CONFIG values ('APL/CodeTarget', 'class',null);
insert into EXPORT_CONFIG values ('APL/CodeKey', '"id"',null);
insert into EXPORT_CONFIG values ('APL/CodeSpace', 'APL_SAMPLES.CENSUS',null);
DO BEGIN
    header           = select * from FUNC_HEADER;
    model_in         = select * from MODEL_TRAIN_BIN;
    export_config    = select * from EXPORT_CONFIG;

"SAP_PA_APL"."sap.pa.apl.base::EXPORT_APPLY_CODE"(:header, :model_in, :export_con
fig, :out_code1);

    -- store result into table
    insert into  "USER_APL"."EXPORT_SQL"           select * from :out_code1;
    -- show result
    select * from "USER_APL"."EXPORT_SQL";
END;
```

8.1.18 EXPORT_APPLY_CODE_FOR_RECO

Exports code so that you can apply a recommendation model outside SAP HANA APL.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, NODES1, NODES2,LINKS, OPERATION_CONFIG, RESULT)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::EXPORT_APPLY_CODE_FOR_RECO"(FUNC_HEADER, MODEL,
NODES1_SCHEMA, NODES1_NAME, NODES2_SCHEMA, NODES2_NAME, LINKS_SCHEMA, LINKS_NAME,
OPERATION_CONFIG, RESULT)
```

The parameters `NODES1_SCHEMA`, `NODES1_NAME`, `NODES2_SCHEMA`, `NODES2_NAME`, `LINKS_SCHEMA`, and `LINKS_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_EXPORT_APPLY_CODE_FOR_RECO"(FUNC_HEADER, MODEL, NODES1, NODES2,LINKS, OPERATION_CONFIG, RESULT)
```

8.1.18.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty).	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL` [page 382]		The model
IN	`NODES` [page 411]		Graph node storage (repository 1)
IN	`NODES` [page 411]		Graph node storage (repository 2)
IN	`LINKS` [page 410]		Graph link storage
IN	`OPERATION_CONFIG` [page 384]	Yes	The configuration of the training operation.
OUT	`RESULT` [page 387]		The exported code. See Result of Operation

8.1.18.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Default	Description
`APL/CodeType`	Yes	`'SQL'`		Type of code to export
`APL/CodeSpace`	No			Table name
`APL/CodeKey`	No			Key name

Key	Required	Supported Values	Default	Description
APL/Top	No		max	Keep top N of recommended items ranked by confidence
APL/IncludeBestSellers	No		false	Include de-facto best seller items into the suggestion list
APL/SkipAlreadyOwned	No		true	Exclude items already owned from the suggestion list

8.1.18.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, EXPORT_APPLY_CODE_FOR_RECO, using a native format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid and trained model (created by APL) in the
MODEL_SORTED, MODEL_NODES1, MODEL_NODES2, MODEL_LINKS tables.
--               For instance, you have used apl_create_reco_model_and_train.sql
before
connect USER_APL password Password1;
-- ----------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ----------------------------------------------------------------------------
drop table EXPORT_APPLY_CODE_SIGNATURE;
create column table EXPORT_APPLY_CODE_SIGNATURE like PROCEDURE_SIGNATURE_T;
-- KxNodes1
drop type MODEL_RECO_NODES1_T;
create type MODEL_RECO_NODES1_T as table (
    "node" INT -- must be of the same SQL type as the User column (UserID here)
);
-- KxNodes2
drop type MODEL_RECO_NODES2_T;
create type MODEL_RECO_NODES2_T as table (
    "node" NVARCHAR(255) -- must be of the same SQL type as the Item column
(ItemPurchased here)
);
-- KxLinks
drop type MODEL_RECO_LINKS_T;
create type MODEL_RECO_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" INT, -- must be of the same SQL type as the User column
(UserID here)
    "KXNODESECOND" NVARCHAR(255), -- must be of the same SQL type as the Item
column (ItemPurchased here)
    "KXNODESECOND_2" NVARCHAR(255) -- must be of the same SQL type as the Item
column (ItemPurchased here)
);

insert into EXPORT_APPLY_CODE_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into EXPORT_APPLY_CODE_SIGNATURE values (2, 'USER_APL','MODEL_NATIVE_T',
'IN');
```

```
insert into EXPORT_APPLY_CODE_SIGNATURE values (3,
'USER_APL','MODEL_RECO_NODES1_T', 'IN');
insert into EXPORT_APPLY_CODE_SIGNATURE values (4,
'USER_APL','MODEL_RECO_NODES2_T', 'IN');
insert into EXPORT_APPLY_CODE_SIGNATURE values (5,
'USER_APL','MODEL_RECO_LINKS_T', 'IN');
insert into EXPORT_APPLY_CODE_SIGNATURE values (6,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into EXPORT_APPLY_CODE_SIGNATURE values (7, 'USER_APL','RESULT_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_EXPORT_APPLY_CODE_FOR_R
ECO');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','EXPORT_APPLY_CODE_FOR_RECO','USE
R_APL', 'APLWRAPPER_EXPORT_APPLY_CODE_FOR_RECO', EXPORT_APPLY_CODE_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
drop table EXPORT_RECO_SQL;
create table EXPORT_RECO_SQL like RESULT_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
-- Export SQL for HANA
drop table EXPORT_CONFIG;
create table EXPORT_CONFIG like OPERATION_CONFIG_T;
insert into EXPORT_CONFIG values ('APL/CodeType', 'HANA');
insert into EXPORT_CONFIG values ('APL/CodeKey', '"TransactionID"'); --(use
double quote to force case sensitivity if the table column name is not created
in all caps)
insert into EXPORT_CONFIG values ('APL/CodeSpace',
'APL_SAMPLES.CUST_TRANSACTIONS');
drop view MODEL_SORTED;
create view MODEL_SORTED AS SELECT *  from MODEL_RECO order by "ID" asc  ;
DO BEGIN
    header       = select * from FUNC_HEADER;
    model        = select * from MODEL_SORTED;
    model_nodes1 = select * from MODEL_NODES1;
    model_nodes2 = select * from MODEL_NODES2;
    model_links  = select * from MODEL_LINKS;
    config       = select * from EXPORT_CONFIG;

APLWRAPPER_EXPORT_APPLY_CODE_FOR_RECO(:header, :model, :model_nodes1, :model_node
s2, :model_links, :config, :out_export);
    -- store result into table
    insert into  "USER_APL"."EXPORT_RECO_SQL"   select * from :out_export;
    -- show result
    select * from "USER_APL"."EXPORT_RECO_SQL";
END;
```

8.1.18.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, EXPORT_APPLY_CODE_FOR_RECO, using a native format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained model (created by APL) in the
MODEL_SORTED, MODEL_NODES1, MODEL_NODES2, MODEL_LINKS tables.
--               For instance, you have used
apl_create_reco_model_and_train_ex.sql before
```

```
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- KxNodes1
drop type MODEL_RECO_NODES1_T;
create type MODEL_RECO_NODES1_T as table (
    "node" INT -- must be of the same SQL type as the User column (UserID here)
);
-- KxNodes2
drop type MODEL_RECO_NODES2_T;
create type MODEL_RECO_NODES2_T as table (
    "node" NVARCHAR(255) -- must be of the same SQL type as the Item column
(ItemPurchased here)
);
-- KxLinks
drop type MODEL_RECO_LINKS_T;
create type MODEL_RECO_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" INT, -- must be of the same SQL type as the User column
(UserID here)
    "KXNODESECOND" NVARCHAR(255), -- must be of the same SQL type as the Item
column (ItemPurchased here)
    "KXNODESECOND_2" NVARCHAR(255) -- must be of the same SQL type as the Item
column (ItemPurchased here)
);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
drop table EXPORT_RECO_SQL;
create table EXPORT_RECO_SQL like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.RESULT";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
-- Export SQL for HANA
drop table EXPORT_CONFIG;
create table EXPORT_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into EXPORT_CONFIG values ('APL/CodeType', 'HANA',null);
insert into EXPORT_CONFIG values ('APL/CodeKey', '"TransactionID"',null); --(use
double quote to force case sensitivity if the table column name is not created
in all caps)
insert into EXPORT_CONFIG values ('APL/CodeSpace',
'APL_SAMPLES.CUST_TRANSACTIONS',null);
DO BEGIN
    header       = select * from FUNC_HEADER;
    model        = select * from MODEL_RECO;
    config       = select * from EXPORT_CONFIG;
    "SAP_PA_APL"."sap.pa.apl.base::EXPORT_APPLY_CODE_FOR_RECO"(:header, :model,
'USER_APL','MODEL_NODES1','USER_APL','MODEL_NODES2','USER_APL','MODEL_LINKS', :co
nfig, :out_export);
    -- store result into table
    insert into  "USER_APL"."EXPORT_RECO_SQL"   select * from :out_export;
    -- show result
    select * from "USER_APL"."EXPORT_RECO_SQL";
END;
```

8.1.19 EXPORT_PROFITCURVES

Exports profit curve data from a trained model.

> **Note**
>
> This function does not support multinomial classification models based on the gradient boosting algorithm. It does support regression and binary classification models based on the same algorithm.

DIRECT

> **Syntax**
>
> ```
> <WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, PROFITCURVES)
> ```

PROCEDURE

> **Syntax**
>
> ```
> "SAP_PA_APL"."sap.pa.apl.base::EXPORT_PROFITCURVES"(FUNC_HEADER, MODEL,
> OPERATION_CONFIG, PROFITCURVES)
> ```

ANY PROCEDURE

> **Syntax**
>
> ```
> "_SYS_AFL"."APL_EXPORT_PROFITCURVES"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
> PROFITCURVES)
> ```

8.1.19.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The trained model to be applied
IN	OPERATION_CONFIG [page 384]	No (The table must be provided but it can be empty.)	The configuration of the export operation
OUT	PROFITCURVES [page 386]		The profit curve data

8.1.19.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Default	Description
APL/CurveType	No	• normalProfit • detected • proba • lift • userProfit • roc • sensitivity • specificity • densityG • densityB • riskDensity • riskProbability • riskOdds • riskLogOdds • regressionTargetMean	detected	The type of curve to be displayed. You can specify one type or several types by using the semi-colon separator (";"), for example: "normalProfit;detected"
APL/CurvePointCount	No	integer	20	The number of points used to display the curve
APL/CurveBasedOnFrequency	No	true/false	true	When true, indicates that the profit curves on different datasets should be synchronized on the frequency (which is the case to visualize results). When false, the synchromization is done on the categories (used to compute Ki fr example)
APL/CurveUsingWeight	No	true/false	true	When true, indicates that the profit curves on different datasets are considered using the weights. Use true when the weight is coming from a stratified sampling strategy. Use false when the weight indicates an importance of the line for the cost function.
APL/CurveUsingGroups	No	true/false	true	When true, indicates that the profit curve should be computed using the group structure found for the specified variable. Otherwise, the basic categories are used instead.

8.1.19.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, EXPORT_PROFITCURVES
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train.sql before
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
drop table EXPORT_PROFITCURVES_SIGNATURE;
create column table EXPORT_PROFITCURVES_SIGNATURE   like PROCEDURE_SIGNATURE_T;
insert into EXPORT_PROFITCURVES_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into EXPORT_PROFITCURVES_SIGNATURE values (2,
'USER_APL','MODEL_BIN_OID_T', 'IN');
insert into EXPORT_PROFITCURVES_SIGNATURE values (3,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into EXPORT_PROFITCURVES_SIGNATURE values (4, 'USER_APL','PROFITCURVE_T',
'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_EXPORT_PROFITCURVES');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','EXPORT_PROFITCURVES','USER_APL',
 'APLWRAPPER_EXPORT_PROFITCURVES', EXPORT_PROFITCURVES_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
drop table EXPORT_CONFIG;
create table EXPORT_CONFIG like OPERATION_CONFIG_T;
insert into EXPORT_CONFIG values ('APL/CurveType', 'proba');
insert into EXPORT_CONFIG values ('APL/CurvePointCount', '10');
insert into EXPORT_CONFIG values ('APL/CurveBasedOnFrequency', 'true');
insert into EXPORT_CONFIG values ('APL/CurveUsingWeight', 'true');
insert into EXPORT_CONFIG values ('APL/CurveUsingGroups', 'true');
drop table PROFITCURVES_OUT;
create table PROFITCURVES_OUT like PROFITCURVE_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header           = select * from FUNC_HEADER;
    model_in         = select * from MODEL_TRAIN_BIN;
    export_config  = select * from EXPORT_CONFIG;
    APLWRAPPER_EXPORT_PROFITCURVES(:header, :model_in, :export_config,
out_export);
    -- store result into table
    insert into  "USER_APL"."PROFITCURVES_OUT" select * from :out_export;
    -- show result
    select * from "USER_APL"."PROFITCURVES_OUT";
END;
```

8.1.19.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, EXPORT_PROFITCURVES, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train_ex.sql
before
connect USER_APL password Password1;
-- ----------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ----------------------------------------------------------------------
drop table USER_APL.FUNC_HEADER;
create table USER_APL.FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into USER_APL.FUNC_HEADER values ('Oid', '#69');
drop table USER_APL.EXPORTCURVE_CONFIG;
create table USER_APL.EXPORTCURVE_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into USER_APL.EXPORTCURVE_CONFIG values ('APL/ModelName', 'Train
Model',null);
insert into USER_APL.EXPORTCURVE_CONFIG values ('APL/ModelComment', 'Published
from APL',null);
insert into USER_APL.EXPORTCURVE_CONFIG values ('APL/ModelSpaceName',
'"MODELS"',null);
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config   = select * from EXPORTCURVE_CONFIG;

"SAP_PA_APL"."sap.pa.apl.base::EXPORT_PROFITCURVES"(:header, :model_in, :config,
out_curves);

    -- show result
    select * from :out_curves;
END;
```

8.1.20 EXPORT_VARIABLEDESCRIPTIONS

Gets the variable descriptions from the specified model.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, VARIABLE_DESC)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::EXPORT_VARIABLEDESCRIPTIONS"(FUNC_HEADER, MODEL,
VARIABLE_DESC)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_EXPORT_VARIABLEDESCRIPTIONS"(FUNC_HEADER, MODEL, VARIABLE_DESC)
```

8.1.20.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The parameters of the model whose variable descriptions you want to read
OUT	VARIABLE_DESC [page 396]		The variable descriptions from the model

8.1.20.2 Example

```
-- ================================================================
-- APL_AREA, EXPORT_VARIABLEDESCRIPTIONS
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid model (created by APL) in the MODEL_BIN table.
--               For instance, apl_createmodel.sql before
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table EXPORT_VARIABLEDESCRIPTIONS_SIGNATURE;
create column table EXPORT_VARIABLEDESCRIPTIONS_SIGNATURE  like
PROCEDURE_SIGNATURE_T;
insert into EXPORT_VARIABLEDESCRIPTIONS_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into EXPORT_VARIABLEDESCRIPTIONS_SIGNATURE values (2,
'USER_APL','MODEL_BIN_OID_T', 'IN');
insert into EXPORT_VARIABLEDESCRIPTIONS_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_OID_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_EXPORT_VARIABLEDESCRIPT
IONS');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','EXPORT_VARIABLEDESCRIPTIONS','US
ER_APL', 'APLWRAPPER_EXPORT_VARIABLEDESCRIPTIONS',
EXPORT_VARIABLEDESCRIPTIONS_SIGNATURE);
```

```
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table VARIABLE_DESC_OUT;
create table VARIABLE_DESC_OUT like VARIABLE_DESC_OID_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_BIN;
    APLWRAPPER_EXPORT_VARIABLEDESCRIPTIONS(:header, :model_in, out_var_desc);

    -- store result into table
    insert into  "USER_APL"."VARIABLE_DESC_OUT" select * from :out_var_desc;
    -- show result
    select * from "USER_APL"."VARIABLE_DESC_OUT";
END;
```

8.1.21 GET_MODEL_DATASET_TYPES

Gets the table type of the input dataset used to create or train the model.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, OUTPUT_TABLE_TYPE)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_DATASET_TYPES"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, OUTPUT_TABLE_TYPE)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_GET_MODEL_DATASET_TYPES"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
OUTPUT_TABLE_TYPE)
```

8.1.21.1 Input/Output Tables

Direction	Type	Required	Descripton
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The specified or trained model.
IN	OPERATION_CONFIG [page 384]	No (The table must be provided but it can be empty.)	The configuration of the operation.
OUT	OUTPUT_TABLE_TYPE [page 385]		The signature of the input dataset used for the model training. The signature uses the same format as the one used by GET_TABLE_TYPE_FOR_APPLY [page 213].

8.1.21.2 Example: PROCEDURE

```
-- ====================================================================
-- APL_AREA, GET_MODEL_DATASET_TYPES, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train_ex.sql)
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table OPERATION_CONFIG;
create table OPERATION_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
drop table TRAINING_SIGNATURE;
create table TRAINING_SIGNATURE like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.TABLE_TYPE";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config    = select * from OPERATION_CONFIG;
```

```
"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_DATASET_TYPES"(:header, :model_in, :conf
ig, out_schema);

    -- store result into table
    insert into  "USER_APL"."TRAINING_SIGNATURE" select * from :out_schema;
    -- show result
    select * from "USER_APL"."TRAINING_SIGNATURE";
END;
```

8.1.22 GET_MODEL_DEBRIEF

Generates debriefing tables from an Automated Analytics or SAP HANA APL model.

Standard debriefing reports can then be generated from these tables using the report functions `sap.pa.apl.debrief.report:<report_name>`.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, DEBRIEF_METRIC, DEBRIEF_PROPERTY,
SUMMARY)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_DEBRIEF"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, DEBRIEF_METRIC, DEBRIEF_PROPERTY, SUMMARY)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_GET_MODEL_DEBRIEF"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
DEBRIEF_METRIC, DEBRIEF_PROPERTY, SUMMARY)
```

8.1.22.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The model to debrief
IN	OPERATION_CONFIG [page 384]	Yes	The configuration of the debrief
OUT	DEBRIEF_PROPERTY [page 365]	Yes	The debrief properties
OUT	DEBRIEF_METRIC [page 364]	Yes	The debrief metrics
OUT	SUMMARY [page 388]	Yes	The training summary

> **Note**
>
> DEBRIEF_PROPERTY and DEBRIEF_METRICS are debrief tables with a private format. Debrief tables are typically transient and must only be used together with the provided reporting functions (see Statistical Reports [page 414]).

8.1.22.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
APL/DebriefVersion	No	A version string like '1.2.2.1'	Explicit version of the schema used by the debrief tables. If not specified, the last version of the debrief format will be used. **Caution** This parameter should be only be provided in cases where advanced usage of debrief tables is required.

Key	Required	Supported Values	Description
`APL/SegmentColumnName`	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

8.1.22.3 Example: PROCEDURE

```
-- ================================================================
-- This scripts demonstrates how to create and train a Gradient
-- Boosting model for a binary classification task
connect USER_APL password Password1;

DO BEGIN
    declare header "SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";

    declare config "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDE
D";
    declare debrief_config "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG
_EXTENDED";
    declare var_desc "SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";

    declare var_role "SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_CO
MPOSITES_OID";

    :header.insert(('Oid', '#42'));
    :header.insert(('LogLevel', '8'));
    :header.insert(('ModelFormat', 'bin'));

    :config.insert(('APL/ModelType', 'regression',null));
    :config.insert(('APL/NumberOfJobs', '4',null));
    :config.insert(('APL/MaxIterations', '200',null));
    :config.insert(('APL/MaxDepth', '4',null));
    :config.insert(('APL/EarlyStoppingPatience', '20',null));
    :config.insert(('APL/EvalMetric', 'RMSE',null));

    :var_role.insert(('age', 'target', null, null, null));
-- create and train a model'
    "SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN"(:header, :config, :var
_desc,:var_role,  'APL_SAMPLES','ADULT01',out_model,out_log,out_sum,out_indic);

-- Generate debrief tables
    "SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_DEBRIEF"(:header, :out_model, :debri
ef_config, out_debrief_metric,out_debrief_property, out_summ);
-- Use Debrief tables
    -- Dump Report about continuous variables and their statistics
    select  * from "SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_Continuous
Variables"(:out_debrief_property, :out_debrief_metric);
END;
```

Related Information

Statistical Reports [page 414]

8.1.23 GET_MODEL_INFO

Gets information from an Automated Analytics or SAP HANA APL model.

You can call this function to retrieve the variable indicators for all datasets of a serialized and trained model (training, validation, and testing datasets). To do this, use the `INDICATORS` table type with the `DATASET` extra column.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, SUMMARY, VARIABLE_ROLES,
VARIABLE_DESC, INDICATORS, PROFITCURVES, OUTPUT_TABLE_TYPE)
```

Note

`OUTPUT_TABLE_TYPE` OUT parameter is optional.

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_INFO"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
SUMMARY, VARIABLE_ROLES, VARIABLE_DESC, INDICATORS, PROFITCURVES)
```

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_INFO_AND_SIGNATURE"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, SUMMARY, VARIABLE_ROLES, VARIABLE_DESCS, INDICATORS, PROFITCURVES,
OUTPUT_TABLE_TYPE)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_GET_MODEL_INFO"(FUNC_HEADER, MODEL, OPERATION_CONFIG, SUMMARY,
VARIABLE_ROLES, VARIABLE_DESC, INDICATORS, PROFITCURVES)
```

Syntax

```
"_SYS_AFL"."APL_GET_MODEL_INFO__OVERLOAD_3_6"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
SUMMARY, VARIABLE_ROLES, VARIABLE_DESC, INDICATORS, PROFITCURVES, OUTPUT_TABLE_TYPE)
```

Related Information

INDICATORS [page 370]

8.1.23.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL` [page 382]	Yes	The model to be read
IN	`OPERATION_CONFIG` [page 384]	No (The table must be provided but it can be empty.)	The configuration of the operation
OUT	`SUMMARY` [page 388]		The training summary
OUT	`VARIABLE_ROLES` [page 398]		The roles of the variables for this training operation
OUT	`VARIABLE_DESC` [page 396]		The variable descriptions from the model
OUT	`INDICATORS` [page 370]		The requested variable indicators. You can use two table structures. Is empty if the model has not been trained.
OUT	`PROFITCURVES` [page 386]		The profit curve data. Is empty if the model has not been trained.
OUT	`OUTPUT_TABLE_TYPE` [page 385]		The signature of the input dataset used for the model training. The signature uses the same format as the one used by GET_TABLE_TYPE_FOR_APPLY [page 213].

8.1.23.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
`APL/IndicatorDataset`	No	• `'Estimation'` (default) • `'Validation'` • `'Test'`	The dataset to be used to retrieve the statistics **Note** You need an INDICATORS [page 370] table as output to retrieve the statistics.

Key	Required	Supported Values	Description
APL/SegmentColumnName	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

8.1.23.3 Example: DIRECT

```
-- ====================================================================
-- APL_AREA, GET_MODEL_INFO, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train.sql)
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table GET_MODEL_INFO_SIGNATURE;
create column table GET_MODEL_INFO_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into GET_MODEL_INFO_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into GET_MODEL_INFO_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',
'IN');
insert into GET_MODEL_INFO_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into GET_MODEL_INFO_SIGNATURE values (4, 'USER_APL','SUMMARY_T', 'OUT');
insert into GET_MODEL_INFO_SIGNATURE values (5,
'USER_APL','VARIABLE_ROLES_WITH_COMPOSITES_OID_T', 'OUT');
insert into GET_MODEL_INFO_SIGNATURE values (6,
'USER_APL','VARIABLE_DESC_OID_T', 'OUT');
insert into GET_MODEL_INFO_SIGNATURE values (7, 'USER_APL','INDICATORS_T',
'OUT');
insert into GET_MODEL_INFO_SIGNATURE values (8, 'USER_APL','PROFITCURVE_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_GET_MODEL_INFO');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','GET_MODEL_INFO','USER_APL',
'APLWRAPPER_GET_MODEL_INFO', GET_MODEL_INFO_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table OPERATION_CONFIG;
create table OPERATION_CONFIG like OPERATION_CONFIG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_OID_T;
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_WITH_COMPOSITES_OID_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
drop table PROFITCURVES;
```

```
create table PROFITCURVES like PROFITCURVE_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config    = select * from OPERATION_CONFIG;
    APLWRAPPER_GET_MODEL_INFO(:header, :model_in, :config, out_summary,
out_variable_roles, out_variable_desc, out_indicators, out_profitcurves);

    -- store result into table
    insert into  "USER_APL"."SUMMARY" select * from :out_summary;
    insert into  "USER_APL"."VARIABLE_ROLES" select * from :out_variable_roles;
    insert into  "USER_APL"."VARIABLE_DESC" select * from :out_variable_desc;
    insert into  "USER_APL"."INDICATORS" select * from :out_indicators;
        insert into  "USER_APL"."PROFITCURVES" select * from :out_profitcurves;
    -- show result
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."VARIABLE_ROLES";
    select * from "USER_APL"."VARIABLE_DESC";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."PROFITCURVES";
    select * from "USER_APL"."INDICATORS" where "KEY" in  ('L1','L2','Linf',
'ErrorMean', 'ErrorStdDev', 'R2' ,  'ClassificationRate' );
END;
```

8.1.23.4 Example: DIRECT (with Multiple Datasets)

```
-- ============================================================
-- APL_AREA, GET_MODEL_INFO, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train.sql)
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table GET_MODEL_INFO_SIGNATURE;
create column table GET_MODEL_INFO_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into GET_MODEL_INFO_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into GET_MODEL_INFO_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',
'IN');
insert into GET_MODEL_INFO_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into GET_MODEL_INFO_SIGNATURE values (4, 'USER_APL','SUMMARY_T', 'OUT');
insert into GET_MODEL_INFO_SIGNATURE values (5,
'USER_APL','VARIABLE_ROLES_WITH_COMPOSITES_OID_T', 'OUT');
insert into GET_MODEL_INFO_SIGNATURE values (6,
'USER_APL','VARIABLE_DESC_OID_T', 'OUT');
insert into GET_MODEL_INFO_SIGNATURE values (7,
'USER_APL','INDICATORS_DATASET_T', 'OUT');
insert into GET_MODEL_INFO_SIGNATURE values (8, 'USER_APL','PROFITCURVE_T',
'OUT');
insert into GET_MODEL_INFO_SIGNATURE values (9, 'USER_APL','TABLE_TYPE_T',
'OUT');
```

```
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_GET_MODEL_INFO');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','GET_MODEL_INFO','USER_APL',
'APLWRAPPER_GET_MODEL_INFO', GET_MODEL_INFO_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table OPERATION_CONFIG;
create table OPERATION_CONFIG like OPERATION_CONFIG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_OID_T;
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_WITH_COMPOSITES_OID_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_DATASET_T;
drop table PROFITCURVES;
create table PROFITCURVES like PROFITCURVE_T;
drop table SCHEMA_TRAINNING;
create table SCHEMA_TRAINNING like TABLE_TYPE_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config    = select * from OPERATION_CONFIG;
    APLWRAPPER_GET_MODEL_INFO(:header, :model_in, :config, out_summary,
out_variable_roles, out_variable_desc, out_indicators,
out_profitcurves ,out_schema_trainning);

    -- store result into table
    insert into  "USER_APL"."SUMMARY" select * from :out_summary;
    insert into  "USER_APL"."VARIABLE_ROLES" select * from :out_variable_roles;
    insert into  "USER_APL"."VARIABLE_DESC" select * from :out_variable_desc;
    insert into  "USER_APL"."INDICATORS" select * from :out_indicators;
    insert into  "USER_APL"."PROFITCURVES" select * from :out_profitcurves;
    insert into  "USER_APL"."SCHEMA_TRAINNING" select *
from :out_schema_trainning;
    -- show result
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."VARIABLE_ROLES";
    select * from "USER_APL"."VARIABLE_DESC";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."PROFITCURVES";
    select * from "USER_APL"."SCHEMA_TRAINNING";
    select * from "USER_APL"."INDICATORS" where "KEY" in  ('L1','L2','Linf',
'ErrorMean', 'ErrorStdDev', 'R2' ,  'ClassificationRate' );
END;
```

8.1.23.5 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, GET_VARIABLE_INDICATORS, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
```

```
-- Assumption 2: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train_ex.sql)
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table OPERATION_CONFIG;
create table OPERATION_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table PROFITCURVES;
create table PROFITCURVES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.PROFITCURVE";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config    = select * from OPERATION_CONFIG;
    "SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_INFO"(:header, :model_in, :config,
out_summary, out_variable_roles, out_variable_desc, out_indicators,
out_profitcurves);

    -- store result into table
    insert into  "USER_APL"."SUMMARY" select * from :out_summary;
    insert into  "USER_APL"."VARIABLE_ROLES" select * from :out_variable_roles;
    insert into  "USER_APL"."VARIABLE_DESC" select * from :out_variable_desc;
    insert into  "USER_APL"."INDICATORS" select * from :out_indicators;
    insert into  "USER_APL"."PROFITCURVES" select * from :out_profitcurves;
    -- show result
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."VARIABLE_ROLES";
    select * from "USER_APL"."VARIABLE_DESC";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."PROFITCURVES";
    select * from "USER_APL"."INDICATORS" where "KEY" in  ('L1','L2','Linf',
'ErrorMean', 'ErrorStdDev', 'R2' ,  'ClassificationRate' );
END;
```

8.1.23.6 Example: PROCEDURE (with Output Table Type)

```
-- ============================================================
-- APL_AREA, GET_MODEL_INFO_AND_SIGNATURE, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
```

```
-- Assumption 2: There's a valid trained model (created by APL procedure call)
in the MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train_ex.sql)
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table OPERATION_CONFIG;
create table OPERATION_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table PROFITCURVES;
create table PROFITCURVES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.PROFITCURVE";
drop table TRAINING_SIGNATURE;
create table TRAINING_SIGNATURE like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.TABLE_TYPE";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config    = select * from OPERATION_CONFIG;

"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_INFO_AND_SIGNATURE"(:header, :model_in,
:config, out_summary, out_variable_roles, out_variable_desc, out_indicators,
out_profitcurves, out_training_signature);

    -- store result into table
    insert into  "USER_APL"."SUMMARY"            select * from :out_summary;
    insert into  "USER_APL"."VARIABLE_ROLES"     select *
from :out_variable_roles;
    insert into  "USER_APL"."VARIABLE_DESC"      select *
from :out_variable_desc;
    insert into  "USER_APL"."INDICATORS"         select * from :out_indicators;
    insert into  "USER_APL"."PROFITCURVES"       select * from :out_profitcurves;
    insert into  "USER_APL"."TRAINING_SIGNATURE" select *
from :out_training_signature;
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."VARIABLE_ROLES";
    select * from "USER_APL"."VARIABLE_DESC";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."PROFITCURVES";
    select * from "USER_APL"."TRAINING_SIGNATURE";
END;
```

8.1.24 GET_TABLE_TYPE_FOR_APPLY

Gets the table type definition expected for the output dataset to perform the APPLY_MODEL or APPLY_MODEL_AND_TEST function.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, OUTPUT_TABLE_TYPE, LOG, SUMMARY)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::GET_TABLE_TYPE_FOR_APPLY"(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET_SCHEMA, DATASET_NAME, OUTPUT_TABLE_TYPE, LOG, SUMMARY)
```

The parameters DATASET_SCHEMA and DATASET_NAME are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_GET_TABLE_TYPE_FOR_APPLY"(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, OUTPUT_TABLE_TYPE, LOG, SUMMARY)
```

8.1.24.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The trained model to be applied
IN	OPERATION_CONFIG [page 384]	No (The table must be provided but it can be empty.)	The configuration of the apply operation. For this function, you must declare the type of Automated Analytics model and you can declare the Apply Extra Mode in the OPERATION_CONFIG table as shown in the next section.
IN	DATASET [page 362]	Yes	The input dataset

Direction	Type	Required	Description
OUT	OUTPUT_TABLE_TYPE [page 385]		The output schema to apply
OUT	LOG [page 381]		The apply log
OUT	SUMMARY [page 388]		The APPLY summary. This output is optional. The summary table is not generated if the table is not declared.

8.1.24.2 OPERATION_CONFIG Parameters

See APPLY_MODEL [page 72].

8.1.24.3 Example: DIRECT

```
-- ====================================================================
-- APL_AREA, GET_TABLE_TYPE_FOR_APPLY, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train.sql before
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CALL_SIGNATURE;
create column table CALL_SIGNATURE   like PROCEDURE_SIGNATURE_T;
insert into CALL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',  'IN');
```

```
insert into CALL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',     'IN');
insert into CALL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into CALL_SIGNATURE values (4, 'USER_APL','ADULT01_T',          'IN');
insert into CALL_SIGNATURE values (5, 'USER_APL','TABLE_TYPE_T',     'OUT');
insert into CALL_SIGNATURE values (6, 'USER_APL','OPERATION_LOG_T',    'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_GET_TABLE_TYPE_FOR_APPL
Y');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA',
'GET_TABLE_TYPE_FOR_APPLY', 'USER_APL', 'APLWRAPPER_GET_TABLE_TYPE_FOR_APPLY',
CALL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like OPERATION_CONFIG_T;
-- TODO: insert apply configuration parameters (to be defined)
drop table SCHEMA_OUT;
create table SCHEMA_OUT like TABLE_TYPE_T;
drop table SCHEMA_LOG;
create table SCHEMA_LOG like OPERATION_LOG_T;
drop table INPUT_DATA;
create table INPUT_DATA like ADULT01_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config   = select * from APPLY_CONFIG;
    dataset  = select * from INPUT_DATA;
    APLWRAPPER_GET_TABLE_TYPE_FOR_APPLY(:header, :model_in, :config, :dataset,
out_schema, out_log);

    -- store result into table
    insert into  "USER_APL"."SCHEMA_OUT" select * from :out_schema;
    insert into  "USER_APL"."SCHEMA_LOG" select * from :out_log;
    -- show result
    select * from "USER_APL"."SCHEMA_LOG";
    select * from "USER_APL"."SCHEMA_OUT";
END;
```

8.1.24.4 Example: PROCEDURE with Advanced Settings

```
-- ============================================================
-- APL_AREA, GET_TABLE_TYPE_FOR_APPLY, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train_ex.sql
before
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
```

```
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into APPLY_CONFIG values ('APL/ApplyPredictedValue', 'true',null);
insert into APPLY_CONFIG values ('APL/ApplyOutlierFlag', 'false',null);
insert into APPLY_CONFIG values ('APL/ApplyProbability', 'false',null);
insert into APPLY_CONFIG values ('APL/ApplyPredictedQuantile', '100',null);
insert into APPLY_CONFIG values ('APL/ApplyPredictedAscendingQuantile',
'100',null);
insert into APPLY_CONFIG values ('APL/
ApplyContribution','education;occupation',null);
drop table SCHEMA_OUT;
create table SCHEMA_OUT like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.TABLE_TYPE";
drop table SCHEMA_LOG;
create table SCHEMA_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config   = select * from APPLY_CONFIG;
    "SAP_PA_APL"."sap.pa.apl.base::GET_TABLE_TYPE_FOR_APPLY"
(:header, :model_in, :config, 'APL_SAMPLES','ADULT01',  out_schema, out_log);

    -- store result into table
    insert into  "USER_APL"."SCHEMA_OUT" select * from :out_schema;
    insert into  "USER_APL"."SCHEMA_LOG" select * from :out_log;
    -- show result
    select * from "USER_APL"."SCHEMA_LOG";
    select * from "USER_APL"."SCHEMA_OUT";
END;
```

8.1.25 GET_TABLE_TYPE_FOR_SOCIAL_APPLY

Gets the expected table type definition of the output dataset for an apply operation.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, MODEL, NODES1, NODES2, LINKS, COMMUNITIES,
ATTRIBUTES1, ATTRIBUTES2, DATASET, OUTPUT_TABLE_TYPE, LOG)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::GET_TABLE_TYPE_FOR_SOCIAL_APPLY"(FUNC_HEADER,
OPERATION_CONFIG, MODEL, NODES1_SCHEMA, NODES1_NAME, NODES2_SCHEMA, NODES2_NAME,
LINKS_SCHEMA, LINKS_NAME, COMMUNITIES_SCHEMA, COMMUNITIES_NAME, ATTRIBUTES1_SCHEMA,
ATTRIBUTES1_NAME, ATTRIBUTES2_SCHEMA, ATTRIBUTES2_NAME, DATASET_SCHEMA, DATASET_NAME,
OUTPUT_TABLE_TYPE, LOG)
```

The parameters NODES1_SCHEMA, NODES1_NAME, NODES2_SCHEMA, NODES2_NAME, LINKS_SCHEMA, LINKS_NAME, COMMUNITIES_SCHEMA, COMMUNITIES_NAME, ATTRIBUTES1_SCHEMA, ATTRIBUTES1_NAME, ATTRIBUTES2_SCHEMA, ATTRIBUTES2_NAME, DATASET_SCHEMA, and DATASET_NAME are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_GET_TABLE_TYPE_FOR_SOCIAL_APPLY"(FUNC_HEADER, OPERATION_CONFIG,
MODEL, NODES1, NODES2, LINKS, COMMUNITIES, ATTRIBUTES1, ATTRIBUTES2, DATASET,
OUTPUT_TABLE_TYPE, LOG)
```

8.1.25.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	OPERATION_CONFIG [page 384]	No	The configuration for the apply operation. You must also declare the Apply Mode as described in the Configuration section below.
IN	MODEL [page 382]	Yes	The trained model
IN	NODES [page 411]	No	The graph node storage (repository1)
IN	NODES [page 411]	No	The graph node storage (repository2)
IN	LINKS	No	Graph link storage
IN	COMMUNITIES	No	Communities storage
IN	ATTRIBUTES [page 405]	No	Attribute storage (repository 1)
IN	ATTRIBUTES [page 405]	No	Attribute storage (repository 2)
IN	DATASET [page 362]	Yes	
OUT	OUTPUT_TABLE_TYPE [page 385]		The description of the table type of the apply output dataset. OUTPUT_TABLE_TYPE [page 385]
OUT	LOG [page 381]		The apply log

See Social Models Serialization [page 400] for COMMUNITIES and LINKS table types.

8.1.25.2 OPERATION_CONFIG Parameters

See APPLY_MODEL [page 72].

> **Note**
>
> `APL/ApplyPredictedAscendingQuantile` alias does not work with this function.

8.1.25.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, GET_TABLE_TYPE_FOR_SOCIAL_APPLY, using a native format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid and trained model (created by APL) in the
native MODEL
--               For instance, you have used
apl_create_social_model_and_train_contact.sql before
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type APPLY_IN_CONTACT_EVENTS_T;
create type APPLY_IN_CONTACT_EVENTS_T as table (
    "CALLER" NVARCHAR(10),
    "KXCOMINDEX" INTEGER
);
-- KxNodes1
drop type MODEL_SOCIAL_NODES1_T;
create type MODEL_SOCIAL_NODES1_T as table (
    "node" NVARCHAR(10)
);
-- KxNodes2
-- not used here
drop type MODEL_SOCIAL_NODES2_T;
create type MODEL_SOCIAL_NODES2_T as table (
    "node" NVARCHAR(255)
);
-- KxLinks
drop type MODEL_SOCIAL_LINKS_T;
create type MODEL_SOCIAL_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" NVARCHAR(10),
    "KXNODEFIRST_2" NVARCHAR(10)
);
--KxCommunities1
drop type MODEL_SOCIAL_COMMUNITIES_T;
create type MODEL_SOCIAL_COMMUNITIES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "KXNODEFIRST" NVARCHAR(10),
    "COM_INDEX" INTEGER,
    "COM_INTRA" INTEGER,
    "COM_EXTRA" INTEGER
);
```

```
-- Only used for transaction graphs with projection, type is not important for
other graphs
drop type MODEL_SOCIAL_ATTRIBUTES_T;
create type MODEL_SOCIAL_ATTRIBUTES_T as table (
    "DUMMY" NVARCHAR(1)
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CALL_SIGNATURE;
create column table CALL_SIGNATURE   like PROCEDURE_SIGNATURE_T;
insert into CALL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',  'IN');
insert into CALL_SIGNATURE values (2, 'USER_APL','MODEL_NATIVE_T',     'IN');
insert into CALL_SIGNATURE values (3, 'USER_APL','MODEL_SOCIAL_NODES1_T',
'IN');
insert into CALL_SIGNATURE values (4, 'USER_APL','MODEL_SOCIAL_NODES2_T',
'IN');
insert into CALL_SIGNATURE values (5, 'USER_APL','MODEL_SOCIAL_LINKS_T',
'IN');
insert into CALL_SIGNATURE values (6,
'USER_APL','MODEL_SOCIAL_COMMUNITIES_T',     'IN');
insert into CALL_SIGNATURE values (7,
'USER_APL','MODEL_SOCIAL_ATTRIBUTES_T',     'IN');
insert into CALL_SIGNATURE values (8,
'USER_APL','MODEL_SOCIAL_ATTRIBUTES_T',     'IN');
insert into CALL_SIGNATURE values (9, 'USER_APL','OPERATION_CONFIG_DETAILED_T',
'IN');
insert into CALL_SIGNATURE values (10,
'USER_APL','APPLY_IN_CONTACT_EVENTS_T',          'IN');
insert into CALL_SIGNATURE values (11, 'USER_APL','TABLE_TYPE_T',     'OUT');
insert into CALL_SIGNATURE values (12, 'USER_APL','OPERATION_LOG_T',    'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_GET_TABLE_TYPE_FOR_SOCI
AL_APPLY');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA',
'GET_TABLE_TYPE_FOR_SOCIAL_APPLY', 'USER_APL',
'APLWRAPPER_GET_TABLE_TYPE_FOR_SOCIAL_APPLY', CALL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like OPERATION_CONFIG_DETAILED_T;
insert into APPLY_CONFIG values ('APL/ApplyMode', 'Default_Mode',null);
-- TODO: insert apply configuration parameters (to be defined)
drop table SCHEMA_OUT;
create table SCHEMA_OUT like TABLE_TYPE_T;
drop table SCHEMA_LOG;
create table SCHEMA_LOG like OPERATION_LOG_T;
drop table INPUT_DATA;
create table INPUT_DATA like APPLY_IN_CONTACT_EVENTS_T;
--insert into INPUT_DATA values('6479988873','9057318773','SMS','2007-04-05
23:13:00',2281,0);
drop view SOCIAL_MODEL_SORTED;
create view SOCIAL_MODEL_SORTED AS SELECT *  from MODEL_SOCIAL order by "ID"
asc  ;
-- select * from SOCIAL_MODEL_SORTED;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header             = select * from FUNC_HEADER;
    config             = select * from APPLY_CONFIG;
```

```
    model              = select * from SOCIAL_MODEL_SORTED;
    model_nodes1       = select * from MODEL_SOCIAL_NODES1;
    model_nodes2       = select * from MODEL_SOCIAL_NODES2;
    model_links        = select * from MODEL_SOCIAL_LINKS;
    model_communities  = select * from MODEL_SOCIAL_COMMUNITIES;
    model_attributes1  = select * from MODEL_SOCIAL_ATTRIBUTES1;
    model_attributes2  = select * from MODEL_SOCIAL_ATTRIBUTES2;
        apply_in           = select * from INPUT_DATA;

    APLWRAPPER_GET_TABLE_TYPE_FOR_SOCIAL_APPLY(
        :header,
        :model,
        :model_nodes1,
        :model_nodes2,
        :model_links,
        :model_communities,
        :model_attributes1,
        :model_attributes2,
        :config,
        :apply_in,
        out_schema,
        out_log);
    -- store result into table
    insert into  "USER_APL"."SCHEMA_OUT"  select * from :out_schema;
    insert into  "USER_APL"."SCHEMA_LOG"  select * from :out_log;
    -- show result
    select * from "USER_APL"."SCHEMA_LOG";
    select * from "USER_APL"."SCHEMA_OUT";
END;
```

8.1.25.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, GET_TABLE_TYPE_FOR_APPLY, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained model (created by APL) in the
MODEL_SOCIAL table.
--               For instance, you have used
apl_create_social_model_and_train_links_only_ex.sql before
-- ----------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type APPLY_IN_CONTACT_EVENTS_T;
create type APPLY_IN_CONTACT_EVENTS_T as table (
    "CALLER" NVARCHAR(10),
    "KXCOMINDEX" INTEGER
);
-- KxNodes1
drop type MODEL_SOCIAL_NODES1_T;
create type MODEL_SOCIAL_NODES1_T as table (
    "node" NVARCHAR(10)
);
-- KxNodes2
-- not used here
drop type MODEL_SOCIAL_NODES2_T;
create type MODEL_SOCIAL_NODES2_T as table (
        "node" NVARCHAR(255)
);
-- KxLinks
```

```
drop type MODEL_SOCIAL_LINKS_T;
create type MODEL_SOCIAL_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" NVARCHAR(10),
    "KXNODEFIRST_2" NVARCHAR(10)
);
--KxCommunities1
drop type MODEL_SOCIAL_COMMUNITIES_T;
create type MODEL_SOCIAL_COMMUNITIES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "KXNODEFIRST" NVARCHAR(10),
    "COM_INDEX" INTEGER,
    "COM_INTRA" INTEGER,
    "COM_EXTRA" INTEGER
);
drop type MODEL_SOCIAL_ATTRIBUTES_T;
create type MODEL_SOCIAL_ATTRIBUTES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "UserID" INTEGER,
    "generation_id" INTEGER,
    "SOURCE_NODES" NVARCHAR(255),  -- same type than ItemPurchased
    "HEAD_NODE_ID" INTEGER,
    "TAIL_NODE_ID" INTEGER
);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNCHEADER;
create table FUNCHEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNCHEADER values ('Oid', '#42');
insert into FUNCHEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into APPLY_CONFIG values ('APL/ApplyMode', 'Default_Mode',null);
-- TODO: insert apply configuration parameters (to be defined)
drop table SCHEMA_OUT;
create table SCHEMA_OUT like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.TABLE_TYPE";
drop table SCHEMA_LOG;
create table SCHEMA_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table INPUT_DATA;
create table INPUT_DATA like APPLY_IN_CONTACT_EVENTS_T;
--insert into INPUT_DATA values('6479988873','9057318773','SMS','2007-04-05
23:13:00',2281,0);
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header             = select * from FUNC_HEADER;
    config             = select * from APPLY_CONFIG;
    model              = select * from MODEL_SOCIAL;

    "SAP_PA_APL"."sap.pa.apl.base::GET_TABLE_TYPE_FOR_SOCIAL_APPLY"(
        :header,
        :model,
        'USER_APL','MODEL_SOCIAL_NODES1',
        'USER_APL','MODEL_SOCIAL_NODES2',
        'USER_APL','MODEL_SOCIAL_LINKS',
        'USER_APL','MODEL_SOCIAL_COMMUNITIES',
        'USER_APL','MODEL_SOCIAL_ATTRIBUTES1',
        'USER_APL','MODEL_SOCIAL_ATTRIBUTES2',
        :config,
        'USER_APL','INPUT_DATA',
        out_schema,
```

```
        out_log);
    -- store result into table
    insert into  "USER_APL"."SCHEMA_OUT"  select * from :out_schema;
    insert into  "USER_APL"."SCHEMA_LOG"  select * from :out_log;
    -- show result
    select * from "USER_APL"."SCHEMA_LOG";
    select * from "USER_APL"."SCHEMA_OUT";
END;
```

8.1.26 GET_VARIABLE_INDICATORS

Gets variable indicators from the specified trained model.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::GET_VARIABLE_INDICATORS"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, INDICATORS)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_GET_VARIABLE_INDICATORS"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
INDICATORS)
```

8.1.26.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The trained model to be read

Direction	Type	Required	Description
IN	OPERATION_CONFIG [page 384]	No (The table must be provided but it can be empty.)	The configuration of the operation
OUT	INDICATORS [page 370]		The requested variable indicators

8.1.26.2 Example: DIRECT

```
-- ====================================================================
-- APL_AREA, GET_VARIABLE_INDICATORS, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train.sql)
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table GET_VARIABLE_INDICATORS_SIGNATURE;
create column table GET_VARIABLE_INDICATORS_SIGNATURE  like
PROCEDURE_SIGNATURE_T;
insert into GET_VARIABLE_INDICATORS_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into GET_VARIABLE_INDICATORS_SIGNATURE values (2,
'USER_APL','MODEL_BIN_OID_T', 'IN');
insert into GET_VARIABLE_INDICATORS_SIGNATURE values (3,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into GET_VARIABLE_INDICATORS_SIGNATURE values (4,
'USER_APL','INDICATORS_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_GET_VARIABLE_INDICATORS
');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','GET_VARIABLE_INDICATORS','USER_A
PL', 'APLWRAPPER_GET_VARIABLE_INDICATORS', GET_VARIABLE_INDICATORS_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table OPERATION_CONFIG;
create table OPERATION_CONFIG like OPERATION_CONFIG_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
```

```
    config   = select * from OPERATION_CONFIG;
    APLWRAPPER_GET_VARIABLE_INDICATORS(:header, :model_in, :config,
out_indicators);

    -- store result into table
    insert into  "USER_APL"."INDICATORS" select * from :out_indicators;
    -- show result
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.26.3 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, GET_VARIABLE_INDICATORS, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train_ex.sql
before
connect USER_APL password Password1;
-- ----------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ----------------------------------------------------------------------
drop table USER_APL.FUNC_HEADER;
create table USER_APL.FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into USER_APL.FUNC_HEADER values ('Oid', '#69');
drop table USER_APL.INDICATORS_CONFIG;
create table USER_APL.INDICATORS_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config   = select * from INDICATORS_CONFIG;

"SAP_PA_APL"."sap.pa.apl.base::GET_VARIABLE_INDICATORS"(:header, :model_in, :conf
ig, out_indicators);

    -- store result into table
    insert into  "USER_APL"."INDICATORS" select * from :out_indicators;
    -- show result
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.27 IMPORT_MODEL

Converts an existing Predictive Analytics native model into the APL bin model format so that APL can use it. The native model is retrieved from the Predictive Analytics metadata repository.

