

SAP HANA Platform SPS 12
Document Version: 1.2 – 2018-01-24

SAP HANA Predictive Analysis Library (PAL)



Content

1	SAP HANA Predictive Analysis Library (PAL).....	5
2	Getting Started with PAL.....	7
2.1	Prerequisites.....	7
2.2	Application Function Library (AFL).....	7
2.3	Security.....	8
2.4	Checking PAL Installation.....	8
2.5	Calling PAL Functions.....	9
	Parameter Table Structure.....	11
	Exception Handling.....	12
2.6	Using PAL in SAP HANA AFM.....	13
3	PAL Functions.....	15
3.1	Clustering Algorithms.....	19
	Affinity Propagation.....	19
	Agglomerate Hierarchical Clustering.....	24
	Anomaly Detection.....	31
	Cluster Assignment.....	37
	DBSCAN.....	46
	Gaussian Mixture Model (GMM).....	52
	K-Means.....	62
	K-Medians.....	75
	K-Medoids.....	81
	LDA Estimation and Inference.....	88
	Self-Organizing Maps.....	101
	Slight Silhouette.....	111
	Incremental Clustering on SAP HANA Smart Data Streaming.....	115
3.2	Classification Algorithms.....	116
	Area Under Curve (AUC).....	116
	Back Propagation Neural Network.....	122
	C4.5 Decision Tree.....	135
	CART Decision Tree.....	142
	CHAID Decision Tree.....	147
	Confusion Matrix.....	154
	KNN.....	158
	Logistic Regression (with Elastic Net Regularization).....	162
	Multi-Class Logistic Regression.....	176

Naive Bayes.	.190
Parameter Selection and Model Evaluation (PSME).	.198
Predict with Tree Model.	.211
Random Forest.	.214
Support Vector Machine.	.222
Incremental Classification on SAP HANA Smart Data Streaming.	.235
3.3 Regression Algorithms.	.236
Bi-Variate Geometric Regression.	.236
Bi-Variate Natural Logarithmic Regression.	.243
Exponential Regression.	.251
Multiple Linear Regression.	.259
Polynomial Regression.	.270
3.4 Association Algorithms.	.278
Apriori.	.278
FP-Growth.	.297
K-Optimal Rule Discovery (KORD).	.309
3.5 Time Series Algorithms.	.314
ARIMA.	.314
Auto ARIMA.	.332
Brown Exponential Smoothing.	.347
Croston's Method.	.354
Forecast Accuracy Measures.	.358
Forecast Smoothing.	.362
Linear Regression with Damped Trend and Seasonal Adjust.	.381
Single Exponential Smoothing.	.387
Double Exponential Smoothing.	.393
Triple Exponential Smoothing.	.400
Seasonality Test.	.409
Trend Test.	.415
White Noise Test.	.419
3.6 Preprocessing Algorithms.	.423
Binning.	.423
Binning Assignment.	.429
Convert Category Type to Binary Vector.	.433
Inter-Quartile Range Test.	.436
Partition.	.440
Posterior Scaling.	.444
Principal Component Analysis (PCA).	.448
Random Distribution Sampling.	.456
Sampling.	.460
Scaling Range.	.470

Substitute Missing Values.	475
Variance Test.	480
3.7 Statistics Algorithms.	484
Chi-Squared Test for Goodness of Fit.	484
Chi-Squared Test for Independent.	487
Cumulative Distribution Function.	491
Distribution Fitting.	494
Grubbs' Test.	504
Kaplan-Meier Survival Analysis.	509
Multivariate Statistics.	515
Quantile Function.	518
Univariate Statistics.	522
Variance Equal Test.	526
3.8 Social Network Analysis Algorithms.	530
Link Prediction.	530
3.9 Miscellaneous.	534
ABC Analysis.	534
Weighted Score Table.	537
4 End-to-End Scenarios.	542
4.1 Scenario: Predict Segmentation of New Customers for a Supermarket.	542
4.2 Scenario: Analyze the Cash Flow of an Investment on a New Product.	546
4.3 Scenario: Survival Analysis.	555
5 Best Practices.	566
6 Important Disclaimer for Features in SAP HANA Platform, Options and Capabilities.	567

1 SAP HANA Predictive Analysis Library (PAL)

This reference describes the Predictive Analysis Library (PAL) delivered with SAP HANA. This application function library (AFL) defines functions that can be called from within SAP HANA SQLScript procedures to perform analytic algorithms.

SAP HANA's SQLScript is an extension of SQL. It includes enhanced control-flow capabilities and lets developers define complex application logic inside database procedures. However, it is difficult to describe predictive analysis logic with procedures.

For example, an application may need to perform a cluster analysis in a huge customer table with 1T records. It is impossible to implement the analysis in a procedure using the simple classic K-means algorithms, or with more complicated algorithms in the data-mining area. Transferring large tables to the application server to perform the K-means calculation is also costly.

The Predictive Analysis Library (PAL) defines functions that can be called from within SQLScript procedures to perform analytic algorithms. This release of PAL includes classic and universal predictive analysis algorithms in nine data-mining categories:

- Clustering
- Classification
- Regression
- Association
- Time Series
- Preprocessing
- Statistics
- Social Network Analysis
- Miscellaneous

The algorithms in PAL were carefully selected based on the following criteria:

- The algorithms are needed for SAP HANA applications.
- The algorithms are the most commonly used based on market surveys.
- The algorithms are generally available in other database products.

PAL on SAP HANA Smart Data Streaming

PAL also provides several incremental machine learning algorithms that learn and update a model on the fly, so that predictions are based on a dynamic model.

For detailed information, see the section on machine learning with streaming in the *SAP HANA Smart Data Streaming: Developer Guide*.

i Note

SAP HANA Smart Data Streaming is only supported on Intel-based hardware platforms.

2 Getting Started with PAL

This section covers the information you need to know to start working with the SAP HANA Predictive Analysis Library.

2.1 Prerequisites

To use the PAL functions, you must:

- Install SAP HANA Platform 1.0 SPS 12.
- Install the Application Function Library (AFL), which includes the PAL.
For information on how to install or update AFL, see the section on installing or updating SAP HANA components in *SAP HANA Server Installation and Update Guide*:

 Note

The revision number of the AFL must match the revision number of SAP HANA. See SAP Note 1898497 for details.

- Enable the Script Server in HANA instance. See SAP Note 1650957 for further information.

Related Information

[SAP Note 1898497](#) 

[SAP Note 1650957](#) 

2.2 Application Function Library (AFL)

You can dramatically increase performance by executing complex computations in the database instead of at the application sever level. SAP HANA provides several techniques to move application logic into the database, and one of the most important is the use of application functions. Application functions are like database procedures written in C++ and called from outside to perform data intensive and complex operations.

Functions for a particular topic are grouped into an application function library (AFL), such as the Predictive Analysis Library (PAL) and the Business Function Library (BFL). Currently, PAL and BFL are delivered in one archive (that is, one SAR file with the name AFL<version_string>.SAR). The AFL archive is not part of the HANA appliance, and must be installed separately by the administrator.

2.3 Security

This section provides detailed security information which can help administrator and architects answer some common questions.

Role Assignment

For each AFL area, there are two roles. You must be assigned one of the roles to execute the functions in the library. The roles for the PAL library are automatically created when the Application Function Library (AFL) is installed. The role names are:

```
AFL__SYS_AFL_AFLPAL_EXECUTE  
AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION
```

i Note

There are 2 underscores between AFL and SYS.

To generate or drop PAL procedures, you also need the following role, which is created when SAP HANA is installed:

```
AFLPM_CREATOR_ERASER_EXECUTE
```

i Note

Once the above roles are automatically created, they cannot be dropped. In other words, even when an area with all its objects is dropped and re-created during system startup, the user still keeps these roles originally granted.

2.4 Checking PAL Installation

To confirm that the PAL functions were installed successfully, you can check the following three public views:

- sys.afl_areas
- sys.afl_packages
- sys.afl_functions

These views are granted to the PUBLIC role and can be accessed by anyone.

To check the views, run the following SQL statements:

```
SELECT * FROM "SYS"."AFL.Areas" WHERE AREA_NAME = 'AFLPAL';  
SELECT * FROM "SYS"."AFL.Packages" WHERE AREA_NAME = 'AFLPAL';  
SELECT * FROM "SYS"."AFL.Functions" WHERE AREA_NAME = 'AFLPAL';
```

The result will tell you whether the PAL functions were successfully installed on your system.

2.5 Calling PAL Functions

To use PAL functions, you must do the following:

- From within SQLScript code, generate a procedure that wraps the PAL function.
- Call the procedure, for example, from an SQLScript procedure.

Step 1 – Generate a PAL Procedure

Any user granted with the AFLPM_CREATOR_ERASER_EXECUTE role can generate an AFLLANG procedure for a specific PAL function. The syntax is shown below:

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('<area_name>', '<function_name>',
'<schema_name>', '<procedure_name>', <signature_table>);
```

- `<area_name>`: Always set to AFLPAL.
- `<function_name>`: A PAL built-in function name.
- `<schema_name>`: A name of the schema that you want to create.
- `<procedure_name>`: A name for the PAL procedure. This can be anything you want.
- `<signature_table>`: A user-defined table variable. The table contains records to describe the position, schema name, table type name, and parameter type, as defined below:

```
(  
  POSITION      int,  
  SCHEMA_NAME   nvarchar(256),  
  TYPE_NAME     nvarchar(256),  
  PARAMETER_TYPE varchar(7)  
)
```

A typical table variable references a table with the following definition:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT

i Note

1. The `SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE` is in invoker mode, which means, the invoker must be allowed to read the table.
2. The records in the signature table must follow this order: first input table types, next parameter table type, and then output table types.

3. The signature table must be created before generating the PAL procedure. The table type names are user-defined. You can find detailed table type definitions for each PAL function in Chapter 3.
4. The names of all the generated procedures and the procedure parameter table types must be unique. The procedure names are defined by users. When generating a PAL procedure, make sure you give a unique procedure name.
5. If you want to drop an existing procedure and then generate it again, you need to call the `SYS.AFLLANG_WRAPPER_PROCEDURE_DROP` procedure to clear the existing procedure. The syntax is as follows:


```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP ('<schema_name>', '<procedure_name>');
```
6. The PAL procedure can be created under any schema once you have the `CREATE ANY` privileges of it.
7. The current PAL does not support the decimal data type. If you have decimal columns in the base tables and want them to be replaced by double in the generated table types, you need to add the following entry in the signature table:


```
(-2, '_SYS_AFL', 'CAST_DECIMAL_TO_DOUBLE', 'INOUT');
(-1, '_SYS_AFL', 'CREATE_TABLE_TYPES', 'INOUT');
```

Then the real tables or types should be inserted starting with ID1, as shown below:

Position	Schema Name	Table Type Name	Parameter Type
-2	_SYS_AFL	CAST_DECIMAL_TO_DOUBLE	INOUT
-1	_SYS_AFL	CREATE_TABLE_TYPES	INOUT
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT

The implicit decimal-to-double conversion is done on SQL layer when a generated procedure with tables including decimal columns is called. Note that the PAL data precision will NOT be higher than double.

8. The AFLLANG procedure generator described in this Step was introduced since SAP HANA SPS 09. For backward compatibility information, see SAP Note 2046767.

Step 2 – Call a PAL Procedure

After generating a PAL procedure, any user that has the `AFL__SYS_AFL_AFLPAL_EXECUTE` or `AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION` role can call the procedure, using the syntax below.

```
CALL <schema_name>.<procedure_name>(
  <data_input_table> {,...},
  <parameter_table>,
  <output_table> {,...}) with overview;
```

- `<schema_name>`: The name of the schema where the procedure is located.
- `<procedure_name>`: The procedure name specified when generating the procedure in Step 1.
- `<data_input_table>`: User-defined name(s) of the procedure's input table(s). Detailed input table definitions for each procedure can be found in Chapter 3.

- <parameter_table>: User-defined name of the procedure's parameter table. The table structure is described in [Parameter Table Structure \[page 11\]](#). Detailed parameter table definition for each procedure can be found in Chapter 3.
- <output_table>: User-defined name(s) of the procedure's output table(s). Detailed output table definition for each procedure can be found in Chapter 3.

i Note

1. The input, parameter, and output tables must be created before calling the procedure.
2. Some PAL algorithms have more than one input table or more than one output table.
3. To call the PAL procedure generated in Step 1, you need the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Related Information

[SAP Note 2046767](#)

2.5.1 Parameter Table Structure

PAL functions use parameter tables to transfer parameter values. Each PAL function has its own parameter table. To avoid a conflict of table names when several users call PAL functions at the same time, the parameter table must be created as a local temporary column table, so that each parameter table has its own unique scope per session.

The table structure is as follows:

Column Name	Data Type	Description
Name	Varchar	Parameter name
intArgs	Integer	Integer parameter value
doubleArgs	Double	Double parameter value
stringArgs	Varchar	String parameter value

Each row contains only one parameter value, either integer, double or string.

The following table is an example of a parameter table with three parameters. The first parameter, THREAD_NUMBER, is an integer parameter. Thus, in the THREAD_NUMBER row, you should fill the parameter value in the intArgs column, and leave the doubleArgs and stringArgs columns blank.

Name	intArgs	doubleArgs	stringArgs
THREAD_NUMBER	1		
SUPPORT		0.2	

Name	intArgs	doubleArgs	stringArgs
VAR_NAME			hello

2.5.2 Exception Handling

Exceptions thrown by a PAL function can be caught by the exception handler in a SQLScript procedure with AFL error code 423.

The following shows an example of handling the exceptions thrown by an ARIMA function. In the example, you are generating the "DM_PAL".PAL_ARIMAXTRAIN_PROC procedure to call an ARIMA function, whose input data table is empty. You create a SQLScript procedure "DM_PAL".PAL_ARIIMAX_NON_EXCEPTION_PROC which calls the "DM_PAL".PAL_ARIMAXTRAIN_PROC procedure. When you call the "DM_PAL".PAL_ARIIMAX_NON_EXCEPTION_PROC procedure, the exception handler in the procedure will catch the errors thrown by the function.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ARIMAX_DATA_T;
CREATE TYPE PAL_ARIMAX_DATA_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "X1" DOUBLE,
    "Xreg" DOUBLE
);
DROP TYPE PAL_ARIMAX_CONTROL_T;
CREATE TYPE PAL_ARIMAX_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_ARIMAX_MODEL_T;
CREATE TYPE PAL_ARIMAX_MODEL_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" VARCHAR (5000)
);
DROP TABLE PAL_ARIMAX_PDATA_TBL;
CREATE COLUMN TABLE PAL_ARIMAX_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES (1,'DM_PAL', 'PAL_ARIMAX_DATA_T','IN');
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES(2,'DM_PAL', 'PAL_ARIMAX_CONTROL_T','IN');
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES(3,'DM_PAL', 'PAL_ARIMAX_MODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_ARIMAXTRAIN_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'ARIMATRAIN', 'DM_PAL',
'PAL_ARIMAXTRAIN_PROC',PAL_ARIMAX_PDATA_TBL);
DROP PROCEDURE "DM_PAL".PAL_ARIIMAX_NON_EXCEPTION_PROC;
CREATE PROCEDURE PAL_ARIIMAX_NON_EXCEPTION_PROC(IN training_data
PAL_ARIMAX_DATA_T, IN para_args PAL_ARIMAX_CONTROL_T)
LANGUAGE SQLSCRIPT AS
BEGIN
    -- used to catch exceptions
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    SELECT ::SQL_ERROR_CODE, ::SQL_ERROR_MESSAGE FROM DUMMY;
    CALL "DM_PAL".PAL_ARIMAXTRAIN_PROC(:training_data, :para_args, result_model);
END;
DROP TABLE PAL_ARIMAX_DATA_TBL;
CREATE COLUMN TABLE PAL_ARIMAX_DATA_TBL LIKE PAL_ARIMAX_DATA_T;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR
(50), "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('P', 1,null,null);

```

```

INSERT INTO #PAL_CONTROL_TBL VALUES ('Q', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('D', 0,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('METHOD', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('STATIONARY', 1,null,null);
DROP TABLE PAL_ARIMAX_MODEL_TBL;
CREATE COLUMN TABLE PAL_ARIMAX_MODEL_TBL LIKE PAL_ARIMAX_MODEL_T;
CALL PAL_ARIMAX_NON_EXCEPTION_PROC(PAL_ARIMAX_DATA_TBL, "#PAL_CONTROL_TBL");

```

Expected Result

::SQL_ERROR_CODE	::SQL_ERROR_MESSAGE
423	SYS_AFL.AFLPAL:ARIMATRAIN: [423] (range 3) AFL error exception: exception 73001021: PAL error[73001021]:Input table is empty

2.6 Using PAL in SAP HANA AFM

The SAP HANA Application Function Modeler (AFM) in SAP HANA Studio supports functions from PAL in flowgraph models. With the AFM, you can easily add PAL function nodes to your flowgraph, specify its parameters and input/output table types, and generate the procedure, all without writing any SQLScript code. You can also execute the procedure to get the output result of the function, and save the auto-generated SQLScript code for future use.

The main procedure is as follows:

1. Create a new flowgraph or open an existing flowgraph in the *Project Explorer* view.

i Note

For details on how to create a flowgraph, see "Creating a Flowgraph" in *SAP HANA Developer Guide for SAP HANA Studio*

2. Specify the target schema by selecting the flowgraph container and editing *Target Schema* in the *Properties* view.
3. Add the input(s) for the flowgraph by doing the following:
 1. Right-click the input anchor region on the left side of the flowgraph container and choose *Add Input*.
 2. Edit the table types of the input by editing the signature the *Properties* view.

i Note

You can also drag a table from the catalog in the *Systems* view to the input anchor region of the flowgraph container.

4. Add a PAL function to the flowgraph by doing the following:
 1. Drag the function node from the *Predictive Analysis Library* compartment of the *Palette* to the flowgraph editing area.
 2. Specify the input table types of the function by selecting the input anchor and editing its signature in the *Properties* view.
 3. Specify the parameters of the function by selecting the parameter input anchor and editing its signature and fixed content in the *Properties* view.

Note

A parameter table usually has fixed table content. If you want to supply the parameter values later when you execute the procedure, clear the *Fixed Content* option.

4. Specify the output table types of the function by selecting the output anchor and editing the signature in the *Properties* view.
5. (Optional) You can add more PAL nodes to the flowgraph if needed and connect them by holding the *Connect* button  from the source anchor and dragging a connection to the destination anchor.
6. Connect the input(s) in the input anchor region of the flowgraph container to the required input anchor(s) of the PAL function node.
7. For the output tables that you want to see the output result after procedure execution, add them to the output anchor region on the right side of the flowgraph container. To do that, move your mouse cursor over the output anchor of the function node, hold the *Connect* button , and drag a connection to the output anchor region.
8. Save the flowgraph by choosing  *File*  in the HANA Studio main menu.
9. Activate the flowgraph by right-clicking the flowgraph in the *Project Explorer* view and choosing  *Team* .

A new procedure is generated in the target schema which is specified in Step 2.

Note

To activate the flowgraph, the database user _SYS_REPO needs SELECT object privileges for objects that are used as data sources.

10. Select the black downward triangle next to the *Execute* button  in the top right corner of the AFM. A context menu appears. It shows the options *Execute in SQL Editor* and *Open in SQL Editor* as well as the option *Execute and Explore* for every output of the flowgraph. In addition, the context menu shows the option *Edit Input Bindings*.
11. (Optional) If the flowgraph has input tables without fixed content, choose the option *Edit Input Bindings*. A wizard appears that allows you to bind all inputs of the flowgraph to data sources in the catalog.

Note

If you do not bind the inputs, AFM will automatically open this wizard when executing the procedure.

12. Choose one of the options *Execute in SQL Editor*, *Open in SQL Editor*, or *Execute and Explore* for one of the outputs of the flowgraph.
The behavior of the AFM depends on the execution mode.
 - *Open in SQL Editor*: Opens a SQL console containing the SQL code to execute the runtime object.
 - *Execute in SQL Editor*: Opens a SQL console containing the SQL code to execute the runtime object and runs this SQL code.
 - *Execute and Explore*: Executes the runtime object and opens the *Data Explorer* view for the chosen output of the flowgraph.
13. Close the flowgraph by choosing   in the HANA Studio main menu.

For more information on how to use AFM, see the "Transforming Data Using SAP HANA Application Function Modeler" section in *SAP HANA Developer Guide for SAP HANA Studio*.

3 PAL Functions

The following are the available algorithms and functions in the Predictive Analysis Library.

Category	PAL Algorithm	Built-in Function Name
Clustering	Affinity Propagation [page 19]	AP
	Agglomerate Hierarchical Clustering [page 24]	HCAGGLOMERATE
	Anomaly Detection [page 31]	ANOMALYDETECTION
	Cluster Assignment [page 37]	CLUSTERASSIGNMENT
	DBSCAN [page 46]	DBSCAN
	Gaussian Mixture Model (GMM) [page 52]	GMM
	K-Means [page 62]	KMEANS VALIDATEKMEANS
	K-Medians [page 75]	KMEDIANS
	K-Medoids [page 81]	KMEDOIDS
	LDA Estimation and Inference [page 88]	LDAESTIMATE LDAINFERENCE
Classification	Self-Organizing Maps [page 101]	SELFORGMAP
	Slight Silhouette [page 111]	SLIGHTSILHOUETTE
	Area Under Curve (AUC) [page 116]	AUC
	Back Propagation Neural Network [page 122]	CREATEBPNN PREDICTWITHBPNN
	C4.5 Decision Tree [page 135]	CREATEDTWITHC45
	CART Decision Tree [page 142]	CART
	CHAID Decision Tree [page 147]	CREATEDTWITHCHAID
	Confusion Matrix [page 154]	CONFUSIONMATRIX

Category	PAL Algorithm	Built-in Function Name
Classification	KNN [page 158]	KNN
	Logistic Regression (with Elastic Net Regularization) [page 162]	LOGISTICREGRESSION FORECASTWITHLOGISTICR
	Multi-Class Logistic Regression [page 176]	LRMCTR LRMCTE
	Naive Bayes [page 190]	NBCTRAIN NBCPREDICT
	Parameter Selection and Model Evaluation (PSME) [page 198]	PSME
	Predict with Tree Model [page 211]	PREDICTWITHDT
	Random Forest [page 214]	RANDOMFORESTTRAIN RANDOMFORESTSCORING
	Support Vector Machine [page 222]	SVMTRAIN SVMPREDICT
Regression	Bi-Variate Geometric Regression [page 236]	GEOREGRESSION FORECASTWITHGEOR
	Bi-Variate Natural Logarithmic Regression [page 243]	LNREGRESSION FORECASTWITHLNR
	Exponential Regression [page 251]	EXPREGRESSION FORECASTWITHEXPREG
	Multiple Linear Regression [page 259]	LRREGRESSION FORECASTWITHLR
	Polynomial Regression [page 270]	POLYNOMIALREGRESSION FORECASTWITHPOLYNOMIALR
Association	Apriori [page 278]	APRIORIRULE LITEAPRIORIRULE APRIORIRULE2
	FP-Growth [page 297]	FPGROWTH

Category	PAL Algorithm	Built-in Function Name
	K-Optimal Rule Discovery (KORD) [page 309]	KORD
Time Series	ARIMA [page 314]	ARIMATRAIN ARIMAFORECAST ARIMAXFORECAST
	Auto ARIMA [page 332]	AUTOARIMA
	Brown Exponential Smoothing [page 347]	BROWNEXPSMOOTH
	Croston's Method [page 354]	CROSTON
	Forecast Accuracy Measures [page 358]	ACCURACYMEASURES
	Forecast Smoothing [page 362]	FORECASTSMOOTHING
	Linear Regression with Damped Trend and Seasonal Adjust [page 381]	LRWITHSEASONALADJUST
	Single Exponential Smoothing [page 387]	SINGLESMOOTH
	Double Exponential Smoothing [page 393]	DOUBLESMOOTH
	Triple Exponential Smoothing [page 400]	TRIPLESMOOTH
	Seasonality Test [page 409]	SEASONALITYTEST
	Trend Test [page 415]	TRENDTEST
	White Noise Test [page 419]	WHITENOISETEST
Preprocessing	Binning [page 423]	BINNING
	Binning Assignment [page 429]	BINNINGASSIGNMENT
	Convert Category Type to Binary Vector [page 433]	CONV2BINARYVECTOR
	Inter-Quartile Range Test [page 436]	IQRTEST
	Partition [page 440]	PARTITION
	Posterior Scaling [page 444]	POSTERIORSCALING

Category	PAL Algorithm	Built-in Function Name
	Principal Component Analysis (PCA) [page 448]	PCA PCAPROJECTION
	Random Distribution Sampling [page 456]	DISTRRANDOM
	Sampling [page 460]	SAMPLING
	Scaling Range [page 470]	SCALINGRANGE
	Substitute Missing Values [page 475]	SUBSTITUTE_MISSING_VALUES
	Variance Test [page 480]	VARIANCETEST
Statistics	Chi-Squared Test for Goodness of Fit [page 484]	CHISQTESTFIT
	Chi-Squared Test for Independent [page 487]	CHISQTESTIND
	Cumulative Distribution Function [page 491]	DISTRPROB
	Distribution Fitting [page 494]	DISTRFIT DISTRFITCENSORED
	Grubbs' Test [page 504]	GRUBBSTEST
	Kaplan-Meier Survival Analysis [page 509]	KMSURV
	Multivariate Statistics [page 515]	MULTIVARSTAT
	Quantile Function [page 518]	DISTRQUANTILE
	Univariate Statistics [page 522]	UNIVARSTAT
	Variance Equal Test [page 526]	VAREQUALTEST
Social Network Analysis	Link Prediction [page 530]	LINKPREDICTION
Miscellaneous	ABC Analysis [page 534]	ABC
	Weighted Score Table [page 537]	WEIGHTEDTABLE

3.1 Clustering Algorithms

This section describes the clustering algorithms that are provided by the Predictive Analysis Library.

3.1.1 Affinity Propagation

Affinity Propagation (AP) is a relatively new clustering algorithm that has been introduced by Brendan J. Frey and Delbert Dueck. The authors described affinity propagation as follows:

"An algorithm that identifies exemplars among data points and forms clusters of data points around these exemplars. It operates by simultaneously considering all data point as potential exemplars and exchanging messages between data points until a good set of exemplars and clusters emerges."

One of the most significant advantages of AP-cluster is that the number of clusters is not necessarily predetermined.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

AP

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'AP', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<INPUT Seed table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <input seed table>,  
<parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, seed, parameter, and output tables must follow types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer or double	Attribute data	
Seed Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column
	2nd column	Integer	Selected seed ID	A subset of the first column of the Input Table.

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description
DISTANCE_METHOD	Integer	<p>The method to compute the distance between two points.</p> <ul style="list-style-type: none">• 1: Manhattan• 2: Euclidean• 3: Minkowski• 4: Chebyshev• 5: Standardised Euclidean• 6: Cosine
CLUSTER_NUMBER	Integer	<p>Number of clusters.</p> <ul style="list-style-type: none">• 0: does not adjust AP Cluster result.• Non-zero integer: If AP cluster number is bigger than CLUSTER_NUMBER, PAL will merge the result to make the cluster number be the value specified for CLUSTER_NUMBER.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
THREAD_NUMBER	Integer	1	Number of threads.	

Name	Data Type	Default Value	Description	Dependency
MAX_ITERATION	Integer	500	Maximum number of iterations.	
CON_ITERATION	Integer	100	When the clusters keep a steady one for the specified times, the algorithm ends.	
DAMP	Double	0.9	Controls the updating velocity. Value range: $0 < \text{DAMP} < 1$	
PREFERENCE	Double	0.5	Determines the preference. Value range: $0 \leq \text{PREFERENCE} \leq 1$	
SEED_RATIO	Double	1	Select a portion of $(\text{SEED_RATIO} * \text{data_number})$ the input data as seed, where data_number is the row_size of the input data. Value range: $0 < \text{SEED_RATIO} \leq 1$	Only valid when the seed table is empty. If SEED_RATIO is 1, all the input data will be the seed.
TIMES	Integer	1	The sampling times.	Only valid when the seed table is empty and SEED_RATIO is less than 1.
MINKOW_P	Integer	3	The power of the Minkowski method.	Only valid when DISTANCE_METHOD is 3.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer	Cluster ID. The range is from 0 to $\text{CLUSTER_NUMBER}-1$.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and

- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_AP_DATA_T;
CREATE TYPE PAL_AP_DATA_T AS TABLE (
ID INTEGER,
ATTRIB1 DOUBLE,
ATTRIB2 DOUBLE
);
DROP TYPE PAL_AP_SEED_T;
CREATE TYPE PAL_AP_SEED_T AS TABLE (
ID INTEGER,
SEED INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
NAME VARCHAR(50),
INTARGS INTEGER,
DOUBLEARGS DOUBLE,
STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_AP_RESULTS_T;
CREATE TYPE PAL_AP_RESULTS_T AS TABLE(
ID INTEGER,
RESULT INTEGER
);
DROP TABLE PAL_AP_PDATA_TBL;
CREATE COLUMN TABLE PAL_AP_PDATA_TBL (
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_AP_PDATA_TBL VALUES (1,'DM_PAL','PAL_AP_DATA_T', 'IN');
INSERT INTO PAL_AP_PDATA_TBL VALUES (2,'DM_PAL','PAL_AP_SEED_T', 'IN');
INSERT INTO PAL_AP_PDATA_TBL VALUES (3,'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_AP_PDATA_TBL VALUES (4,'DM_PAL','PAL_AP_RESULTS_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_AP');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'AP', 'DM_PAL', 'PAL_AP',
PAL_AP_PDATA_TBL);
DROP TABLE PAL_AP_DATA_TBL;
CREATE COLUMN TABLE PAL_AP_DATA_TBL LIKE PAL_AP_DATA_T;
INSERT INTO PAL_AP_DATA_TBL VALUES(1,0.10,0.10);
INSERT INTO PAL_AP_DATA_TBL VALUES(2,0.11,0.10);
INSERT INTO PAL_AP_DATA_TBL VALUES(3,0.10,0.11);
INSERT INTO PAL_AP_DATA_TBL VALUES(4,0.11,0.11);
INSERT INTO PAL_AP_DATA_TBL VALUES(5,0.12,0.11);
INSERT INTO PAL_AP_DATA_TBL VALUES(6,0.11,0.12);
INSERT INTO PAL_AP_DATA_TBL VALUES(7,0.12,0.12);
INSERT INTO PAL_AP_DATA_TBL VALUES(8,0.12,0.13);
INSERT INTO PAL_AP_DATA_TBL VALUES(9,0.13,0.12);
INSERT INTO PAL_AP_DATA_TBL VALUES(10,0.13,0.13);
INSERT INTO PAL_AP_DATA_TBL VALUES(11,0.13,0.14);
INSERT INTO PAL_AP_DATA_TBL VALUES(12,0.14,0.13);
INSERT INTO PAL_AP_DATA_TBL VALUES(13,10.10,10.10);
INSERT INTO PAL_AP_DATA_TBL VALUES(14,10.11,10.10);
INSERT INTO PAL_AP_DATA_TBL VALUES(15,10.10,10.11);
INSERT INTO PAL_AP_DATA_TBL VALUES(16,10.11,10.11);
INSERT INTO PAL_AP_DATA_TBL VALUES(17,10.11,10.12);
INSERT INTO PAL_AP_DATA_TBL VALUES(18,10.12,10.11);
INSERT INTO PAL_AP_DATA_TBL VALUES(19,10.12,10.12);
INSERT INTO PAL_AP_DATA_TBL VALUES(20,10.12,10.13);
INSERT INTO PAL_AP_DATA_TBL VALUES(21,10.13,10.12);
INSERT INTO PAL_AP_DATA_TBL VALUES(22,10.13,10.13);
INSERT INTO PAL_AP_DATA_TBL VALUES(23,10.13,10.14);
INSERT INTO PAL_AP_DATA_TBL VALUES(24,10.14,10.13);

```

```

DROP TABLE PAL_AP_SEED_TBL;
CREATE COLUMN TABLE PAL_AP_SEED_TBL LIKE PAL_AP_SEED_T;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('MAX_ITERATION',500,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('CON_ITERATION',100,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('DAMP',null,0.9,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('PREFERENCE',null,0.5,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('DISTANCE_METHOD',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('CLUSTER_NUMBER',0,null,null);
DROP TABLE PAL_AP_RESULTS_TBL;
CREATE COLUMN TABLE PAL_AP_RESULTS_TBL LIKE PAL_AP_RESULTS_T;
CALL DM_PAL.PAL_AP(PAL_AP_DATA_TBL, PAL_AP_SEED_TBL, #PAL_CONTROL_TBL,
PAL_AP_RESULTS_TBL) with OVERVIEW;
SELECT * FROM PAL_AP_RESULTS_TBL;

```

Expected Result

	ID	RESULT
1	1	0
2	2	0
3	3	0
4	4	0
5	5	0
6	6	0
7	7	0
8	8	0
9	9	0
10	10	0
11	11	0
12	12	0
13	13	1
14	14	1
15	15	1
16	16	1
17	17	1
18	18	1
19	19	1
20	20	1
21	21	1
22	22	1
23	23	1
24	24	1

3.1.2 Agglomerate Hierarchical Clustering

Hierarchical clustering is a widely used clustering method which can find natural groups within a set of data. The idea is to group the data into a hierarchy or a binary tree of the subgroups. A hierarchical clustering can be either agglomerate or divisive, depending on the method of hierarchical decomposition.

The implementation in PAL follows the agglomerate approach, which merges the clusters with a bottom-up strategy. Initially, each data point is considered as an own cluster. The algorithm iteratively merges two clusters based on the dissimilarity measure in a greedy manner and forms a larger cluster. Therefore, the input data must be numeric and a measure of dissimilarity between sets of data is required, which is achieved by using the following two parameters:

- An appropriate metric (a measure of distance between pairs of groups)
- A linkage criterion which specifies the distances between groups

An advantage of hierarchical clustering is that it does not require the number of clusters to be specified as the input. And the hierarchical structure can also be used for data summarization and visualization.

The agglomerate hierarchical clustering functions in PAL now supports eight kinds of appropriate metrics and seven kinds of linkage criteria.

Support for Category Attributes

If the input data has category attributes, you must set the DISTANCE_FUNC parameter to Gower Distance to support calculating the distance matrix. Gower Distance is calculated in the following way.

Suppose that the items x_i and x_j have K attributes, the distance between x_i and x_j is:

$$s(i, j) = \sum_k (s_{ijk} * w_k) / \sum_k w_k$$

For continuous attributes,

$$s_{ijk} = (x_{ik} - x_{jk}) / R_k$$

$$w_k = 1$$

R_k is the range of values for the k^{th} variable; w_k is set by user and the default is 1.

For category attributes,

If $x_{ik} = x_{jk}$: $s_{ijk} = 0$

Other cases: $s_{ijk} = 1$

Prerequisites

- The first column of the input data is an ID column and the other columns are of integer, double, varchar, or nvarchar data type.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

HCAGGLOMERATE

This is a clustering function using the agglomerate hierarchical clustering algorithm.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'HCAGGLOMERATE',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<CombineProcess OUTPUT table type>	OUT
4	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <combine process output table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, combine process, and result tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Varchar, nvarchar, or integer	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar	Attribute data	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
CLUSTER_NUM	Integer	1	<p>Number of clusters after agglomerative hierarchical clustering algorithm.</p> <p>Value range: between 1 and the initial number of input data</p>	
DISTANCE_FUNC	Integer	8	<p>Measure of distance between two clusters.</p> <ul style="list-style-type: none"> • 1: Manhattan Distance • 2: Euclidean Distance • 3: Minkowski Distance • 4: Chebyshev Distance • 6: Cosine • 7: Pearson Correlation • 8: Squared Euclidean Distance • 9: Jaccard Distance • 10: Gower Distance <p>Note 1: For Jaccard Distance, non-zero input data will be treated as 1, and zero input data will be treated as 0.</p> <p>Jaccard Distance = $(M_{01} + M_{10}) / (M_{11} + M_{01} + M_{10})$</p> <p>Note 2: Only Gower Distance supports category attributes.</p>	When CLUSTER_METHOD is 5, 6, or 7, this parameter must be set to 8.

Name	Data Type	Default Value	Description	Dependency
CLUSTER_METHOD	Integer	5	<p>Linkage type between two clusters.</p> <ul style="list-style-type: none"> • 1: Nearest Neighbor (single linkage) • 2: Furthest Neighbor (complete linkage) • 3: Group Average (UPGMA) • 4: Weighted Average (WPGMA) • 5: Centroid Clustering • 6: Median Clustering • 7: Ward Method <p>Note: For clustering method 5, 6, or 7, the corresponding distance function (DISTANCE_FUNC) must be set to 8.</p>	
THREAD_NUM	Integer	1	<p>Number of threads.</p> <p>Value range: between 1 and 512</p>	
DISTANCE_DIMENSION	Double	3	<p>Distance dimension can be set if DISTANCE_FUNC is set to 3 (Minkowski Distance). The value should be no less than 1.</p>	Only valid when DISTANCE_FUNC is 3.
NORMALIZE_TYPE	Integer	0	<p>Normalization type:</p> <ul style="list-style-type: none"> • 0: does nothing • 1: Z score standardize • 2: transforms to new range: -1 to 1 • 3: transforms to new range: 0 to 1 	

Name	Data Type	Default Value	Description	Dependency
CATEGORY_COL	Integer	No default value	Indicates whether the column (column index starts from zero) is category variable. By default, 'varchar' or 'nvarchar' is category variable and 'integer' or 'double' is continuous variable. Note: Only integer, varchar, or nvarchar columns can be set to be categorical column.	
CATEGORY_WEIGHTS	Double	1	Represents the weight of category columns. Default: 1	

Output Tables

Table	Column	Column Data Type	Description	Constraint
Combine Process	1st column	Integer	Cluster stage.	
	2nd column	Varchar, nvarchar, or integer	One of the clusters that is to be combined in one combine stage, name as its row number in the input data table.	After the combining, the new cluster will be named as the smaller one. The type must be the same as the ID type in the input table.
	3rd column	Varchar, nvarchar, or integer	The other cluster to be combined in the same combine stage, named as its row number in the input data table.	The type must be the same as the ID type in the input table.
	4th column	Double	Distance between the two combined clusters.	
Result	1st column	Varchar, nvarchar, or integer	ID of the input data.	The type must be the same as the type of ID in the input table.
	2nd column	Integer	Cluster number after applying the hierarchical agglomerate algorithm.	

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE DATA_T;
CREATE TYPE DATA_T AS TABLE(
  "ID" VARCHAR(100),
  "X1" DOUBLE,
  "X2" DOUBLE,
  "X3" VARCHAR(50)
);

DROP TYPE PARAMETERS_T;
CREATE TYPE PARAMETERS_T AS TABLE(
  "NAME" VARCHAR (50),
  "INT_ARGS" INTEGER,
  "DOUBLE_ARGS" DOUBLE,
  "STRING_ARGS" VARCHAR (100)
);

DROP TYPE COMBINEPROCESS_T;
CREATE TYPE COMBINEPROCESS_T AS TABLE(
  "STAGE" INT,
  "CLUSTER_A" VARCHAR(100),
  "CLUSTER_B" VARCHAR(100),
  "MINDISTANCE" DOUBLE
);
DROP TYPE RESULT_T;
CREATE TYPE RESULT_T AS TABLE(
  "ID" VARCHAR(100),
  "CLUSTER" INT
);
DROP table PDATA_TBL;
CREATE column table PDATA_TBL("POSITION" INT,"SCHEMA_NAME"
NVARCHAR(256),"TYPE_NAME" NVARCHAR (256),"PARAMETER_TYPE" VARCHAR (7));
insert into PDATA_TBL values (1,'DM_PAL','DATA_T','IN');
insert into PDATA_TBL values (2,'DM_PAL','PARAMETERS_T','IN');
insert into PDATA_TBL values (3,'DM_PAL','COMBINEPROCESS_T','OUT');
insert into PDATA_TBL values (4,'DM_PAL','RESULT_T','OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_HCAGGLOMERATE');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','HCAGGLOMERATE','DM_PAL','PAL_HCAGGLOMERATE',PDATA_TBL);
DROP TABLE DATA_TBL;
CREATE COLUMN TABLE DATA_TBL like DATA_T;
INSERT INTO DATA_TBL VALUES ('0' , 0.5, 0.5, 'A');
INSERT INTO DATA_TBL VALUES ('1' , 1.5, 0.5, 'B');
INSERT INTO DATA_TBL VALUES ('2' , 1.5, 1.5, 'B');
INSERT INTO DATA_TBL VALUES ('3' , 0.5, 1.5, 'B');
INSERT INTO DATA_TBL VALUES ('4' , 1.1, 1.2, 'B');
INSERT INTO DATA_TBL VALUES ('5' , 0.5, 15.5, 'B');
INSERT INTO DATA_TBL VALUES ('6' , 1.5, 15.5, 'C');
INSERT INTO DATA_TBL VALUES ('7' , 1.5, 16.5, 'C');
INSERT INTO DATA_TBL VALUES ('8' , 0.5, 16.5, 'C');
INSERT INTO DATA_TBL VALUES ('9' , 1.2, 16.1, 'C');
INSERT INTO DATA_TBL VALUES ('10', 15.5, 15.5, 'C');
INSERT INTO DATA_TBL VALUES ('11', 16.5, 15.5, 'D');
INSERT INTO DATA_TBL VALUES ('12', 16.5, 16.5, 'D');
INSERT INTO DATA_TBL VALUES ('13', 15.5, 16.5, 'D');
INSERT INTO DATA_TBL VALUES ('14', 15.6, 16.2, 'D');
INSERT INTO DATA_TBL VALUES ('15', 15.5, 0.5, 'D');
INSERT INTO DATA_TBL VALUES ('16', 16.5, 0.5, 'A');
INSERT INTO DATA_TBL VALUES ('17', 16.5, 1.5, 'A');
INSERT INTO DATA_TBL VALUES ('18', 15.5, 1.5, 'A');

```

```

INSERT INTO DATA_TBL VALUES ('19', 15.7, 1.6, 'A');
DROP TABLE PARAMETERS_TBL;
CREATE COLUMN TABLE PARAMETERS_TBL like PARAMETERS_T;
INSERT INTO PARAMETERS_TBL VALUES ('THREAD_NUM',8,null,null);
INSERT INTO PARAMETERS_TBL VALUES ('CLUSTER_NUM',4,null,null);
INSERT INTO PARAMETERS_TBL VALUES ('CLUSTER_METHOD',4,null,null);
INSERT INTO PARAMETERS_TBL VALUES ('DISTANCE_FUNC',10,null,null);
INSERT INTO PARAMETERS_TBL VALUES ('DISTANCE_DIMENSION',null,3,null);
INSERT INTO PARAMETERS_TBL VALUES ('NORMALIZE_TYPE',0,null,null);
INSERT INTO PARAMETERS_TBL VALUES ('CATEGORY_WEIGHTS',null,0.1,null);
INSERT INTO PARAMETERS_TBL VALUES ('CATEGORY_COL',3,null,null);
DROP TABLE COMBINEPROCESS_TBL;
CREATE COLUMN TABLE COMBINEPROCESS_TBL like COMBINEPROCESS_T;
DROP TABLE RESULT_TBL;
CREATE COLUMN TABLE RESULT_TBL like RESULT_T;
CALL DM_PAL.PAL_HCAGGLOMERATE(DATA_TBL, PARAMETERS_TBL, COMBINEPROCESS_TBL,
RESULT_TBL) with overview;
select * from COMBINEPROCESS_TBL;
select * from RESULT_TBL;

```

Expected Result

COMBINEPROCESS_TBL:

	STAGE	CLUSTER_A	CLUSTER_B	MINDISTANCE
1	1	18	19	0.0187
2	2	13	14	0.025
3	3	7	9	0.0437
4	4	2	4	0.0438
5	5	2	3	0.0594
6	6	17	18	0.0594
7	7	6	7	0.0594
8	8	11	12	0.0625
9	9	11	13	0.0906
10	10	16	17	0.0922
11	11	6	8	0.0953
12	12	1	2	0.0953
13	13	0	1	0.1727
14	14	5	6	0.1727
15	15	10	11	0.175
16	16	15	16	0.1805
17	17	0	15	1.0381
18	18	5	10	1.0425
19	19	0	5	1.5146

RESULT_TBL:

ID	CLUSTER
0	1
1	1
2	1
3	1
4	1
5	2
6	2
7	2
8	2
9	2
10	3
11	3
12	3
13	3
14	3
15	4
16	4
17	4
18	4
19	4

3.1.3 Anomaly Detection

Anomaly detection is used to find the existing data objects that do not comply with the general behavior or model of the data. Such data objects, which are grossly different from or inconsistent with the remaining set of data, are called anomalies or outliers. Sometimes anomalies are also referred to as discordant observations, exceptions, aberrations, surprises, peculiarities or contaminants in different application domains.

Anomalies in data can translate to significant (and often critical) actionable information in a wide variety of application domains. For example, an anomalous traffic pattern in a computer network could mean that a hacked computer is sending out sensitive data to an unauthorized destination. An anomalous MRI image may indicate presence of malignant tumors. Anomalies in credit card transaction data could indicate credit card or identity theft or anomalous readings from a space craft sensor could signify a fault in some component of the space craft.

PAL uses k-means to realize anomaly detection in two steps:

1. Use k-means to group the origin data into k clusters.
2. Identify some points that are far from all cluster centers as anomalies.

Prerequisites

- The input data contains an ID column and the other columns are of integer or double data type.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

ANOMALYDETECTION

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'ANOMALYDETECTION',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Outliers table type>	OUT
4	<schema_name>	<Statistics table type>	OUT (optional)
5	<schema_name>	<Centers table type>	OUT (optional)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input_table>, <parameter_table>,
<outliers_output_table>, <statistics_output_table>, <centers_output_table>) with
overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	It must be the first column.
	Other columns	Integer or double	Attribute data	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
GROUP_NUMBER	Integer	No default value	Number of groups (k). If k is not specified, the G-means method will be used to determine the number of clusters.
DISTANCE_LEVEL	Integer	2	Computes the distance between the item and the cluster center. <ul style="list-style-type: none">• 1: Manhattan distance• 2: Euclidean distance• 3: Minkowski distance
OUTLIER_PERCENTAGE	Double	0.1	Indicates the proportion of anomalies in the source data.
OUTLIER_DEFINE	Integer	2	Specifies which point should be defined as outlier: <ul style="list-style-type: none">• 1: Max distance between the point and the center it belongs to• 2: Max sum distance from the point to all centers
MAX_ITERATION	Integer	100	Maximum number of iterations.
INIT_TYPE	Integer	4	Controls how the initial centers are selected: <ul style="list-style-type: none">• 1: First k observations• 2: Random with replacement• 3: Random without replacement• 4: Patent of selecting the init center (US 6,882,998 B1)

Name	Data Type	Default Value	Description
NORMALIZATION	Integer	0	<p>Normalization type:</p> <ul style="list-style-type: none"> • 0: No • 1: Yes. For each point $X(x_1, x_2, \dots, x_n)$, the normalized value will be $X'(x_1/S, x_2/S, \dots, x_n/S)$, where $S = x_1 + x_2 + \dots + x_n$. • 2: For each column C, get the min and max value of C, and then $C[i] = (C[i] - \text{min}) / (\text{max} - \text{min})$.
THREAD_NUMBER	Integer	1	Number of threads. The default is 1.
EXIT_THRESHOLD	Double	0.000000001	Threshold (actual value) for exiting the iterations.

Output Tables

Table	Column	Column Data Type	Description
Outliers	1st column	Integer, bigint, varchar, or nvarchar	ID
	Other columns	Integer or double	Coordinates of outliers
Statistics	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer	ID of cluster center
	3rd column	Double	distance value
Centers	1st column	Integer	ID of cluster center
	Other columns	Integer or double	Coordinates of centers

i Note

The statistics and centers output table were introduced in SAP HANA SPS 09. The version with only the outliers output table is also supported.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
```

```

DROP TYPE PAL_AD_DATA_T;
CREATE TYPE PAL_AD_DATA_T AS TABLE(
    "ID" INTEGER,
    "V000" DOUBLE,
    "V001" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_AD_RESULT_T;
CREATE TYPE PAL_AD_RESULT_T AS TABLE(
    "ID" INTEGER,
    "V000" DOUBLE,
    "V001" DOUBLE
);
DROP TYPE PAL_AD_STATISTIC_T;
CREATE TYPE PAL_AD_STATISTIC_T AS TABLE(
    "ID" INTEGER,
    "CLUSTER_ID" INTEGER,
    "SCORE" DOUBLE
);
DROP TYPE PAL_AD_CENTERS_T;
CREATE TYPE PAL_AD_CENTERS_T AS TABLE(
    "CLUSTER_ID" INTEGER,
    "V000" DOUBLE,
    "V001" DOUBLE
);
DROP TABLE PAL_AD_PDATA_TBL;
CREATE COLUMN TABLE PAL_AD_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_AD_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_AD_DATA_T', 'IN');
INSERT INTO PAL_AD_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_AD_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_AD_RESULT_T', 'OUT');
INSERT INTO PAL_AD_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_AD_STATISTIC_T', 'OUT');
INSERT INTO PAL_AD_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_AD_CENTERS_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_ANOMALY_DETECTION_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'ANOMALYDETECTION',
'DM_PAL', 'PAL_ANOMALY_DETECTION_PROC', PAL_AD_PDATA_TBL);
DROP TABLE PAL_AD_DATA_TBL;
CREATE COLUMN TABLE PAL_AD_DATA_TBL LIKE PAL_AD_DATA_T;
INSERT INTO PAL_AD_DATA_TBL VALUES (0, 0.5, 0.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (1, 1.5, 0.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (2, 1.5, 1.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (3, 0.5, 1.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (4, 1.1, 1.2);
INSERT INTO PAL_AD_DATA_TBL VALUES (5, 0.5, 15.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (6, 1.5, 15.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (7, 1.5, 16.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (8, 0.5, 16.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (9, 1.2, 16.1);
INSERT INTO PAL_AD_DATA_TBL VALUES (10, 15.5, 15.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (11, 16.5, 15.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (12, 16.5, 16.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (13, 15.5, 16.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (14, 15.6, 16.2);
INSERT INTO PAL_AD_DATA_TBL VALUES (15, 15.5, 0.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (16, 16.5, 0.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (17, 16.5, 1.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (18, 15.5, 1.5);
INSERT INTO PAL_AD_DATA_TBL VALUES (19, 15.7, 1.6);

```

```

INSERT INTO PAL_AD_DATA_TBL VALUES (20, -1.0, -1.0);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('GROUP_NUMBER',4,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INIT_TYPE',4,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('DISTANCE_LEVEL',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION',100,null,null);
DROP TABLE PAL_AD_RESULT_TBL;
CREATE COLUMN TABLE PAL_AD_RESULT_TBL LIKE PAL_AD_RESULT_T;
DROP TABLE PAL_AD_STATISTIC_TBL;
CREATE COLUMN TABLE PAL_AD_STATISTIC_TBL LIKE PAL_AD_STATISTIC_T;
DROP TABLE PAL_AD_CENTERS_TBL;
CREATE COLUMN TABLE PAL_AD_CENTERS_TBL LIKE PAL_AD_CENTERS_T;
CALL DM_PAL.PAL_ANOMALY_DETECTION_PROC(PAL_AD_DATA_TBL, #PAL_CONTROL_TBL,
PAL_AD_RESULT_TBL, PAL_AD_STATISTIC_TBL, PAL_AD_CENTERS_TBL) with OVERVIEW;
SELECT * FROM PAL_AD_RESULT_TBL;
SELECT * FROM PAL_AD_STATISTIC_TBL;
SELECT * FROM PAL_AD_CENTERS_TBL;

```

Expected Result

PAL_AD_RESULT_TBL:

ID	V000	V001
20	-1	-1
16	16.5	0.5

PAL_AD_STATISTIC_TBL:

ID	CLUSTER_ID	SCORE
20	0	60.619864431293074
16	2	54.110424127543936

PAL_AD_CENTERS_TBL:

CLUSTER_ID	V000	V001
0	0.683333333333332	0.7000000000000001
1	15.919999999999998	16.04
2	15.940000000000001	1.1199999999999999
3	1.04	16.02

3.1.4 Cluster Assignment

Cluster assignment is used to assign data to the clusters that were previously generated by some clustering methods such as K-means, DBSCAN (Density-Based Spatial Clustering of Applications with Noise), and SOM (Self-Organizing Maps).

This algorithm requires that the corresponding clustering procedures save cluster information, or cluster model, which also includes the control parameters for consistency. It assumes that new data is from similar distribution as previous data, and will not update the cluster information.

For clusters generated by K-means, distances between new data and cluster centers are calculated, and then the new data is assigned to the cluster with the smallest distance.

For clusters generated by DBSCAN, all core objects are stored. For each piece of new data, the algorithm tries to find a core object in some formed cluster whose distance is less than the value of the RADIUS parameter. If such a core object is found, the new data is then assigned to the corresponding cluster, otherwise it is assigned to cluster -1, indicating that it is noise. It is possible that a piece of data can belong to more than one cluster, which can be further divided into the following two cases:

- If the number of core objects whose distances to the new data is less than the MINPTS parameter value, meaning that the new data is a border object, the new data is assigned to the cluster where there is a core object having the smallest distance to the new data.
- If the number of core objects whose distances to the new data is not less than MINPTS, which means the new data is also a core object, it is then assigned to cluster -2, indicating that it belongs to more than one cluster. In this case, re-running the DBSCAN function is highly suggested.

For clusters generated by SOM, similar to K-means, the distances between new data and weight vector are calculated, and the new data is then assigned to the cluster with the smallest distance.

Prerequisites

- No missing or null data in the inputs.
- Data types must be identical to those in the corresponding clustering procedure.
- The data types of the ID columns in the data input table and the result output table must be identical.

CLUSTERASSIGNMENT

This function directly assigns data to clusters based on the previous cluster model, without running clustering procedure thoroughly. It currently supports the K-means, DBSCAN, and SOM clustering methods.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'CLUSTERASSIGNMENT',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Cluster Model INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <cluster model input table>, <parameter table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar Note: For SOM clustering method, attribute data only supports numeric values,hence its column data type can only be integer or double.	Attribute data	
Cluster Model	1st column	Integer	Cluster model ID	This must be the first column.
	2nd column	CLOB, varchar, or nvarchar	Cluster model saved as JSON string	The table must be a column table. The minimum length of each unit (row) is 5000.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

None.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer	The assigned cluster number
	3rd column	Double	Distance between a given point and the cluster center (K-means), nearest core object (DBSCAN), or weight vector (SOM).

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

For K-means:

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_KMEANS_DATA_T;
CREATE TYPE PAL_KMEANS_DATA_T AS TABLE(
  "ID" INTEGER,
  "V000" DOUBLE,
  "V001" VARCHAR(2),
  "V002" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
  "NAME" VARCHAR(100),
  "INTARGS" INTEGER,
  "DOUBLEARGS" DOUBLE,
  "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_KMEANS_ASSIGNED_T;
CREATE TYPE PAL_KMEANS_ASSIGNED_T AS TABLE(
  "ID" INTEGER,
  "CLUSTER" INTEGER,
  "DISTANCE" DOUBLE,
  "SLIGHT_SILHOUETTE" DOUBLE
);
DROP TYPE PAL_KMEANS_CENTERS_T;
CREATE TYPE PAL_KMEANS_CENTERS_T AS TABLE(
  "CLUSTER_ID" INTEGER,
  "V000" DOUBLE,
```

```

"V001" VARCHAR(2),
"V002" DOUBLE
);
DROP TYPE PAL_KMEANS_SIL_CENTERS_T;
CREATE TYPE PAL_KMEANS_SIL_CENTERS_T AS TABLE(
"CLUSTER_ID" INTEGER,
"SLIGHT_SILOUETTE" DOUBLE
);
DROP TYPE PAL_KMEANS_STATISTIC_T;
CREATE TYPE PAL_KMEANS_STATISTIC_T AS TABLE(
"NAME" VARCHAR(50),
"VALUE" DOUBLE
);
DROP TYPE PAL_KMEANS_MODEL_T;
CREATE TYPE PAL_KMEANS_MODEL_T AS TABLE(
"JID" INTEGER,
"JSMODEL" VARCHAR(5000)
);
DROP TABLE PAL_KMEANS_PDATA_TBL;
CREATE COLUMN TABLE PAL_KMEANS_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_KMEANS_DATA_T', 'IN');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_KMEANS_ASSIGNED_T',
'OUT');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_KMEANS_CENTERS_T',
'OUT');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (5, 'DM_PAL',
'PAL_KMEANS_SIL_CENTERS_T', 'OUT');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (6, 'DM_PAL', 'PAL_KMEANS_STATISTIC_T',
'OUT');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (7, 'DM_PAL', 'PAL_KMEANS_MODEL_T',
'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_KMEANS_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'KMEANS', 'DM_PAL',
'PAL_KMEANS_PROC', PAL_KMEANS_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
"NAME" VARCHAR (100),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('GROUP_NUMBER', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INIT_TYPE', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('DISTANCE_LEVEL', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 100, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXIT_THRESHOLD', null, 1.0E-6, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_WEIGHTS', null, 0.5, null);
DROP TABLE PAL_KMEANS_DATA_TBL;
CREATE COLUMN TABLE PAL_KMEANS_DATA_TBL LIKE PAL_KMEANS_DATA_T;
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (0, 0.5, 'A', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (1, 1.5, 'A', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (2, 1.5, 'A', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (3, 0.5, 'A', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (4, 1.1, 'B', 1.2);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (5, 0.5, 'B', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (6, 1.5, 'B', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (7, 1.5, 'B', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (8, 0.5, 'B', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (9, 1.2, 'C', 16.1);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (10, 15.5, 'C', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (11, 16.5, 'C', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (12, 16.5, 'C', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (13, 15.5, 'C', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (14, 15.6, 'D', 16.2);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (15, 15.5, 'D', 0.5);

```

```

INSERT INTO PAL_KMEANS_DATA_TBL VALUES (16, 16.5, 'D', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (17, 16.5, 'D', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (18, 15.5, 'D', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (19, 15.7, 'A', 1.6);
DROP TABLE PAL_KMEANS_ASSIGNED_TBL;
CREATE COLUMN TABLE PAL_KMEANS_ASSIGNED_TBL LIKE PAL_KMEANS_ASSIGNED_T;
DROP TABLE PAL_KMEANS_CENTERS_TBL;
CREATE COLUMN TABLE PAL_KMEANS_CENTERS_TBL LIKE PAL_KMEANS_CENTERS_T;
DROP TABLE PAL_KMEANS_SIL_CENTERS_TBL;
CREATE COLUMN TABLE PAL_KMEANS_SIL_CENTERS_TBL LIKE PAL_KMEANS_SIL_CENTERS_T;
DROP TABLE PAL_KMEANS_STATISTIC_TBL;
CREATE COLUMN TABLE PAL_KMEANS_STATISTIC_TBL LIKE PAL_KMEANS_STATISTIC_T;
DROP TABLE PAL_KMEANS_MODEL_TBL;
CREATE COLUMN TABLE PAL_KMEANS_MODEL_TBL LIKE PAL_KMEANS_MODEL_T;
CALL "DM_PAL".PAL_KMEANS_PROC(PAL_KMEANS_DATA_TBL, #PAL_CONTROL_TBL,
PAL_KMEANS_ASSIGNED_TBL, PAL_KMEANS_CENTERS_TBL, PAL_KMEANS_SIL_CENTERS_TBL,
PAL_KMEANS_STATISTIC_TBL, PAL_KMEANS_MODEL_TBL) with OVERVIEW;
----- NEXT CLUSTER ASSIGNMENT -----
DROP TYPE PAL_CLUSTER_ASSIGNED_T;
CREATE TYPE PAL_CLUSTER_ASSIGNED_T AS TABLE(
"ID" INTEGER,
"CLUSTER" INTEGER,
"DISTANCE" DOUBLE
);
DROP TABLE PAL_CLUSTERASSIGNMENT_PDATA_TBL;
CREATE COLUMN TABLE PAL_CLUSTERASSIGNMENT_PDATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_KMEANS_DATA_T', 'IN');
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_KMEANS_MODEL_T', 'IN');
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_CLUSTER_ASSIGNED_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL',
'PAL_CLUSTERASSIGNMENT PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'CLUSTERASSIGNMENT',
'DM_PAL', 'PAL_CLUSTERASSIGNMENT PROC', PAL_CLUSTERASSIGNMENT_PDATA_TBL);
DROP TABLE PAL_KMEANS_DATA_TBL;
CREATE COLUMN TABLE PAL_KMEANS_DATA_TBL LIKE PAL_KMEANS_DATA_T;
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (0, 0.5, 'A', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (1, 1.5, 'A', 7.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (2, 1.5, 'B', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (3, 10.5, 'F', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (4, 2.1, 'B', 1.2);
DROP TABLE PAL_CLUSTER_ASSIGNED_TBL;
CREATE COLUMN TABLE PAL_CLUSTER_ASSIGNED_TBL LIKE PAL_CLUSTER_ASSIGNED_T;
CALL "DM_PAL".PAL_CLUSTERASSIGNMENT PROC(PAL_KMEANS_DATA_TBL,
PAL_KMEANS_MODEL_TBL, #PAL_CONTROL_TBL, PAL_CLUSTER_ASSIGNED_TBL) with OVERVIEW;
SELECT * FROM PAL_CLUSTER_ASSIGNED_TBL;

```

Expected Result

PAL_CLUSTER_ASSIGNED_TBL:

	ID	CLUSTER	DISTANCE
1	0	0	0.8910879487969621
2	1	0	6.619229627558714
3	2	3	0.8356835545457004
4	3	1	6.101329978011609
5	4	0	1.6574729500657326

For DBSCAN:

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DBSCAN_DATA_T;
CREATE TYPE PAL_DBSCAN_DATA_T AS TABLE ( ID integer, ATTRIB1 double, ATTRIB2
double, ATTRIB3 VARCHAR(10));
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE( NAME varchar(50), INTARGS integer,
DOUBLEARGS double, STRINGARGS varchar(100));
DROP TYPE PAL_DBSCAN_RESULTS_T;
CREATE TYPE PAL_DBSCAN_RESULTS_T AS TABLE( ID integer, RESULT integer);
DROP TYPE PAL_DBSCAN_MODEL_T;
CREATE TYPE PAL_DBSCAN_MODEL_T AS TABLE( JID integer, JSMODEL VARCHAR(5000));
DROP TABLE PAL_DBSCAN_FDATA_TBL;
CREATE COLUMN TABLE PAL_DBSCAN_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_DBSCAN_PDATA_TBL VALUES (1, 'DM_PAL','PAL_DBSCAN_DATA_T', 'IN');
INSERT INTO PAL_DBSCAN_PDATA_TBL VALUES (2, 'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_DBSCAN_PDATA_TBL VALUES (3,
'DM_PAL','PAL_DBSCAN_RESULTS_T', 'OUT');
INSERT INTO PAL_DBSCAN_PDATA_TBL VALUES (4, 'DM_PAL','PAL_DBSCAN_MODEL_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DBSCAN');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','DBSCAN',
'DM_PAL','PAL_DBSCAN', PAL_DBSCAN_PDATA_TBL);
DROP TABLE PAL_DBSCAN_DATA_TBL;
CREATE COLUMN TABLE PAL_DBSCAN_DATA_TBL ( ID integer, ATTRIB1 double, ATTRIB2
double, ATTRIB3 varchar(10));
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(1,0.10,0.10, 'B');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(2,0.11,0.10, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(3,0.10,0.11, 'C');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(4,0.11,0.11, 'B');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(5,0.12,0.11, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(6,0.11,0.12, 'E');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(7,0.12,0.12, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(8,0.12,0.13, 'C');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(9,0.13,0.12, 'D');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(10,0.13,0.13, 'D');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(11,0.13,0.14, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(12,0.14,0.13, 'C');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(13,10.10,10.10, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(14,10.11,10.10, 'F');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(15,10.10,10.11, 'E');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(16,10.11,10.11, 'E');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(17,10.11,10.12, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(18,10.12,10.11, 'B');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(19,10.12,10.12, 'B');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(20,10.12,10.13, 'D');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(21,10.13,10.12, 'F');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(22,10.13,10.13, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(23,10.13,10.14, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(24,10.14,10.13, 'D');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(25,4.10,4.10, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(26,7.11,7.10, 'C');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(27,-3.10,-3.11, 'C');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(28,16.11,16.11, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(29,20.11,20.12, 'C');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(30,15.12,15.11, 'A');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL( NAME varchar(50), INTARGS
integer, DOUBLEARGS double, STRINGARGS varchar(100));
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('AUTO_PARAM',null,null,'true');
INSERT INTO #PAL_CONTROL_TBL VALUES('DISTANCE_METHOD',1,null,null);
DROP TABLE PAL_DBSCAN_RESULTS_TBL;
CREATE COLUMN TABLE PAL_DBSCAN_RESULTS_TBL( ID integer, RESULT integer);
DROP TABLE PAL_DBSCAN_MODEL_TBL;
CREATE COLUMN TABLE PAL_DBSCAN_MODEL_TBL like PAL_DBSCAN_MODEL_T;

```

```

CALL DM_PAL.PAL_DBSCAN(PAL_DBSCAN_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_DBSCAN_RESULTS_TBL, PAL_DBSCAN_MODEL_TBL) with overview;
----- NEXT CLUSTER ASSIGNMENT -----
DROP TYPE PAL_CLUSTER_ASSIGNED_T;
CREATE TYPE PAL_CLUSTER_ASSIGNED_T AS TABLE(
"ID" INTEGER,
"CLUSTER" INTEGER,
"DISTANCE" DOUBLE
);
DROP TABLE PAL_CLUSTERASSIGNMENT_PDATA_TBL;
CREATE COLUMN TABLE PAL_CLUSTERASSIGNMENT_PDATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_DBSCAN_DATA_T', 'IN');
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_DBSCAN_MODEL_T', 'IN');
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_CLUSTER_ASSIGNED_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE_DROP('DM_PAL',
'PAL_CLUSTERASSIGNMENT_PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE_CREATE('AFLPAL', 'CLUSTERASSIGNMENT',
'DM_PAL', 'PAL_CLUSTERASSIGNMENT_PROC', PAL_CLUSTERASSIGNMENT_PDATA_TBL);
DROP TABLE PAL_DESCAN_DATA_TBL;
CREATE COLUMN TABLE PAL_DBSCAN_DATA_TBL LIKE PAL_DBSCAN_DATA_T;
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES (1,0.10, 0.10, 'B');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES (2,0.10, -2.10, 'A');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES (3,3.10,0.10, 'B');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES (4,10.10,10.10, 'D');
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES (5,3.10,-0.50, 'C');
DROP TABLE PAL_CLUSTER_ASSIGNED_TBL;
CREATE COLUMN TABLE PAL_CLUSTER_ASSIGNED_TBL LIKE PAL_CLUSTER_ASSIGNED_T;
CALL "DM_PAL".PAL_CLUSTERASSIGNMENT_PROC(PAL_DBSCAN_DATA_TBL,
PAL_DBSCAN_MODEL_TBL, #PAL_CONTROL_TBL, PAL_CLUSTER_ASSIGNED_TBL) with OVERVIEW;
SELECT * FROM PAL_CLUSTER_ASSIGNED_TBL;

```

Expected Result

PAL_CLUSTER_ASSIGNED_TBL:

	ID	CLUSTER	DISTANCE
1	1	0	0
2	2	0	2.21
3	3	-1	3
4	4	1	0.050000...
5	5	-1	3.59

For SOM:

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SOM_DATA_T;
CREATE TYPE PAL_SOM_DATA_T AS TABLE(
"TRANS_ID" INTEGER,
"V000" DOUBLE,
"V001" DOUBLE
);
DROP TYPE PAL_SOM_MAP_T;
CREATE TYPE PAL_SOM_MAP_T AS TABLE(
"CELL_ID" INTEGER,
"WEIGHT000" DOUBLE,
"WEIGHT001" DOUBLE,

```

```

"NUMS_TUPLE" INTEGER
);
DROP TYPE PAL_SOM_RESASSIGN_T;
CREATE TYPE PAL_SOM_RESASSIGN_T AS TABLE(
"TRANS_ID" INTEGER,
"CELL_ID" INTEGER
);
DROP TYPE PAL_SOM_MODEL_T;
CREATE TYPE PAL_SOM_MODEL_T AS TABLE(
"JID" INTEGER,
"JSONMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
"NAME" VARCHAR(100),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_SOM_PDATA_TBL;
CREATE COLUMN TABLE PAL_SOM_PDATA_TBL(
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_SOM_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_SOM_DATA_T', 'IN');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_SOM_MAP_T', 'OUT');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_SOM_RESASSIGN_T', 'OUT');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_SOM_MODEL_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_SELF_ORG_MAP_PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'SELFORGMAP', 'DM_PAL',
'PAL_SELF_ORG_MAP_PROC', PAL_SOM_PDATA_TBL);
DROP TABLE PAL_SOM_DATA_TBL;
CREATE COLUMN TABLE PAL_SOM_DATA_TBL LIKE PAL_SOM_DATA_T;
INSERT INTO PAL_SOM_DATA_TBL VALUES (0, 0.1, 0.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (1, 0.22, 0.25);
INSERT INTO PAL_SOM_DATA_TBL VALUES (2, 0.3, 0.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (3, 0.4, 0.5);
INSERT INTO PAL_SOM_DATA_TBL VALUES (4, 0.5, 1.0);
INSERT INTO PAL_SOM_DATA_TBL VALUES (5, 1.1, 15.1);
INSERT INTO PAL_SOM_DATA_TBL VALUES (6, 2.2, 11.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (7, 1.3, 15.3);
INSERT INTO PAL_SOM_DATA_TBL VALUES (8, 1.4, 15.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (9, 3.5, 15.9);
INSERT INTO PAL_SOM_DATA_TBL VALUES (10, 13.1, 1.1);
INSERT INTO PAL_SOM_DATA_TBL VALUES (11, 16.2, 1.5);
INSERT INTO PAL_SOM_DATA_TBL VALUES (12, 16.3, 1.3);
INSERT INTO PAL_SOM_DATA_TBL VALUES (13, 12.4, 2.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (14, 16.9, 1.9);
INSERT INTO PAL_SOM_DATA_TBL VALUES (15, 49.0, 40.1);
INSERT INTO PAL_SOM_DATA_TBL VALUES (16, 50.1, 50.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (17, 50.2, 48.3);
INSERT INTO PAL_SOM_DATA_TBL VALUES (18, 55.3, 50.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (19, 50.4, 56.5);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
"NAME" VARCHAR(100),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 3000, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('HEIGHT_OF_MAP', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('WIDTH_OF_MAP', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NORMALIZATION', 0, null, null);

```

```

DROP TABLE PAL_SOM_MAP_TBL;
CREATE COLUMN TABLE PAL_SOM_MAP_TBL LIKE PAL_SOM_MAP_T;
DROP TABLE PAL_SOM_RESASSIGN_TBL;
CREATE COLUMN TABLE PAL_SOM_RESASSIGN_TBL LIKE PAL_SOM_RESASSIGN_T;
DROP TABLE PAL_SOM_MODEL_TBL;
CREATE COLUMN TABLE PAL_SOM_MODEL_TBL LIKE PAL_SOM_MODEL_T;
CALL "DM_PAL".PAL_SELF_ORG_MAP PROC(PAL_SOM_DATA_TBL, #PAL_CONTROL_TBL,
PAL_SOM_MAP_TBL, PAL_SOM_RESASSIGN_TBL, PAL_SOM_MODEL_TBL) with OVERVIEW;
----- NEXT CLUSTER ASSIGNMENT -----
DROP TYPE PAL_CLUSTER_ASSIGNED_T;
CREATE TYPE PAL_CLUSTER_ASSIGNED_T AS TABLE(
"ID" INTEGER,
"CLUSTER" INTEGER,
"DISTANCE" DOUBLE
);
DROP TABLE PAL_CLUSTERASSIGNMENT_PDATA_TBL;
CREATE COLUMN TABLE PAL_CLUSTERASSIGNMENT_PDATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_SOM_DATA_T', 'IN');
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_SOM_MODEL_T', 'IN');
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_CLUSTERASSIGNMENT_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_CLUSTER_ASSIGNED_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL',
'PAL_CLUSTERASSIGNMENT_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CLUSTERASSIGNMENT',
'DM_PAL', 'PAL_CLUSTERASSIGNMENT_PROC', PAL_CLUSTERASSIGNMENT_PDATA_TBL);
DROP TABLE PAL_SOM_DATA_TBL;
CREATE COLUMN TABLE PAL_SOM_DATA_TBL LIKE PAL_SOM_DATA_T;
INSERT INTO PAL_SOM_DATA_TBL VALUES (0, 0.1, 0.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (1, 0.22, 0.25);
INSERT INTO PAL_SOM_DATA_TBL VALUES (2, 15.3, 5.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (3, 5.4, 6.5);
INSERT INTO PAL_SOM_DATA_TBL VALUES (4, 55.3, 50.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (5, 50.4, 56.5);
DROP TABLE PAL_CLUSTER_ASSIGNED_TBL;
CREATE COLUMN TABLE PAL_CLUSTER_ASSIGNED_TBL LIKE PAL_CLUSTER_ASSIGNED_T;
CALL "DM_PAL".PAL_CLUSTERASSIGNMENT_PROC(PAL_SOM_DATA_TBL, PAL_SOM_MODEL_TBL,
#PAL_CONTROL_TBL, PAL_CLUSTER_ASSIGNED_TBL) with OVERVIEW;
SELECT * FROM PAL_CLUSTER_ASSIGNED_TBL;

```

Expected Result

PAL_CLUSTER_ASSIGNED_TBL:

	ID	CLUSTER	DISTANCE
1	0	15	0.3425638194064086
2	1	15	0.239675959281097
3	2	12	4.006336638246733
4	3	10	4.417253003463392
5	4	0	3.93181962204711
6	5	0	3.892501434760311

Related Information

[K-Means \[page 62\]](#)

[DBSCAN \[page 46\]](#)

[Self-Organizing Maps \[page 101\]](#)

3.1.5 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based data clustering algorithm. It finds a number of clusters starting from the estimated density distribution of corresponding nodes.