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::IMPORT_MODEL"(FUNC_HEADER, KXADMIN, MODEL_NAME,
MODEL_VERSION, OUTPUT_MODEL)
```

> **Note**
>
> The direct technique and the ANY procedure technique do not support this function.

8.1.27.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	ADMIN [page 353]	Yes	The table in which Predictive Analytics stores the list of models
IN	String	Yes	The name of the Predictive Analytics model to import
IN	Integer	Yes	The version of the Predictive Analytics model to import
OUT	MODEL [page 382]	Yes	The table in which the APL model is saved using the APL bin model format

8.1.27.2 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, IMPORT_MODEL, using a model from Predictive Analytics
--
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: There is a valid and trained Predictive Analytics's model named
'Train Model' in table "KXADMIN"
--                 For instance, you have used apl_publish_ex.sql before
-- @depend(apl_publish_ex.sql)
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create COLUMN table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
drop table APPLY_CONFIG;
create COLUMN table APPLY_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
drop table APPLY_OUT;
DO BEGIN
    header = select * from FUNC_HEADER;
    in_kxadmin = select * from KXADMIN;
    apply_config = select * from APPLY_CONFIG;
    -- import a Predictive Analytics's model
```

```
    call "SAP_PA_APL"."sap.pa.apl.base::IMPORT_MODEL"(:header, :in_kxadmin,
'Train Model', 1, out_import_model);
    -- use it for an apply
    call
"SAP_PA_APL"."sap.pa.apl.base::GET_TABLE_TYPE_FOR_APPLY"(:header, :out_import_mod
el, :apply_config, 'APL_SAMPLES', 'CENSUS', out_schema , out_log);
    call
"SAP_PA_APL"."sap.pa.apl.base::CREATE_TABLE_FROM_TABLE_TYPE"('USER_APL',
'APPLY_OUT', :out_schema);
    call
"SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL"(:header, :out_import_model, :apply_co
nfig, 'APL_SAMPLES', 'CENSUS', 'USER_APL', 'APPLY_OUT', applyout_log,
applyout_summary);
END;
SELECT * FROM "USER_APL"."APPLY_OUT";
```

8.1.28 IMPORT_VARIABLEDESCRIPTIONS

Updates a model with the specified variable descriptions.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, VARIABLE_DESC, MODEL)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::IMPORT_VARIABLEDESCRIPTIONS"(FUNC_HEADER, MODEL,
VARIABLE_DESC, MODEL)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_IMPORT_VARIABLEDESCRIPTIONS"(FUNC_HEADER, MODEL, VARIABLE_DESC,
MODEL)
```

You already have a model table and a table of valid variable descriptions created by SAP HANA APL in the `VARIABLE_DESC` table.

8.1.28.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The parameters of the model to update
IN	VARIABLE_DESC [page 396]	Yes	The new variable descriptions
OUT	MODEL [page 382]		The updated model which now contains the updated variable descriptions

8.1.28.2 Example

```
-- ================================================================
-- APL_AREA, IMPORT_VARIABLEDESCRIPTIONS
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid model (created by APL) in the MODEL_BIN table.
--               For instance, you have used apl_createmodel.sql before
-- Assumption 4: There're valid variable descriptions (created by APL) in the
VARIABLE_DESC_OUT table.
--               For instance, you have used apl_createmodel.sql before
-- ----------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ----------------------------------------------------------------------------
connect USER_APL password Password1;
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table IMPORT_VARIABLEDESCRIPTIONS_SIGNATURE;
create column table IMPORT_VARIABLEDESCRIPTIONS_SIGNATURE  like
PROCEDURE_SIGNATURE_T;
insert into IMPORT_VARIABLEDESCRIPTIONS_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into IMPORT_VARIABLEDESCRIPTIONS_SIGNATURE values (2,
'USER_APL','MODEL_BIN_OID_T', 'IN');
insert into IMPORT_VARIABLEDESCRIPTIONS_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_T', 'IN');
insert into IMPORT_VARIABLEDESCRIPTIONS_SIGNATURE values (4,
'USER_APL','MODEL_BIN_OID_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_IMPORT_VARIABLEDESCRIPT
IONS');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','IMPORT_VARIABLEDESCRIPTIONS','US
ER_APL', 'APLWRAPPER_IMPORT_VARIABLEDESCRIPTIONS',
IMPORT_VARIABLEDESCRIPTIONS_SIGNATURE);
-- ----------------------------------------------------------------------------
```

```
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
-- the input variable descriptions are going to be created from an existing
table VARIABLE_DESC_OUT, previously populated by a CREATE* APL functions
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
insert into VARIABLE_DESC (RANK, NAME, STORAGE, VALUETYPE, KEYLEVEL,
ORDERLEVEL, MISSINGSTRING, GROUPNAME, DESCRIPTION) select RANK, NAME, STORAGE,
VALUETYPE, KEYLEVEL, ORDERLEVEL, MISSINGSTRING, GROUPNAME, DESCRIPTION from
VARIABLE_DESC_OUT;
update VARIABLE_DESC set MISSINGSTRING='xxx';
update VARIABLE_DESC set DESCRIPTION='new description for age' where NAME='age';
update VARIABLE_DESC set DESCRIPTION='new description for workclass' where
NAME='workclass';
update VARIABLE_DESC set VALUETYPE='ordinal' where NAME='age';
update VARIABLE_DESC set ORDERLEVEL='1' where NAME='fnlwgt';
select * from VARIABLE_DESC;
drop table MODEL2_BIN;
create table MODEL2_BIN like MODEL_BIN_OID_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_BIN;
    var_desc = select * from VARIABLE_DESC;
    APLWRAPPER_IMPORT_VARIABLEDESCRIPTIONS(:header, :model_in, :var_desc,
out_model);

    -- store result into table
    insert into  "USER_APL"."MODEL2_BIN" select * from :out_model;
    -- show result
    select * from "USER_APL"."MODEL2_BIN";
END;
```

8.1.29 PARALLEL_APPLY_MODEL

Applies a new model to a dataset.

This function is designed for scale-out scenarios, where the processing of the input dataset can be parallelized. The parallel processing can only be enabled for the input dataset when the whole technical stack is based on SAP HANA 2.0 or upwards.

However, it is recommended that you use the APPLY_MODEL function in parallel mode instead of the PARALLEL_APPLY_MODEL function. The APPLY_MODEL function with the parallel mode activated is the preferred option for parallelizing scoring. The PARALLEL_APPLY_MODEL function should be regarded as deprecated, except in certain cases involving some specific SAP HANA on-premise configurations with multiple nodes.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, DATASET)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_PARALLEL_APPLY_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, DATASET)
```

Before applying a model, make sure that it is available by checking that `ModelAvailable` is `true` in the summary indicator of the train operation.

Note

The Stored Procedure technique does not support this function.

Related Information

APPLY_MODEL [page 72]

8.1.29.1 Input/Output Tables

Direction	Type	Required	Descripton
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL` [page 382]	Yes	The trained model to be applied. Most APL functions create, read, write, or update a predictive model. This model is specific to Automated Analytics.
IN	`OPERATION_CONFIG` [page 384]	No (The table must be provided but it can be empty.)	The configuration of the apply operation. For this function, you must declare the type of Automated Analytics model and you can declare the optional Apply Extra Mode in the `OPERATION_CONFIG` table as described in the next section.
IN	`DATASET` [page 362]	Yes	Your input dataset

Direction	Type	Required	Descripton
OUT	DATASET [page 362]		Your output dataset. The structure of this output dataset depends on the selected targets and selected extra-mode values (if any). To get the table type of the output dataset, you can : • Infer the table type, with simple rules • Use the GET_TABLE_TYPE_FOR_APPLY function

8.1.29.2 Example

```
-- ================================================================
-- APL_AREA, PARALLEL APPLY_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train.sql)
-- Assumption 4: You are using HANA 2.0+ and running APL built with AFL SDK 2.0
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- Ouput table type: dataset
drop type ADULT01_T_OUT;
create type ADULT01_T_OUT as table (
    "KxIndex" INTEGER,
    "class" INTEGER,
    "rr_class" DOUBLE
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table APPLY_MODEL_SIGNATURE;
```

```
create column table APPLY_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into APPLY_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (4, 'USER_APL','ADULT01_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (5, 'USER_APL','ADULT01_T_OUT',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_APPLY_MODEL');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','PARALLEL_APPLY_MODEL','USER_APL'
, 'APLWRAPPER_APPLY_MODEL', APPLY_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like OPERATION_CONFIG_T;
-- TODO: insert training configuration parameters (to be defined)
drop table ADULT01_APPLY;
create column table ADULT01_APPLY like ADULT01_T_OUT;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header       = select * from FUNC_HEADER;
    model_in     = select * from MODEL_TRAIN_BIN;
    apply_config = select * from APPLY_CONFIG;
    dataset      = select * from APL_SAMPLES.ADULT01;
    call APLWRAPPER_APPLY_MODEL(:header, :model_in, :apply_config,  :dataset,
out_apply) WITH HINT(PARALLEL_BY_PARAMETER_PARTITIONS(p4));
    -- store result into table
    insert into  "USER_APL"."ADULT01_APPLY"   select * from :out_apply;
    -- show result
    select * from "USER_APL"."ADULT01_APPLY";
END;
```

8.1.30 PARALLEL_APPLY_RECO_MODEL

Applies a recommendation model to the dataset.

This function is designed for Scaled-Out scenarios, where the processing of the input dataset can be parallelized. The parallel processing can only be enabled for the input dataset when the whole technical stack is based on SAP HANA 2.0 or upwards.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, NODES1, NODES2,LINKS, OPERATION_CONFIG, DATASET,
RECO_CODE)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_PARALLEL_APPLY_RECO_MODEL"(FUNC_HEADER, MODEL, NODES1, NODES2,LINKS,
OPERATION_CONFIG, DATASET, RECO_CODE)
```

Note

The Stored Procedure technique does not support this function.

Related Information

APPLY_RECO_MODEL [page 108]

8.1.30.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The model
IN	NODES [page 411]	Yes	Graph node storage (repository 1)
IN	NODES [page 411]	Yes	Graph node storage (repository 2)
IN	LINKS [page 410]	Yes	Graph link storage
IN	OPERATION_CONFIG [page 384]	No	The configuration of the training operation
IN	DATASET [page 362]	Yes	The name of your training dataset
OUT	RECO_SCORE [page 411]		The resulting recommendation score

8.1.30.2 Example

```
-- ============================================================
-- APL_AREA, APPLY_RECO_MODEL, using a native format for the model
```

```
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_SORTED, MODEL_NODES1, MODEL_NODES2, MODEL_LINKS tables.
--               For instance you have used apl_create_reco_model_and_train.sql
before.
-- Assumption 4: You are using HANA 2.0+ and running APL built with AFL SDK 2.0
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- KxNodes1
drop type MODEL_RECO_NODES1_T;
create type MODEL_RECO_NODES1_T as table (
    "node" INT -- must be of the same SQL type as the User column (UserID here)
);
-- KxNodes2
drop type MODEL_RECO_NODES2_T;
create type MODEL_RECO_NODES2_T as table (
    "node" NVARCHAR(255) -- must be of the same SQL type as the Item column
(ItemPurchased here)
);
-- KxLinks
drop type MODEL_RECO_LINKS_T;
create type MODEL_RECO_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" INT, -- must be of the same SQL type as the User column
(UserID here)
    "KXNODESECOND" NVARCHAR(255), -- must be of the same SQL type as the Item
column (ItemPurchased here)
    "KXNODESECOND_2" NVARCHAR(255) -- must be of the same SQL type as the Item
column (ItemPurchased here)
);
drop type RECO_SCORE_T;
create type RECO_SCORE_T as table (
    "UserID" INTEGER,
    "ItemPurchased" NVARCHAR(128),
    "sn_rec_rule_id" INTEGER,
    "sn_rec_kxReco" NVARCHAR(128),
    "sn_rec_source"  NVARCHAR(128),
    "sn_rec_score" DOUBLE
);
drop type USER_T;
create type USER_T as table (
    "UserID" INTEGER -- must be of the same SQL type as the User column (UserID
here)
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table APPLY_MODEL_SIGNATURE;
create column table APPLY_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into APPLY_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_NATIVE_T', 'IN');
insert into APPLY_MODEL_SIGNATURE values (3, 'USER_APL','MODEL_RECO_NODES1_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (4, 'USER_APL','MODEL_RECO_NODES2_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (5, 'USER_APL','MODEL_RECO_LINKS_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (6, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into APPLY_MODEL_SIGNATURE values (7, 'USER_APL','USER_T',          'IN');
```

```
insert into APPLY_MODEL_SIGNATURE values (8, 'USER_APL','RECO_SCORE_T',
'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_APPLY_RECO_MODEL');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','PARALLEL_APPLY_RECO_MODEL','USER
_APL', 'APLWRAPPER_APPLY_RECO_MODEL', APPLY_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like OPERATION_CONFIG_T;
-- TODO: insert training configuration parameters (to be defined)
insert into APPLY_CONFIG values ('APL/Top', '5');                        --
optional (default: max)
insert into APPLY_CONFIG values ('APL/IncludeBestSellers', 'false');      --
optional (default: false)
insert into APPLY_CONFIG values ('APL/SkipAlreadyOwned', 'true');       --
optional (default: true)
drop table USER_APPLYIN;
create column table USER_APPLYIN like USER_T;
insert into USER_APPLYIN values ('23');
drop table USER_APPLYOUT;
create column table USER_APPLYOUT like RECO_SCORE_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
drop view MODEL_SORTED;
create view MODEL_SORTED AS SELECT *  from MODEL_RECO order by "ID" asc  ;
DO BEGIN
    header       = select * from FUNC_HEADER;
    model        = select * from MODEL_SORTED;
    model_nodes1 = select * from MODEL_NODES1;
    model_nodes2 = select * from MODEL_NODES2;
    model_links  = select * from MODEL_LINKS;
    config       = select * from APPLY_CONFIG;
    apply_in     = select * from USER_APPLYIN;
    call
APLWRAPPER_APPLY_RECO_MODEL(:header, :model, :model_nodes1, :model_nodes2, :model
_links, :config, :apply_in, out_apply) WITH
HINT(PARALLEL_BY_PARAMETER_PARTITIONS(p7));
    -- store result into table
    insert into  "USER_APL"."USER_APPLYOUT"   select * from :out_apply;
    -- show result
    select * from "USER_APL"."USER_APPLYOUT";
END;
```

8.1.31 PARALLEL_APPLY_SOCIAL_MODEL

Applies a social model on a new dataset.

This function is designed for Scaled-Out scenarios, where the processing of the input dataset can be parallelized. The parallel processing can only be enabled for the input dataset when the whole technical stack is based on SAP HANA 2.0 or upwards.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, NODES1, NODES2, LINKS, COMMUNITIES, ATTRIBUTES1,
ATTRIBUTES2, OPERATION_CONFIG, DATASET, DATASET)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_PARALLEL_APPLY_SOCIAL_MODEL"(FUNC_HEADER, MODEL, NODES1, NODES2,
LINKS, COMMUNITIES, ATTRIBUTES1, ATTRIBUTES2, OPERATION_CONFIG, DATASET, DATASET)
```

Note

The Stored Procedure technique does not support this function.

Related Information

GET_TABLE_TYPE_FOR_SOCIAL_APPLY [page 216]

8.1.31.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The model to be applied
IN	NODES [page 411]	No	The graph node storage (repository 1)
IN	NODES [page 411]	No	The graph node storage (repository 2)
IN	LINKS	No	Graph link storage
IN	COMMUNITIES	No	Communities storage
IN	ATTRIBUTES [page 405]	No	Attributes storage (repository 1)
IN	ATTRIBUTES [page 405]	No	Attributes storage (repository 2)

Direction	Type	Required	Description
IN	OPERATION_CONFIG [page 384]	No	The configuration of the apply operation
IN	DATASET [page 362]	Yes	The input dataset The input dataset must contain the following variables: • One variable for each population • The KXCOMINDEX variable, which contains communities identifiers. It should be a nominal integer.
OUT	DATASET		The output dataset. The structure of the output dataset depends on the selected targets and selected mode values (if any). To get the table type of the output dataset, you can: • Infer the table type with simple rules • Use the GET_TABLE_TYPE_FOR_SOCIAL_APPLY function

See Social Models Serialization [page 400] for COMMUNITIES and LINKS table types.

8.1.31.2 Example

```
-- ================================================================
-- APL_AREA, APPLY_SOCIAL_MODEL, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid and trained model (created by APL) in the
MODMOEL_SOCIAL table.
--               For instance, you have used
apl_create_social_model_and_train_links_only.sql before
-- ----------------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type APPLY_IN_CONTACT_EVENTS_T;
create type APPLY_IN_CONTACT_EVENTS_T as table (
    "CALLER" NVARCHAR(10),
    "KXCOMINDEX" INTEGER
);
-- KxNodes1
-- KxNodes1
drop type MODEL_SOCIAL_NODES1_T;
create type MODEL_SOCIAL_NODES1_T as table (
    "node" NVARCHAR(10)
);
-- KxNodes2
```

```
-- not used here
drop type MODEL_SOCIAL_NODES2_T;
create type MODEL_SOCIAL_NODES2_T as table (
    "node" NVARCHAR(255)
);
-- KxLinks
drop type MODEL_SOCIAL_LINKS_T;
create type MODEL_SOCIAL_LINKS_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "WEIGHT" DOUBLE,
    "KXNODEFIRST" NVARCHAR(10),
    "KXNODEFIRST_2" NVARCHAR(10)
);
--KxCommunities1
drop type MODEL_SOCIAL_COMMUNITIES_T;
create type MODEL_SOCIAL_COMMUNITIES_T as table (
    "GRAPH_NAME" NVARCHAR(255),
    "KXNODEFIRST" NVARCHAR(10),
    "COM_INDEX" INTEGER,
    "COM_INTRA" INTEGER,
    "COM_EXTRA" INTEGER
);
-- Only used for transaction graphs with projection, type is not important for
other graphs
drop type MODEL_SOCIAL_ATTRIBUTES_T;
create type MODEL_SOCIAL_ATTRIBUTES_T as table (
    "DUMMY" NVARCHAR(1)
);
drop type SOCIAL_SCORE_T;
create type SOCIAL_SCORE_T as table (
    "CALLER" NVARCHAR(10),
    "kxComIndex" INT,
    "sn_graph01_cm_idx" INT,
    "sn_graph01_cm_idx_1" INT,
    "sn_graph01_cm_idx_2" INT,
    "sn_graph01_cm_idx_3" INT,
    "sn_graph01_cm_c_off" INT,
    "sn_graph01_cm_c_off_1" INT,
    "sn_graph01_cm_c_off_2" INT,
    "sn_graph01_cm_c_off_3" INT,
    "sn_graph01_cm_r_off" DOUBLE,
    "sn_graph01_cm_r_off_1" DOUBLE,
    "sn_graph01_cm_r_off_2" DOUBLE,
    "sn_graph01_cm_r_off_3" DOUBLE,
    "sn_graph01_cm_rl" NVARCHAR(5000),
    "sn_graph01_cm_sz" INT,"sn_graph01_cm_sz_1" INT,
    "sn_graph01_cm_sz_2" INT,"sn_graph01_cm_sz_3" INT,
    "sn_graph01_i_dg" INT,"sn_graph01_o_dg" INT,
    "sn_graph01_i_w_dg" DOUBLE,"sn_graph01_o_w_dg" DOUBLE,
    "sn_graph01_sg" NVARCHAR(5000),"sn_graph01_i_c_off" DOUBLE,
    "sn_graph01_o_c_off" DOUBLE,"sn_graph01_i_r_off" DOUBLE,
    "sn_graph01_o_r_off" DOUBLE,"sn_graph01_tc" INT);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CALL_SIGNATURE;
create column table CALL_SIGNATURE   like PROCEDURE_SIGNATURE_T;
insert into CALL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',  'IN');
insert into CALL_SIGNATURE values (2, 'USER_APL','MODEL_NATIVE_T',     'IN');
insert into CALL_SIGNATURE values (3, 'USER_APL','MODEL_SOCIAL_NODES1_T',
'IN');
insert into CALL_SIGNATURE values (4, 'USER_APL','MODEL_SOCIAL_NODES2_T',
'IN');
insert into CALL_SIGNATURE values (5, 'USER_APL','MODEL_SOCIAL_LINKS_T',
'IN');
insert into CALL_SIGNATURE values (6,
'USER_APL','MODEL_SOCIAL_COMMUNITIES_T',     'IN');
```

```
insert into CALL_SIGNATURE values (7,
'USER_APL','MODEL_SOCIAL_ATTRIBUTES_T',     'IN');
insert into CALL_SIGNATURE values (8,
'USER_APL','MODEL_SOCIAL_ATTRIBUTES_T',     'IN');
insert into CALL_SIGNATURE values (9, 'USER_APL','OPERATION_CONFIG_DETAILED_T',
'IN');
insert into CALL_SIGNATURE values (10,
'USER_APL','APPLY_IN_CONTACT_EVENTS_T',          'IN');
insert into CALL_SIGNATURE values (11, 'USER_APL','SOCIAL_SCORE_T',     'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_APPLY_SOCIAL_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA',
'PARALLEL_APPLY_SOCIAL_MODEL', 'USER_APL', 'APLWRAPPER_APPLY_SOCIAL_MODEL',
CALL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table APPLY_CONFIG;
create table APPLY_CONFIG like OPERATION_CONFIG_DETAILED_T;
insert into APPLY_CONFIG values ('APL/ApplyMode', 'Default_Mode', null);
-- TODO: insert apply configuration parameters (to be defined)
drop table SCHEMA_OUT;
create table SCHEMA_OUT like TABLE_TYPE_T;
drop table INPUT_DATA;
create table INPUT_DATA like APPLY_IN_CONTACT_EVENTS_T;
insert into INPUT_DATA values('6479988873',0);
drop table APPLY_OUT;
create table APPLY_OUT like SOCIAL_SCORE_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header             = select * from FUNC_HEADER;
    config             = select * from APPLY_CONFIG;
    model              = select * from MODEL_SOCIAL order by "ID" asc  ;
    model_nodes1       = select * from MODEL_SOCIAL_NODES1;
    model_nodes2       = select * from MODEL_SOCIAL_NODES2;
    model_links        = select * from MODEL_SOCIAL_LINKS;
    model_communities  = select * from MODEL_SOCIAL_COMMUNITIES;
    model_attributes1  = select * from MODEL_SOCIAL_ATTRIBUTES1;
    model_attributes2  = select * from MODEL_SOCIAL_ATTRIBUTES2;
    apply_in           = select * from INPUT_DATA;

    call APLWRAPPER_APPLY_SOCIAL_MODEL(
        :header,
        :model,
        :model_nodes1,
        :model_nodes2,
        :model_links,
        :model_communities,
        :model_attributes1,
        :model_attributes2,
        :config,
        :apply_in,
        out_apply) WITH HINT(PARALLEL_BY_PARAMETER_PARTITIONS(p10));
    -- store result into table
    insert into  "USER_APL"."APPLY_OUT"  select * from :out_apply;
    -- show result
    select * from "USER_APL"."APPLY_OUT";
END;
```

8.1.32 PUBLISH_MODEL

Publishes a model created by SAP HANA APL into Automated Analytics compatible tables.

Use this function to publish a model that the Predictive Analytics user can load using the Automated Analytics module. Typically, in Predictive Analytics, you use Auto Analytics/Modeler to load the published model and select the ODBC data source pointing to the HANA location where the published model resides. These tables are the ones used by the Automated Analytics Modeler for its ODBC store.

> **Note**
>
> This function does not support Recommendation and Social Analysis models.

DIRECT

> **Syntax**

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, ADMIN, ADMIN, MODEL,
PUBLISHING_SUMMARY)
```

PROCEDURE

> **Syntax**

```
"SAP_PA_APL"."sap.pa.apl.base::PUBLISH_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
ADMIN, ADMIN, MODEL, PUBLISHING_SUMMARY)
```

ANY PROCEDURE

> **Syntax**

```
"_SYS_AFL"."APL_PUBLISH_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG, ADMIN, ADMIN,
MODEL, PUBLISHING_SUMMARY)
```

8.1.32.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The current model

Direction	Type	Required	Description
IN	OPERATION_CONFIG [page 384]	Yes	The configuration of the publish operation. For this function, you must declare the additional configuration parameters as shown in the next section
IN	ADMIN	Yes	The existing KXADMIN table
OUT	ADMIN		The updated KXADMIN table. The table is updated with a new entry.
OUT	MODEL [page 382]		The table which contains the published model
OUT	PUBLISHING_SUMMARY [page 387]		A summary of the publish operation

8.1.32.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
APL/ModelName	Yes		Model name for the published model
APL/ModelSpaceName	Yes		Fully qualified names of the table passed as "published model" parameter
APL/ModelComment	No		The publish comment

8.1.32.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, PUBLISH_MODEL
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train.sql before
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- Generate APLWRAPPER_PUBLISH_MODEL
drop table CALL_SIGNATURE;
create column table CALL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into CALL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into CALL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T', 'IN');
insert into CALL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into CALL_SIGNATURE values (4, 'USER_APL','ADMIN_T', 'IN');
insert into CALL_SIGNATURE values (5, 'USER_APL','ADMIN_T', 'OUT');
insert into CALL_SIGNATURE values (6, 'USER_APL','MODEL_NATIVE_T', 'OUT');
```

```
insert into CALL_SIGNATURE values (7, 'USER_APL','SUMMARY_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_PUBLISH_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','PUBLISH_MODEL','USER_APL',
'APLWRAPPER_PUBLISH_MODEL', CALL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
drop table PUBLISH_CONFIG;
create table PUBLISH_CONFIG like OPERATION_CONFIG_T;
insert into PUBLISH_CONFIG values ('APL/ModelName', 'Train Model');
insert into PUBLISH_CONFIG values ('APL/ModelComment', 'Published from APL');
insert into PUBLISH_CONFIG values ('APL/ModelSpaceName', '"MODELS"');
-- Warning: These drop/create statements are going to remove an existing and
working version of the KXADMIN table
-- In a regular use case (not this sample), the KXADMIN table already exists and
it's managed by the InfiniteInsight modeler tool.
drop table USER_APL.KXADMIN;
create table USER_APL.KXADMIN like ADMIN_T;
drop table USER_APL.MODELS;
create table USER_APL.MODELS like MODEL_NATIVE_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header           = select * from FUNC_HEADER;
    model_in         = select * from MODEL_TRAIN_BIN;
    config           = select * from PUBLISH_CONFIG;
    kxadmin          = select * from USER_APL.KXADMIN;

    APLWRAPPER_PUBLISH_MODEL(:header, :model_in, :config, :kxadmin, out_kxadmin,
out_models, out_summ);
    -- store result into table
    insert into  "USER_APL"."KXADMIN"          select * from :out_kxadmin;
    insert into  "USER_APL"."MODELS"           select * from :out_models;
    insert into  "USER_APL"."SUMMARY"          select * from :out_summ;
    -- show result
    select * from "USER_APL"."KXADMIN";
    select * from "USER_APL"."MODELS" ;
    select * from "USER_APL"."SUMMARY";
END;
```

8.1.32.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, PUBLISH_MODEL
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train_ex.sql
before
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table USER_APL.FUNC_HEADER;
```

```
create table USER_APL.FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into USER_APL.FUNC_HEADER values ('Oid', '#69');
drop table USER_APL.PUBLISH_CONFIG;
create table USER_APL.PUBLISH_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into USER_APL.PUBLISH_CONFIG values ('APL/ModelName', 'Train Model',null);
insert into USER_APL.PUBLISH_CONFIG values ('APL/ModelComment', 'Published from
APL',null);
insert into USER_APL.PUBLISH_CONFIG values ('APL/ModelSpaceName',
'"MODELS"',null);
-- Warning: These drop/create statements are going to remove an existing and
working version of the KXADMIN table
-- In a regular use case (not this sample), the KXADMIN table already exists and
it's managed by the InfiniteInsight modeler tool.
drop table USER_APL.KXADMIN;
create table USER_APL.KXADMIN like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.ADMIN";
drop table USER_APL.MODELS;
create table USER_APL.MODELS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_NATIVE";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header           = select * from FUNC_HEADER;
    model_in         = select * from MODEL_TRAIN_BIN;
    config           = select * from PUBLISH_CONFIG;
    kxadmin          = select * from USER_APL.KXADMIN;


"SAP_PA_APL"."sap.pa.apl.base::PUBLISH_MODEL"(:header, :model_in, :config, :kxadm
in, out_kxadmin, out_models, out_summ);
    -- store result into table
    insert into  "USER_APL"."KXADMIN"          select * from :out_kxadmin;
    insert into  "USER_APL"."MODELS"           select * from :out_models;
    insert into  "USER_APL"."SUMMARY"          select * from :out_summ;
    -- show result
    select * from USER_APL.KXADMIN;
    select * from USER_APL.MODELS;
    select * from SUMMARY;
END;
```

8.1.33 RETRAIN_MODEL

Retrains an existing trained model.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, MODEL, LOG, SUMMARY,
INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::RETRAIN_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
DATASET_SCHEMA, DATASET_NAME, MODEL, LOG, SUMMARY, INDICATORS)
```

The parameters DATASET_SCHEMA and DATASET_NAME are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_RETRAIN_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, MODEL,
LOG, SUMMARY, INDICATORS)
```

8.1.33.1 Input/Output Tables

Direction	Type		Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The model to be retrained
IN	OPERATION_CONFIG [page 384]	No (The table must be provided but it can be empty.)	The configuration of the training operation
IN	DATASET [page 362]	Yes	The name of your training dataset
OUT	MODEL [page 382]	Yes	The retrained model
OUT	LOG [page 381]	Yes	The training log
OUT	SUMMARY [page 388]	Yes	The training summary
OUT	INDICATORS [page 370]	Yes	The variable statistics and indicators

8.1.33.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values (declare one only)	Default	Description
`APL/VariableAutoSelection`	No	• `true` • `false`	`false`	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.
`APL/ExcludeLeakVariables`	No	• `true` • `false`	`false`	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.

8.1.33.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, RETRAIN_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid _trained_ model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train.sql before
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
```

```
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table RETRAIN_MODEL_SIGNATURE;
create column table RETRAIN_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into RETRAIN_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into RETRAIN_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',
'IN');
insert into RETRAIN_MODEL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into RETRAIN_MODEL_SIGNATURE values (4, 'USER_APL','ADULT01_T', 'IN');
insert into RETRAIN_MODEL_SIGNATURE values (5, 'USER_APL','MODEL_BIN_OID_T',
'OUT');
insert into RETRAIN_MODEL_SIGNATURE values (6, 'USER_APL','OPERATION_LOG_T',
'OUT');
insert into RETRAIN_MODEL_SIGNATURE values (7, 'USER_APL','SUMMARY_T', 'OUT');
insert into RETRAIN_MODEL_SIGNATURE values (8, 'USER_APL','INDICATORS_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_RETRAIN_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','RETRAIN_MODEL','USER_APL',
'APLWRAPPER_RETRAIN_MODEL', RETRAIN_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TRAINING_CONFIG;
create table TRAINING_CONFIG like OPERATION_CONFIG_T;
-- TODO: insert training configuration parameters (to be defined)
drop table MODEL_RETRAINED_BIN;
create table MODEL_RETRAINED_BIN like MODEL_BIN_OID_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in  = select * from MODEL_TRAIN_BIN;
    train_config   = select * from TRAINING_CONFIG;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;

    APLWRAPPER_RETRAIN_MODEL(:header, :model_in, :train_config, :dataset,
out_train_model, out_train_log, out_sum, out_indic);
    -- store result into table
    insert into  "USER_APL"."MODEL_RETRAINED_BIN"  select *
from :out_train_model;
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_train_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_RETRAINED_BIN";
    select * from "USER_APL"."OPERATION_LOG";
```

```
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.33.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, RETRAIN_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin-ex.sql).
-- Assumption 2: There's a valid _trained_ model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train_ex.sql
before
connect USER_APL password Password1;
-- ----------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ----------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TRAINING_CONFIG;
create table TRAINING_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
-- TODO: insert training configuration parameters (to be defined)
drop table MODEL_RETRAINED_BIN;
create table MODEL_RETRAINED_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
-- ----------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ----------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in  = select * from MODEL_TRAIN_BIN;
    train_config   = select * from TRAINING_CONFIG;
    var_role = select * from VARIABLE_ROLES;


"SAP_PA_APL"."sap.pa.apl.base::RETRAIN_MODEL"(:header, :model_in, :train_config,
'APL_SAMPLES','ADULT01', out_train_model, out_train_log, out_sum, out_indic);
    -- store result into table
    insert into  "USER_APL"."MODEL_RETRAINED_BIN"  select *
from :out_train_model;
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_train_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_RETRAINED_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
```

```
END;
```

8.1.34 RETRAIN_MODEL (3 Datasets)

Retrains an existing trained model on 3 datasets (training, validation, and testing).

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, TRAINING_DATASET,
VALIDATION_DATASET, TESTING_DATASET, MODEL, LOG, SUMMARY, INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::RETRAIN_MODEL_MULTI_DATASET"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, TRAINING_DATASET_SCHEMA, TRAINING_DATASET_NAME,
VALIDATION_DATASET_SCHEMA, VALIDATION_DATASET_NAME, TESTING_DATASET_SCHEMA,
TESTING_DATASET_NAME, MODEL, LOG, SUMMARY, INDICATORS)
```

The parameters `TRAINING_DATASET_SCHEMA`, `TRAINING_DATASET_NAME`, `VALIDATION_DATASET_SCHEMA`, `VALIDATION_DATASET_NAME`, `TESTING_DATASET_SCHEMA`, and `TESTING_DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_RETRAIN_MODEL__OVERLOAD_6_4"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
TRAINING_DATASET, VALIDATION_DATASET, TESTING_DATASET, MODEL, LOG, SUMMARY, INDICATORS)
```

8.1.34.1 Input/Output Tables

Direction	Type		Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL` [page 382]	Yes	The model to be retrained

Direction	Type		Description
IN	`OPERATION_CONFIG` [page 384]	No (The table must be provided but it can be empty.)	The configuration of the training operation
IN	`TRAINING_DATASET` [page 362]	Yes	The name of your training dataset
IN	`VALIDATION_DATASET` [page 362]	Yes	The name of your validation dataset
IN	`TESTING_DATASET` [page 362]	No (The table must be provided but it can be empty.)	The name of your testing dataset
OUT	`MODEL` [page 382]	Yes	The retrained model
OUT	`LOG` [page 381]	Yes	The training log
OUT	`SUMMARY` [page 388]	Yes	The training summary
OUT	`INDICATORS` [page 370]	Yes	The variable statistics and indicators

8.1.34.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values (declare one only)	Default	Description
`APL/VariableAutoSelection`	No	• `true` • `false`	`false`	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.
`APL/ExcludeLeakVariables`	No	• `true` • `false`	`false`	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.

8.1.34.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, RETRAIN_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid _trained_ model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used
apl_createmodel_and_train_custom_cutting.sql before
-- ----------------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ----------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ----------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table RETRAIN_MODEL_SIGNATURE;
create column table RETRAIN_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into RETRAIN_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into RETRAIN_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',
'IN');
insert into RETRAIN_MODEL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into RETRAIN_MODEL_SIGNATURE values (4, 'USER_APL','ADULT01_T', 'IN');
insert into RETRAIN_MODEL_SIGNATURE values (5, 'USER_APL','ADULT01_T', 'IN');
insert into RETRAIN_MODEL_SIGNATURE values (6, 'USER_APL','ADULT01_T', 'IN');
insert into RETRAIN_MODEL_SIGNATURE values (7, 'USER_APL','MODEL_BIN_OID_T',
'OUT');
insert into RETRAIN_MODEL_SIGNATURE values (8, 'USER_APL','OPERATION_LOG_T',
'OUT');
insert into RETRAIN_MODEL_SIGNATURE values (9, 'USER_APL','SUMMARY_T', 'OUT');
insert into RETRAIN_MODEL_SIGNATURE values (10, 'USER_APL','INDICATORS_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_RETRAIN_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','RETRAIN_MODEL','USER_APL',
'APLWRAPPER_RETRAIN_MODEL', RETRAIN_MODEL_SIGNATURE);
-- ----------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ----------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
```

```
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TRAINING_CONFIG;
create table TRAINING_CONFIG like OPERATION_CONFIG_T;
-- TODO: insert training configuration parameters (to be defined)
drop table MODEL_RETRAINED_BIN;
create table MODEL_RETRAINED_BIN like MODEL_BIN_OID_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
drop view  TEST;
create view  TEST as (select * from APL_SAMPLES.ADULT01);
drop view ESTIMATION;
create view ESTIMATION as (select * from APL_SAMPLES.ADULT01);
drop view  VALIDATION;
create view VALIDATION as (select * from APL_SAMPLES.ADULT01);
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in  = select * from MODEL_TRAIN_BIN;
    train_config   = select * from TRAINING_CONFIG;
    var_role = select * from VARIABLE_ROLES;
    estimation  = select * from ESTIMATION;
    validation  = select * from VALIDATION;
    test  = select * from TEST;

APLWRAPPER_RETRAIN_MODEL(:header, :model_in, :train_config, :estimation,:validati
on,:test, out_train_model, out_train_log, out_sum, out_indic);
    -- store result into table
    insert into  "USER_APL"."MODEL_RETRAINED_BIN"  select *
from :out_train_model;
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_train_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_RETRAINED_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.34.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, RETRAIN_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin-ex.sql).
-- Assumption 2: There's a valid _trained_ model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used
apl_createmodel_and_train_custom_cutting_ex.sql before
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
```

```
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TRAINING_CONFIG;
create table TRAINING_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
-- TODO: insert training configuration parameters (to be defined)
drop table MODEL_RETRAINED_BIN;
create table MODEL_RETRAINED_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop view  TEST;
create view  TEST as (select * from APL_SAMPLES.ADULT01);
drop view ESTIMATION;
create view ESTIMATION as (select * from APL_SAMPLES.ADULT01);
drop view  VALIDATION;
create view VALIDATION as (select * from APL_SAMPLES.ADULT01);
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in  = select * from MODEL_TRAIN_BIN;
    train_config   = select * from TRAINING_CONFIG;
    var_role = select * from VARIABLE_ROLES;

"SAP_PA_APL"."sap.pa.apl.base::RETRAIN_MODEL_MULTI_DATASET"(:header, :model_in, :
train_config, 'USER_APL','ESTIMATION','USER_APL','VALIDATION','USER_APL','TEST',
out_train_model, out_train_log, out_sum, out_indic);
    -- store result into table
    insert into  "USER_APL"."MODEL_RETRAINED_BIN"  select *
from :out_train_model;
    insert into  "USER_APL"."OPERATION_LOG"        select * from :out_train_log;
    insert into  "USER_APL"."SUMMARY"              select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"           select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_RETRAINED_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.35 TEST_MODEL

Tests a model against a dataset and adds related statistics to the model.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, MODEL, LOG, INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::TEST_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
DATASET_SCHEMA, DATASET_NAME, MODEL, LOG, INDICATORS)
```

The parameters DATASET_SCHEMA and DATASET_NAME are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_TEST_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG, DATASET, MODEL, LOG,
INDICATORS)
```

8.1.35.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The trained model to be tested
IN	OPERATION_CONFIG [page 384]	No (The table must be provided but it can be empty.)	The configuration of the test operation
IN	DATASET [page 362]	Yes	The input dataset
OUT	MODEL [page 382]	Yes	The updated model, with additional statistics
OUT	LOG [page 381]	Yes	The test log produced by the Automated Analytics engine
OUT	INDICATORS [page 370]	Yes	The variable statistics and indicators

8.1.35.2 OPERATION_CONFIG Parameters

Model	Key	Required	Supported Values	Description
Clustering Regression/ Classification (Legacy) Regression (Gradient Boosting) Binary Classification (Gradient Boosting) Multinomial Classification Time Series	`APL/SegmentColumnName`	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

8.1.35.3 Example: DIRECT

```
-- ====================================================================
-- APL_AREA, TEST_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train.sql)
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
```

```
drop table TEST_MODEL_SIGNATURE;
create column table TEST_MODEL_SIGNATURE like PROCEDURE_SIGNATURE_T;
insert into TEST_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into TEST_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',
'IN');
insert into TEST_MODEL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into TEST_MODEL_SIGNATURE values (4, 'USER_APL','ADULT01_T',
'IN');
insert into TEST_MODEL_SIGNATURE values (5, 'USER_APL','MODEL_BIN_OID_T',
'OUT');
insert into TEST_MODEL_SIGNATURE values (6, 'USER_APL','OPERATION_LOG_T',
'OUT');
insert into TEST_MODEL_SIGNATURE values (7, 'USER_APL','INDICATORS_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_TEST_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','TEST_MODEL', 'USER_APL',
'APLWRAPPER_TEST_MODEL', TEST_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TEST_CONFIG;
create table TEST_CONFIG like OPERATION_CONFIG_T;
drop table TEST_MODEL_BIN;
create table TEST_MODEL_BIN like MODEL_BIN_OID_T;
drop table TEST_LOG;
create column table TEST_LOG like OPERATION_LOG_T;
drop table TEST_INDICATORS;
create table TEST_INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    test_config = select * from TEST_CONFIG;
    dataset  = select * from APL_SAMPLES.ADULT01;
    APLWRAPPER_TEST_MODEL(:header, :model_in, :test_config, :dataset, out_model,
out_test_log, out_test_indic);

    -- store result into table
    insert into  "USER_APL"."TEST_MODEL_BIN"   select * from :out_model;
    insert into  "USER_APL"."TEST_LOG"         select * from :out_test_log;
    insert into  "USER_APL"."TEST_INDICATORS"  select * from :out_test_indic;
    -- show result
    select * from "USER_APL"."TEST_MODEL_BIN";
    select * from "USER_APL"."TEST_LOG";
    select * from "USER_APL"."TEST_INDICATORS";
END;
```

8.1.35.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, TEST_MODEL, using a binary format for the model
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
```

```
-- Assumption 2: The APL table types have been created (see
apl_create_table_types_ex.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train_ex.sql)
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TEST_CONFIG;
create table TEST_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
drop table TEST_MODEL_BIN;
create table TEST_MODEL_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table TEST_LOG;
create column table TEST_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table TEST_INDICATORS;
create table TEST_INDICATORS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    test_config = select * from TEST_CONFIG;

"SAP_PA_APL"."sap.pa.apl.base::TEST_MODEL"(:header, :model_in, :test_config,'APL_
SAMPLES','ADULT01', out_model, out_test_log, out_test_indic);

    -- store result into table
    insert into  "USER_APL"."TEST_MODEL_BIN"   select * from :out_model;
    insert into  "USER_APL"."TEST_LOG"         select * from :out_test_log;
    insert into  "USER_APL"."TEST_INDICATORS"  select * from :out_test_indic;
    -- show result
    select * from "USER_APL"."TEST_MODEL_BIN";
    select * from "USER_APL"."TEST_LOG";
    select * from "USER_APL"."TEST_INDICATORS";
END;
```

8.1.36 TEST_MODEL_DEBRIEF

Tests a model against a dataset. This function outputs an updated model with related statistics and its debrief tables in one call.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, TRAINED_MODEL, OPERATION_CONFIG, DATASET, UPDATED_MODEL,
LOG, SUMMARY, DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::TEST_MODEL_DEBRIEF"(FUNC_HEADER, TRAINED_MODEL,
OPERATION_CONFIG, DATASET_SCHEMA, DATASET_NAME, UPDATED_MODEL, LOG, SUMMARY,
DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

The parameters `DATASET_SCHEMA` and `DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_TEST_MODEL__OVERLOAD_4_5"(FUNC_HEADER, TRAINED_MODEL,
OPERATION_CONFIG, DATASET, UPDATED_MODEL, LOG, SUMMARY, DEBRIEF_METRIC,
DEBRIEF_PROPERTY)
```

8.1.36.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL` [page 382]	Yes	The trained model to be tested
IN	`OPERATION_CONFIG` [page 384]	No (The table must be provided but it can be empty.)	The configuration of the test operation
IN	`DATASET` [page 362]	Yes	The input dataset
OUT	`MODEL` [page 382]	Yes	The updated model
OUT	`LOG` [page 381]	Yes	The test log produced by the Automated Analytics engine
OUT	`SUMMARY` [page 388]	Yes	The test summary
OUT	DEBRIEF_METRIC [page 364]	Yes	The debrief metrics
OUT	DEBRIEF_PROPERTY [page 365]	Yes	The debrief properties

8.1.36.2 OPERATION_CONFIG Parameters

Model	Key	Required	Supported Values	Description
Clustering Regression/ Classification (Legacy) Regression (Gradient Boosting) Binary Classification (Gradient Boosting) Multinomial Classification Time Series	`APL/SegmentColumnName`	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID

8.1.36.3 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, APPLY_MODEL, using a binary format for the model
-- This script demonstrates the application of the binary classification model
-- to predict the target and to get the individual contributions per input
--
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--  @depend(segmented_create_train.sql)
connect USER_APL password Password1;
drop table TEST_SUMMARY;
create table TEST_SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table TEST_METRIC;
create table TEST_METRIC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_METRIC_OID";
drop table TEST_PROPERTY;
create table TEST_PROPERTY like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_PROPERTY_OID";
drop table TEST_MODEL_BIN;
create table TEST_MODEL_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table TEST_OPERATION_LOG;
create table TEST_OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
DO BEGIN
    declare header "SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
    declare config
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
    declare model "SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
    declare out_model "SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
    declare out_log
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
    declare out_summary  "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
    declare out_metric
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_METRIC_OID";
```

```
    declare out_property
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_PROPERTY_OID";
    model = select * from MODEL_TRAIN_BIN;
    :header.insert(('Oid', '#42'));
    :header.insert(('LogLevel', '2'));
    :header.insert(('MaxTasks', '2'));  -- define nb parallel tasks to use for
train
    :config.insert(('APL/SegmentColumnName', 'Seg',null)); -- define the column
used as the segmentation colum

"SAP_PA_APL"."sap.pa.apl.base::TEST_MODEL_DEBRIEF"(:header, :model, :config,'USER
_APL','ADULT01_SORTED', out_model, out_log, out_summary,
out_metric,out_property);
    -- store result into table
    insert into  TEST_PROPERTY       select * from :out_property;
    insert into  TEST_METRIC         select * from :out_metric;
    insert into  TEST_SUMMARY        select * from :out_summary;
    insert into  TEST_MODEL_BIN      select * from :out_model;
    insert into  TEST_OPERATION_LOG  select * from :out_log;
END;
```

8.1.37 TRAIN_MODEL

Trains your analytic model on a known dataset.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, VARIABLE_ROLES DATASET, MODEL, LOG,
SUMMARY, INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::TRAIN_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
VARIABLE_ROLES, DATASET_SCHEMA, DATASET_NAME, MODEL, LOG, SUMMARY, INDICATORS)
```

The parameters `DATASET_SCHEMA` and `DATASET_NAME` are strings.

You can have `MODEL` as a single OUT parameter. If you use the direct or stored procedure technique, simply remove `LOG`, `SUMMARY` and `INDICATORS` from the signature.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_TRAIN_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG, VARIABLE_ROLES
DATASET, MODEL, LOG, SUMMARY, INDICATORS)
```

Syntax

```
"_SYS_AFL"."APL_TRAIN_MODEL__OVERLOAD_5_1"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
VARIABLE_ROLES DATASET, MODEL)
```

8.1.37.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The model to be trained
IN	OPERATION_CONFIG [page 384]	No (The table must be provided but it can be empty.)	The configuration of the training operation. In addition, all Common APL Aliases for Model Training [page 336] are also supported.
IN	VARIABLE_ROLES [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training operation. The roles of the variables for this training. This can also be used to define composite variables.
IN	DATASET [page 362]	Yes	The name of your training dataset
OUT	MODEL [page 382]		The updated trained model
OUT	LOG [page 381]		The training log produced by the Automated Analytics engine
OUT	SUMMARY [page 388]		The training summary
OUT	INDICATORS [page 370]		The variable statistics and indicators

8.1.37.2 OPERATION_CONFIG Parameters

Key	Supported Values

`APL/CuttingStrategy`	• Mandatory for time series models, otherwise optional
	Partition strategy for the training dataset.
	`'random'`
	`'periodic'`
	`'sequential'`
	`'periodic with test at end'`
	`'random with test at end'`
	`'sequential with no test'`
	`'periodic with no test'`
	`'random with no test'` (default value for regression/classification and clustering)
	For a time series models, only use the following values:
	`'sequential'`
	`'sequential with no test'` (default value)

`APL/VariableAutoSelection`	• Optional `true` `false` (default value) Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.

`APL/ExcludeLeakVariables`	• Optional
	`true`
	`false`
	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.

`APL/CalculateSQLExpressions`	• Optional `enabled` `disabled` (default value) Generate SQL expression for clustering model

Key	Supported Values
APL/IncludeCategoryNormalProfits	[Classification/Regression models only] • Optional Calculate the normal profits per variable category and populate the indicators output table accordingly.
APL/NbClusters	Number of clusters for the model. Default value: 10
APL/NbClustersMin	Cluster count range customization, lower number of clusters. Overrides NbClusters.
APL/NbClustersMax	Cluster count range customization, upper number of clusters. Overrides NbClusters.

8.1.37.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, TRAIN_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid model (created by APL) in the MODEL_BIN table.
--               For instance, you have used apl_createmodel.sql before
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table TRAIN_MODEL_SIGNATURE;
create column table TRAIN_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into TRAIN_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into TRAIN_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T', 'IN');
```

```
insert into TRAIN_MODEL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into TRAIN_MODEL_SIGNATURE values (4, 'USER_APL','VARIABLE_ROLES_T',
'IN');
insert into TRAIN_MODEL_SIGNATURE values (5, 'USER_APL','ADULT01_T', 'IN');
insert into TRAIN_MODEL_SIGNATURE values (6, 'USER_APL','MODEL_BIN_OID_T',
'OUT');
insert into TRAIN_MODEL_SIGNATURE values (7, 'USER_APL','OPERATION_LOG_T',
'OUT');
insert into TRAIN_MODEL_SIGNATURE values (8, 'USER_APL','SUMMARY_T', 'OUT');
insert into TRAIN_MODEL_SIGNATURE values (9, 'USER_APL','INDICATORS_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_TRAIN_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','TRAIN_MODEL','USER_APL',
'APLWRAPPER_TRAIN_MODEL', TRAIN_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TRAINING_CONFIG;
create table TRAINING_CONFIG like OPERATION_CONFIG_T;
-- training configuration is optional, hence the empty table
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- variable roles are optional, hence the empty table
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like MODEL_BIN_OID_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in  = select * from MODEL_BIN;
    train_config   = select * from TRAINING_CONFIG;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;


APLWRAPPER_TRAIN_MODEL(:header, :model_in, :train_config, :var_role, :dataset,
out_train_model, out_train_log, out_sum, out_indic);
    -- store result into table
    insert into  "USER_APL"."MODEL_TRAIN_BIN"  select * from :out_train_model;
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_train_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_TRAIN_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.37.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, TRAIN_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid model (created by APL) in the MODEL_BIN table.
--               For instance, you have used apl_createmodel_ex.sql before
-- ----------------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------------
connect USER_APL password Password1;
-- ----------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ----------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TRAINING_CONFIG;
create table TRAINING_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
-- training configuration is optional, hence the empty table
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
-- variable roles are optional, hence the empty table
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
-- ----------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ----------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in  = select * from MODEL_BIN;
    train_config   = select * from TRAINING_CONFIG;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;


"SAP_PA_APL"."sap.pa.apl.base::TRAIN_MODEL"(:header, :model_in, :train_config, :v
ar_role, 'APL_SAMPLES','ADULT01', out_train_model, out_train_log, out_sum,
out_indic);
    -- store result into table
    insert into  "USER_APL"."MODEL_TRAIN_BIN"  select * from :out_train_model;
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_train_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_TRAIN_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.38 TRAIN_MODEL (3 Datasets)

Trains your analytic model on three datasets (training, validation, and testing).

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, VARIABLE_ROLES, TRAINING_DATASET,
VALIDATION_DATASET, TESTING_DATASET, MODEL, LOG, SUMMARY, INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::TRAIN_MODEL_MULTI_DATASET"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, VARIABLE_ROLES, TRAINING_DATASET_SCHEMA, TRAINING_DATASET_NAME,
VALIDATION_DATASET_SCHEMA, VALIDATION_DATASET_NAME, TESTING_DATASET_SCHEMA,
TESTING_DATASET_NAME, MODEL, LOG, SUMMARY, INDICATORS)
```

The parameters `TRAINING_DATASET_SCHEMA`, `TRAINING_DATASET_NAME`, `VALIDATION_DATASET_SCHEMA`, `VALIDATION_DATASET_NAME`, `TESTING_DATASET_SCHEMA`, and `TESTING_DATASET_NAME` are strings.