DBSCAN requires two parameters: scan radius (eps) and the minimum number of points required to form a cluster (minPts). The algorithm starts with an arbitrary starting point that has not been visited. This point's eps-neighborhood is retrieved, and if the number of points it contains is equal to or greater than minPts, a cluster is started. Otherwise, the point is labeled as noise. These two parameters are very important and are usually determined by user.

PAL provides a method to automatically determine these two parameters. You can choose to specify the parameters by yourself or let the system determine them for you.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

DBSCAN

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'DBSCAN', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Cluster Model OUTPUT table type>	OUT (optional)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>, <cluster model output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar	Attribute data	

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description	Dependency
AUTO_PARAM	Varchar	Specifies whether the MINPTS and RADIUS parameters are determined automatically or by user. <ul style="list-style-type: none">• True: automatically determines the parameters• False: uses parameter values provided by user	
MINPTS	Integer	Specifies the minimum number of points required to form a cluster.	Only valid when AUTO_PARAM is False.
RADIUS	Double	Specifies the scan radius (eps).	Only valid when AUTO_PARAM is False.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
THREAD_NUMBER	Integer	1	Specifies the number of threads.	
DISTANCE_METHOD	Integer	2	<p>Specifies the method to compute the distance between two points.</p> <ul style="list-style-type: none"> • 1: Manhattan • 2: Euclidean • 3: Minkowski • 4: Chebyshev • 5: Standardized Euclidean • 6: Cosine 	
MINKOW_P	Integer	3	Specifies the power of the Minkowski method.	Only valid when DISTANCE_METHOD is 3.
CATEGORY_COL	Integer	-1	Indicates whether the column is category variable. By default, 'string' is category variable, and 'integer' or 'double' is continuous variable. The value of -1 means the default will be used.	
CATEGORY_WEIGHTS	Double	0.707	Represents the weight of category attributes (γ).	

Output Tables

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer	<p>Cluster ID (from 0 to cluster_number – 1)</p> <p>Note: -1 means the point is labeled as noise.</p>	

Table	Column	Column Data Type	Description	Constraint
Cluster Model (optional)	1st column	Integer	Cluster model ID	This must be the first column.
	2nd column	CLOB, varchar, or nvarchar	Cluster model saved as JSON string	The table must be a column table. The minimum length of each unit (row) is 5000.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DBSCAN_DATA_T;
CREATE TYPE PAL_DBSCAN_DATA_T AS TABLE ( ID integer, ATTRIB1 double, ATTRIB2 double);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE( NAME varchar(50), INTARGS integer,
DOUBLEARGS double, STRINGARGS varchar(100));
DROP TYPE PAL_DBSCAN_RESULTS_T;
CREATE TYPE PAL_DBSCAN_RESULTS_T AS TABLE( ID integer, RESULT integer);
DROP TYPE PAL_DBSCAN_MODEL_T;
CREATE TYPE PAL_DBSCAN_MODEL_T AS TABLE(ID integer, MODEL VARCHAR(5000));
DROP TABLE PAL_DBSCAN_PDATA_TBL;
CREATE COLUMN TABLE PAL_DBSCAN_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_DBSCAN_PDATA_TBL VALUES (1, 'DM_PAL','PAL_DBSCAN_DATA_T', 'IN');
INSERT INTO PAL_DBSCAN_PDATA_TBL VALUES (2, 'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_DBSCAN_PDATA_TBL VALUES (3,
'DM_PAL','PAL_DBSCAN_RESULTS_T','OUT');
INSERT INTO PAL_DBSCAN_PDATA_TBL VALUES (4, 'DM_PAL','PAL_DBSCAN_MODEL_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DBSCAN');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'DBSCAN',
'DM_PAL', 'PAL_DBSCAN', 'PAL_DBSCAN_PDATA_TBL');
DROP TABLE PAL_DBSCAN_DATA_TBL;
CREATE COLUMN TABLE PAL_DBSCAN_DATA_TBL ( ID integer, ATTRIB1 double, ATTRIB2
double);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(1,0.10,0.10);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(2,0.11,0.10);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(3,0.10,0.11);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(4,0.11,0.11);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(5,0.12,0.11);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(6,0.11,0.12);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(7,0.12,0.12);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(8,0.12,0.13);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(9,0.13,0.12);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(10,0.13,0.13);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(11,0.13,0.14);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(12,0.14,0.13);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(13,10.10,10.10);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(14,10.11,10.10);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(15,10.10,10.11);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(16,10.11,10.11);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(17,10.11,10.12);

```

```

INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(18,10.12,10.11);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(19,10.12,10.12);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(20,10.12,10.13);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(21,10.13,10.12);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(22,10.13,10.13);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(23,10.13,10.14);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(24,10.14,10.13);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(25,4.10,4.10);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(26,7.11,7.10);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(27,-3.10,-3.11);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(28,16.11,16.11);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(29,20.11,20.12);
INSERT INTO PAL_DBSCAN_DATA_TBL VALUES(30,15.12,15.11);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL( NAME varchar(50), INTARGS
integer, DOUBLEARGS double, STRINGARGS varchar(100));
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('AUTO_PARAM',null,null,'true');
INSERT INTO #PAL_CONTROL_TBL VALUES('DISTANCE_METHOD',1,null,null);
DROP TABLE PAL_DBSCAN_RESULTS_TBL;
CREATE COLUMN TABLE PAL_DBSCAN_RESULTS_TBL like PAL_DBSCAN_RESULTS_T;
DROP TABLE PAL_DBSCAN_MODEL_TBL;
CREATE COLUMN TABLE PAL_DBSCAN_MODEL_TBL like PAL_DBSCAN_MODEL_T;
CALL DM_PAL.PAL_DBSCAN(PAL_DBSCAN_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_DBSCAN_RESULTS_TBL, PAL_DBSCAN_MODEL_TBL) with overview;
SELECT * FROM PAL_DBSCAN_RESULTS_TBL;
SELECT * FROM PAL_DBSCAN_MODEL_TBL;

```

Expected Result

PAL_DBSCAN_RESULTS_TBL:

	ID	RESULT
1	1	0
2	2	0
3	3	0
4	4	0
5	5	0
6	6	0
7	7	0
8	8	0
9	9	0
10	10	0
11	11	0
12	12	0
13	13	1
14	14	1
15	15	1
16	16	1
17	17	1
18	18	1
19	19	1
20	20	1
21	21	1
22	22	1
23	23	1
24	24	1
25	25	-1
26	26	-1
27	27	-1
28	28	-1
29	29	-1
30	30	-1

PAL_DBSCAN_MODEL_TBL:

	ID	MODEL
1	0	{"Algorithm":"DBSCAN","Cluster":[{"ClusterID":0,"CoreObject":[{"CoreContCoor":[0.10,0.1...]

3.1.6 Gaussian Mixture Model (GMM)

GMM is a Gaussian mixture model in which each component has its own weight, mean, and covariance matrix.

Weight means the importance of a Gaussian distribution in the GMM, and mean and covariance matrix are the basic parameters of a Gaussian distribution, as shown in the following formula:

$$\begin{aligned} P(\mathbf{x} | \theta_1 \dots \theta_K p_1 \dots p_K) &= \sum_{k=1}^K p_k p(\mathbf{x} | \theta_k) \\ p(\mathbf{x} | \theta_k) &= p(\mathbf{x} | \mu_k \Sigma_k) \\ &= \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} e^{-\frac{1}{2} (\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)} \end{aligned}$$

Expectation maximization (EM) algorithm is used to inference all of the unknown parameters of GMM. The algorithm performs two steps: the expectation step and the maximization step.

The expectation step calculates the contribution of training sample i to the Gaussian k :

$$\gamma_{i,k}^j = \frac{p_k^j P(\mathbf{x}_i | \theta_k^j)}{\sum_{k'} p_{k'}^j P(\mathbf{x}_i | \theta_{k'}^j)}$$

The maximization step calculates the parameters weight, mean, and covariance matrix:

$$\begin{aligned} p_k^{j+1} &= \frac{1}{N} \sum_{i=1}^N \frac{p_k^j P(\mathbf{x}_i | \theta_k^j)}{\sum_{k'} p_{k'}^j P(\mathbf{x}_i | \theta_{k'}^j)} \\ \mu_k^{j+1} &= \frac{1}{\sum_{i=1}^N \gamma_{i,k}^j} \sum_{i=1}^N \gamma_{i,k}^j \mathbf{x}_i \\ \Sigma_k^{j+1} &= \frac{1}{\sum_{i=1}^N \gamma_{i,k}^j} \sum_{i=1}^N \gamma_{i,k}^j (\mathbf{x}_i - \mu_k) (\mathbf{x}_i - \mu_k)^T \end{aligned}$$

GMM can be used in image segmentation, clustering, and so on. It gives the probability of a sample belonging to each Gaussian component.

Prerequisite

- No missing or null data in the inputs

GMM

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'GMM', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Input table type>	IN
2	<schema_name>	<Parameter table type>	IN
3	<schema_name>	<Initialize parameter table type>	IN
3	<schema_name>	<Cluster result table type>	OUT
4	<schema_name>	<Models table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>,  
<initialize parameter table>, <cluster result table>, <models table>) with  
overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar	Attribute data	

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
INIT_MODE	Integer	<p>Specifies the initialization mode.</p> <ul style="list-style-type: none"> • 0: Sets the number of the Gaussian distributions of GMM in the initialize parameter table. • 1: Sets the seed data (initialization data) in the initialize parameter table. The number of the different seed data equals to the number of the components in GMM.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
THREAD_NUMBER	Integer	1	<p>Specifies the number of threads.</p> <p>Value range: [1,512]</p>	
MAX_ITERATION	Integer	100	<p>Specifies the maximum iteration number of the EM algorithm.</p> <p>Value range: [1, +∞)</p>	
CATEGORY_COL	Integer	-1	<p>Indicates the column as category column. The value of -1 means there is no column to specify.</p> <p>Value range: [-1, column_number-1]</p>	Only valid when the column is integer or string.
CATEGORY_WEIGHT	Double	0.707	<p>Represents the weight of category attributes (γ).</p> <p>Value range: [0, +∞)</p>	
ERROR_TOL	Double	1e-5	<p>Specifies the error tolerance, which is the stop condition.</p> <p>Value range: (0, +∞)</p>	

Name	Data Type	Default Value	Description	Dependency
OUTPUT_FORMAT	Integer	0	<p>Specifies the output format. The value can be 0 or 1.</p> <ul style="list-style-type: none"> • 0: specifies the output format 0 • 1: specifies the output format 1 <p>For details of the two output formats, see the Output Tables section in this topic.</p>	

Initialize Parameter Table

Table	Column	Column Data Type	Description	Constraint
Initialize Parameter	1st column	Integer or string	ID	This must be the first column.
	2nd column	Integer or double	<p>When INIT_MODE is set to 0 in the parameter table, then this value is the number of the components in GMM. In this case, only the first row of this table is used.</p> <p>When INIT_MODE is set to 1 in the parameter table, then each row represents a seed which is the sequence number of the data in data table (starting from 0). There can be more than one seed. For example, setting seed data to 1,2,10 means selecting the 1th, 2th, and 10th data in the input data table as the seeds.</p>	

Output Tables

Output format 0:

Table	Column	Column Data Type	Description	Constraint
Cluster Result	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer	The clustering result. Assign a data to a Gaussian distribution component with the highest probability.	
	Other columns	Double	The probabilities the data belongs to each component in GMM.	
Models	Columns	Double	GMM model. Each column stores a Gaussian distribution model with the weight, mean, and covariance matrix as a vector.	The number of columns is dependent on the number of components in GMM.

Output format 1:

Table	Column	Column Data Type	Description	Constraint
Cluster Result	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer	The clustering result. All the possible labels of a sample.	
	3rd column	Double	The probabilities the data belongs to each component in GMM.	
Models	1st column	varchar or nvarchar	GMM model. Each row stores a Gaussian distribution model with the weight, mean, and covariance matrix as a JSON format.	The number of rows is dependent on the number of components in GMM.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_GMM_DATA_T;
CREATE TYPE PAL_GMM_DATA_T AS TABLE (
ID INTEGER,
ATTRIBUTE1 DOUBLE,
ATTRIBUTE2 DOUBLE,
ATTRIBUTE3 varchar(100)
);
DROP TYPE PAL_GMM_INIT_T;
CREATE TYPE PAL_GMM_INIT_T AS TABLE (
ID INTEGER,
CLUSTERNUM INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
NAME VARCHAR(50),
INTARGS INTEGER,
DOUBLEARGS DOUBLE,
STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_GMM_RESULTS_T;
CREATE TYPE PAL_GMM_RESULTS_T AS TABLE(
ID INTEGER,
RESULT INTEGER,
PROB0 DOUBLE,
PROB1 DOUBLE
);
DROP TYPE PAL_GMM_RESULTSMODEL_T;
CREATE TYPE PAL_GMM_RESULTSMODEL_T AS TABLE(
MIXTURE1 DOUBLE,
MIXTURE2 DOUBLE
);
DROP TABLE PAL_GMM_PDATA_TBL;
CREATE COLUMN TABLE PAL_GMM_PDATA_TBL (
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_GMM_PDATA_TBL VALUES (1,'DM_PAL','PAL_GMM_DATA_T', 'IN');
INSERT INTO PAL_GMM_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_GMM_PDATA_TBL VALUES (3,'DM_PAL','PAL_GMM_INIT_T', 'IN');
INSERT INTO PAL_GMM_PDATA_TBL VALUES (4,'DM_PAL','PAL_GMM_RESULTS_T', 'OUT');
INSERT INTO PAL_GMM_PDATA_TBL VALUES (5,'DM_PAL','PAL_GMM_RESULTSMODEL_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_GMM');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'GMM', 'DM_PAL', 'PAL_GMM',
PAL_GMM_PDATA_TBL);
DROP TABLE PAL_GMM_DATA_TBL;
CREATE COLUMN TABLE PAL_GMM_DATA_TBL LIKE PAL_GMM_DATA_T;
INSERT INTO PAL_GMM_DATA_TBL VALUES(0,0.10,0.10,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(1,0.11,0.10,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(2,0.10,0.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(3,0.11,0.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(4,0.12,0.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(5,0.11,0.12,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(6,0.12,0.12,'A');
```

```

INSERT INTO PAL_GMM_DATA_TBL VALUES(7,0.12,0.13,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(8,0.13,0.12,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(9,0.13,0.13,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(10,0.13,0.14,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(11,0.14,0.13,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(12,10.10,10.10,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(13,10.11,10.10,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(14,10.10,10.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(15,10.11,10.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(16,10.11,10.12,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(17,10.12,10.11,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(18,10.12,10.12,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(19,10.12,10.13,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(20,10.13,10.12,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(21,10.13,10.13,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(22,10.13,10.14,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(23,10.14,10.13,'B');
DROP TABLE PAL_GMM_INIT_TBL;
CREATE COLUMN TABLE PAL_GMM_INIT_TBL LIKE PAL_GMM_INIT_T;
INSERT INTO PAL_GMM_INIT_TBL VALUES(0,2);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('MAX_ITERATION',500,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('INIT_MODE',0,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('ERROR_TOL',null,0.001,null);
DROP TABLE PAL_GMM_RESULTS_TBL;
CREATE COLUMN TABLE PAL_GMM_RESULTS_TBL LIKE PAL_GMM_RESULTS_T;
DROP TABLE PAL_GMM_RESULTSMODEL_TBL;
CREATE COLUMN TABLE PAL_GMM_RESULTSMODEL_TBL LIKE PAL_GMM_RESULTSMODEL_T;
CALL DM_PAL.PAL_GMM(PAL_GMM_DATA_TBL, #PAL_CONTROL_TBL,
PAL_GMM_INIT_TBL,PAL_GMM_RESULTS_TBL,PAL_GMM_RESULTSMODEL_TBL) with OVERVIEW;
SELECT * FROM PAL_GMM_RESULTS_TBL;
SELECT * FROM PAL_GMM_RESULTSMODEL_TBL;

```

Expected Results

PAL_GMM_RESULTS_TBL:

	ID	RESULT	PROB0	PR...
1	0	0	1	0.0...
2	1	0	1	0.0...
3	2	0	1	0.0...
4	3	0	1	0.0...
5	4	0	1	0.0...
6	5	0	1	0.0...
7	6	0	1	0.0...
8	7	0	1	0.0...
9	8	0	1	0.0...
10	9	0	1	0.0...
11	10	0	1	0.0...
12	11	0	1	0.0...
13	12	1	0	1
14	13	1	0	1
15	14	1	0	1
16	15	1	0	1
17	16	1	0	1
18	17	1	0	1
19	18	1	0	1
20	19	1	0	1
21	20	1	0	1
22	21	1	0	1
23	22	1	0	1
24	22	1	0	1

PAL_GMM_RESULTSMODEL_TBL:

	MIXTURE1	MIXTURE2
1	0.5	0.5
2	0.1281944444444442	10.96152777777777
3	0.1281944444444442	10.96152777777777
4	0.1805010277777777	0.3610020555555555
5	0.000257815586419...	0.7111374452160478
6	0.000220621141975...	0.7111002507716032
7	0.002694016408179...	0.02582167480709...
8	0.000220621141975...	0.7111002507716032
9	0.000257815586419...	0.7111374452160478
10	0.002242763838734...	0.02582167480709...
11	0.002694016408179...	0.02582167480709...
12	0.002242763838734...	0.02582167480709...
13	0.060342706895611...	0.06092106111505...

Example 2

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_GMM_DATA_T;
CREATE TYPE PAL_GMM_DATA_T AS TABLE (
ID INTEGER,
ATTRIBUTE1 DOUBLE,
ATTRIBUTE2 DOUBLE,
ATTRIBUTE3 varchar(100)
);
DROP TYPE PAL_GMM_INIT_T;
CREATE TYPE PAL_GMM_INIT_T AS TABLE (
ID INTEGER,
CLUSTERNUM INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
NAME VARCHAR(50),
INTARGS INTEGER,
DOUBLEARGS DOUBLE,
STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_GMM_RESULTS_T;
CREATE TYPE PAL_GMM_RESULTS_T AS TABLE(
ID INTEGER,
RESULT INTEGER,
PROB0 DOUBLE
);
DROP TYPE PAL_GMM_RESULTSMODEL_T;
CREATE TYPE PAL_GMM_RESULTSMODEL_T AS TABLE(
MODELS VARCHAR(1000)
);
DROP TABLE PAL_GMM_PDATA_TBL;
CREATE COLUMN TABLE PAL_GMM_PDATA_TBL (
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_GMM_PDATA_TBL VALUES (1,'DM_PAL','PAL_GMM_DATA_T', 'IN');
INSERT INTO PAL_GMM_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_GMM_PDATA_TBL VALUES (3,'DM_PAL','PAL_GMM_INIT_T', 'IN');
INSERT INTO PAL_GMM_PDATA_TBL VALUES (4,'DM_PAL','PAL_GMM_RESULTS_T', 'OUT');
INSERT INTO PAL_GMM_PDATA_TBL VALUES (5,'DM_PAL','PAL_GMM_RESULTSMODEL_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_GMM');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'GMM', 'DM_PAL', 'PAL_GMM',
PAL_GMM_PDATA_TBL);
DROP TABLE PAL_GMM_DATA_TBL;
CREATE COLUMN TABLE PAL_GMM_DATA_TBL LIKE PAL_GMM_DATA_T;
INSERT INTO PAL_GMM_DATA_TBL VALUES(0,0.10,0.10,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(1,0.11,0.10,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(2,0.10,0.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(3,0.11,0.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(4,0.12,0.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(5,0.11,0.12,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(6,0.12,0.12,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(7,0.12,0.13,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(8,0.13,0.12,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(9,0.13,0.13,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(10,0.13,0.14,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(11,0.14,0.13,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(12,10.10,10.10,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(13,10.11,10.10,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(14,10.10,10.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(15,10.11,10.11,'A');
INSERT INTO PAL_GMM_DATA_TBL VALUES(16,10.11,10.12,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(17,10.12,10.11,'B');
```

```

INSERT INTO PAL_GMM_DATA_TBL VALUES(18,10.12,10.12,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(19,10.12,10.13,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(20,10.13,10.12,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(21,10.13,10.13,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(22,10.13,10.14,'B');
INSERT INTO PAL_GMM_DATA_TBL VALUES(23,10.14,10.13,'B');
DROP TABLE PAL_GMM_INIT_TBL;
CREATE COLUMN TABLE PAL_GMM_INIT_TBL LIKE PAL_GMM_INIT_T;
INSERT INTO PAL_GMM_INIT_TBL VALUES(0,2);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('MAX_ITERATION',500,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('INIT_MODE',0,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('ERROR_TOL',null,0.001,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('OUTPUT_FORMAT',1,null,null);
DROP TABLE PAL_GMM_RESULTS_TBL;
CREATE COLUMN TABLE PAL_GMM_RESULTS_TBL LIKE PAL_GMM_RESULTS_T;
DROP TABLE PAL_GMM_RESULTSMODEL_TBL;
CREATE COLUMN TABLE PAL_GMM_RESULTSMODEL_TBL LIKE PAL_GMM_RESULTSMODEL_T;
CALL DM_PAL.PAL_GMM(PAL_GMM_DATA_TBL, #PAL_CONTROL_TBL,
PAL_GMM_INIT_TBL,PAL_GMM_RESULTS_TBL,PAL_GMM_RESULTSMODEL_TBL) with OVERVIEW;
SELECT * FROM PAL_GMM_RESULTS_TBL;
SELECT * FROM PAL_GMM_RESULTSMODEL_TBL;

```

Expected Results

PAL_GMM_RESULTS_TBL:

	ID	RESULT	PROBO
1	0	0	1
2	0	1	0.00000000000000...
3	1	0	1
4	1	1	0.00000000000000...
5	2	0	1
6	2	1	0.00000000000000...
7	3	0	1
8	3	1	0.00000000000000...
9	4	0	1
10	4	1	0.00000000000000...
11	5	0	1
12	5	1	0.00000000000000...
13	6	0	1
14	6	1	0.00000000000000...
15	7	0	1
16	7	1	0.00000000000000...
17	8	0	1
18	8	1	0.00000000000000...
19	9	0	1
20	9	1	0.00000000000000...
21	10	0	1
22	10	1	0.00000000000000...
23	11	0	1
24	11	1	0.00000000000000...

PAL_GMM_RESULTSMODEL_TBL:

	MODELS
1	{"GaussModel":{"mean":[0.128194444444444,0.128194444444444,0.180501027...]
2	{"GaussModel":{"mean":[10.9615277777778,10.9615277777778,0.36100205555...]

3.1.7 K-Means

In predictive analysis, k-means clustering is a method of cluster analysis. The k-means algorithm partitions n observations or records into k clusters in which each observation belongs to the cluster with the nearest center. In marketing and customer relationship management areas, this algorithm uses customer data to track customer behavior and create strategic business initiatives. Organizations can thus divide their customers into segments based on variants such as demography, customer behavior, customer profitability, measure of risk, and lifetime value of a customer or retention probability.

Clustering works to group records together according to an algorithm or mathematical formula that attempts to find centroids, or centers, around which similar records gravitate. The most common algorithm uses an iterative refinement technique. It is also referred to as Lloyd's algorithm:

Given an initial set of k means m_1, \dots, m_k , the algorithm proceeds by alternating between two steps:

- Assignment step: assigns each observation to the cluster with the closest mean.
- Update step: calculates the new means to be the center of the observations in the cluster.

The algorithm repeats until the assignments no longer change.

The k-means implementation in PAL supports multi-thread, data normalization, different distance level measurement, and cluster quality measurement (Silhouette). The implementation does not support categorical data, but this can be managed through data transformation. The first K and random K starting methods are supported.

Support for Categorical Attributes

If an attribute is of category type, it will be converted to a binary vector and then be used as a numerical attribute. For example, in the below table, "Gender" is of category type.

Customer ID	Age	Income	Gender
T1	31	10,000	Female
T2	27	8,000	Male

Because "Gender" has two distinct values, it will be converted into a binary vector with two dimensions:

Customer ID	Age	Income	Gender_1	Gender_2
T1	31	10,000	0	1
T2	27	8,000	1	0

Thus, the Euclidean distance between T1 and T2 is:

$$d(T_1, T_2) = \sqrt{(31-27)^2 + (10000-8000)^2} = \sqrt{16 + 40000} = \sqrt{4016}$$

Where γ is the weight to be given to the transposed categorical attributes to lessen the impact on the clustering from the 0/1 attributes. Then you can use the traditional method to update the mean of every cluster.

Assuming one cluster only has T1 and T2, the mean is:

Customer ID	Age	Income	Gender_1	Gender_2
Center1	29.0	9000.0	0.5	0.5

The means of categorical attributes will not be outputted. Instead, the means will be replaced by the modes similar to the K-Modes algorithm. Take the below center for example:

	Age	Income	Gender_1	Gender_2
Center	29.0	9000.0	0.25	0.75

Because "Gender_2" is the maximum value, the output will be:

	Age	Income	Gender
Center	29.0	9000.0	Female

Prerequisites

- The input data contains an ID column and the other columns are of integer or double data type.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

KMEANS

This is a clustering function using the k-means algorithm.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'KMEANS', '<schema_name>', 
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Position	Schema Name	Table Type Name	Parameter Type
4	<schema_name>	<Center Point OUTPUT table type>	OUT
5	<schema_name>	<Center Statistics table type>	OUT (optional)
6	<schema_name>	<Statistic table type>	OUT (optional)
7	<schema_name>	<Cluster Model OUTPUT table type>	OUT (optional)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>, <center point output table>, <cluster model output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar	Attribute data	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
GROUP_NUMBER	Integer	No default value	<p>Number of groups (k).</p> <p>Note: If this parameter is not specified, you must specify the range of k using GROUP_NUM-BER_MIN and GROUP_NUM-BER_MAX. Then the algorithm will iterate through the range and return the k with the highest slight silhouette.</p>	
GROUP_NUM-BER_MIN	Integer	No default value	<p>Lower boundary of the range that k falls in.</p> <p>Note: You must specify either an exact value or a range for k. If both are specified, the exact value will be used.</p>	
GROUP_NUM-BER_MAX	Integer	No default value	<p>Upper boundary of the range that k falls in.</p> <p>Note: You must specify either an exact value or a range for k. If both are specified, the exact value will be used.</p>	
DISTANCE_LEVEL	Integer	2	<p>Computes the distance between the item and the cluster center.</p> <ul style="list-style-type: none"> • 1: Manhattan distance • 2: Euclidean distance • 3: Minkowski distance • 4: Chebyshev distance 	

Name	Data Type	Default Value	Description	Dependency
MINKOWSKI_POWER	Double	3.0	When you use the Minkowski distance, this parameter controls the value of power.	Only valid when DISTANCE_LEVEL is 3.
MAX_ITERATION	Integer	100	Maximum iterations.	
INIT_TYPE	Integer	4	Controls how the initial centers are selected: <ul style="list-style-type: none"> • 1: First k observations • 2: Random with replacement • 3: Random without replacement • 4: Patent of selecting the init center (US 6,882,998 B1) 	
NORMALIZATION	Integer	0	Normalization type: <ul style="list-style-type: none"> • 0: No • 1: Yes. For each point X (x_1, x_2, \dots, x_n), the normalized value will be $X' = (x_1/S, x_2/S, \dots, x_n/S)$, where $S = x_1 + x_2 + \dots + x_n$. • 2: for each column C, get the min and max value of C, and then $C'[i] = (C[i] - \text{min}) / (\text{max} - \text{min})$. 	
THREAD_NUMBER	Integer	1	Number of threads.	
CATEGORY_COL	Integer	No default value	Indicates whether the column is category variable. By default, 'string' is category variable, and 'integer' or 'double' is continuous variable.	

Name	Data Type	Default Value	Description	Dependency
CATEGORY_WEIGHTS	Double	0.707	Represents the weight of category attributes (γ).	
EXIT_THRESHOLD	Double	0.000000001	Threshold (actual value) for exiting the iterations.	

Output Tables

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer	Clustered item assigned to class number	
	3rd column	Double	The distance between the given point and the cluster center	
Center Points	1st column	Integer	Cluster center ID	This must be the first column.
	Other columns	Double, varchar, or nvarchar	Cluster center coordinates	

Or

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer	Clustered item assigned to class number	
	3rd column	Double	The distance between the given point and the cluster center	
	4th column	Double	Estimated value (slight silhouette)	

Table	Column	Column Data Type	Description	Constraint
Center Points	1st column	Integer	Cluster center ID	This must be the first column.
	Other columns	Double, varchar, or nvarchar	Cluster center coordinates	
Center Statistics	1st column	Integer	Cluster center ID	This must be the first column.
	2nd column	Double	Statistic value (slight silhouette)	
Statistics	1st column	Varchar or nvarchar	Statistic name	This must be the first column.
	2nd column	Double	Statistic value	
Cluster Model	1st column	Integer	Cluster model ID	This must be the first column.
	2nd column	CLOB, varchar, or nvarchar	Cluster model saved as JSON string	The table must be a column table. The minimum length of each unit (row) is 5000.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL ;
DROP TYPE PAL_KMEANS_DATA_T;
CREATE TYPE PAL_KMEANS_DATA_T AS TABLE(
    "ID" INTEGER,
    "V000" DOUBLE,
    "V001" VARCHAR(2),
    "V002" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_KMEANS_ASSIGNED_T;
CREATE TYPE PAL_KMEANS_ASSIGNED_T AS TABLE(
    "ID" INTEGER,
    "CLUSTER" INTEGER,
    "DISTANCE" DOUBLE,
    "SLIGHT_SILHOUETTE" DOUBLE
)
;
```

```

) ;
DROP TYPE PAL_KMEANS_CENTERS_T;
CREATE TYPE PAL_KMEANS_CENTERS_T AS TABLE(
    "CLUSTER_ID" INTEGER,
    "V000" DOUBLE,
    "V001" VARCHAR(2),
    "V002" DOUBLE
);
DROP TYPE PAL_KMEANS_SIL_CENTERS_T;
CREATE TYPE PAL_KMEANS_SIL_CENTERS_T AS TABLE(
    "CLUSTER_ID" INTEGER,
    "SLIGHT_SILOUETTE" DOUBLE
);
DROP TYPE PAL_KMEANS_STATISTIC_T;
CREATE TYPE PAL_KMEANS_STATISTIC_T AS TABLE(
    "NAME" VARCHAR(50),
    "VALUE" DOUBLE
);
DROP TYPE PAL_KMEANS_MODEL_T;
CREATE TYPE PAL_KMEANS_MODEL_T AS TABLE(
    "JID" INTEGER,
    "JSMODEL" VARCHAR(5000)
);
DROP TABLE PAL_KMEANS_PDATA_TBL;
CREATE COLUMN TABLE PAL_KMEANS_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_KMEANS_DATA_T', 'IN');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_KMEANS_ASSIGNED_T',
'OUT');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_KMEANS_CENTERS_T',
'OUT');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (5, 'DM_PAL',
'PAL_KMEANS_SIL_CENTERS_T', 'OUT');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (6, 'DM_PAL', 'PAL_KMEANS_STATISTIC_T',
'OUT');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (7, 'DM_PAL', 'PAL_KMEANS_MODEL_T',
'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_KMEANS_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'KMEANS', 'DM_PAL',
'PAL_KMEANS_PROC', 'PAL_KMEANS_PDATA_TBL');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('GROUP_NUMBER', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INIT_TYPE', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('DISTANCE_LEVEL', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 100, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXIT_THRESHOLD', null, 1.0E-6, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_WEIGHTS', null, 0.5, null);
DROP TABLE PAL_KMEANS_DATA_TBL;
CREATE COLUMN TABLE PAL_KMEANS_DATA_TBL LIKE PAL_KMEANS_DATA_T;
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (0, 0.5, 'A', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (1, 1.5, 'A', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (2, 1.5, 'A', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (3, 0.5, 'A', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (4, 1.1, 'B', 1.2);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (5, 0.5, 'B', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (6, 1.5, 'B', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (7, 1.5, 'B', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (8, 0.5, 'B', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (9, 1.2, 'C', 16.1);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (10, 15.5, 'C', 15.5);

```

```

INSERT INTO PAL_KMEANS_DATA_TBL VALUES (11, 16.5, 'C', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (12, 16.5, 'C', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (13, 15.5, 'C', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (14, 15.6, 'D', 16.2);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (15, 15.5, 'D', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (16, 16.5, 'D', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (17, 16.5, 'D', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (18, 15.5, 'D', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (19, 15.7, 'A', 1.6);

DROP TABLE PAL_KMEANS_ASSIGNED_TBL;
CREATE COLUMN TABLE PAL_KMEANS_ASSIGNED_TBL LIKE PAL_KMEANS_ASSIGNED_T;
DROP TABLE PAL_KMEANS_CENTERS_TBL;
CREATE COLUMN TABLE PAL_KMEANS_CENTERS_TBL LIKE PAL_KMEANS_CENTERS_T;
DROP TABLE PAL_KMEANS_SIL_CENTERS_TBL;
CREATE COLUMN TABLE PAL_KMEANS_SIL_CENTERS_TBL LIKE PAL_KMEANS_SIL_CENTERS_T;
DROP TABLE PAL_KMEANS_STATISTIC_TBL;
CREATE COLUMN TABLE PAL_KMEANS_STATISTIC_TBL LIKE PAL_KMEANS_STATISTIC_T;
DROP TABLE PAL_KMEANS_MODEL_TBL;
CREATE COLUMN TABLE PAL_KMEANS_MODEL_TBL LIKE PAL_KMEANS_MODEL_T;
CALL "DM_PAL".PAL_KMEANS_PROC(PAL_KMEANS_DATA_TBL, #PAL_CONTROL_TBL,
PAL_KMEANS_ASSIGNED_TBL, PAL_KMEANS_CENTERS_TBL, PAL_KMEANS_SIL_CENTERS_TBL,
PAL_KMEANS_STATISTIC_TBL, PAL_KMEANS_MODEL_TBL) with OVERVIEW;
SELECT * FROM PAL_KMEANS_ASSIGNED_TBL;
SELECT * FROM PAL_KMEANS_CENTERS_TBL;
SELECT * FROM PAL_KMEANS_SIL_CENTERS_TBL;
SELECT * FROM PAL_KMEANS_STATISTIC_TBL;
SELECT * FROM PAL_KMEANS_MODEL_TBL;

```

Expected Result

PAL_KMEANS_ASSIGNED_TBL:

	ID	CLUSTER	DISTANCE	SLIGHT_SILOUETTE
1	0	0	0.8910879487969621	0.9443700340289641
2	1	0	0.8639170309648472	0.9424783519324079
3	2	0	0.8062521617810958	0.9462881150703033
4	3	0	0.8356835545457009	0.944941534308603
5	4	0	0.7445708631492213	0.9502343526704448
6	5	3	0.8910879487969616	0.9407325552117667
7	6	3	0.8356835545457004	0.9444124645366334
8	7	3	0.8062521617810963	0.9465187213724425
9	8	3	0.8639170309648476	0.9461211197877126
10	9	3	0.744570863149222	0.9498994935338706
11	10	2	0.8255266112967905	0.945092293345542
12	11	2	0.9338858664836682	0.9379022885331548
13	12	2	0.8816915771701814	0.9450081614922042
14	13	2	0.7643178163332063	0.9491595311531508
15	14	2	0.9234563013492032	0.939282992463176
16	15	1	0.9016844685872387	0.940436243125126
17	16	1	0.9768852992055255	0.9393860344624405
18	17	1	0.8181783288648536	0.9458778291291957
19	18	1	0.7227990303872558	0.9521697340450603
20	19	1	1.1023417395491886	0.9256786403653229

PAL_KMEANS_CENTERS_TBL:

	CLUSTER_ID	V000	V001	V002
1	0	1.02	A	1.04
2	1	15.940000000000001	D	1.119999999999999
3	2	15.919999999999998	C	16.04
4	3	1.04	B	16.02

PAL_KMEANS_SIL_CENTERS_TBL:

	CLUSTER_ID	SLIGHT_SILOUETTE
1	0	0.9456624776021446
2	1	0.9407096962254291
3	2	0.9432890533974454
4	3	0.9455368708884853

PAL_KMEANS_STATISTIC_TBL:

	NAME	VALUE
1	SlightSilhouette	0.943799524528376

PAL_KMEANS_MODEL_TBL:

	JID	JSMODEL
1	0	{"Algorithm": "KMEANS", "Category": [{"CategoryCol": 2, "CategoryName": ["A", "B", "C", "D"]}, ...]

VALIDATEKMEANS

This is a quality measurement function for k-means clustering. The current version of VALIDATEKMEANS does not support category attributes. You can use the CONV2BINARYVECTOR algorithm to convert category attributes into binary vectors, and then use them as continuous attributes.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'VALIDATEKMEANS',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Type INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN

Position	Schema Name	Table Type Name	Parameter Type
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <type input table>,
<parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Data	1st column	Integer	ID
	Other columns	Integer or double	Attribute data
Type Data/ Class Data	1st column	Integer	ID
	2nd column	Integer	Class type

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
VARIABLE_NUM	Integer	Number of variables

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Name
	2nd column	Double	Measure result

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and

- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
--table type for conv2binaryvector
DROP TYPE PAL_CONV2_IN_T;
CREATE TYPE PAL_CONV2_IN_T AS TABLE(
    "ID" INTEGER,
    "V001" VARCHAR(5)
);
DROP TYPE PAL_CONV2_OUT_T;
CREATE TYPE PAL_CONV2_OUT_T AS TABLE(
    "ID" INTEGER,
    "A0" INTEGER,
    "A1" INTEGER,
    "A2" INTEGER,
    "A3" INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
--table type for silhouette
DROP TYPE PAL_SILHOUETTE_DATA_T;
CREATE TYPE PAL_SILHOUETTE_DATA_T AS TABLE(
    "ID" INTEGER,
    "V000" DOUBLE,
    "A0" INTEGER,
    "A1" INTEGER,
    "A2" INTEGER,
    "A3" INTEGER,
    "V002" DOUBLE
);
DROP TYPE PAL_SILHOUETTE_ASSIGN_T;
CREATE TYPE PAL_SILHOUETTE_ASSIGN_T AS TABLE(
    "ID" INTEGER,
    "CLASS_LABEL" INTEGER
);
DROP TYPE PAL_SILHOUETTE_RESULT_T;
CREATE TYPE PAL_SILHOUETTE_RESULT_T AS TABLE(
    "Silhouette" VARCHAR(15),
    "VALUE" DOUBLE
);
--create procedure
DROP TABLE PAL_CONV2_PDATA_TBL;
CREATE COLUMN TABLE PAL_CONV2_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_CONV2_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_CONV2_IN_T', 'IN');
INSERT INTO PAL_CONV2_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_CONV2_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CONV2_OUT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_CONV2BINARY_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CONV2BINARYVECTOR',
'DM_PAL', 'PAL_CONV2BINARY_PROC', PAL_CONV2_PDATA_TBL);
DROP TABLE PAL_SILHOUETTE_PDATA_TBL;
CREATE COLUMN TABLE PAL_SILHOUETTE_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);

```

```

INSERT INTO PAL_SILHOUETTE_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_SILHOUETTE_DATA_T', 'IN');
INSERT INTO PAL_SILHOUETTE_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_SILHOUETTE_ASSIGN_T', 'IN');
INSERT INTO PAL_SILHOUETTE_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_SILHOUETTE_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_SILHOUETTE_RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_SILHOUETTE_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'VALIDATEKMEANS',
'DM_PAL', 'PAL_SILHOUETTE_PROC', PAL_SILHOUETTE_PDATA_TBL);
--prepare data
DROP TABLE PAL_KMEANS_DATA_TBL;
CREATE COLUMN TABLE PAL_KMEANS_DATA_TBL(
    "ID" INTEGER,
    "V000" DOUBLE,
    "V001" VARCHAR(5),
    "V002" DOUBLE
);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (0, 0.5, 'A', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (1, 1.5, 'A', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (2, 1.5, 'A', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (3, 0.5, 'A', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (4, 1.1, 'B', 1.2);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (5, 0.5, 'B', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (6, 1.5, 'B', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (7, 1.5, 'B', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (8, 0.5, 'B', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (9, 1.2, 'C', 16.1);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (10, 15.5, 'C', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (11, 16.5, 'C', 15.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (12, 16.5, 'C', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (13, 15.5, 'C', 16.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (14, 15.6, 'D', 16.2);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (15, 15.5, 'D', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (16, 16.5, 'D', 0.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (17, 16.5, 'D', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (18, 15.5, 'D', 1.5);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (19, 15.7, 'A', 1.6);
DROP TABLE PAL_CONV2_OUT_TBL;
CREATE COLUMN TABLE PAL_CONV2_OUT_TBL LIKE PAL_CONV2_OUT_T;
DROP VIEW PAL_CONV2_IN_V;
CREATE VIEW PAL_CONV2_IN_V AS SELECT "ID", "V001" FROM PAL_KMEANS_DATA_TBL;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('OUT_PUT_COLUMNS', 5, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
--call procedure
CALL "DM_PAL".PAL_CONV2BINARY_PROC(PAL_CONV2_IN_V, #PAL_CONTROL_TBL,
PAL_CONV2_OUT_TBL) with OVERVIEW;
--prepare data
DROP TABLE PAL_SILHOUETTE_ASSIGN_TBL;
CREATE COLUMN TABLE PAL_SILHOUETTE_ASSIGN_TBL LIKE PAL_SILHOUETTE_ASSIGN_T;
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (0, 0);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (1, 0);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (2, 0);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (3, 0);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (4, 0);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (5, 1);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (6, 1);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (7, 1);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (8, 1);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (9, 1);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (10, 2);

```

```

INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (11, 2);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (12, 2);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (13, 2);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (14, 2);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (15, 3);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (16, 3);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (17, 3);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (18, 3);
INSERT INTO PAL_SILHOUETTE_ASSIGN_TBL VALUES (19, 3);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('VARIABLE_NUM', 6, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
DROP VIEW PAL_SILHOUETTE_V;
CREATE VIEW PAL_SILHOUETTE_V AS (
    SELECT A."ID", "V000", "A0", "A1", "A2", "A3", "V002" FROM
PAL_KMEANS_DATA_TBL AS A JOIN PAL_CONV2_OUT_TBL AS B ON A.ID=B.ID
);
DROP TABLE PAL_SILHOUETTE_RESULT_TBL;
CREATE COLUMN TABLE PAL_SILHOUETTE_RESULT_TBL LIKE PAL_SILHOUETTE_RESULT_T;
--call procedure
CALL "DM_PAL".PAL_SILHOUETTE_PROC(PAL_SILHOUETTE_V, PAL_SILHOUETTE_ASSIGN_TBL,
#PAL_CONTROL_TBL, PAL_SILHOUETTE_RESULT_TBL) with OVERVIEW;
SELECT * FROM PAL_SILHOUETTE_RESULT_TBL;

```

Expected Result

PAL_SILHOUETTE_RESULT_TBL:

Silhouette	VALUE
Silhouette	0.9275111543031423

3.1.8 K-Medians

K-medians is a clustering algorithm similar to K-means. K-medians and K-means both partition n observations into K clusters according to their nearest cluster center. In contrast to K-means, while calculating cluster centers, K-medians uses medians of each feature instead of means of it.

A median value is the middle value of a set of values arranged in order.

Given an initial set of K cluster centers: m_1, \dots, m_k , the algorithm proceeds by alternating between the following two steps and repeats until the assignments no longer change.

- Assignment step: assigns each observation to the cluster with the closest center.
- Update step: calculates the new median of each feature of each cluster to be the new center of that cluster.

The K-medians implementation in PAL supports multi-threads, data normalization, different distance level measurements, and cluster quality measurement (Silhouette). The implementation does not support categorical data, but this can be managed through data transformation. Because median method cannot apply to categorical data, the K-medians implementation uses the most frequent one instead. The first K and random K starting methods are supported.

Support for Categorical Attributes

If an attribute is of category type, it will be converted to a binary vector and then be used as a numerical attribute. For example, in the below table, "Gender" is of category type.

Customer ID	Age	Income	Gender
T1	31	10,000	Female
T2	27	8,000	Male

Because "Gender" has two distinct values, it will be converted into a binary vector with two dimensions:

Customer ID	Age	Income	Gender_1	Gender_2
T1	31	10,000	0	1
T2	27	8,000	1	0

Thus, the Euclidean distance between T1 and T2 is:

$$d(T_1, T_2) = \sqrt{(31-27)^2 + (10000-8000)^2} + \gamma \sqrt{(0-1)^2 + (1-0)^2}$$

Where γ is the weight to be given to the transposed categorical attributes to lessen the impact on the clustering from the 0/1 attributes.

Prerequisites

- The input data contains an ID column and the other columns are of integer, varchar, nvarchar, or double data type.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

KMEDIANs

This is a clustering function using the K-medians algorithm.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'KMEDIANS',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Position	Schema Name	Table Type Name	Parameter Type
4	<schema_name>	<Center Point OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>, <center point output table>) WITH OVERVIEW;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar	Attribute data	

Parameter Table

Mandatory Parameters

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
GROUP_NUMBER	Integer	Number of groups (k).

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
DISTANCE_LEVEL	Integer	2	<p>Computes the distance between the item and the cluster center.</p> <ul style="list-style-type: none"> • 1: Manhattan distance • 2: Euclidean distance • 3: Minkowski distance • 4: Chebyshev distance 	

Name	Data Type	Default Value	Description	Dependency
MINKOWSKI_POWER	Double	3.0	When you use the Minkowski distance, this parameter controls the value of power.	Only valid when DISTANCE_LEVEL is 3.
MAX_ITERATION	Integer	100	Maximum iterations.	
INIT_TYPE	Integer	4	Controls how the initial centers are selected: <ul style="list-style-type: none"> • 1: first k observations • 2: random with replacement • 3: random without replacement • 4: patent of selecting the initial center (US 6,882,998 B1) 	
NORMALIZATION	Integer	0	Normalization type: <ul style="list-style-type: none"> • 0: No • 1: Yes. For each point X (x_1, x_2, \dots, x_n), the normalized value will be $X' = (x_1/S, x_2/S, \dots, x_n/S)$, where $S = \sqrt{ x_1 ^2 + x_2 ^2 + \dots + x_n ^2}$. • 2: For each column C, get the min and max value of C, and then $C'[i] = (C[i] - \text{min}) / (\text{max} - \text{min})$. 	
THREAD_NUMBER	Integer	1	Number of threads.	
CATEGORY_COL	Integer	No default value	Indicates whether the column is category variable. By default, 'string' is category variable, and 'integer' or 'double' is continuous variable.	
CATEGORY_WEIGHTS	Double	0.707	Represents the weight of category attributes (γ).	
EXIT_THRESHOLD	Double	0.000000001	Threshold (actual value) for exiting the iterations.	

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer	Clustered item assigned to class number
	3rd column	Double	The distance between the cluster and each point in the cluster.
Center Points	1st column	Integer	Cluster center ID
	Other columns	Double, varchar, or nvarchar	Outputs means if the column is continuous variable; outputs modes if the column is category variable.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
--CREATE TABLE TYPE
DROP TYPE PAL_KMEDIANs_DATA_T;
CREATE TYPE PAL_KMEDIANs_DATA_T AS TABLE(
    "ID" INTEGER,
    "V000" DOUBLE,
    "V001" VARCHAR(5),
    "V002" DOUBLE
);
DROP TYPE PAL_KMEDIANs_ASSIGN_T;
CREATE TYPE PAL_KMEDIANs_ASSIGN_T AS TABLE(
    "ID" INTEGER,
    "CENTER_ASSIGN" INTEGER,
    "DISTANCE" DOUBLE
);
DROP TYPE PAL_KMEDIANs_CENTERS_T;
CREATE TYPE PAL_KMEDIANs_CENTERS_T AS TABLE(
    "CENTER_ID" INTEGER,
    "V000" DOUBLE,
    "V001" VARCHAR(5),
    "V002" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
--CREATE PROCEDURE
DROP TABLE PAL_KMEDIANs_PDATA_TBL;
CREATE COLUMN TABLE PAL_KMEDIANs_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
)

```

```

);
DELETE FROM PAL_KMEDIANSPDATA_TBL;
INSERT INTO PAL_KMEDIANSPDATA_TBL VALUES (1, 'DM_PAL', 'PAL_KMEDIANSDATA_T',
'IN');
INSERT INTO PAL_KMEDIANSPDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_KMEDIANSPDATA_TBL VALUES (3, 'DM_PAL', 'PAL_KMEDIANSSIGN_T',
'OUT');
INSERT INTO PAL_KMEDIANSPDATA_TBL VALUES (4, 'DM_PAL',
'PAL_KMEDIANSCENTERS_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_KMEDIANSPROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'KMEDIANS', 'DM_PAL',
'PAL_KMEDIANSPROC', 'PAL_KMEDIANSPDATA_TBL');
--CREATE TABLES
DROP TABLE PAL_KMEDIANSDATA_TBL;
CREATE COLUMN TABLE PAL_KMEDIANSDATA_TBL LIKE PAL_KMEDIANSDATA_T;
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (0, 0.5, 'A', 0.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (1, 1.5, 'A', 0.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (2, 1.5, 'A', 1.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (3, 0.5, 'A', 1.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (4, 1.1, 'B', 1.2);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (5, 0.5, 'B', 15.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (6, 1.5, 'B', 15.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (7, 1.5, 'B', 16.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (8, 0.5, 'B', 16.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (9, 1.2, 'C', 16.1);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (10, 15.5, 'C', 15.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (11, 16.5, 'C', 15.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (12, 16.5, 'C', 16.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (13, 15.5, 'C', 16.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (14, 15.6, 'D', 16.2);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (15, 15.5, 'D', 0.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (16, 16.5, 'D', 0.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (17, 16.5, 'D', 1.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (18, 15.5, 'D', 1.5);
INSERT INTO PAL_KMEDIANSDATA_TBL VALUES (19, 15.7, 'A', 1.6);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD NUMBER', 3, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('GROUP NUMBER', 4, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INIT_TYPE', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('DISTANCE_LEVEL', 2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 100, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXIT_THRESHOLD', NULL, 1.0E-6, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NORMALIZATION', 0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_WEIGHTS', NULL, 0.5, NULL);
--INSERT INTO #PAL_CONTROL_TBL VALUES ('MINKOWSKI_POWER', NULL, 3, NULL);
DROP TABLE PAL_KMEDIANSSIGN_TBL;
CREATE COLUMN TABLE PAL_KMEDIANSSIGN_TBL LIKE PAL_KMEDIANSSIGN_T;
DROP TABLE PAL_KMEDIANSCENTERS_TBL;
CREATE COLUMN TABLE PAL_KMEDIANSCENTERS_TBL LIKE PAL_KMEDIANSCENTERS_T;
--CALL PROCEDURE
CALL DM_PAL.PAL_KMEDIANSPROC(PAL_KMEDIANSDATA_TBL, "#PAL_CONTROL_TBL",
PAL_KMEDIANSSIGN_TBL, PAL_KMEDIANSCENTERS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_KMEDIANSCENTERS_TBL;
SELECT * FROM PAL_KMEDIANSSIGN_TBL;

```

Expected Results

PAL_KMEDIANSCENTERS_TBL:

	CENTER_ID	V000	V001	V002
1	0	1.1	A	1.2
2	1	15.7	D	1.5
3	2	15.6	C	16.2
4	3	1.2	B	16.1

PAL_KMEDIANASSIGN_TBL:

	ID	CENTER_ASSIGN	DISTANCE
1	0	0	0.9219544457292888
2	1	0	0.8062257748298549
3	2	0	0.4999999999999999...
4	3	0	0.670820393249937
5	4	0	0.7071067811865476
6	5	3	0.9219544457292896
7	6	3	0.6708203932499383
8	7	3	0.4999999999999989
9	8	3	0.8062257748298542
10	9	3	0.7071067811865476
11	10	2	0.7071067811865468
12	11	2	1.1401754250991378
13	12	2	0.9486832980505143
14	13	2	0.3162277660168385
15	14	2	0.7071067811865476
16	15	1	1.0198039027185568
17	16	1	1.2806248474865702
18	17	1	0.8000000000000007
19	18	1	0.1999999999999993
20	19	1	0.8071067811865477

3.1.9 K-Medoids

This is a clustering algorithm related to the K-means algorithm. Both k-medoids and k-means algorithms partition n observations into k clusters in which each observation is assigned to the cluster with the closest center. In contrast to k-means algorithm, k-medoids clustering does not calculate means, but medoids to be the new cluster centers.

A medoid is defined as the center of a cluster, whose average dissimilarity to all the objects in the cluster is minimal. Compared with the K-means algorithm, K-medoids is more robust to noise and outliers.

Given an initial set of K medoids: m₁, ..., m_K, the algorithm proceeds by alternating between two steps as below:

- Assignment step: assigns each observation to the cluster with the closest center.
- Update step: calculates the new medoid to be the center of the observations for each cluster.

The algorithm repeats until the assignments no longer change.

In PAL, the K-medoids algorithm supports multi-threads, data normalization, different distance level measurements, and cluster quality measurement (Silhouette). It does not support categorical data, but this can be managed through data transformation. The first K and random K starting methods are supported.

Support for Categorical Attributes

If an attribute is of category type, it will be converted to a binary vector and then be used as a numerical attribute. For example, in the below table, "Gender" is of category type.

Customer ID	Age	Income	Gender
T1	31	10,000	Female
T2	27	8,000	Male

Because "Gender" has two distinct values, it will be converted into a binary vector with two dimensions:

Customer ID	Age	Income	Gender_1	Gender_2
T1	31	10,000	0	1
T2	27	8,000	1	0

Thus, the Euclidean distance between T1 and T2 is:

$$d(T_1, T_2) = \sqrt{(31-27)^2 + (10000-8000)^2} + \gamma \sqrt{(0-1)^2 + (1-0)^2}$$

Where γ is the weight to be given to the transposed categorical attributes to lessen the impact on the clustering from the 0/1 attributes. Then you can use the traditional method to update the medoid of every cluster.

Prerequisites

- The input data contains an ID column and the other columns are of integer, varchar, nvarchar, or double data type.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

KMEDOIDS

This is a clustering function using the K-medoids algorithm.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'KMEDOIDS',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Center Point OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>, <center point output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar	Attribute data	

Parameter Table

Mandatory Parameters

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
GROUP_NUMBER	Integer	Number of groups (k).

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
DISTANCE_LEVEL	Integer	2	<p>Computes the distance between the item and the cluster center.</p> <ul style="list-style-type: none"> • 1: Manhattan distance • 2: Euclidean distance • 3: Minkowski distance • 4: Chebyshev distance 	
MINKOWSKI_POWER	Double	3.0	When you use the Minkowski distance, this parameter controls the value of power.	Only valid when DISTANCE_LEVEL is 3.
MAX_ITERATION	Integer	100	Maximum iterations.	
INIT_TYPE	Integer	4	Controls how the initial centers are selected: <ul style="list-style-type: none"> • 1: First k observations • 2: Random with replacement • 3: Random without replacement • 4: Patent of selecting the init center (US 6,882,998 B1) 	
NORMALIZATION	Integer	0	Normalization type: <ul style="list-style-type: none"> • 0: No • 1: Yes. For each point X (x_1, x_2, \dots, x_n), the normalized value will be $X' = (x_1/S, x_2/S, \dots, x_n/S)$, where $S = x_1 + x_2 + \dots + x_n$. • 2: For each column C, get the min and max value of C, and then $C'[i] = (C[i] - \text{min}) / (\text{max} - \text{min})$. 	
THREAD_NUMBER	Integer	1	Number of threads.	

Name	Data Type	Default Value	Description	Dependency
CATEGORY_COL	Integer	No default value	Indicates whether the column is category variable. By default, 'string' is category variable, and 'integer' or 'double' is continuous variable.	
CATEGORY_WEIGHTS	Double	0.707	Represents the weight of category attributes (γ).	
EXIT_THRESHOLD	Double	0.000000001	Threshold (actual value) for exiting the iterations.	

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer	Clustered item assigned to class number
	3rd column	Double	The distance between the cluster and each point in the cluster
Center Points	1st column	Integer	Cluster center ID
	Other columns	Double, varchar, or nvarchar	Outputs means if the column is continuous variable; outputs modes if the column is category variable.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_KMEDOIDS_DATA_T;
CREATE TYPE PAL_KMEDOIDS_DATA_T AS TABLE (
    "ID" INTEGER,
    "V000" DOUBLE,
    "V001" VARCHAR(5),
    "V002" DOUBLE
);
DROP TYPE PAL_KMEDOIDS_ASSIGN_T;
CREATE TYPE PAL_KMEDOIDS_ASSIGN_T AS TABLE (
    "ID" INTEGER,

```

```

    "CENTER_ASSIGN" INTEGER,
    "DISTANCE" DOUBLE
);

DROP TYPE PAL_KMEDOIDS_CENTERS_T;
CREATE TYPE PAL_KMEDOIDS_CENTERS_T AS TABLE(
    "CENTER_ID" INTEGER,
    "V000" DOUBLE,
    "V001" VARCHAR(5),
    "V002" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_KMEDOIDS_PDATA_TBL;
CREATE COLUMN TABLE PAL_KMEDOIDS_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_KMEDOIDS_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_KMEDOIDS_DATA_T',
'IN');
INSERT INTO PAL_KMEDOIDS_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_KMEDOIDS_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_KMEDOIDS_ASSIGN_T',
'OUT');
INSERT INTO PAL_KMEDOIDS_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_KMEDOIDS_CENTERS_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_KMEDOIDS_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'KMEDOIDS', 'DM_PAL',
'PAL_KMEDOIDS_PROC', DM_PAL.PAL_KMEDOIDS_PDATA_TBL);
DROP TABLE PAL_KMEDOIDS_DATA_TBL;
CREATE COLUMN TABLE PAL_KMEDOIDS_DATA_TBL LIKE PAL_KMEDOIDS_DATA_T;
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (0, 0.5, 'A', 0.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (1, 1.5, 'A', 0.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (2, 1.5, 'A', 1.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (3, 0.5, 'A', 1.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (4, 1.1, 'B', 1.2);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (5, 0.5, 'B', 15.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (6, 1.5, 'B', 15.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (7, 1.5, 'B', 16.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (8, 0.5, 'B', 16.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (9, 1.2, 'C', 16.1);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (10, 15.5, 'C', 15.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (11, 16.5, 'C', 15.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (12, 16.5, 'C', 16.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (13, 15.5, 'C', 16.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (14, 15.6, 'D', 16.2);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (15, 15.5, 'D', 0.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (16, 16.5, 'D', 0.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (17, 16.5, 'D', 1.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (18, 15.5, 'D', 1.5);
INSERT INTO PAL_KMEDOIDS_DATA_TBL VALUES (19, 15.7, 'A', 1.6);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INT,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('GROUP_NUMBER', 4, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INIT_TYPE', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('DISTANCE_LEVEL', 2, NULL, NULL);

```

```

INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 100, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXIT_THRESHOLD', NULL, 1.0E-6, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NORMALIZATION', 0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_WEIGHTS', NULL, 0.5, NULL);
DROP TABLE PAL_KMEDOIDS_ASSIGN_TBL;
CREATE COLUMN TABLE PAL_KMEDOIDS_ASSIGN_TBL LIKE PAL_KMEDOIDS_ASSIGN_T;

DROP TABLE PAL_KMEDOIDS_CENTERS_TBL;
CREATE COLUMN TABLE PAL_KMEDOIDS_CENTERS_TBL LIKE PAL_KMEDOIDS_CENTERS_T;
CALL DM_PAL.PAL_KMEDOIDS_PROC(PAL_KMEDOIDS_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_KMEDOIDS_ASSIGN_TBL, PAL_KMEDOIDS_CENTERS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_KMEDOIDS_ASSIGN_TBL;
SELECT * FROM PAL_KMEDOIDS_CENTERS_TBL;

```

Expected Result

PAL_KMEDOIDS_ASSIGN_TBL:

ID	CENTER_ASSIGN	DISTANCE
0	0	1.4142135623730951
1	0	1.0
2	0	0.0
3	0	1.0
4	0	1.2071067811865475
5	3	1.4142135623730951
6	3	1.0
7	3	0.0
8	3	1.0
9	3	1.2071067811865466
10	2	1.0
11	2	1.4142135623730951
12	2	1.0
13	2	0.0
14	2	1.0233345472033861
15	1	1.0
16	1	1.4142135623730951
17	1	1.0
18	1	0.0
19	1	0.930713578936526

PAL_KMEDOIDS_CENTERS_TBL:

CENTER_ID	V000	V001	V002
0	1.5	A	1.5
1	15.5	D	1.5
2	15.5	C	16.5
3	1.5	B	16.5

3.1.10 LDA Estimation and Inference

Latent Dirichlet allocation (LDA) is a generative model in which each item of a collection is modeled as a distribution over an underlying set of groups (topics). In the context of text modeling, it posits that each document in the text corpora is composed by several topics with different probabilities and each word belongs to certain topics with different probabilities. In PAL, the parameter inference is done via Gibbs sampling.

LDAESTIMATE

This function performs LDA estimation.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LDAESTIMATE',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Input table type>	IN
2	<schema_name>	<Parameter table type>	IN
3	<schema_name>	<Dictionary table type>	OUT
4	<schema_name>	<Topic word distribution table type>	OUT
5	<schema_name>	<Document topic distribution table type>	OUT
6	<schema_name>	<General information table type>	OUT
7	<schema_name>	<Topic top words table type>	OUT
8	<schema_name>	<Word topic assignment table type>	OUT (optional)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output tables>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, varchar, or nvarchar	Document ID
	2nd column	VARCHAR or nvarchar	Document texts separated by certain delimit

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
TOPICS	Integer	Expected number of topics in the corpus

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
ALPHA	Double	50/Value of parameter TOPICS	Hyper-parameter of the Dirichlet prior to document-topic distribution	
BETA	Double	0.1	Hyper-parameter of the Dirichlet prior to topic-word distribution	
BURNIN	Integer	0	Number of omitted Gibbs iterations at the beginning	
ITERATION	Integer	2000	Number of Gibbs iterations	
THIN	Integer	1	Number of omitted in-between Gibbs iterations	

Name	Data Type	Default Value	Description	Dependency
SEED	Integer	0	Indicates the seed used to initialize the random number generator. <ul style="list-style-type: none">• 0: uses the system time• Not 0: uses the specified seed	
MAX_TOP_WORDS	Integer	0	Specifies the maximum number of words to be output for each topic.	Only valid when the topic top words output table is provided. It cannot be used together with parameter THRESH-OLD_TOP_WORDS.
THRESH-OLD_TOP_WORDS	Double	0	The algorithm outputs top words for each topic if the probability is larger than this threshold.	Only valid when the topic top words output table is provided. It cannot be used together with parameter MAX_TOP_WORDS.
INIT	Integer	0	Specifies initialization method for Gibbs sampling: <ul style="list-style-type: none">• 0: Uniform• 1: Initialization by sampling	
DELIMIT	Varchar	Space	Specifies the delimits to separate each word in the document. For example, if the words are separated by <code>,</code> and <code>:</code> , then the delimit can be <code>,:</code> .	

Output Tables

Table	Column	Column Data Type	Description
Dictionary	1st column	Integer	Word ID

Table	Column	Column Data Type	Description
	2nd column	Varchar or nvarchar	Word text
Topic Word Distribution	1st column	Integer	Topic ID
	2nd column	Integer	Word ID
	3rd column	Double	Probability of word given topic
Document Topic Distribution	1st column	Integer, varchar, or nvarchar (must be the same as the input data table)	Document ID
	2nd column	Integer	Topic ID
	3rd column	Double	Probability of topic given document
General Information	1st column	Varchar or nvarchar	Name
	2nd column	Integer	Value of integer type
	3rd column	Double	Value of double type
	4th column	Varchar or nvarchar	Value of varchar or nvarchar type
Topic Top Words	1st column	Integer	Topic ID
	2nd column	Varchar or nvarchar	Word separated by space
Word Topic Assignment (optional)	1st column	Integer, varchar, or nvarchar (must be the same as the input data table)	Document ID
	2nd column	Integer	Word ID
	3rd column	Integer	Topic ID

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE DATA_T;
CREATE TYPE DATA_T AS TABLE (
    "ID" INTEGER,
```

```

    "TEXT" VARCHAR (1000)
);

DROP TYPE PARAMETERS_T;
CREATE TYPE PARAMETERS_T AS TABLE(
    "NAME" VARCHAR (50),
    "INT_ARGS" INTEGER,
    "DOUBLE_ARGS" DOUBLE,
    "STRING_ARGS" VARCHAR (100)
);
DROP TYPE DICTIONARY_T;
CREATE TYPE DICTIONARY_T AS TABLE(
    "WORDID" INTEGER,
    "TEXT" VARCHAR (100)
);
DROP TYPE TOPICWORDDISTRIBUTION_T;
CREATE TYPE TOPICWORDDISTRIBUTION_T AS TABLE(
    "TOPICID" INTEGER,
    "WORDID" INTEGER,
    "PROBABILITY" DOUBLE
);
DROP TYPE DOCTOPICDISTRIBUTION_T;
CREATE TYPE DOCTOPICDISTRIBUTION_T AS TABLE(
    "DOCUMENTID" INTEGER,
    "TOPICID" INTEGER,
    "PROBABILITY" DOUBLE
);
DROP TYPE GENERALINFO_T;
CREATE TYPE GENERALINFO_T AS TABLE(
    "NAME" VARCHAR (50),
    "INT_ARGS" INTEGER,
    "DOUBLE_ARGS" DOUBLE,
    "STRING_ARGS" VARCHAR (100)
);
DROP TYPE TOPWORDS_T;
CREATE TYPE TOPWORDS_T AS TABLE(
    "TOPICID" INTEGER,
    "TEXT" VARCHAR (1000)
);
DROP TABLE PDATA_TBL;
CREATE COLUMN TABLE PDATA_TBL("POSITION" INT,"SCHEMA_NAME"
NVARCHAR(256),"TYPE_NAME" NVARCHAR (256),"PARAMETER_TYPE" VARCHAR (7));
INSERT INTO PDATA_TBL VALUES (1,'DM_PAL','DATA_T','IN');
INSERT INTO PDATA_TBL VALUES (2,'DM_PAL','PARAMETERS_T','IN');
INSERT INTO PDATA_TBL VALUES (3,'DM_PAL','DICTIONARY_T','OUT');
INSERT INTO PDATA_TBL VALUES (4,'DM_PAL','TOPICWORDDISTRIBUTION_T','OUT');
INSERT INTO PDATA_TBL VALUES (5,'DM_PAL','DOCTOPICDISTRIBUTION_T','OUT');
INSERT INTO PDATA_TBL VALUES (6,'DM_PAL','GENERALINFO_T','OUT');
INSERT INTO PDATA_TBL VALUES (7,'DM_PAL','TOPWORDS_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_LDAESTIMATE');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL','LDAESTIMATE','DM_PAL','PAL_LDAESTIMATE',PDATA_TBL);
DROP TABLE DATA_TBL;
CREATE COLUMN TABLE DATA_TBL like DATA_T;
INSERT INTO DATA_TBL VALUES (10 , 'cpu harddisk graphiccard cpu monitor keyboard
cpu memory memory');
INSERT INTO DATA_TBL VALUES (20 , 'tires mountainbike wheels valve helmet
mountainbike rearfender tires mountainbike mountainbike');
INSERT INTO DATA_TBL VALUES (30 , 'carseat toy strollers toy toy spoon toy
strollers toy carseat');
INSERT INTO DATA_TBL VALUES (40 , 'sweaters sweaters sweaters boots sweaters
rings vest vest shoe sweaters');
DROP TABLE PARAMETERS_TBL;
CREATE COLUMN TABLE PARAMETERS_TBL like PARAMETERS_T;
INSERT INTO PARAMETERS_TBL VALUES ('TOPICS',6,null,null);
INSERT INTO PARAMETERS_TBL VALUES ('BURNIN',50,null,null);
INSERT INTO PARAMETERS_TBL VALUES ('THIN',10,null,null);

```

```

INSERT INTO PARAMETERS_TBL VALUES ('ITERATION',100,null,null);
INSERT INTO PARAMETERS_TBL VALUES ('SEED',33,null,null);
INSERT INTO PARAMETERS_TBL VALUES ('ALPHA',null,0.1,null);
INSERT INTO PARAMETERS_TBL VALUES ('MAX_TOP_WORDS',5,null,null);
DROP TABLE DICTIONARY_TBL;
CREATE COLUMN TABLE DICTIONARY_TBL LIKE DICTIONARY_T;
DROP TABLE TOPICWORDDIST_TBL;
CREATE COLUMN TABLE TOPICWORDDIST_TBL LIKE TOPICWORDDISTRIBUTION_T;
DROP TABLE DOCTOPICDIST_TBL;
CREATE COLUMN TABLE DOCTOPICDIST_TBL LIKE DOCTOPICDISTRIBUTION_T;
DROP TABLE GENERALINFO_TBL;
CREATE COLUMN TABLE GENERALINFO_TBL LIKE GENERALINFO_T;
DROP TABLE TOPWORDS_TBL;
CREATE COLUMN TABLE TOPWORDS_TBL LIKE TOPWORDS_T;
CALL DM_PAL.PAL_LDAESTIMATE(DATA_TBL, PARAMETERS_TBL, DICTIONARY_TBL,
TOPICWORDDIST_TBL, DOCTOPICDIST_TBL, GENERALINFO_TBL, TOPWORDS_TBL) with
overview;
SELECT * FROM DICTIONARY_TBL;
SELECT * FROM TOPICWORDDIST_TBL;
SELECT * FROM DOCTOPICDIST_TBL;
SELECT * FROM GENERALINFO_TBL;
SELECT * FROM TOPWORDS_TBL;