You can have `MODEL` as a single OUT parameter. If you use the direct or stored procedure technique, simply remove `LOG`, `SUMMARY` and `INDICATORS` from the signature.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_TRAIN_MODEL__OVERLOAD_7_4"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
VARIABLE_ROLES TRAINING_DATASET, VALIDATION_DATASET, TESTING_DATASET, MODEL, LOG,
SUMMARY, INDICATORS)
```

Syntax

```
"_SYS_AFL"."APL_TRAIN_MODEL__OVERLOAD_7_1"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
VARIABLE_ROLES TRAINING_DATASET, VALIDATION_DATASET, TESTING_DATASET, MODEL)
```

8.1.38.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The model to be trained
IN	OPERATION_CONFIG [page 384]	No (The table must be provided but it can be empty.)	The configuration of the training operation. In addition, all Common APL Aliases for Model Training [page 336] are also supported.
IN	VARIABLE_ROLES [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training operation. The roles of the variables for this training. This can also be used to define composite variables.
IN	TRAINING_DATASET [page 362]	Yes	The name of your training dataset
IN	VALIDATION_DATASET [page 362]	Yes	The name of your validation dataset
IN	TESTING_DATASET [page 362]	No (The table must be provided but it can be empty.)	The name of your testing dataset
OUT	MODEL [page 382]		The updated trained model
OUT	LOG [page 381]		The training log produced by the Automated Analytics engine
OUT	SUMMARY [page 388]		The training summary
OUT	INDICATORS [page 370]		The variable statistics and indicators

8.1.38.2 OPERATION_CONFIG Parameters

Key	Supported Values (declare one only)

`APL/CuttingStrategy`	• Mandatory for timeseries type models, otherwise optional
	Cutting strategy for the training dataset. `'random'` (default value for regression/classificationn and clustering)
	`'periodic'`
	`'sequential'`
	`'periodic with test at end'`
	`'random with test at end'`
	`'sequential with no test'`
	`'periodic with no test'`
	`'random with no test'`
	For a timeseries model, only use the following values (there is no default value):
	`'sequential'`
	`'sequential with no test'`

`APL/VariableAutoSelection`	• Optional `true` `false` (default value) Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.
`APL/ExcludeLeakVariables`	• Optional `true` `false` (default value) Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.
`APL/CalculateSQLExpressions`	• Optional `enabled` `disabled` (default value) Generate SQL expression for clustering model

Key	Supported Values (declare one only)
APL/IncludeCategoryNormalProfits	[Classification/Regression models only] • Optional Calculate the normal profits per variable category and populate the indicators output table accordingly.
APL/NbClusters	Number of clusters for the model. Default value: 10
APL/NbClustersMin	Cluster count range customization, lower number of clusters. Overrides NbClusters.
APL/NbClustersMax	Cluster count range customization, upper number of clusters. Overrides NbClusters.

8.1.38.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, TRAIN_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid model (created by APL) in the MODEL_BIN table.
--               For instance, you have used apl_createmodel_custom_cutting.sql
before
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table TRAIN_MODEL_SIGNATURE;
create column table TRAIN_MODEL_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into TRAIN_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
```

```
insert into TRAIN_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T', 'IN');
insert into TRAIN_MODEL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into TRAIN_MODEL_SIGNATURE values (4, 'USER_APL','VARIABLE_ROLES_T',
'IN');
insert into TRAIN_MODEL_SIGNATURE values (5, 'USER_APL','ADULT01_T', 'IN');
insert into TRAIN_MODEL_SIGNATURE values (6, 'USER_APL','ADULT01_T', 'IN');
insert into TRAIN_MODEL_SIGNATURE values (7, 'USER_APL','ADULT01_T', 'IN');
insert into TRAIN_MODEL_SIGNATURE values (8, 'USER_APL','MODEL_BIN_OID_T',
'OUT');
insert into TRAIN_MODEL_SIGNATURE values (9, 'USER_APL','OPERATION_LOG_T',
'OUT');
insert into TRAIN_MODEL_SIGNATURE values (10, 'USER_APL','SUMMARY_T', 'OUT');
insert into TRAIN_MODEL_SIGNATURE values (11, 'USER_APL','INDICATORS_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_TRAIN_MODEL');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','TRAIN_MODEL','USER_APL',
'APLWRAPPER_TRAIN_MODEL', TRAIN_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TRAINING_CONFIG;
create table TRAINING_CONFIG like OPERATION_CONFIG_T;
-- training configuration is optional, hence the empty table
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- variable roles are optional, hence the empty table
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like MODEL_BIN_OID_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
drop view  TEST;
create view  TEST as (select * from APL_SAMPLES.ADULT01);
drop view ESTIMATION;
create view ESTIMATION as (select * from APL_SAMPLES.ADULT01);
drop view  VALIDATION;
create view VALIDATION as (select * from APL_SAMPLES.ADULT01);
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in  = select * from MODEL_BIN;
    train_config   = select * from TRAINING_CONFIG;
    var_role = select * from VARIABLE_ROLES;
    estimation  = select * from ESTIMATION;
    validation  = select * from VALIDATION;
    test  = select * from TEST;


APLWRAPPER_TRAIN_MODEL(:header, :model_in, :train_config, :var_role, :estimation,
:validation,:test, out_train_model, out_train_log, out_sum, out_indic);
    -- store result into table
    insert into  "USER_APL"."MODEL_TRAIN_BIN"  select * from :out_train_model;
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_train_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_TRAIN_BIN";
```

```
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.38.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, TRAIN_MODEL, using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid model (created by APL) in the MODEL_BIN table.
--               For instance, you have used
apl_createmodel_custom_cutting_ex.sql before
-- ----------------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------------
connect USER_APL password Password1;
SET SESSION 'APL_CACHE_SCHEMA' = 'APL_CACHE';
-- ----------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ----------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table TRAINING_CONFIG;
create table TRAINING_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
-- training configuration is optional, hence the empty table
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
-- variable roles are optional, hence the empty table
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop view  TEST;
create view  TEST as (select * from APL_SAMPLES.ADULT01);
drop view ESTIMATION;
create view ESTIMATION as (select * from APL_SAMPLES.ADULT01);
drop view  VALIDATION;
create view VALIDATION as (select * from APL_SAMPLES.ADULT01);
-- ----------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ----------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in  = select * from MODEL_BIN;
    train_config   = select * from TRAINING_CONFIG;
    var_role = select * from VARIABLE_ROLES;
```

```
"SAP_PA_APL"."sap.pa.apl.base::TRAIN_MODEL_MULTI_DATASET"(:header, :model_in, :tr
ain_config, :var_role,
'USER_APL','ESTIMATION','USER_APL','VALIDATION','USER_APL','TEST',
out_train_model, out_train_log, out_sum, out_indic);
    -- store result into table
    insert into  "USER_APL"."MODEL_TRAIN_BIN"  select * from :out_train_model;
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_train_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    -- show result
    select * from "USER_APL"."MODEL_TRAIN_BIN";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.1.39 UPDATE_MODEL

Changes the model parameters based on the specified operation configuration and model format.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, MODEL)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::UPDATE_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG,
MODEL)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_UPDATE_MODEL"(FUNC_HEADER, MODEL, OPERATION_CONFIG, MODEL)
```

8.1.39.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`MODEL` [page 382]	Yes	The model to be updated
IN	`OPERATION_CONFIG` [page 384]	No (The table must be provided but it can be empty.)	The configuration of the update operation. For this function, you must declare the type of Automated Analytics model and you can declare the optional partition strategy in the `OPERATION_CONFIG` table as described in the next section.
OUT	`MODEL` [page 382]		The updated model

8.1.39.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
`APL/ForgetTraining`	No	• `'true'` • `'false'` (default value)	The trained model is set back to its `SpecifiedModel` state.
`APL/ModelName`	No	String	The new name of the model

8.1.39.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, UPDATE_MODEL
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid and published model in the MODELS table.
--               For instance, you have used apl_publish.sql before.
connect USER_APL password Password1;
-- ----------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ----------------------------------------------------------------------
```

```
drop table UPDATE_MODEL_SIGNATURE;
create column table UPDATE_MODEL_SIGNATURE like PROCEDURE_SIGNATURE_T;
insert into UPDATE_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into UPDATE_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_NATIVE_T', 'IN');
insert into UPDATE_MODEL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into UPDATE_MODEL_SIGNATURE values (4, 'USER_APL','MODEL_BIN_OID_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_UPDATE_MODEL_1');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','UPDATE_MODEL','USER_APL',
'APLWRAPPER_UPDATE_MODEL_1', UPDATE_MODEL_SIGNATURE);
delete from  UPDATE_MODEL_SIGNATURE;
insert into UPDATE_MODEL_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into UPDATE_MODEL_SIGNATURE values (2, 'USER_APL','MODEL_BIN_OID_T',
'IN');
insert into UPDATE_MODEL_SIGNATURE values (3, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into UPDATE_MODEL_SIGNATURE values (4, 'USER_APL','MODEL_BIN_OID_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_UPDATE_MODEL_2');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','UPDATE_MODEL','USER_APL',
'APLWRAPPER_UPDATE_MODEL_2', UPDATE_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table EXPORT_SQL;
create table EXPORT_SQL like RESULT_T;
drop table MODEL_UPDATED_BIN;
create table MODEL_UPDATED_BIN like MODEL_BIN_OID_T;
drop table MODEL_TRAIN_BIN;
create table MODEL_TRAIN_BIN like MODEL_BIN_OID_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
drop table UPDATE_CONFIG;
create table UPDATE_CONFIG like OPERATION_CONFIG_T;
DO BEGIN
    header           = select * from FUNC_HEADER;
    config           = select * from UPDATE_CONFIG;
    model_in         = select * from  "MODELS"  order by "ID" asc;
    APLWRAPPER_UPDATE_MODEL_1(:header, :model_in,    :config, out_model_1);
    APLWRAPPER_UPDATE_MODEL_2(:header, :out_model_1, :config, out_model_2);
    -- store result into table
    insert into  "USER_APL"."MODEL_UPDATED_BIN"   select * from :out_model_1;
    insert into  "USER_APL"."MODEL_TRAIN_BIN"     select * from :out_model_2;
    -- show result
    select * from "USER_APL"."MODEL_UPDATED_BIN";
    select * from "USER_APL"."MODEL_TRAIN_BIN";
END;
```

8.1.39.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, UPDATE_MODEL
```

```
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train_ex.sql
before
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table USER_APL.FUNC_HEADER;
create table USER_APL.FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into USER_APL.FUNC_HEADER values ('Oid', '#69');
drop table USER_APL.UPDATE_CONFIG;
create table USER_APL.UPDATE_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
drop table USER_APL.MODEL_UPDATED_BIN;
create COLUMN table USER_APL.MODEL_UPDATED_BIN like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header           = select * from FUNC_HEADER;
    config           = select * from UPDATE_CONFIG;
    model_in         = select * from MODEL_TRAIN_BIN;

"SAP_PA_APL"."sap.pa.apl.base::UPDATE_MODEL"(:header, :model_in, :config, :out_mo
del);

    -- store result into table
    insert into  "USER_APL"."MODEL_UPDATED_BIN"           select *
from :out_model;
    -- show result
    select * from "USER_APL"."MODEL_UPDATED_BIN";
END;
```

8.2 Predictive Business Functions

This group of functions comprises high-level and simpler data-mining functions that focus on the business-value of the predictive functions, while hiding the predictive model that's used behind the scenes.

8.2.1 COMPARE_DATA

Compares a given dataset to a reference dataset.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES,
REFERENCE_DATASET, NEW_DATASET, LOG, SUMMARY, DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::COMPARE_DATA"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESCS, VARIABLE_ROLES, REFERENCE_DATASET_SCHEMA, REFERENCE_DATASET_NAME,
NEW_DATASET_SCHEMA, NEW_DATASET_NAME, LOG, SUMMARY, DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

The parameters REFERENCE_DATASET_SCHEMA, REFERENCE_DATASET_NAME, NEW_DATASET_SCHEMA, and NEW_DATASET_NAME are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_COMPARE_DATA"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS,
VARIABLE_ROLES, REFERENCE_DATASET, NEW_DATASET, LOG, SUMMARY, DEBRIEF_METRIC,
DEBRIEF_PROPERTY)
```

8.2.1.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No You must provide the table, but it can be empty.	The function header can be used to set the logging level for instance.
IN	OPERATION_CONFIG [page 384]	No	The configuration of the compare operation.
IN	VARIABLE_DESC [page 396]	No You must provide the table, but it can be empty.	The variable descriptions for the input datasets.
IN	VARIABLE_ROLES [page 398]	No You must provide the table, but it can be empty.	The roles of the variables.
IN	REFERENCE_DATASET	Yes	The name of the table or view used as reference dataset.
IN	NEW_DATASET	Yes	The name of the table or view used as the new dataset that will be compared to the reference dataset.
OUT	LOG [page 381]		The log of the compare operation.
OUT	SUMMARY [page 388]		The summary of the compare operation.
OUT	DEBRIEF_METRIC [page 364]		The debrief metrics.

Direction	Type	Required	Description
OUT	DEBRIEF_PROPERTY [page 365]		The debrief properties.

8.2.1.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description	Default
APL/ModelType	No	• 'statbuilder' • 'variable-encoder'	Type of the Automated Analytics model. To compute group statistics use the model type 'variable-encoder'.	'statbuilder'
APL/CuttingStrategy	No	• 'random' • 'random with no test'	Partition strategy for the training dataset only when the model type is 'variable-encoder'. For the model type 'statbuilder', this parameter is ignored.	'random with no test'
APL/SegmentColumnName	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID.	APL/SegmentColumnName Name of the column that stores the segment ID.

8.2.1.3 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, COMPARE_DATA
-- This script compares a given dataset to a reference dataset
--
connect USER_APL password Password1;
drop view DATASET_REF;
drop view DATASET_NEW;
create view DATASET_REF as select "age","education","marital-
status","occupation","capital-gain" from APL_SAMPLES.ADULT01 where
"sex"='Male';
create view DATASET_NEW as select "age","education","marital-
status","occupation","capital-gain" from APL_SAMPLES.ADULT01 where
"sex"='Female';
DO BEGIN
    declare header "SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
    declare config
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
    declare var_desc
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
    declare var_role
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
    declare out_log   "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
```

```
    declare out_sum   "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
    declare out_debrief_metric
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_METRIC_OID";
    declare out_debrief_property
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_PROPERTY_OID";
    :header.insert(('Oid', 'Male Vs Female'));
    "SAP_PA_APL"."sap.pa.apl.base::COMPARE_DATA" (
    :header, :config, :var_desc, :var_role, 'USER_APL', 'DATASET_REF',
'USER_APL', 'DATASET_NEW',
    out_log, out_sum, out_debrief_metric, out_debrief_property );

    select * from :out_log;
    select * from :out_sum;
    select * from SAP_PA_APL."sap.pa.apl.debrief.report::Deviation_ByVariable"
(:out_debrief_property,:out_debrief_metric);
    select * from
SAP_PA_APL."sap.pa.apl.debrief.report::Deviation_CategoryFrequencies"
(:out_debrief_property,:out_debrief_metric);
END;
```

8.2.1.4 Example: DIRECT

```
-- ============================================================
-- APL_AREA, COMPARE_DATA
-- This script compares a given dataset to a reference dataset
--
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ----------------------------------------------------------------------
-- Create table type for the datasets
-- ----------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type PEOPLE_T;
create type PEOPLE_T as table (
    "age" INTEGER,
    "education" NVARCHAR(32),
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "capital-gain" INTEGER
);
-- ----------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ----------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table COMPARE_DATA_SIGNATURE;
create column table COMPARE_DATA_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into COMPARE_DATA_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into COMPARE_DATA_SIGNATURE values (2, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into COMPARE_DATA_SIGNATURE values (3, 'USER_APL','VARIABLE_DESC_T',
'IN');
insert into COMPARE_DATA_SIGNATURE values (4, 'USER_APL','VARIABLE_ROLES_T',
'IN');
insert into COMPARE_DATA_SIGNATURE values (5, 'USER_APL','PEOPLE_T', 'IN');
insert into COMPARE_DATA_SIGNATURE values (6, 'USER_APL','PEOPLE_T', 'IN');
insert into COMPARE_DATA_SIGNATURE values (7, 'USER_APL','OPERATION_LOG_T',
'OUT');
insert into COMPARE_DATA_SIGNATURE values (8, 'USER_APL','SUMMARY_T', 'OUT');
```

```
insert into COMPARE_DATA_SIGNATURE values (9, 'USER_APL','DEBRIEF_METRIC_OID_T',
'OUT');
insert into COMPARE_DATA_SIGNATURE values (10,
'USER_APL','DEBRIEF_PROPERTY_OID_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_COMPARE_DATA');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','COMPARE_DATA','USER_APL',
'APLWRAPPER_COMPARE_DATA', COMPARE_DATA_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', 'My Comparison');
drop table COMPARE_DATA_CONFIG;
create table COMPARE_DATA_CONFIG like OPERATION_CONFIG_T;
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from COMPARE_DATA_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset_ref = select "age","education","marital-
status","occupation","capital-gain" from APL_SAMPLES.ADULT01 where
"sex"='Male';
    dataset_new = select "age","education","marital-
status","occupation","capital-gain" from APL_SAMPLES.ADULT01 where
"sex"='Female';

    APLWRAPPER_COMPARE_DATA (
    :header, :config, :var_desc,:var_role, :dataset_ref, :dataset_new,
    out_log, out_sum, out_debrief_metric, out_debrief_property );

    select * from :out_log;
    select * from :out_sum;
    select * from SAP_PA_APL."sap.pa.apl.debrief.report::Deviation_ByVariable"
(:out_debrief_property,:out_debrief_metric);
    select * from
SAP_PA_APL."sap.pa.apl.debrief.report::Deviation_CategoryFrequencies"
(:out_debrief_property,:out_debrief_metric);
END;
```

8.2.2 DEBRIEF_APPLY_RESULT

Returns statistics and profit curves based on 3 datasets and obtained after the Statbuilder model is applied.

When you call this function, you actually call the following series of functions:

1. `"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN_MULTI_DATASET"`
2. `"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_INFO"` for the training dataset
3. `"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_INFO"` for the validation dataset
4. `"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_INFO"` for the testing dataset

The association of `CREATE_MODEL_AND_TRAIN` and `GET_MODEL_INFO` allows you to retrieve the complete debriefing information from the `INDICATORS`, `SUMMARY`, and `PROFITCURVES` tables:

- KPIs
- Performance curves
- Basic statistics on variable and dataset

The `INDICATORS` output table has an additional column called `DATASET` to know from which dataset the statistics are computed.

> **Note**
>
> See Model Types [page 13] for a definition of the Statbuilder model.

DIRECT

> **Syntax**
>
> `<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, DATASET, DATASET, DATASET, LOG, SUMMARY, INDICATORS, PROFITCURVES)`

PROCEDURE

> **Syntax**
>
> `"SAP_PA_APL"."sap.pa.apl.base::DEBRIEF_APPLY_RESULT"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, TRAINING_DATASET_SCHEMA, TRAINING_DATASET_NAME, VALIDATION_DATASET_SCHEMA, VALIDATION_DATASET_NAME, TESTING_DATASET_SCHEMA, TESTING_DATASET_NAME, LOG, SUMMARY, INDICATORS, PROFITCURVES)`

The parameters `TRAINING_DATASET_SCHEMA`, `TRAINING_DATASET_NAME`, `VALIDATION_DATASET_SCHEMA`, `VALIDATION_DATASET_NAME`, `TESTING_DATASET_SCHEMA`, and `TESTING_DATASET_NAME` are strings.

ANY PROCEDURE

> **Syntax**
>
> `"_SYS_AFL"."APL_DEBRIEF_APPLY_RESULT"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, DATASET, DATASET, DATASET, LOG, SUMMARY, INDICATORS, PROFITCURVES)`

Related Information

CREATE_MODEL_AND_TRAIN (3 Datasets) [page 142]
GET_MODEL_INFO [page 206]

8.2.2.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	OPERATION_CONFIG [page 384]	No	The configuration of the profiling operation
IN	VARIABLE_DESC [page 396]	No (The table must be provided but it can be empty.)	The variable descriptions for the input dataset, as expected by the AutomatedAnalytics engine. If the table is empty, then the variable descriptions are automatically guessed by the Automated Analytics engine.
IN	VARIABLE_ROLES [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables. This can also be used to define composite variables.
IN	DATASET [page 362]	Yes	The name of the training dataset
IN	DATASET [page 362]	No	The name of the validation dataset
IN	DATASET [page 362]	No	The name of the testing dataset
OUT	LOG [page 381]		The profiling log
OUT	SUMMARY [page 388]		The profiling summary, similar to a training summary
OUT	INDICATORS [page 370]		The variable statistics and indicators
OUT	PROFITCURVES [page 386]		The profit curve

8.2.2.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
APL/VariableEstimatorOf	Yes	Concatenated string with ';' for example "VariableName;TargetName"	Indicates an input variable as estimator of a target variable

In addition, all parameters described in the OPERATION_CONFIG of EXPORT_PROFITCURVES [page 196] are supported. They determine the PROFITCURVES table content.

8.2.2.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, DEBRIEF_APPLY_RESULT using a binary format for the model
-- Assumption 1: The users & privileges have been created & granted (see
APL_admin.sql).
-- Assumption 2: The APL table types have been created (see
APL_create_table_types.sql).
-- Assumption 3: There's a valid apply for a model (created by APL) IN the
ADULT01_APPLY table.
-- @depend(apl_apply_model.sql)
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
drop type ADULT01_T_OUT;
create type ADULT01_T_OUT as table (
    "KxIndex" INTEGER,
    "class" INTEGER,
    "rr_class" DOUBLE
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table DEBRIEF_APPLY_RESULT_SIGNATURE;
create column table DEBRIEF_APPLY_RESULT_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (1, 'USER_APL',
'FUNCTION_HEADER_T', 'IN');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (2, 'USER_APL',
'OPERATION_CONFIG_T', 'IN');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (3, 'USER_APL',
'VARIABLE_DESC_T', 'IN');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (4, 'USER_APL',
'VARIABLE_ROLES_T', 'IN');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (5, 'USER_APL',
'ADULT01_T_OUT', 'IN');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (6, 'USER_APL',
'ADULT01_T_OUT', 'IN');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (7, 'USER_APL',
'ADULT01_T_OUT', 'IN');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (8, 'USER_APL',
'OPERATION_LOG_T', 'OUT');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (9, 'USER_APL', 'SUMMARY_T',
'OUT');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (10, 'USER_APL',
'INDICATORS_DATASET_T', 'OUT');
insert into DEBRIEF_APPLY_RESULT_SIGNATURE values (11, 'USER_APL',
'PROFITCURVE_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_DEBRIEF_APPLY_RESULT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA', 'DEBRIEF_APPLY_RESULT',
'USER_APL', 'APLWRAPPER_DEBRIEF_APPLY_RESULT', DEBRIEF_APPLY_RESULT_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create COLUMN table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', 'foobar');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table STAT_CONFIG;
create table STAT_CONFIG like OPERATION_CONFIG_T;
insert into STAT_CONFIG values ('APL/ModelType', 'statbuilder');
insert into STAT_CONFIG values ('APL/VariableEstimatorOf', 'rr_class;class');
insert into STAT_CONFIG values ('APL/CurveType','detected');
```

```
drop table STAT_VARIABLE_DESC;
create table STAT_VARIABLE_DESC like VARIABLE_DESC_T;
insert into STAT_VARIABLE_DESC
values(0,'KxIndex','integer','continuous',1,0,'','','');
insert into STAT_VARIABLE_DESC
values(1,'class','integer','nominal',0,0,'','','');
insert into STAT_VARIABLE_DESC
values(2,'rr_class','number','continuous',0,0,'','','');
drop table STAT_VARIABLE_ROLES;
create table STAT_VARIABLE_ROLES like VARIABLE_ROLES_T;
insert into STAT_VARIABLE_ROLES values ('class', 'target');
drop table OPERATION_LOG;
create COLUMN table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create COLUMN table SUMMARY like SUMMARY_T;
drop table INDICATORS_DATASET;
create COLUMN table INDICATORS_DATASET like INDICATORS_DATASET_T;
drop table PROFITCURVE;
create COLUMN table PROFITCURVE like PROFITCURVE_T;
drop view ESTIMATION;
create view ESTIMATION as (select * from USER_APL.ADULT01_APPLY where "KxIndex"
<= 39000 order by "KxIndex");
drop view VALIDATION;
create view VALIDATION as (select * from USER_APL.ADULT01_APPLY where "KxIndex"
> 39000 and "KxIndex" <= 44000 order by "KxIndex");
drop view TEST_VIEW;
create view TEST_VIEW as (select * from USER_APL.ADULT01_APPLY where "KxIndex" >
44000 order by "KxIndex");
DO BEGIN
    header      = select * from FUNC_HEADER;
    config      = select * from STAT_CONFIG;
    var_desc    = select * from STAT_VARIABLE_DESC;
    var_role    = select * from STAT_VARIABLE_ROLES;
    estimation  = select * from ESTIMATION;
    validation  = select * from VALIDATION;
    test        = select * from TEST_VIEW;


APLWRAPPER_DEBRIEF_APPLY_RESULT(:header, :config, :var_desc, :var_role, :estimati
on,:validation,:test,out_log, out_summ, out_indic, out_curve);

    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"       select * from :out_log;
    insert into  "USER_APL"."SUMMARY"             select * from :out_summ;
    insert into  "USER_APL"."INDICATORS_DATASET"  select * from :out_indic;
    insert into  "USER_APL"."PROFITCURVE"         select * from :out_curve;
    -- show result
    select * from INDICATORS_DATASET;
    select * from PROFITCURVE;
END;
```

8.2.3 FORECAST

Builds and trains an Automated Analytics time-series model, and then applies the model to obtain the forecast data.

This function lets you use different techniques to create a time-series model and generate a forecast:

- The default technique of the Automated Analytics engine
- The exponential smoothing technique
- The linear regression technique

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, DATASET, DATASET, LOG, SUMMARY, INDICATORS)
```

You can have `DATASET` as single OUT parameter.

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::FORECAST"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, INPUT_DATASET_SCHEMA, INPUT_DATASET_NAME, OUTPUT_DATASET_SCHEMA, OUTPUT_DATASET_NAME, LOG, SUMMARY, INDICATORS)
```

The parameters `INPUT_DATASET_SCHEMA`, `INPUT_DATASET_NAME`, `OUTPUT_DATASET_SCHEMA`, and `OUTPUT_DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_FORECAST"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, DATASET, DATASET, LOG, SUMMARY, INDICATORS)
```

Syntax

```
"_SYS_AFL"."APL_FORECAST__OVERLOAD_5_1"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, DATASET, DATASET)
```

Related Information

Segmented Modeling [page 26]

8.2.3.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	OPERATION_CONFIG [page 384]	Yes	The configuration of the training operation. For this function, you must declare the type of Automated Analytics model and you can declare the optional Cutting Strategy in the `OPERATION_CONFIG` table as shown in the next section.
IN	VARIABLE_DESC [page 396]	No (The table must be provided but it can be empty.)	The variable descriptions for the input dataset, as expected by the AutomatedAnalytics engine.
IN	VARIABLE_ROLES [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training
IN	DATASET [page 362]	Yes	The name of your input (training) dataset
OUT	DATASET [page 362]		The resulting forecast The default output, when no extra mode is requested, contains 3 columns: • The time point column. The name and the SQL type of this column are the same as the ones used in the input dataset. • The target column. The name and the SQL type of this column are the same as the ones used in the input dataset. • The forecast column. The name of the column is '`kts_1`', and the SQL type is the same as the target.
OUT	LOG [page 381]		The training log
OUT	SUMMARY [page 388]		The training summary
OUT	INDICATORS [page 370]		The variable statistics and indicators

8.2.3.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
`APL/ActivateExplanations`	No	• `'true'` • `'false'` (default value)	Generates information about the impact of cycles and influencers (categorical and numerical) for use in the statistical reports `TimeSeries_ImpactCycles`, `TimeSeries_ImpactInfluencersNumerical`, and `TimeSeries_ImpactInfluencersCategorical`

Key	Required	Supported Values	Description
APL/ApplyExtraMode	Yes	• 'No Extra' (default value) • 'Forecasts and Error Bars' • 'Stable Components and Error Bars' • 'First Forecast with Stable Components and Residues and Error Bars'	• 'No Extra' Generates time point, target, and forecast columns • 'Forecasts and Error Bars' Includes error bars with the forecast values The error bar at a given horizon H is an interval centered around the forecasted value, whose width is calculated as 1.96 * 2* RMSE (Root Mean Square Error). RMSE is calculated on the actual vs predicted value observed in a validation set when predicting at horizon H. The weighted value of 1.96 corresponds to a confidence level of 95%. The value of 1.64 corresponds to a confidence level of 90%. The value of 2.58 corresponds to a confidence level of 99%. The error bar columns are 'kts_1_lowerlimit_95%' and 'kts_1_upperlimit_95%'. • 'Stable Components and Error Bars' Includes forecast values, error bars, component values (trend, cycles, fluctuations) and their corresponding residuals in the output • 'First Forecast with Stable Components and Residues and Error Bars' Generates the same output as 'Stable Components and Error Bars' but includes only the first horizon

Key	Required	Supported Values	Description
APL/CuttingStrategy	No	• 'sequential with no test' • 'sequential' (default value)	The Cutting Strategy defines how a training set is cut under three subsets (training, validation and testing datasets) when needed. Depending on the model (model type, number of targets ...) not all Cutting Strategies can be used. See Automated Analytics documentation for details.
APL/DecomposeInfluencers	No	• 'true' • 'false' (default value)	Separates the influencers from the trend and cycles components and shows their impact in a dedicated column called ExtraPreds in the ApplyOut result table. Requires 'First Forecast with Stable Components and Residues and Error Bars' as APL/ApplyExtraMode. In addition, this alias is required to compute and expose the contributions during the learning phase.
APL/ForceNegativeForecast	No	• 'true' • 'false' (default value)	Activates a mode where the positive forecasts are ignored, that is, replaced with zero
APL/ForcePositiveForecast	No	• 'true' • 'false' (default value)	Activates a mode where the negative forecasts are ignored, that is, replaced with zero
APL/ForecastFallbackMethod	No	'LinearRegression', 'ExponentialSmoothing' (default)	Sets the method that will be used if the one specified with APL/ForecastMethod fails, for example when there are too few data points
APL/ForecastMaxCyclics	No	Positive integer. Default is 200.	Length of the longest cycle the model will try to detect. Controls the way that the model analyzes the periodicities in the signal. Also limited by the size of the training dataset. You can disable the cyclic analysis by setting this parameter to 0.
APL/ForecastMaxLags	No	Positive integer. Default is quarter of the training dataset size.	Defines the maximum dependency of the signal on its own past values. Controls the way that the model analyzes the random fluctuations in the signal. You can set this parameter to 0 to disable the fluctuations analysis.
APL/ForecastMethod	No	'Default', 'LinearRegression', 'ExponentialSmoothing'	Uses a forecast method different from the default Automated Analytics time-series algorithm

Key	Required	Supported Values	Description
`APL/Horizon`	Yes		Number of forecast time points
`APL/LastTrainingTimePoint`	No	A date of which format is YYYY:MM:DD HH:mm:ss	Value in the time point column which represents the last date for the training dataset. This alias is required if your input dataset has extra predictable variables or rows where the target is not set.
`APL/SegmentColumnName`	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID
`APL/SmoothingCycleLength`	No	[1..n] with n = integer value	Sets the cycle/seasonal length to be used for the smoothing instead of the cycle length candidates automatically determined by the Automated Analytics engine based on the time granularity, for example: month -> 4 (quarterly) or 12 (yearly).
`APL/TimePointColumnName`	Yes		Name of the column in the dataset that contains the time points of the time series
`APL/WithExtraPredictable`	No	• `'true'` (default value) • `'false'`	Uses all input variables as extra predictable variables

Related Information

APL/ApplyExtraMode [page 85]

8.2.3.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, FORECAST
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ----------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------
connect USER_APL password Password1;
drop type CASHFLOWS_FULL_T;
create type CASHFLOWS_FULL_T as table (
    "Date" DAYDATE,
    "WorkingDaysIndices" INTEGER,
    "ReverseWorkingDaysIndices" INTEGER,
    "MondayMonthInd" INTEGER,
    "TuesdayMonthInd" INTEGER,
```

```
    "WednesdayMonthInd" INTEGER,
    "ThursdayMonthInd" INTEGER,
    "FridayMonthInd" INTEGER,
    "BeforeLastMonday" INTEGER,
    "LastMonday" INTEGER,
    "BeforeLastTuesday" INTEGER,
    "LastTuesday" INTEGER,
    "BeforeLastWednesday" INTEGER,
    "LastWednesday" INTEGER,
    "BeforeLastThursday" INTEGER,
    "LastThursday" INTEGER,
    "BeforeLastFriday" INTEGER,
    "LastFriday" INTEGER,
    "Last5WDaysInd" INTEGER,
    "Last5WDays" INTEGER,
    "Last4WDaysInd" INTEGER,
    "Last4WDays" INTEGER,
    "LastWMonth" INTEGER,
    "BeforeLastWMonth" INTEGER,
    "Cash" DECIMAL(17,6)
);
-- ------------------------------------------------------------------------
-- Create table type for the forecast output
-- ------------------------------------------------------------------------
drop type FORECAST_OUT_T;
create type FORECAST_OUT_T as table (
    "Date" DAYDATE,
    "Cash" DOUBLE,
    "kts_1" DOUBLE
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table FORECAST_SIGNATURE;
create column table FORECAST_SIGNATURE like PROCEDURE_SIGNATURE_T;
insert into FORECAST_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into FORECAST_SIGNATURE values (2, 'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into FORECAST_SIGNATURE values (3, 'USER_APL','VARIABLE_DESC_T', 'IN');
insert into FORECAST_SIGNATURE values (4, 'USER_APL','VARIABLE_ROLES_T', 'IN');
insert into FORECAST_SIGNATURE values (5, 'USER_APL','CASHFLOWS_FULL_T', 'IN');
insert into FORECAST_SIGNATURE values (6, 'USER_APL','FORECAST_OUT_T', 'OUT');
insert into FORECAST_SIGNATURE values (7, 'USER_APL','OPERATION_LOG_T', 'OUT');
insert into FORECAST_SIGNATURE values (8, 'USER_APL','SUMMARY_T', 'OUT');
insert into FORECAST_SIGNATURE values (9, 'USER_APL','INDICATORS_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_FORECAST');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','FORECAST','USER_APL',
'APLWRAPPER_FORECAST', FORECAST_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
-- Create a view which contains the sorted dataset
drop view CASHFLOWS_SORTED;
create view CASHFLOWS_SORTED as select * from APL_SAMPLES.CASHFLOWS_FULL order
by "Date" asc;
drop table FORECAST_CONFIG;
create table FORECAST_CONFIG like OPERATION_CONFIG_T;
insert into FORECAST_CONFIG values ('APL/Horizon', '21');
insert into FORECAST_CONFIG values ('APL/TimePointColumnName', 'Date');
insert into FORECAST_CONFIG values ('APL/LastTrainingTimePoint', '2001-12-29
00:00:00');
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to use guess variables
```

```
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- insert into VARIABLE_ROLES values ('Last4WDays', 'skip');
-- insert into VARIABLE_ROLES values ('Last5WDaysInd', 'skip');
-- insert into VARIABLE_ROLES values ('BeforeLastThursday', 'skip');
-- insert into VARIABLE_ROLES values ('Last4WDaysInd', 'skip');
-- insert into VARIABLE_ROLES values ('WorkingDaysIndices', 'skip');
-- insert into VARIABLE_ROLES values ('LastThursday', 'skip');
-- insert into VARIABLE_ROLES values ('BeforeLastFriday', 'skip');
-- insert into VARIABLE_ROLES values ('Last5WDays', 'skip');
-- insert into VARIABLE_ROLES values ('BeforeLastWednesday', 'skip');
-- insert into VARIABLE_ROLES values ('ReverseWorkingDaysIndices', 'skip');
-- insert into VARIABLE_ROLES values ('BeforeLastTuesday', 'skip');
-- insert into VARIABLE_ROLES values ('LastTuesday', 'skip');
-- insert into VARIABLE_ROLES values ('ThursdayMonthInd', 'skip');
-- insert into VARIABLE_ROLES values ('FridayMonthInd', 'skip');
-- insert into VARIABLE_ROLES values ('LastMonday', 'skip');
-- insert into VARIABLE_ROLES values ('LastWednesday', 'skip');
-- insert into VARIABLE_ROLES values ('BeforeLastWMonth', 'skip');
-- insert into VARIABLE_ROLES values ('WednesdayMonthInd', 'skip');
-- insert into VARIABLE_ROLES values ('LastFriday', 'skip');
-- insert into VARIABLE_ROLES values ('BeforeLastMonday', 'skip');
-- insert into VARIABLE_ROLES values ('TuesdayMonthInd', 'skip');
-- insert into VARIABLE_ROLES values ('MondayMonthInd', 'skip');
-- insert into VARIABLE_ROLES values ('LastWMonth', 'skip');
insert into VARIABLE_ROLES values ('Date', 'input');
insert into VARIABLE_ROLES values ('Cash', 'target');
drop table FORECAST_OUT;
create table FORECAST_OUT like FORECAST_OUT_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config    = select * from FORECAST_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from USER_APL.CASHFLOWS_SORTED;

    APLWRAPPER_FORECAST(:header,  :config, :var_desc, :var_role, :dataset,
out_forecast, out_log, out_summary, out_indicators);

    -- store result into table
    insert into  "USER_APL"."FORECAST_OUT"     select * from :out_forecast;
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_summary;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indicators;
    -- show result
    select * from "USER_APL"."FORECAST_OUT" order by "Date" asc;
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.2.3.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, FORECAST
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create table type for the forecast output
-- ------------------------------------------------------------------------
drop type FORECAST_OUT_T;
create type FORECAST_OUT_T as table (
    "Date" DAYDATE,
    "Cash" DOUBLE,
    "kts_1" DOUBLE
);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
-- Create a view which contains the sorted dataset
drop view CASHFLOWS_SORTED;
create view CASHFLOWS_SORTED as select * from APL_SAMPLES.CASHFLOWS_FULL order
by "Date" asc;
drop table FORECAST_CONFIG;
create table FORECAST_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into FORECAST_CONFIG values ('APL/Horizon', '20',null);
insert into FORECAST_CONFIG values ('APL/TimePointColumnName', 'Date',null);
insert into FORECAST_CONFIG values ('APL/LastTrainingTimePoint', '2001-12-29
00:00:00',null);
insert into FORECAST_CONFIG values ('APL/ForcePositiveForecast', 'true',null);
insert into FORECAST_CONFIG values ('APL/WithExtraPredictable', 'true',null);
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to use guess variables
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
insert into VARIABLE_ROLES values ('Date', 'input',NULL,NULL,'#42');
insert into VARIABLE_ROLES values ('Cash', 'target',NULL,NULL,'#42');
drop table FORECAST_OUT;
create table FORECAST_OUT like FORECAST_OUT_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config    = select * from FORECAST_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
```

```
"SAP_PA_APL"."sap.pa.apl.base::FORECAST"(:header,  :config, :var_desc, :var_role,
 'USER_APL','CASHFLOWS_SORTED', 'USER_APL','FORECAST_OUT', out_log, out_summary,
out_indicators);

    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_summary;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indicators;
    -- show result
    select * from "USER_APL"."FORECAST_OUT" order by "Date" asc;
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.2.4 FORECAST_AND_DEBRIEF

Builds and trains an Automated Analytics time-series model, and then applies the model to obtain the forecast data and debriefing information.

This function lets you use different techniques to create a time-series model and generate a forecast:

- The default technique of the AutomatedAnalytics engine
- The exponential smoothing technique
- The linear regressions technique

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, DATASET,
DATASET, LOG, SUMMARY, INDICATORS, DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

You can have `DATASET` as single OUT parameter.

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::FORECAST_AND_DEBRIEF"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESCS, VARIABLE_ROLES, INPUT_DATASET_SCHEMA, INPUT_DATASET_NAME,
OUTPUT_DATASET_SCHEMA, OUTPUT_DATASET_NAME, LOG, SUMMARY, INDICATORS,  DEBRIEF_METRIC,
DEBRIEF_PROPERTY)
```

The parameters `INPUT_DATASET_SCHEMA`, `INPUT_DATASET_NAME`, `OUTPUT_DATASET_SCHEMA`, and `OUTPUT_DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_FORECAST__OVERLOAD_5_6"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESCS, VARIABLE_ROLES, DATASET, DATASET, LOG, SUMMARY, INDICATORS,
DEBRIEF_METRIC, DEBRIEF_PROPERTY)
```

Related Information

Segmented Modeling [page 26]

8.2.4.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`OPERATION_CONFIG` [page 384]	Yes	The configuration of the training operation. For this function, you must declare the type of Automated Analytics model and you can declare the optional Cutting Strategy in the `OPERATION_CONFIG` table as shown in the next section.
IN	`VARIABLE_DESC` [page 396]	No (The table must be provided but it can be empty.)	The variable descriptions for the input dataset, as expected by the AutomatedAnalytics engine.
IN	`VARIABLE_ROLES` [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training
IN	`DATASET` [page 362]	Yes	The name of your input (training) dataset
OUT	`DATASET` [page 362]		The resulting forecast The default output, when no extra mode is requested, contains 3 columns: • The time point column. The name and the SQL type of this column are the same as the ones used in the input dataset. • The target column. The name and the SQL type of this column are the same as the ones used in the input dataset. • The forecast column. The name of the column is '`kts_1`', and the SQL type is the same as the target.
OUT	`LOG` [page 381]		The training log
OUT	`SUMMARY` [page 388]		The training summary

Direction	Type	Required	Description
OUT	`INDICATORS [page 370]`		The variable statistics and indicators
OUT	DEBRIEF_METRIC [page 364]	Yes	The debrief metrics
OUT	DEBRIEF_PROPERTY [page 365]	Yes	The debrief properties

8.2.4.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Description
`APL/ActivateExplanations`	No	• `'true'` • `'false'` (default value)	Generates information about the impact of cycles and influencers (categorical and numerical) for use in the statistical reports `TimeSeries_ImpactCycles`, `TimeSeries_ImpactInfluencersNumerical`, and `TimeSeries_ImpactInfluencersCategorical`

Key	Required	Supported Values	Description
APL/ApplyExtraMode	Yes	• 'No Extra' (default value) • 'Forecasts and Error Bars' • 'Stable Components and Error Bars' • 'First Forecast with Stable Components and Residues and Error Bars'	• 'No Extra' Generates time point, target, and forecast columns • 'Forecasts and Error Bars' Includes error bars with the forecast values The error bar at a given horizon H is an interval centered around the forecasted value, whose width is calculated as 1.96 * 2* RMSE (Root Mean Square Error). RMSE is calculated on the actual vs predicted value observed in a validation set when predicting at horizon H. The weighted value of 1.96 corresponds to a confidence level of 95%. The value of 1.64 corresponds to a confidence level of 90%. The value of 2.58 corresponds to a confidence level of 99%. The error bar columns are 'kts_1_lowerlimit_95%' and 'kts_1_upperlimit_95%'. • 'Stable Components and Error Bars' Includes forecast values, error bars, component values (trend, cycles, fluctuations) and their corresponding residuals in the output • 'First Forecast with Stable Components and Residues and Error Bars' Generates the same output as 'Stable Components and Error Bars' but includes only the first horizon

Key	Required	Supported Values	Description
APL/CuttingStrategy	No	• 'sequential with no test' • 'sequential' (default value)	The Cutting Strategy defines how a training set is cut under three subsets (training, validation and testing datasets) when needed. Depending on the model (model type, number of targets ...) not all Cutting Strategies can be used. See Automated Analytics documentation for details.
APL/DecomposeInfluencers	No	• 'true' • 'false' (default value)	Separates the influencers from the trend and cycles components and shows their impact in a dedicated column called ExtraPreds in the ApplyOut result table. Requires 'First Forecast with Stable Components and Residues and Error Bars' as APL/ApplyExtraMode. In addition, this alias is required to compute and expose the contributions during the learning phase.
APL/ForceNegativeForecast	No	• 'true' • 'false' (default value)	Activates a mode where the positive forecasts are ignored, that is, replaced with zero
APL/ForcePositiveForecast	No	• 'true' • 'false' (default value)	Activates a mode where the negative forecasts are ignored, that is, replaced with zero
APL/ForecastFallbackMethod	No	'LinearRegression', 'ExponentialSmoothing' (default)	Sets the method that will be used if the one specified with APL/ForecastMethod fails, for example when there are too few data points
APL/ForecastMaxCyclics	No	Positive integer. Default is 200.	Length of the longest cycle the model will try to detect. Controls the way that the model analyzes the periodicities in the signal. Also limited by the size of the training dataset. You can disable the cyclic analysis by setting this parameter to 0.
APL/ForecastMaxLags	No	Positive integer. Default is quarter of the training dataset size.	Defines the maximum dependency of the signal on its own past values. Controls the way that the model analyzes the random fluctuations in the signal. You can set this parameter to 0 to disable the fluctuations analysis.
APL/ForecastMethod	No	'Default', 'LinearRegression', 'ExponentialSmoothing'	Uses a forecast method different from the default Automated Analytics time-series algorithm

Key	Required	Supported Values	Description
`APL/Horizon`	Yes		Number of forecast time points
`APL/LastTrainingTimePoint`	No	A date of which format is YYYY:MM:DD HH:mm:ss	Value in the time point column which represents the last date for the training dataset. This alias is required if your input dataset has extra predictable variables or rows where the target is not set.
`APL/LocalExplanations/Activate`	No	• `'true'` • `'false'` (default value)	Generates information about the decomposition of the forecasted value for each forecast period for use in the statistical report `TimeSeries_ForecastBreakdown`
`APL/SegmentColumnName`	No	String, no default value (the input dataset does not have a segment ID column and only one model is trained)	Name of the column that stores the segment ID
`APL/SmoothingCycleLength`	No	[1..n] with n = integer value	Sets the cycle/seasonal length to be used for the smoothing instead of the cycle length candidates automatically determined by the Automated Analytics engine based on the time granularity, for example: month -> 4 (quarterly) or 12 (yearly).
`APL/TimePointColumnName`	Yes		Name of the column in the dataset that contains the time points of the time series
`APL/WithExtraPredictable`	No	• `'true'` (default value) • `'false'`	Uses all input variables as extra predictable variables

Related Information

APL/ApplyExtraMode [page 85]

8.2.4.3 Example: PROCEDURE

```
-- ================================================================
connect USER_APL password Password1;
drop type FORECAST_OUT_T;
create type FORECAST_OUT_T as table (
    "Date" DAYDATE,
    "Cash" DOUBLE,
    "kts_1" DOUBLE
);
drop table FORECAST_OUT;
create table FORECAST_OUT like FORECAST_OUT_T;
```

```
drop view CASHFLOWS_SORTED;
create view CASHFLOWS_SORTED as select * from "APL_SAMPLES"."CASHFLOWS_FULL"
order by "Date" asc;
DO BEGIN
    declare header "SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
    declare config
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
    declare debrief_config
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
    declare var_desc
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
    declare var_role
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
    declare out_model "SAP_PA_APL"."sap.pa.apl.base::BASE.T.MODEL_BIN_OID";
    declare out_log   "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
    declare out_sum   "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
    declare out_indic "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
    declare out_debrief_metric
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_METRIC_OID";
    declare out_debrief_property
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_PROPERTY_OID";
    :header.insert(('Oid', '#42'));
    :header.insert(('LogLevel', '8'));
    :header.insert(('ModelFormat', 'bin'));
    :config.insert(('APL/Horizon', '20',null));
    :config.insert(('APL/TimePointColumnName', 'Date',null));
    :config.insert(('APL/LastTrainingTimePoint', '2001-12-29 00:00:00',null));
    :config.insert(('APL/ForcePositiveForecast', 'true',null));
    :config.insert(('APL/WithExtraPredictable', 'true',null));
    :var_role.insert(('Date', 'input',null,null,'Daily Xtra'));
    :var_role.insert(('Cash', 'target',null,null,'Daily Xtra'));
    insert into :var_desc  select *, '#42' from APL_SAMPLES.CASHFLOWS_DESC;

"SAP_PA_APL"."sap.pa.apl.base::FORECAST_AND_DEBRIEF"(:header,:config,:var_desc,:v
ar_role,'USER_APL','CASHFLOWS_SORTED',
'USER_APL','FORECAST_OUT',out_log,out_sum,out_indic,out_debrief_metric,out_debrie
f_property);

    -- Dump Statistics Report
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_Partition"(:out_debrief_prope
rty, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_Variables"(:out_debrief_prope
rty, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_ContinuousVariables"(:out_deb
rief_property, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_CategoryFrequencies"(:out_deb
rief_property, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_GroupFrequencies"(:out_debrie
f_property, :out_debrief_metric);
    -- Dump Statistics Continuous Target
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::ContinuousTarget_Statistics"(:out_debrie
f_property, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::ContinuousTarget_CrossStatistics"(:out_d
ebrief_property, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::ContinuousTarget_GroupCrossStatistics"(:
out_debrief_property, :out_debrief_metric);
    -- Dump TimeSeries Report
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ModelOverview"(:out_debrief_p
roperty, :out_debrief_metric);
```

```
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Components"(:out_debrief_prop
erty, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_DetrendedExtraPredictable"(:o
ut_debrief_property, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_VariablesContribution"(:out_d
ebrief_property, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Performance"(:out_debrief_pro
perty, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_PerformanceByHorizon"(:out_de
brief_property, :out_debrief_metric);
    select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Outliers"(:out_debrief_proper
ty, :out_debrief_metric);
END;
```