```

Expected Results

DICTIONARY_TBL:

	WORDID	TEXT
1	17	boots
2	12	carseat
3	0	cpu
4	2	grap...
5	1	hard...
6	10	helmet
7	4	keyb...
8	5	mem...
9	3	moni...
10	7	mou...
11	11	rearf...
12	18	rings
13	20	shoe
14	15	spoon
15	14	stroll...
16	16	swea...
17	6	tires
18	13	toy
19	9	valve
20	19	vest
21	8	wheels

TOPICWORDDIST_TBL:

	TOPICID	WORDID	PROBABILITY
1	0	0	0.2792792792792793
2	0	1	0.09909909909909911
3	0	2	0.09909909909909911
4	0	3	0.09909909909909911
5	0	4	0.09909909909909911
6	0	5	0.1891891891891892
7	0	6	0.009009009009009...
8	0	7	0.009009009009009...
9	0	8	0.009009009009009...
10	0	9	0.009009009009009...
11	0	10	0.009009009009009...
12	0	11	0.009009009009009...
13	0	12	0.009009009009009...
14	0	13	0.009009009009009...
15	0	14	0.009009009009009...
16	0	15	0.009009009009009...
17	0	16	0.009009009009009...
18	0	17	0.009009009009009...
19	0	18	0.009009009009009...
20	0	19	0.009009009009009...
21	0	20	0.009009009009009...
22	1	0	0.008264462809917...
23	1	1	0.008264462809917...
24	1	2	0.008264462809917

DOCTOPICDIST_TBL:

	DOCUMENTID	TOPICID	PROBABILITY
1	10	0	0.9479166666666666
2	10	1	0.01041666666666668
3	10	2	0.01041666666666668
4	10	3	0.01041666666666668
5	10	4	0.01041666666666668
6	10	5	0.01041666666666668
7	20	0	0.009433962264150945
8	20	1	0.009433962264150945
9	20	2	0.009433962264150945
10	20	3	0.009433962264150945
11	20	4	0.9528301886792453
12	20	5	0.009433962264150945
13	30	0	0.009433962264150945
14	30	1	0.009433962264150945
15	30	2	0.009433962264150945
16	30	3	0.009433962264150945
17	30	4	0.009433962264150945
18	30	5	0.9528301886792453
19	40	0	0.009433962264150945
20	40	1	0.9528301886792453
21	40	2	0.009433962264150945
22	40	3	0.009433962264150945
23	40	4	0.009433962264150945
24	40	5	0.009433962264150945

GENERALINFO_TBL:

	NAME	INT_ARGS	DOUBLE_ARGS	STRING_ARGS
1	TOPICS	6	0	
2	DOCUMENTS	4	0	
3	VOCABULA...	21	0	
4	BURNIN	50	0	
5	THIN	10	0	
6	SEED	33	0	
7	ITERATION	100	0	
8	INIT	0	0	
9	ALPHA	0	0.1	
10	BETA	0	0.1	
11	LOG_LIKELI...	0	-64.21315054...	
12	DELIMIT	0	0	

TOPWORDS_TBL:

	TOPICID	TEXT
1	0	cpu memory graphiccard keyboard harddisk
2	1	sweaters vest shoe rings boots
3	2	strollers tires graphiccard cpu valve
4	3	strollers tires graphiccard cpu valve
5	4	mountainbike tires rearfender helmet valve
6	5	toy strollers carseat spoon graphiccard

LDAINFERENCE

This function infersences the topic assignment for new documents based on the previous LDA estimation results.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LDAINFERENCE',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Input table type>	IN
2	<schema_name>	<Parameter table type>	IN
3	<schema_name>	<Topic word distribution table type>	IN
4	<schema_name>	<Dictionary table type>	IN
5	<schema_name>	<General information table type>	IN
6	<schema_name>	<Document topic distribution table type>	OUT
7	<schema_name>	<General information table type>	OUT (optional)
8	<schema_name>	<Word topic assignment table type>	OUT (optional)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input tables>, <parameter table>, <output tables>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, varchar, or nvarchar	Document ID
	2nd column	VARCHAR or NVARCHAR	Document texts separated by certain delimit
Topic Word Distribution	1st column	Integer	Topic ID
	2nd column	Integer	Word ID
	3rd column	Double	Probability of word given topic
Dictionary	1st column	Integer	Word ID
	2nd column	VARCHAR or NVARCHAR	Word text
General Information	1st column	VARCHAR or NVARCHAR	Name
	2nd column	Integer	Value of integer type
	3rd column	Double	Value of double type
	4th column	VARCHAR or NVARCHAR	Value of varchar or nvarchar type

Parameter Table

Mandatory Parameter

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
BURNIN	Integer	0	Number of omitted Gibbs iterations at the beginning. This value takes precedence over the corresponding one in the general information table.	
ITERATION	Integer	2000	Number of Gibbs iterations. This value takes precedence over the corresponding one in the general information table.	
THIN	Integer	1	Number of omitted in-between Gibbs iterations. This value takes precedence over the corresponding one in the general information table.	
SEED	Integer	0	<p>Indicates the seed used to initialize the random number generator. This value takes precedence over the corresponding one in the general information table.</p> <ul style="list-style-type: none"> • 0: uses the system time • Not 0: uses the specified seed 	
INIT	Integer	0	<p>Specifies initialization method for Gibbs sampling. This value takes precedence over the corresponding one in the general information table.</p> <ul style="list-style-type: none"> • 0: Uniform • 1: Initialization by sampling 	

Name	Data Type	Default Value	Description	Dependency
DELIMIT	Varchar	Space	<p>Specifies the delimits to separate each word in the document. This value takes precedence over the corresponding one in the general information table.</p> <p>For example, if the words are separated by <code>,</code> and <code>:</code>, then the delimit can be <code>,</code><code>:</code>.</p>	Only valid for the first alternative input table schema.

Output Tables

Table	Column	Column Data Type	Description
Document Topic Distribution	1st column	Integer, varchar, or nvarchar (must be the same as the input data table)	Document ID
	2nd column	Integer	Topic ID
	3rd column	Double	Probability of topic given document
General Information (optional)	1st column	VARCHAR or NVARCHAR	Name
	2nd column	Integer	Value of integer type
	3rd column	Double	Value of double type
	4th column	VARCHAR or NVARCHAR	Value of varchar or nvarchar type
Word Topic Assignment (optional)	1st column	Integer, varchar, or nvarchar (must be the same as the input data table)	Document ID
	2nd column	Integer	Word ID
	3rd column	Integer	Topic ID

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE DATA_T;
CREATE TYPE DATA_T AS TABLE(
    "ID" INTEGER,
    "TEXT" VARCHAR (1000)
);

DROP TYPE PARAMETERS_T;
CREATE TYPE PARAMETERS_T AS TABLE(
    "NAME" VARCHAR (50),
    "INT_ARGS" INTEGER,
    "DOUBLE_ARGS" DOUBLE,
    "STRING_ARGS" VARCHAR (100)
);
DROP TYPE TOPICWORDDISTRIBUTION_T;
CREATE TYPE TOPICWORDDISTRIBUTION_T AS TABLE(
    "TOPICID" INTEGER,
    "WORDID" INTEGER,
    "PROBABILITY" DOUBLE
);
DROP TYPE DICTIONARY_T;
CREATE TYPE DICTIONARY_T AS TABLE(
    "WORDID" INTEGER,
    "TEXT" VARCHAR (100)
);
DROP TYPE GENERALINFO_T;
CREATE TYPE GENERALINFO_T AS TABLE(
    "NAME" VARCHAR (50),
    "INT_ARGS" INTEGER,
    "DOUBLE_ARGS" DOUBLE,
    "STRING_ARGS" VARCHAR (100)
);
DROP TYPE DOCTOPICDISTRIBUTION_T;
CREATE TYPE DOCTOPICDISTRIBUTION_T AS TABLE(
    "DOCUMENTID" INTEGER,
    "TOPICID" INTEGER,
    "PROBABILITY" DOUBLE
);
DROP TABLE PDATA_TBL;
CREATE COLUMN TABLE PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR (256), "PARAMETER_TYPE" VARCHAR (7));
INSERT INTO PDATA_TBL VALUES (1,'DM_PAL','DATA_T','IN');
INSERT INTO PDATA_TBL VALUES (2,'DM_PAL','PARAMETERS_T','IN');
INSERT INTO PDATA_TBL VALUES (3,'DM_PAL','TOPICWORDDISTRIBUTION_T','IN');
INSERT INTO PDATA_TBL VALUES (4,'DM_PAL','DICTIONARY_T','IN');
INSERT INTO PDATA_TBL VALUES (5,'DM_PAL','GENERALINFO_T','IN');
INSERT INTO PDATA_TBL VALUES (6,'DM_PAL','DOCTOPICDISTRIBUTION_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_LDAINFERENCE');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','LDAINFERENCE','DM_PAL','PAL_LDAINFERENCE',PDATA_TBL);
DROP TABLE DATA_TBL;
CREATE COLUMN TABLE DATA_TBL LIKE DATA_T;
INSERT INTO DATA_TBL VALUES (10, 'toy toy spoon bowl cpu');
DROP TABLE PARAMETERS_TBL;
CREATE COLUMN TABLE PARAMETERS_TBL LIKE PARAMETERS_T;
DROP TABLE DOCTOPICDIST_TBL;
CREATE COLUMN TABLE DOCTOPICDIST_TBL LIKE DOCTOPICDISTRIBUTION_T;
CALL DM_PAL.PAL_LDAINFERENCE(DATA_TBL, PARAMETERS_TBL, TOPICWORDDIST_TBL,
DICTIONARY_TBL, GENERALINFO_TBL, DOCTOPICDIST_TBL) with overview;
SELECT * FROM DOCTOPICDIST_TBL;

```

Expected Result

DOCTOPICDIST_TBL:

	DOCUMENT...	TOPICID	PROBABILITY
1	10	0	0.23913043478260873
2	10	1	0.02173913043478261
3	10	2	0.02173913043478261
4	10	3	0.02173913043478261
5	10	4	0.02173913043478261
6	10	5	0.673913043478261

3.1.11 Self-Organizing Maps

Self-organizing feature maps (SOMs) are one of the most popular neural network methods for cluster analysis. They are sometimes referred to as Kohonen self-organizing feature maps, after their creator, Teuvo Kohonen, or as topologically ordered maps. SOMs aim to represent all points in a high-dimensional source space by points in a low-dimensional (usually 2-D or 3-D) target space, such that the distance and proximity relationships are preserved as much as possible. This makes SOMs useful for visualizing low-dimensional views of high-dimensional data, akin to multidimensional scaling.

SOMs can also be viewed as a constrained version of k-means clustering, in which the cluster centers tend to lie in low-dimensional manifold in the feature or attribute space. The learning process mainly includes three steps:

1. Initialize the weighted vectors in each unit.
2. Select the Best Matching Unit (BMU) for every point and update the weighted vectors of BMU and its neighbors.
3. Repeat Step 2 until convergence or the maximum iterations are reached.

An important variant is batch SOM, which updates the weighted vectors only at the end of every learning epoch. It requires that the whole set of training data is present, and is independent on the order of input vectors.

The SOM approach has many applications such as virtualization, web document clustering, and speech recognition.

Prerequisites

- The first column of the input data is an ID column and the other columns are of integer or double data type.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

SELFORGMAP

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'SELFORGMAP',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Map OUTPUT table type>	OUT
4	<schema_name>	<Assign OUTPUT table type>	OUT
5	<schema_name>	<Cluster Model OUTPUT table type>	OUT (optional)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <map output table>, <assign output table>, <cluster model output model>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer or double	Attribute data	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
MAX_ITERATION	Integer	1000 plus 500 times the number of neurons in the lattice	Maximum number of iterations. Note that the training might not converge if this value is too small, for example, less than 1000.
CONVERGENCE_CRITERION	Double	1.0e-6	If the largest difference of the successive maps is less than this value, the calculation is regarded as convergence, and SOM is completed consequently.
NORMALIZATION	Integer	0	Normalization type: <ul style="list-style-type: none"> • 0: No • 1: Transform to new range (0.0, 1.0) • 2: Z-score normalization
THREAD_NUMBER	Integer	1	Number of threads.
RANDOM_SEED	Integer	-1	<ul style="list-style-type: none"> • -1: Random • 0: Sets every weight to zero • Other value: Uses this value as seed
HEIGHT_OF_MAP	Integer	10	Indicates the height of the map.
WIDTH_OF_MAP	Integer	10	Indicates the width of the map.
KERNEL_FUNCTION	Integer	1	Represents the neighborhood kernel function. <ul style="list-style-type: none"> • 1: Gaussian • 2: Bubble/Flat
ALPHA	Double	0.5	Specifies the learning rate.
LEARNING_RATE	Integer	1	Indicates the decay function for learning rate. <ul style="list-style-type: none"> • 1: Exponential • 2: Linear

Name	Data Type	Default Value	Description
SHAPE_OF_GRID	Integer	2	<p>Indicates the shape of the grid.</p> <ul style="list-style-type: none"> • 1: Rectangle • 2: Hexagon
RADIUS	Double	The bigger value of HEIGHT_OF_MAP and WIDTH_OF_MAP	Specifies the scan radius.
BATCH_SOM	Integer	0	<p>Indicates whether batch SOM is carried out.</p> <ul style="list-style-type: none"> • 0: Classical SOM • 1: Batch SOM <p>For batch SOM, KERNEL_FUNCTION is always Gaussian, and the LEARNING_RATE factors take no effect.</p>

i Note

The SOMs algorithm has been updated since SAP HANA SPS 12, so the algorithm results may be different from the results in the previous versions. The default value (1: Gaussian) for the KERNEL_FUNCTION parameter is not available in SAP HANA SPS 11 or lower, which only supports the bubble neighborhood function.

Output Tables

Table	Column	Column Data Type	Description	Constraint
SOM Map	1st column	Integer	Unit cell ID.	This must be the first column.
	Other columns except the last one	Double	Weight vectors used to simulate the original tuples.	
	Last column	Integer	Number of original tuples that every unit cell contains.	
SOM Assign	1st column	Integer, varchar, or nvarchar	ID of original tuples	This must be the first column.

Table	Column	Column Data Type	Description	Constraint
	2nd column	Integer	ID of the unit cells	
Cluster Model (optional)	1st column	Integer	Cluster model ID	This must be the first column.
	2nd column	CLOB, varchar, or nvarchar	Cluster model saved as JSON string	The table must be a column table. The minimum length of each unit (row) is 5000.

The SOM Assign output table can also use the following format:

Table	Column	Column Data Type	Description
SOM Assign	1st column	Integer, bigint, varchar, or nvarchar	ID of original tuples
	2nd column	Integer	Best match unit (BMU)
	3rd column	Double	The distance between the tuple and its BMU
	4th column	Integer	Second BMU
	5th column	Integer	Indicates whether the BMU and the second BMU are adjacent. <ul style="list-style-type: none"> • 0: Not adjacent • 1: Adjacent

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1: Classic SOM

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SOM_DATA_T;
CREATE TYPE PAL_SOM_DATA_T AS TABLE(
    "TRANS_ID" INTEGER,
    "V000" DOUBLE,
    "V001" DOUBLE
);
DROP TYPE PAL_SOM_MAP_T;
CREATE TYPE PAL_SOM_MAP_T AS TABLE(
    "CELL_ID" INTEGER,
    "WEIGHT000" DOUBLE,

```

```

        "WEIGHT001" DOUBLE,
        "NUMS_TUPLE" INTEGER
    );
DROP TYPE PAL_SOM_RESASSIGN_T;
CREATE TYPE PAL_SOM_RESASSIGN_T AS TABLE(
    "TRANS_ID" INTEGER,
    "BMU_ID" INTEGER,
    "DISTANCE" DOUBLE,
    "SECOND_BUM" INTEGER,
    "IS_ADJACENT" INTEGER
);
DROP TYPE PAL_SOM_MODEL_T;
CREATE TYPE PAL_SOM_MODEL_T AS TABLE(
    "JID" INTEGER,
    "JSMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_SOM_PDATA_TBL;
CREATE COLUMN TABLE PAL_SOM_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_SOM_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_SOM_DATA_T', 'IN');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_SOM_MAP_T', 'OUT');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_SOM_RESASSIGN_T', 'OUT');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_SOM_MODEL_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_SELF_ORG_MAP_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'SELFORGMAP', 'DM_PAL',
'PAL_SELF_ORG_MAP_PROC', PAL_SOM_PDATA_TBL);
DROP TABLE PAL_SOM_DATA_TBL;
CREATE COLUMN TABLE PAL_SOM_DATA_TBL LIKE PAL_SOM_DATA_T;
INSERT INTO PAL_SOM_DATA_TBL VALUES (0, 0.1, 0.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (1, 0.22, 0.25);
INSERT INTO PAL_SOM_DATA_TBL VALUES (2, 0.3, 0.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (3, 0.4, 0.5);
INSERT INTO PAL_SOM_DATA_TBL VALUES (4, 0.5, 1.0);
INSERT INTO PAL_SOM_DATA_TBL VALUES (5, 1.1, 15.1);
INSERT INTO PAL_SOM_DATA_TBL VALUES (6, 2.2, 11.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (7, 1.3, 15.3);
INSERT INTO PAL_SOM_DATA_TBL VALUES (8, 1.4, 15.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (9, 3.5, 15.9);
INSERT INTO PAL_SOM_DATA_TBL VALUES (10, 13.1, 1.1);
INSERT INTO PAL_SOM_DATA_TBL VALUES (11, 16.2, 1.5);
INSERT INTO PAL_SOM_DATA_TBL VALUES (12, 16.3, 1.3);
INSERT INTO PAL_SOM_DATA_TBL VALUES (13, 12.4, 2.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (14, 16.9, 1.9);
INSERT INTO PAL_SOM_DATA_TBL VALUES (15, 49.0, 40.1);
INSERT INTO PAL_SOM_DATA_TBL VALUES (16, 50.1, 50.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (17, 50.2, 48.3);
INSERT INTO PAL_SOM_DATA_TBL VALUES (18, 55.3, 50.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (19, 50.4, 56.5);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);

```

```

INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 4000, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('HEIGHT_OF_MAP', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('WIDTH_OF_MAP', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NORMALIZATION', 0, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('RANDOM_SEED', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SHAPE_OF_GRID', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('KERNEL_FUNCTION', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LEARNING_RATE', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CONVERGENCE_CRITERION', null, 1.0e-6,
null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BATCH_SOM', 0, null, null);
DROP TABLE PAL_SOM_MAP_TBL;
CREATE COLUMN TABLE PAL_SOM_MAP_TBL LIKE PAL_SOM_MAP_T;
DROP TABLE PAL_SOM_RESASSIGN_TBL;
CREATE COLUMN TABLE PAL_SOM_RESASSIGN_TBL LIKE PAL_SOM_RESASSIGN_T;
DROP TABLE PAL_SOM_MODEL_TBL;
CREATE COLUMN TABLE PAL_SOM_MODEL_TBL LIKE PAL_SOM_MODEL_T;
CALL "DM_PAL".PAL_SELF_ORG_MAP PROC(PAL_SOM_DATA_TBL, #PAL_CONTROL_TBL,
PAL_SOM_MAP_TBL, PAL_SOM_RESASSIGN_TBL, PAL_SOM_MODEL_TBL) with OVERVIEW;
SELECT * FROM PAL_SOM_MAP_TBL;
SELECT * FROM PAL_SOM_RESASSIGN_TBL;
SELECT * FROM PAL_SOM_MODEL_TBL;

```

Expected Result

PAL_SOM_MAP_TBL:

	CELL_ID	WEIGHT000	WEIGHT001	NUMS_TUPLE
1	0	52.837688436447	53.4653266227774	2
2	1	50.15025126124673	49.24522613704447	2
3	2	18.597606740665...	27.174589674066...	0
4	3	1.2676711221847...	15.267671122184...	3
5	4	49.00002113918622	40.1000986237828	1
6	5	33.43099411575232	34.4504914886213	0
7	6	3.499980745400457	15.89997204453496	1
8	7	2.2000010370442...	11.200050825324...	1
9	8	24.814991938100...	11.738392237008...	0
10	9	20.69625298338181	8.31512780833647	0
11	10	9.817004542651661	6.453149527206734	0
12	11	1.1444060344046...	4.24620510253602	0
13	12	16.4690145112879	1.5680111780752...	3
14	13	12.748241216635...	1.7532663311415...	2
15	14	3.8868516028923...	0.8463564826804...	0
16	15	0.3059695792263...	0.4737270589467...	5

PAL_SOM_RESASSIGN_TBL:

	TRANS_ID	BMU_ID	DISTANCE	SECOND_BUM	IS_ADJACENT
1	0	15	0.3425638194064086	14	1
2	1	15	0.239675959281097	14	1
3	2	15	0.07396833847719...	14	1
4	3	15	0.09763189777148...	14	1
5	4	15	0.5609019635109566	11	1
6	5	3	0.2371227750115362	6	1
7	6	7	0.00005083590337...	3	1
8	7	3	0.04571993746308...	6	1
9	8	3	0.1871412937003927	6	1
10	9	6	0.00003394477322...	3	1
11	10	13	0.741950902066637	12	1
12	11	12	0.27747851741467...	13	1
13	12	12	0.3168531151798187	13	1
14	13	13	0.7345314039567026	12	1
15	14	12	0.5440267175088088	13	1
16	15	4	0.00010086384747...	1	1
17	16	1	0.9560953501821456	0	1
18	17	1	0.9465344088619049	0	1
19	18	0	3.93181962204711	1	1
20	19	0	3.892501434760311	1	1

PAL_SOM_MODEL_TBL:

	JID	JSONMODEL
1	0	{"Algorithm": "SOM", "Cluster": [{"CellID": 0, "CellWeight": [52.8376884364470, 53.46532662277...]

Example 2: Batch SOM

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SOM_DATA_T;
CREATE TYPE PAL_SOM_DATA_T AS TABLE(
    "TRANS_ID" INTEGER,
    "V000" DOUBLE,
    "V001" DOUBLE
);
DROP TYPE PAL_SOM_MAP_T;
CREATE TYPE PAL_SOM_MAP_T AS TABLE(
    "CELL_ID" INTEGER,
    "WEIGHT000" DOUBLE,
    "WEIGHT001" DOUBLE,
    "NUMS_TUPLE" INTEGER
);
DROP TYPE PAL_SOM_RESASSIGN_T;
CREATE TYPE PAL_SOM_RESASSIGN_T AS TABLE(
    "TRANS_ID" INTEGER,
    "BMU_ID" INTEGER,
    "DISTANCE" DOUBLE,
    "SECOND_BUM" INTEGER,
    "IS_ADJACENT" INTEGER
);
DROP TYPE PAL_SOM_MODEL_T;

```

```

CREATE TYPE PAL_SOM_MODEL_T AS TABLE(
    "JID" INTEGER,
    "JSMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_SOM_PDATA_TBL;
CREATE COLUMN TABLE PAL_SOM_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_SOM_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_SOM_DATA_T', 'IN');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_SOM_MAP_T', 'OUT');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_SOM_RESASSIGN_T', 'OUT');
INSERT INTO PAL_SOM_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_SOM_MODEL_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_SELF_ORG_MAP_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'SELFORGMAP', 'DM_PAL',
'PAL_SELF_ORG_MAP_PROC', PAL_SOM_PDATA_TBL);
DROP TABLE PAL_SOM_DATA_TBL;
CREATE COLUMN TABLE PAL_SOM_DATA_TBL LIKE PAL_SOM_DATA_T;
INSERT INTO PAL_SOM_DATA_TBL VALUES (0, 0.1, 0.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (1, 0.22, 0.25);
INSERT INTO PAL_SOM_DATA_TBL VALUES (2, 0.3, 0.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (3, 0.4, 0.5);
INSERT INTO PAL_SOM_DATA_TBL VALUES (4, 0.5, 1.0);
INSERT INTO PAL_SOM_DATA_TBL VALUES (5, 1.1, 15.1);
INSERT INTO PAL_SOM_DATA_TBL VALUES (6, 2.2, 11.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (7, 1.3, 15.3);
INSERT INTO PAL_SOM_DATA_TBL VALUES (8, 1.4, 15.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (9, 3.5, 15.9);
INSERT INTO PAL_SOM_DATA_TBL VALUES (10, 13.1, 1.1);
INSERT INTO PAL_SOM_DATA_TBL VALUES (11, 16.2, 1.5);
INSERT INTO PAL_SOM_DATA_TBL VALUES (12, 16.3, 1.3);
INSERT INTO PAL_SOM_DATA_TBL VALUES (13, 12.4, 2.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (14, 16.9, 1.9);
INSERT INTO PAL_SOM_DATA_TBL VALUES (15, 49.0, 40.1);
INSERT INTO PAL_SOM_DATA_TBL VALUES (16, 50.1, 50.2);
INSERT INTO PAL_SOM_DATA_TBL VALUES (17, 50.2, 48.3);
INSERT INTO PAL_SOM_DATA_TBL VALUES (18, 55.3, 50.4);
INSERT INTO PAL_SOM_DATA_TBL VALUES (19, 50.4, 56.5);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 4000, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('HEIGHT_OF_MAP', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('WIDTH_OF_MAP', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NORMALIZATION', 0, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('RANDOM_SEED', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SHAPE_OF_GRID', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CONVERGENCE_CRITERION', null, 1.0e-6, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BATCH_SOM', 1, null, null);
DROP TABLE PAL_SOM_MAP_TBL;
CREATE COLUMN TABLE PAL_SOM_MAP_TBL LIKE PAL_SOM_MAP_T;
DROP TABLE PAL_SOM_RESASSIGN_TBL;

```

```

CREATE COLUMN TABLE PAL_SOM_RESASSIGN_TBL LIKE PAL_SOM_RESASSIGN_T;
DROP TABLE PAL_SOM_MODEL_TBL;
CREATE COLUMN TABLE PAL_SOM_MODEL_TBL LIKE PAL_SOM_MODEL_T;
CALL "DM_PAL".PAL_SELF_ORG_MAP_PROC(PAL_SOM_DATA_TBL, #PAL_CONTROL_TBL,
PAL_SOM_MAP_TBL, PAL_SOM_RESASSIGN_TBL, PAL_SOM_MODEL_TBL) with OVERVIEW;
SELECT * FROM PAL_SOM_MAP_TBL;
SELECT * FROM PAL_SOM_RESASSIGN_TBL;
SELECT * FROM PAL_SOM_MODEL_TBL;

```

Expected Result

PAL_SOM_MAP_TBL:

	CELL_ID	WEIGHT000	WEIGHT001	NUMS_TUPLE
1	0	52.84999945658175	53.44999913928276	2
2	1	50.1500005329579	49.250000813645634	2
3	2	20.746431803615...	28.80495128030353	0
4	3	1.2666666710832...	15.266666661878148	3
5	4	49.00000004184145	40.10000018828649	1
6	5	26.86732052090711	28.548872848086965	0
7	6	3.4999997141985...	15.899999060120619	1
8	7	2.2000002460521...	11.200000982976063	1
9	8	24.599999999837...	11.199999999870379	0
10	9	20.64999999957...	8.05000000001944	0
11	10	9.666666666746288	6.466666666574443	0
12	11	1.031428571466213	4.207142857134969	0
13	12	16.466666175365...	1.566666690901242	3
14	13	12.750001105426...	1.7499999454722142	2
15	14	3.86000000000000...	0.8357142857142857	0
16	15	0.304	0.47000000000000...	5

PAL_SOM_RESASSIGN_TBL:

	TRANS_ID	BMU_ID	DISTANCE	SECOND_BUM	IS_ADJACENT
1	0	15	0.3384021276528858	14	1
2	1	15	0.23549097647256043	14	1
3	2	15	0.07011419257183242	14	1
4	3	15	0.10057832768544128	14	1
5	4	15	0.5650805252351209	11	1
6	5	3	0.23570226013252726	6	1
7	6	7	0.000001013303308...	3	1
8	7	3	0.04714045234209299	6	1
9	8	3	0.1885618085794018	6	1
10	9	6	0.000000982372509...	3	1
11	10	13	0.7382405809194243	12	1
12	11	12	0.2748732376203747	13	1
13	12	12	0.31446579789741474	13	1
14	13	13	0.7382417251051312	12	1
15	14	12	0.5467076902028115	13	1
16	15	4	0.000000192879515...	1	1
17	16	1	0.9513140950128068	0	1
18	17	1	0.9513156640315913	0	1
19	18	0	3.9121601976881384	1	1
20	19	0	3.9121608591195196	1	1

PAL_SOM_MODEL_TBL:

	JID	JSONMODEL
1	0	{"Algorithm": "SOM", "Cluster": [{"CellID": 0, "CellWeight": [52.84999945658175, 53.4499991392...]

3.1.12 Slight Silhouette

Silhouette refers to a method used to validate the cluster of data.

The complex of Silhouette is $O(N^2)$, where N represents the number of records. When N is very large, silhouette will cost much time.

In consideration of the efficient, PAL provides a lite version of Silhouette called Slight Silhouette. Suppose you have N records. For every record i, the following is defined:

$$S(i) = \frac{B(i) - A(i)}{\max(B(i), A(i))}$$

Where $A(i)$ represents the distance between i and the center of the cluster it belongs to, and $B(i)$ is the minimum distance between i and other cluster centers. Finally the below formula is derived:

$$S = \frac{1}{N} \sum_{i=1}^N S(i)$$

It is clear that $-1 \leq S \leq 1$. -1 indicates poor clustering result, and 1 stands for good result.

For attributes of category type, you can pre-process the input data using the method described in K-means.

Prerequisites

The input data does not contain null value.

SLIGHTSILHOUETTE

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'SLIGHTSILHOUETTE',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	Last column	Integer, bigint, varchar, or nvarchar	Class label	This must be the last column.

Table	Column	Column Data Type	Description	Constraint
	Other columns	Integer, double, varchar, or nvarchar	Attribute data	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
DISTANCE_LEVEL	Integer	2	<p>Computes the distance between the item and the cluster center.</p> <ul style="list-style-type: none"> • 1: Manhattan distance • 2: Euclidean distance • 3: Minkowski distance • 4: Chebyshev distance 	
MINKOWSKI_POWER	Double	3	When you use the Minkowski distance, this parameter controls the value of power.	Only valid when DISTANCE_LEVEL is 3.
NORMALIZATION	Integer	0	<p>Normalization type:</p> <ul style="list-style-type: none"> • 0: No • 1: Yes. For each point X (x_1, x_2, \dots, x_n), the normalized value will be $X' = (x_1/S, x_2/S, \dots, x_n/S)$, where $S = \sqrt{ x_1 ^2 + x_2 ^2 + \dots + x_n ^2}$. • 2: For each column C, get the min and max value of C, and then $C'[i] = (C[i] - \min)/(max - \min)$. 	
THREAD_NUMBER	Integer	1	Number of threads.	

Name	Data Type	Default Value	Description	Dependency
CATEGORY_COL	Integer	No default value	Indicates whether the column is category variable. By default, 'string' is category variable, and 'integer' or 'double' is continuous variable.	
CATEGORY_WEIGHTS	Double	0.707	Represents the weight of category attributes (γ).	

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Double	Validation value

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SLIGHT_SIL_T;
CREATE TYPE PAL_SLIGHT_SIL_T AS TABLE(
    "V000" DOUBLE,
    "V001" VARCHAR(5),
    "V002" DOUBLE,
    "CLUSTER" INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_SLIGHT_SIL_RESULT_T;
CREATE TYPE PAL_SLIGHT_SIL_RESULT_T AS TABLE(
    "SILHOUETTE" DOUBLE
);
DROP TABLE PAL_SLIGHT_SIL_PDATA_TBL;
CREATE COLUMN TABLE PAL_SLIGHT_SIL_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_SLIGHT_SIL_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_SLIGHT_SIL_T', 'IN');
INSERT INTO PAL_SLIGHT_SIL_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_SLIGHT_SIL_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_SLIGHT_SIL_RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_SLIGHT_SIL_PROC');

```

```

CALL "SYS".AFLLANG_WRAPPER PROCEDURE_CREATE('AFLPAL', 'SLIGHTSILHOUETTE',
'DM_PAL', 'PAL_SLIGHT_SIL_PROC', PAL_SLIGHT_SIL_PDATA_TBL);
DROP TABLE PAL_SLIGHT_SIL_TBL;
CREATE COLUMN TABLE PAL_SLIGHT_SIL_TBL LIKE PAL_SLIGHT_SIL_T;
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (0.5, 'A', 0.5, 0);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (1.5, 'A', 0.5, 0);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (1.5, 'A', 1.5, 0);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (0.5, 'A', 1.5, 0);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (1.1, 'B', 1.2, 0);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (0.5, 'B', 15.5, 1);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (1.5, 'B', 15.5, 1);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (1.5, 'B', 16.5, 1);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (0.5, 'B', 16.5, 1);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (1.2, 'C', 16.1, 1);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (15.5, 'C', 15.5, 2);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (16.5, 'C', 15.5, 2);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (16.5, 'C', 16.5, 2);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (15.5, 'C', 16.5, 2);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (15.6, 'D', 16.2, 2);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (15.5, 'D', 0.5, 3);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (16.5, 'D', 0.5, 3);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (16.5, 'D', 1.5, 3);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (15.5, 'D', 1.5, 3);
INSERT INTO PAL_SLIGHT_SIL_TBL VALUES (15.7, 'A', 1.6, 3);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NORMALIZATION', 0, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('DISTANCE_LEVEL', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_WEIGHTS', null, 0.7, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_COL', 1, null, null);
DROP TABLE PAL_SLIGHT_SIL_RESULT_TBL;
CREATE COLUMN TABLE PAL_SLIGHT_SIL_RESULT_TBL LIKE PAL_SLIGHT_SIL_RESULT_T;
CALL "DM_PAL".PAL_SLIGHT_SIL_PROC(PAL_SLIGHT_SIL_TBL, #PAL_CONTROL_TBL,
PAL_SLIGHT_SIL_RESULT_TBL) with OVERVIEW;
SELECT * FROM PAL_SLIGHT_SIL_RESULT_TBL;

```

Expected Result

SILHOUETTE
0.9385943815746565

3.1.13 Incremental Clustering on SAP HANA Smart Data Streaming

For information on incremental clustering on SAP HANA Smart Data Streaming, see the example of DenStream clustering in the *SAP HANA Smart Data Streaming: Developer Guide*.

i Note

SAP HANA Smart Data Streaming is only supported on Intel-based hardware platforms.

3.2 Classification Algorithms

This section describes the classification algorithms that are provided by the Predictive Analysis Library.

3.2.1 Area Under Curve (AUC)

Area under curve (AUC) is a traditional method to evaluate the performance of classification algorithms. Basically, it can evaluate the binary classifier, but can be extended to multiple-class condition easily.

In an area under curve algorithm, curve is the receiver operating characteristic (ROC) curve. The curve can be obtained by plotting the true positive rate (TPR) against the false positive rate (FPR) at several threshold. The calculation formulas are listed below.

$$TPR = \frac{TP}{TP+FN}$$

$$FPR = \frac{FP}{FP+TN}$$

Where:

TP: true positive

FN: false negative

FP: false positive

TN: true negative

After plotting the ROC curve, you can calculate the area under the curve by using numerical integral algorithms such as Simpson's rule. The value of AUC ranges from 0.5 to 1. If the AUC equals to 1, the classifier is expected to have perfect performance.

Prerequisite

No missing or null data in the inputs.

AUC

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'AUC', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Input data table type>	IN
2	<schema_name>	<Parameter table type>	IN
3	<schema_name>	<AUC table type>	OUT
4	<schema_name>	<ROC table type>	OUT

Or

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Input data table type>	IN
2	<schema_name>	<Input data table type>	IN
3	<schema_name>	<Parameter table type>	IN
4	<schema_name>	<AUC table type>	OUT
5	<schema_name>	<ROC table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input data tables>, <parameter table>, <AUC table>, <ROC table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table(s)

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer	Original label	The original label should be 0 or 1.
	3rd column	Double	The probability belonging to positive	The probability should be the probability of a sample belonging to 1.