8.2.4.4 Example: Segmented Forecast PROCEDURE

```
-- @required(hanaMinimumVersion,2.0.40)
-- ================================================================
connect USER_APL password Password1;

drop type FORECAST_OUT_T;
create type FORECAST_OUT_T as table (
    "Seg"  INTEGER,
    "Date" DAYDATE,
    "Cash" DOUBLE,
    "kts_1" DOUBLE
);

drop table FORECAST_OUT;
create table FORECAST_OUT like FORECAST_OUT_T;

drop table "CASHFLOWS_SEG";
create table "CASHFLOWS_SEG" as (select *, 1 as "Seg" from
"APL_SAMPLES"."CASHFLOWS_FULL" limit 1);


drop procedure "timeseries_fill_segmented_dataset";
create procedure "timeseries_fill_segmented_dataset"(in nb_seg integer )
as BEGIN
    declare i integer;
    truncate table "CASHFLOWS_SEG";

    -- insert segment with not enough values to trigger an error
    insert into "CASHFLOWS_SEG" select  *,  -1 as "Seg" from
"APL_SAMPLES"."CASHFLOWS_FULL"  limit 1;

   -- insert n segments
    for i in 1..:nb_seg do
       insert into "CASHFLOWS_SEG" select  *,:i as "Seg" from
"APL_SAMPLES"."CASHFLOWS_FULL";
    end for;
END;

call "timeseries_fill_segmented_dataset"(4);
```

```
drop view "CASHFLOWS_SORTED";
create view "CASHFLOWS_SORTED" as select * from "CASHFLOWS_SEG" order by "Seg" ,
"Date" asc;

drop table "FORECAST_SUMMARY";
create table "FORECAST_SUMMARY" like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";

drop table "FORECAST_INDICATORS";
create table "FORECAST_INDICATORS" like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";

drop table "FORECAST_DEBRIEF_METRIC";
create table "FORECAST_DEBRIEF_METRIC" like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_METRIC_OID";

drop table "FORECAST_DEBRIEF_PROPERTY";
create table "FORECAST_DEBRIEF_PROPERTY" like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_PROPERTY_OID";

drop table "FORECAST_LOG";
create table "FORECAST_LOG" like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";

DO BEGIN
    declare header "SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
    declare config
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
    declare var_desc
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
    declare var_role
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
    declare out_log   "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
    declare out_sum   "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
    declare out_indic "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
    declare out_debrief_metric
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_METRIC_OID";
    declare out_debrief_property
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.DEBRIEF_PROPERTY_OID";

    truncate table "USER_APL"."FORECAST_SUMMARY";
    truncate table "USER_APL"."FORECAST_LOG";
    truncate table "USER_APL"."FORECAST_OUT";
    truncate table "USER_APL"."FORECAST_INDICATORS";
    truncate table "USER_APL"."FORECAST_DEBRIEF_METRIC";
    truncate table "USER_APL"."FORECAST_DEBRIEF_PROPERTY";

    :header.insert(('Oid', '#42'));
    :header.insert(('LogLevel', '1'));
    :header.insert(('ModelFormat', 'bin'));
    :header.insert(('MaxTasks', '4'));  -- define nb parallel tasks to use for
forecast

    :config.insert(('APL/SegmentColumnName', 'Seg',null)); -- define the column
used as the segmentation colum
    :config.insert(('APL/Horizon', '20',null));
    :config.insert(('APL/TimePointColumnName', 'Date',null));
    :config.insert(('APL/LastTrainingTimePoint', '2001-12-29 00:00:00',null));
    :config.insert(('APL/ForcePositiveForecast', 'true',null));
    :config.insert(('APL/WithExtraPredictable', 'true',null));

    :var_role.insert(('Date', 'input',null,null,'Daily Xtra'));
    :var_role.insert(('Cash', 'target',null,null,'Daily Xtra'));

    insert into :var_desc  select *, '#42' from APL_SAMPLES.CASHFLOWS_DESC;
```

```
"SAP_PA_APL"."sap.pa.apl.base::FORECAST_AND_DEBRIEF"(:header,:config,:var_desc,:v
ar_role,'USER_APL','CASHFLOWS_SORTED',
'USER_APL','FORECAST_OUT',out_log,out_sum,out_indic,out_debrief_metric,out_debrie
f_property);

    insert into  "USER_APL"."FORECAST_SUMMARY"           select * from :out_sum;
    insert into  "USER_APL"."FORECAST_LOG"               select * from :out_log;
    insert into  "USER_APL"."FORECAST_INDICATORS"        select *
from :out_indic;
    insert into  "USER_APL"."FORECAST_DEBRIEF_METRIC"    select *
from :out_debrief_metric;
    insert into  "USER_APL"."FORECAST_DEBRIEF_PROPERTY"  select *
from :out_debrief_property;

END;

select  * from
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ModelOverview"("USER_APL"."FO
RECAST_DEBRIEF_PROPERTY",  "USER_APL"."FORECAST_DEBRIEF_METRIC");
```

Related Information

Segmented Modeling [page 26]

8.2.5 GET_KEY_INFLUENCERS_FROM_MODEL

Gets the key influencers from a trained Classification or Regression model.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, MODEL, OPERATION_CONFIG, SUMMARY, INDICATORS, INFLUENCERS,
CONTINUOUS_GROUPS, OTHER_GROUPS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::GET_KEY_INFLUENCERS_FROM_MODEL"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, SUMMARY, INDICATORS, INFLUENCERS, CONTINUOUS_GROUPS, OTHER_GROUPS)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_GET_KEY_INFLUENCERS_FROM_MODEL"(FUNC_HEADER, MODEL,
OPERATION_CONFIG, SUMMARY, INDICATORS, INFLUENCERS, CONTINUOUS_GROUPS, OTHER_GROUPS)
```

8.2.5.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	MODEL [page 382]	Yes	The trained model
IN	OPERATION_CONFIG [page 384]	No	The configuration of the training operation. For this function, you must declare the type of AutomatedAnalytics model and you can declare the optional parameters in Common APL Aliases for Model Training [page 336].
OUT	SUMMARY [page 388]		The training summary
OUT	INDICATORS [page 370]		The key indicators
OUT	INFLUENCERS [page 380]		The key influencers
OUT	CONTINUOUS_GROUPS [page 362]		The description of the groups for continuous variables
OUT	OTHER_GROUPS [page 385]		The description of the groups for non-continuous variables

8.2.5.2 Example: DIRECT

```
-- ================================================================
-- APL_AREA, GET_KEY_INFLUENCERS_FROM_MODEL
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- Assumption 3: There's a valid trained model (created by APL) in the
MODEL_TRAIN_BIN table.
-- @depend(apl_createmodel_and_train.sql)
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
```

```
drop table GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE;
create column table GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE like
PROCEDURE_SIGNATURE_T;

insert into GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE values (2,
'USER_APL','MODEL_BIN_OID_T', 'IN');
insert into GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE values (3,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE values (4,
'USER_APL','SUMMARY_T', 'OUT');
insert into GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE values (5,
'USER_APL','INDICATORS_T', 'OUT');
insert into GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE values (6,
'USER_APL','INFLUENCERS_T', 'OUT');
insert into GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE values (7,
'USER_APL','CONTINUOUS_GROUPS_T', 'OUT');
insert into GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE values (8,
'USER_APL','OTHER_GROUPS_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_GET_KEY_INFLUENCERS_FRO
M_MODEL');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','GET_KEY_INFLUENCERS_FROM_MODEL',
'USER_APL', 'APLWRAPPER_GET_KEY_INFLUENCERS_FROM_MODEL',
GET_KEY_INFLUENCERS_FROM_MODEL_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table OPERATION_CONFIG;
create table OPERATION_CONFIG like OPERATION_CONFIG_T;
drop table INFLUENCERS;
create table INFLUENCERS like INFLUENCERS_T;
drop table CONTINUOUS_GROUPS;
create table CONTINUOUS_GROUPS like CONTINUOUS_GROUPS_T;
drop table OTHER_GROUPS;
create table OTHER_GROUPS like OTHER_GROUPS_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config   = select * from OPERATION_CONFIG;
    APLWRAPPER_GET_KEY_INFLUENCERS_FROM_MODEL(:header, :model_in, :config,
out_sum,out_indic,out_influencer, out_continous_groups, out_other_groups);

    -- store result into table
    insert into  "USER_APL"."SUMMARY"           select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"        select * from :out_indic;
    insert into  "USER_APL"."INFLUENCERS"       select * from :out_influencer;
    insert into  "USER_APL"."CONTINUOUS_GROUPS" select *
from :out_continous_groups;
    insert into  "USER_APL"."OTHER_GROUPS"      select * from :out_other_groups;
    -- show result
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."INFLUENCERS";
```

```
    select * from "USER_APL"."CONTINUOUS_GROUPS";
    select * from "USER_APL"."OTHER_GROUPS";
END;
```

8.2.5.3 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, GET_KEY_INFLUENCERS_FROM_MODEL
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- Assumption 2: There's a valid and trained model (created by APL) in the
MODEL_TRAIN_BIN table.
--               For instance, you have used apl_createmodel_and_train_ex.sql
before
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table USER_APL.OPERATION_CONFIG;
create table USER_APL.OPERATION_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
drop table INFLUENCERS;
create table INFLUENCERS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INFLUENCERS";
drop table CONTINUOUS_GROUPS;
create table CONTINUOUS_GROUPS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.CONTINUOUS_GROUPS";
drop table OTHER_GROUPS;
create table OTHER_GROUPS like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OTHER_GROUPS";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    model_in = select * from MODEL_TRAIN_BIN;
    config   = select * from OPERATION_CONFIG;

"SAP_PA_APL"."sap.pa.apl.base::GET_KEY_INFLUENCERS_FROM_MODEL"(:header, :model_in
, :config, out_sum,out_indic,out_influencer, out_continous_groups,
out_other_groups);

    -- store result into table
    insert into  "USER_APL"."SUMMARY"           select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"        select * from :out_indic;
    insert into  "USER_APL"."INFLUENCERS"       select * from :out_influencer;
    insert into  "USER_APL"."CONTINUOUS_GROUPS" select *
from :out_continous_groups;
    insert into  "USER_APL"."OTHER_GROUPS"      select * from :out_other_groups;
    -- show result
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."INFLUENCERS";
```

```
    select * from "USER_APL"."CONTINUOUS_GROUPS";
    select * from "USER_APL"."OTHER_GROUPS";
END;
```

8.2.6 KEY_INFLUENCERS

Finds the key influencers of a dataset.

> → Remember
>
> This function does not support multinomial classification models.

DIRECT

> {·} Syntax
>
> `<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, DATASET, LOG, SUMMARY, INDICATORS, INFLUENCERS, CONTINUOUS_GROUPS, OTHER_GROUPS)`

PROCEDURE

> {·} Syntax
>
> `"SAP_PA_APL"."sap.pa.apl.base::KEY_INFLUENCERS"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, DATASET_SCHEMA, DATASET_NAME, LOG, SUMMARY, INDICATORS, INFLUENCERS, CONTINUOUS_GROUPS, OTHER_GROUPS)`

The parameters `DATASET_SCHEMA` and `DATASET_NAME` are strings.

ANY PROCEDURE

> {·} Syntax
>
> `"_SYS_AFL"."APL_KEY_INFLUENCERS"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESCS, VARIABLE_ROLES, DATASET, LOG, SUMMARY, INDICATORS, INFLUENCERS, CONTINUOUS_GROUPS, OTHER_GROUPS)`

8.2.6.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	OPERATION_CONFIG [page 384]	Yes	The configuration of the training operation. For this function, you must declare the type of AutomatedAnalytics model and you can declare the optional parameters in Common APL Aliases for Model Training [page 336].
IN	VARIABLE_DESC [page 396]	No (The table must be provided but it can be empty.)	The descriptions of the variables for this training
IN	VARIABLE_ROLES [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training
IN	DATASET [page 362]	Yes	The training dataset
OUT	LOG [page 381]		The training log
OUT	SUMMARY [page 388]		The training summary
OUT	INDICATORS [page 370]		The key indicators
OUT	INFLUENCERS [page 380]		The key influencers
OUT	CONTINUOUS_GROUPS [page 362]		The description of the groups for continuous variables
OUT	OTHER_GROUPS [page 385]		The description of the groups for non-continuous variables

8.2.6.2 Example

```
-- ============================================================
-- APL_AREA, KEY_INFLUENCERS
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
```

```
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table KEY_INFLUENCERS_SIGNATURE;
create column table KEY_INFLUENCERS_SIGNATURE like PROCEDURE_SIGNATURE_T;

insert into KEY_INFLUENCERS_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into KEY_INFLUENCERS_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into KEY_INFLUENCERS_SIGNATURE values (3, 'USER_APL','VARIABLE_DESC_T',
'IN');
insert into KEY_INFLUENCERS_SIGNATURE values (4, 'USER_APL','VARIABLE_ROLES_T',
'IN');
insert into KEY_INFLUENCERS_SIGNATURE values (5, 'USER_APL','ADULT01_T', 'IN');
insert into KEY_INFLUENCERS_SIGNATURE values (6, 'USER_APL','OPERATION_LOG_T',
'OUT');
insert into KEY_INFLUENCERS_SIGNATURE values (7, 'USER_APL','SUMMARY_T', 'OUT');
insert into KEY_INFLUENCERS_SIGNATURE values (8, 'USER_APL','INDICATORS_T',
'OUT');
insert into KEY_INFLUENCERS_SIGNATURE values (9, 'USER_APL','INFLUENCERS_T',
'OUT');
insert into KEY_INFLUENCERS_SIGNATURE values (10,
'USER_APL','CONTINUOUS_GROUPS_T', 'OUT');
insert into KEY_INFLUENCERS_SIGNATURE values (11, 'USER_APL','OTHER_GROUPS_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_KEY_INFLUENCERS');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','KEY_INFLUENCERS','USER_APL',
'APLWRAPPER_KEY_INFLUENCERS', KEY_INFLUENCERS_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table KEY_INFLUENCERS_CONFIG;
create table KEY_INFLUENCERS_CONFIG like OPERATION_CONFIG_T;
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- default roles are fine: all the variables but the last one are going to be
input variables, the last one is the target
drop table VARIABLE_DESC;
```

```
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to use guess variables
drop table INFLUENCERS;
create table INFLUENCERS like INFLUENCERS_T;
drop table CONTINUOUS_GROUPS;
create table CONTINUOUS_GROUPS like CONTINUOUS_GROUPS_T;
drop table OTHER_GROUPS;
create table OTHER_GROUPS like OTHER_GROUPS_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from KEY_INFLUENCERS_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;

APLWRAPPER_KEY_INFLUENCERS(:header, :config, :var_desc,  :var_role, :dataset,
out_log, out_sum,out_indic,out_influencer, out_continous_groups,
out_other_groups);

    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"          select * from :out_log;
    insert into  "USER_APL"."SUMMARY"           select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"        select * from :out_indic;
    insert into  "USER_APL"."INFLUENCERS"       select * from :out_influencer;
    insert into  "USER_APL"."CONTINUOUS_GROUPS" select *
from :out_continous_groups;
    insert into  "USER_APL"."OTHER_GROUPS"      select * from :out_other_groups;
    -- show result
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."INFLUENCERS";
    select * from "USER_APL"."CONTINUOUS_GROUPS";
    select * from "USER_APL"."OTHER_GROUPS";
END;
```

8.2.7 PROFILE_DATA

Performs a data profiling operation on the specified dataset.

DIRECT

Syntax

`<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET, LOG, SUMMARY, INDICATORS, VARIABLE_DESC)`

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::PROFILE_DATA"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESC, VARIABLE_ROLES, DATASET_SCHEMA, DATASET_NAME, LOG, SUMMARY, INDICATORS,
VARIABLE_DESC)
```

The parameters `DATASET_SCHEMA` and `DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_PROFILE_DATA"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC,
VARIABLE_ROLES, DATASET, LOG, SUMMARY, INDICATORS, VARIABLE_DESC)
```

8.2.7.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`VARIABLE_DESC` [page 396]	No (The table must be provided but it can be empty.)	The description of the variables used in the training data source. If no variable description is provided (i.e. empty table), then the variable descriptions are automatically guessed by the Automated Analytics engine.
IN	`VARIABLE_ROLES` [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this training
IN	`OPERATION_CONFIG` [page 384]	Yes	The configuration of the training operation. For this function, you must declare the type of Automated Analytics model and you can declare the optional partition strategy in the `OPERATION_CONFIG` table as shown in the next section.
IN	`DATASET` [page 362]	Yes	The name of your dataset to be profiled
OUT	`LOG` [page 381]		The profiling log produced by the AutomatedAnalytics engine

Direction	Type	Required	Description
OUT	SUMMARY [page 388]		The profiling summary
OUT	INDICATORS [page 370]		The variable statistics and indicators
OUT	VARIABLE_DESC [page 396]		The variable descriptions from the model

8.2.7.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Default	Description
APL/ModelType	Yes	• 'regression/classification' (legacy) • 'regression' (gradient boosting) • 'binary classification' (gradient boosting) • 'multiclass' for multinomial classification models (gradient boosting) • 'clustering' • 'timeseries' • 'statbuilder'		Type of the Automated Analytics model

Key	Required	Supported Values	Default	Description
APL/CuttingStrategy	Yes for time-series models, otherwise No	For classification, regression and clustering models: • 'random' • 'periodic' • 'sequential' • 'periodic with test at end' • 'random with test at end' • 'sequential with no test' • 'periodic with no test' • 'random with no test' For a time series model: 'sequential' 'sequential with no test'	'random' for classification, regression, and clustering, 'sequential with no test' for time-series	Partition strategy for the training dataset
APL/VariableAutoSelection	No	• 'true' • 'false'	'false'	Auto selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.

Key	Required	Supported Values	Default	Description
`APL/ExcludeLeakVariables`	No	• `true` • `false`	`'false'`	Exclude leak variable allows you to exclude leak variables from output. To exclude leak variables, set both APL/VariableAutoSelection and APL/ExcludeLeakVariable to `true`. When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `"Leak Variable"`.
`APL/TargetKey`	No		The least frequent category of the Target variable	The category of the Target variable that will be used as the positive target in the model
`APL/CalculateSQLExpressions`	No	• `'enabled'` • `'disabled'`	`'disabled'`	Generates SQL expression for clustering model
`APL/IncludeCategoryNormalProfits`	No			
`APL/NbClusters`	No		`10`	Number of clusters for the model
`APL/NbClustersMin`	No			Cluster count range customization, lower number of clusters. Overrides `NbClusters`.
`APL/NbClustersMax`	No			Cluster count range customization, upper number of clusters. Overrides `NbClusters`.

Related Information

List of Excluded Variables [page 458]
INDICATORS [page 370]

8.2.7.3 Example: DIRECT

```
-- ====================================================================
-- APL_AREA, PROFILE_DATA
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
drop table PROFILE_DATA_SIGNATURE;
create column table PROFILE_DATA_SIGNATURE like PROCEDURE_SIGNATURE_T;

insert into PROFILE_DATA_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T',
'IN');
insert into PROFILE_DATA_SIGNATURE values (2, 'USER_APL','OPERATION_CONFIG_T',
'IN');
insert into PROFILE_DATA_SIGNATURE values (3, 'USER_APL','VARIABLE_DESC_T',
'IN');
insert into PROFILE_DATA_SIGNATURE values (4, 'USER_APL','VARIABLE_ROLES_T',
'IN');
insert into PROFILE_DATA_SIGNATURE values (5, 'USER_APL','ADULT01_T', 'IN');
insert into PROFILE_DATA_SIGNATURE values (6, 'USER_APL','OPERATION_LOG_T',
'OUT');
insert into PROFILE_DATA_SIGNATURE values (7, 'USER_APL','SUMMARY_T', 'OUT');
insert into PROFILE_DATA_SIGNATURE values (8, 'USER_APL','INDICATORS_T', 'OUT');
insert into PROFILE_DATA_SIGNATURE values (9, 'USER_APL','VARIABLE_DESC_OID_T',
'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_PROFILE_DATA');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','PROFILE_DATA','USER_APL',
'APLWRAPPER_PROFILE_DATA', PROFILE_DATA_SIGNATURE);
```

```
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table PROFILE_DATA_CONFIG;
create table PROFILE_DATA_CONFIG like OPERATION_CONFIG_T;
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- default roles are adjusted: we want all variables to be considered as input
variables when profiling the dataset.
insert into VARIABLE_ROLES values ('class', 'input');
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to let the engine guess variable descriptions
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
drop table VARIABLE_DESC_OUT;
create table VARIABLE_DESC_OUT like VARIABLE_DESC_OID_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from PROFILE_DATA_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;

APLWRAPPER_PROFILE_DATA(:header, :config, :var_desc,:var_role, :dataset,out_log,o
ut_sum,out_indic, out_var_desc);

    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"     select * from :out_log;
    insert into  "USER_APL"."SUMMARY"           select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"        select * from :out_indic;
    insert into  "USER_APL"."VARIABLE_DESC_OUT" select * from :out_var_desc;
    -- show result
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."VARIABLE_DESC_OUT";
END;
```

8.2.7.4 Example: PROCEDURE

```
-- ============================================================
-- APL_AREA, PROFILE_DATA
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
```

```
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table PROFILE_DATA_CONFIG;
create table PROFILE_DATA_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
-- default roles are adjusted: we want all variables to be considered as input
variables when profiling the dataset.
insert into VARIABLE_ROLES values ('class', 'input', null,null,null);
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to let the engine guess variable descriptions
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table VARIABLE_DESC_OUT;
create table VARIABLE_DESC_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header           = select * from FUNC_HEADER;
    config           = select * from PROFILE_DATA_CONFIG;
    variable_desc    = select * from VARIABLE_DESC;
    variable_roles    = select * from VARIABLE_ROLES;

"SAP_PA_APL"."sap.pa.apl.base::PROFILE_DATA"(:header, :config, :variable_desc, :v
ariable_roles, 'APL_SAMPLES','ADULT01', out_operation_log, out_summary,
out_indicators, out_variable_desc);
    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"      select *
from :out_operation_log;
    insert into  "USER_APL"."SUMMARY"            select * from :out_summary;
    insert into  "USER_APL"."INDICATORS"         select * from :out_indicators;
    insert into  "USER_APL"."VARIABLE_DESC_OUT"  select *
from :out_variable_desc;
    -- show result
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."VARIABLE_DESC_OUT";
END;
```

8.2.8 PROFILE_DATA_AND_GET_ASSOCIATIONRULES

Performs data profiling on a dataset and generates association rules.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET, LOG, SUMMARY, INDICATORS, VARIABLE_DESC, ASSOCIATION_RULES, RULE_ITEMS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::PROFILE_DATA_AND_GET_ASSOCIATIONRULES"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET_SCHEMA, DATASET_NAME, LOG, SUMMARY, INDICATORS, VARIABLE_DESC, ASSOCIATION_RULES, RULE_ITEMS)
```

The parameters `DATASET_SCHEMA` and `DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_PROFILE_DATA_AND_GET_ASSOCIATIONRULES"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET, LOG, SUMMARY, INDICATORS, VARIABLE_DESC, ASSOCIATION_RULES, RULE_ITEMS)
```

8.2.8.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header defines the operating parameters of a function call. For instance, it can be used to set the logging level or to provide an operation ID. A function header is a table made of a collection of string pairs {key, value}. It is an optional parameter, which means the table can be empty.
IN	`OPERATION_CONFIG` [page 384]	No (The table must be provided but it can be empty.)	The additional configuration parameters, as described in the next section.
IN	`VARIABLE_DESC` [page 396]	No (The table must be provided but it can be empty.)	The description of the variables used in the training data source. If no variable description is provided (i.e. empty table), then the variable descriptions are automatically guessed by the Automated Analytics engine.
IN	`VARIABLE_ROLES` [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for this operation. This can also be used to define composite variables.

Direction	Type	Required	Description
IN	DATASET [page 362]		The dataset to be profiled
OUT	LOG [page 381]		The profiling log produced by the AutomatedAnalytics engine
OUT	SUMMARY [page 388]		The profiling summary, similar to a training summary
OUT	INDICATORS [page 370]		The variable statistics and indicators
OUT	VARIABLE_DESC [page 396]		The description of the variables
OUT	ASSOCIATION_RULES [page 361]		The association rules. The association rule antecedent and consequent refer to items described in the rules items.
OUT	RULE_ITEMS [page 388]		The rule items referenced in the association rules

8.2.8.2 OPERATION_CONFIG Parameters

Key	Supported Values	Default	Description
APL/AssocRulesMinSupport	[0,1] floating point, or]1,n] integer	0.20	Minimum support required for a rule. With a value > 1, we consider the number of sessions. With a value between 0 and 1, we consider a percentage of the number of sessions.
APL/AssocRulesMinConfidence	[0,1] floating point value	0.95	Gives the minimum threshold for the confidence of a rule

8.2.8.3 Example: DIRECT

```
-- ================================================================
-- APL_AREA, PROFILE_DATA_AND_GET_ASSOCIATIONRULES
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ----------------------------------------------------------------------
-- Create table type for the dataset
-- ----------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type ADULT01_T;
create type ADULT01_T as table (
```

```
    "age" INTEGER,
    "workclass" NVARCHAR(32),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(32),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(32),
    "occupation" NVARCHAR(32),
    "relationship" NVARCHAR(32),
    "race" NVARCHAR(32),
    "sex" NVARCHAR(16),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(32),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
drop table PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE;
create column table PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE like
PROCEDURE_SIGNATURE_T;

insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_T', 'IN');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (4,
'USER_APL','VARIABLE_ROLES_T', 'IN');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (5,
'USER_APL','ADULT01_T', 'IN');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (6,
'USER_APL','OPERATION_LOG_T', 'OUT');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (7,
'USER_APL','SUMMARY_T', 'OUT');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (8,
'USER_APL','INDICATORS_T', 'OUT');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (9,
'USER_APL','VARIABLE_DESC_OID_T', 'OUT');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (10,
'USER_APL','ASSOCIATION_RULES_T', 'OUT');
insert into PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE values (11,
'USER_APL','RULE_ITEMS_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_PROFILE_DATA_AND_GET_AS
SOCIATIONRULES');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','PROFILE_DATA_AND_GET_ASSOCIATION
RULES','USER_APL', 'APLWRAPPER_PROFILE_DATA_AND_GET_ASSOCIATIONRULES',
PROFILE_DATA_AND_GET_ASSOCIATIONRULES_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table ASSOCRULES_CONFIG;
create table ASSOCRULES_CONFIG like OPERATION_CONFIG_T;
insert into ASSOCRULES_CONFIG values ('APL/AssocRulesMinSupport','0.2');
insert into ASSOCRULES_CONFIG values ('APL/AssocRulesMinConfidence','0.97');
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- default roles are adjusted: we want all variables to be considered as input
variables when profiling the dataset.
insert into VARIABLE_ROLES values ('class', 'input');
```

```
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to let the engine guess variable descriptions
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
drop table VARIABLE_DESC_OUT;
create table VARIABLE_DESC_OUT like VARIABLE_DESC_OID_T;
drop table ASSOCIATION_RULES;
create table ASSOCIATION_RULES like ASSOCIATION_RULES_T;
drop table RULE_ITEMS;
create table RULE_ITEMS like RULE_ITEMS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from ASSOCRULES_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;

APLWRAPPER_PROFILE_DATA_AND_GET_ASSOCIATIONRULES(:header, :config, :var_desc,:var
_role, :dataset,out_log,out_sum,out_indic, out_var_desc, out_association_rules,
out_rule_items);

    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"     select * from :out_log;
    insert into  "USER_APL"."SUMMARY"           select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"        select * from :out_indic;
    insert into  "USER_APL"."VARIABLE_DESC_OUT" select * from :out_var_desc;
    insert into  "USER_APL"."ASSOCIATION_RULES" select *
from :out_association_rules;
    insert into  "USER_APL"."RULE_ITEMS"        select * from :out_rule_items;
    -- show result
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."VARIABLE_DESC_OUT";
    select * from "USER_APL"."RULE_ITEMS" order by ITEM_ID;
    select * from "USER_APL"."ASSOCIATION_RULES" order by RULE_CONFIDENCE desc;
END;
```

8.2.8.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, PROFILE_DATA_AND_GET_ASSOCIATIONRULES_AND_GET_ASSOCIATIONRULES
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
```

```
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
drop table ASSOCRULES_CONFIG;
create table ASSOCRULES_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into ASSOCRULES_CONFIG values ('APL/AssocRulesMinSupport','0.2','');
insert into ASSOCRULES_CONFIG values ('APL/AssocRulesMinConfidence','0.97','');
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
-- default roles are adjusted: we want all variables to be considered as input
variables when profiling the dataset.
insert into VARIABLE_ROLES values ('class', 'input',null,null,null);
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to let the engine guess variable descriptions
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table VARIABLE_DESC_OUT;
create table VARIABLE_DESC_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
drop table ASSOCIATION_RULES;
create table ASSOCIATION_RULES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.ASSOCIATION_RULES";
drop table RULE_ITEMS;
create table RULE_ITEMS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.RULE_ITEMS";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header           = select * from FUNC_HEADER;
    config           = select * from ASSOCRULES_CONFIG;
    variable_desc    = select * from VARIABLE_DESC;
    variable_roles    = select * from VARIABLE_ROLES;

"SAP_PA_APL"."sap.pa.apl.base::PROFILE_DATA_AND_GET_ASSOCIATIONRULES"(:header, :c
onfig, :variable_desc, :variable_roles, 'APL_SAMPLES','ADULT01',
out_operation_log, out_summary, out_indicators, out_variable_desc,
out_association_rules, out_rule_items);
    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"      select *
from :out_operation_log;
    insert into  "USER_APL"."SUMMARY"            select * from :out_summary;
    insert into  "USER_APL"."INDICATORS"         select * from :out_indicators;
    insert into  "USER_APL"."VARIABLE_DESC_OUT"  select *
from :out_variable_desc;
    insert into  "USER_APL"."ASSOCIATION_RULES"  select *
from :out_association_rules;
    insert into  "USER_APL"."RULE_ITEMS"         select * from :out_rule_items;
    -- show result
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."VARIABLE_DESC_OUT";
    select * from "USER_APL"."RULE_ITEMS" order by ITEM_ID;
    select * from "USER_APL"."ASSOCIATION_RULES" order by RULE_CONFIDENCE desc;
END;
```

8.2.9 RECOMMEND

Performs a one-shot recommendation. Consists in building and training a recommendation model, and then applying the model to obtain the recommendation score and items for all users present in the transaction table.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, TRANSACTION, USER,
RECO_SCORE, LOG, SUMMARY, INDICATORS)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::RECOMMEND"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESC, TRANSACTION_SCHEMA, TRANSACTION_NAME, USER_DATA_SCHEMA, USER_DATA_NAME,
RECO_SCORE_SCHEMA, RECO_SCORE_NAME, LOG, SUMMARY, INDICATORS)
```

The parameters `TRANSACTION_SCHEMA`, `TRANSACTION_NAME`, `USER_DATA_SCHEMA`, `USER_DATA_NAME`, `RECO_SCORE_SCHEMA`, and `RECO_SCORE_NAME` are strings.

You can have the recommendation score as a single OUT parameter.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_RECOMMEND"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, TRANSACTION,
USER, RECO_SCORE, LOG, SUMMARY, INDICATORS)
```

Syntax

```
"_SYS_AFL"."APL_RECOMMEND__OVERLOAD_5_1"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESC, TRANSACTION, USER, RECO_SCORE)
```

8.2.9.1 Input/Output Tables

Direction	Type	Required	Description
IN	`FUNC_HEADER` [page 365]	No (The table must be provided but it can be empty.)	The function header
IN	`OPERATION_CONFIG` [page 384]	Yes	The configuration of the operation.

Direction	Type	Required	Description
IN	VARIABLE_DESC [page 396]	No (The table must be provided but it can be empty.)	The variable descriptions for the input dataset, as expected by the Automated Analytics engine. If no variable description is provided (i.e. empty table), then the variable descriptions are automatically guessed by the Automated Analytics engine.
IN	TRANSACTION [page 412]	Yes	The transaction dataset. It should contain mandatory columns User and Item and optionally Weight and Date columns
IN	USER [page 396]	Yes	The transaction dataset. It should contain mandatory column User having same name as transaction table
OUT	RECO_SCORE [page 411]		The resulting recommendation score
OUT	LOG [page 381]		The training log produced by the AutomatedAnalytics engine
OUT	SUMMARY [page 388]		The training summary
OUT	INDICATORS [page 370]		The variable statistics and indicators

8.2.9.2 OPERATION_CONFIG Parameters

Key	Required	Supported Values	Default	Description
APL/User	Yes			User entity : bipartite source node
APL/Item	Yes			Item/product entity: bipartite destination node
APL/Weight	No			Graph edge weight
APL/Date	No			Transaction date
APL/StartDate	No			Start date filter
APL/EndDate	No			End date filter
APL/BestSeller	No		50000	BestSeller threshold
APL/MaxTopNodes	No		100000	Max top nodes

Key	Required	Supported Values	Default	Description
APL/MinimumSupport	No		2	Minimum support to consider the association rule
APL/MinimumConfidence	No		0.05	Minimum confidence to consider the association rule
APL/MinimumPredictivePower	No		disable	Minimum KI to consider the association rule
APL/Top	No		Max	Keep top N of recommended items ranked by confidence
APL/IncludeBestSellers	No		false	Include de-facto best seller items into the suggestion list
APL/SkipAlreadyOwned	No		true	Exclude items already owned from the suggestion list

8.2.9.3 Example: DIRECT

```
-- ====================================================================
-- APL_AREA, RECOMMEND
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
drop type CUST_TRANSACTIONS_T;
create type CUST_TRANSACTIONS_T as table (
    "UserID" INTEGER,
    "ItemPurchased" NVARCHAR(18),
    "Date_PutInCaddy" DATETIME,
    "Quantity" INTEGER,
    "TransactionID" INTEGER
);
drop type USER_T;
create type USER_T as table (
    "UserID" INTEGER
);
-- ------------------------------------------------------------------------
-- Create table type for the RECOMMEND output
-- ------------------------------------------------------------------------
drop type RECO_SCORE_T;
create type RECO_SCORE_T as table (
    "UserID" INTEGER,
    "ItemPurchased" NVARCHAR(128),
    "sn_rec_rule_id" INTEGER,
```

```
    "sn_rec_kxReco" NVARCHAR(128),
    "sn_rec_source"  NVARCHAR(128),
    "sn_rec_score" DOUBLE
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table RECO_SIGNATURE;
create column table RECO_SIGNATURE like PROCEDURE_SIGNATURE_T;
insert into RECO_SIGNATURE values (1, 'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into RECO_SIGNATURE values (2, 'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into RECO_SIGNATURE values (3, 'USER_APL','VARIABLE_DESC_T', 'IN');
insert into RECO_SIGNATURE values (4, 'USER_APL','CUST_TRANSACTIONS_T', 'IN');
insert into RECO_SIGNATURE values (5, 'USER_APL','USER_T', 'IN');
insert into RECO_SIGNATURE values (6, 'USER_APL','RECO_SCORE_T', 'OUT');
insert into RECO_SIGNATURE values (7, 'USER_APL','OPERATION_LOG_T', 'OUT');
insert into RECO_SIGNATURE values (8, 'USER_APL','SUMMARY_T', 'OUT');
insert into RECO_SIGNATURE values (9, 'USER_APL','INDICATORS_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_RECO');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','RECOMMEND','USER_APL',
'APLWRAPPER_RECO', RECO_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
-- Model config
drop table RECO_CONFIG;
create table RECO_CONFIG like OPERATION_CONFIG_T;
insert into RECO_CONFIG values ('APL/User', 'UserID');                  --
mandatory
insert into RECO_CONFIG values ('APL/Item', 'ItemPurchased');           --
mandatory
insert into RECO_CONFIG values ('APL/Weight', 'Quantity');              --
optional
insert into RECO_CONFIG values ('APL/Date', 'Date_PutInCaddy');         --
optional
insert into RECO_CONFIG values ('APL/BestSeller', '1000000');           --
optional (default: 50000)
insert into RECO_CONFIG values ('APL/MinimumSupport', '4');             --
optional (default: 2)
insert into RECO_CONFIG values ('APL/MinimumConfidence', '0.1');        --
optional (default: 0.05)
insert into RECO_CONFIG values ('APL/MinimumPredictivePower', '0.01');  --
optional (default: disable)
insert into RECO_CONFIG values ('APL/MaxTopNodes', '1000');             --
optional (default: 100000)
-- Apply configuration
--insert into RECO_CONFIG values ('APL/Top', '6');                        --
optional (default: max)
insert into RECO_CONFIG values ('APL/IncludeBestSellers', 'true');      --
optional (default: false)
insert into RECO_CONFIG values ('APL/SkipAlreadyOwned', 'false');       --
optional (default: true)
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to use guess variables
drop table CUST_TO_SCORE;
create table CUST_TO_SCORE like USER_T;
-- no data from CUST_TO_SCORE means that the APL internally considers all users
present in the transactional data
--insert into CUST_TO_SCORE values ('23');
--insert into CUST_TO_SCORE values ('24');
--insert into CUST_TO_SCORE values ('33');
--insert into CUST_TO_SCORE values ('75');
```

```
drop table RECO_SCORE;
create table RECO_SCORE like RECO_SCORE_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header       = select * from FUNC_HEADER;
    config       = select * from RECO_CONFIG;
    var_desc     = select * from VARIABLE_DESC;
    dataset      = select * from APL_SAMPLES.CUST_TRANSACTIONS;
    user         = select * from USER_APL.CUST_TO_SCORE;
    APLWRAPPER_RECO(:header, :config, :var_desc, :dataset, :user, out_score,
out_log, out_summ, out_indic);
    -- store result into table
    insert into  "USER_APL"."RECO_SCORE"    select * from :out_score;
    insert into  "USER_APL"."OPERATION_LOG" select * from :out_log;
    insert into  "USER_APL"."SUMMARY"       select * from :out_summ;
    insert into  "USER_APL"."INDICATORS"    select * from :out_indic;
    -- show result
    select * from "USER_APL"."RECO_SCORE" order by "UserID";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.2.9.4 Example: PROCEDURE

```
-- ================================================================
-- APL_AREA, RECOMMEND
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
drop type CUST_TRANSACTIONS_T;
create type CUST_TRANSACTIONS_T as table (
    "UserID" INTEGER,
    "ItemPurchased" NVARCHAR(128),
    "Date_PutInCaddy" DATETIME,
    "Quantity" INTEGER,
    "TransactionID" INTEGER
);
drop type USER_T;
create type USER_T as table (
    "UserID" INTEGER
);
-- ------------------------------------------------------------------------
-- Create table type for the RECOMMEND output
-- ------------------------------------------------------------------------
drop type RECO_SCORE_T;
create type RECO_SCORE_T as table (
    "UserID" INTEGER,
    "ItemPurchased" NVARCHAR(128),
    "sn_rec_rule_id" INTEGER,
```

```
    "sn_rec_kxReco" NVARCHAR(128),
    "sn_rec_source"  NVARCHAR(128),
    "sn_rec_score" DOUBLE
);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
-- Model config
drop table RECO_CONFIG;
create table RECO_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_DETAILED";
insert into RECO_CONFIG values ('APL/User', 'UserID',null);                   --
mandatory
insert into RECO_CONFIG values ('APL/Item', 'ItemPurchased',null);            --
mandatory
insert into RECO_CONFIG values ('APL/Weight', 'Quantity',null);               --
optional
insert into RECO_CONFIG values ('APL/Date', 'Date_PutInCaddy',null);          --
optional
insert into RECO_CONFIG values ('APL/BestSeller', '1000000',null);            --
optional (default: 50000)
insert into RECO_CONFIG values ('APL/MinimumSupport', '4',null);              --
optional (default: 2)
insert into RECO_CONFIG values ('APL/MinimumConfidence', '0.1',null);         --
optional (default: 0.05)
insert into RECO_CONFIG values ('APL/MinimumPredictivePower', '0.01',null);   --
optional (default: disable)
insert into RECO_CONFIG values ('APL/MaxTopNodes', '1000',null);              --
optional (default: 100000)
-- Apply configuration
--insert into RECO_CONFIG values ('APL/Top', '6');                          --
optional (default: max)
insert into RECO_CONFIG values ('APL/IncludeBestSellers', 'true',null);       --
optional (default: false)
insert into RECO_CONFIG values ('APL/SkipAlreadyOwned', 'false',null);        --
optional (default: true)
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to use guess variables
drop table CUST_TO_SCORE;
create table CUST_TO_SCORE like USER_T;
-- no data from CUST_TO_SCORE means that the APL internally considers all users
present in the transactional data
--insert into CUST_TO_SCORE values ('23');
--insert into CUST_TO_SCORE values ('24');
--insert into CUST_TO_SCORE values ('33');
--insert into CUST_TO_SCORE values ('75');
drop table RECO_SCORE;
create table RECO_SCORE like RECO_SCORE_T;
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header       = select * from FUNC_HEADER;
```

```
    config       = select * from RECO_CONFIG;
    var_desc     = select * from VARIABLE_DESC;
    "SAP_PA_APL"."sap.pa.apl.base::RECOMMEND"(:header, :config, :var_desc,
'APL_SAMPLES','CUST_TRANSACTIONS', 'USER_APL','CUST_TO_SCORE',
'USER_APL','RECO_SCORE', out_log, out_summ, out_indic);
    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG" select * from :out_log;
    insert into  "USER_APL"."SUMMARY"       select * from :out_summ;
    insert into  "USER_APL"."INDICATORS"    select * from :out_indic;
    -- show result
    select * from "USER_APL"."RECO_SCORE" order by "UserID";
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
END;
```

8.2.10 SCORING_EQUATION

Creates a new regression/binary classification model, trains it, and exports its scoring equation in the specified format.

As EXPORT_PROFITCURVES [page 196], this function can generate the profit curve data.

DIRECT

Syntax

```
<WRAPPER_NAME>(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC, VARIABLE_ROLES, DATASET,
LOG, SUMMARY, INDICATORS, RESULT, PROFITCURVES )
```

In the direct technique, `PROFITCURVES` is optional. When you include the `PROFITCURVES` table, the `OPERATION_CONFIG` settings are different.

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::SCORING_EQUATION"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESC, VARIABLE_ROLES, DATASET_SCHEMA, DATASET_NAME, LOG, SUMMARY, INDICATORS,
RESULT,PROFITCURVES)
```

The parameters `DATASET_SCHEMA` and `DATASET_NAME` are strings.

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_SCORING_EQUATION"(FUNC_HEADER, OPERATION_CONFIG, VARIABLE_DESC,
VARIABLE_ROLES, DATASET, LOG, SUMMARY, INDICATORS, RESULT)
```

Syntax

```
"_SYS_AFL"."APL_SCORING_EQUATION__OVERLOAD_5_5"(FUNC_HEADER, OPERATION_CONFIG,
VARIABLE_DESC, VARIABLE_ROLES, DATASET, LOG, SUMMARY, INDICATORS, RESULT, PROFITCURVES)
```

8.2.10.1 Input/Output Tables

Direction	Type	Required	Description
IN	FUNC_HEADER [page 365]	No (The table must be provided but it can be empty.)	The function header
IN	OPERATION_CONFIG [page 384]	Yes	The configuration of the export operation. See the sections below. The settings for this parameter depend on whether you use the fifth (PROFITCURVES) parameter in the function declaration.
IN	VARIABLE_DESC [page 396]	No (The table must be provided but it can be empty.)	The variable descriptions for the input dataset, as expected by the Automated Analytics engine. If no variable description is provided (i.e. empty table), then the variable descriptions are automatically guessed by the Automated Analytics engine.
IN	VARIABLE_ROLES [page 398]	No (The table must be provided but it can be empty.)	The roles of the variables for the model training phase
IN	DATASET [page 362]	Yes	The training dataset
OUT	LOG [page 381]		The training log produced by the AutomatedAnalytics engine
OUT	SUMMARY [page 388]		The training summary
OUT	INDICATORS [page 370]		The variable statistics and indicators
OUT	RESULT [page 387]		The scoring equation
OUT	PROFITCURVES [page 386]		The profit curve data exported

8.2.10.2 OPERATION_CONFIG Parameters

All aliases supported by EXPORT_APPLY_CODE [page 183] and all Common APL Aliases for Model Training [page 336] are supported.