Or

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer, varchar, or nvarchar	Original label	The original label of the samples.
Data2	1st column	Integer, varchar, or nvarchar	ID	The type must be the same as the first column type of the previous table.
	2nd column	Integer, varchar, or nvarchar	All of the possible predictive labels	The type must be the same as the second column type of the previous table.
	3rd column	Double	The probability belonging to each possible label	For each sample data, the sum of the probabilities belonging to each possible label should be 1.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads.

Output Tables

Table	Column	Column Data Type	Description
AUC	1st column	Integer, varchar, or nvarchar	Value name: AUC
	2nd column	Double	AUC value

Table	Column	Column Data Type	Description
ROC	1st column	Integer	ID
	2nd column	Double	False positive rate for various threshold setting
	3rd column	Double	True positive rate for various threshold setting

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_AUC_DATA_T;
CREATE TYPE PAL_AUC_DATA_T AS TABLE (
ID INTEGER,
ORIGINAL INTEGER,
PREDICT DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
NAME VARCHAR(50),
INTARGS INTEGER,
DOUBLEARGS DOUBLE,
STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_AUC_T;
CREATE TYPE PAL_AUC_T AS TABLE(
NAME VARCHAR(100),
AUC DOUBLE
);
DROP TYPE PAL_ROC_T;
CREATE TYPE PAL_ROC_T AS TABLE(
ID INTEGER,
FPR DOUBLE,
TPR DOUBLE
);
DROP TABLE PAL_AUC_PDATA_TBL;
CREATE COLUMN TABLE PAL_AUC_PDATA_TBL (
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_AUC_PDATA_TBL VALUES (1,'DM_PAL','PAL_AUC_DATA_T', 'IN');
INSERT INTO PAL_AUC_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_AUC_PDATA_TBL VALUES (3,'DM_PAL','PAL_AUC_T', 'OUT');
INSERT INTO PAL_AUC_PDATA_TBL VALUES (4,'DM_PAL','PAL_ROC_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_AUC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'AUC', 'DM_PAL', 'PAL_AUC',
PAL_AUC_PDATA_TBL );
DROP TABLE PAL_AUC_DATA_TBL;
CREATE COLUMN TABLE PAL_AUC_DATA_TBL LIKE PAL_AUC_DATA_T;

```

```

INSERT INTO PAL_AUC_DATA_TBL VALUES(1,0,0.07);
INSERT INTO PAL_AUC_DATA_TBL VALUES(2,0,0.01);
INSERT INTO PAL_AUC_DATA_TBL VALUES(3,0,0.85);
INSERT INTO PAL_AUC_DATA_TBL VALUES(4,0,0.3);
INSERT INTO PAL_AUC_DATA_TBL VALUES(5,0,0.5);
INSERT INTO PAL_AUC_DATA_TBL VALUES(6,1,0.5);
INSERT INTO PAL_AUC_DATA_TBL VALUES(7,1,0.2);
INSERT INTO PAL_AUC_DATA_TBL VALUES(8,1,0.8);
INSERT INTO PAL_AUC_DATA_TBL VALUES(9,1,0.2);
INSERT INTO PAL_AUC_DATA_TBL VALUES(10,1,0.95);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',1,NULL,NULL);
DROP TABLE PAL_AUC_TBL;
CREATE COLUMN TABLE PAL_AUC_TBL LIKE PAL_AUC_T;
DROP TABLE PAL_ROC_TBL;
CREATE COLUMN TABLE PAL_ROC_TBL LIKE PAL_ROC_T;
CALL DM_PAL.PAL_AUC(PAL_AUC_DATA_TBL, #PAL_CONTROL_TBL, PAL_AUC_TBL,
PAL_ROC_TBL) with OVERVIEW;
SELECT * FROM PAL_AUC_TBL;
SELECT * FROM PAL_ROC_TBL;

```

Expected Result

PAL_AUC_TBL:

	NAME	AUC
1	AUC	0.66

PAL_ROC_TBL:

	ID	FPR	TPR
1	0	1	1
2	1	0.8	1
3	2	0.6	1
4	3	0.6	0.6
5	4	0.4	0.6
6	5	0.2	0.4
7	6	0.2	0.2
8	7	0	0.2
9	8	0	0

Example 2

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_AUC_DATA_T;
CREATE TYPE PAL_AUC_DATA_T AS TABLE (
ID INTEGER,
ORIGINAL INTEGER
);
DROP TYPE PAL_AUC_DATA2_T;
CREATE TYPE PAL_AUC_DATA2_T AS TABLE (
ID INTEGER,
PREDICT INTEGER,
PROB DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
NAME VARCHAR(50),
INTARGS INTEGER,

```

```

DOUBLEARGS DOUBLE,
STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_AUC_T;
CREATE TYPE PAL_AUC_T AS TABLE(
NAME VARCHAR(100),
AUC DOUBLE
);
DROP TYPE PAL_ROC_T;
CREATE TYPE PAL_ROC_T AS TABLE(
ID INTEGER,
FPR DOUBLE,
TPR DOUBLE
);
DROP TABLE PAL_AUC_PDATA_TBL;
CREATE COLUMN TABLE PAL_AUC_PDATA_TBL (
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_AUC_PDATA_TBL VALUES (1,'DM_PAL','PAL_AUC_DATA_T', 'IN');
INSERT INTO PAL_AUC_PDATA_TBL VALUES (2,'DM_PAL','PAL_AUC_DATA2_T', 'IN');
INSERT INTO PAL_AUC_PDATA_TBL VALUES (3,'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_AUC_PDATA_TBL VALUES (4,'DM_PAL','PAL_AUC_T', 'OUT');
INSERT INTO PAL_AUC_PDATA_TBL VALUES (5,'DM_PAL','PAL_ROC_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_AUC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'AUC', 'DM_PAL', 'PAL_AUC',
PAL_AUC_PDATA_TBL);
DROP TABLE PAL_AUC_DATA_TBL;
CREATE COLUMN TABLE PAL_AUC_DATA_TBL LIKE PAL_AUC_DATA_T;
INSERT INTO PAL_AUC_DATA_TBL VALUES(1,1);
INSERT INTO PAL_AUC_DATA_TBL VALUES(2,1);
INSERT INTO PAL_AUC_DATA_TBL VALUES(3,1);
INSERT INTO PAL_AUC_DATA_TBL VALUES(4,2);
INSERT INTO PAL_AUC_DATA_TBL VALUES(5,2);
INSERT INTO PAL_AUC_DATA_TBL VALUES(6,2);
INSERT INTO PAL_AUC_DATA_TBL VALUES(7,3);
INSERT INTO PAL_AUC_DATA_TBL VALUES(8,3);
INSERT INTO PAL_AUC_DATA_TBL VALUES(9,3);
INSERT INTO PAL_AUC_DATA_TBL VALUES(10,3);
DROP TABLE PAL_AUC_DATA2_TBL;
CREATE COLUMN TABLE PAL_AUC_DATA2_TBL LIKE PAL_AUC_DATA2_T;
INSERT INTO PAL_AUC_DATA2_TBL VALUES(1,1,0.9);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(1,2,0.05);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(1,3,0.05);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(2,1,0.8);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(2,2,0.05);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(2,3,0.15);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(3,1,0.8);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(3,2,0.1);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(3,3,0.1);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(4,1,0.1);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(4,2,0.8);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(4,3,0.1);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(5,1,0.2);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(5,2,0.7);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(5,3,0.1);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(6,1,0.05);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(6,2,0.9);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(6,3,0.05);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(7,1,0.1);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(7,2,0.1);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(7,3,0.8);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(8,1,0);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(8,2,0);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(8,3,1);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(9,1,0.2);

```

```

INSERT INTO PAL_AUC_DATA2_TBL VALUES(9,2,0.1);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(9,3,0.7);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(10,1,0.2);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(10,2,0.2);
INSERT INTO PAL_AUC_DATA2_TBL VALUES(10,3,0.6);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',1,NULL,NULL);
DROP TABLE PAL_AUC_TBL;
CREATE COLUMN TABLE PAL_AUC_TBL LIKE PAL_AUC_T;
DROP TABLE PAL_ROC_TBL;
CREATE COLUMN TABLE PAL_ROC_TBL LIKE PAL_ROC_T;
CALL DM_PAL.PAL_AUC(PAL_AUC_DATA_TBL,PAL_AUC_DATA2_TBL, #PAL_CONTROL_TBL,
PAL_AUC_TBL, PAL_ROC_TBL) with OVERVIEW;
SELECT * FROM PAL_AUC_TBL;
SELECT * FROM PAL_ROC_TBL;

```

Expected Result

PAL_AUC_TBL:

	NAME	AUC
1	AUC	1

PAL_ROC_TBL:

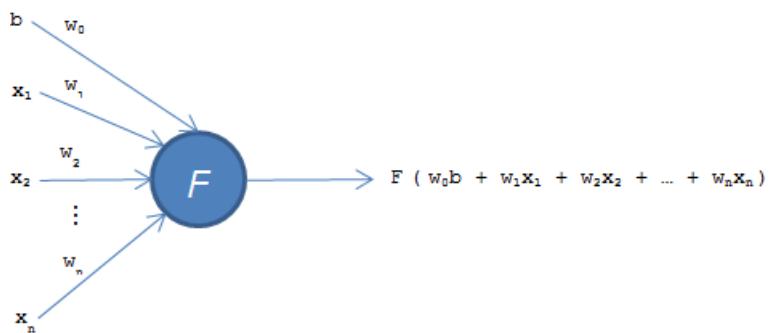
	ID	FPR	TPR
1	0	1	1
2	1	0.9	1
3	2	0.65	1
4	3	0.25	1
5	4	0.2	1
6	5	0	1
7	6	0	0.9
8	7	0	0.7
9	8	0	0.3
10	9	0	0.1
11	10	0	0

3.2.2 Back Propagation Neural Network

Neural network is a calculation model inspired by biological nervous system. The functionality of neural network is determined by its network structure and connection weights between neurons. Back propagation neural network (BPNN) is one of the very popular types for its training method called back propagation.

Single Neuron

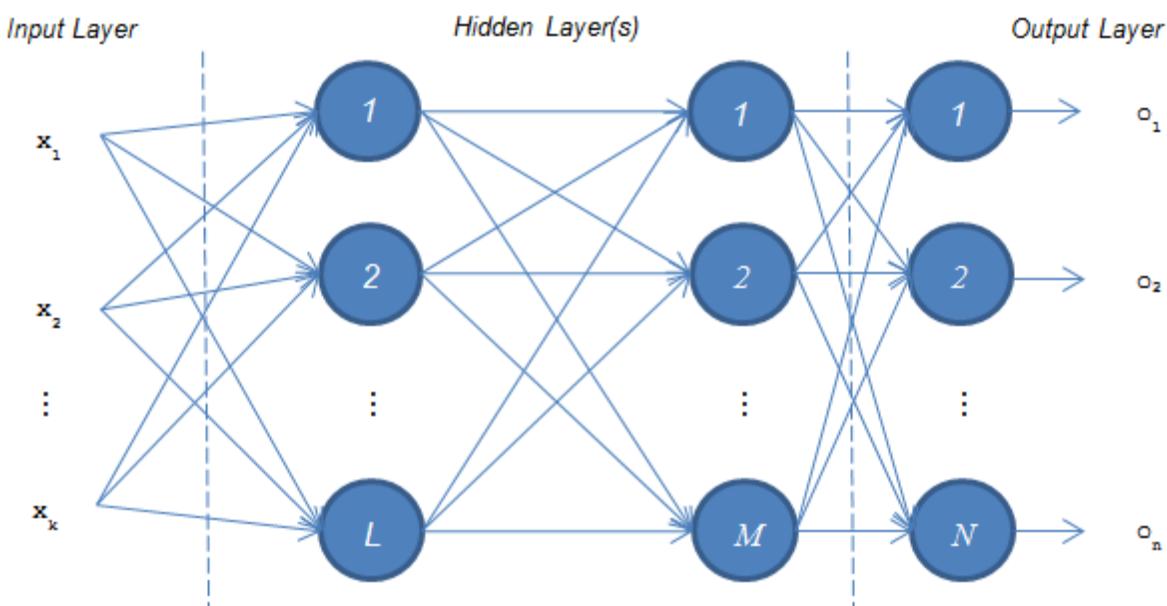
A neuron takes signals from outside and transforms them to a single value as output:



- The function F is called activate function.
- x_i ($i = 1, 2, \dots, n$) is the outside signal and b is a constant bias usually set to 1.
- w_i ($i = 0, 1, \dots, n$) is the weight value on each connection.

Neural Network Structure

A neural network consists of three parts: input layer, hidden layer(s), and output layer. Each layer owns several neurons and there are connections between layers. Signals are given in input layer and transmit through connections and layers. Finally output layer gives transformed signals.



Steps

BPNN performs three steps as follows:

1. Build the network structure and initialize the network.
2. Feed the training data repeatedly to train the network.
3. Make prediction with the trained network.

Training

During the training, BPNN adjusts the connection weights by comparing its output and expected target to make the output more accurate. The following training styles are available:

- Batch training: weights updating is based on the error of the entire package of training patterns. Thus, in one round the weights are updated once.
- Stochastic training: weights updating is based on the error of a single training pattern. Thus, in one round the weights are updated for each pattern.

Support for Categorical Attributes

If an attribute is of category type, it will be converted to a binary vector and then be used as numerical attributes. For example, in the below table, "Gender" is of category type.

Customer ID	Age	Income	Gender
T1	31	10,000	Female
T2	27	8,000	Male

Because "Gender" has two distinct values, it will be converted into a binary vector with two dimensions:

Customer ID	Age	Income	Gender_1	Gender_2
T1	31	10,000	0	1
T2	27	8,000	1	0

Prerequisites

For training data:

- The data are of integer, varchar, nvarchar, or double data type and do not contain null value. Otherwise the algorithm will issue errors.
- If it is for classification, then the last column is considered as the label column and is of integer, varchar, or nvarchar type.
- If it is for regression, then you should specify how many last columns are considered as target values, and they are of integer or double type.

For predicted data:

- The data are of integer, varchar, nvarchar, or double data type and does not contain null value. Otherwise the algorithm will issue errors.
- The first column is ID column and should be of integer type.
- The column order and column number of the predicted data are the same as the order and number used in model training.

CREATEBPNN

This function trains a BPNN model from input data.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CREATEBPNN',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Model OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>, <model output table>) WITH OVERVIEW;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Training data	Columns	Integer, double, varchar, or nvarchar	Attribute data
	Last column(s)	Integer, varchar, or nvarchar for classification; Integer or double for regression.	Label column for classification; Target value for regression.

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description
HIDDEN_LAYER_ACTIVE_FUNC	Integer	Active function code for the hidden layer.
OUTPUT_LAYER_ACTIVE_FUNC	Integer	Active function code for the output layer.
LEARNING_RATE	Double	Specifies the learning rate.
MOMENTUM_FACTOR	Double	Specifies the momentum factor.

Name	Data Type	Description
HIDDEN_LAYER_SIZE	Varchar	Specifies the size of each hidden layer in the format of "2, 3, 4". The value 0 will be ignored, for example, "2, 0, 3" is equal to "2, 3".

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
MAX_ITERATION	Integer	100	Maximum iterations.	
FUNCTIONALITY	Integer	0	Specifies the prediction type: <ul style="list-style-type: none">• 0: Classification• 1: Regression	
TARGET_COL-UMN_NUM	Integer	1	Specifies the number of target value columns for regression.	Ignored when FUNCTIONALITY is 0.
TRAINING_STYLE	Integer	1	Specifies the training style: <ul style="list-style-type: none">• 0: Batch• 1: Stochastic	
NORMALIZATION	Integer	0	Specifies the normalization type: <ul style="list-style-type: none">• 0: None• 1: Z-transform• 2: Scalar	
WEIGHT_INIT	Integer	0	Specifies the weight initial value: <ul style="list-style-type: none">• 0: All zeros• 1: Normal distribution• 2: Uniform distribution in range (0, 1)	
CATEGORY_COL	Integer	No default value	Indicates whether the column of integer type is category variable. By default, 'string' is category variable and 'integer' or 'double' is continuous variable.	
THREAD_NUMBER	Integer	1	Number of threads.	Ignored when TRAINING_STYLE is 1.

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Statistic result name
	2nd column	Double	Statistic result value
Model	1st column	Varchar or nvarchar	Model type
	2nd column	CLOB	Model saved as string

Active Function Code

- TANH = 1
- LINEAR = 2
- SIGMOID_ASYMMETRIC = 3
- SIGMOID_SYMMETRIC = 4
- GAUSSIAN_ASYMMETRIC = 5
- GAUSSIAN_SYMMETRIC = 6
- ELLIOT_ASYMMETRIC = 7
- ELLIOT_SYMMETRIC = 8
- SIN_ASYMMETRIC = 9
- SIN_SYMMETRIC = 10
- COS_ASYMMETRIC = 11
- COS_SYMMETRIC = 12

Examples

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Classification example:

```

SET SCHEMA DM_PAL;
--TRAINING PART
--CREATE TABLE TYPE
DROP TYPE PAL_TRAIN_NN_DATA_T;
CREATE TYPE PAL_TRAIN_NN_DATA_T AS TABLE(
    "V000" INTEGER,
    "V001" DOUBLE,
    "V002" VARCHAR(10),
    "V003" INTEGER,
    "LABEL" VARCHAR(2)
);
DROP TYPE PAL_TRAIN_NN_RESULT_T;
CREATE TYPE PAL_TRAIN_NN_RESULT_T AS TABLE(
    "NAME" VARCHAR(100),
    "VALUE" DOUBLE
);
DROP TYPE PAL_NN_MODEL_T;
CREATE TYPE PAL_NN_MODEL_T AS TABLE(
    "NAME" VARCHAR(100),
    "MODEL" CLOB
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "PARAMETERS" CLOB
);

```

```

    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
--CREATE PROCEDURE
DROP TABLE PAL_NN_PDATA_TBL;
CREATE COLUMN TABLE PAL_NN_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
DELETE FROM PAL_NN_PDATA_TBL;
INSERT INTO PAL_NN_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_TRAIN_NN_DATA_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_TRAIN_NN_RESULT_T',
'OUT');
INSERT INTO PAL_NN_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_NN_MODEL_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_NN_TRAIN');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CREATEBPNN', 'DM_PAL',
'PAL_NN_TRAIN', 'PAL_NN_PDATA_TBL');
--RUN
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('HIDDEN_LAYER_SIZE', NULL, NULL, '10, 10');
INSERT INTO #PAL_CONTROL_TBL VALUES ('HIDDEN_LAYER_ACTIVE_FUNC', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('OUTPUT_LAYER_ACTIVE_FUNC', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LEARNING_RATE', NULL, 0.001, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MOMENTUM_FACTOR', NULL, 0.00001, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FUNCTIONALITY', 0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TRAINING_STYLE', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_COL', 3, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 100, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NORMALIZATION', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('WEIGHT_INIT', 1, NULL, NULL);
DROP TABLE PAL_TRAIN_NN_RESULT_TBL;
CREATE COLUMN TABLE PAL_TRAIN_NN_RESULT_TBL LIKE PAL_TRAIN_NN_RESULT_T;
DROP TABLE PAL_CLASSIFICATION_NN_MODEL_TBL;
CREATE COLUMN TABLE PAL_CLASSIFICATION_NN_MODEL_TBL LIKE PAL_NN_MODEL_T;
DROP TABLE PAL_TRAIN_NN_DATA_TBL;
CREATE COLUMN TABLE PAL_TRAIN_NN_DATA_TBL LIKE PAL_TRAIN_NN_DATA_T;
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 1, 1.71, 'AC', 0, 'AA');
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 10, 1.78, 'CA', 5, 'AB');
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 17, 2.36, 'AA', 6, 'AA');
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 12, 3.15, 'AA', 2, 'C');
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 7, 1.05, 'CA', 3, 'AB');
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 6, 1.50, 'CA', 2, 'AB');
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 9, 1.97, 'CA', 6, 'C');
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 5, 1.26, 'AA', 1, 'AA');
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 12, 2.13, 'AC', 4, 'C');
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 18, 1.87, 'AC', 6, 'AA');
--CALL PROCEDURE
CALL DM_PAL.PAL_NN_TRAIN(PAL_TRAIN_NN_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_TRAIN_NN_RESULT_TBL, PAL_CLASSIFICATION_NN_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_TRAIN_NN_RESULT_TBL;
SELECT * FROM PAL_CLASSIFICATION_NN_MODEL_TBL;

```

Expected Results

Note: Your result may look slightly different from the following results.

PAL_TRAIN_NN_RESULT_TBL:

	NAME	VALUE
1	ERROR	0.3905914856941014

PAL_CLASSIFICATION_NN_MODEL_TBL:

	NAME	MODEL
1	JSON	{"CurrentVersion": "1.0", "DataDictionary": [{"dataType": "integer", "name": "V000", "optype": "c..."}]}

Regression example:

```
SET SCHEMA DM_PAL;
--TRAINING PART
--CREATE TABLE TYPE
DROP TYPE PAL_TRAIN_NN_DATA_T;
CREATE TYPE PAL_TRAIN_NN_DATA_T AS TABLE(
    "V000" INTEGER,
    "V001" DOUBLE,
    "V002" VARCHAR(10),
    "V003" INTEGER,
    "T001" DOUBLE,
    "T002" DOUBLE,
    "T003" DOUBLE
);
DROP TYPE PAL_TRAIN_NN_RESULT_T;
CREATE TYPE PAL_TRAIN_NN_RESULT_T AS TABLE(
    "NAME" VARCHAR(100),
    "VALUE" DOUBLE
);
DROP TYPE PAL_NN_MODEL_T;
CREATE TYPE PAL_NN_MODEL_T AS TABLE(
    "NAME" VARCHAR(100),
    "MODEL" CLOB
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
--CREATE PROCEDURE
DROP TABLE PAL_NN_PDATA_TBL;
CREATE COLUMN TABLE PAL_NN_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
DELETE FROM PAL_NN_PDATA_TBL;
INSERT INTO PAL_NN_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_TRAIN_NN_DATA_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_TRAIN_NN_RESULT_T', 'OUT');
INSERT INTO PAL_NN_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_NN_MODEL_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_NN_TRAIN');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CREATEBPNN', 'DM_PAL',
'PAL_NN_TRAIN', PAL_NN_PDATA_TBL);
--RUN
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
```

```

"NAME" VARCHAR(100),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 3, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('HIDDEN_LAYER_SIZE', NULL, NULL, '10, 5');
INSERT INTO #PAL_CONTROL_TBL VALUES ('HIDDEN_LAYER_ACTIVE_FUNC', 9, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('OUTPUT_LAYER_ACTIVE_FUNC', 9, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LEARNING_RATE', NULL, 0.001, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MOMENTUM_FACTOR', NULL, 0.00001, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FUNCTIONALITY', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TRAINING_STYLE', 0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TARGET_COLUMN_NUM', 3, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 100, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NORMALIZATION', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('WEIGHT_INIT', 1, NULL, NULL);
DROP TABLE PAL_TRAIN_NN_RESULT_TBL;
CREATE COLUMN TABLE PAL_TRAIN_NN_RESULT_TBL LIKE PAL_TRAIN_NN_RESULT_T;
DROP TABLE PAL_REGRESSION_NN_MODEL_TBL;
CREATE COLUMN TABLE PAL_REGRESSION_NN_MODEL_TBL LIKE PAL_NN_MODEL_T;
DROP TABLE PAL_TRAIN_NN_DATA_TBL;
CREATE COLUMN TABLE PAL_TRAIN_NN_DATA_TBL LIKE PAL_TRAIN_NN_DATA_T;
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 1, 1.71, 'AC', 0, 12.7, 2.8, 3.06);
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 10, 1.78, 'CA', 5, 12.1, 8.0, 2.65);
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 17, 2.36, 'AA', 6, 10.1, 2.8, 3.24);
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 12, 3.15, 'AA', 2, 28.1, 5.6, 2.24);
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 7, 1.05, 'CA', 3, 19.8, 7.1, 1.98);
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 6, 1.50, 'CA', 2, 23.2, 4.9, 2.12);
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 9, 1.97, 'CA', 6, 24.5, 4.2, 1.05);
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 5, 1.26, 'AA', 1, 13.6, 5.1, 2.78);
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 12, 2.13, 'AC', 4, 13.2, 1.9, 1.34);
INSERT INTO PAL_TRAIN_NN_DATA_TBL VALUES( 18, 1.87, 'AC', 6, 25.5, 3.6, 2.14);
--CALL PROCEDURE
CALL DM_PAL.PAL_NN_TRAIN(PAL_TRAIN_NN_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_TRAIN_NN_RESULT_TBL, PAL_REGRESSION_NN_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_TRAIN_NN_RESULT_TBL;
SELECT * FROM PAL_REGRESSION_NN_MODEL_TBL;

```

Expected Results

Note: Your result may look slightly different from the following results.

PAL_TRAIN_NN_RESULT_TBL:

	NAME	VALUE
1	ERROR	0.00000038315881844974654

PAL_REGRESSION_NN_MODEL_TBL:

	NAME	MODEL
1	JSON	{"CurrentVersion":"1.0","DataDictionary": [{"dataType":"integer","name":"V000","optype":"c..."}]}

PREDICTWITHBPNN

This function makes prediction with a trained BPNN model.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PREDICTWITHBPNN',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<MODEL table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<OUTPUT table type>	OUT
5	<schema_name>	<SOFTMAX table type>	OUT (optional: only used for classification)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <model input table>,
<parameter table>, <output table>) WITH OVERVIEW;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

The model input table must store the trained neural network model.

Signature

Input Tables

Table	Column	Column Data Type	Description
Predicted data	1st column	Integer	ID
	Other columns	Integer, double, varchar, or nvarchar	Data to make prediction
Predicted model	1st column	Varchar or nvarchar	Model type
	2nd column	CLOB, varchar, or nvarchar	Model string

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	Other column(s)	Same as the last column of the training table if the model is for classification; and double if the model is for regression.	Classification or regression result
Softmax (optional: only used for classification)	1st column	Integer	ID
	2nd column	Same as the data type of the last column of the training data table.	Category label for classification
	3rd column	Double	Softmax value

Examples

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.
- Table PAL_CLASSIFICATION_NN_MODEL_TBL and PAL_REGRESSION_NN_MODEL_TBL store the trained neural network model under schema DM_PAL.

Classification example:

```

SET SCHEMA DM_PAL;
--PREDICT PART
DROP TYPE PAL_PREDICT_NN_DATA_T;
CREATE TYPE PAL_PREDICT_NN_DATA_T AS TABLE(
    "ID" INTEGER,
    "V000" INTEGER,
    "V001" DOUBLE,
    "V002" VARCHAR(10),
    "V003" INTEGER
);
DROP TYPE PAL_PREDICT_NN_RESULT_T;
CREATE TYPE PAL_PREDICT_NN_RESULT_T AS TABLE(
    "ID" INTEGER,
    "LABEL" VARCHAR(2)
);
DROP TYPE PAL_PREDICT_NN_SOFTMAX_T;
CREATE TYPE PAL_PREDICT_NN_SOFTMAX_T AS TABLE(
    "ID" INTEGER,
    "LABEL" VARCHAR(2),
    "SOFTMAX" DOUBLE
);
DROP TYPE PAL_NN_MODEL_T;
CREATE TYPE PAL_NN_MODEL_T AS TABLE(
    "NAME" VARCHAR(100),
    "MODEL" CLOB
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);

```

```

);
--CREATE PROCEDURE
DROP TABLE PAL_NN_PDATA_TBL;
CREATE COLUMN TABLE PAL_NN_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
DELETE FROM PAL_NN_PDATA_TBL;
INSERT INTO PAL_NN_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_PREDICT_NN_DATA_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_NN_MODEL_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_PREDICT_NN_RESULT_T',
'OUT');
INSERT INTO PAL_NN_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_PREDICT_NN_SOFTMAX_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_NN_PREDICT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PREDICTWITHBPNN', 'DM_PAL',
'PAL_NN_PREDICT', 'PAL_NN_PDATA_TBL');
--RUN
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 3, NULL, NULL);
DROP TABLE PAL_PREDICT_NN_DATA_TBL;
CREATE COLUMN TABLE PAL_PREDICT_NN_DATA_TBL LIKE PAL_PREDICT_NN_DATA_T;
INSERT INTO PAL_PREDICT_NN_DATA_TBL VALUES(1, 3, 1.91, 'AC', 3);
INSERT INTO PAL_PREDICT_NN_DATA_TBL VALUES(2, 7, 2.18, 'CA', 6);
INSERT INTO PAL_PREDICT_NN_DATA_TBL VALUES(3, 11, 1.96, 'AA', 8);
DROP TABLE PAL_PREDICT_NN_RESULT_TBL;
CREATE COLUMN TABLE PAL_PREDICT_NN_RESULT_TBL LIKE PAL_PREDICT_NN_RESULT_T;
DROP TABLE PAL_PREDICT_NN_SOFTMAX_TBL;
CREATE COLUMN TABLE PAL_PREDICT_NN_SOFTMAX_TBL LIKE PAL_PREDICT_NN_SOFTMAX_T;
--CALL PROCEDURE
CALL DM_PAL.PAL_NN_PREDICT(PAL_PREDICT_NN_DATA_TBL,
PAL_CLASSIFICATION_NN_MODEL_TBL, "#PAL_CONTROL_TBL", PAL_PREDICT_NN_RESULT_TBL,
PAL_PREDICT_NN_SOFTMAX_TBL) WITH OVERVIEW;
SELECT * FROM PAL_PREDICT_NN_RESULT_TBL;
SELECT * FROM PAL_PREDICT_NN_SOFTMAX_TBL;

```

Expected Results

Note: Your result may look slightly different from the following results.

PAL_PREDICT_NN_RESULT_TBL:

	ID	LABEL
1	1	AB
2	2	AB
3	3	AA

PAL_PREDICT_NN_SOFTMAX_TBL:

	ID	LABEL	SOFTMAX
1	1	AA	0.2799027256015996
2	1	AB	0.36097586941213605
3	1	C	0.3591214049862644
4	2	AA	0.31459810726209425
5	2	AB	0.36040238317699796
6	2	C	0.32499950956090773
7	3	AA	0.3626561879155719
8	3	AB	0.27638646412805445
9	3	C	0.36095734795637363

Regression example:

```

SET SCHEMA DM_PAL;
--PREDICT PART
DROP TYPE PAL_PREDICT_NN_DATA_T;
CREATE TYPE PAL_PREDICT_NN_DATA_T AS TABLE(
    "ID" INTEGER,
    "V000" INTEGER,
    "V001" DOUBLE,
    "V002" VARCHAR(10),
    "V003" INTEGER
);
DROP TYPE PAL_PREDICT_NN_RESULT_T;
CREATE TYPE PAL_PREDICT_NN_RESULT_T AS TABLE(
    "ID" INTEGER,
    "T001" DOUBLE,
    "T002" DOUBLE,
    "T003" DOUBLE
);
DROP TYPE PAL_NN_MODEL_T;
CREATE TYPE PAL_NN_MODEL_T AS TABLE(
    "NAME" VARCHAR(100),
    "MODEL" CLOB
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
--CREATE PROCEDURE
DROP TABLE PAL_NN_PDATA_TBL;
CREATE COLUMN TABLE PAL_NN_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
DELETE FROM PAL_NN_PDATA_TBL;
INSERT INTO PAL_NN_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_PREDICT_NN_DATA_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_NN_MODEL_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_NN_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_PREDICT_NN_RESULT_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_NN_PREDICT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PREDICTWITHBPNN', 'DM_PAL',
'PAL_NN_PREDICT', PAL_NN_PDATA_TBL);
--RUN

```

```

DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 3, NULL, NULL);
DROP TABLE PAL_PREDICT_NN_DATA_TBL;
CREATE COLUMN TABLE PAL_PREDICT_NN_DATA_TBL LIKE PAL_PREDICT_NN_DATA_T;
INSERT INTO PAL_PREDICT_NN_DATA_TBL VALUES(1, 1, 1.71, 'AC', 0);
INSERT INTO PAL_PREDICT_NN_DATA_TBL VALUES(2, 10, 1.78, 'CA', 5);
INSERT INTO PAL_PREDICT_NN_DATA_TBL VALUES(3, 17, 2.36, 'AA', 6);
DROP TABLE PAL_PREDICT_NN_RESULT_TBL;
CREATE COLUMN TABLE PAL_PREDICT_NN_RESULT_TBL LIKE PAL_PREDICT_NN_RESULT_T;
--CALL PROCEDURE
CALL DM.PAL.PAL_NN_PREDICT(PAL_PREDICT_NN_DATA_TBL, PAL_REGRESSION_NN_MODEL_TBL,
"#PAL_CONTROL_TBL", PAL_PREDICT_NN_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_PREDICT_NN_RESULT_TBL;

```

Expected Result

Note: Your result may look slightly different from the following result.

PAL_PREDICT_NN_RESULT_TBL:

	ID	T001	T002	T003
1	1	12.70024061322675	2.802439002998227	3.0609571045880517
2	2	12.105465268678843	7.984660176837242	2.650836161389475
3	3	10.100444846186642	2.799904263701653	3.2325896890976

3.2.3 C4.5 Decision Tree

A decision tree is used as a classifier for determining an appropriate action or decision among a predetermined set of actions for a given case. A decision tree helps you to effectively identify the factors to consider and how each factor has historically been associated with different outcomes of the decision. A decision tree uses a tree-like structure of conditions and their possible consequences. Each node of a decision tree can be a leaf node or a decision node.

- Leaf node: mentions the value of the dependent (target) variable.
- Decision node: contains one condition that specifies some test on an attribute value. The outcome of the condition is further divided into branches with sub-trees or leaf nodes.

As a classification algorithm, C4.5 builds decision trees from a set of training data, using the concept of information entropy. The training data is a set of already classified samples. At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits it into subsets in one class or the other. Its criterion is the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then proceeds recursively until meeting some stopping criteria such as the minimum number of cases in a leaf node.

The C4.5 decision tree functions implemented in PAL support both discrete and continuous values. In PAL implementation, the REP (Reduced Error Pruning) algorithm is used as pruning method.

Prerequisites

- The column order and column number of the predicted data are the same as the order and number used in tree model building.
- The last column of the training data is used as a predicted field and is of discrete type. The predicted data set has an ID column.
- The table used to store the tree model is a column table.
- The target column of training data must not have null values, and other columns should have at least one valid value (not null).

i Note

C4.5 decision tree treats null as a special value.

CREATEDWITHC45

This function creates a decision tree from the input training data.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'CREATEDWITHC45',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input_table>, <parameter_table>,
  <result_output_table>, <PMML_output_table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Training / Historical Data	Columns	Varchar, nvarchar, integer, or double	Table used to build the predictive tree model	Discrete value: integer, varchar, or nvarchar Continuous value: integer or double
	Last column	Varchar, nvarchar, or integer	Target variable (class label)	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
PERCENTAGE	Double	1.0	Specifies the percentage of the input data to be used to build the tree model. For example, if you set this parameter to 0.7, 70% of the training data will be used to build the tree model, and 30% will be used to prune the tree model.	
MIN_RECORDS_OF_PARENT	Integer	1	Specifies the stop condition: if the number of records is less than the parameter value, the algorithm will stop splitting.	
MIN_RECORDS_OF_LEAF	Integer	0	Promises the minimum number of records in each leaf.	

Name	Data Type	Default Value	Description	Dependency
MAX_DEPTH	Integer	Number of columns in the input table which contains the training data	Specifies the stop condition: if the depth of the tree model is greater than the parameter value, the algorithm will stop splitting.	
THREAD_NUMBER	Integer	1	Number of threads.	
IS_SPLIT_MODEL	Integer	1	Indicates whether the string of the tree model should be split or not. If the value is not 0, the tree model will be split, and the length of each unit is 5000.	
CONTINUOUS_COL	Integer	Detected from input data	Indicates which columns are continuous attributes. The default behavior is: <ul style="list-style-type: none">• String or integer: categorical• Double: continuous	
SPLIT_THRESHOLD	Double	0.05	Specifies the stop condition: if the information gain ratio is less than this value, the algorithm will stop splitting.	
<name of target value>	Double	Detected from input data	Specifies the priori probability of every class label.	