8.2.10.3 Example: DIRECT Without PROFITCURVES OUT Table

```
-- ============================================================
```

```
-- APL_AREA, SCORING_EQUATION using a binary format for the model
-- This script generates the scoring equation in SQL from an existing dataset
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
drop type CENSUS_T;
create type CENSUS_T as table (
    "id" INTEGER,
    "age" INTEGER,
    "workclass" NVARCHAR(16),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(12),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(21),
    "occupation" NVARCHAR(17),
    "relationship" NVARCHAR(14),
    "race" NVARCHAR(18),
    "sex" NVARCHAR(6),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(26),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table SCORING_EQUATION_SIGNATURE;
create column table SCORING_EQUATION_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into SCORING_EQUATION_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into SCORING_EQUATION_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into SCORING_EQUATION_SIGNATURE values (3, 'USER_APL','VARIABLE_DESC_T',
'IN');
insert into SCORING_EQUATION_SIGNATURE values (4, 'USER_APL','VARIABLE_ROLES_T',
'IN');
insert into SCORING_EQUATION_SIGNATURE values (5, 'USER_APL','CENSUS_T', 'IN');
insert into SCORING_EQUATION_SIGNATURE values (6, 'USER_APL','OPERATION_LOG_T',
'OUT');
insert into SCORING_EQUATION_SIGNATURE values (7, 'USER_APL','SUMMARY_T', 'OUT');
insert into SCORING_EQUATION_SIGNATURE values (8, 'USER_APL','INDICATORS_T',
'OUT');
insert into SCORING_EQUATION_SIGNATURE values (9, 'USER_APL','RESULT_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_SCORING_EQUATION');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','SCORING_EQUATION','USER_APL',
'APLWRAPPER_SCORING_EQUATION', SCORING_EQUATION_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table SCORING_EQUATION_CONFIG;
create table SCORING_EQUATION_CONFIG like OPERATION_CONFIG_T;
insert into SCORING_EQUATION_CONFIG values ('APL/CodeType', 'HANA');
insert into SCORING_EQUATION_CONFIG values ('APL/CodeTarget', 'class');
```

```
insert into SCORING_EQUATION_CONFIG values ('APL/CodeKey', '"id"');
insert into SCORING_EQUATION_CONFIG values ('APL/CodeSpace',
'APL_SAMPLES.CENSUS');
insert into SCORING_EQUATION_CONFIG values ('APL/ApplyExtraMode', 'No Extra');
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to make the engine guess the variable descriptions
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- variable roles are optional, hence the empty table
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
drop table SCORING_EQUATION;
create table SCORING_EQUATION like RESULT_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from SCORING_EQUATION_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.CENSUS;


APLWRAPPER_SCORING_EQUATION(:header, :config, :var_desc, :var_role, :dataset,
out_log, out_sum, out_indic, out_equation);

    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    insert into  "USER_APL"."SCORING_EQUATION" select * from :out_equation;
    -- show result
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."SCORING_EQUATION";
END;
```

8.2.10.4 Example: DIRECT with PROFITCURVES OUT Table

```
-- ============================================================
-- APL_AREA, SCORING_EQUATION, overloaded version with 5+5 parameters, using a
binary format for the model
-- This script generates the scoring equation in SQL from an existing dataset
-- This script also generates the profit curves associated with the underlying
model.
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: the APL table types have been created (see
apl_create_table_types.sql).
-- ------------------------------------------------------------------------
-- Create table type for the dataset
-- ------------------------------------------------------------------------
connect USER_APL password Password1;
-- Input table type: dataset
```

```
drop type CENSUS_T;
create type CENSUS_T as table (
    "id" INTEGER,
    "age" INTEGER,
    "workclass" NVARCHAR(16),
    "fnlwgt" INTEGER,
    "education" NVARCHAR(12),
    "education-num" INTEGER,
    "marital-status" NVARCHAR(21),
    "occupation" NVARCHAR(17),
    "relationship" NVARCHAR(14),
    "race" NVARCHAR(18),
    "sex" NVARCHAR(6),
    "capital-gain" INTEGER,
    "capital-loss" INTEGER,
    "hours-per-week" INTEGER,
    "native-country" NVARCHAR(26),
    "class" INTEGER
);
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table SCORING_EQUATION_SIGNATURE;
create column table SCORING_EQUATION_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into SCORING_EQUATION_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into SCORING_EQUATION_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
insert into SCORING_EQUATION_SIGNATURE values (3,  'USER_APL','VARIABLE_DESC_T',
'IN');
insert into SCORING_EQUATION_SIGNATURE values (4,
'USER_APL','VARIABLE_ROLES_T', 'IN');
insert into SCORING_EQUATION_SIGNATURE values (5,  'USER_APL','CENSUS_T', 'IN');
insert into SCORING_EQUATION_SIGNATURE values (6,  'USER_APL','OPERATION_LOG_T',
'OUT');
insert into SCORING_EQUATION_SIGNATURE values (7,  'USER_APL','SUMMARY_T',
'OUT');
insert into SCORING_EQUATION_SIGNATURE values (8,  'USER_APL','INDICATORS_T',
'OUT');
insert into SCORING_EQUATION_SIGNATURE values (9,  'USER_APL','RESULT_T', 'OUT');
insert into SCORING_EQUATION_SIGNATURE values (10, 'USER_APL','PROFITCURVE_T',
'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_SCORING_EQUATION');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','SCORING_EQUATION','USER_APL',
'APLWRAPPER_SCORING_EQUATION', SCORING_EQUATION_SIGNATURE);
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like FUNCTION_HEADER_T;
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table SCORING_EQUATION_CONFIG;
create table SCORING_EQUATION_CONFIG like OPERATION_CONFIG_T;
insert into SCORING_EQUATION_CONFIG values ('APL/CodeType', 'HANA');
insert into SCORING_EQUATION_CONFIG values ('APL/CodeTarget', 'class');
insert into SCORING_EQUATION_CONFIG values ('APL/CodeKey', '"id"');
insert into SCORING_EQUATION_CONFIG values ('APL/CodeSpace',
'APL_SAMPLES.CENSUS');
insert into SCORING_EQUATION_CONFIG values ('APL/ApplyExtraMode', 'No Extra');
drop table VARIABLE_DESC;
create table VARIABLE_DESC like VARIABLE_DESC_T;
-- let this table empty to make the engine guess the variable descriptions
drop table VARIABLE_ROLES;
```

```
create table VARIABLE_ROLES like VARIABLE_ROLES_T;
-- variable roles are optional, hence the empty table
drop table OPERATION_LOG;
create table OPERATION_LOG like OPERATION_LOG_T;
drop table SUMMARY;
create table SUMMARY like SUMMARY_T;
drop table INDICATORS;
create table INDICATORS like INDICATORS_T;
drop table SCORING_EQUATION;
create table SCORING_EQUATION like RESULT_T;
drop table PROFITCURVES_OUT;
create table PROFITCURVES_OUT like PROFITCURVE_T;
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from SCORING_EQUATION_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.CENSUS;


APLWRAPPER_SCORING_EQUATION(:header, :config, :var_desc, :var_role, :dataset,
out_log, out_sum, out_indic, out_equation, out_curves);

    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    insert into  "USER_APL"."SCORING_EQUATION" select * from :out_equation;
    insert into  "USER_APL"."PROFITCURVES_OUT" select * from :out_curves;
    -- show result
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."SCORING_EQUATION";
    select * from "USER_APL"."PROFITCURVES_OUT";
END;
```

8.2.10.5 Example: PROCEDURE Without PROFITCURVES OUT Table

```
-- ============================================================
-- APL_AREA, SCORING_EQUATION using a binary format for the model
-- This script generates the scoring equation in SQL from an existing dataset
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table SCORING_EQUATION_CONFIG;
create table SCORING_EQUATION_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
```

```
insert into SCORING_EQUATION_CONFIG values ('APL/CodeType', 'HANA',null);
insert into SCORING_EQUATION_CONFIG values ('APL/CodeTarget', 'class',null);
insert into SCORING_EQUATION_CONFIG values ('APL/CodeKey', '"id"',null);
insert into SCORING_EQUATION_CONFIG values ('APL/CodeSpace',
'APL_SAMPLES.CENSUS',null);
insert into SCORING_EQUATION_CONFIG values ('APL/ApplyExtraMode', 'No
Extra',null);
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to make the engine guess the variable descriptions
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
-- variable roles are optional, hence the empty table
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table SCORING_EQUATION;
create table SCORING_EQUATION like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.RESULT";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from SCORING_EQUATION_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;


"SAP_PA_APL"."sap.pa.apl.base::SCORING_EQUATION"(:header, :config, :var_desc, :va
r_role, 'APL_SAMPLES','CENSUS',  out_log, out_sum, out_indic,
out_equation,out_curves);

    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    insert into  "USER_APL"."SCORING_EQUATION" select * from :out_equation;
    -- show result
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."SCORING_EQUATION";
END;
```

8.2.10.6 Example: PROCEDURE with PROFITCURVES OUT Table

```
-- ================================================================
-- APL_AREA, SCORING_EQUATION, overloaded version with 5+5 parameters, using a
binary format for the model
-- This script generates the scoring equation in SQL from an existing dataset
-- This script also generates the profit curves associated to the underlyin
model.
-- Assumption 1: the users & privileges have been created & granted (see
apl_admin_ex.sql).
```

```
connect USER_APL password Password1;
-- ------------------------------------------------------------------------
-- Create the input/output tables used as arguments for the APL function
-- ------------------------------------------------------------------------
drop table FUNC_HEADER;
create table FUNC_HEADER like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
insert into FUNC_HEADER values ('Oid', '#42');
insert into FUNC_HEADER values ('LogLevel', '8');
insert into FUNC_HEADER values ('ModelFormat', 'bin');
drop table SCORING_EQUATION_CONFIG;
create table SCORING_EQUATION_CONFIG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTENDED";
insert into SCORING_EQUATION_CONFIG values ('APL/CodeType', 'HANA',null);
insert into SCORING_EQUATION_CONFIG values ('APL/CodeTarget', 'class',null);
insert into SCORING_EQUATION_CONFIG values ('APL/CodeKey', '"id"',null);
insert into SCORING_EQUATION_CONFIG values ('APL/CodeSpace',
'APL_SAMPLES.CENSUS',null);
insert into SCORING_EQUATION_CONFIG values ('APL/ApplyExtraMode', 'No
Extra',null);
drop table VARIABLE_DESC;
create table VARIABLE_DESC like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";
-- let this table empty to make the engine guess the variable descriptions
drop table VARIABLE_ROLES;
create table VARIABLE_ROLES like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH_COMPOSITES_OID";
-- variable roles are optional, hence the empty table
drop table OPERATION_LOG;
create table OPERATION_LOG like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_LOG";
drop table SUMMARY;
create table SUMMARY like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.SUMMARY";
drop table INDICATORS;
create table INDICATORS like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.INDICATORS";
drop table SCORING_EQUATION;
create table SCORING_EQUATION like "SAP_PA_APL"."sap.pa.apl.base::BASE.T.RESULT";
drop table PROFITCURVES_OUT;
create table PROFITCURVES_OUT like
"SAP_PA_APL"."sap.pa.apl.base::BASE.T.PROFITCURVE";
-- ------------------------------------------------------------------------
-- Execute the APL function using its AFL wrapper and the actual input/output
tables
-- ------------------------------------------------------------------------
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from SCORING_EQUATION_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;

"SAP_PA_APL"."sap.pa.apl.base::SCORING_EQUATION"(:header, :config, :var_desc, :va
r_role,'APL_SAMPLES','CENSUS',  out_log, out_sum, out_indic, out_equation,
out_curves);

    -- store result into table
    insert into  "USER_APL"."OPERATION_LOG"    select * from :out_log;
    insert into  "USER_APL"."SUMMARY"          select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"       select * from :out_indic;
    insert into  "USER_APL"."SCORING_EQUATION" select * from :out_equation;
    insert into  "USER_APL"."PROFITCURVES_OUT" select * from :out_curves;
    -- show result
    select * from "USER_APL"."OPERATION_LOG";
    select * from "USER_APL"."SUMMARY";
    select * from "USER_APL"."INDICATORS";
    select * from "USER_APL"."SCORING_EQUATION";
    select * from "USER_APL"."PROFITCURVES_OUT";
END;
```

8.3 Technical Functions

This group of functions contains technical functions only. Technical functions deal with the non-predictive aspects of SAP HANA APL, like version information.

8.3.1 PING

Gets the version and configuration information about the installation.

You must already have performed the following:

- The users and privileges have been created and granted (see apl_admin.sql).
- The APL table types have been created (see apl_create_table_types.sql).

Version and configuration information is returned as a collection of properties, i.e. a collection of string pairs {name, value}.

DIRECT

Syntax

```
<WRAPPER_NAME>(INFO)
```

PROCEDURE

Syntax

```
"SAP_PA_APL"."sap.pa.apl.base::PING"(INFO)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_AREA_PING_PROC"(INFO)
```

Input/Output Tables

Direction	Type	Description
OUT	INFO [page 380]	The configuration information.

Example

```
-- ============================================================
```

```
-- APL_AREA, PING
-- Assumption 1: The users & privileges have been created & granted (see
apl_admin.sql).
-- Assumption 2: The APL table types have been created (see
apl_create_table_types.sql).
connect USER_APL password Password1;
drop table PING_OUTPUT;
create table PING_OUTPUT like PING_OUTPUT_T;
DO BEGIN
    _SYS_AFL.APL_AREA_PING_PROC(ping);
    -- store result into table
    insert into  "USER_APL"."PING_OUTPUT"  select * from :ping;
    -- show result
    select * from PING_OUTPUT;
END;
```

8.3.2 PROGRESS_CLEANUP

Removes the progress messages from the `"_SYS_AFL"."FUNCTION_PROGRESS_IN_APL_AREA"` progress table.

DIRECT

Syntax

```
<WRAPPER_NAME>(EXECUTION_ID, VALUE)
```

ANY PROCEDURE

Syntax

```
"_SYS_AFL"."APL_AREA_PROGRESS_CLEANUP_PROC"(EXECUTION_ID, VALUE)
```

Note

This function is not available in the Stored Procedure technique.

Input/Output Parameters

Direction	Type	Description
IN	EXECUTION_ID	Execution ID of the messages to clean
OUT	VALUE	Dummy return value

Related Information

FUNC_HEADER [page 365]

8.4 Utility Stored Procedures

The `sap.pa.apl` package also contains utility stored procedures.

> **Note**
>
> These functions are only available for the Stored Procedure technique.

SAP_PA_APL Stored Procedure	Description
`"SAP_PA_APL"."sap.pa.apl.base::IMPORT_MODEL"`	Converts a Predictive Analytics native model into the APL bin model format, based on a `KXADMIN` table
`"SAP_PA_APL"."sap.pa.apl.base::CREATE_TABLE_FROM_TABLE_TYPE"`	Creates a table using the specified name and schema, from the supplied table type definition, as returned by `GET_TABLE_TYPE_FOR_APPLY`
`"SAP_PA_APL"."sap.pa.apl.base::CREATE_TYPE_FROM_TABLE_TYPE"`	Creates a table type using the specified name and schema, from the supplied table type definition, as returned by `GET_TABLE_TYPE_FOR_APPLY`
`"SAP_PA_APL"."sap.pa.apl.base::CLEANUP"`	Either purges the cache used by the stored procedures (input parameter = 1) or browses its contents (input parameter = 0)

Related Information

IMPORT_MODEL [page 224]
GET_TABLE_TYPE_FOR_APPLY [page 213]

8.5 Common APL Aliases for Model Training

Lists the aliases of the configuration parameters of the functions.

Most of the functions can be parameterized using configuration parameters. Each function has its own set of mandatory and optional parameters. These parameters can be expressed using either the full path of the parameter as in KXSHELL scripts or a so-called alias name, always prefixed by `APL/`, which allows an operation to be configured in a much simpler way. The following aliases can be used whenever a model is created or is going to be trained.

> **Note**
>
> For information about the aliases that can be used when a model is applied, see APPLY_MODEL [page 72].

Binary Classification/Regression Models

The following aliases are shared by the functions `CREATE_MODEL`, `CREATE_MODEL_AND_TRAIN`, `TRAIN_MODEL`, `KEY_INFLUENCERS` and `SCORING_EQUATION`. These aliases are optional.

Key	Supported Values	Default	Description
`APL/CorrelationsLowerBound`	Real number	`0.5`	Allows you to set a threshold to define if a correlation has to be displayed or not
`APL/CorrelationsMaxKept`	Positive integer	`1024`	Allows you to set the maximum number of displayed correlations. This parameter only accepts an unsigned integer.
`APL/CuttingStrategy`	• `'periodic'` • `'sequential'` • `'periodic with test at end'` • `'random with test at end'` • `'sequential with no test'` • `'periodic with no test'` • `'random with no test'`	`'random with no test'`	Partition strategy for the training dataset See Partition Strategies [page 64].
`APL/ExcludeLeakVariables`	• `'true'` • `'false'`	`'false'`	Allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.
`APL/ExcludeLowPredictiveConfidence`	• `'System'` • `'Enabled'` • `'Disabled'`	`'System'` the value (true/false) is automatically selected by the system	Indicates whether variables with a low predictive confidence (KR) must be excluded from the modeling
`APL/PolynomialDegree`	Positive integer	`1`	The degree of the polynomial model. This parameter is set by the user before the learning phase and cannot be changed later on for a given model.
`APL/RiskFitting`	• `'Frequency_Based'` • `'PDO_Based'`	`'Frequency_Based'`	Allows you to control the way risk score fitting is performed

Key	Supported Values	Default	Description
APL/RiskFittingMinCumulatedFrequency	Positive integer	15	When RiskFitting is set to 'Frequency_Based', indicates the frequency of extreme scores to be skipped
APL/RiskFittingNbPDO	Positive integer	2	When RiskFitting is set to 'PDO_Based'
APL/RiskFittingUseWeights	• 'true' • 'false'	'false'	Indicates whether to use score bin frequency as weights
APL/RiskGDO	Real number	9	Allows you to specify a high probability that will be associated with a high score
APL/RiskMode	• 'true' • 'false'	'false'	Allows you to activate the risk mode for a classification model. It allows advanced users to ask a binary classification model to translate its internal equation obtained with no constraints into a specified range of scores associated with a specific initial score. When this mode is activated, the different types of encoding that are used internally for continuous and ordinal variables are merged in a single representation, allowing a simpler view of the model internal equations. To use this mode, you need to choose a range of scores associated with probabilities.
APL/RiskPDO	Real number	15	Allows you to specify the low score associated with the low probability
APL/RiskScore	Real number	615	Allows you to specify a low probability that will be associated with a low score
APL/ScoreBinsCount	Positive integer	20	The number of bins for score variables
APL/TargetKey	Binary target	The least frequent category of the Target variable	The category of the Target variable that will be used as the positive target in the model (binary classification models only)
APL/VariableAutoSelection	• 'true' • 'false'	'false'	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.

Key	Supported Values	Default	Description
APL/VariableSelectionBestIteration	• 'true' • 'false'	'true'	Indicates which model of the variable selection process will be used. Usually the best model is the one before last, however the quality of the last model can be sufficient for your needs and you may want to use it instead.
APL/VariableSelectionMinNbOfFinalVariables	Positive integer	1	Allows you to set the minimum number of variables to keep in the final model
APL/VariableSelectionMaxNbOfFinalVariables	Positive integer	-1 for all variables	Allows you to set the maximum number of variables to keep in the final model. APL/VariableAutoSelection must be set to true.
APL/VariableSelectionMode	• 'ContributionBased' • 'VariableBased'	'ContributionBased'	Allows you to specify the type of automatic process variable selection
APL/VariableSelectionNbVariablesRemovedByStep	Positive integer	1	Allows you to specify, in the case of the automatic variable selection is in VariableBased, the number of skipped variables by iteration process
APL/VariableSelectionPercentageOfContributionKeptByStep	Real number	0.95	Allows you to specify the percentage amount of information to keep
APL/VariableSelectionQualityBar	Real number	0.05	Allows you to specify the quality loss by iteration. A higher value indicates less strict selection criteria, generally resulting in more iterations and fewer resulting variables.
APL/VariableSelectionQualityCriteria	• 'None' • 'KiKr' • 'Ki''Kr'	'KiKr'	Allows you to set the type of quality criteria to be used for the automatic variables selection

Binary Classification Models (Gradient Boosting)

These aliases are optional.

Key	Supported Values	Default	Description
`APL/CorrelationsLowerBound`	Real number	`0.5`	Allows you to set a threshold to define if a correlation has to be displayed or not
`APL/CorrelationsMaxKept`	Positive integer	`1024`	Allows you to set the maximum number of displayed correlations. This parameter only accepts an unsigned integer.
`APL/EarlyStoppingPatience`	Positive integer	`10`	If the performance did not improve after `EarlyStoppingPatience` iterations, there is no need to make further attempts to improve the model and so the training stops before reaching `MaxIterations`.
`APL/EvalMetric`	• `'LogLoss'` • `'AUC'` • `'ClassificationError'`	`'LogLoss'`	List of metrics used to evaluate the model performance on the validation dataset along the boosting iterations. You can define multiple metrics separated by commas, like `'LogLoss,AUC'`. Only the last metric is considered by the early stopping feature.
`APL/ExcludeLeakVariables`	• `'true'` • `'false'`	`'false'`	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.
`APL/Interactions`	• `'true'` • `'false'`	`'false'`	Allows the generation of variable interactions. This alias is only relevant for the Variable Interaction Matrix report. It has no effect on the Variable Interaction Effect report. **Note** Performance can be affected when interactions are generated. Depending on the number of variables and target categories (for multiclass classification), the calculation of variable interactions can lead to very long processing times.

Key	Supported Values	Default	Description
APL/InteractionsMaxKept	Integer	5	The number of highest interactions to be output for each variable. This alias is only relevant for the Variable Interaction Matrix report. It has no effect on the Variable Interaction Effect report.
APL/LearningRate	Real number	0.1	This weight parameter controls the model regularization to avoid the well-known overfitting risk. A small value improves the model generalization to unseen datasets at the expense of the computational cost.
APL/MaxDepth	Positive integer	4	Maximum depth of the decision tree that is added as a base learner to the model at each boosting iteration
APL/MaxIterations	Positive integer	1000	Maximum number of boosting iterations to fit the model with the training dataset
APL/NumberOfJobs	Positive integer	4	Number of threads allocated to the model training and apply parallelization. The working threads are not under control of the SAP HANA workload class management. We recommend that you compute the maximum number of threads you want to allocate to SAP HANA APL as follows: the number of concurrent requests allowed multiplied by the maximum number of threads per request. The maximum number of threads is 8.
APL/PredictionOutputMaxSize	Integer	100	The maximum memory size in MB of the prediction output. With interactions, the standard memory footprint is multiplied by the number of variables, which can lead to very high memory usage if there are 100 variables.
APL/Retrain	• 'Auto' • 'Always' • 'Never'	'Auto'	Activates the retraining of the model on the whole training set (estimation and validation) using the best parameters found. 'Auto' activates the retraining on small datasets only to avoid a large increase in training time.
APL/VariableAutoSelection	• 'true' • 'false'	'false'	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.
APL/VariableSelectionMaxIterations	Positive integer	2	Allows you to specify the maximum number of iterations in the variable selection process.

Key	Supported Values	Default	Description
`APL/VariableSelectionMaxNbOfFinalVariables`	Positive integer	`-1` for all variables	Allows you to set the maximum number of variables to keep in the final model. `APL/VariableAutoSelection` must be set to true.
`APL/VariableSelectionPercentageOfContributionKeptByStep`	Real number	`0.90`	Allows you to specify the percentage amount of information to keep
`APL/VariableSelectionQualityBar`	Real number	`0.01`	Allows you to specify the maximum absolute performance (Area Under the ROC Curve) difference between the initial model (with all input variables) and the model resulting from the variable selection process. A higher value leads to more iterations and fewer resulting variables.

Regression Models (Gradient Boosting)

These aliases are optional.

Key	Supported Values	Default	Description
`APL/CorrelationsLowerBound`	Real number	`0.5`	Allows you to set a threshold to define if a correlation has to be displayed or not
`APL/CorrelationsMaxKept`	Positive integer	`1024`	Allows you to set the maximum number of displayed correlations. This parameter only accepts an unsigned integer.
`APL/EarlyStoppingPatience`	Positive integer	`10`	If the performance did not improve after `EarlyStoppingPatience` iterations, there is no need to make further attempts to improve the model and so the training stops before reaching `MaxIterations`.

Key	Supported Values	Default	Description
APL/EvalMetric	• 'MAE' • 'MeanPseudoHuberError' • 'RMSE' • 'TweedieNegativeLogLikelihood'	'RMSE'	List of metrics used to evaluate the model performance on the validation dataset along the boosting iterations. You can define multiple metrics separated by commas, like 'MAE,RMSE'. Only the last metric is considered by the early stopping feature.
APL/ExcludeLeakVariables	• 'true' • 'false'	'false'	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator VariableExclusionReason with the value of 'Leak Variable'.
APL/Interactions	• 'true' • 'false'	'false'	Allows the generation of variable interactions. This alias is only relevant for the Variable Interaction Matrix report. It has no effect on the Variable Interaction Effect report. **Note** Performance can be affected when interactions are generated. Depending on the number of variables and target categories (for multiclass classification), the calculation of variable interactions can lead to very long processing times.
APL/InteractionsMaxKept	Integer	5	The number of highest interactions to be output for each variable. This alias is only relevant for the Variable Interaction Matrix report. It has no effect on the Variable Interaction Effect report.
APL/LearningRate	Real number	0.1	This weight parameter controls the model regularization to avoid the well-known overfitting risk. A small value improves the model generalization to unseen datasets at the expense of the computational cost.
APL/MaxDepth	Positive integer	4	Maximum depth of the decision tree that is added as a base learner to the model at each boosting iteration
APL/MaxIterations	Positive integer	1000	Maximum number of boosting iterations to fit the model with the training dataset

Key	Supported Values	Default	Description
APL/NumberOfJobs	Positive integer	4	Number of threads allocated to the model training and apply parallelization. The working threads are not under control of the SAP HANA workload class management. We recommend that you compute the maximum number of threads you want to allocate to SAP HANA APL as follows: the number of concurrent requests allowed multiplied by the maximum number of threads per request. The maximum number of threads is 8.
APL/Objective	• 'reg:pseudohubererror' • 'reg:squarederror' • 'reg:tweedie'	'reg:squarederror'	Objective (loss) function minimized by the gradient boosting algorithm
APL/PredictionOutputMaxSize	Integer	100	The maximum memory size in MB of the prediction output. With interactions, the standard memory footprint is multiplied by the number of variables, which can lead to very high memory usage if there are 100 variables.
APL/Retrain	• 'Auto' • 'Always' • 'Never'	'Auto'	Activates the retraining of the model on the whole training set (estimation and validation) using the best parameters found. 'Auto' activates the retraining on small datasets only to avoid a large increase in training time.
APL/TweedieVariancePower	Real number in the range [1,2[	1.5	Parameter that controls the variance of the Tweedie distribution
APL/VariableAutoSelection	• 'true' • 'false'	'false'	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.
APL/VariableSelectionMaxIterations	Positive integer	2	Allows you to specify the maximum number of iterations in the variable selection process.
APL/VariableSelectionMaxNbOfFinalVariables	Positive integer	-1 for all variables	Allows you to set the maximum number of variables to keep in the final model. APL/VariableAutoSelection must be set to true.

Key	Supported Values	Default	Description
`APL/VariableSelectionPercentageOfContributionKeptByStep`	Real number	`0.90`	Allows you to specify the percentage amount of information to keep
`APL/VariableSelectionQualityBar`	Real number	`0.01`	Allows you to specify the maximum relative performance (Root Mean Squared Error) difference between the initial model (with all input variables) and the model resulting from the variable selection process. A higher value leads to more iterations and fewer resulting variables.

Multinomial Classification Models

These aliases are optional.

Key	Supported Values	Default	Description
`APL/EarlyStoppingPatience`	Positive integer	`10`	If the performance did not improve after `EarlyStoppingPatience` iterations, there is no need to make further attempts to improve the model and so the training stops before reaching `MaxIterations`.
`APL/EvalMetric`	• `'MultiClassClassificationError'` • `'MultiClassLogLoss'`	`'MultiClassLogLoss'`	List of metrics used to evaluate the model performance on the validation dataset along the boosting iterations. You can define multiple metrics separated by commas, like `'MultiClassClassificationError,MultiClassLogLoss'`. Only the last metric is considered by the early stopping feature.
`APL/ExcludeLeakVariables`	• `'true'` • `'false'`	`'false'`	Exclude Leak Variables allows you to exclude the perfect variables from the model. Perfect variables are equivalent to the target (Ki>0.98 and KR>0.9). When leak variables are excluded, APL includes the excluded variables in the ClassificationRegression_VariableExclusion report and the indicator `VariableExclusionReason` with the value of `'Leak Variable'`.

Key	Supported Values	Default	Description
APL/Interactions	• 'true' • 'false'	'false'	Allows the generation of variable interactions. This alias is only relevant for the Variable Interaction Matrix report. It has no effect on the Variable Interaction Effect report. **Note** Performance can be affected when interactions are generated. Depending on the number of variables and target categories (for multiclass classification), the calculation of variable interactions can lead to very long processing times.
APL/InteractionsMaxKept	Integer	5	The number of highest interactions to be output for each variable. This alias is only relevant for the Variable Interaction Matrix report. It has no effect on the Variable Interaction Effect report.
APL/LearningRate	Real number	0.1	This weight parameter controls the model regularization to avoid the well-known overfitting risk. A small value improves the model generalization to unseen datasets at the expense of the computational cost.
APL/MaxDepth	Positive integer	4	Maximum depth of the decision tree that is added as a base learner to the model at each boosting iteration
APL/MaxIterations	Positive integer	1000	Maximum number of boosting iterations to fit the model with the training dataset
APL/NumberOfJobs	Positive integer	4	Number of threads allocated to the model training and apply parallelization. The working threads are not under control of the SAP HANA workload class management. We recommend that you compute the maximum number of threads you want to allocate to SAP HANA APL as follows: the number of concurrent requests allowed multiplied by the maximum number of threads per request. The maximum number of threads is 8.
APL/PredictionOutputMaxSize	Integer	100	The maximum memory size in MB of the prediction output. With interactions, the standard memory footprint is multiplied by the number of variables, which can lead to very high memory usage if there are 100 variables.

Key	Supported Values	Default	Description
APL/Retrain	• 'Auto' • 'Always' • 'Never'	'Auto'	Activates the retraining of the model on the whole training set (estimation and validation) using the best parameters found. 'Auto' activates the retraining on small datasets only to avoid a large increase in training time.
APL/VariableAutoSelection	• 'true' • 'false'	'false'	Auto-selection allows you to automatically reduce the number of variables in the model in relation to quality criteria.
APL/VariableSelectionMaxIterations	Positive integer	2	Allows you to specify the maximum number of iterations in the variable selection process.
APL/VariableSelectionMaxNbOfFinalVariables	Positive integer	-1 for all variables	Allows you to set the maximum number of variables to keep in the final model. APL/VariableAutoSelection must be set to true.
APL/VariableSelectionPercentageOfContributionKeptByStep	Real number	0.90	Allows you to specify the percentage amount of information to keep
APL/VariableSelectionQualityBar	Real number	0.01	Allows you to specify the maximum absolute performance (Balanced Classification Rate) difference between the initial model (with all input variables) and the model resulting from the variable selection process. A higher value leads to more iterations and fewer resulting variables.

Clustering Models

The following aliases are shared by the functions CREATE_MODEL, CREATE_MODEL_AND_TRAIN, TRAIN_MODEL, KEY_INFLUENCERS and SCORING_EQUATION. These aliases are optional.

Key	Supported Values	Default	Description
APL/CalculateCrossStatistics	• 'disabled' • 'enabled'	'enabled'	Allows you to enable/disable the computation of cross statistics between cluster ID and input variables
APL/CalculateSQLExpressions	• 'true' • 'false' 'enabled' and 'disabled' are also supported for backward compatibility reasons.	'false'	Generates a SQL expression for the clustering model
APL/CuttingStrategy	• 'periodic' • 'sequential' • 'periodic with test at end' • 'random with test at end' • 'sequential with no test' • 'periodic with no test' • 'random with no test'	'random with no test'	Partition strategy for the training dataset See Partition Strategies [page 64].
APL/Distance	• 'L1' • 'L2' • 'LInf' • 'SystemDetermined'	'SystemDetermined'	Allows you to fix the distance used to compare encoded input data
APL/EncodingStrategy	• 'Uniform' • 'Unsupervised' • 'TargetMean' • 'Natural' • 'SystemDetermined' • 'MinMax' • 'MinMaxCentered' • 'StdDevNormalization'	'Unsupervised' for unsupervised clustering and 'TargetMean' for supervised clustering	Allows you to drive the kind of encoding expected from Data Encoding
APL/NbClusters	Positive integer	10	Number of clusters for the model
APL/NbClustersMin	Positive integer		Cluster count range customization, lower number of clusters. Overrides APL/NbClusters.
APL/NbClustersMax	Positive integer		Cluster count range customization, upper number of clusters. Overrides APL/NbClusters.

Time Series Models

The following aliases are shared by the functions `CREATE_MODEL`, `CREATE_MODEL_AND_TRAIN`, `TRAIN_MODEL`, and `FORECAST`.

Key	Required	Supported Values	Default	Description
`APL/CuttingStrategy`	No	• `'sequential'` • `'sequential with no test'`	`'sequential with no test'`	Partition strategy for the training dataset See Partition Strategies [page 64].
`APL/ForceNegativeForecast`	No	• `'true'` • `'false'`	`'false'`	Activates a mode where the positive forecasts are ignored, that is, replaced with zero
`APL/ForcePositiveForecast`	No	• `'true'` • `'false'`	`'false'`	Activates a mode where the negative forecasts are ignored, that is, replaced with zero
`APL/ForecastFallbackMethod`	No	• `'LinearRegression'` • `'ExponentialSmoothing'`	`'ExponentialSmoothing'`	Sets the method that will be used if the one specified with `APL/ForecastMethod` fails, for example when there are too few data points
`APL/ForecastMaxCyclics`	No	Positive integer	`200`	Length of the longest cycle the model will try to detect. Controls the way that the model analyzes the periodicities in the signal. Also limited by the size of the training dataset. You can disable the cyclic analysis by setting this parameter to `0`.
`APL/ForecastMaxLags`	No	Positive integer	Default is quarter of the training dataset size	Defines the maximum dependency of the signal on its own past values. Controls the way that the model analyzes the random fluctuations in the signal. You can set this parameter to `0` to disable the fluctuations analysis.
`APL/ForecastMethod`	No	• `'Default'` • `'LinearRegression'` • `'ExponentialSmoothing'`	`'Default'`	Uses a forecast method different from the default Automated Analytics time-series algorithm
`APL/Horizon`	Yes			Number of forecast time points

Key	Required	Supported Values	Default	Description
`APL/LastRowWithExtraPredictable`	No	Positive integer		Sets the last row in the dataset that has all forecasting information. Is necessary if the dataset has extra predictable variables or rows where the target is not set.
`APL/LastTrainingTimePoint`	No	A date of which format is YYYY:MM:DD HH:mm:ss		Value in the time point column which represents the last date for the training dataset. This alias is required if your input dataset has extra predictable variables or rows where the target is not set.
`APL/SmoothingCycleLength`	No	[1..n] with n = integer value		Sets the cycle/seasonal length to be used for the smoothing instead of the cycle length candidates automatically determined by the Automated Analytics engine based on the time granularity, for example: month -> 4 (quarterly) or 12 (yearly).
`APL/TimePointColumnName`	No			Name of the column in the dataset that contains the time points of the time-series
`APL/WithExtraPredictable`	No	• `'true'` • `'false'`	`'true'`	Adds all input variables as extra predictable variables

Statbuilder Model

This parameter is used by the function `DEBRIEF_APPLY_RESULT`. It is mandatory.

Key	Supported Values	Default	Description
`APL/VariableEstimatorOf`	Concatenated string with a semi-colon like';' like `'<variable_name>;<target_name>'`		An input variable as estimator of a target variable

All Models

These parameters are optional.

Key	Supported Values	Default	Description
APL/IndicatorDataset	• 'Estimation' • 'Validation' • 'Test'		The dataset to be used to retrieve statistics **Note** You need an INDICATORS [page 370] table as output to retrieve statistics.
APL/LastTrainingRow	Integer	Default is to use all dataset rows.	The last valid row that SAP HANA APL identifies as being part of the training dataset
APL/UsePhysicalNameForApply	• 'true' • 'false'	'false'	Uses the input variable's physical name instead of its logical name to create the output variables's physical name
APL/UseTrainPhysicalNameBindingForApplyIn	• 'true' • 'false'	'false'	Forces the same set of physical names that were used for the variables during the training phase to be reused in the ApplyIn dataset
APL/UseUppercaseForSmartOutput	• 'true' • 'false'	'false'	Converts the output variable's name to uppercase

Key	Supported Values	Default	Description
`APL/VariableConverter`	No	• `'abap_converter'` • `'default'`	`'abap_converter'` indicates that the following variables with an ABAP data type are converted into variables with a standard ISO 8601 format: • ABAP date (DATS) • ABAP time (TIMS) • ABAP short timestamp (TIMESTAMP/TZNTSTMPS) • ABAP long timestamp (TIMESTAMPL/TZNTSTMPL) Conversion is done during training or apply. See APPLY_MODEL [page 72]. `'default'` indicates that the guessed variable storage is kept as is in the model. **Note** To override the default behavior, you can set this alias to a specific variable by using the 3-column structure of the OPERATION_CONFIG [page 384] table type, for example: `insert into CREATE_AND_TRAIN_CONFIG values ('APL/VariableConverter','abap_converter', '<VariableName>');`

9 Table Type Reference

All the input and output parameters of the functions are either SQL tables or SQL views.

The tables used as function parameters must comply with the table types and adhere to the following rules:

- The expected columns must be available.
- The columns must use the expected names.
- The column names must not have blank characters to avoid errors when generating the AFL wrapper.
- The columns must have the expected data types or compatible data types.

Each table type reference describes the expected table structure with the following information:

- Column name
- SQL data type of the column
- Description of the column

> → Remember
>
> SAP recommends you rely on the predefined table types to manage the lifecycle of the parameters. The SAP HANA APL package contains sample SQL scripts that can help setting up the full set of predefined table types.

Related Information

Package Contents
Table Types [page 45]

9.1 ADMIN

The structure expected for tables of the main Predictive Analytics metadata repository type `KXADMIN`, which can be input or output.

Column	SQL Data Type	Description
`NAME`	VARCHAR(255)	Name of the model
`Class`	VARCHAR(255)	Type of model
`CLASSVERSION`	DOUBLE	Version of the model type
`SPACENAME`	VARCHAR(255)	Name of the table in which the model is stored

Column	SQL Data Type	Description
VERSION	INT	Version of the model
CREATIONDATE	LONGDATE	Date of creation
Comment	VARCHAR(255)	An optional description

9.2 APPLY_OUT_TYPE

The structure expected for tables of type APPLY_OUT_TYPE.

The recommended and correct way to get the schema of the APPLY_OUT table is to use the GET_TABLE_TYPE_FOR_APPLY function. In this section we provide 'as is' a set of rules to determine the schema of the APPLY_OUT table in some very basic scenarios. These rules are not complete, they cannot be used for all kinds of scenarios and APL/ApplyExtraMode parameters. These rules might not be valid for complex scenarios with several targets of different types. If you are unsure about the rules or the results you are getting, use GET_TABLE_TYPE_FOR_APPLY to get the correct APPLY_OUT table type.

Expected Table Structure for Binary Classification with One or More Nominal Targets (Two Possible Values Only)

Column	SQL Data Type	Description
<id1>	SQL type of <id1>	The first ID column in the dataset. KxIndex if there is none.
<id2>	SQL type of <id2>	The second ID column in the dataset. Skip the column if there is only one or no ID.
...	...	As many ID columns as there are IDs in the dataset, as specified in the variable description.
<target1>	SQL type of <target1>	The first target column in the dataset.
<target2> (if there is more than one target)	SQL type of <target2>	The second target column in the dataset. Skip this column if there is only one target.
...	...	As many target columns as there are targets in the dataset, as specified in the variable description.
rr_<target1>	DOUBLE	The predicted value of the row for <target1>, according to the model's prediction.

Column	SQL Data Type	Description
`rr_<target2>` (if there is more than one target)	DOUBLE	The predicted value of the row for `<target2>`, according to the model's prediction. Skip this column if there is only one target.
...	...	As many `rr_<targetn>` columns as there are targets in the dataset, as specified in the variable description.
`decision_rr_<target1>`	SQL of type `<target1>`	The predicted value of the row for `<target1>`, according to the model's prediction.
`proba_decision_rr_<target1>`	DOUBLE	The probability value associated with the classification decision. This column is in the table if the parameter used is `'APL/ApplyProbaDecision'`.
`decision_rr_<target2>` (if there is more than one target)	SQL of type `<target2>`	The predicted value of the row for `<target2>`, according to the model's prediction. Skip this column if there is only one target.
`proba_decision_rr_<target2>` (if there is more than one target)	DOUBLE	The probability value associated with the classification decision. Skip this column if there is only one target.
...	...	As many `decision_rr_<targetn>` as there are targets in the dataset, as specified in the variable description.
...	DOUBLE	As many `proba_decision_rr_<targetn>` as there are targets in the dataset, as specified in the variable description.
`proba_rr_<TargetName>`	DOUBLE	The probability of the row being a positive target. This column is in the table if `APL/ApplyExtraMode` is `'Probability'`.

Expected Table Structure for Binary Classification (Gradient Boosting)

Column	SQL Data Type	Description
`<id1>`	SQL type of `<id1>`	The first ID column in the dataset. `KxIndex` if there is none.
`<id2>`	SQL type of `<id2>`	The second ID column in the dataset. Skip the column if there is only one or no ID.

Column	SQL Data Type	Description
...	...	As many ID columns as there are IDs in the dataset, as specified in the variable description.
`<TargetName>`	SQL type of the target	The target column in the dataset.
`gb_score_<TargetName>`	DOUBLE	The score of the target class. This column is in the table if `APL/ApplyExtraMode` is `'Score'`.
`gb_proba_<TargetName>`	DOUBLE	The probability of the row being a positive target. This column is in the table if `APL/ApplyExtraMode` is `'Probability'`.
`gb_decision_<TargetName>`	SQL of type the target	The classification decision by considering the class having the highest prediction probability. This column is in the table if `APL/ApplyExtraMode` is `'Decision'`.
`gb_best_proba_<TargetName>`	DOUBLE	The probability value associated with the classification decision. This column is in the table in addition to the `gb_decision_<TargetName>` column if `APL/ApplyExtraMode` is `'BestProbabilityAndDecision'`.
`gb_contrib_<VariableName>`	DOUBLE	The contribution of each variable to the score. This column is in the table if `APL/ApplyExtraMode` is `'Individual Contributions'`.
`gb_contrib_constant_bias`	DOUBLE	The constant base value of the score. This column is in the table if `APL/ApplyExtraMode` is `'Individual Contributions'`.

Expected Table Structure for Clustering with Default APL/ApplyExtraMode ('No Extra') and No Target

Column	SQL Data Type	Description
`<id1>`	SQL type of `<id1>`	The first ID column in the dataset (`KxIndex` if there is none).
`<id2>`	SQL type of `<id2>`	The second ID column in the dataset. Skip the column if there is only one or no ID.

Column	SQL Data Type	Description
...	...	As many ID columns as there are IDs in the dataset, as specified in the variable description.
kc_<clusterID>	INTEGER	The cluster number that the row belongs to, according to the model's prediction.

Expected Table Structure for Clustering with Default APL/ApplyExtraMode ('No Extra') and One or More Targets

Column	SQL Data Type	Description
<id1>	SQL type of <id1>	The first ID column in the dataset. KxIndex if there is none.
<id2>	SQL type of <id2>	The second ID column in the dataset. Skip the column if there is only one or no ID.
...	...	As many ID columns as there are IDs in the dataset, as specified in the variable description.
<target1>	SQL type of <target1>	The first target column in the dataset.
<target2> (if there is more than one target)	SQL type of <target2>	The second target column in the dataset. Skip this column if there is only one target.
...	...	As many target columns as there are targets in the dataset, as specified in the variable description.
kc_<target1>	INTEGER	The cluster number that the row belongs to for <target1>, according to the model's prediction.
kc_<target2> (if there is more than one target)	INTEGER	The cluster number that the row belongs to for <target2>, according to the model's prediction. Skip this column if there is only one target.
...	INTEGER	As many kc_<targetn> columns as there are targets in the dataset, as specified in the variable description.

Expected Table Structure for Multinomial Classification

Column	SQL Data Type	Description
`<id1>`	SQL type of `<id1>`	The first ID column in the dataset. `KxIndex` if there is none.
`<id2>`	SQL type of `<id2>`	The second ID column in the dataset. Skip the column if there is only one or no ID.
...	...	As many ID columns as there are IDs in the dataset, as specified in the variable description.
`<TargetName>`	SQL type of the target	The target column in the dataset.
`gb_score_<TargetName>_<Class Name>`	DOUBLE	The score/margin value of the target class. This column is in the table if `APL/ApplyExtraMode` is `'Score'`.
...	...	As many `gb_score_<TargetName>_<Class Name>` columns as there are classes.
`gb_proba_<TargetName>_<Class Name>`	DOUBLE	The probability of the row being a positive target. This column is in the table if `APL/ApplyExtraMode` is `'Probability'`.
...	...	As many `gb_proba_<TargetName>_<Class Name>` columns as there are classes.
`gb_decision_<TargetName>`	SQL type of the target	The classification decision by considering the class having the highest prediction probability. This column is in the table if `APL/ApplyExtraMode` is `'Decision'` or `'BestProbabilityAndDecision'`.
`gb_best_proba_<TargetName>`	DOUBLE	The probability value associated with the classification decision. This column is in the table in addition to the `gb_decision_<TargetName>` column if `APL/ApplyExtraMode` is `'BestProbabilityAndDecision'`.

Expected Table Structure for Regression (Gradient Boosting) with Default APL/ApplyExtraMode ('Score')

Column	SQL Data Type	Description
<id1>	SQL type of <id1>	The first ID column in the dataset. KxIndex if there is none.
<id2> (if there is more than one ID column in the input table)	SQL type of <id2>	The second ID column in the dataset. Skip the column if there is only one or no ID.
...	...	As many ID columns as there are IDs in the dataset, as specified in the variable description.
<TargetName>	SQL type of the target	The target column in the dataset.
gb_score_<TargetName>	SQL type of the target	The predicted value of the row for the target, according to the model's prediction.

Expected Table Structure for Regression (Legacy) with Default APL/ApplyExtraMode ('No Extra') and One or More Continuous Targets

Column	SQL Data Type	Description
<id1>	SQL type of <id1>	The first ID column in the dataset. KxIndex if there is none.
<id2>	SQL type of <id2>	The second ID column in the dataset. Skip the column if there is only one or no ID.
...	...	As many ID columns as there are IDs in the dataset, as specified in the variable description.
<target1>	SQL type of <target1>	The first target column in the dataset.
<target2>	SQL type of <target2>	The second target column in the dataset. Skip this column if there is only one target.
...	...	As many target columns as there are targets in the dataset, as specified in the variable description.
rr_<target1>	SQL type of <target1>	The predicted value of the row for <target1>, according to the model's prediction.

Column	SQL Data Type	Description
`rr_<target2>`	SQL type of `<target2>`	The predicted value of the row for `<target2>`, according to the model's prediction. Skip this column if there is only one target.
...	...	As many `rr_<targetn>` columns as there are targets in the dataset, as specified in the variable description.

Expected Table Structure for Time-Series with Default APL/ApplyExtraMode ('No Extra')

Column	SQL Data Type	Description
`Date`	SQL type of `Date`	The date you want to learn on/predict the values for. Usually this is also the ID column.
`<id1>`	SQL type of `<id1>`	The first ID column in the dataset. Skip the column if `date` is the ID.
...	...	As many ID columns as there are IDs in the dataset, as specified in the variable description.
`<TargetName>`	SQL type of the target	The target column you want to predict.
`kts_1`	SQL type of the target	The predicted value of the row for the target, according to the model's prediction.
`kts_2`	SQL type of the target	An internal column to compute the prediction. Skip this column if HORIZON = 1.
...	...	As many `kts_<n>` columns as you have specified HORIZONs in your model.

Expected Table Structure for Time-Series FORECAST Function with Default APL/ApplyExtraMode ('No Extra')

Column	SQL Data Type	Description
`Date`	SQL type of `Date`	The date you want to learn on/predict the values for. Usually this is also the ID column.
`<id1>`	SQL type of `<id1>`	The first ID column in the dataset. Skip the column if `Date` is the ID.
...	...	As many ID columns as there are IDs in the dataset, as specified in the variable description.
`<TargetName>`	SQL type of the target	The target column you want to predict.
`kts_1`	SQL type of the target	The predicted value of the row for the target, according to the model's prediction.

Related Information

Quantiles of Predicted Values [page 68]

9.3 ASSOCIATION_RULES

The structure expected for tables of type `ASSOCIATION_RULES`.

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	Operation ID
`RULE_ID`	INTEGER	Rule ID
`ANTECEDENT_ITEM_ID1`	INTEGER	Item ID #1 in the antecedent of the rule. This ID is an foreign key pointing to the Rule Items table.
`ANTECEDENT_ITEM_ID2`	INTEGER	Item ID #2 in the antecedent of the rule. This ID is an foreign key pointing to the Rule Items table.
`ANTECEDENT_ITEM_ID3`	INTEGER	Item ID #3 in the antecedent of the rule. This ID is an foreign key pointing to the Rule Items table.

Column	SQL Data Type	Description
CONSEQUENT_ID	INTEGER	Consequent ID of the rule. This ID is an foreign key pointing to the Rule Items table.
RULE_KI	DOUBLE	Rule KI
RULE_LIFT	DOUBLE	Rule lift
RULE_CONFIDENCE	DOUBLE	Rule confidence
RULE_SUPPORT	INTEGER	Rule support
RULE_SIZE	INTEGER	Rule size
ANTECEDENT_SUPPORT	INTEGER	Antecent support
CONSEQUENT_SUPPORT	INTEGER	Consequent support

9.4 CONTINUOUS_GROUPS

The structure expected for tables of type CONTINUOUS_GROUPS (groups for continuous targets).

Column	SQL Data Type	Description
OID	(N)VARCHAR, (N)CLOB	The operation ID (OID). If no OID was initially specified, then a new one is generated.
VARIABLE	(N)VARCHAR, (N)CLOB	Variable name, i.e. the influencer name
TARGET	(N)VARCHAR, (N)CLOB	Target name
GROUP	(N)VARCHAR, (N)CLOB	Group name
HIGHERBOUND	(N)VARCHAR, (N)CLOB	Higher bound value
HIGHERBOUND_IN	INTEGER	1 = higher bound is included, otherwise 0
LOWERBOUND	(N)VARCHAR, (N)CLOB	Lower bound value
LOWERBOUND_IN	INTEGER	1 = lower bound is included, otherwise 0

9.5 DATASET

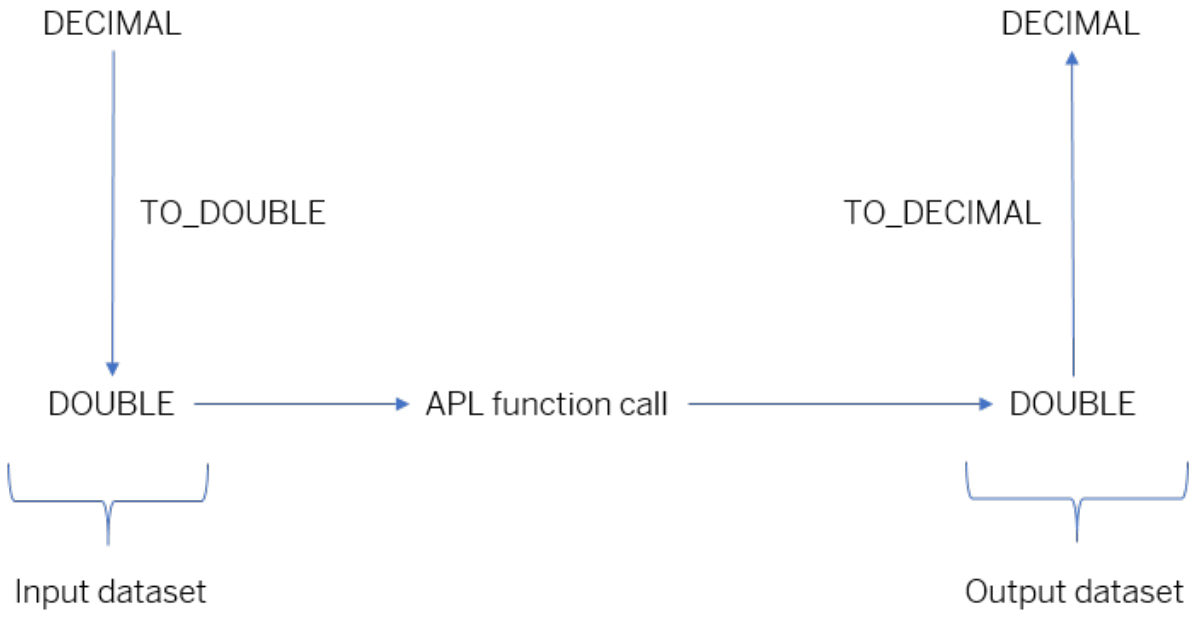
The structure expected for tables of type DATASET that are used for training models and applying models.

Datasets used for training or applying models can contain any number of columns (within SAP HANA and AFL limits). The supported SQL datatypes for the dataset columns are all the datatypes supported by AFL:

SQL Data Type
`"INTEGER"`
`"DOUBLE"`
`"CLOB"`
`"NCLOB"`
`"VARCHAR(n)"`
`"NVARCHAR(n)"`
`"DATE"`
`"TIME"`
`"SECONDDATE"`
`"TIMESTAMP"`
`"BIGINT"`
`"DECIMAL"`
`"DECIMAL(precision,scale)"` where 0 < precision <= 38 and 0 <= scale <= precision

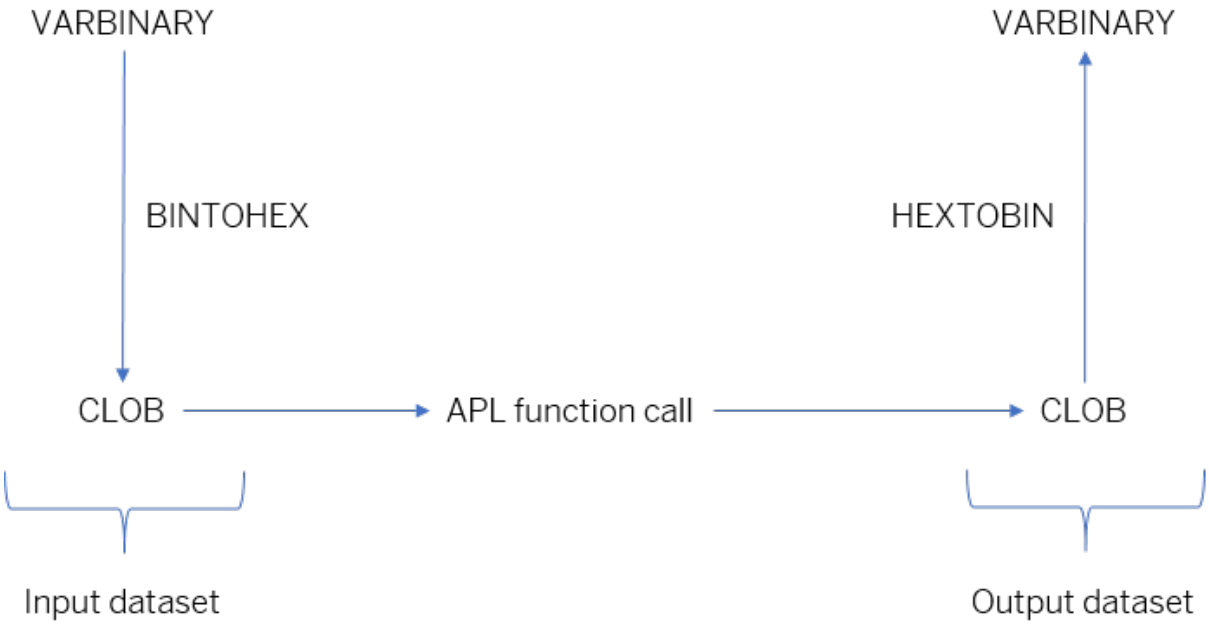
Support of DECIMAL Data Type

The `"DECIMAL"` data type is fully supported on SAP HANA 2.0 SPS03 and beyond, but not on previous versions of SAP HANA 1.0 and 2.0. However, you can use this type in your input dataset if you first convert it to `"DOUBLE"` before calling an APL function. You do this conversion with the `TO_DOUBLE` SQL function. To use it in your output dataset, you must convert it back from `"DOUBLE"` to `"DECIMAL"` by using the `TO_DECIMAL` SQL function. For more information about the functions, see *SAP HANA SQL and System Views Reference*.