Name	Data Type	Default Value	Description	Dependency
SELECTED_FEATURES	Varchar	Detected from input data	A string to specify the features that will be processed. The pattern is "X ₁ ,...,X _n ", where X _i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed.	
DEPENDENT_VARIABLE	Varchar	Detected from input data	Column name in the data table used as dependent variable. If this parameter is not specified, the last column of the training data will be used as dependent variable.	
IS_OUTPUT_RULES	Integer	0	If this parameter is set to 1, the algorithm will extract all decision rules from the tree model and save them to the result table which is used to save the PMML model.	
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export PMML tree model. • 1: Exports PMML tree model in single row. • 2: Exports PMML tree model in several rows, and the minimum length of each row is 5000 characters. 	Only valid when IS_OUTPUT_RULES is 0.

Output Tables

Table	Column	Column Data Type	Description	Constraint
Tree model of JSON format	1st column	Integer	ID	
	2nd column	CLOB, varchar, or nvarchar	Tree model saved as a JSON string.	The table must be a column table. The minimum length of every unit (row) is 5000.
Tree model of PMML format	1st column	Integer	ID	
	2nd column	CLOB, varchar, or nvarchar	Tree model in PMML format	

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_C45_DATA_T;
CREATE TYPE PAL_C45_DATA_T AS TABLE(
    "OUTLOOK" VARCHAR(20),
    "TEMP" DOUBLE,
    "HUMIDITY" DOUBLE,
    "WINDY" VARCHAR(10),
    "CLASS" VARCHAR(20)
);
DROP TYPE PAL_C45_TREEMODEL_T;
CREATE TYPE PAL_C45_TREEMODEL_T AS TABLE(
    "ID" INTEGER,
    "TREE_MODEL" CLOB
);
DROP TYPE PAL_C45_PMMODEL_T;
CREATE TYPE PAL_C45_PMMODEL_T AS TABLE(
    "ID" INTEGER,
    "TREE_MODEL" CLOB
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_C45_PDATA_TBL;
CREATE COLUMN TABLE PAL_C45_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_C45_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_C45_DATA_T', 'IN');
INSERT INTO PAL_C45_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_C45_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_C45_TREEMODEL_T', 'OUT');
INSERT INTO PAL_C45_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_C45_PMMODEL_T',
'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_C45_PROC');

```

```

CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'CREATEDTWITHC45',
'DM_PAL', 'PAL_C45_PROC', 'PAL_C45_PDATA_TBL');
DROP TABLE PAL_C45_DATA_TBL;
CREATE COLUMN TABLE PAL_C45_DATA_TBL LIKE PAL_C45_DATA_T;
INSERT INTO PAL_C45_DATA_TBL VALUES ('Sunny', 75, 70, 'Yes', 'Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Sunny', 80, 90, 'Yes', 'Do not Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Sunny', 85, 85, 'No', 'Do not Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Sunny', 72, 95, 'No', 'Do not Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Sunny', 69, 70, 'No', 'Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Overcast', 72, 90, 'Yes', 'Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Overcast', 83, 78, 'No', 'Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Overcast', 64, 65, 'Yes', 'Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Overcast', 81, 75, 'No', 'Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Rain', 71, 80, 'Yes', 'Do not Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Rain', 65, 70, 'Yes', 'Do not Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Rain', 75, 80, 'No', 'Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Rain', 68, 80, 'No', 'Play');
INSERT INTO PAL_C45_DATA_TBL VALUES ('Rain', 70, 96, 'No', 'Play');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
  "NAME" VARCHAR (100),
  "INTARGS" INTEGER,
  "DOUBLEARGS" DOUBLE,
  "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SPLIT_THRESHOLD', null, 0.05, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_DEPTH', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_RECORDS_OF_PARENT', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_RECORDS_OF_LEAF', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('IS_OUTPUT_RULES', 1, null, null);
DROP TABLE PAL_C45 TREEMODEL_TBL;
CREATE COLUMN TABLE PAL_C45 TREEMODEL_TBL LIKE PAL_C45 TREEMODEL_T;
DROP TABLE PAL_C45 PMMLMODEL_TBL;
CREATE COLUMN TABLE PAL_C45 PMMLMODEL_TBL LIKE PAL_C45 PMMLMODEL_T;
CALL "DM_PAL".PAL_C45_PROC(PAL_C45_DATA_TBL, #PAL_CONTROL_TBL,
PAL_C45_TREEMODEL_TBL, PAL_C45_PMMLMODEL_TBL) with OVERVIEW;
SELECT * FROM PAL_C45 TREEMODEL_TBL;
SELECT * FROM PAL_C45 PMMLMODEL_TBL;

```

Expected Result

PAL_C45_TREEMODEL_TBL:

ID	TREE_MODEL
0	{"CurrentVersion":"VERSION1.3","DataDictionary": [{"DataType": "Str", "Name": "OUTLOOK", "Value": "Sunny"}, {"DataType": "Num", "Name": "TEMP", "Value": 75}, {"DataType": "Num", "Name": "HUMIDITY", "Value": 70}, {"DataType": "Str", "Name": "WINDY", "Value": "Yes"}, {"DataType": "Str", "Name": "PREDICTION", "Value": "Play"}]}

PAL_C45_PMMLMODEL_TBL:

ID	TREE_MODEL
0	(TEMP>=84) => Do not Play
1	(TEMP<84) && (OUTLOOK=Overcast) => Play
2	(TEMP<84) && (OUTLOOK=Sunny) && (HUMIDITY<82.5) => Play
3	(TEMP<84) && (OUTLOOK=Sunny) && (HUMIDITY>=82.5) => Do not Play
4	(TEMP<84) && (OUTLOOK=Rain) && (WINDY=Yes) => Do not Play
5	(TEMP<84) && (OUTLOOK=Rain) && (WINDY=No) => Play

3.2.4 CART Decision Tree

Classification and regression tree (CART) is used for classification or regression and only supports binary split. CART is a recursive partitioning method similar to C4.5 decision tree. It uses GINI index or TWOING for classification, and least square error for regression. In PAL, CART only supports the GINI split strategy. The surrogate split method is used to support missing values when creating the tree model.

Prerequisites

- The target column of training data must not have null values, and other columns should have at least one valid value (not null).
- The table used to store the tree model is a column table.

CART

This function is used for classification or regression.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'CART', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Training table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Tree Model OUTPUT table type>	OUT
4	<schema_name>	<Statistic OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<training_table>, <parameter_table>,  
<tree_model_output_table>, <statistic_output_table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Training Data	Columns	Varchar, nvarchar, double, or integer	Independent fields	
	Last column	Varchar, nvarchar, double, or integer	Dependent field	Null values are not allowed.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
SPLIT_CRITERIA	Integer	103	Indicates the split strategy: <ul style="list-style-type: none">• 103: GINI	
SPLIT_THRESHOLD	Double	0.05	Specifies the stop condition: if the improvement value of the best split is less than the specified value, the algorithm will stop splitting.	
MAX_DEPTH	Integer	Number of columns in the training data	Specifies the stop condition: if the depth of the tree model is greater than the specified value, the algorithm will stop splitting.	
MIN_RECORDS_OF_PARENT	Integer	1	Specifies the stop condition: if the number of records is less than the specified value, the algorithm will stop splitting.	
MIN_RECORDS_OF_LEAF	Integer	0	Promises the minimum number of records in each leaf.	

Name	Data Type	Default Value	Description	Dependency
PERCENTAGE	Double	1.0	<p>Specifies the percentage of the input data that will be used to build the tree model.</p> <p>For example, if you set this parameter to 0.7, then 70% of the training data will be used to build the tree model and 30% will be used to prune the tree model.</p>	
USE_SURROGATE	Integer	1	<p>Indicates whether to use surrogate split when null values are encountered.</p> <ul style="list-style-type: none"> • 0: Does not use surrogate split • 1: Uses surrogate split 	
CONTINUOUS_COL	Integer	Detected from input data	<p>Indicates which columns are continuous attributes. The default behavior is:</p> <ul style="list-style-type: none"> • String or integer: categorical • Double: continuous 	
<name of target value>	Double	Detected from input data	Specifies the priori probability of every class label.	
SELECTED_FEATURES	Varchar	Detected from input data	A string to specify the features that will be processed. The pattern is "X ₁X _n ", where X _i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed.	

Name	Data Type	Default Value	Description	Dependency
DEPENDENT_VARIABLE	Varchar	Detected from input data	Column name in the data table used as dependent variable. If this parameter is not specified, the last column of the training data will be used as dependent variable.	
IS_OUTPUT_RULES	Integer	0	If this parameter is set to 1, the algorithm will extract all decision rules from the tree model and save them into the tree model table.	
IS_SPLIT_MODEL	Integer	1	Indicates whether the string of the JSON tree model should be split or not. If IS_OUTPUT_RULES and PMML_EXPORT are both set to 0, the algorithm will output JSON tree model. If the value is not 0, the JSON tree model will be split, and the length of each unit is 5000.	
THREAD_NUMBER	Integer	1	Number of threads.	
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export PMML tree model. • 1: Exports PMML tree model in single row. • 2: Exports PMML tree model in several rows, and the minimum length of each row is 5000 characters. 	Only valid when IS_OUTPUT_RULES is 0.

Output Tables

Table	Column	Column Data Type	Description
Tree Model	1st column	Integer	ID
	2nd column	Varchar or nvarchar	The length should be 5000.
Statistics	1st column	Varchar or nvarchar	Dependent field name
	2nd column	Varchar or nvarchar	Dependent field name
	3rd column	Integer	Number of records

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_CART_DATA_T;
CREATE TYPE PAL_CART_DATA_T AS TABLE(
    "OUTLOOK" VARCHAR(20),
    "TEMP" DOUBLE,
    "HUMIDITY" DOUBLE,
    "WINDY" VARCHAR(10),
    "CLASS" VARCHAR(20)
);
DROP TYPE PAL_CART TREEMODEL_T;
CREATE TYPE PAL_CART_TREEMODEL_T AS TABLE(
    "ID" INTEGER,
    "TREE_MODEL" VARCHAR(5000)
);
DROP TYPE PAL_CART_STATISTIC_T;
CREATE TYPE PAL_CART_STATISTIC_T AS TABLE(
    "CLASS1" VARCHAR(20),
    "CLASS2" VARCHAR(20),
    "NUMBER" INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_CART_PDATA_TBL;
CREATE COLUMN TABLE PAL_CART_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_CART_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_CART_DATA_T', 'IN');
INSERT INTO PAL_CART_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_CART_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CART_TREEMODEL_T',
'OUT');
INSERT INTO PAL_CART_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_CART_STATISTIC_T',
'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_CART_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CART', 'DM_PAL',
'PAL_CART_PROC', PAL_CART_PDATA_TBL);
DROP TABLE PAL_CART_DATA_TBL;
CREATE COLUMN TABLE PAL_CART_DATA_TBL LIKE PAL_CART_DATA_T;

```

```

INSERT INTO PAL_CART_DATA_TBL VALUES ('Sunny', 75, 70, 'Yes', 'Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Sunny', 80, 90, 'Yes', 'Do not Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Sunny', 85, 85, 'No', 'Do not Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Sunny', 72, 95, 'No', 'Do not Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Sunny', 69, 70, 'No', 'Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Overcast', 72, 90, 'Yes', 'Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Overcast', 83, 78, 'No', 'Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Overcast', 64, 65, 'Yes', 'Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Overcast', 81, 75, 'No', 'Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Rain', 71, 80, 'Yes', 'Do not Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Rain', 65, 70, 'Yes', 'Do not Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Rain', 75, 80, 'No', 'Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Rain', 68, 80, 'No', 'Play');
INSERT INTO PAL_CART_DATA_TBL VALUES ('Rain', 70, 96, 'No', 'Play');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SPLIT_CRITERIA', 103, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SPLIT_THRESHOLD', null, 0.001, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_DEPTH', 5, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_RECORDS_OF_PARENT', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_RECORDS_OF_LEAF',1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT', 2, null, null);
DROP TABLE PAL_CART_TREEMODEL_TBL;
CREATE COLUMN TABLE PAL_CART_TREEMODEL_TBL LIKE PAL_CART_TREEMODEL_T;
DROP TABLE PAL_CART_STATISTIC_TBL;
CREATE COLUMN TABLE PAL_CART_STATISTIC_TBL LIKE PAL_CART_STATISTIC_T;
CALL "DM_PAL".PAL_CART_PROC(PAL_CART_DATA_TBL, #PAL_CONTROL_TBL,
    PAL_CART_TREEMODEL_TBL, PAL_CART_STATISTIC_TBL) with OVERVIEW;
SELECT * FROM PAL_CART_TREEMODEL_TBL;
SELECT * FROM PAL_CART_STATISTIC_TBL;

```

Expected Result:

PAL_CART_TREEMODEL_TBL:

ID	TREE_MODEL
1	<PMML version="4.0" xmlns="http://www.dmg.org...

PAL_CART_STATISTIC_TBL:

CLASS1	CLASS2	NUMBER
Play	Play	9
Play	Do not Play	0
Do not Play	Play	0
Do not Play	Do not Play	5

3.2.5 CHAID Decision Tree

CHAID stands for CHi-squared Automatic Interaction Detection. It is similar to the C4.5 decision tree. CHAID is a classification method for building decision trees by using chi-square statistics to identify optimal splits.

CHAID examines the cross tabulations between each of the input fields and the outcome, and tests for significance using a chi-square independence test. If more than one of these relations is statistically significant, CHAID will select the input field that is the most significant (smallest p value). CHAID can generate non-binary trees.

Prerequisites

- The target column of the training data must not have null values, and other columns should have at least one valid value (not null).
- The table used to store the tree model is a column table.

i Note

CHAID treats null values as special values.

CREATEDWITHCHAID

This function creates a decision tree from the input training data. It can be used for classification.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'CREATEDWITHCHAID',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Training table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<JSON Tree Model table type>	OUT
4	<schema_name>	<PMML Tree Model table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<training_table>, <parameter_table>,
<json_tree_model_table>, <PMML_tree_model_table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Training Data	Columns	Varchar, nvarchar, integer, or double	Independent fields	
	Last column	Varchar, nvarchar, or integer	Dependent field	Null values are not allowed.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
SPLIT_THRESHOLD	Double	0.05	Specifies the stop condition: if the p-value of the best split is greater than or equal to the specified value, the algorithm will stop splitting.	
MERGE_THRESHOLD	Double	0.05	Specifies the merge condition: if the metric value is greater than or equal to the specified value, the algorithm will merge two branches.	
MIN_RECORDS_OF_PARANT	Integer	1	Specifies the stop condition: if the number of records is less than the specified value, the algorithm will stop splitting.	
MIN_RECORDS_OF_LEAF	Integer	0	Promises the minimum number of records in each leaf.	

Name	Data Type	Default Value	Description	Dependency
PERCENTAGE	Double	1.0	<p>Specifies the percentage of the input data that will be used to build the tree model.</p> <p>For example, if you set this parameter to 0.7, then 70% of the training data will be used to build the tree model and 30% will be used to prune the tree model.</p>	
MAX_DEPTH	Integer	Number of columns in the training data	Specifies the stop condition: if the depth of the tree model is greater than the specified value, the algorithm will stop splitting.	
MAX_BRANCH	Integer	10	Specifies the maximum number of branches.	
THREAD_NUMBER	Integer	1	Number of threads.	
IS_SPLIT_MODEL	Integer	1	<p>Indicates whether the string of the JSON tree model should be split or not.</p> <p>If the value is not 0, the JSON tree model will be split, and the length of each unit is 5000.</p>	
CONTINUOUS_COL	Integer	Detected from input data	<p>Indicates which columns are continuous attributes. The default behavior is:</p> <ul style="list-style-type: none"> • String or integer: categorical • Double: continuous 	

Name	Data Type	Default Value	Description	Dependency
<name of target value>	Double	Detected from input data	Specifies the prior probability of every class label.	
SELECTED_FEATURES	Varchar	Detected from input data	A string to specify the features that will be processed. The pattern is "X ₁X _n ", where X _i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed.	
DEPENDENT_VARIABLE	Varchar	Detected from input data	Column name in the data table used as dependent variable. If this parameter is not specified, the last column of the training data will be used as dependent variable.	
DISCRETIZATION_TYPE	Integer	0	Specifies the strategy for discretizing continuous attributes: <ul style="list-style-type: none"> • 0: MDLPC • 1: Equal Frequency 	
<column name>	Integer	10	If the column is continuous and DISCRETIZATION_TYPE is set to 1, you can use this parameter to specify the number of bins	Only valid when DISCRETIZATION_TYPE is 1.
IS_OUTPUT_RULES	Integer	0	If this parameter is set to 1, the algorithm will extract all decision rules from the tree model and save them into the PMML tree model table.	

Name	Data Type	Default Value	Description	Dependency
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export PMML tree model. • 1: Exports PMML tree model in single row. • 2: Exports PMML tree model in several rows, and the minimum length of each row is 5000 characters. 	Only valid when IS_OUTPUT_RULES is 0.

Output Tables

Table	Column	Column Data Type	Description	Constraint
JSON Tree Model	1st column	Integer	ID	
	2nd column	Varchar or nvarchar	Tree model saved as a JSON string	The minimum length of each unit (row) is 5000.
PMML Tree Model	1st column	Integer	ID	
	2nd column	Varchar or nvarchar	Tree model in PMML format	The minimum length of each unit (row) is 5000.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_CHAID_DATA_T;
CREATE TYPE PAL_CHAID_DATA_T AS TABLE(
    "OUTLOOK" VARCHAR(20),
    "TEMP" DOUBLE,
    "HUMIDITY" DOUBLE,
    "WINDY" VARCHAR(10),
    "CLASS" VARCHAR(20)
);
DROP TYPE PAL_CHAID TREEMODEL_T;
CREATE TYPE PAL_CHAID TREEMODEL_T AS TABLE(
    "ID" INTEGER,
    "TREE_MODEL" CLOB
);
DROP TYPE PAL_CHAID_PMMILMODEL_T;
CREATE TYPE PAL_CHAID_PMMILMODEL_T AS TABLE(
    "ID" INTEGER,

```

```

    "PMML_MODEL" CLOB
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_CHAID_PDATA_TBL;
CREATE COLUMN TABLE PAL_CHAID_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_CHAID_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_CHAID_DATA_T', 'IN');
INSERT INTO PAL_CHAID_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_CHAID_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CHAID_TREEMODEL_T',
'OUT');
INSERT INTO PAL_CHAID_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_CHAID_PMMILMODEL_T',
'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_CHAID_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CREATEDTWITHCHAID',
'DM_PAL', 'PAL_CHAID_PROC', 'PAL_CHAID_PDATA_TBL');
DROP TABLE PAL_CHAID_DATA_TBL;
CREATE COLUMN TABLE PAL_CHAID_DATA_TBL LIKE PAL_CHAID_DATA_T;
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Sunny', 75, 70, 'Yes', 'Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Sunny', 80, 90, 'Yes', 'Do not Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Sunny', 85, 85, 'No', 'Do not Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES (NULL, 72, 95, 'No', 'Do not Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Sunny', 69, 70, 'No', 'Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Overcast', 72, 90, 'Yes', 'Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Overcast', 83, 78, 'No', 'Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Overcast', 64, 65, 'Yes', 'Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Overcast', 81, 75, 'No', 'Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Rain', 71, 80, 'Yes', 'Do not Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Rain', 65, 70, 'Yes', 'Do not Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Rain', 75, 80, 'No', 'Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Rain', 68, 80, 'No', 'Play');
INSERT INTO PAL_CHAID_DATA_TBL VALUES ('Rain', 70, 96, 'No', 'Play');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SPLIT_THRESHOLD', null, 0.2, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MERGE_THRESHOLD', null, 0.05, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_DEPTH', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_RECORDS_OF_PARENT', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT', 2, null, null);
DROP TABLE PAL_CHAID_TREEMODEL_TBL;
CREATE COLUMN TABLE PAL_CHAID_TREEMODEL_TBL LIKE PAL_CHAID_TREEMODEL_T;
DROP TABLE PAL_CHAID_PMMILMODEL_TBL;
CREATE COLUMN TABLE PAL_CHAID_PMMILMODEL_TBL LIKE PAL_CHAID_PMMILMODEL_T;
CALL "DM_PAL".PAL_CHAID_PROC(PAL_CHAID_DATA_TBL, #PAL_CONTROL_TBL,
PAL_CHAID_TREEMODEL_TBL, PAL_CHAID_PMMILMODEL_TBL) with OVERVIEW;
SELECT * FROM PAL_CHAID_TREEMODEL_TBL;
SELECT * FROM PAL_CHAID_PMMILMODEL_TBL;

```

Expected Result

PAL_CHAID_TREEMODEL_TBL:

ID	TREE_MODEL
0	{"CurrentVersion":"VERSION1.3","DataDictionary...}

PAL_CHAID_PMMML_TBL:

ID	TREE_MODEL
1	<PMML version="4.0" xmlns="http://www.dmg.org/PMML...>

3.2.6 Confusion Matrix

Confusion matrix is a traditional method to evaluate the performance of classification algorithms, including the multiple-class condition.

The following is an example confusion matrix of a 3-class classification problem. The rows show the original labels and the columns show the predicted labels. For example, the number of class 0 samples classified as class 0 is a ; the number of class 0 samples classified as class 2 is c .

	Class 0	Class 1	Class 2
Class 0	a	b	c
Class 1	d	e	f
Class 2	g	h	i

From the confusion matrix, you can compute the precision, recall, and F1-score for each class. In the above example, the precision and recall of class 0 are:

$$\text{precision}_{\text{class}0} = \frac{a}{a+d+g}$$

$$\text{recall}_{\text{class}0} = \frac{a}{a+b+c}$$

F1-score is a combination of precision and recall as follows:

$$\text{F1score}_{\text{class}0} = \frac{2 * \text{precision}_{\text{class}0} * \text{recall}_{\text{class}0}}{\text{precision}_{\text{class}0} + \text{recall}_{\text{class}0}}$$

You can also calculate the F β -score:

$$\text{F}\beta\text{score}_{\text{class}0} = \frac{(1+\beta^2) * \text{precision}_{\text{class}0} * \text{recall}_{\text{class}0}}{\beta^2 * \text{precision}_{\text{class}0} + \text{recall}_{\text{class}0}}$$

Prerequisite

- No missing or null data in the inputs

CONFUSIONMATRIX

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'CONFUSIONMATRIX',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Input table type>	IN
2	<schema_name>	<Parameter table type>	IN
3	<schema_name>	<Confusion matrix table type>	OUT
4	<schema_name>	<Classification report table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input tables>, <parameter tables>,
<confusion matrix table>, <classification report table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer, varchar, or nvarchar	Original label	The data type of the 2nd and 3rd columns must be the same.
	3rd column	Integer, varchar, or nvarchar	Predicted label	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
BETA	Double	1	This parameter is used to compute the F β -score. The value range is (0, $+\infty$).

Output Tables

Table	Column	Column Data Type	Description
Confusion Matrix	1st column	Varchar or nvarchar	Class name
	Other columns	Integer	Columns numbers are dependent on the class numbers in the given data. Each column stores the count of the corresponding predicted label and each row stores the count of the corresponding original label. The whole confusion matrix is stored in these columns.
Classification Report	1st column	Varchar or nvarchar	Class name
	2nd column	Double	The recall of each class
	3rd column	Double	The precision of each class
	4th column	Double	The F-measure of each class
	5th column	Integer	The support - sample number in each class

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and

- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_CM_DATA_T;
CREATE TYPE PAL_CM_DATA_T AS TABLE (
ID INTEGER,
ORIGINAL INTEGER,
PREDICT INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
NAME VARCHAR(50),
INTARGS INTEGER,
DOUBLEARGS DOUBLE,
STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_CM_CONFUSIONMATRIX_T;
CREATE TYPE PAL_CM_CONFUSIONMATRIX_T AS TABLE(
ID varchar(100),
CLASS1 INTEGER,
CLASS2 INTEGER
);
DROP TYPE PAL_CM_CLASSIFICATIONREPORT_T;
CREATE TYPE PAL_CM_CLASSIFICATIONREPORT_T AS TABLE(
ID varchar(100),
RECALL DOUBLE,
PRECISIONV DOUBLE,
FMEASURE DOUBLE,
SUPPORT INTEGER
);
DROP TABLE PAL_CM_PDATA_TBL;
CREATE COLUMN TABLE PAL_CM_PDATA_TBL (
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_CM_PDATA_TBL VALUES (1,'DM_PAL','PAL_CM_DATA_T', 'IN');
INSERT INTO PAL_CM_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_CM_PDATA_TBL VALUES (3,'DM_PAL','PAL_CM_CONFUSIONMATRIX_T',
'OUT');
INSERT INTO PAL_CM_PDATA_TBL VALUES (4,'DM_PAL','PAL_CM_CLASSIFICATIONREPORT_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_CM');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CONFUSIONMATRIX', 'DM_PAL',
'PAL_CM', PAL_CM_PDATA_TBL);
DROP TABLE PAL_CM_DATA_TBL;
CREATE COLUMN TABLE PAL_CM_DATA_TBL LIKE PAL_CM_DATA_T;
INSERT INTO PAL_CM_DATA_TBL VALUES(1,1,1);
INSERT INTO PAL_CM_DATA_TBL VALUES(2,1,1);
INSERT INTO PAL_CM_DATA_TBL VALUES(3,1,1);
INSERT INTO PAL_CM_DATA_TBL VALUES(4,1,2);
INSERT INTO PAL_CM_DATA_TBL VALUES(5,1,1);
INSERT INTO PAL_CM_DATA_TBL VALUES(6,2,2);
INSERT INTO PAL_CM_DATA_TBL VALUES(7,2,1);
INSERT INTO PAL_CM_DATA_TBL VALUES(8,2,2);
INSERT INTO PAL_CM_DATA_TBL VALUES(9,2,2);
INSERT INTO PAL_CM_DATA_TBL VALUES(10,2,2);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
INSERT INTO #PAL_CONTROL_TBL VALUES('BETA',NULL,1,null);
DROP TABLE PAL_CM_CONFUSIONMATRIX_TBL;
CREATE COLUMN TABLE PAL_CM_CONFUSIONMATRIX_TBL LIKE PAL_CM_CONFUSIONMATRIX_T;
DROP TABLE PAL_CM_CLASSIFICATIONREPORT_TBL;
CREATE COLUMN TABLE PAL_CM_CLASSIFICATIONREPORT_TBL LIKE
PAL_CM_CLASSIFICATIONREPORT_T;

```

```

CALL DM PAL.PAL_CM(PAL_CM_DATA_TBL, #PAL_CONTROL_TBL,
PAL_CM_CONFUSIONMATRIX_TBL, PAL_CM_CLASSIFICATIONREPORT_TBL) with OVERVIEW;
SELECT * FROM PAL_CM_CONFUSIONMATRIX_TBL;
SELECT * FROM PAL_CM_CLASSIFICATIONREPORT_TBL;

```

Expected Results

PAL_CM_CONFUSIONMATRIX_TBL:

	ID	CLASS1	CLASS2
1	1	4	1
2	2	1	4

PAL_CM_CLASSIFICATIONREPORT_TBL:

	ID	RECALL	PRECISIONV	FME...	SUPPORT
1	1	0.8	0.8	0.800...	5
2	2	0.8	0.8	0.800...	5
3	mean	0.8	0.8	0.800...	5

3.2.7 KNN

K-Nearest Neighbor (KNN) is a memory based classification method with no explicit training phase. In the testing phase, given a query sample x , its top K nearest samples are found in the training set first, then the label of x is assigned as the most frequent label of the K nearest neighbors. In this release of PAL, the description of each sample should be real numbers. In order to speed up the search, the KD-tree searching method is provided.

Prerequisites

- The first column of the training data and input data is an ID column. The second column of the training data is of class type. The class type column is of integer type. Other data columns are of integer or double type.
- The input data does not contain null value.

KNN

This is a classification function using the KNN algorithm.

Procedure Generation

```

CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'KNN', '<schema_name>',
'<procedure_name>', <signature_table>);

```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Training INPUT table type>	IN
2	<schema_name>	<Class INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<training input table>, <class input table>,
<parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Training Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer, varchar, or nvarchar	Class type
	Other columns	Integer or double	Attribute data
Class Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	Other columns	Integer or double	Attribute data

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
K_NEAREST_NEIGHBOURS	Integer	1	Number of nearest neighbors (k).
ATTRIBUTE_NUM	Integer	1	Number of attributes.

Name	Data Type	Default Value	Description
VOTING_TYPE	Integer	1	Voting type: • 0: Majority voting • 1: Distance-weighted voting
METHOD	Integer	0	Searching method. • 0: Brute force searching • 1: KD-tree searching
THREAD_NUMBER	Integer	1	Number of threads.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer, varchar, or nvarchar	Class type

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_KNN_DATA_T;
CREATE TYPE PAL_KNN_DATA_T AS TABLE(
    "ID" INTEGER,
    "TYPE" INTEGER,
    "X1" DOUBLE,
    "X2" DOUBLE
);
DROP TYPE PAL_KNN_CLASSDATA_T;
CREATE TYPE PAL_KNN_CLASSDATA_T AS TABLE(
    "ID" INTEGER,
    "X1" DOUBLE,
    "X2" DOUBLE
);
DROP TYPE PAL_KNN_RESULT_T;
CREATE TYPE PAL_KNN_RESULT_T AS TABLE(
    "ID" INTEGER,
    "TYPE" INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_KNN_PDATA_TBL;
CREATE COLUMN TABLE PAL_KNN_PDATA_TBL(

```