Support of BINARY and VARBINARY Data Types

You can use the `"BINARY"` and `"VARBINARY"` data types in your input dataset if you first convert them to `"BLOB"` and `"CLOB"` respectively before any call to an APL function. You do this conversion by using the `BINTOHEX('<binary value>')` SQL function. To use these types in your output dataset, you must convert them back from `"BLOB"` or `"CLOB"` with the help of the `HEXTOBIN('<clob value>')` SQL function. For more information about the functions, see *SAP HANA SQL and System Views Reference*.



9.6 DEBRIEF_METRIC

The structure expected for tables of type `DEBRIEF_METRIC`.

When debrief tables are generated, debriefing metrics are produced. These tables are only designed to be accessed by the statistical reports.

Column	SQL Data Type
`MODEL_ID`	INTEGER
`OWNER_ID`	INTEGER
`DATASET_ID`	INTEGER
`OWNER_TYPE`	(N)VARCHAR, (N)CLOB
`NAME`	(N)VARCHAR, (N)CLOB
`VALUE`	DOUBLE

> **Note**
>
> This table type is available as `sap.pa.apl.base::BASE.T.DEBRIEF_METRIC`.

9.7 DEBRIEF_PROPERTY

The structure expected for tables of type `DEBRIEF_PROPERTY`.

When debrief tables are generated, debriefing properties are produced. These tables are only designed to be accessed by the statistical reports.

Column	SQL Data Type
`OID`	(N)VARCHAR, (N)CLOB
`OWNER_ID`	INTEGER
`OWNER_TYPE`	(N)VARCHAR, (N)CLOB
`NAME`	(N)VARCHAR, (N)CLOB
`VALUE`	(N)VARCHAR, (N)CLOB
`LONG_VALUE`	NCLOB
`D_VALUE`	DOUBLE
`I_VALUE`	INTEGER

> **Note**
>
> This table type is available as `sap.pa.apl.base::BASE.T.DEBRIEF_PROPERTY`.

9.8 FUNC_HEADER

The structure expected for tables of type `FUNC_HEADER` that defines the operating parameters of a function call.

A function header is provided to a function through a database table. A function header can be used to set the logging level or to provide an operation id. A function header is made of a collection of string pairs {KEY, VALUE}. It's the first input parameter of a function. The table is optional, an empty table can be used. In this case, default values are applied. The `FUNC_HEADER` table defines the format of the `MODEL` and the `LOG_LEVEL`.

Column	SQL Data Type	Description
KEY	(N)VARCHAR, (N)CLOB	The parameter key
VALUE	(N)VARCHAR, (N)CLOB	The parameter value

The supported parameters are the following:

Parameter name	Description
`Oid`	• Optional • Supported values: Any string • Default value: None. An operation ID. This optional ID can be provided by the APL caller to tag all the inserted rows in the the output tables.

`ModelFormat`	• Optional
	• Values:
	  • `bin` or `application/vnd.sap.aa`
	  • `zbin` or `application/vnd.sap.aa+zlib`
	  • `txt` or `text/plain` (not compatible with APL calls using the procedure technique)
	• Default value: `bin`.
	The requested output format for the model. This impacts the table structure of the model output table.

`LogLevel`	• Optional • Min value: `0` (Disabled) • Max value: `10` • Default: `8` This impacts the amount of progress messages and logging information produced by SAP HANA APL and the Automated Analytics engine. These messages can then be retrieved from the log operation table.
`Language`	• Optional • Values: `'en'`, `'fr'`, `'de'`, `'es'`, `'ja'`, `'pt'`, `'ru'`, `'zh'` • Default: `'en'` This impacts the language used in the outputs of the Automated Analytics engine, especially the messages in the operation log.
`DateFormat`	• Optional • Syntax: XXX[:Z], where XXX is a group of 3 letters (among 'Y', 'M' and 'D'), indicating in which order Year, Month and Day are represented. Z is a symbol giving the separator use between each of these. • Default: `'YMD:-'` This impacts the formatting of the dates.

Parameter name	Description
`Precision`	• Optional • Values: `0`, `1`, `2`... `17`, where `0` has the special meaning of "best width for a readable value". • Default: no value means best readable notation with maximum width (17) (`'auto:17'`). • Incorrect value: `'auto'` the best readable notation is used instead and the default width is 6. The number of digits after the decimal separator. This impacts the formatted values in the indicators provided by various functions. See Precision Formats [page 368] for the complete list of available formats.
`DecimalSymbol`	• Optional • Values: `0` for dot, `1` for comma • Default: `0` This impacts the formatted floating point values in the indicators provided by various functions
`Cancelable`	• Optional • Values: `'true'` for enabling cancellation capability, otherwise `'false'` • Default: `0` Allows the function to be canceled while it's running. Only functions that include a train or apply or test operation support cancellation.
`CancelCheckInterval`	• Optional • Values: Integer value greater than `0` • Default: `10` Interval (in seconds) between each cancellation check while an operation is running.
`ProgressLog`	• Optional • Values: `'true'` for enabling progress logging capability, `'noClean'` for enabling progress logging capability without clean progress message at the end, `'async'` is same `'noClean'` but the progress message generated with asynchronous function to avoid blocking the working process, `'false'` otherwise • Default: `'false'` This impacts the logging into the `"_SYS_AFL"."FUNCTION_PROGRESS_IN_APL_AREA"` progress table. Use the PROGRESS_CLEANUP [page 335] function to clean progress message.

Parameter name	Description
`CheckOperationConfig`	• Optional • Values: `'true'`, `'false'` • Default: `'false'` Checks that all defined configuration options are actually used. If not, the operation fails.

`MaxTasks`	• Optional
	• Type: Integer
	• Values: `0` to `n`
	  • `0`: The maximum number of tasks authorized by the current workload of the current SAP HANA instance
	  • `n` > `0`: A maximum of `n` tasks will be used in the apply process, or to process the set of segments in parallel
	• Default: `1`: Only one task is used and there is no parallelization of the apply process or of the segmented modeling process
	Defines the maximum number of tasks that will be used in the apply process or the segmented modeling process.

Example

When you declare the `FUNC_HEADER` table for a binary model that uses log level 6, use the following format:

```
create table FUNC_HEADER like FUNCTION_HEADER_T;
  insert into FUNC_HEADER values ('OID', 'foobar');
  insert into FUNC_HEADER values ('LOG_LEVEL', '6');
  insert into FUNC_HEADER values ('MODEL_FORMAT', 'bin');
```

Precision Formats

You can set the formatting of numbers by using either numbers or formatting styles:

- Supported numbers are integers from 1 to 17 and define the mantissa width.
- The 0 value implies the best width for a readable value.
- Supported formatting styles are `auto`, `fix`, `exp`, and `compat`.

You can use styles alone or with an additional width. The associated width influences the formatted number length.

Style	Description
`auto`	The best readable notation, either fixed or scientific notation

Style	Description
`fix`	The fixed notation (sign mantissa)
`exp`	The scientific notation (sign mantissa 'e' sign exp)
`compat`	The compatibility style. This style is implicitly used when the value is a number. The `'5'` value and `'compat:5'` are the same formatting.

The following table explains the different combinations of numbers and styles:

	If you use a simple width without style or if you use the `compat` style	If you use `auto`, `fix`, or `exp`
If the width is not specified	Then it is equivalent to `'auto:17'`.	Then it is equivalent to `'<style>'` with default width (6).
If the width is incorrect	Then it is equivalent to `'auto'` with default width (6).	
If the width is 0		Then it is `'<style>:<width>'`.
If the width is between 1 and 17	Then it is equivalent to `'fix:<width>'`.	

Example

Style	Example
`auto`	`'auto'`: best readable notation with default width (6) `'auto:3'`: best readable notation with width 3
`fix`	`'fix'`: fixed scientific notation with default width (6) `'fix:5'`: fixed scientific notation with width 5
`exp`	`'exp'`: scientific notation with default width (6) `'exp:7'`: scientific notation with width 7
`compat`	`'compat'`: same as `'auto:17'` (same effect as unspecified `Precision` value) `'compat:0'`: same as `'0'` or `'auto'`. `'compat:9'`: same as `'9'` or `'fix:9'`.

Related Information

MODEL [page 382]
LOG [page 381]

9.9 INDICATORS

The structure expected for tables of type `INDICATORS` and the names of all possible indicators.

When training, testing or querying a model, it's possible to retrieve variable indicators or statistics. For each variable, a collection of indicators may be retrieved. These indicators are described using the following attributes: {variable name, indicator name, indicator value, indicator detail (when applicable)}. Indicators are returned from a function through an output database table. The output table contains estimator indicators for regression models, to help plotting the regression curve.

Expected Table Structure (INDICATORS_T)

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	Operation ID
`TARGET`	(N)VARCHAR, (N)CLOB	Name of the target, when the indicator is based on it, for example: predictive power of predictive confidence
`VARIABLE`	(N)VARCHAR, (N)CLOB	Variable name
`KEY`	(N)VARCHAR, (N)CLOB	Indicator name
`VALUE`	(N)VARCHAR, (N)CLOB	Indicator value
`DETAIL`	(N)VARCHAR, (N)CLOB	Indicator detail

> **Note**
>
> An extra string column is appended, with the OID.

The following table type is used with the DEBRIEF_APPLY_RESULT [page 277], GET_MODEL_INFO [page 206] functions in the case of multiple training datasets.

You can also use this table type with the CREATE_MODEL_AND_TRAIN [page 137] to debrief the model from the testing partition of the dataset.

Expected Table Structure for Multiple Datasets (INDICATORS_DATASET_T)

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	Operation ID
`TARGET`	(N)VARCHAR, (N)CLOB	Name of the target, when the indicator is based on it, for example: predictive power of predictive confidence
`VARIABLE`	(N)VARCHAR, (N)CLOB	Variable name

Column	SQL Data Type	Description
KEY	(N)VARCHAR, (N)CLOB	Indicator name
VALUE	(N)VARCHAR, (N)CLOB	Indicator value
DETAIL	(N)VARCHAR, (N)CLOB	Indicator detail
DATASET	(N)VARCHAR, (N)CLOB	Dataset name

Indicators

Indicators are sorted by the type of model that they depend on. Indicators may depend on one or more types of models.

All Models Except Gradient Boosting Models

Indicator Name	Description
IndicatorDataset	The name of the dataset used to generated indicators, such as Estimation, Validation, and Test
Max	Maximum value for the variable (continuous only)
Mean	Mean of the variable (continuous only)
Min	Minimum value for the variable (continuous only)
StandardDeviation	Mean of the distance between the variable values and the mean (continuous only)

All Models Except Recommendation and Regression (Gradient Boosting) Models

Indicator Name	Description
CategoryFrequency	Frequency in percentage of the values of each variable in the training dataset
PredictionConfidence	Also known as **KR**. This the robustness indicator of the models generated using Automated Analytics. It indicates the capacity of the model to achieve the same performance when it is applied to a new dataset exhibiting the same characteristics as the training dataset.
PredictivePower	Also known as **KI**. This is the quality indicator of the models generated using Automated Analytics. It corresponds to the proportion of information contained in the target variable that the explanatory variables are able to explain.

TEST_MODEL Function Only

Indicator Name	Description
CategoryProbaDeviation	Probability that the category is significantly different from its distribution
CategoryTargetProbaDeviation	Probability that the target is significantly different from its distribution for the category
GroupProbaDeviation	Probability that the target is significantly different from its distribution for the group

Indicator Name	Description
GroupTargetProbaDeviation	Probability that the group is significantly different from its distribution in StatFor-Code
ProbaDeviation	Probability that there is a significant deviation in the distribution of categories (or bands in the case of continuous variables) between this dataset and the dataset jointed by the parameter StatForCode. This probability is computed through a CHI-square test.

Binary Classification Models (Legacy)

Indicator Name	Description
KS	Kolmogorof-Smirnoff of the variable with respect to this target variable
AUC	Area Under the ROC Curve of the variable with respect to this target variable
GINI	GINI index of the variable with respect to this target variable
TargetKey	The category of the Target variable that will be used as the positive target in the model
Threshold	Threshold calculated by the engine on Train or defined by the user (nominal variables)
True_Positive	Number of correctly predicted positive observations
Percent_True_Positive	True_Positive as a percentage
True_Negative	Number of correctly predicted negative observations
Percent_True_Negative	True_Negative as a percentage
False_Negative	Number of actual negative observations that have been predicted positive
Percent_False_Negative	False_Negative as a percentage
False_Positive	Number of actual positive observations that have been predicted negative
Percent_False_Positive	False_Positive as a percentage
Actual_Positive	Total number of actual positive observations
Percent_Actual_Positive	Actual_Positive as a percentage
Actual_Negative	Total number of actual negative observations
Percent_Actual_Negative	Actual_Negative as a percentage
Predicted_Positive	Total number of predictive positive observations
Percent_Predicted_Positive	Predicted_Positive as a percentage
Predicted_Negative	Total number of predictive positive observations
Percent_Predicted _Negative	Predicted_Negative as a percentage
Accuracy	Percentage of observations accurately classified by the model when applied on the training dataset
Sensitivity	Percentage of actual positive observations that have been correctly predicted

Indicator Name	Description
`Specificity`	Percentage of negative observations that have been correctly predicted
`Precision`	Percentage of detected positive observations that are actually positive observations
`F1_Score`	Harmonic mean of Precision and Sensitivity
`Profit`	The profit derived from the classification model based on the specified cost parameters
`Random_Profit`	The profit based on a random guess (without a classification model) using the specified cost parameters
`Gain`	The gap between `Profit` and `Random_Profit` (that is, `Profit` minus `Random_Profit`)
`Dataset_size`	The size of the dataset

Regression Models (Gradient Boosting), Regression/Classification Models, and Statbuilder

Indicator Name	Description
`MAPE`	Mean Absolute Percentage Error
`SMAPE`	Symmetric Mean Absolute Percentage Error
`MaxAPE`	Maximum Absolute Percentage Error, largest absolute percentage error over all records
`APEUnder5Pct`	Proportion of the absolute percentage errors that stand below 5%
`APEUnder10Pct`	Proportion of the absolute percentage errors that stand below 10%
`ErrorMean`	The mean of the error, that is, the difference between predicted values and actual values
`ErrorStdDev`	The standard deviation
`EstimatorSegmentMean`	Mean value of the estimator segment for the category. Detail contains the category label (Continuous or Nominal targets)
`EstimatorTargetMean`	Mean value of the target estimator. Detail contains the category label (Continuous or Nominal targets)
`EstimatorTargetStandardDeviation`	Standard deviation of the target estimator. Detail contains the category label
`L1`	The residual mean (the mean of the absolute value of the difference between the predicted value and the actual value), also known as City Block distance or Manhattan distance.
`L2`	The square root of the mean of square residuals
`LInf`	The maximum residual absolute value
`R2`	Used in in regression case, that is with a continuous target. The R2 is computed as the square correlation between the target and the model output.
`MeanAbsoluteError`	The residual mean. Has the same meaning as `L1`.
`RootMeanSquareError`	The square root of the mean of square residuals. Has the same meaning as `L2`.

Indicator Name	Description
`MaxAbsoluteError`	The maximum residual absolute value. Has the same meaning as `LInf`.

Regression and Binary Classification Models (Legacy)

Indicator Name	Description
`CategoryNormalProfit`	Normal profit value for each category of each variable/target combination
`CorrelatedVariable`	The correlations observed in the model between a variable and another (column details) for a target
`GroupFrequency`	Frequency in percentage of the wanted target value in the entire dataset
`GroupNormalProfit`	Normal profit value for each grouep values of each variable/target combination.
`GroupSignificance`	Significance of the grouped values (groups are constructed by target) of each variable
`GroupTargetMean`	Mean of the target for cases belonging to this Group.
`MaximumSmartVariableContribution`	The maximum smart variable contributions including only the maximum of similar variables
`NbVariablesKept`	Number of input variables kept by the auto-selection process
`VariableExclusionReason`	Reason why a variable has been excluded (`"Fully Compressed"`: fully compressed variable with respect to the target variable, "`Small KI on Estimation`","`Small KI on Validation`","`Large KI Difference`": sensible difference between the estimation and validation datasets,`"Small KR"` (on the estimation dataset), `"Leak Variable"`)

Binary Classification Models (Gradient Boosting)

Indicator Name	Description
KS	Kolmogorof-Smirnoff of the variable with respect to this target variable
AUC	Area Under the ROC Curve of the variable with respect to this target variable
`GINI`	GINI index of the variable with respect to this target variable
`TargetKey`	The category of the target variable that will be used as the positive target in the model
`LogLoss`	Logarithmic loss measuring the binary classification performance based on the prediction probabilities. This loss value is unbound and lower value indicates better classification performance.
`PredictiveConfidence`	Also known as **KR**. This the robustness indicator of the models generated using Automated Analytics. It indicates the capacity of the model to achieve the same performance when it is applied to a new dataset exhibiting the same characteristics as the training dataset.

Indicator Name	Description
PredictivePower	Also known as **KI**. This is the quality indicator of the models generated using Automated Analytics. It corresponds to the proportion of information contained in the target variable that the explanatory variables are able to explain.
ClassificationRate	The ratio of the variance of the model output with respect to the variance of the target. Used in the binary classification case, that is, with a nominal target classification rate.
EstimatorTargetKeyWeight	Weight of positive results found in a segment for a target. Detail contains the category label.
EstimatorTargetOtherKeyWeight	Weight of negative results found in a segment for a target. Detail contains the category label.
EstimatorTargetWeight	Total weight of positive and negative results in a segment for a target. In case of nominal targets, equals `EstimatorTargetKeyWeight` + `EstimatorTargetOtherKeyWeight` in this segment. Detail contains the category label.
WeightTotal	The total weight of positive and negative results in the Estimation dataset
InteractionMainEffect	The main effect of the variable
InteractionEffect	The cumulated interaction effect of all other variables
InteractionValue	The strongest variables that the variable interacts with
Threshold	Threshold calculated by the engine on Train or defined by the user (nominal variables)
True_Positive	Number of correctly predicted positive observations
Percent_True_Positive	`True_Positive` as a percentage
True_Negative	Number of correctly predicted negative observations
Percent_True_Negative	`True_Negative` as a percentage
False_Negative	Number of actual negative observations that have been predicted positive
Percent_False_Negative	`False_Negative` as a percentage
False_Positive	Number of actual positive observations that have been predicted negative
Percent_False_Positive	`False_Positive` as a percentage
Actual_Positive	Total number of actual positive observations
Percent_Actual_Positive	`Actual_Positive` as a percentage
Actual_Negative	Total number of actual negative observations
Percent_Actual_Negative	`Actual_Negative` as a percentage
Predicted_Positive	Total number of predictive positive observations
Percent_Predicted_Positive	`Predicted_Positive` as a percentage

Indicator Name	Description
`Predicted_Negative`	Total number of predictive positive observations
`Percent_Predicted _Negative`	`Predicted_Negative` as a percentage
`Accuracy`	Percentage of observations accurately classified by the model when applied on the training dataset
`Sensitivity`	Percentage of actual positive observations that have been correctly predicted
`Specificity`	Percentage of negative observations that have been correctly predicted
`Precision`	Percentage of detected positive observations that are actually positive observations
`F1_Score`	Harmonic mean of Precision and Sensitivity
`Profit`	The profit derived from the classification model based on the specified cost parameters
`Random_Profit`	The profit based on a random guess (without a classification model) using the specified cost parameters
`Gain`	The gap between `Profit` and `Random_Profit` (that is, `Profit` minus `Random_Profit`)
`Dataset_size`	The size of the dataset
`BalancedErrorRate`	Unweighted average of the classification error rates measured for each target class. This error value is between 0 and 1. Lower is better.
`BalancedClassificationRate`	Unweighted average of the classification rates measured for each target class. This value is between 0 and 1. Higher is better. The sum of `BalancedErrorRate` and `BalancedClassificationRate` is equal to 1.
`CohenKappa`	The kappa statistic, which expresses the level of agreement between the predicted and the true labels. The value is between -1 and 1. The maximum value means complete agreement, while zero or lower means chance agreement.

Regression Models (Gradient Boosting)

Indicator Name	Description
`InteractionMainEffect`	The main effect of the variable
`InteractionEffect`	The cumulated interaction effect of all other variables
`InteractionValue`	The strongest variables that the variable interacts with

Multinomial Classification Models

Indicator Name	Description
`MultiClassLogLoss`	Logarithmic loss measuring the classification performance based on the prediction probabilities. This loss value is unbound and lower value indicates better classification performance.
`BalancedErrorRate`	Unweighted average of the classification error rates measured for each target class. This error value is between 0 and 1. Lower is better.
`BalancedClassificationRate`	Unweighted average of the classification rates measured for each target class. This value is between 0 and 1. Higher is better. The sum of `BalancedErrorRate` and `BalancedClassificationRate` is equal to 1.
`MacroPrecision`	Unweighted average of the precision measured for each target class. This value is between 0 and 1. Higher is better.
`MacroRecall`	Unweighted average of the recall measured for each target class. This value is between 0 and 1. Higher is better.
`MacroF1Score`	Unweighted average of the F1 score measured for each target class. This value is between 0 and 1. Higher is better.
`MicroPrecision`	Weighted average of the precision measured for each target class. The weight is determined by the target class frequency. This value is between 0 and 1. Higher is better.
`MicroRecall`	Weighted average of the recall measured for each target class. The weight is determined by the target class frequency. This value is between 0 and 1. Higher is better.
`MicroF1Score`	Weighted average of the F1 score measured for each target class. This value is between 0 and 1. Higher is better.
`Precision`	Precision measured for each target class which is provided in the `DETAIL` column of the `INDICATORS` table. This value is between 0 and 1. Higher is better.
`Recall`	Recall measured for each target class which is provided in the `DETAIL` column of the `INDICATORS` table. This value is between 0 and 1. Higher is better.
`F1Score`	F1 score measured for each target class which is provided in the `DETAIL` column of the `INDICATORS` table. This value is between 0 and 1. Higher is better.
`VariableContribution`	The relative importance of an individual input variable to explain the target
`CohenKappa`	The kappa statistic, which expresses the level of agreement between the predicted and the true labels. The value is between -1 and 1. The maximum value means complete agreement, while zero or lower means chance agreement.
`InteractionMainEffect`	The main effect of the variable
`InteractionEffect`	The cumulated interaction effect of all other variables
`InteractionValue`	The strongest variables that the variable interacts with

Clustering Models

Indicator Name	Description
`ClusterCategoryFrequency`	The frequency in percentage of the values of each variable for a tuple cluster/Category. The value format is clusterId+Char(0x1F)+CategorieValue.

Indicator Name	Description
`ClusterFrequency`	The percentage of population gathered in the cluster
`ClusterIntraInertia`	The average of the squared distances between the centroid and every individual. Available per cluster and dataset.
`ClusterKL`	The Kullback-Leibler divergence between cluster distribution for this dimension (that is, this input variable) and population distribution
`ClusterRSS`	The residual sum of squares, that is, the sum of the squared distances between the centroid and every individual. Available per cluster and dataset.
`ClusterSimplifiedSilhouette`	This indicator is derived from the average ratio of the distances of all individuals to their two closest centroids. Available per cluster and dataset.
`ClusterSQL`	The cluster SQL expression
`ClusterTargetMean`	The mean value of the target for data assigned to the cluster
`ClusterTargetStandardDeviation`	The standard deviation of the target in this cluster, when the segmentation has been surpervised by a continuous target
`FinalClusterCount`	The number of clusters found by the model
`InitialClusterCount`	The number of clusters that have been asked for by the user
`IntraInertia`	The sum of the `ClusterIntraInertia`. Available per dataset.
`SimplifiedSilhouette`	The weighted average of the `ClusterSimplifiedSilhouette`. Available per dataset.
`Overlap`	When there is an overlap between two clusters or more
RSS	The sum of the `ClusterRSS`. Available per dataset.
`SumKIKR`	Available per dataset, in supervised mode only
`UnassignedRecords`	An unassigned record is record that cannot be included in any of the clusters

Time Series Models

Indicator Name	Description
`Cycles`	Periodic elements that can be found at least twice in the dataset.
`Fluctuations`	What is left when the trend and the cycles have been extracted.
`L1`	The residual mean (the mean of the absolute value of the difference between the predicted value and the actual value), also known as City Block distance or Manhattan distance.
`L2`	The square root of the mean of square residuals
MAPE	MAPE value for each forecast period. The forecast period is identified by its logical name (column details).
`MeanAbsoluteError`	The residual mean. Has the same meaning as `L1`.
MPE	Mean Percentage Error
MASE	Mean Absolute Scaled Error
P2	The square correlation between the target and the model output

Indicator Name	Description
`RootMeanSquareError`	The square root of the mean of square residuals. Has the same meaning as `L2`.
`R2`	The R2, or determination coefficient, is the ratio of the variance of the model output with respect to the variance of the target.
`SMAPE`	Symmetric Mean Absolute Percentage Error
`Trend`	The trend is the general orientation of the signal (Polynomial, Linear, and so on).
`U2`	1-r, where r is the ratio of the errors of the model output with respect to the variance of the target.

Other Cases

Indicator Name	Model	Description
`ClassificationRate`	Binary classification and Statbuilder models	The ratio of the variance of the model output with respect to the variance of the target. Used in the binary classification case, that is, with a nominal target classification rate.
`EstimatorTargetKeyWeight`	Binary classification and Statbuilder models, nominal targets	The weight of positive results found in a segment for a target. Detail contains the category label.
`EstimatorTargetOtherKeyWeight`	Binary classification and Statbuilder models, nominal targets	The weight of negative results found in a segment for a target. Detail contains the category label.
`EstimatorTargetWeight`	Binary classification, Regression (gradient boosting) ,and Statbuilder models, continuous or nominal targets	The total weight of positive and negative results in a segment for a target. In case of nominal targets, equals `EstimatorTargetKeyWeight` + `EstimatorTargetOtherKeyWeight` in this segment. Detail contains the category label.
`WeightTotal`	Binary classification, clustering and Statbuilder models, nominal targets	The total weight of positive and negative results in the Estimation dataset

> **Note**
>
> The Estimator indicators can be used to plot the regression curve. Each segment has a category label provided in the `DETAIL` column. The following indicators share the same category label, that is, the same detail value:
>
> - `EstimatorSegmentMean` provides the "wizard" value.
> - `EstimatorTargetMean` provides the estimator mean for the target.
> - `EstimatorTargetStandardDeviation` provides the standard deviation for each target mean value.
>
> For more information about confusion matrix numbers, see *Automated Analytics User Guides and Scenarios* on the SAP Help portal.

9.10 INFLUENCERS

The structure expected for tables of type `INFLUENCERS`.

The influencers are the variables that influence a target, ranked by their contribution to the target

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	Operation ID
`TARGET`	(N)VARCHAR, (N)CLOB	Tagret name
`RANK`	INTEGER	Influencer ranking, 1 being the top influencer.
`VARIABLE`	(N)VARCHAR, (N)CLOB	Variable name: the influencer name
`STORAGE`	(N)VARCHAR, (N)CLOB	Storage. Possible values: • number • integer • string • date • datetime • angle
`VALUETYPE`	(N)VARCHAR, (N)CLOB	Value type. Possible values: • nominal • ordinal • continuous • textual
`CONTRIBUTION`	DOUBLE	Contribution
`BASED_ON`	(N)VARCHAR, (N)CLOB	Base variable name, for generated variables
`COMPOSITION_FUNCTION`	(N)VARCHAR, (N)CLOB	Composition function name on the base variable, for generated variables.

9.11 INFO

The structure expected for tables of type `INFO`.

Version and configuration information is returned as a collection of properties, i.e. a collection of string pairs { name, value }. Expected table structure:

Column	SQL Data Type	Description
`NAME`	VARCHAR, size 128	The property name

Column	SQL Data Type	Description
VALUE	VARCHAR, size 1024	The property value

The possible properties are the following:

Name (VARCHAR, size 128)	Description (VARCHAR, size 1024)
APL.Version.Major	APL major version number
APL.Version.Minor	APL minor version number
APL.Version.ServicePack	APL service pack number
APL.Version.Patch	APL patch version number
APL.Info	APL version string
AFLSDK.Version.Major	AFL SDK major version number
AFLSDK.Version.Minor	AFL SDK minor version number
AFLSDK.Version.Patch	AFL SDK patch version number
AFLSDK.Info	AFL SDK version string
AutomatedAnalytics.Version.Major	Automated Analytics Engine major version number
AutomatedAnalytics.Version.Minor	Automated Analytics Engine minor version number
AutomatedAnalytics.Version.ServicePack	Automated Analytics Engine patch version number
AutomatedAnalytics.Version.Patch	Automated Analytics Engine patch version number
AutomatedAnalytics.Info	Automated Analytics Engine version string

9.12 LOG

The structure expected for tables of type LOG.

This is the operation log. When training or applying a model, the Automated Analytics engine produces status/warning/error messages. These messages are returned from a function through an output database table.

Column	SQL Data Type	Description
OID	(N)VARCHAR, (N)CLOB)	Operation ID (OID)
TIMESTAMP	TIMESTAMP	Message timestamp
LEVEL	INTEGER	Message level
ORIGIN	(N)VARCHAR, (N)CLOB)	Message origin
MESSAGE	NCLOB	The actual message

9.13 MODEL

The structures expected for tables of type `MODEL`.

Most functions create, read, write, or update a predictive model. This model is specific to Automated Analytics. SAP HANA APL supports different model formats, for different integration scenarios.

Supported Parameter Names

Model Format	Description
`bin` or `application/vnd.sap.aa`	Opaque binary format. This is the Automated Analytics native serialization format.
`zbin` or `application/vnd.sap.aa+zlib`	Opaque binary format, compressed. This is a compressed version of the bin format.
`txt` or `text/plain`	Open text format. The model is defined using a collection of parameters, that is, a collection of {key, value} pairs.
`native` or `application/vnd.sap.aa.native`	Native format. The model is saved in readable format by the kernel.

The expected model format can be specified in the function header (see FUNC_HEADER [page 365]). The requested model format determines the table structure used for exchanging the model with APL. The format of an input model is automatically guessed.

> **Note**
>
> You cannot use the procedure technique with models in text format.

Output MODEL Table Structure (bin and zbin)

For `bin` and `zbin` output models (the default format), the expected table structure is:

bin and zbin Output MODEL Table Structure

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	The operation ID (OID). If no OID was initially specified, then a new one is generated.
`FORMAT`	(N)VARCHAR, (N)CLOB	The model format version string.
`LOB`	CLOB	The Automated Analytics model, in a serialized and binary format (and potentially compressed).

> ⚠ Caution
>
> For `zbin` output models on SAP HANA 2.0 SPS03 only, the expected table structure is:
>
> | Column | SQL Data Type | Description |
> |---|---|---|
> | `OID` | (N)VARCHAR, (N)CLOB | The operation ID (OID). If no OID was initially specified, then a new one is generated. |
> | `FORMAT` | (N)VARCHAR, (N)CLOB | The model format version string. |
> | `LOB` | LargeBinary | The Automated Analytics model, in a serialized and binary format (and potentially compressed). |
>
> You can use this `MODEL` table structure with the direct technique only.

Output MODEL Table Structure (text)

For text output models (new model or updated model), the expected table structure is:

txt MODEL Table Structure

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	The operation ID (OID). If no OID was initially specified, then a new one is generated.
`ID`	INTEGER	Model parameter ID.
`KEY`	(N)VARCHAR, (N)CLOB	Model parameter path.
`VALUES`	(N)CLOB	Model parameter value.

> ⓘ Note
>
> The `VALUES` column also supports the SQL data type (N)VARCHAR(1024) for backward compatibility with existing code used to define regression or classification models.

> ⚠ Caution
>
> The model format `txt` is available for debug or advanced purposes. This format consumes a lot of memory. In order to minimize the memory used, it is advisable to use one of the other formats available or to purge the content of the model by calling the UPDATE_MODEL function with the configuration parameter `APL/ForgetTraining` set to `'true'`.

Input MODEL Table Structure

The tables are as described above. For input models, the OID column is ignored.

When models are stored in the native format (that is, in SAP HANA tables), they must be sorted before they are used by APL. This applies, for example, to social models and recommendation models. For example:

```
-- A model stored using the native format must be sorted when consumed by APL
CREATE MODEL_SORTED AS SELECT * from MODEL_NATIVE_FORMAT ORDER BY "ID" ASC;
DO BEGIN
    header       = select * from FUNC_HEADER;
    model        = select * from MODEL_SORTED;
    apply_config = select * from APPLY_CONFIG;
    "SAP_PA_APL"."sap.pa.apl.base::APPLY_MODEL"(:header, :model, :apply_config,
'APL_SAMPLES','ADULT01', 'USER_APL','ADULT01_APPLY' , out_apply_log, out_sum);
    -- store result into table
...
END;
```

9.14 OPERATION_CONFIG

The structure expected for tables of type `OPERATION_CONFIG`.

Most of the functions can be parameterized using parameters, known as the operation configuration parameters. The `OPERATION_CONFIG` table defines the list of mandatory and optional parameters that a particular function can handle.

> **Note**
>
> The `OPERATION_CONFIG` parameters are specific to the APL function used.

The structure of these parameters can be one of the following:

- A collection of string pairs {`KEY`, `VALUE`} for simple-key parameters
- A collection of string triplets {`KEY`, `VALUE`, `CONTEXT`} for compound-key parameters.

The `KEY` parameter can be one of the following:

- The full path of the parameter, as documented and natively handled by the Automated Analytics engine
- A so-called alias name, always prefixed by `APL/`, which allows configuring an operation in a much simpler way. There's no global list of the well-known aliases. Each APL function handles and documents its own set of aliases.

The `CONTEXT` parameter represents the context in which the alias must be set. For instance, if a model has multiple targets, you set the target to be used in `CONTEXT` along with the alias in `KEY` and its value in `VALUE`:

```
insert into APPLY_CONFIG values ('APL/ApplyExtraMode', 'Advanced Apply
Settings', null);
insert into APPLY_CONFIG values ('APL/ApplyProbability', 'true', 'TARGET_A');
insert into APPLY_CONFIG values ('APL/ApplyProba', 'true', 'TARGET_A');
insert into APPLY_CONFIG values ('APL/ApplyProbability', 'true', 'TARGET_B');
insert into APPLY_CONFIG values ('APL/ApplyProba', 'true', 'TARGET_B');
```

Operation parameters are provided to a function through a database table.

Expected Table Structure for Simple Parameters

Column	SQL Data Structure	Description
KEY	(N)VARCHAR, (N)CLOB	The parameter alias name, or the parameter full path as described in the Automated Analytics documentation
VALUE	(N)VARCHAR, (N)CLOB	The parameter value

Expected Table Structure for Compound Parameters

Column	SQL Data Type	Description
KEY	(N)VARCHAR, (N)CLOB	The parameter alias name, or the parameter full path as described in the Automated Analytics documentation
VALUE	(N)VARCHAR, (N)CLOB	The parameter value
CONTEXT	(N)VARCHAR, (N)CLOB	The parameter context

9.15 OTHER_GROUPS

The structure expected for tables of type `OTHER_GROUPS` (groups for non-continuous targets).

Column	SQL Data Type	Description
OID	(N)VARCHAR, (N)CLOB	The operation ID (aka OID). If no OID was initially specified, then a new one is generated.
VARIABLE	(N)VARCHAR, (N)CLOB	Variable name, i.e. the influencer name
TARGET	(N)VARCHAR, (N)CLOB	Target name
GROUP	(N)VARCHAR, (N)CLOB	Group name
CATEGORY	(N)VARCHAR, (N)CLOB	Category name

9.16 OUTPUT_TABLE_TYPE

The structure expected for the table returned by the `GET_TABLE_TYPE_FOR_APPLY` function.

These types are also returned by the `GET_MODEL_DATASET_TYPES` and `GET_MODEL_INFO` functions.

Column	SQL Data Type	Description
OID	(N)VARCHAR, (N)CLOB	Operation ID
POSITION	INTEGER	Position column

Column	SQL Data Type	Description
NAME	(N)VARCHAR, (N)CLOB	Column name
KIND	(N)VARCHAR, (N)CLOB	Column type (SQL type)
PRECISION	INTEGER	Column precision
SCALE	INTEGER	Column scale
MAXIMUM_LENGTH	INTEGER	Column max length (-1 for Max possible length)

Related Information

GET_MODEL_INFO [page 206]
GET_MODEL_DATASET_TYPES [page 201]
GET_TABLE_TYPE_FOR_APPLY [page 213]

9.17 PROFITCURVES

The structure expected for tables of type PROFITCURVES.

Column	SQL data type	Description
OID	(N)VARCHAR, (N)CLOB	Operation ID
TYPE	(N)VARCHAR, (N)CLOB	Curve type
VARIABLE	(N)VARCHAR, (N)CLOB	Variable name
TARGET	(N)VARCHAR, (N)CLOB	Target name
Label	(N)VARCHAR, (N)CLOB	Label
Frequency	(N)VARCHAR, (N)CLOB	Frequency value
Random	(N)VARCHAR, (N)CLOB	Random value
Wizard	(N)VARCHAR, (N)CLOB	Wizard value
Estimation	(N)VARCHAR, (N)CLOB	Training value
Validation	(N)VARCHAR, (N)CLOB	Validation value
Test	(N)VARCHAR, (N)CLOB	Testing value
ApplyIn	(N)VARCHAR, (N)CLOB	ApplyIn value (defined when a model has been tested (TEST_MODEL, APPLY_MODEL_AND_TEST)

9.18 PUBLISHING_SUMMARY

The structure expected for tables of type `PUBLISHING_SUMMARY`.

When publishing a model, debriefing information related to the publish operation is produced. This is known as the publishing summary. This information is a set of indicators, provided as string pairs {KEY, VALUE}.

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	The operation ID
KEY	(N)VARCHAR, (N)CLOB	Indicator name
VALUE	(N)VARCHAR, (N)CLOB	Indicator value

> **Note**
>
> An extra string column is appended, with the OID.

This summary contains an overview of the publishing operation provided by the Automated Analytics engine. The following table provides the full list of publishing summary indicators:

Indicator Name	Origin	Description
`ModelPublishVersion`	Automated Analytics engine	Model publish version
`ModelPublishDate`	Automated Analytics engine	Model publish date

9.19 RESULT

The structure expected for tables of type `RESULT`.

When operation is performed on a model, the result is returned as list of keys, whose values are:

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	Operation ID
KEY	(N)VARCHAR, (N)CLOB	Key name
VALUE	(N)VARCHAR, (N)CLOB	Key value

> **Note**
>
> An extra string column is appended, with the OID.

9.20 RULE_ITEMS

The structure expected for tables of type `RULE_ITEMS`.

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	Operation ID
`ITEM_ID`	INTEGER	Item ID. This ID is used a foreign key in the Association Rules table.
`VARIABLE`	(N)VARCHAR, (N)CLOB	Variable name
`CATEGORY`	(N)VARCHAR, (N)CLOB	Variable category

9.21 SUMMARY

The structure expected for tables of type `SUMMARY`.

When training or applying a model, debriefing information related to the operation is produced. This is known as the summary. This information is a set of indicators provided as string pairs {`KEY`, `VALUE`}.

Column	SQL Data Type	Description
`OID`	(N)VARCHAR, (N)CLOB	The operation ID
`KEY`	(N)VARCHAR, (N)CLOB	Indicator name
`VALUE`	(N)VARCHAR, (N)CLOB	Indicator value

> **Note**
>
> An extra string column is appended with the `OID`.

Summary Indicators

The summary contents depend on the type of model. The summary contains performance indicators provided by APL, and an overview of the training operation provided by the Automated Analytics engine. The following tables provide the full list of training summary indicators:

Indicator name	Origin	Description
`ModelAuthor`	Automated Analytics engine	Identity of the user who trained the model

Indicator name	Origin	Description
`ModelAvailable`	Automated Analytics engine	`true` when a model has been successfully trained and generated, `false` otherwise
`ModelBuildDate`	Automated Analytics engine	Date and time when the model was built
`ModelCuttingStrategy`	Automated Analytics engine	Partition strategy applied on the training dataset for building the model
`ModelDatasetName`	Automated Analytics engine	Name of the dataset
`ModelEngineName`	Automated Analytics engine	Name of the feature used to build the model: • Kxen.RobustRegression • Kxen.SmartSegmenter • Kxen.TimeSeries
`ModelLearningTime`	Automated Analytics engine	Total learning time (in seconds)
`ModelRecordCount`	Automated Analytics engine	Number of records in the dataset (Training Validation record count)
`ModelRecordCountEstimation`	Automated Analytics engine	Number of records of the training dataset used for training the model
`ModelRecordCountTest`	Automated Analytics engine	Number of records of the test dataset used for training the model
`ModelRecordCountValidation`	Automated Analytics engine	Number of records of the validation dataset used for training the model
`ModelSelectedVariableCount`	Automated Analytics engine	Number of explanatory variables actually used by the resulting model
`ModelState`	Automated Analytics engine	The model state, either `SpecifiedModel` or `TrainedModel`
`ModelVariableCount`	Automated Analytics engine	Number of explanatory variables used by the resulting model

Association Rules

Indicator name	Origin	Description
`AssocRuleCount`	Automated Analytics engine	Number of association rules generated
`AssocRuleFrequentItemSetCount`	Automated Analytics engine	Number of frequent item sets, that is the number of itemsets whose support is superior to the minimum support set by the user.
`AssocRuleItemCount`	Automated Analytics engine	Number of items found in the transaction dataset

Indicator name	Origin	Description
AssocRuleMaxLength	Automated Analytics engine	Maximum size of a rule, i.e. number of antecedent items plus number of consequent items (always 1).
AssocRuleMinConfidence	Automated Analytics engine	Minimum threshold for the confidence of a rule.
AssocRuleMinSupport	Automated Analytics engine	Minimum support required for a rule
AssocRuleSessionCount	Automated Analytics engine	Number of sessions treated

Binary Classification Models (Gradient Boosting and Legacy)

Indicator name	Origin	Description
AplParallelTasks	APL	If segmented modeling or parallel apply has been activated, the number of HANA tasks actually used to do the segmented modeling or apply. This number depends on the actual workload of the SAP HANA instance and the current workload class.
AplTaskElapsedTime	APL	If segmented modeling or parallel apply has been activated, for each task, the total time in milliseconds (ms)
AplTaskStatus	APL	If segmented modeling has been activated, for each segment, 'Succeed' or 'Failed' if a model cannot be built. In such a case, a detailed error can be found in the LOG table.
AplTotalElapsedTime	APL	If segmented modeling or parallel apply has been activated, the user time in milliseconds (ms) of the whole scoring process (training as well as scoring) or the whole apply process. This is the time spent inside the APL plugin and does not include the time spent in the SQL infrastructure and wrappers.

Binary Classification Models (Gradient Boosting)

Indicator name	Origin	Description
BestIteration	Automated Analytics engine	The boosting iteration index having the best performance on validation dataset

Indicator name	Origin	Description
`EarlyStoppingPatience`	Automated Analytics engine	Maximum number of iterations used to improve the performance of the model. If the performance did not improve after `EarlyStoppingPatience` iterations, there is no need to make further attempts to improve the model and so the training stops before reaching `MaxIterations`.
`EvalMetric`	Automated Analytics engine	List of metrics used to evaluate the model performance on validation dataset along the boosting iterations
`LearningRate`	Automated Analytics engine	This weight parameter controls the model regularization to avoid the well-known overfitting risk. A small value improves the model generalization upon unseen dataset at the expense of the computational cost
`MaxDepth`	Automated Analytics engine	Maximum depth of the decision tree which is added as a base learner to the model at each boosting iteration
`MaxIterations`	Automated Analytics engine	Maximum number of boosting iterations to fit the model with the training dataset
`NumberOfJobs`	Automated Analytics Engine	The number of native threads. These are not monitored by the SAP HANA resource manager. **Note** The `NumberOfJobs` parameter is currently always set to 1.

Clustering Models

Indicator name	Origin	Description
`AplParallelTasks`	APL	If segmented modeling or parallel apply has been activated, the number of HANA tasks actually used to do the segmented modeling or apply. This number depends on the actual workload of the SAP HANA instance and the current workload class.
`AplTaskElapsedTime`	APL	If segmented modeling or parallel apply has been activated, for each task, the total time in milliseconds (ms)
`AplTaskStatus`	APL	If segmented modeling has been activated, for each segment, 'Succeed' or 'Failed' if a model cannot be built. In such a case, a detailed error can be found in the LOG table.

Indicator name	Origin	Description
`AplTotalElapsedTime`	APL	If segmented modeling or parallel apply has been activated, the user time in milliseconds (ms) of the whole scoring process (training as well as scoring) or the whole apply process. This is the time spent inside the APL plugin and does not include the time spent in the SQL infrastructure and wrappers.
`ModelMaxClusters`	Automated Analytics engine	Maximum number of clusters that have been asked for by the user
`ModelMinClusters`	Automated Analytics engine	Minimum number of clusters that have been asked for by the user
`ModelSqlEnabled`	Automated Analytics engine	Indicates if the SQL expressions for the clusters definitions have been calculated (`true`) or not (`false`)

Multinomial Classification Models (Gradient Boosting)

Indicator name	Origin	Description
`AplParallelTasks`	APL	If segmented modeling or parallel apply has been activated, the number of HANA tasks actually used to do the segmented modeling or apply. This number depends on the actual workload of the SAP HANA instance and the current workload class.
`AplTaskElapsedTime`	APL	If segmented modeling or parallel apply has been activated, for each task, the total time in milliseconds (ms)
`AplTaskStatus`	APL	If segmented modeling has been activated, for each segment, 'Succeed' or 'Failed' if a model cannot be built. In such a case, a detailed error can be found in the LOG table.
`AplTotalElapsedTime`	APL	If segmented modeling or parallel apply has been activated, the user time in milliseconds (ms) of the whole scoring process (training as well as scoring) or the whole apply process. This is the time spent inside the APL plugin and does not include the time spent in the SQL infrastructure and wrappers.
`BestIteration`	Automated Analytics engine	The boosting iteration index having the best performance on validation dataset
`EarlyStoppingPatience`	Automated Analytics engine	Maximum number of iterations used to improve the performance of the model. If the performance did not improve after `EarlyStoppingPatience` iterations, there is no need to make further attempts to improve the model and so the training stops before reaching `MaxIterations`.

Indicator name	Origin	Description
`EvalMetric`	Automated Analytics engine	List of metrics used to evaluate the model performance on validation dataset along the boosting iterations
`LearningRate`	Automated Analytics engine	This weight parameter controls the model regularization to avoid the well-known overfitting risk. A small value improves the model generalization upon unseen dataset at the expense of the computational cost
`MaxDepth`	Automated Analytics engine	Maximum depth of the decision tree which is added as a base learner to the model at each boosting iteration
`MaxIterations`	Automated Analytics engine	Maximum number of boosting iterations to fit the model with the training dataset
`NumberOfClasses`	Automated Analytics engine	Number of target classes found from the training dataset and considered for the multiclass modeling
`NumberOfJobs`	Automated Analytics Engine	The number of native threads. These are not monitored by the SAP HANA resource manager. **Note** The `NumberOfJobs` parameter is currently always set to 1.

Recommendation Models

Indicator name	Origin	Description
`RecoFilteredLineCount`	Automated Analytics engine	Number of filtered lines in the Recommender
`RecoItemCount`	Automated Analytics engine	Number of Separate Items in the Recommender
`RecoRuleCount`	Automated Analytics engine	Number of Rules in the Recommender
`TransBestSellerCount`	Automated Analytics engine	Number of Best Sellers in the filtered dataset
`TransDistinctItemCount`	Automated Analytics engine	Number of Separate Users
`TransDictinctUserCount`	Automated Analytics engine	Number of Separate Users
`TransFilteredLineCount`	Automated Analytics engine	Number of filtered lines in the dataset
`TransPairUserItemCount`	Automated Analytics engine	Number of User/Item Pairs in the filtered dataset

Regression Models (Gradient Boosting and Legacy)

Indicator name	Origin	Description
`AplParallelTasks`	APL	If segmented modeling or parallel apply has been activated, the number of HANA tasks actually used to do the segmented modeling or apply. This number depends on the actual workload of the SAP HANA instance and the current workload class.
`AplTaskElapsedTime`	APL	If segmented modeling or parallel apply has been activated, for each task, the total time in milliseconds (ms)
`AplTaskStatus`	APL	If segmented modeling has been activated, for each segment, 'Succeed' or 'Failed' if a model cannot be built. In such a case, a detailed error can be found in the LOG table.
`AplTotalElapsedTime`	APL	If segmented modeling or parallel apply has been activated, the user time in milliseconds (ms) of the whole scoring process (training as well as scoring) or the whole apply process. This is the time spent inside the APL plugin and does not include the time spent in the SQL infrastructure and wrappers.