```

"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_KNN_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_KNN_DATA_T', 'IN');
INSERT INTO PAL_KNN_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_KNN_CLASSDATA_T', 'IN');
INSERT INTO PAL_KNN_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_KNN_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_KNN_RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_KNN_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'KNN', 'DM_PAL',
'PAL_KNN_PROC', 'PAL_KNN_PDATA_TBL');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('K_NEAREST_NEIGHBOURS',3,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ATTRIBUTE_NUM',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('VOTING_TYPE',0,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',8,null,null);
DROP TABLE PAL_KNN_DATA_TBL;
CREATE COLUMN TABLE PAL_KNN_DATA_TBL LIKE PAL_KNN_DATA_T;
INSERT INTO PAL_KNN_DATA_TBL VALUES (0,2,1,1);
INSERT INTO PAL_KNN_DATA_TBL VALUES (1,3,10,10);
INSERT INTO PAL_KNN_DATA_TBL VALUES (2,3,10,11);
INSERT INTO PAL_KNN_DATA_TBL VALUES (3,3,10,10);
INSERT INTO PAL_KNN_DATA_TBL VALUES (4,1,1000,1000);
INSERT INTO PAL_KNN_DATA_TBL VALUES (5,1,1000,1001);
INSERT INTO PAL_KNN_DATA_TBL VALUES (6,1,1000,999);
INSERT INTO PAL_KNN_DATA_TBL VALUES (7,1,999,999);
INSERT INTO PAL_KNN_DATA_TBL VALUES (8,1,999,1000);
INSERT INTO PAL_KNN_DATA_TBL VALUES (9,1,1000,1000);
DROP TABLE PAL_KNN_CLASSDATA_TBL;
CREATE COLUMN TABLE PAL_KNN_CLASSDATA_TBL LIKE PAL_KNN_CLASSDATA_T;
INSERT INTO PAL_KNN_CLASSDATA_TBL VALUES (0,2,1);
INSERT INTO PAL_KNN_CLASSDATA_TBL VALUES (1,9,10);
INSERT INTO PAL_KNN_CLASSDATA_TBL VALUES (2,9,11);
INSERT INTO PAL_KNN_CLASSDATA_TBL VALUES (3,15000,15000);
INSERT INTO PAL_KNN_CLASSDATA_TBL VALUES (4,1000,1000);
INSERT INTO PAL_KNN_CLASSDATA_TBL VALUES (5,500,1001);
INSERT INTO PAL_KNN_CLASSDATA_TBL VALUES (6,500,999);
INSERT INTO PAL_KNN_CLASSDATA_TBL VALUES (7,199,999);
DROP TABLE PAL_KNN_RESULTS_TBL;
CREATE COLUMN TABLE PAL_KNN_RESULTS_TBL LIKE PAL_KNN_RESULT_T;
CALL "DM_PAL".PAL_KNN_PROC(PAL_KNN_DATA_TBL, PAL_KNN_CLASSDATA_TBL,
"#PAL_CONTROL_TBL", PAL_KNN_RESULTS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_KNN_RESULTS_TBL;

```

Expected Result

PAL_KNN_RESULTS_TBL:

ID	Type
0	3
1	3
2	3
3	1
4	1
5	1
6	1
7	1

3.2.8 Logistic Regression (with Elastic Net Regularization)

Logistic regression models the relationship between a dichotomous dependent variable (also known as explained variable) and one or more continuous or categorical independent variables (also known as explanatory variables). It models the log odds of the dependent variable as a linear combination of the independent variables.

This function can only handle binary-class classification problems. For multiple-class classification problems, refer to *Multi-Class Logistic Regression*.

Considering a training data set with n samples and m explanatory variables, the logistic regression model is made by:

$$h_{(\theta_0, \theta)}(x) = 1 / (1 + \exp(-(\theta_0 + \theta^T x)))$$

Where θ_0 is the intercept, θ represents coefficients $\theta_1, \dots, \theta_m$ and $\theta^T x = \theta_1 x_1 + \dots + \theta_m x_m$

Assuming that there are only two class labels, {0,1}, you can get the below formula:

$$P(y = 1 | x; (\theta_0, \theta)) = h_{(\theta_0, \theta)}(x)$$

$$P(y = 0 | x; (\theta_0, \theta)) = 1 - h_{(\theta_0, \theta)}(x)$$

And combine them into:

$$P(y | x; (\theta_0, \theta)) = h_{(\theta_0, \theta)}(x)^y (1 - h_{(\theta_0, \theta)}(x))^{1-y}$$

Here $\theta_0, \theta_1, \dots, \theta_m$ can be obtained through the Maximum Likelihood Estimation (MLE) method.

The likelihood function is:

$$L(\theta_0, \theta) = \prod_{i=1}^n P(y^{(i)} | x^{(i)}; (\theta_0, \theta))$$

The log-likelihood function is:

$$\begin{aligned} l(\theta_0, \theta) &= \ln(\prod_{i=1}^n P(y^{(i)} | x^{(i)}; (\theta_0, \theta))) \\ &= \ln(\prod_{i=1}^n (h_{(\theta_0, \theta)}(x^{(i)}))^y^{(i)} (1 - h_{(\theta_0, \theta)}(x^{(i)}))^{1-y^{(i)}}) \\ &= \sum_{i=1}^n (y^{(i)} \ln(h_{(\theta_0, \theta)}(x^{(i)})) + (1-y^{(i)}) \ln(1 - h_{(\theta_0, \theta)}(x^{(i)}))) \end{aligned}$$

Newton method, Gradient Descend, BFGS and Cyclical Coordinate Descend (primarily for elastic net penalized object function) are provided to minimize objective function which is opposite in sign with log likelihood function. For fast convergence, Newton method and BFGS are preferred.

Elastic net regularization seeks to find coefficients which can minimize:

$$\min_{(\theta_0, \theta) \in \mathbb{R}^{m+1}} \left(-\frac{l(\theta_0, \theta)}{n} + \lambda * P_\alpha(\theta) \right)$$

$$\text{Where } P_\alpha(\theta) = (1-\alpha) \frac{1}{2} \|\theta\|_2^2 + \alpha \|\theta\|_1.$$

Here $\alpha \in [0, 1]$ and $\lambda \geq 0$. If $\alpha=0$, we have the ridge regularization; if $\alpha=1$, we have the LASSO regularization.

Function FORECASTWITHLOGISTICR is used to predict the labels for the testing data.

Prerequisites

- No missing or null data in inputs.
- Data is numeric, or categorical.
- Given m independent variables, there must be at least m+1 records available for analysis.

LOGISTICREGRESSION

This is a logistic regression function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LOGISTICREGRESSION',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
	<schema_name>	<Statistics OUTPUT table type>	OUT (optional)
5	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>, <statistics output table>, <PMML output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	Columns	Integer, double, varchar, or nvarchar	Independent variables
	Type column	Integer, varchar, or nvarchar	Dependent Variables Note: For integer dependent variables, it must take value 0 or 1.

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description	Dependency
CLASS_MAPO	Varchar	Specifies the dependent variable value which will be mapped to 0.	Only valid when the column type of dependent variables is varchar or nvarchar.
CLASS_MAP1	Varchar	Specifies the dependent variable value which will be mapped to 1.	Only valid when the column type of dependent variables is varchar or nvarchar.
CATEGORY_COL	Integer	Indicates whether the integer type column holds discrete independent variables. By default, varchar or nvarchar type column holds discrete independent variables and integer or double type column holds continuous independent variables.	

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
METHOD	Integer	0	<ul style="list-style-type: none"> • 0: Newton iteration method. • 1: Gradient-decent method. • 2: Cyclical coordinate descent method to fit elastic net. regularized logistic regression. • 3: LBFGS method (recommended when having many independent variables). • 4: Stochastic gradient descent method (recommended when dealing with very large dataset). 	
STEP_SIZE	Double	No default value	Step size for line searching.	Only valid when METHOD is 1. When ignoring this optional parameter, optimal step-size will be determined automatically at each iteration.
ENET_LAMBDA	Double	No default value	Penalized weight. The value should be equal to or greater than 0.	Only valid when METHOD is 2.
ENET_ALPHA	Double	1.0	The elastic net mixing parameter. The value range is between 0 and 1 inclusively. <ul style="list-style-type: none"> • 0: Ridge penalty • 1: LASSO penalty 	Only valid when METHOD is 2.

Name	Data Type	Default Value	Description	Dependency
MAX_ITERATION	Integer	100 or 100000	When METHOD is 0,1 or 3, the convex optimizer may return sub-optimal results after the maximum number of iterations. When METHOD is 2, if convergence is not reached after the maximum number of passes over training data, an error will be generated.	When MTHOD is 2, MAX_ITERATION defaults to 100000; Otherwise it defaults to 100.
EXIT_THRESHOLD	Double	1.0e-6 or 1.0e-7	Convergence threshold for exiting iterations.	When MTHOD is 2, EXIT_THRESHOLD defaults to 1.0E-7; Otherwise it defaults to 1.0E-6.
LBFGS_M	Integer	6	Number of past updates needed to be kept.	Only available when METHOD is 3.
THREAD_NUMBER	Integer	1	Number of threads.	
STAT_INF	Integer	0	<ul style="list-style-type: none"> • 0: Does not proceed statistical inference • 1: Proceeds statistical inference 	
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export logistic regression model in PMML. • 1: Exports logistic regression model in PMML in single row. • 2: Exports logistic regression model in PMML in several rows, and the minimum length of each row is 5000 characters. 	

Name	Data Type	Default Value	Description	Dependency
SELECTED_FEATURES	Varchar	No default value	A string to specify the features that will be processed. The pattern is " X_1, \dots, X_n ", where X_i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed.	Only used when you need to indicate the needed features.
DEPENDENT_VARIABLE	Varchar	No default value	Column name in the data table used as dependent variable.	Only used when you need to indicate the dependence.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer, varchar, or nvarchar	ID Note: If the SELECTED_FEATURES parameter is specified, the column data type must be varchar or nvarchar.
	2nd column	Integer or double	Value A_i <ul style="list-style-type: none"> • A_0: intercept • A_1: beta coefficient for X_1 • A_2: beta coefficient for X_2 • ...
	3rd column	Double	(When STAT_INF = 1) Zscore of each coefficient parameter
	4th column	Double	(When STAT_INF = 1) Pvalue of each coefficient parameter
Statistics (optional)	1st column	Varchar or nvarchar	Only supports AIC
	2nd column	Double	AIC value
PMML Result (logistic regression model)	1st column	Integer	ID

Table	Column	Column Data Type	Description
	2nd column	CLOB, varchar, or nvarchar	Logistic regression model in PMML format

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1: Fitting logistic regression model without penalties

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_LOGISTICR_DATA_T;
CREATE TYPE PAL_LOGISTICR_DATA_T AS TABLE("V1" VARCHAR (50), "V2" DOUBLE, "V3"
INTEGER,"CATEGORY" INTEGER);
DROP TYPE PAL_LOGISTICR_RESULT_T;
CREATE TYPE PAL_LOGISTICR_RESULT_T AS TABLE("Coefficient"
varchar(50), "CoefficientValue" DOUBLE, "ZSCORE" DOUBLE, "Pr(>|Z|)" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(50), "INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_LOGISTICR_STAT_T;
CREATE TYPE PAL_LOGISTICR_STAT_T AS TABLE( "STATISTICS" VARCHAR(20), "VALUE"
DOUBLE);
DROP TYPE PAL_LOGISTICR_PMMODEL_T;
CREATE TYPE PAL_LOGISTICR_PMMODEL_T AS TABLE( "ID" INTEGER, "PMMODEL"
VARCHAR(5000));
DROP table PAL_LOGISTICR_PDATA_TBL;
CREATE column_table PAL_LOGISTICR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_LOGISTICR_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_LOGISTICR_DATA_T','IN');
INSERT INTO PAL_LOGISTICR_PDATA_TBL VALUES (2,'DM_PAL', 'PAL_CONTROL_T','IN');
INSERT INTO PAL_LOGISTICR_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_LOGISTICR_RESULT_T','OUT');
INSERT INTO PAL_LOGISTICR_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_LOGISTICR_STAT_T','OUT');
INSERT INTO PAL_LOGISTICR_PDATA_TBL VALUES (5,'DM_PAL',
'PAL_LOGISTICR_PMMODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_LOGISTICR_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'LOGISTICREGRESSION',
'DM_PAL', 'PAL_LOGISTICR_PROC',PAL_LOGISTICR_PDATA_TBL);
DROP TABLE PAL_LOGISTICR_DATA_TBL;
CREATE COLUMN TABLE PAL_LOGISTICR_DATA_TBL LIKE PAL_LOGISTICR_DATA_T;
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',2.62,0,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',2.875,0,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',2.32,1,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',3.215,2,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',3.44,3,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',3.46,0,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',3.57,1,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',3.19,2,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',3.15,3,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',3.44,0,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',3.44,1,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',4.07,3,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',3.73,1,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',3.78,2,0);

```

```

INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',5.25,2,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',5.424,3,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',5.345,0,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',2.2,1,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',1.615,2,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',1.835,0,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',2.465,3,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',3.52,1,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',3.435,0,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',3.84,2,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',3.845,3,0);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',1.935,1,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',2.14,0,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',1.513,1,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',3.17,3,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',2.77,0,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('B',3.57,0,1);
INSERT INTO PAL_LOGISTICR_DATA_TBL VALUES ('A',2.78,3,1);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(50),
"INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('METHOD', 0, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXIT_THRESHOLD', null, 0.000001, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 8, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 1000, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_COL', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('STAT_INF', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT', 1, null, null);
DROP TABLE PAL_LOGISTICR_RESULTS_TBL;
CREATE COLUMN TABLE PAL_LOGISTICR_RESULTS_TBL LIKE PAL_LOGISTICR_RESULT_T;
DROP TABLE PAL_LOGISTICR_STAT_TBL;
CREATE COLUMN TABLE PAL_LOGISTICR_STAT_TBL LIKE PAL_LOGISTICR_STAT_T;
DROP TABLE PAL_LOGISTICR_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_LOGISTICR_PMMODEL_TBL LIKE PAL_LOGISTICR_PMMODEL_T;
CALL DM_PAL.PAL_LOGISTICR_PROC(PAL_LOGISTICR_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_LOGISTICR_RESULTS_TBL, PAL_LOGISTICR_STAT_TBL, PAL_LOGISTICR_PMMODEL_TBL)
WITH OVERVIEW;
SELECT * FROM PAL_LOGISTICR_RESULTS_TBL;
SELECT * FROM PAL_LOGISTICR_STAT_TBL;
SELECT * FROM PAL_LOGISTICR_PMMODEL_TBL;

```

Expected Result

PAL_LOGISTICR_RESULTS_TBL:

Coefficient	CoefficientValue	ZSCORE	Pr(> Z)
PAL_INTERCEPT	15.579880883789475	2.6208775928069477	0.008770374619226473
V1_PAL_DELIMIT_B	0	?	?
V1_PAL_DELIMIT_A	1.4649032807603797	0.8545979551916913	0.3927737600161363
V2	-4.819739813293363	-2.623421944925517	0.008705138919993693
V3_PAL_DELIMIT_0	0	?	?
V3_PAL_DELIMIT_1	-2.7941390516543...	-1.164522321158132	0.24421240297560054
V3_PAL_DELIMIT_2	-4.807857664089216	-1.249381612685957	0.2115255304033754
V3_PAL_DELIMIT_3	-2.7809183868777...	-1.4030416392628...	0.16060442254097484

PAL_LOGISTICR_STAT_TBL:

STATISTICS	VALUE
AIC	26.832334767429185

PAL_LOGISTICR_PMMODEL_TBL:

ID	PMMODEL
1 <PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ><Header copyright="SAP" ><Application name="PAL" version="...">	

Example 2: Fitting logistic regression model with elastic net penalties

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ENET_LOGISTICR_DATA_T;
CREATE TYPE PAL_ENET_LOGISTICR_DATA_T AS TABLE("V1" DOUBLE, "V2" INTEGER, "V3" DOUBLE, "V4" INTEGER, "V5" INTEGER, "V6" VARCHAR(50), "CATEGORY" INTEGER);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(50), "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_ENET_LOGISTICR_RESULT_T;
CREATE TYPE PAL_ENET_LOGISTICR_RESULT_T AS TABLE("Coefficient" varchar(50), "CoefficientValue" DOUBLE);
DROP TYPE PAL_ENET_LOGISTICR_STAT_T;
CREATE TYPE PAL_ENET_LOGISTICR_STAT_T AS TABLE( "STATISTICS" VARCHAR(20), "VALUE" DOUBLE);
DROP TYPE PAL_ENET_LOGISTICR_PDATA_TBL;
CREATE TYPE PAL_ENET_LOGISTICR_PMMODEL_T AS TABLE( "ID" INTEGER, "PMMODEL" VARCHAR(5000));
DROP TABLE PAL_ENET_LOGISTICR_PDATA_TBL;
CREATE COLUMN TABLE PAL_ENET_LOGISTICR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ENET_LOGISTICR_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_ENET_LOGISTICR_DATA_T', 'IN');
INSERT INTO PAL_ENET_LOGISTICR_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T',
'IN');
INSERT INTO PAL_ENET_LOGISTICR_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_ENET_LOGISTICR_RESULT_T', 'OUT');
INSERT INTO PAL_ENET_LOGISTICR_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_ENET_LOGISTICR_STAT_T', 'OUT');
INSERT INTO PAL_ENET_LOGISTICR_PDATA_TBL VALUES (5, 'DM_PAL',
'PAL_ENET_LOGISTICR_PMMODEL_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_ENET_LOGISTICR_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'LOGISTICREGRESSION',
'DM_PAL', 'PAL_ENET_LOGISTICR_PROC', PAL_ENET_LOGISTICR_PDATA_TBL);
DROP TABLE PAL_ENET_LOGISTICR_DATA_TBL;
CREATE COLUMN TABLE PAL_ENET_LOGISTICR_DATA_TBL LIKE PAL_ENET_LOGISTICR_DATA_T;
INSERT INTO PAL_ENET_LOGISTICR_DATA_TBL VALUES (4,1, 0.86,0,0, 'blue', 1);
INSERT INTO PAL_ENET_LOGISTICR_DATA_TBL VALUES (8,0, 0.45,1,2, 'blue', 1);
INSERT INTO PAL_ENET_LOGISTICR_DATA_TBL VALUES (7,1, 0.99,1,2, 'yellow', 0);
INSERT INTO PAL_ENET_LOGISTICR_DATA_TBL VALUES (12,1, 0.84,2,2, 'red', 1);
INSERT INTO PAL_ENET_LOGISTICR_DATA_TBL VALUES (6,1, 0.85,2,2, 'red', 0);
INSERT INTO PAL_ENET_LOGISTICR_DATA_TBL VALUES (9,0, 0.67,3,3, 'yellow', 0);
INSERT INTO PAL_ENET_LOGISTICR_DATA_TBL VALUES (10,1, 0.91,2,2, 'yellow', 0);
INSERT INTO PAL_ENET_LOGISTICR_DATA_TBL VALUES (14,0, 0.29,0,0, 'red', 1);
INSERT INTO PAL_ENET_LOGISTICR_DATA_TBL VALUES (7,0, 0.88, 1,0, 'yellow', 1);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(50),
"INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('METHOD', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_COL', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_COL', 3, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_COL', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ENET_LAMBDA', null, 0.03613, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ENET_ALPHA', null, 0.7, null);
DROP TABLE PAL_ENET_LOGISTICR_RESULTS_TBL;
CREATE COLUMN TABLE PAL_ENET_LOGISTICR_RESULTS_TBL LIKE
PAL_ENET_LOGISTICR_RESULT_T;
DROP TABLE PAL_ENET_LOGISTICR_STAT_TBL;
CREATE COLUMN TABLE PAL_ENET_LOGISTICR_STAT_TBL LIKE PAL_ENET_LOGISTICR_STAT_T;
DROP TABLE PAL_ENET_LOGISTICR_PMMODEL_TBL;

```

```

CREATE COLUMN TABLE PAL_ENET_LOGISTICR_PMMODEL_TBL LIKE
PAL_ENET_LOGISTICR_PMMODEL_T;
CALL DM_PAL.PAL_ENET_LOGISTICR_PROC(PAL_ENET_LOGISTICR_DATA_TBL,
"#PAL_CONTROL_TBL", PAL_ENET_LOGISTICR_RESULTS_TBL, PAL_ENET_LOGISTICR_STAT_TBL,
PAL_ENET_LOGISTICR_PMMODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ENET_LOGISTICR_RESULTS_TBL;
SELECT * FROM PAL_ENET_LOGISTICR_STAT_TBL;
SELECT * FROM PAL_ENET_LOGISTICR_PMMODEL_TBL;

```

Expected Results

PAL_ENET_LOGISTICR_RESULTS_TBL:

Coefficient	CoefficientValue
_PAL_INTERCEPT_	0.8091202301150909
V1	0.3063841008369403
V2_PAL_DELIMIT_1	0
V2_PAL_DELIMIT_0	1.3252547817581035
V3	0
V4_PAL_DELIMIT_0	0
V4_PAL_DELIMIT_1	0.06806660315285...
V4_PAL_DELIMIT_2	-1.1371289950300...
V4_PAL_DELIMIT_3	-2.2918113790925...
V5_PAL_DELIMIT_0	0
V5_PAL_DELIMIT_2	-2.351076677243712
V5_PAL_DELIMIT_3	-2.2870574119158...
V6_PAL_DELIMIT_blue	0
V6_PAL_DELIMIT_yellow	-2.5861015507446...
V6_PAL_DELIMIT_red	0

PAL_ENET_LOGISTICR_STAT_TBL:

STATISTICS	VALUE
AIC	20.782694610073833

PAL_ENET_LOGISTICR_PMMODEL_TBL:

ID	PMMODEL
1	<PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ><Header copyright="SAP" ><Application name="PAL" version="...">

FORECASTWITHLOGISTICR

This function performs predication with logistic regression result.

Procedure Generation

```

CALL SYS.AFLLANG_WRAPPER PROCEDURE_CREATE ('AFLPAL', 'FORECASTWITHLOGISTICR',
'<schema_name>', '<procedure_name>', <signature_table>);

```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Coefficient INPUT table type>	IN
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <parameter table>,
<coefficient input table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Predictive Data	1st column	Integer	ID
	Other columns	Integer, double, varchar, or nvarchar	Independent variables
Coefficient	1st column	Integer, varchar, or nvarchar	ID
	2nd column	Integer, double, varchar, nvarchar, or CLOB	Coefficients or PMML model. Note: Varchar, nvarchar, and CLOB types are only valid for PMML model.

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description	Dependency
CATEGORY_COL	Integer	Indicates whether the integer type column holds discrete independent variables. By default, varchar or nvarchar type column holds discrete independent variables and integer or double column holds continuous independent variables.	
CLASS_MAP0	Varchar	The same value as LOGISTIC_REGRESSION's parameter.	Only valid when the column type of dependent variable is varchar or nvarchar.
CLASS_MAP1	Varchar	The same value as LOGISTIC_REGRESSION's parameter.	Only valid when the column type of dependent variable is varchar or nvarchar.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
THREAD_NUMBER	Integer	1	Number of threads	
MODEL_FORMAT	Integer	0	Specifies model format. • 0: coefficients in table • 1: PMML format	Must be set to 1 when CATEGORY model is used.

Output Table

Table	Column	Column Data Type	Description
Fitted Result	1st column	Integer	ID
	2nd column	Integer or double	Value Y_i
	3rd column	Integer, varchar, or nvarchar	Category

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and

- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_FLOGISTICR_PREDICT_T;
CREATE TYPE PAL_FLOGISTICR_PREDICT_T AS TABLE("ID" INTEGER, "V1"
VARCHAR(5000), "V2" DOUBLE, "V3" INTEGER);
DROP TYPE PAL_FLOGISTICR_COEFFICIENT_T;
CREATE TYPE PAL_FLOGISTICR_COEFFICIENT_T AS TABLE("ID" INTEGER, "PMMLModel"
VARCHAR(5000));
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(50), "INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_FLOGISTICR_FITTED_T;
CREATE TYPE PAL_FLOGISTICR_FITTED_T AS TABLE("ID" INTEGER, "Fitted" DOUBLE, "TYPE"
INTEGER);
DROP table PAL_FLOGISTICR_PDATA_TBL;
CREATE column_table PAL_FLOGISTICR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_FLOGISTICR_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_FLOGISTICR_PREDICT_T','IN');
INSERT INTO PAL_FLOGISTICR_PDATA_TBL VALUES (2,'DM_PAL', 'PAL_CONTROL_T','IN');
INSERT INTO PAL_FLOGISTICR_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_FLOGISTICR_COEFFICIENT_T','IN');
INSERT INTO PAL_FLOGISTICR_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_FLOGISTICR_FITTED_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL',
'PAL_FORECASTWITHLOGISTICR_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','FORECASTWITHLOGISTICR',
'DM_PAL', 'PAL_FORECASTWITHLOGISTICR_PROC',PAL_FLOGISTICR_PDATA_TBL);
DROP TABLE PAL_FLOGISTICR_PREDICTDATA_TBL;
CREATE COLUMN TABLE PAL_FLOGISTICR_PREDICTDATA_TBL LIKE PAL_FLOGISTICR_PREDICT_T;
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (0,'B',2.62,0);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (1,'B',2.875,0);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (2,'A',2.32,1);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (3,'A',3.215,2);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (4,'B',3.44,3);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (5,'B',3.46,0);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (6,'A',3.57,1);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (7,'B',3.19,2);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (8,'A',3.15,3);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (9,'B',3.44,0);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (10,'B',3.44,1);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (11,'A',4.07,3);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (12,'A',3.73,1);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (13,'B',3.78,2);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (14,'B',5.25,2);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (15,'A',5.424,3);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (16,'A',5.345,0);
INSERT INTO PAL_FLOGISTICR_PREDICTDATA_TBL VALUES (17,'B',2.2,1);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(50),
"INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CATEGORY_COL',3,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MODEL_FORMAT',1,null,null);
DROP TABLE PAL_FLOGISTICR_COEFFICIENT_TBL;
CREATE COLUMN TABLE PAL_FLOGISTICR_COEFFICIENT_TBL LIKE
PAL_FLOGISTICR_COEFFICIENT_T;
INSERT INTO PAL_FLOGISTICR_COEFFICIENT_TBL VALUES (0, ' <PMML version="4.0"
xmlns="http://www.dmg.org/PMML-4_0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" >
<Header copyright="SAP" >
<Application name="PAL" version="1.0" />
</Header>

```

```

<DataDictionary numberOfFields="4" >
<DataField name="V1" optype="categorical" dataType="string" >
<Value value="B" />
<Value value="A" />
</DataField>
<DataField name="V2" optype="continuous" dataType="double" />
<DataField name="V3" optype="categorical" dataType="integer" >
<Value value="0" />
<Value value="1" />
<Value value="2" />
<Value value="3" />
</DataField>
<DataField name="CATEGORY" optype="continuous" dataType="integer" />
</DataDictionary>
<RegressionModel modelName="Instance for regression" functionName="regression"
algorithmName="LogisticRegression" targetFieldName="CATEGORY" >
<MiningSchema>
<MiningField name="V1" usageType="active" />
<MiningField name="V2" usageType="active" />
<MiningField name="V3" usageType="active" />
<MiningField name="CATEGORY" usageType="predicted" />
</MiningSchema>
<RegressionTable targetCategory="0" intercept="15.5798">
<NumericPredictor name="V2" exponent="1" coefficient="-4.81973"/>
<CategoricalPredictor name="V1" value="B" coefficient="0"/>
<CategoricalPredictor name="V1" value="A" coefficient="1.4649"/>
<CategoricalPredictor name="V3" value="0" coefficient="0"/>
<CategoricalPredictor name="V3" value="1" coefficient="-2.79413"/>
<CategoricalPredictor name="V3" value="2" coefficient="-4.80785"/>
<CategoricalPredictor name="V3" value="3" coefficient="-2.78091"/>
</RegressionTable>
<RegressionTable targetCategory="1" intercept="0"/>
</RegressionModel>
</PMML>');
DROP TABLE PAL_FLOGISTICR_FITTED_TBL;
CREATE COLUMN TABLE PAL_FLOGISTICR_FITTED_TBL LIKE PAL_FLOGISTICR_FITTED_T;
CALL DM_PAL.PAL_FORECASTWITHLOGISTICR_PROC(PAL_FLOGISTICR_PREDICTDATA_TBL,
"#PAL_CONTROL_TBL", PAL_FLOGISTICR_COEEFICIENT_TBL, PAL_FLOGISTICR_FITTED_TBL)
WITH OVERVIEW;
SELECT * FROM PAL_FLOGISTICR_FITTED_TBL;

```

Expected Result

PAL_FLOGISTICR_FITTED_TBL:

ID	Fitted	TYPE
0	0.9503664768538536	1
1	0.8485338101359745	1
2	0.9555899541051046	1
3	0.0370217909311029	0
4	0.022293255152265937	0
5	0.25041495841654465	0
6	0.0494625473619361	0
7	0.009922968787010407	0
8	0.2853042037029366	0
9	0.2689403203445937	0
10	0.022006920968331595	0
11	0.004714147927172445	0
12	0.023500094755269794	0
13	0.0005830949822641085	0
14	4.886616600609822E-7	0
15	0.000006938695990520421	0
16	0.00016379840700718923	0
17	0.8986516608622442	1

Related Information

[Multi-Class Logistic Regression \[page 176\]](#)

3.2.9 Multi-Class Logistic Regression

In many business scenarios we want to train a classifier with more than two classes. Multi-class logistic regression (also referred to as multi-nomial logistic regression) extends binary logistic regression algorithm (two classes) to multi-class cases.

The inputs and outputs of multi-class logistic regression are similar to those of logistic regression. In the training phase, the inputs are features and labels of the samples in the training set, and the outputs are some vectors. In the testing phase, the inputs are features of the samples in the testing set and the output of the training phase, and the outputs are the labels of the samples in the testing set.

Algorithms

Let P , K be the number of features and number of labels, respectively.

Training Phase

In the training set, let N be the number of samples, $X \in \mathbb{R}^{N \times P}$ be the features where $x_{i,p}$ is the p-th feature of the i-th sample, and $y \in \mathbb{R}^N$ be the labels where $y_i \in \{1, 2, \dots, K\}$ is the label of the i-th sample. The output of the training phase is denoted as $w^* \in \mathbb{R}^{(P+1) \times K}$, where $w_{p,k}^*$ ($p \leq P$) corresponds the weight of the p-th feature for the k-th class, and $w_{p+1,k}^*$ corresponds the constant for the k-th class. w^* is obtained by solving the following optimization problem:

$$w^* = \arg \max_w \prod_i \frac{\sum_k 1(y_i = k) \exp(x_{i,p} w_{p,k} + w_{p+1,k})}{\sum_k \exp(x_{i,p} w_{p,k} + w_{p+1,k})}$$

s.t. $w_{p,K} = 0$

Testing Phase

In the testing set, let \tilde{N} be the number of samples, $\tilde{X} \in \mathbb{R}^{N \times P}$ be the features where $\tilde{x}_{i,p}$ is the p-th feature of the i-th sample. Let $\tilde{y} \in \mathbb{R}^{\tilde{N}}$ be the unknown labels where $\tilde{y}_i \in \{1, 2, \dots, K\}$ is the label of the i-th sample, and \tilde{c}_i be the prediction confidences of the prediction where \tilde{c}_i is the confidence (likelihood) of the i-th sample. \tilde{y}, \tilde{c} are computed as follows,

$$\tilde{y}_i = \arg \max_k \exp(\tilde{x}_{i,p} w_{p,k} + w_{p+1,k})$$

$$\tilde{c}_i = \frac{\exp(\tilde{x}_{i,p} w_{p,\tilde{y}_i} + w_{p+1,\tilde{y}_i})}{\sum_k \exp(\tilde{x}_{i,p} w_{p,k} + w_{p+1,k})}$$

LRMCTR

This is the algorithm for the training phase.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LRMCTR', '<schema_name>',
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<DATA table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<MODEL table type>	OUT
4	<schema_name>	<PMML table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data table>, <parameter table>, <model table>, <PMML table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column Index	Column Data Type	Description
Data	1, 2, ..., P	Integer, varchar, nvarchar, or double	Features, X
	P+1	Varchar or nvarchar	Label, y

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
MAX_ITERATION	Integer	100	Maximum number of iterations of the optimization.	

Name	Data Type	Default Value	Description	Dependency
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export logistic regression model in PMML. • 1: Exports logistic regression model in PMML. 	
SELECTED_FEATURES	Varchar	No default value	A string to specify the features that will be processed. The pattern is "X ₁X _n ", where X _i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed.	Only used when you need to indicate the needed features.
DEPENDENT_VARIABLE	Varchar	No default value	Column name in the data table used as dependent variable.	Only used when you need to indicate the dependence.
CATEGORY_COL	Integer	No default value	<p>Indicates whether the integer type column holds discrete independent variables.</p> <p>By default, varchar or nvarchar type column holds discrete independent variables and integer or double type column holds continuous independent variables.</p>	

Output Tables

Table	Column	Column Data Type	Description
Model	1st column	Integer, varchar, or nvarchar	Similar to p, corresponding to a feature. Note: If the SELECTED_FEATURES parameter is specified, the column data type must be varchar or nvarchar.
	2nd column	VARCHAR or nvarchar	Similar to k, corresponding to a label
	3rd column	Double	Wpk
PMML	1st column	Integer	ID
	2nd column	CLOB, varchar, or nvarchar	Model in PMML format

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_LRMCTR_DATA_T;
CREATE TYPE PAL_LRMCTR_DATA_T
    AS TABLE
    (
        "V1" VARCHAR (50),
        "V2" DOUBLE,
        "V3" INTEGER,
        "CATEGORY" INTEGER
    );
DROP TABLE PAL_LRMCTR_DATA_TBL;
CREATE COLUMN TABLE PAL_LRMCTR_DATA_TBL LIKE PAL_LRMCTR_DATA_T;
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',2.62,0,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',2.875,0,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',2.32,1,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',3.215,2,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',3.44,3,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',3.46,0,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',3.57,1,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',3.19,2,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',3.15,3,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',3.44,0,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',3.44,1,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',4.07,3,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',3.73,1,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',3.78,2,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',5.25,2,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',5.424,3,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',5.345,0,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',2.2,1,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',1.615,2,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',1.835,0,1);

```

```

INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',2.465,3,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',3.52,1,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',3.435,0,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',3.84,2,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',3.845,3,0);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',1.935,1,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',2.14,0,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',1.513,1,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',3.17,3,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',2.77,0,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('B',3.57,0,1);
INSERT INTO PAL_LRMCTR_DATA_TBL VALUES ('A',2.78,3,1);
DROP TYPE PAL_LRM_C_MODEL_T;
CREATE TYPE PAL_LRM_C_MODEL_T
    AS TABLE
    (
        "P" varchar(100),
        "K" INTEGER,
        "W" DOUBLE
    );
DROP TYPE PAL_LRM_C_PMM_T;
CREATE TYPE PAL_LRM_C_PMM_T
    AS TABLE
    (
        "ID" INT,
        "PMMLMODEL" VARCHAR(5000)
    );
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T
    AS TABLE
    (
        "Name" VARCHAR (50),
        "INTARGS" INTEGER,
        "DOUBLEARGS" DOUBLE,
        "STRINGARGS" VARCHAR (100)
    );
DROP table PAL_LRMCTR_INIT_TBL;
CREATE column table
    PAL_LRMCTR_INIT_TBL
    (
        "POSITION" INT,
        "SCHEMA_NAME" NVARCHAR(256),
        "TYPE_NAME" NVARCHAR(256),
        "PARAMETER_TYPE" VARCHAR(7)
    );
insert into PAL_LRMCTR_INIT_TBL values (1, 'DM_PAL', 'PAL_LRMCTR_DATA_T', 'IN');
insert into PAL_LRMCTR_INIT_TBL values (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
insert into PAL_LRMCTR_INIT_TBL values (3, 'DM_PAL', 'PAL_LRM_C_MODEL_T', 'OUT');
insert into PAL_LRMCTR_INIT_TBL values (4, 'DM_PAL', 'PAL_LRM_C_PMM_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_LRMCTR_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'LRMCTR', 'DM_PAL',
    'PAL_LRMCTR_PROC', 'PAL_LRMCTR_INIT_TBL');
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL like PAL_CONTROL_T;
INSERT INTO PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 500, null, null);
INSERT INTO PAL_CONTROL_TBL VALUES ('PMML_EXPORT', 1, null, null);
--INSERT INTO PAL_CONTROL_TBL VALUES ('SELECTED_FEATURES', null, null, 'V2,V3');
DROP TABLE PAL_LRM_C_MODEL_TBL;
CREATE COLUMN TABLE PAL_LRM_C_MODEL_TBL like PAL_LRM_C_MODEL_T;
DROP TABLE PAL_LRM_C_PMM_TBL;
CREATE COLUMN TABLE PAL_LRM_C_PMM_TBL like PAL_LRM_C_PMM_T;
CALL DM_PAL.PAL_LRMCTR_PROC(PAL_LRMCTR_DATA_TBL, PAL_CONTROL_TBL,
    PAL_LRM_C_MODEL_TBL, PAL_LRM_C_PMM_TBL) WITH overview;
SELECT * FROM PAL_LRMCTR_DATA_TBL;
SELECT * FROM PAL_CONTROL_TBL;
SELECT * FROM PAL_LRM_C_MODEL_TBL;
SELECT * FROM PAL_LRM_C_PMM_TBL;

```

Expected Result

PAL_LRMCTE_MODEL_TBL:

	P	K	W
1	V2	1	-4.846389688235673
2	V2	0	0
3	V3	1	-0.907471631768088
4	V3	0	0
5	V1_PAL_KEY_DELIMIT_A	1	4.73858631368325
6	V1_PAL_KEY_DELIMIT_A	0	0
7	V1_PAL_KEY_DELIMIT_B	1	5.848018735943367
8	V1_PAL_KEY_DELIMIT_B	0	0
9	_PAL_INTERCEPT_	1	10.5866050496266...
10	_PAL_INTERCEPT_	0	0

PAL_LRMCTE_PMML_TBL:

ID	PMMLMODEL
1	<PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.dmg.org/PMML-4_0 PMML-4_0.xsd">

LRMCTE

This is the algorithm for the testing phase.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LRMCTE', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<DATA table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<MODEL table type>	IN
4	<schema_name>	<RESULT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data table>, <parameter table>, <model table>, <result table>) with overview ;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Data Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	Sample ID
	Other columns (2, 3, ..., P+1)	Integer, varchar, nvarchar, or double	Features, X

Input Model Table

Non-PMML format:

Table	Column	Column Data Type	Description
Model	1st column	Integer, varchar, or nvarchar	Similar to p, corresponding to a feature
	2nd column	Varchar or nvarchar	Similar to k, corresponding to a label
	3rd column	Double	W _{pk}

PMML format:

Table	Column	Column Data Type	Description
Model	1st column	Integer	ID
	2nd column	CLOB, varchar, or nvarchar	Model in PMML format

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
MODEL_TYPE	Integer	1	<ul style="list-style-type: none"> • 1: Model table should be in non-PMML format • 2: Model table should be in PMML format
VERBOSE_OUTPUT	Integer	0	<ul style="list-style-type: none"> • 1: Outputs the probability of all label categories • 0: Outputs the category of the highest probability only

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	2nd column	Varchar or nvarchar	Predicted label for the sample
	3rd column	Double	Confidence for the prediction of the sample

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Prediction with non-PMML input:

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_LRMCTE_DATA_T;
CREATE TYPE PAL_LRMCTE_DATA_T
    AS TABLE
    (
        "ID" INTEGER,
        "V1" VARCHAR (50),
        "V2" DOUBLE,
        "V3" INTEGER
    );

DROP TYPE PAL_LRMCTE_RESULT_T;
CREATE TYPE PAL_LRMCTE_RESULT_T
    AS TABLE
    (
        "ID" INTEGER,
        "Y" varchar(100),
        "Lik" DOUBLE
    );
DROP table PAL_LRMCTE_INIT_TBL;

```

```

CREATE column table PAL_LRMCTE_INIT_TBL ("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
insert into PAL_LRMCTE_INIT_TBL values (1,'DM_PAL', 'PAL_LRMCTE_DATA_T','IN');
insert into PAL_LRMCTE_INIT_TBL values (2,'DM_PAL', 'PAL_CONTROL_T','IN');
insert into PAL_LRMCTE_INIT_TBL values (3,'DM_PAL', 'PAL_LRMCTE_MODEL_T','IN');
insert into PAL_LRMCTE_INIT_TBL values (4,'DM_PAL', 'PAL_LRMCTE_RESULT_T','OUT');
DROP TABLE PAL_LRMCTE_RESULT_TBL;
CREATE COLUMN TABLE PAL_LRMCTE_RESULT_TBL like PAL_LRMCTE_RESULT_T;
DROP TABLE PAL_LRMCTE_DATA_TBL;
CREATE COLUMN TABLE PAL_LRMCTE_DATA_TBL like PAL_LRMCTE_DATA_T;
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (0,'B',2.62,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (1,'B',2.875,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (2,'A',2.32,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (3,'A',3.215,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (4,'B',3.44,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (5,'B',3.46,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (6,'A',3.57,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (7,'B',3.19,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (8,'A',3.15,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (9,'B',3.44,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (10,'B',3.44,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (11,'A',4.07,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (12,'A',3.73,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (13,'B',3.78,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (14,'B',5.25,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (15,'A',5.424,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (16,'A',5.345,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (17,'B',2.2,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (18,'B',1.615,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (19,'A',1.835,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (20,'B',2.465,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (21,'A',3.52,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (22,'A',3.435,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (23,'B',3.84,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (24,'B',3.845,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (25,'A',1.935,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (26,'B',2.14,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (27,'B',1.513,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (28,'A',3.17,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (29,'B',2.77,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (30,'B',3.57,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (31,'A',2.78,3);
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL like PAL_CONTROL_T;
INSERT INTO PAL_CONTROL_TBL VALUES ('MODEL_TYPE', 1, null, null);
INSERT INTO PAL_CONTROL_TBL VALUES ('VERBOSE_OUTPUT', 0, null, null);
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_LRMCTE_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'LRMCTE', 'DM_PAL',
'PAL_LRMCTE_PROC', PAL_LRMCTE_INIT_TBL);
delete from PAL_LRMCTE_RESULT