Regression Models (Gradient Boosting)

Indicator name	Origin	Description
`BestIteration`	Automated Analytics engine	The boosting iteration index having the best performance on validation dataset
`EarlyStoppingPatience`	Automated Analytics engine	Maximum number of iterations used to improve the performance of the model. If the performance did not improve after `EarlyStoppingPatience` iterations, there is no need to make further attempts to improve the model and so the training stops before reaching `MaxIterations`.
`EvalMetric`	Automated Analytics engine	List of metrics used to evaluate the model performance on validation dataset along the boosting iterations
`LearningRate`	Automated Analytics engine	This weight parameter controls the model regularization to avoid the well-known overfitting risk. A small value improves the model generalization upon unseen dataset at the expense of the computational cost
`MaxDepth`	Automated Analytics engine	Maximum depth of the decision tree which is added as a base learner to the model at each boosting iteration

Indicator name	Origin	Description
`MaxIterations`	Automated Analytics engine	Maximum number of boosting iterations to fit the model with the training dataset
`NumberOfJobs`	Automated Analytics Engine	The number of native threads. These are not monitored by the SAP HANA resource manager. **Note** The `NumberOfJobs` parameter is currently always set to 1.

Time-Series Models

Indicator name	Origin	Description
`AplParallelTasks`	APL	The number of HANA tasks actually used to do the segmented modeling. This number depends on the actual workload of the SAP HANA instance and the current workload class.
`AplTaskElapsedTime`	APL	If segmented modeling has been activated, for each segment, the total time in milliseconds (ms)
`AplTaskStatus`	APL	If segmented modeling has been activated, for each segment, 'Succeed' or 'Failed' if a model cannot be built. In such a case, a detailed error can be found in the LOG table.
`AplTotalElapsedTime`	APL	If segmented modeling has been activated, the user time in milliseconds (ms) of the whole scoring process (training as well as scoring). This is the time spent inside the APL plugin and does not include the time spent in the SQL infrastructure and wrappers.
`ModelTimeSeriesFirstDate`	Automated Analytics engine	Time-Series First Date
`ModelTimeSeriesHorizon`	Automated Analytics engine	Time-Series Horizon
`ModelTimeSeriesLastDate`	Automated Analytics engine	Time-Series Last Date
`ModelTimeSeriesMAPE`	Automated Analytics engine	Time-Series Horizon-Wide MAPE
`ModelTimeSeriesMaxHorizon`	Automated Analytics engine	Time-Series Maximum Horizon

9.22 USER

The structure expected for tables that list the users for the analysis.

Column	SQL Data Type	Description
`UserID`	The `$User` ID column in the dataset.	A list of users for who you want to determine suggestions/recommendations.

9.23 VARIABLE_DESC

The structure expected for tables of type `VARIABLE_DESC`.

Depending on the function used, `VARIABLE_DESC` is used as an input table or an output table. The variable descriptions table provides metadata about a dataset variable. Some functions allow specifying variable descriptions. Other functions generate variable descriptions. Variable descriptions are provided to a function through a database table.

Column	SQL Data Type	Description
1st column	INTEGER	Variable rank
2nd column	(N)VARCHAR, (N)CLOB	Variable name
3rd column	(N)VARCHAR, (N)CLOB	Storage. Possible values: • `number` (the variable contains only "computable" numbers (be careful a telephone number, or an account number should not be considered numbers) • `integer` • `string` (: the variable contains character strings) • `date` (the variable contains dates) • `datetime` (the variable contains date and time stamps) • `angle`

Column	SQL Data Type	Description
4th column	(N)VARCHAR, (N)CLOB	Value type: the value type of the variable. Possible values: • `nominal` (categorical variable which is the only possible value for a string) • `ordinal` (discrete numeric variable where the relative order is important) **Note** Since APL 2414, a clear error message is now returned when a variable is described as string/ordinal. • `continuous` (a numeric variable from which mean, variance, etc. can be computed) • `textual` (textual variable containing phrases, sentences or complete texts)
5th column	INTEGER	Indicates if the variable is a key. Possible values: • `0` if false • `1` if true
6th column	INTEGER	Order level used to sort dataset rows (0,1, 2,...,n). **Note** Order level has no effect on the APL functions.
7th column	(N)VARCHAR, (N)CLOB	Missing string value: the string used in the data description file to represent missing values (for example, "999" or "#Empty" - without the quotes)
8th column	(N)VARCHAR, (N)CLOB	Group name.
9th column	(N)VARCHAR, (N)CLOB	Variable description: an additional description label for the variable.
OID (10th column)	(N)VARCHAR, (N)CLOB	The operation ID, if set. Otherwise a new one is generated. This column is optional.

Note

- The order and the SQL data types for the variable description tables are important.

- When variable descriptions are returned as an output table from a function, an extra string column is appended to the output table, with the `OID`.

When you create a table of variable descriptions, use the following format:

```
create table VARIABLE_DESC like VARIABLE_DESC_T;
  insert into VARIABLE_DESC values (0, 'age', 'integer', 'continuous', 0, 0,
'???', NULL, 'age variable description');
  insert into VARIABLE_DESC values (1, 'workclass', 'string', 'nominal', 0, 0,
'???', NULL, 'workclass variable description');
  insert into VARIABLE_DESC values (2, 'education', 'string', 'nominal', 0, 0,
'???', NULL, 'education variable description');
  insert into VARIABLE_DESC values (3, 'marital-status', 'string', 'nominal', 0,
0, '???', NULL, 'marital-status variable description');
  insert into VARIABLE_DESC values (4, 'occupation', 'string', 'nominal', 0, 0,
'???', NULL, 'occupation variable description');
  ...
```

9.24 VARIABLE_ROLES

The structure expected for tables of type `VARIABLE_ROLES`.

When training a model, the roles of the variables can be specified. These variable roles are provided as string pairs {variable name, variable role}. Variable roles are provided to a function through a database table.

In data modeling, variables may have four roles. They may be:

- Target variables (also known as output variables): a target variable is the variable that you seek to explain, or for which you want to predict the values in an application dataset. It corresponds to your domain-specific business issue. In some businesses, target variables may also be known as variables to be explained or dependant variables.
- Explanatory variables (also known as input variables): an input variable describes your data and serves to explain a target variable. Explanatory variables may also be known as causal variables or independent variables.
- Weight variables: a weight variable allows one to assign a relative weight to each of the observations it describes, and actively orient the training process.
- Skipped variables: these variables are ignored during the training process.

The roles of the variables are specified when the model is trained. This is optional though, since SAP HANA APL and the Automated Analytics engine automatically apply a default set of business rules:

- If no variable role is specified, then all the dataset variables are considered as 'input' variables, except the last one which is considered as the 'target'.
- If a variable has been described as a key (in the variable descriptions), then it's automatically considered as 'skip' before applying any new roles.
- If the specified roles contain a target, then all the other existing targets are reset to 'input' before applying the new roles.
- All the specified variables roles take precedence over these rules.

These variable roles are provided as string pairs {`NAME`, `ROLE`}.

Expected Table Structure

Column	SQL Data Type	Description
NAME	(N)VARCHAR, (N)CLOB	Variable name
ROLE	(N)VARCHAR, (N)CLOB	Variable role. Supported roles: • `input` • `skip` • `target` • `weight`

Composite Variables

The table that's used for setting the variable roles can also be used to define composite variables. A composite variable allows defining and using a new virtual variable - known as the composite - which combines other existing variables of the dataset - known as the components. A composite variable is the cross product of two or more variables.

> **Note**
>
> Composite variables can be used with clustering and classification/regression models only. When defining composite variables, the variable role table structure is extended to allow setting both variable roles and compositions:

Expected Table Structure for Composite Variables

Column	SQL Data Type	Description
NAME	(N)VARCHAR, (N)CLOB	Variable name
ROLE	(N)VARCHAR, (N)CLOB	Variable role. Supported roles: • `input` • `skip` • `target` • `weight`
COMPOSITION_TYPE	(N)VARCHAR, (N)CLOB	Type of composition for the composite variable. Supported composition types: • `cross` • `latitude` • `longitude` • `null` Use `null` when setting the role of a non-composite variable.

Column	SQL Data Type	Description
COMPONENT_NAME	(N)VARCHAR, (N)CLOB	Component name that's part of the composite variable. A component name is expected to be a variable name that belongs to the dataset. Use `null` when setting the role of a non-composite variable.

This table structure allows setting both variable roles and composite variables. If you need to use this table structure because there's a composite variable to be set, and if you also need to set a role on a non-composite variable (e.g. setting the target on a variable of the dataset), then you must set the `COMPOSITION_TYPE` and `COMPONENT_NAME` column values to `null`. When defining a `cross` composite variable that's composed of `n` variables, you need to set `n` rows in this table, i.e. one row per component. Moreover, `cross` composite variables support ranking: the order of the components in the composite can be explicitly set (by default it's undefined, because it depends on how the table is scanned by SAP HANA). In order to enforce the order of the components of a `cross` composite variable, you can post-fix `cross` with a integer value that sets the rank of the component in the composite, e.g. `cross0`, `cross1`, `cross2`, ... `crossN`

9.25 Table Types for Social Modeling

These table types are used by the social model functions.

> **Note**
>
> Social models and recommendation models can be stored using the native format. Therefore, before a social or recommendation model is used, it must first be sorted. For more information, see *MODEL* (section *Input MODEL Table Structure*).

Related Information

MODEL [page 382]

9.25.1 Social Models Serialization

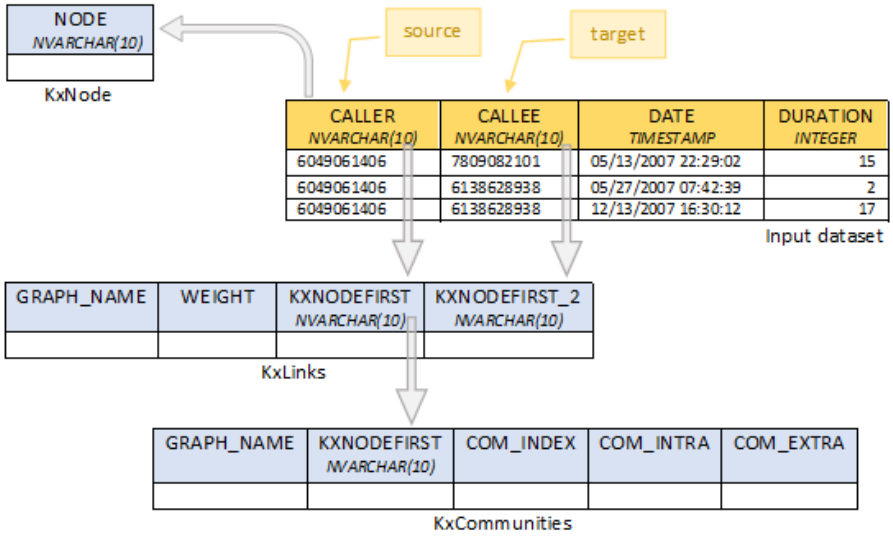
When creating and training social models, their serialization scheme varies greatly according to the input dataset. The following tables are part of this scheme and must be created beforehand:

- Nodes tables (KxNodes1 and KxNodes2)
- Links table (KxLinks)
- Communities table (KxCommunity)
- Attributes tables (KxAttributes1 and KxAttributes2)

The following diagrams provide some general rules about the structure of these tables and their relationships with the input dataset table.

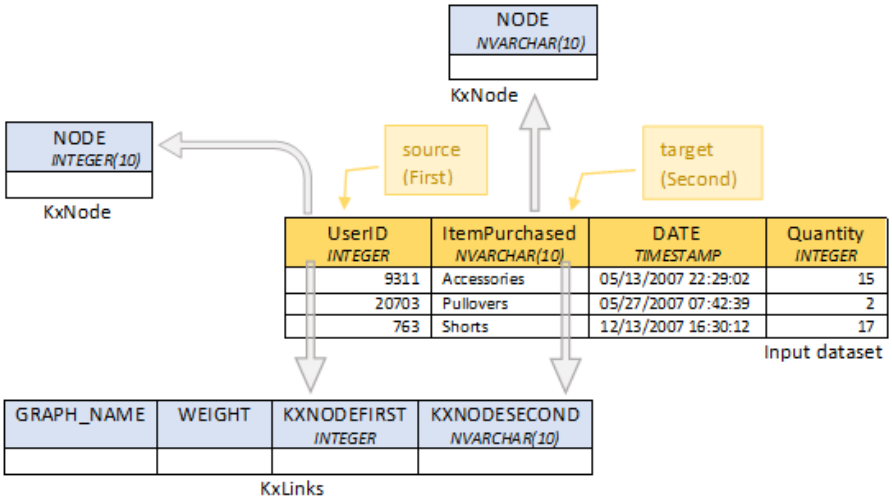
For Links, Nearest Neighbors and Contact types

For these types of graphs, there is only one population, and by convention, this population is named First. From the input dataset, one column is defined as source and another as target, and based this information, a graph can be built and represented as a collection of links between nodes. The nodes and the links are stored respectively in the KxNodes1 and KxLinks tables. The layout of these tables and the types of their most relevant columns are depicted in the diagram below:

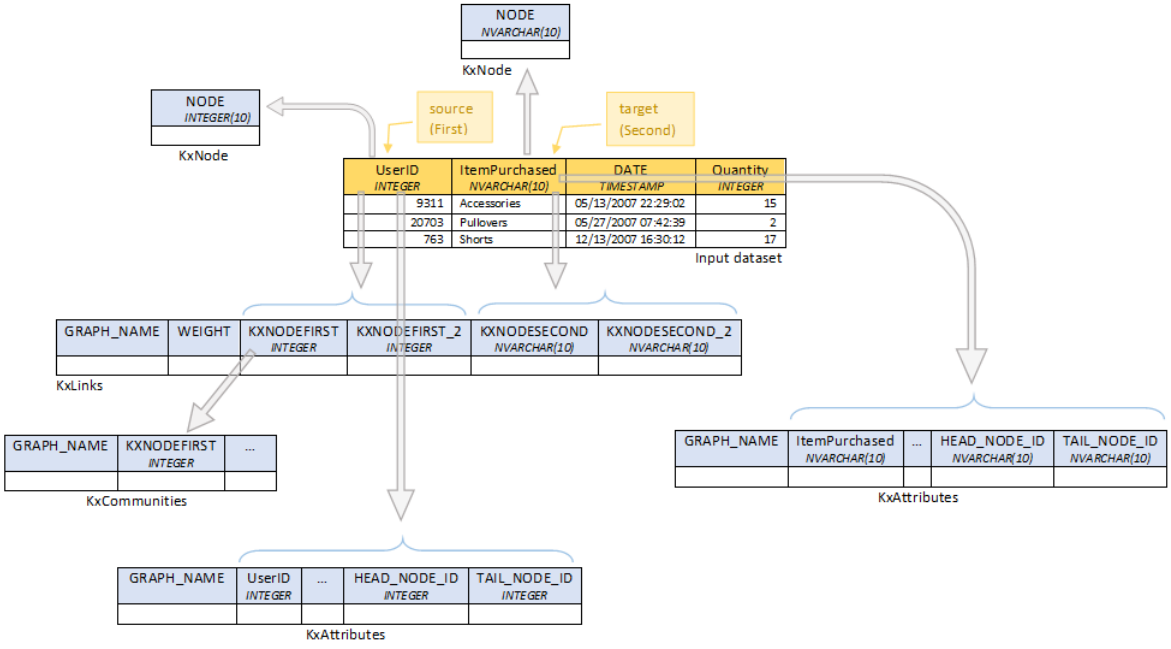


For Transaction type

For this kind of graph, two populations are defined, and by convention, they are named respectively First and Second. In this case, the nodes are stored in the KxNodes1 (for the first population) and KxNodes2 (for the second population) tables, whereas the KxLinks table contains the links. The layout of these tables and the types of their most relevant columns are depicted in the diagram below:



However, for this kind of graph, usually an additional specific post-processing step (projection) is performed, in order to generate new derived graphs. Let's assume, we want to derive two graphs from both populations, and in addition, perform a 'Communities detection' on the derived graph from the 'First' population. In this case, the tables structure is depicted in the following diagram:



Nodes tables

For Links, Nearest Neighbors and Contact Types

Based on the example depicted in the first figure above.

For Links, Nearest Neighbors and Contact Types

Column	SQL Data Type	Description
node	NVARCHAR(10)	Must be the same SQL type as the identifier column (eg. CALLER)

For Transaction type

For transaction type graphs, two tables are needed (based on the example described in the second figure above:

For Transaction type

Column	SQL Data Type	Description
node	INTEGER	Must be the same SQL type as the identifier column of the first population(eg. UserID)

And:

For Transaction type

Column	SQL Data Type	Description
node	NVARCHAR(10)	Must be the same SQL type as the identifier column of the second population(eg. ItemPurchased)

Links table

For Links, Nearest Neighbors and Contact Types

For Links, Nearest neighbors, and Contact Types

Column	SQL Data Type	Description
GRAPH_NAME	NVARCHAR(255)	
WEIGHT	DOUBLE	
KXNODEFIRST	NVARCHAR(10)	Must be the same SQL type as the identifier column (eg. CALLER)
KXNODEFIRST_2	NVARCHAR(10)	Must be the same SQL type as the identifier column (eg. CALLER)

For Transaction Type Without Projection

For Transaction Type Without Projection

Column	SQL Data Type	Description
GRAPH_NAME	NVARCHAR(255)	
WEIGHT	DOUBLE	
KXNODEFIRST	NVARCHAR(10)	Must be the same SQL type as the identifier column of the first population (eg. UserID)
KXNODESECOND	NVARCHAR(10)	Must be the same SQL type as the identifier column of the second population(eg. ItemPurchased)

For Transaction Type With Projection From the First Population

For Transaction Type With Projection From the First Population

Column	SQL Data Type	Description
GRAPH_NAME	NVARCHAR(255)	
WEIGHT	DOUBLE	
KXNODEFIRST	INTEGER	Must be the same SQL type as the identifier column of the first population (eg. UserID)
KXNODEFIRST_2	INTEGER	Must be the same SQL type as the identifier column of the first population (eg. UserID)
KXNODESECOND	NVARCHAR(10)	Must be the same SQL type as the identifier column of the second population (eg. ItemPurchased)

Communities

If Communities detection is not enabled, any table structure is accepted by the social related methods. Based on the example depicted in the first figure above.

Communities

Column	SQL Data Type	Description
GRAPH_NAME	NVARCHAR(255)	

Column	SQL Data Type	Description
KXNODEFIRST	NVARCHAR(10)	Must be the same SQL type as the identifier column of the first population (eg. UserID)
COM_INDEX	INTEGER	
COM_INTRA	INTEGER	
COM_EXTRA	INTEGER	

> **Note**
>
> Communities detection cannot be performed on Transaction graphs.

9.25.2 ATTRIBUTES

The structures expected for tables of type ATTRIBUTES.

If no derived graphs are defined, any table structure is accepted by the social related methods based on the example depicted in Social Models Serialization [page 400] ("Graph model output tables Fig-3").

Attributes

Column	SQL Data Type	Description
`GRAPH_NAME`	NVARCHAR(255)	
`UserID`	INTEGER	The same name and type as the first population
`generation_id`	INTEGER	The column name `generation_id` must be lowercase.
`HEAD_NODE_ID`	INTEGER	
`TAIL_NODE_ID`	INTEGER	

And:

Attributes

Column	SQL Data Type	Description
`GRAPH_NAME`	NVARCHAR(255)	
`ItemPurchased`	NVARCHAR(10)	The same name and type as the second population
`generation_id`	INTEGER	The column name `generation_id` must be lowercase.
`HEAD_NODE_ID`	INTEGER	

Column	SQL Data Type	Description
TAIL_NODE_ID	INTEGER	

9.25.3 GRAPH_FILTERS

The structure expected for tables of type GRAPH_FILTERS.

As several graphs can be built from the same dataset, a slice of data can be dedicated to a given graph. This can be achieved with the graph filters. A filter can be a range (in this case, a minimum and maximum values have to be defined) or a set of values to be accepted or discarded.

Column	SQL Data Type	Description
GRAPH_NAME	NVARCHAR(255)	Graph name
COLUMN	NVARCHAR(127)	Column to be filtered
OPERATOR	NVARCHAR(10)	May be 'accept', 'discard', 'minimum', or 'maximum'
VALUE	NCLOB	Value to be used in the filter

The operators 'minimum', and 'maximum' allow a range definitions. The operators 'accept', and 'discard' define a set of discrete values, and are similar respectively to the 'IN' and 'NOT IN' SQL operators.

9.25.4 GRAPH_INDICATORS

The structure expected for tables of type GRAPH_INDICATORS.

When building graphs, it's possible to retrieve information on the created graphs. These indicators are described using the following attributes: { graph name, indicator name, indicator value}.

Column	SQL Data Type	Description
OID	NVARCHAR(50)	Operation ID
GRAPH_NAME	NVARCHAR(100)	Name of the created graph
KEY	NVARCHAR(127)	Indicator name
VALUE	NCLOB	Indicator value

9.25.5 GRAPH_POST_PROCESSING

The structure expected for tables of type `GRAPH_POST_PROCESSING`.

The social model creation function provides several post processings tools:

Tool	Description
Detecting communities	Identifies a community in the dataset. A community can be seen as a subgraph in which the density of internal connections is larger than the connections with the rest of nodes in the network. Communities detection cannot be performed on transaction-typed graphs.
Mega-hub detection optimization	A node can be considered as a mega-hub when its degree is above a certain threshold. Removing these highly connected nodes can dramatically improve processing time, and put away irrelevant nodes from the analysis.
Pairing nodes	Node pairing allows identifying an item (customer, product, etc.) present in two different graphs (thus with two different nodes), by means of its common neighbors.
Projection	Define a bipartite projection for an entity (source or target column) from a transaction graph. **Note** A projection can only be defined on transaction-typed graphs

You can enable the mentioned tools by configuring a table with the following expected structure:

Expected structure

Column	SQL Data Type	Description
NAME	NVARCHAR(100)	User-defined name of the processing
TYPE	NVARCHAR(50)	May be 'communities', 'megahub', 'pairing', or 'bipartiteprojection'
KEY	NVARCHAR(1000)	Post processing parameter name
VALUE	NVARCHAR(100)	Post processing parameter value

The parameters for the different type of post processings are:

Detecting Communities

Key	Required	Supported Values	Description
GraphName	Yes	name of a defined graph	Name of the graph for which the post processing is enabled

Key	Required	Supported Values	Description
Epsilon	Yes	number with scientific notation (eg.: 8.85E-5)	Value below which the modularity gain is no longer interesting enough to justify another iteration of the algorithm.
MaxIterations	Yes	number	Maximum number of iterations after which the algorithm should stop if the Epsilon criteria has not been reached
SeedGraph	Yes	name of a defined graph	Graph from which you want to use the communities as seed for the current graph. Can only reference a graph where communities detection has been defined.
IgnoreSelfLoop	Yes	'true' or 'false'	Reduces the number of communities and gives a better insight in the graph structure.
DisableSuperCommunity	Yes	'true' or 'false'	Reduces the number of communities and gives a better insight in the graph structure.

Mega-hub Detection

Key	Required	Supported Values	Description
GraphName	Yes	name of a defined graph	Name of the graph for which the post processing is enabled
Population	Yes	'First' ('First' or 'Second' for transaction graphs)	Defines the population for which the detection is performed.
DeviationFactor	Yes	number	Configures the formula used to calculate an automatic threshold.
Threshold	Yes	number	Manual threshold

> **Note**
>
> DeviationFactor and Threshold are exclusive.

Pairing Nodes

Key	Required	Supported Values	Description
FirstGraph	Yes	Name of a defined graph	Name of the first graph of this algorithm
SecondGraph	Yes	Name of a defined graph	Name of the second graph of this algorithm
CountThreshold	Yes	number	Defines the minimum number of neighbors two nodes must have in common to be paired
TopN	Yes	number	Defines the number of pairings you want to keep among the highest ranking ones
PairingGraph	Yes	graph name	Name of the output graph
RatioThreshold	Yes	number (eg. 0.5)	Keeps only the pairings for which the Jaccard index in higher than or equal to this value
WeightedRatio	Yes	'true' or 'false'	Applies the weight variable declared when setting column roles to be used to compute the Pairing Type
IncludeNeighbors	Yes	'true' or 'false'	Generates an additional graph where the links are labeled with the number of common neighbors
PairingType	Yes	'Ratio', 'Jaccard', 'Independence_Ratio', 'Confidence', 'Clustering'	Allows you to select the type of pairing to be used to compute the Top N pairings

Projection

Key	Required	Supported Values	Description
SourceGraph	Yes	Name of a defined graph	The name of the transaction graph from which the current graph is derived
TargetGraph	Yes	Graph name	Name of the projected graph
Population	Yes	'First' or 'Second'	The entity used as node for the current graph
MinSupport	Yes	number	The number of items two entities have in common. Any link with support value below this number will not be created
MaxIterations	Yes	number	Maximum number of iterations after which the algorithm should stop

Key	Required	Supported Values	Description
TopN	Yes	number	Defines the number of pairings you want to keep among the highest ranking ones
OptimizeSpeed	Yes	'true' or 'false'	Enabling this option will speed up the process but will consume more memory
Weight	Yes	'Support', 'Jaccard Index', 'Independence Probability', 'Cosine'	Indicates which value to assign as a weight for the links
MinConfidence	Yes	number	Links with confidence value higher than the defined value will be kept
MinPredictivePower	Yes	number	Links with predictive power value higher than the defined value will be kept
Directed	Yes	'true' or 'false'	Enabling this option will differentiate created links according to their direction and provide directed rules
TargetEventRequired	Yes	'true' or 'false'	Enabling this option will define the population of your graphs
MaxTopNodes	Yes	number	Defines the maximum number of nodes to keep

9.25.6 LINKS

The structure expected for tables of type `LINKS`.

Recommendation requires some extra table to be persisted, typically the `KxLinks1` table.

Column	SQL Data Type	Description
`GRAPH_NAME`	NVARCHAR(255)	Graph name
`WEIGHT`	DOUBLE	Link weight
`KXNODEFIRST`	Same as the `$User` column in the dataset	Node Id related to User entity
`KXNODESECOND`	Same as the `$Item` column in the dataset	Node Id related to Item entity

Column	SQL Data Type	Description
KXNODESECOND_2	Same as the $Item column in the dataset	Node Id related to Community Item entity

9.25.7 NODES

The structure expected for tables of type NODES.

Recommendation Model Serialization (KxNode1)

This recommendation requires an extra table to be persisted, typically the KxNodes1 table.

Column	SQL Data Type	Description
node	Same as the $User column in the dataset.	Node ID related to User entity.

Recommendation Model Serialization (KxNode2)

This recommendation requires an extra table to be persisted, typically the KxNodes2 table.

Column	SQL Data Type	Description
node	Same as the $Item column in the dataset.	Node ID related to Item entity.

9.25.8 RECO_SCORE

The structure expected for tables of type RECO_SCORE.

Column	SQL Data Type	Description
$User	Same as the $User column in the dataset	Column identifying the User in the dataset (Set in the APL/User parameter)
$Item	Same as the $Item column in the dataset	Column identifying the Item in the dataset (Set in the APL/Item parameter)
sn_rec_rule_id	INTEGER	Recommendation rule ID

Column	SQL Data Type	Description
sn_rec_kxReco	(N)VARCHAR, (N)CLOB	Recommended item
sn_rec_source	(N)VARCHAR, (N)CLOB	Recommender name
sn_rec_score	DOUBLE	Recommendation score

9.25.9 TRANSACTION

The structure expected for tables of type `TRANSACTION`.

This table defines transaction facts from a User entity to a Product/Item entity, used to create the recommendation model.

Column	SQL Data Type	Description
$User	Same as the $User column in the data-set	Column identifying the User in the data-set (Set in the APL/User parameter)
$Item	Same as the $Item column in the data-set	Column identifying the Item in the data-set (Set in the APL/Item parameter)
$Date	DATETIME	Transaction time stamp (optinal)
$Quantity	INTEGER	Transaction weight/quantity (optinal)
$TransactionID	*	Transaction ID

9.25.10 APPLY MODE for Social Modeling

The apply mode defines the type of output you want to generate.

The supported modes when applying a social model are:

Social APPLY MODE

Mode	The Generated Data Will Contain:
Default_Mode	All the information available on the node and its neighbors
Circle_Mode	The number of neighbors, the average of the neighbors' attributes, and additional information on the neighbors, if available.
Centrality_Mode	An evaluation of its centrality by analyzing its local clustering and its number of neighbors

Mode	The Generated Data Will Contain:
Neighbors_Mode	A list with all its neighbors and additional information on them.
Describe_Mode	The list of all information available for this node.

9.25.11 Graph Types

Define the graph creation type according to the input dataset:

Graph Types

This option...	Generates graphs...	With the following information on the link...
`link`	Linking the identifiers from the source column with the identifiers from the target column.	
`contact`	Containing only the links found at least a specified number of times.	Number of occurrences of the link
`transaction`	Containing the relations between objects of a different nature, such as customers and products for example	
`nearestneighbors`	Containing a specific number of nearest neighbors for each node. The nearest neighbors are determined by their relative distance, which must be indicated in an additional column.	Distance

10 Statistical Reports

A model generated by SAP HANA APL contains valuable information that can help you understand how your data is distributed and how the model discovered and used patterns in the input data.

This information can be difficult to retrieve because there are different locations where it could be stored, such as in the INDICATORS table, in the summary, in the progress logs, or in the model itself. To allow this information to be retrieved more easily, SAP HANA APL provides a set of standard statistical reports, which can be called using SAP HANA SQLScript.

Domains

Each report has a valid domain. A domain can include the following types of information:

Information Type	Description
Model generic	Generic information, available for all model types
Variables	Information about variables, available for all model types
Continuous variables	Detailed information about continuous variables, available for all model types
Nominal variables	Detailed information about nominal variables, available for all model types
Binary target	Detailed information about the binary target, available for all model types
Multiclass target	Detailed information about the multiclass target, available for all model types
Continuous target	Detailed information about the continuous target, available for all model types
Specific to a type of model	Detailed information valid for one specific type of model, currently only regression and time series models

Using Reports

You typically use reports as follows:

1. You train a model by calling, for example, `"SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN"`.
2. You generate debrief tables by calling `"SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_DEBRIEF"`.

> **Note**
>
> - Although debrief table types are available for use, the content of the actual debrief tables is private to SAP and might evolve over time. It is strongly recommended to always use the standard reports to access their content.
> - Debrief tables are typically transient. As a best practice, you should generate them each time you generate the report so that their content is always up to date and reflects the current version in APL. You should delete them after the report has been generated.

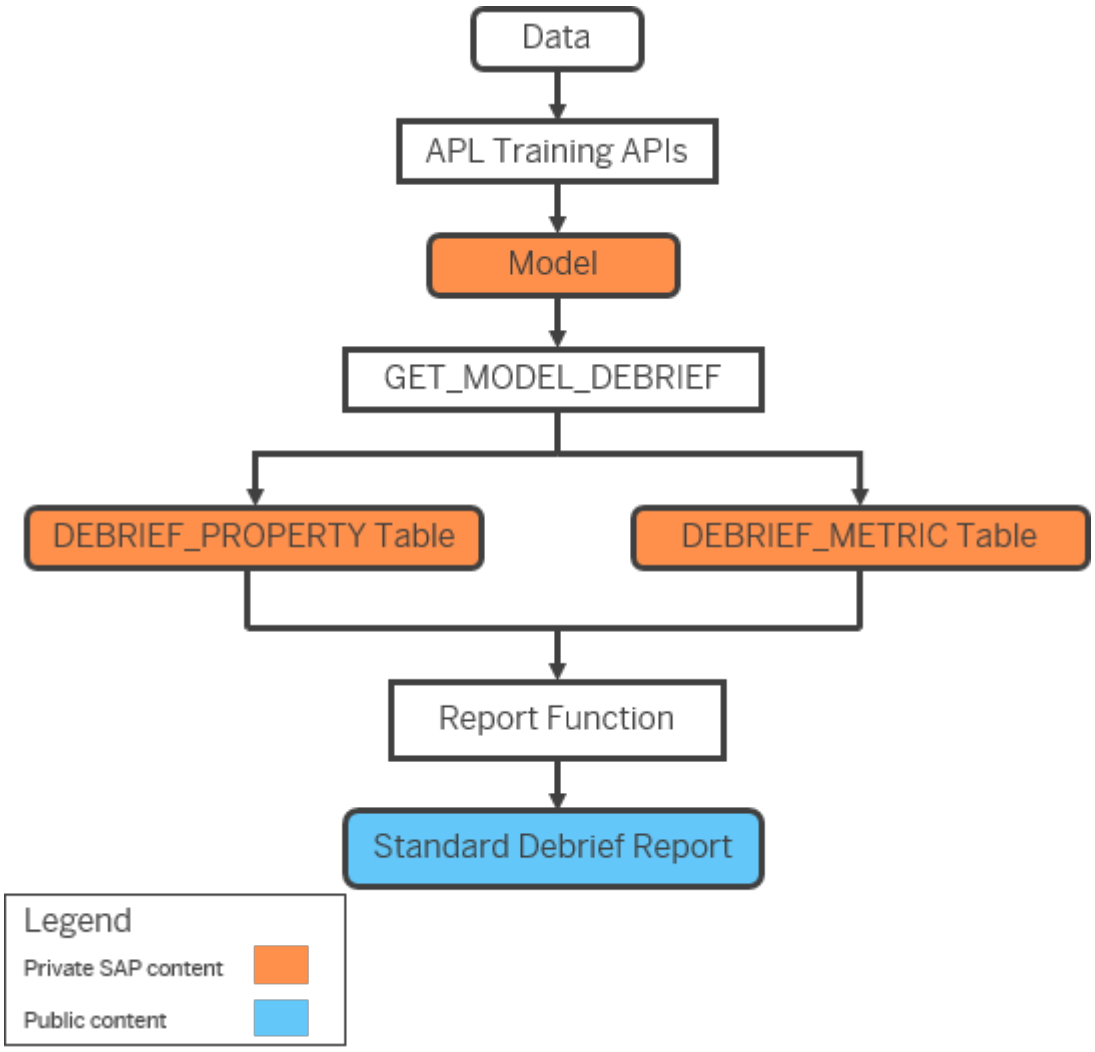
3. You generate a report by calling a specific report function as follows:

```
"SAP_PA_APL"."sap.pa.apl.debrief.report::<report_name>"
```

> **Sample Code**
>
> ```
> "SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_ContinuousVariables"
> ```

4. You store the information derived from the reports and delete the debrief tables.

The overall process is shown graphically below, with the private and publicly available information indicated where applicable:



API

Each report is wrapped in a dedicated SQLScript function so that it can be handled like a standard table:

Code Syntax

```
SELECT * FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::<report_name>"(<debrief_property_table>,<debrief_metric_table>)
```

Example

Sample Code

Procedure mode

```
connect USER_APL password Password1;

DO BEGIN
    declare header "SAP_PA_APL"."sap.pa.apl.base::BASE.T.FUNCTION_HEADER";
    declare config "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CONFIG_EXTE
NDED";
    declare debrief_config "SAP_PA_APL"."sap.pa.apl.base::BASE.T.OPERATION_CON
FIG_EXTENDED";
    declare var_desc "SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_DESC_OID";

    declare var_role "SAP_PA_APL"."sap.pa.apl.base::BASE.T.VARIABLE_ROLES_WITH
_COMPOSITES_OID";

    :header.insert(('Oid', '#42'));
    :header.insert(('LogLevel', '8'));
    :header.insert(('ModelFormat', 'bin'));

    :config.insert(('APL/ModelType', 'regression',null));
    :config.insert(('APL/NumberOfJobs', '4',null));
    :config.insert(('APL/MaxIterations', '200',null));
    :config.insert(('APL/MaxDepth', '4',null));
    :config.insert(('APL/EarlyStoppingPatience', '20',null));
    :config.insert(('APL/EvalMetric', 'RMSE',null));

    :var_role.insert(('age', 'target', null, null, null));
-- build the model
    "SAP_PA_APL"."sap.pa.apl.base::CREATE_MODEL_AND_TRAIN"(:header, :config, :
var_desc,:var_role,  'APL_SAMPLES','ADULT01',out_model,out_log,out_sum,out_ind
ic);
-- generate debrief tables
    "SAP_PA_APL"."sap.pa.apl.base::GET_MODEL_DEBRIEF"(HEADER_TABLE,
MODEL_TABLE, DEBRIEF_CONFIG_TABLE,DEBRIEF_METRIC_TABLE,
DEBRIEF_PROPERTY_TABLE, SUMMARY_TABLE);

    -- Dump Statistics Report
    SELECT * FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_ContinuousVariables"(DEBRI
EF_PROPERTY_TABLE,DEBRIEF_METRIC_TABLE);

-- debrief tables are deleted
END;
```

10.1 Generic

The reports in this section provide generic information about the model.

10.1.1 List of Partitions

The `Statistics_Partition` report lists the partitions that were built from the training dataset. It is available for all model types.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_Partition"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_Partition"(PROPERTY_BAG,METRI
C_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Statistics_Partition
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Build Date	TIMESTAMP	Date and time when the model was built
Partitioning Method	NVARCHAR(512)	Strategy for partitioning the training dataset into subsets. See Partition Strategies [page 64].

Field Name	Type	Description
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Weight	DOUBLE	Number of rows in the partition
% Weight	DOUBLE	Number of rows in the partition as a percentage

10.1.2 List of Variables

The `Statistics_Variables` report lists all the variables in the model together with their roles. It is available for all model types.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_Variables"
```

Syntax (Procedure Only)

```
SELECT * FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_Variables"(PROPERTY_BAG,METRI
C_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Statistics_Variables
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Variable	NVARCHAR(512)	Name of the variable
Value	NVARCHAR(512)	Kind of variable: nominal, ordinal, continuous
Storage	NVARCHAR(512)	Type of variable: integer, numeric, date, datetime, string
Missing Value Weight	DOUBLE	Number of missing values
% Missing Value Weight	DOUBLE	Number of missing values as a percentage
Role	NVARCHAR(512)	Role of the variable in the model: target, input, skip, or weight

10.1.3 Category Frequencies of Variables and Partitions

The `Statistics_CategoryFrequencies` report lists the category frequencies for each variable and partition. It is available for all model types.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_CategoryFrequencies"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_CategoryFrequencies"(PROPERTY
_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Direction	Type	Required	Description
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Statistics_CategoryFrequencies
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Variable	NVARCHAR(512)	Name of the variable
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Category	NCLOB	Variable category
Weight	DOUBLE	Weight of the category in the dataset
% Weight	DOUBLE	Frequency of the category in the dataset as a percentage
Category Order	INTEGER	Order of the category in the dataset

10.1.4 Group Frequencies of Variables and Partitions

The `Statistics_GroupFrequencies` report lists the group frequencies for each variable and partition. It is available for all model types.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_GroupFrequencies"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_GroupFrequencies"(PROPERTY_BA
G,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Statistics_GroupFrequencies
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Variable	NVARCHAR(512)	Name of the variable
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Group	NCLOB	Group of variable categories
Weight	DOUBLE	Weight of the group in the dataset
% Weight	DOUBLE	Frequency of the group in the dataset as a percentage
Group Order	INTEGER	Order of the group in the dataset

10.1.5 List of Continuous Variables

The `Statistics_ContinuousVariables` report lists the continuous variables with their descriptive statistics. It is available for all model types. However, it only applies to continuous variables.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_ContinuousVariables"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Statistics_ContinuousVariables"(PROPERTY
_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Statistics_ContinuousVariables
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Variable	NVARCHAR(512)	Name of the variable
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Minimum	DOUBLE	Minimum value of the variable
Maximum	DOUBLE	Maximum value of the variable
Mean	DOUBLE	Mean of the variable
Standard Deviation	DOUBLE	Standard deviation of the variable

10.2 Target Types

The reports in this section provide detailed information about the target types.

10.2.1 Binary Target

The reports in this section provide detailed information about the binary target.

10.2.1.1 Distribution of the Binary Target

The `BinaryTarget_Statistics` report gives the distribution of the binary target. It is available for all model types. However, it only applies when the target is binary.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_Statistics"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_Statistics"(PROPERTY_BAG,ME
TRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.BinaryTarget_Statistics
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Target Key	NVARCHAR(512)	Value of the target variable used as the positive target in the model
% Positive Weight	DOUBLE	Weight of positive targets as a percentage
% Negative Weight	DOUBLE	Weight of negative targets as a percentage
Weight	DOUBLE	Total weight of positive and negative targets in the dataset

10.2.1.2 ROC Curve of the BinaryTarget

The `BinaryTarget_CurveRoc` report provides the ROC curve of the binary target. It is available for all model types. However, it only applies when the target is binary.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_CurveRoc"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_CurveRoc"(PROPERTY_BAG,METR
IC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Direction	Type	Required	Description
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.BinaryTarget_CurveRoc
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Row Id	INTEGER	Row ID
Target	NVARCHAR(512)	Name of the target
False Positive Rate	DOUBLE	False positive rate
Random	DOUBLE	Value for a random guess
Wizard	DOUBLE	Value for a perfect model
True Positive Rate (Estimation)	DOUBLE	True positive rate of the estimation dataset
True Positive Rate (Validation)	DOUBLE	True positive rate of the validation dataset
True Positive Rate (Test)	DOUBLE	True positive rate of the test dataset

10.2.1.3 Lift Curve of the BinaryTarget

The `BinaryTarget_CurveLift` report provides the lift curve of the binary target. It is available for all model types. However, it only applies when the target is binary.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_CurveLift"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_CurveLift"(PROPERTY_BAG,MET
RIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.BinaryTarget_CurveLift
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Row Id	INTEGER	Row ID
Target	NVARCHAR(512)	Name of the target
% Population	DOUBLE	Percentage of the population
Random	DOUBLE	Value for a random guess
Wizard	DOUBLE	Value for a perfect model
Lift (Estimation)	DOUBLE	Lift value of the estimation dataset
Lift (Validation)	DOUBLE	Lift value of the validation dataset
Lift (Test)	DOUBLE	Lift value of the test dataset

10.2.1.4 Gain Curve of the BinaryTarget

The `BinaryTarget_CurveGain` report provides the gain curve of the binary target. It is available for all model types. However, it only applies when the target is binary.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_CurveGain"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_CurveGain"(PROPERTY_BAG,MET
RIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.BinaryTarget_CurveGain
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Row Id	INTEGER	Row ID
Target	NVARCHAR(512)	Name of the target
% Population	DOUBLE	Percentage of the population
Random	DOUBLE	Value for a random guess

Field Name	Type	Description
Wizard	DOUBLE	Value for a perfect model
Gain (Estimation)	DOUBLE	Cumulative gain value of the estimation dataset
Gain (Validation)	DOUBLE	Cumulative gain value of the validation dataset
Gain (Test)	DOUBLE	Cumulative gain value of the test dataset

10.2.1.5 Distribution of the Binary Target by Category

The `BinaryTarget_CrossStatistics` report gives the distribution of the binary target by category for each variable and partition. It is available for all model types. However, it only applies when the target is binary.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_CrossStatistics"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_CrossStatistics"(PROPERTY_B
AG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.BinaryTarget_CrossStatistics
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Variable	NVARCHAR(512)	Name of the variable
Category	NCLOB	Variable category
Weight	DOUBLE	Weight of the category in the dataset
% Weight	DOUBLE	Weight of the category as a percentage
% Target Negative	DOUBLE	Weight of negative targets as a percentage
% Target Positive	DOUBLE	Weight of positive targets as a percentage
Category Order	INTEGER	Order of the category in the dataset

10.2.1.6 Distribution of the Binary Target by Group

The `BinaryTarget_GroupCrossStatistics` report gives the distribution of the binary target by group for each variable and partition. It is available for all model types. However, it only applies when the target is binary.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_GroupCrossStatistics"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::BinaryTarget_GroupCrossStatistics"(PROPE
RTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.BinaryTarget_GroupCrossStatistics
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Variable	NVARCHAR(512)	Name of the variable
Group	NCLOB	Group of variable categories
Weight	DOUBLE	Weight of the group in the dataset
% Weight	DOUBLE	Weight of the group as a percentage
% Target Negative	DOUBLE	Weight of negative targets as a percentage
% Target Positive	DOUBLE	Weight of positive targets as a percentage
Group Order	INTEGER	Order of the group in the dataset

10.2.2 Multiclass Target

The report in this section provides detailed information about the multiclass target.

10.2.2.1 Distribution of the Multiclass Target

The `MultiClassTarget_Statistics` report gives the distribution of the multiclass target. It is available for all model types. However, it applies only when when the target is nominal with more than two possible values.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::MultiClassTarget_Statistics"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::MultiClassTarget_Statistics"(PROPERTY_BA
G,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.MultiClassTarget_Statistics
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test

Field Name	Type	Description
Target Key	NVARCHAR(512)	Value of the target variable used as the positive target in the model
Category	NCLOB	Name of the category
Weight	DOUBLE	Weight of the category
% Weight	DOUBLE	Weight of the category in the dataset as a percentage

10.2.3 Continuous Target

The reports in this section provide detailed information about the continuous target.

10.2.3.1 Continuous Target Statistics

The `ContinuousTarget_Statistics` report provides descriptive statistics for the continuous target. It is available for all model types. However, it only applies when when the target is continuous.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::ContinuousTarget_Statistics"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ContinuousTarget_Statistics"(PROPERTY_BA
G,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Direction	Type	Required	Description
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ContinuousTarget_Statistics
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Minimum	DOUBLE	Minimum value of the variable
Maximum	DOUBLE	Maximum value of the variable
Mean	DOUBLE	Mean of the variable
Standard Deviation	DOUBLE	Standard deviation of the variable

10.2.3.2 Continuous Target Statistics by Category

The `ContinuousTarget_CrossStatistics` report provides descriptive statistics for the continuous target by category for each variable and partition. It is available for all model types. However, it only applies when when the target is continuous.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ContinuousTarget_CrossStatistics"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ContinuousTarget_CrossStatistics"(PROPER
TY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ContinuousTarget_CrossStatistics
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Variable	NVARCHAR(512)	Name of the variable
Category	NCLOB	Variable category
Weight	DOUBLE	Weight of the category
% Weight	DOUBLE	Weight of the category as a percentage
Target Mean	DOUBLE	Mean value of the target
Target Standard Deviation	DOUBLE	Standard deviation of the target
Category Order	INTEGER	Order of the category in the dataset

10.2.3.3 Continuous Target Statistics by Group

The `ContinuousTarget_GroupCrossStatistics` report provides descriptive statistics for the continuous target by group for each variable and partition. It is available for all model types. However, it only applies when when the target is continuous.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ContinuousTarget_GroupCrossStatistics"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ContinuousTarget_GroupCrossStatistics"(P
ROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ContinuousTarget_GroupCrossStatistics
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Variable	NVARCHAR(512)	Name of the variable
Group	NCLOB	Group of variable categories
Weight	DOUBLE	Weight of the group
% Weight	DOUBLE	Weight of the group as a percentage
Target Mean	DOUBLE	Mean value of the target
Target Standard Deviation	DOUBLE	Standard deviation of the target
Group Order	INTEGER	Order of the group in the dataset

10.3 Time Series Models

The reports in this section provide detailed information about time series models.

10.3.1 Model Overview

The `TimeSeries_ModelOverview` report provides an overview of the model. It is only available for time series models.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ModelOverview"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ModelOverview"(PROPERTY_BAG,M
ETRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_ModelOverview
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Build Date	TIMESTAMP	Date and time when the model was built
Learn Time (sec)	INTEGER	Total learning time (in seconds)
Engine Name	NVARCHAR(512)	Name of the component used to build the model: Kxen.TimeSeries

Field Name	Type	Description
Date Variable	NVARCHAR(512)	Name of the date variable
Target Variable	NVARCHAR(512)	Name of the target variable
First Date	NVARCHAR(512)	First date
Last Date	NVARCHAR(512)	Last date
Horizon	INTEGER	Number of forecast periods
Granularity	NVARCHAR(512)	Time granularity

10.3.2 Model Components

The `TimeSeries_Components` report lists the components of the model. It is only available for time series models.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Components"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Components"(PROPERTY_BAG,METR
IC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_Components
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Component Type	NVARCHAR(512)	Component type of the time series: Trend, Cycles, Fluctuations
Component Value	NVARCHAR(512)	Value of the component

10.3.3 Model Outliers

The `TimeSeries_Outliers` report lists the outliers of the model. It is only available for time series models.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Outliers"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Outliers"(PROPERTY_BAG,METRIC
_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_Outliers
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Date	NVARCHAR(512)	Date of the outlier
Signal	DOUBLE	Actual value
Forecast	DOUBLE	Predicted value

10.3.4 Model Performance by Horizon

The `TimeSeries_PerformanceByHorizon` report provides the per-horizon performance indicators of the model. It is only available for time series models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_PerformanceByHorizon"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_PerformanceByHorizon"(PROPERT
Y_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Direction	Type	Required	Description
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_PerformanceByHorizon
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Horizon	INTEGER	Period number in the forecast horizon
MAE	DOUBLE	Mean absolute error (MAE)
MAPE	DOUBLE	Mean absolute percentage error (MAPE)
SMAPE	DOUBLE	Symmetric mean absolute percentage error (SMAPE)
MPE	DOUBLE	Mean percentage error (MPE)
RMSE	DOUBLE	Root-mean-square error (RMSE)
R2	DOUBLE	Ratio of the variance of the model output with respect to the variance of the target
Error Mean	DOUBLE	Mean of the error
Error Standard Deviation	DOUBLE	Standard deviation of the error

10.3.5 Model Performance

The `TimeSeries_Performance` report provides the horizon-wide performance indicators of the model. It is only available for time series models.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Performance"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Performance"(PROPERTY_BAG,MET
RIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_Performance
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
MAE	DOUBLE	Mean absolute error (MAE)
MAPE	DOUBLE	Mean absolute percentage error (MAPE)
SMAPE	DOUBLE	Symmetric mean absolute percentage error (SMAPE)
MASE	DOUBLE	Mean absolute scaled error (MASE)
MPE	DOUBLE	Mean percentage error (MPE)
RMSE	DOUBLE	Root-mean-square error (RMSE)
R2	DOUBLE	Ratio of the variance of the model output with respect to the variance of the target
Error Mean	DOUBLE	Mean of the error
Error Standard Deviation	DOUBLE	Standard deviation of the error

10.3.6 Model Change Points

The `TimeSeries_ChangePoints` report gives the change points detected in the trend. It is only available for time series models and only when they use a piecewise linear trend.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ChangePoints"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ChangePoints"(PROPERTY_BAG,ME
TRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_ChangePoints
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Date	NVARCHAR(512)	Date of the change point
Change Point	DOUBLE	Predicted value of the change point

10.3.7 Model Decomposition

The `TimeSeries_Decomposition` report gives the impact of each item within a forecasting model. It is only available for time series models.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Decomposition"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_Decomposition"(PROPERTY_BAG,M
ETRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_Decomposition
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Row	INTEGER	Display order
Type	NVARCHAR(512)	Item type: Trend, Cycles, Influencers, Fluctuations, Residuals

Field Name	Type	Description
Item	NVARCHAR(512)	Item from the time series decomposition
Relative Impact	DOUBLE	Impact of the item (the total for a model is 1)

10.3.8 Forecast Breakdown

The `TimeSeries_ForecastBreakdown` report gives the absolute contribution of the different time series model components (trend, cycles, influencers, and fluctuations) for each forecast period. It is only available for time series models.

Prerequisites

The following settings are required:

- The alias `APL/LocalExplanations/Activate` must be set to `'true'`.
- The alias `APL/ActivateExplanations` must be set to `'true'`.
- The alias `APL/ApplyExtraMode` must be set to `'First Forecast with Stable Components and Residues and Error Bars'`.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ForecastBreakdown"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ForecastBreakdown"(PROPERTY_B
AG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_ForecastBreakdown
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Date	NVARCHAR(512)	Date of the forecast period
Row	INTEGER	Display order
Type	NVARCHAR(512)	Item type: Trend, Cycles, Influencers, Fluctuations
Item	NVARCHAR(512)	Item (for example, monthOfYear)
Value	INTEGER	Item value (for example, 9 for September)
Impact	DOUBLE	Absolute impact of the item
Total	DOUBLE	Total value of the forecast period

Related Information

APPLY_MODEL_DEBRIEF [page 102]
FORECAST_AND_DEBRIEF [page 291]

10.3.9 Impact of Cycles

The `TimeSeries_ImpactCycles` report gives the impact of cycles within a forecasting model. It is only available for time series models with the alias `APL/ActivateExplanations` set to `'true'`.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ImpactCycles"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ImpactCycles"(PROPERTY_BAG,ME
TRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_ImpactCycles
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Cycle Rank	INTEGER	Cycle ranking in the model
Cycle Name	NVARCHAR(512)	Cycle name
Row	INTEGER	Display order

Field Name	Type	Description
Period	NVARCHAR(512)	Cycle period (for example, M10 for month)
Season	NVARCHAR(512)	Cycle season (for example, 2021 for year)
Impact	DOUBLE	Impact of the cycle

Related Information

FORECAST [page 281]
FORECAST_AND_DEBRIEF [page 291]

10.3.10 Impact of Numerical Influencers

The `TimeSeries_ImpactInfluencersNumerical` report gives the impact of numerical influencers within a forecasting model. It is only available for time series models with the alias `APL/ActivateExplanations` set to `'true'`.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ImpactInfluencersNumerical"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ImpactInfluencersNumerical"(P
ROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_ImpactInfluencersNumerical
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Influencer Rank	INTEGER	Influencer ranking in the model
Influencer Name	NVARCHAR(512)	Influencer name
Influencer Value	DOUBLE	Influencer value
Impact	DOUBLE	Impact of the influencer

Related Information

FORECAST [page 281]
FORECAST_AND_DEBRIEF [page 291]

10.3.11 Impact of Categorical Influencers

The `TimeSeries_ImpactInfluencersCategorical` report gives the impact of categorical influencers within a forecasting model. It is only available for time series models with the alias `APL/ActivateExplanations` set to `'true'`.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ImpactInfluencersCategorical"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_ImpactInfluencersCategorical
"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_ImpactInfluencersCategorical
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Influencer Rank	INTEGER	Influencer ranking in the model
Influencer Name	NVARCHAR(512)	Influencer name
Category Rank	INTEGER	Category ranking for the influencer
Category Value	NCLOB	Category value
Category Impact	DOUBLE	Impact of the category

Related Information

FORECAST [page 281]
FORECAST_AND_DEBRIEF [page 291]

10.3.12 Variables Ranked by Contribution to Trend

The `TimeSeries_VariablesContribution` report lists all the variables ranked by their contribution to the trend. It is only available for time series models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_VariablesContribution"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_VariablesContribution"(PROPER
TY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_VariablesContribution
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Variable	NVARCHAR(512)	Name of the variable
Rank	INTEGER	Influencer ranking

Field Name	Type	Description
Contribution	DOUBLE	Individual contribution of the variable to the trend
Cumulative	DOUBLE	Cumulative contribution

10.3.13 Cyclic Variable Analysis on the Detrended Signal

The `TimeSeries_DetrendedExtraPredictable` report provides a cyclic variable analysis on the detrended signal. It is only available for time series models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_DetrendedExtraPredictable"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::TimeSeries_DetrendedExtraPredictable"(PR
OPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.TimeSeries_DetrendedExtraPredictable
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Variable	NVARCHAR(512)	Name of the variable
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Category	NCLOB	Variable category
Target Mean	DOUBLE	Mean value of the target
Target Standard Deviation	DOUBLE	Standard deviation of the target mean value

10.4 Classification and Regression Models

The reports in this section provide detailed information about classification and regression models.

10.4.1 List of Correlated Variable Pairs

The `ClassificationRegression_VariablesCorrelation` report lists the pairs of correlated variables. It is only available for classification and regression models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesCorrel
ation"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesCorrel
ation"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ClassificationRegression_VariablesCorrelation
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
First Variable	NVARCHAR(512)	Name of the first variable
Second Variable	NVARCHAR(512)	Name of the second variable
Correlation Coefficient	DOUBLE	Coefficient of the correlation between two variables

10.4.2 Variables Ranked by Contribution to Explaining the Target

The `ClassificationRegression_VariablesContribution` report lists the variables ranked by their contribution to explaining the target. It is only available for classification and regression models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesContri
bution"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesContri
bution"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ClassificationRegression_VariablesContributio
n
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Variable	NVARCHAR(512)	Name of the variable
Rank	INTEGER	Influencer ranking
Contribution	DOUBLE	Individual contribution of the variable
Method	NVARCHAR(512)	Method used to compute the contribution. It can be `ExactSHAP` or `MaximumSmartContribution`.
Cumulative	DOUBLE	Cumulative contribution

10.4.3 Variable Interaction Effect

The `ClassificationRegression_InteractionEffect` report gives the main effects and interaction effects for all variables. It is only available for classification and regression models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_InteractionEffe
ct"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_InteractionEffe
ct"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ClassificationRegression_InteractionEffect
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Variable	NVARCHAR(512)	Name of the variable

Field Name	Type	Description
Main Effect	DOUBLE	Main effect of the variable
Interaction Effect	DOUBLE	Interaction effect of the variable with all the other variables of the model

Related Information

Variable Interactions [page 67]

10.4.4 Variable Interaction Matrix

The `ClassificationRegression_InteractionMatrix` report gives the interaction matrix of the top 6 variables ranked by their interaction effect. It is only available for classification and regression models.

You can use the `APL/InteractionsMaxKept` alias to change the dimension of the interaction matrix. To fill this report, the `APL/Interactions` alias must be set to true.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_InteractionMatr
ix"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_InteractionMatr
ix"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Direction	Type	Required	Description
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ClassificationRegression_InteractionMatrix
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
First Variable	NVARCHAR(512)	Name of the first variable
Second Variable	NVARCHAR(512)	Name of the second variable
Effect	DOUBLE	Interaction effect of the combination of the first and second variables When First Variable = Second Variable (this is the diagonal of the matrix), the effect value corresponds to the main effect of the variable.

Related Information

Variable Interactions [page 67]

10.4.5 List of Excluded Variables

The `ClassificationRegression_VariablesExclusion` report lists the excluded variables together with the reason for their exclusion. It is only available for classification and regression models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesExclus
ion"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesExclus
ion"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ClassificationRegression_VariablesExclusion
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Variable	NVARCHAR(512)	Name of the variable
Target	NVARCHAR(512)	Name of the target
Reason for Exclusion	NVARCHAR(512)	The reason why the variable was removed from the model: • Redundant • Low contributory • Only composed of unacceptably small categories • Constant • Small KI on estimation • Fully compressed

10.4.6 Variable Selection Summary

The `ClassificationRegression_VariablesSelectionSummary` report gives the number of input variables at each step of the auto-selection process and indicates whether the model of that step is the selected model. It is only available for classification and regression models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesSelect
ionSummary"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesSelect
ionSummary"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ClassificationRegression_VariablesSelectionSu
mmary
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Step	INTEGER	Step number of the variable selection process

Field Name	Type	Description
Nb Of Variables	INTEGER	Number of variables in the specified step
Chosen Model	NVARCHAR(512)	Indicates if the model of the specified step is the selected model (Yes or No)

10.4.7 Variable Selection Details

The `ClassificationRegression_VariablesSelectionDetails` report lists the excluded variables at each step of the auto-selection process. It is only available for classification and regression models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesSelect
ionDetails"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesSelect
ionDetails"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ClassificationRegression_VariablesSelectionDe
tails
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Step	INTEGER	Step number of the variable selection process
Variable	NVARCHAR(512)	Name of the variable selected in the specified step and to be excluded in the next step

10.4.8 Variable Selection Performance

The `ClassificationRegression_VariablesSelectionPerformance` report provides the performance of the model at each step of the auto-selection process. It is only available for classification and regression models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesSelect
ionPerformance"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_VariablesSelect
ionPerformance"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ClassificationRegression_VariablesSelectionPe
rformance
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Step	INTEGER	Step number of the variable selection process
Partition	NVARCHAR(512)	Name of the dataset: Validation
Indicator	NVARCHAR(512)	Name of the performance indicator: AUC for gradient boosting binary classification, RMSE for gradient boosting regression, Balanced Classification Rate for gradient boosting multinomial classification, otherwise Predictive Power and Prediction Confidence
Value	DOUBLE	Value of the performance indicator

10.4.9 Binary Classification Confusion Matrix

The `Classification_BinaryClass_ConfusionMatrix` report provides the confusion matrix numbers of the model. It is only available for binary classification models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::Classification_BinaryClass_ConfusionMatr
ix"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Classification_BinaryClass_ConfusionMatr
ix"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Classification_BinaryClass_ConfusionMatrix
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Positive Target	NVARCHAR(512)	Value of the target variable used as the positive target in the model
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Actual	NVARCHAR(512)	Actual observations: Actual All, Actual Negative, Actual Positive
Predicted Positive	DOUBLE	Number of positive predictions
Predicted Negative	DOUBLE	Number of negative predictions
Predicted All	DOUBLE	Total number of predictions
% Predicted Positive	DOUBLE	Number of positive predictions as a percentage
% Predicted Negative	DOUBLE	Number of negative predictions as a percentage
% Predicted All	DOUBLE	Total number of predictions as a percentage

10.4.10 Multinomial Classification Confusion Matrix

The `Classification_MultiClass_ConfusionMatrix` report provides the confusion matrix numbers of the model. It is only available for multinomial classification models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::Classification_MultiClass_ConfusionMatri
x"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Classification_MultiClass_ConfusionMatri
x"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Classification_MultiClass_ConfusionMatrix
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test

Field Name	Type	Description
Actual Class	NVARCHAR(512)	Actual class of the target variable
Predicted Class	NVARCHAR(512)	Predicted class of the target variable
Weight	DOUBLE	Weight in the dataset
% Weight	DOUBLE	Weight in the dataset as a percentage

10.4.11 Multinomial Classification Performance by Class

The `Classification_MultiClass_Performance_by_Class` report provides the performance indicators for each class of the model. It is only available for multinomial classification models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::Classification_MultiClass_Performance_by
_Class"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Classification_MultiClass_Performance_by
_Class"(PROPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Classification_MultiClass_Performance_by_Clas
s
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Class	NVARCHAR(512)	Class of the target variable
Indicator	NVARCHAR(512)	Name of the indicator: F1 Score, Precision, Recall
Value	DOUBLE	Value of the indicator

10.4.12 Model Performance

The `ClassificationRegression_Performance` report provides the performance indicators of the model. It is available for classification models (including multinomial models) and regression models.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_Performance"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::ClassificationRegression_Performance"(PR
OPERTY_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.ClassificationRegression_Performance
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined
Target	NVARCHAR(512)	Name of the target
Partition	NVARCHAR(512)	Name of the dataset: Estimation, Validation, Test
Indicator	NVARCHAR(512)	Name of the indicator (see the table below)
Value	DOUBLE	Value of the indicator

Returned Indicators by Model Type

Regression Models (Legacy and Gradient Boosting)	Binary Classification Models (Legacy and Gradient Boosting)	Multiclass Classification Models (Gradient Boosting)
APE under 100%	Population Ratio	Macro Precision
APE under 5%	Specificity	Macro Recall
Error Mean	Threshold	Macro F1 Score
Error Standard Deviation	False Positive Rate	Micro Precision
Mean AE	Precision	Micro Recall
RMSE	Recall	Micro F1 Score
Max AE	F1 Score	Balanced Classification Rate
Max APE	Log Loss	Balanced Error Rate
MAPE	Predictive Power	Cohen's Kappa
SMAPE	Prediction Confidence	
R2	AUC	
	Classification Rate	
	Balanced Classification Rate	
	Balanced Error Rate	
	Cohen's Kappa	

Related Information

INDICATORS [page 370]

10.5 Deviations

The reports in this section provide detailed information about deviations.

Related Information

Data Comparison [page 20]

10.5.1 Deviations By Variable

The `Deviation_ByVariable` report filters variables that have a deviation indicator over 0.95.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_ByVariable"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_ByVariable"(PROPERTY_BAG,METRI
C_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Deviation_ByVariable
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined.
Variable	NVARCHAR(512)	Name of the variable.
Deviation Indicator	DOUBLE	Deviation estimate ranging from 0 to 1.

10.5.2 Deviations By Category

The `Deviation_ByCategory` report filters categories that have a deviation indicator over 0.95.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_ByCategory"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_ByCategory"(PROPERTY_BAG,METRI
C_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Deviation_ByCategory
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined.
Variable	NVARCHAR(512)	Name of the variable.
Category Order	INTEGER	Order of the category in the dataset.
Category	NCLOB	Variable category.
Deviation Indicator	DOUBLE	Deviation estimate ranging from 0 to 1.

10.5.3 Deviations By Group

The `Deviation_ByGroup` report filters groups that have a deviation indicator over 0.95.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_ByGroup"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_ByGroup"(PROPERTY_BAG,METRIC_B
AG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Deviation_ByGroup
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined.
Target	NVARCHAR(512)	Name of the target.
Variable	NVARCHAR(512)	Name of the variable.
Group Order	INTEGER	Order of the group in the dataset.
Group	NCLOB	Group of variable categories.

Field Name	Type	Description
Deviation Indicator	DOUBLE	Deviation estimate ranging from 0 to 1.

10.5.4 Deviations Category Frequencies

The `Deviation_CategoryFrequencies` report gives the change between a reference dataset and a new dataset by category.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_CategoryFrequencies"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_CategoryFrequencies"(PROPERTY_
BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Deviation_CategoryFrequencies
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined.
Variable	NVARCHAR(512)	Name of the variable.
Category Order	INTEGER	Order of the category in the dataset.
Category	NVARCHAR(512)	Variable category.
Ref Weight	DOUBLE	Weight of the category in the reference dataset.
New Weight	DOUBLE	Weight of the category in the new dataset.
Change	DOUBLE	Change between the reference dataset and the new dataset.
Ref % Weight	DOUBLE	Percentage weight of the category in the reference dataset.
New % Weight	DOUBLE	Percentage weight of the category in the new dataset.
% Change	DOUBLE	Percentage change between the reference dataset and the new dataset.
Abs % Change	DOUBLE	Absolute percentage change between the reference dataset and the new dataset.

10.5.5 Deviations Group Frequencies

The `Deviation_GroupFrequencies` report gives the change between a reference dataset and a new dataset by group.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_GroupFrequencies"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_GroupFrequencies"(PROPERTY_BAG
,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	DEBRIEF_PROPERTY [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.
IN	DEBRIEF_METRIC [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Deviation_GroupFrequencies
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined.
Target	NVARCHAR(512)	Name of the target.
Variable	NVARCHAR(512)	Name of the variable.
Group Order	INTEGER	Order of the group in the dataset.
Group	NVARCHAR(512)	Group of variable categories.
Ref Weight	DOUBLE	Weight of the category in the reference dataset.
New Weight	DOUBLE	Weight of the category in the new dataset.
Change	DOUBLE	Change between the reference dataset and the new dataset.
Ref % Weight	DOUBLE	Percentage weight of the group in the reference dataset.
New % Weight	DOUBLE	Percentage weight of the group in the new dataset.
% Change	DOUBLE	Percentage change between the reference dataset and the new dataset.
Abs % Change	DOUBLE	Absolute percentage change between the reference dataset and the new dataset.

10.5.6 Target Based Deviations By Category

The `TargetBasedByCategory` report filters categories that have a target based deviation over 0.95.

API

```
FUNCTION
"SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_TargetBasedByCategory"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_TargetBasedByCategory"(PROPERT
Y_BAG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Deviation_TargetBasedByCategory
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined.
Target	NVARCHAR(512)	Name of the target.
Variable	NVARCHAR(512)	Name of the variable.
Category Order	INTEGER	Order of the category in the dataset.
Category	NCLOB	Variable category.

Field Name	Type	Description
Deviation Indicator	DOUBLE	Deviation estimate ranging from 0 to 1.

10.5.7 Target Based Deviations By Group

The `Deviation_TargetBasedByGroup` report filters groups that have a target based deviation over 0.95.

API

```
FUNCTION "SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_TargetBasedByGroup"
```

Syntax (Procedure Only)

```
SELECT* FROM
"SAP_PA_APL"."sap.pa.apl.debrief.report::Deviation_TargetBasedByGroup"(PROPERTY_B
AG,METRIC_BAG)
```

Input Tables

Direction	Type	Required	Description
IN	`DEBRIEF_PROPERTY` [page 365]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.
IN	`DEBRIEF_METRIC` [page 364]	Yes	Debrief table produced by the GET_MODEL_DEBRIEF [page 203] function.

Return Type

```
sap.pa.apl.debrief.report::REPORT.T.Deviation_TargetBasedByGroup
```

Field Name	Type	Description
Oid	NVARCHAR(512)	Operation ID set when the model was defined.

Field Name	Type	Description
Target	NVARCHAR(512)	Name of the target.
Variable	NVARCHAR(512)	Name of the variable.
Group Order	INTEGER	Order of the group in the dataset.
Group	NCLOB	Group of variable categories.
Deviation Indicator	DOUBLE	Deviation estimate ranging from 0 to 1.

11 Runtimes

Using the SAP HANA APL EXPORT_APPLY_CODE function, you can export the scoring code of a predictive model built by SAP HANA APL in several programming languages. This allows you to make predictions on systems where SAP HANA APL is not available.

For some exported code types (C++, Java, and Javascript), runtime files are needed to execute the exported code. Reference implementations of these runtimes are provided.

Related Information

SAP HANA APL GitHub Repository - Samples and Runtimes

11.1 C++ Runtime

The C++ runtime supports the following model types:

- Regression
- Classification
- Clustering

For more information, see *Supported Code Export Types*.

Using the C++ Runtime

A detailed description of the usage of the C++ runtime is provided in https://github.com/SAP-samples/hana-apl-apis-runtimes/tree/main/runtimes/cpp#readme .

The sources of a full sample of a C++ executable for Windows and Linux are provided in https://github.com/SAP-samples/hana-apl-apis-runtimes/tree/main/runtimes/cpp/Sample#readme .

Related Information

Supported Code Export Types [page 187]

11.2 Java Runtime

The Java runtime supports the following model types:

- Regression
- Classification
- Clustering

For more information, see *Supported Code Export Types*.

Using the Java Runtime

A compiled version of a Java runtime is provided in https://github.com/SAP-samples/hana-apl-apis-runtimes/blob/main/runtimes/java/KxJRT.jar . It also includes a sample program to read and score text files.

A detailed description and sample of the usage of this Java runtime is provided in https://github.com/SAP-samples/hana-apl-apis-runtimes/tree/main/runtimes/java#readme .

Full documentation of `KxJRT.jar` is available in https://github.com/SAP-samples/hana-apl-apis-runtimes/tree/main/runtimes/java/javadoc as well as in its source files in https://github.com/SAP-samples/hana-apl-apis-runtimes/tree/main/runtimes/java/src .

Related Information

Supported Code Export Types [page 187]

11.3 JavaScript Runtime

SAP HANA APL provides a JavaScript runtime in which you can make unitary predictions based on any predictive model that has been exported in JSON format using the SAP HANA APL EXPORT_APPLY_CODE function.

The runtime supports the following model types:

- Regression and regression (gradient boosting)
- Binary classification (gradient boosting)
- Multiclass classification (gradient boosting)

For more information, see *Supported Code Export Types*.

Requirements

The runtime is a JavaScript module and should be executed in a JavaScript environment, such as a browser or Node.js.

The JavaScript runtime leverages the Asynchronous Module Definition (AMD) API and therefore requires a third-party library that implements this API:

- In-browser environment
  A popular implementation of the AMD API is RequireJS, which can be downloaded from https://requirejs.org/ .
  Some other implementations are also available, like Dojo Toolkit (starting from version 1.7.0).
- Node.js environment
  As an alternative solution for Node.js, you can install the package `amdefine` either globally, if you have enough rights, or locally in the directory where the runtime is located or in any parent directory as follows:
  - Global installation

    ```
    npm install -g amdefine
    ```

  - Local installation

    ```
    cd <runtime directory or any parent directory>
    npm install amdefine
    ```

Related Information

EXPORT_APPLY_CODE [page 183]
Supported Code Export Types [page 187]

11.3.1 Load the JavaScript Runtime

The JavaScript runtime is available in the `https://github.com/SAP-samples/hana-apl-apis-runtimes/tree/main/runtimes/javascript` directory.

The JavaScript runtime consists of two JavaScript files:

- `autoRuntime.js`: This file contains the main runtime.
- `dateCoder.js`: This file provides utilities for date encoding. It is loaded by `autoRuntime.js`.

- In-browser environment
  a. Add the following content to the `index.html` file:

     ```
     <!-- data-main attribute tells to load main.js after require.js is loaded
     -->
     <script
       data-main="<path_to_script_folder>/main"
       src="<path_to_requirejs_folder>/require.js">
     </script>
     ```

b. Add the following content to the `main.js` file:

```
require(
  [... , '<path_to_runtime_folder>/autoRuntime', ...],
  function(..., runtime, ...) {
    // use the runtime here...
  }
);
```

- Node.js environment
  a. Add the following content:

```
var runtime = require('<path_to_runtime_folder>/autoRuntime');
// use the runtime here...
```

Related Information

https://github.com/SAP-samples/hana-apl-apis-runtimes/tree/main/runtimes/javascript

11.3.2 Using the JavaScript Runtime

Once the JavaScript runtime has been loaded, you are ready to use it.

This would be a typical workflow:

1. Create a predictive engine based on the JSON model export
2. Request basic information about the model:
   - Model type
   - Target
   - Target type
3. Get all the model influencers, then let users enter influencer values, and finally collect the influencer values that were entered
4. Ask the engine for a prediction based on the influencer values that were entered

11.3.2.1 Create a Predictive Engine

You create a predictive engine based on the JSON model exported earlier.

Use the `createEngine()` method as follows:

```
var engine = runtime.createEngine(jsonExportAsObject);
```

The input parameter of the `createEngine()` method must be a real JSON object. Depending on how the JSON model was exported, it might be a string. In this case you can simply convert it to an object as follows:

```
var jsonExportAsObject = JSON.parse(jsonExportAsString);
```

11.3.2.2 Retrieve Model Information

Once the predictive engine has been created, you can request some basic information about the model.

Use the `getModelInfo()` method as follows:

```
var modelInfo = engine.getModelInfo();
```

The method returns the following information:

Parameter	Value	Description
`"modelType"`	• "regression" • "binaryClass" • "multiClass"	The type of model
`"target"`	<string>	The name of the target variable
`"targetType"`	• "number" • "integer" • "string"	The type of target variable

11.3.2.3 Get the Model Influencers

The influencers are the variables that influence a target.

1. Use the `getInfluencers()` method as follows:

   ```
   var modelInfluencers = engine.getInfluencers();
   ```

   The `getInfluencers()` method returns an array which contains all the influencers of the model. Each item of the array is an object with the following properties:

Property	Value	Description
`"variable"`	<string>	The name of the influencer
`"valueType"`	• "continuous" • "nominal" • "ordinal"	The value type of the variable
`"storageType"`	• "number" • "integer" • "[u]string" • "date[time]"	The type of value stored in the variable
`"values"`	<array>	An array that contains all the distinct known values of an influencer of type nominal or ordinal integer. A known value is contained in the training dataset.

2. Let users enter influencer values for prediction simulations.

   Using the information about the influencers from the previous step, you could build a dynamic UI to allow users to enter influencer values for prediction simulations.
3. Collect the influencer values that were entered. Format the values as an array, where each item has the following properties:

Property	Value	Description
`"variable"`	<string>	The name of the influencer
`"value"`	<any>	The value of the influencer

11.3.2.4 Get a Prediction

Once the influencer values have been collected and formatted, you can get a prediction by calling the `getScore()` method.

Use the `getScore()` method as follows:

```
var prediction = engine.getScore(values, options);
```

Input parameters:

- `values`: An array containing the values of the influencers, as described in step 3 in *Get the Model Influencers*.
- `options`: An optional object containing the prediction options:

Option	Value	Description
`"interactions"`	• true • false	Set the value to `true` to generate the interactions. The default value is `false`.

The method returns an array with the following properties:

Property	Value	Description
`"score"`	<number>	The prediction score (regression result)
`"decision"`	<any>	The prediction decision (for binary or multi-class classification)
`"proba"`	<number>	The probability of the decision (for binary or multi-class classification)
`"contributionArray"`	<array>	An array that contains the contribution of each influencer

Each item of the contribution array (`contributionArray`) contains the following properties:

Property	Value	Description
`"influencerName"`	<string>	The name of the influencer
`"influencerContribution"`	<number>	The raw contribution of the influencer
`"normalizedContribution"`	<number>	The normalized contribution of the influencer
`"interactions"`	<array>	An array of interactions if `options.interactions` is set to true

normalizedContribution

The `"normalizedContribution"` property contains the z-score of the influencer contribution, which illustrates the relation between the contribution value and the mean of the contributions.

Assuming the contribution values follow a normal distribution, the empirical rule, that is, the three-sigma rule or the 68-95-99.7 rule, can be used:

- 68% of the observations fall between the mean and one standard deviation (sigma).
- 95% of the observations fall between the mean and two standard deviations.
- 99% of the observations fall between the mean and three standard deviations.

Therefore, the strength of each contribution can be evaluated using the normalized value. For example:

- A normalized contribution < 1 would be a weak contribution.
- A normalized contribution >= 1 and < 2 would be a meaningful contribution.
- A normalized contribution > 2 would be a strong contribution.

options.interactions

If `options.interactions` is set to true, the influencer contribution contains an additional property called `"interactions"` with an array of interaction values:

Property	Type	Description
`"variable"`	<string>	The name of the influencer the current influencer is interacting with
`"interaction"`	<number>	The value of the interaction

Example

An example of a prediction object is shown below:

```
{
  "score": 1.234,         // The final prediction for a regression
  "decision": "1",        // The predicted class for a binary or multi-class
classification
  "proba": 0.789,         // The prediction probability for a binary or
multiclass classification
  "contributionArray": [
    {
      "influencerName": "var1",
```

```
      "influencerContribution": 1.123,
      "normalizedContribution": 1.456,
      "interactions": [
        {
          "variable": "var1",
          "interaction": 5.432   // Interaction btw. 'var1' and 'var1' = the
main effect for 'var1'
        },
        {
          "variable": "var2",
          "interaction": 1.234
        },
        ...
      ]
    },
    ...
  ]
}
```

Note the following about interactions:

- Interactions are available for gradient boosting models only.
- The `interactions` array contains a value for the interaction between the current influencer and itself. This value is called the main effect. The sum of all other interaction values is called the interaction effect.
- For a given influencer, the sum of all the interaction values, that is, the main effect plus the interaction effect, is equal to the influencer contribution.

Related Information

Get the Model Influencers [page 483]

12 Sample Use Case: Predicting Auto Insurance Fraud

A use case demonstrating how SAP HANA APL can be used with SAP HANA Studio to anticipate potential fraudulent insurance claims.

This example shows how an insurance company assesses past insurance frauds in order to create a category of clients that may be susceptible to make fraudulent claims.

> **Note**
>
> The scripts and the sample dataset are part of the package.

The first step of the analysis is to prepare the main input tables. One of them contains data that has already been analyzed, that contains some known fraud cases. This table is used to train the model. The results are used to indicate which variables to use as the target and describe the claims data.

After considering past data and past fraudulent claims, you train the model using that data in order to produce an updated model that will be applied to the new data in order to detect potential fraud risks. After training the model, the `CREATE_MODEL_AND_TRAIN` function returns summary information regarding the model as well as indicators like the Predictive Power (KI) of the model or the Prediction Confidence of the results (KR). At the end of the data mining process, the function produces scores in the form of a table that can be queried.

12.1 The Business Data

The business data for this use case can be found in two datasets (Past Claims and New Claims) that are part of the samples provided with SAP HANA APL.

The Past Claims Dataset

The Past Claims dataset is used to train the model in order to create a model adapted to the dataset and the analysis requirements.

This dataset contains 2,000 rows, mainly legitimate claims with a few fraudulent claims. The data with known fraud allows you to learn about fraud. Here is a preview of the past claims dataset.

Claim id	Days to report	Bodily injury amount	Property damage	Previous claims	Payment method	is rear end collision	Prem amount	Age	Gender	...	policy holder	is fraud
CL_0000765	8	0	1957	0	CC	No	Safedriving...	52	Male	...	Y	No
CL_0000832	30	2541	3843	0	CC	No	Safedriving...	85	Female	...	N	No
CL_0002015	4	0	25719	0	CC	No	Standard	45	Male	...	N	No
CL_0002854	0	0	83	1	Auto	No	Standard	75	Male	...	N	No
CL_0002869	22	0	1264	0	CC	Yes	Standard	48	Female	...	N	No
CL_0003400	3	9903	7333	0	Auto	No	Safedriving...	41	Male	...	N	Yes
CL_0005084	14	0	1882	0	CC	No	Safedriving...	26	Female	...	Y	No
CL_0005346	12	15399	8864	0	Auto	No	Standard	73	Male	...	N	No
CL_0005677	23	2577	8883	0	CC	Yes	Safedriving...	62	Male	...	N	No
CL_0005897	15	0	6390	0	CC	Yes	Standard	56	Male	...	N	No
CL_0006217	15	0	9936	0	Auto	No	Standard	27	Female	...	N	No
CL_0006348	13	0	1881	0	Auto	No	Standard	88	Male	...	N	No
CL_0006535	0	0	145	0	CC	No	Standard	56	Female	...	N	No
CL_0006728	27	0	2046	2	CH	No	Standard	32	Female	...	N	No

The New Claims Dataset

The New Claims dataset contains 147 recent claims for which we want to evaluate the risk of fraud.

The New Claims dataset follows the same structure as the Past Claims dataset except that it doesn't have the target column is_fraud. Here is a preview of the dataset.

Claim id	Days to report	Bodily injury amount	Property damage	Previous claims	Payment method	is rear end collision	Prem amount	Age	Gender	...	policy holder
CL_0959524	6	0	1066	0	CH	Yes	Standard	97	Male	...	N
CL_0959946	0	0	432	0	CH	Yes	Standard	90	Female	...	N
CL_0960121	5	0	2714	0	CC	No	Standard	23	Male	...	Y
CL_0960195	26	0	7193	0	CC	Yes	Safedriving..	55	Male	...	Y
CL_0960294	15	23385	28730	0	Auto	No	Standard	40	Female	...	N
CL_0960379	20	14718	7258	0	Auto	No	Standard	45	Female	...	N
CL_0960411	0	0	46	0	CH	Yes	Standard	77	Male	...	Y
CL_0960946	25	0	5843	0	Auto	Yes	Standard	80	Male	...	Y
CL_0961067	30	0	2583	0	CC	Yes	Standard	27	Male	...	N
CL_0961964	11	0	8713	0	CC	No	Standard	42	Female	...	N
CL_0962914	30	15936	20244	0	CC	Yes	Standard	47	Male	...	Y
CL_0963254	14	0	1621	0	Auto	No	Safedriving..	70	Male	...	N
CL_0963313	15	0	9936	0	Auto	No	Standard	27	Female	...	N
CL_0964670	7	0	13034	0	CC	No	Standard	79	Male	...	Y

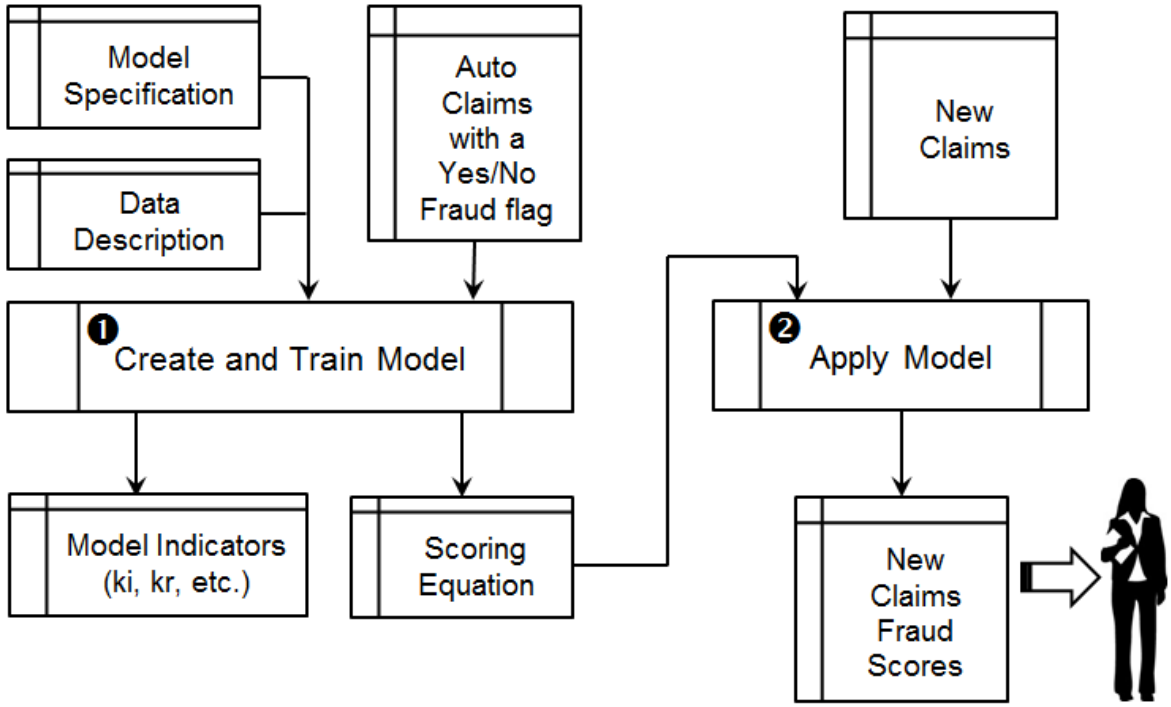
12.2 Overall Workflow

A global picture of the sample use case.

In this example you use two functions:

- `CREATE_MODEL_AND_TRAIN` to create a classification model and train it against the past claims
- `APPLY_MODEL` to calculate a fraud prediction score for each individual recent claim

These two functions are represented with their inputs and outputs in the following diagram:



> **Note**
>
> The APL functions don't come preconfigured with the installation. If you use the direct technique, they must be created using the AFL system procedure `SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE`. See Calling a Function with the Direct Technique [page 47].

12.3 The Learning Phase

You use input tables that describe the fraud prediction model to train the model. When the model training is completed, the SAP HANA APL function provides the learning results in the form of tables.

12.3.1 Training Inputs

Here are the input tables that describe the fraud prediction model and the claims data. Those tables are read by the `CREATE_MODEL_AND_TRAIN` function.

FUNC_HEADER

key	value
Oid	Claims

CREATE_AND_TRAIN_CONFIG

key	value
APL/ModelType	regression/classification
APL/CuttingStrategy	random with no test

VARIABLE_DESC

rank	name	storage	value type	key level	order level	missing string	group name	description
0	CLAIM_ID	string	nominal	1	0			Unique Identifier of a claim
1	DAYS_TO_REPORT	integer	continuous	0	0			
2	BODILY_INJURY_AMOUNT	integer	continuous	0	0			
3	PROPERTY_DAMAGE	integer	continuous	0	0			
4	PREVIOUS_CLAIMS	integer	ordinal	0	0			Number of previous claims
5	PAYMENT_METHOD	string	nominal	0	0			
6	IS_REAR_END_COLLISION	string	nominal	0	0			
7	PREM_AMOUNT	string	nominal	0	0			
8	AGE	integer	continuous	0	0			
9	GENDER	string	nominal	0	0			
10	MARITAL_STATUS	string	nominal	0	0			
11	INCOME_ESTIMATE	number	continuous	0	0			
12	INCOME_CATEGORY	integer	ordinal	0	0			
13	POLICY_HOLDER	string	nominal	0	0			
14	IS_FRAUD	string	nominal	0	0			Yes/No flag

The `CLAIM_ID` column is a key.

VARIABLE_ROLES

name	role

-

Since the column that stores the information to predict is not indicated, the Automated Analytics engine considers the last column (which happens to be `IS_FRAUD`) as the target variable.

12.3.2 Training Outputs

When the model training is completed, the function provides the learning results in the form of tables. You can then run an SQL query in SAP HANA Studio against the table `INDICATORS` in order to know how good the model is.

```
select
 case when key = 'PredictivePower' then 'Model KI' else 'Model KR' end as
"Quality Indicators",
```

```
 round(to_double(value) *100 , 2) as "Percent Value"
from
 INDICATORS
where
 OID = 'Claims' and VARIABLE = 'IS_FRAUD' and KEY in
('PredictivePower','PredictionConfidence')
```

Quality Indicators	Percent Value
Model KI	68.00
Model KR	94.68

Let's query that same table to display how much each variable contributes in explaining the fraud.

```
select
 OID as "Model Name",
 row_number() OVER (partition by OID order by VALUE desc) as "Rank",
 VARIABLE as "Explanatory Variable",
 round(to_double(VALUE) *100 , 2) as "Individual Contribution",
 round(sum(to_double(VALUE)) OVER (partition by OID order by VALUE desc) *100
,2)
 as "Cumulative Contribution"
from
 INDICATORS
where
 OID = 'Claims'  and TARGET = 'IS_FRAUD' and
 KEY = 'MaximunSmartVariableContribution'
order by 4 desc
```

Model Name	Rank	Explanatory Variable	Individual Contribution	Cumulative Contribution
Claims	1	BODILY_INJURY_AMOUNT	46.30	46.30
Claims	2	DAYS_TO_REPORT	15.02	61.33
Claims	3	AGE	10.78	72.10
Claims	4	INCOME_CATEGORY	7.45	79.55
Claims	5	PAYMENT_METHOD	4.56	84.11
Claims	6	GENDER	3.98	88.09
Claims	7	PROPERTY_DAMAGE	3.83	91.92
Claims	8	INCOME_ESTIMATE	3.44	95.36
Claims	9	IS_REAR_END_COLLISION	3.08	98.43
Claims	10	PREM_AMOUNT	1.57	100.00

Another table called SUMMARY allows you to check how long it takes for the Automated Analytics engine to learn from the past data.

```
select
 case key
   when 'ModelVariableCount'           then 'Initial Number of Variables'
   when 'ModelSelectedVariableCount'    then 'Number of Explanatory Variables'
   when 'ModelRecordCount'             then 'Number of Records'
   when 'ModelLearningTime'            then 'Time to learn in seconds'
   else null
  end as "Training Summary",
 to_double(value) as "Value"
from
 SUMMARY
```

```
where
 OID = 'Claims' and (KEY ='ModelLearningTime' or KEY like 'Model%Count')
order by 1
```

Training Summary	Value
Initial Number of Variables	15
Number of Explanatory Variables	13
Number of Records	2,000
Time to learn in seconds	1

Out of the 15 variables available in the dataset, the engine selects the `IS_FRAUD` variable as the target and automatically excludes the `CLAIM_ID` variable knowing that it is a key.

12.4 The Prediction Phase

After applying the classification model, you are able to communicate to the business people the results of the prediction made by the Automated Analytics engine.

12.4.1 Apply Inputs

Here are the input tables prepared before calling the `APPLY_MODEL` function.

FUNC_HEADER

key	value
Oid	Claims

You are reusing here the same table that was defined previously for the `CREATE_MODEL_AND_TRAIN` function.

APPLY_CONFIG

key	value
APL/ApplyExtraMode	Decision

You indicate in the `APPLY_CONFIG` table that, in addition to the score values, you want to obtain the decision information telling you whether a claim is determined to be fraudulent.

MODEL_TRAIN_BIN

OID	FORMAT	LOB
Claims	application/vnd.sap.aa	KXEN SERIAL...

The table `MODEL_TRAIN_BIN` was generated during the learning phase by the `CREATE_MODEL_AND_TRAIN` function and is read by the `APPLY_MODEL` function to retrieve the model definition and the scoring equation.

12.4.2 Apply Outputs

After applying the classification model, you are able to communicate to the business people the results of the prediction made by the Automated Analytics engine. You can check the list of all scores showing first the claims that are the most likely to be fraudulent.

```
select
 CLAIM_ID as "Claim ID",
  case "decision_rr_IS_FRAUD"
   when 'Yes' then 'Fraudulent Claim'
   when 'No'  then 'Legitimate Claim'
   else Null
  End as "Prediction",
  round("proba_decision_rr_IS_FRAUD" *100,2) as "Percent Likelihood",
  round("rr_IS_FRAUD" * 100 ,2) as "Fraud Score"
  from
   CLAIMS_SCORES
  order by 2 asc, 3 desc, 4 desc
```

Claim ID	Prediction	Percent Likelihood	Fraud Score
CL_1031721	Fraudulent Claim	45.44	78.94
CL_0995635	Fraudulent Claim	45.44	77.99
CL_0977425	Fraudulent Claim	45.44	75.17
CL_0995311	Fraudulent Claim	45.44	74.12
CL_0995189	Fraudulent Claim	45.44	72.27
CL_0970439	Fraudulent Claim	45.43	68.03
CL_0982255	Fraudulent Claim	45.43	68.03
CL_1019445	Fraudulent Claim	45.43	66.33
CL_0991940	Fraudulent Claim	45.43	60.37
CL_1027985	Fraudulent Claim	45.43	58.83
CL_0973639	Fraudulent Claim	45.43	58.08
CL_0983978	Fraudulent Claim	45.43	58.08
CL_0999780	Legitimate Claim	100.00	-2.27
CL_1031836	Legitimate Claim	100.00	-2.33
CL_0995280	Legitimate Claim	100.00	-2.48
CL_1014322	Legitimate Claim	100.00	-2.68
CL_1009901	Legitimate Claim	100.00	-2.73
CL_1033533	Legitimate Claim	100.00	-2.73
CL_0967006	Legitimate Claim	100.00	-2.84
...	...	...	...

You can also take the business dataset with the recent claims, add to it the fraud prediction information and filter the list to show only the determined fraudulent claims.

```
select
```

```
 N.*,
  round("proba_decision_rr_IS_FRAUD" *100,2) as "Fraud Likelihood",
  round("rr_IS_FRAUD" * 100 ,2) as "Fraud
          Score"
  from
    CLAIMS_SCORES S, APL_SAMPLE_DATA.AUTO_CLAIMS_NEW N
  where
    S.CLAIM_ID = N.CLAIM_ID and S."decision_rr_IS_FRAUD" = 'Yes'
  order by S."proba_decision_rr_IS_FRAUD" desc, S."rr_IS_FRAUD" desc
```

Claim id	Days to report	Bodily injury amount	Property damage	Previous claims	Payment method	is rear end collision	Prem amount	Age	Gender	...	Fraud likelyhood	Fraud score
CL_1031721	30	29,685	21,630	0	Auto	Yes	Standard	19	Female	...	45.44	78.94
CL_0995635	2	27,090	1,518	1	CC	No	Standard	53	Female	...	45.44	77.99
CL_0977425	30	24,978	7,059	0	Auto	No	Standard	94	Male	...	45.44	75.17
CL_0995311	25	25,599	1,028	0	CH	No	Standard	59	Male	...	45.44	74.12
CL_0995189	27	25,662	992	0	CH	No	Standard	48	Male	...	45.44	72.27
CL_0982255	18	19,239	11,681	0	CC	Yes	Standard	32	Male	...	45.43	68.03
CL_0970439	18	19,239	11,681	0	CC	Yes	Standard	32	Male	...	45.43	68.03
CL_1019445	9	18,591	1,195	0	CC	Yes	Standard	93	Male	...	45.43	66.33
CL_0991940	27	2,715	1,464	0	Auto	Yes	Safedriving...	91	Female	...	45.43	60.37
CL_1027985	16	9,162	4,512	0	CC	Yes	Standard	35	Female	...	45.43	58.83
CL_0983978	16	4,599	9,045	0	CH	No	Standard	53	Female	...	45.43	58.08
CL_0973639	16	4,599	9,045	0	CH	No	Standard	53	Female	...	45.43	58.08

13 Troubleshooting

This chapter provides guidelines for resolving common problems that might occur.

13.1 Check if SAP HANA APL is Correctly Loaded

You can check the indexserver logs to see why SAP HANA APL was not correctly loaded. To do so:

1. Launch SAP HANA Studio.
2. Open your SAP HANA instance administration screen (Double-click on SAP HANA instance or right-click->Configuration and Monitoring->Open Administration)
3. Open the Diagnosis Files tab.
4. Sort by "Modified" and open the latest indexserver logs.
5. If necessary, display more lines and search for keyword "APL" to find the related error messages.

> **Note**
>
> Don't forget to provide this error log when contacting support.

13.2 Script Error During Stored Procedure Creation

If the SQL fails when creating the `CREATE_MODEL_AND_TRAIN_SIGNATURE` table, or when executing an "insert" statement, then it probably means that you have not run the `apl_create_table_types.sql` script or that you have not created a table type to describe the dataset (`ADULT01_T`).

If the SQL fails when creating the procedure:

```
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_MODEL_AND_TRAIN','USER_AP
L', 'APLWRAPPER_CREATE_MODEL_AND_TRAIN', CREATE_MODEL_AND_TRAIN_SIGNATURE);
```

Then it might be because the signature of the procedure (input and output table types) does not match the expected one.

```
-- ------------------------------------------------------------------------
-- Create AFL wrappers for the APL function
-- ------------------------------------------------------------------------
-- the AFL wrapper generator needs the signature of the expected stored proc
drop table CREATE_MODEL_AND_TRAIN_SIGNATURE;
create column table CREATE_MODEL_AND_TRAIN_SIGNATURE  like PROCEDURE_SIGNATURE_T;
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (1,
'USER_APL','FUNCTION_HEADER_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (2,
'USER_APL','OPERATION_CONFIG_T', 'IN');
```

```
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (3,
'USER_APL','VARIABLE_DESC_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (4,
'USER_APL','VARIABLE_ROLES_T', 'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (5, 'USER_APL','ADULT01_T',
'IN');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (6,
'USER_APL','MODEL_BIN_OID_T', 'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (7,
'USER_APL','OPERATION_LOG_T', 'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (8, 'USER_APL','SUMMARY_T',
'OUT');
insert into CREATE_MODEL_AND_TRAIN_SIGNATURE values (9,
'USER_APL','INDICATORS_T', 'OUT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('USER_APL','APLWRAPPER_CREATE_MODEL_AND_TRAIN'
);
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('APL_AREA','CREATE_MODEL_AND_TRAIN','USER_AP
L', 'APLWRAPPER_CREATE_MODEL_AND_TRAIN', CREATE_MODEL_AND_TRAIN_SIGNATURE);
```

13.3 Error During Stored Procedure Execution

While running the following procedure:

```
DO BEGIN
    header   = select * from FUNC_HEADER;
    config   = select * from CREATE_AND_TRAIN_CONFIG;
    var_desc = select * from VARIABLE_DESC;
    var_role = select * from VARIABLE_ROLES;
    dataset  = select * from APL_SAMPLES.ADULT01;

APLWRAPPER_CREATE_MODEL_AND_TRAIN(:header, :config, :var_desc,:var_role, :dataset
, out_model,  out_log, out_sum, out_indic);
    insert into  "USER_APL"."MODEL_TRAIN_BIN" select * from :out_model;
    insert into  "USER_APL"."OPERATION_LOG"   select * from :out_log;
    insert into  "USER_APL"."SUMMARY"         select * from :out_sum;
    insert into  "USER_APL"."INDICATORS"      select * from :out_indic;
END;
```

If the procedure execution fails and the procedure creation does not happen, then it might be because of the following issues:

- SAP HANA issue (No reference to SAP HANA APL or Automated Analytics)
  - The procedure parameters do not match the expected type defined, for example: the `APL_SAMPLES_ADULT01` table does not match the `ADULT01_T` table type defined in the procedure creation. The error message usually include the following information "invalid column name".
- SAP HANA APL issue (Error message contains AFLFunctionFatal exception[APL error] Automated Analytics error)
  - One of the parameter defined in the input tables is incorrect (for example the VARIABLE_ROLES table reference a variable not present in the dataset, or the CONFIG table contains uncorrect entries). The error message usually include the following info `"KxCommonInterf::IKxenParameter::setSubValue()"`
  - The variable description is incorrect (duplicate entry, wrong variable name/desc ...). The error message usually include the following info `""KxCommonInterf::IKxenModel::readSpaceDescription()""`

13.4 Supporting SAP HANA Cloud Multitenancy

When you use the procedure method to call APL in an SAP HANA Cloud multitenant scenario, you must use the APL_CACHE_SCHEMA session variable.

Using the procedure method to call APL in an SAP HANA Cloud tenant triggers an error message:

```
Object is out of boundary. Tenant user cannot create an object in a non-tenant schema.
```

A tenant user cannot write outside the tenant database. The procedure mode tries to create objects that are stored in a cache (technically, a shared schema SAP_PA_APL) outside the tenant database, and this fails.

The workaround uses the session variable APL_CACHE_SCHEMA, which allows changing the schema where APL stores its cache. Within an SAP HANA Cloud tenant, two possible values for this session variable exist:

- Use the current schema of the tenant user to store APL cache:
  ```
  SET 'APL_CACHE_SCHEMA' = CURRENT_USER;
  ```
- Create a schema in the tenant and use it to store APL cache:
  ```
  SET 'APL_CACHE_SCHEMA' = '<schema_name>';
  ```

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

© 2025 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP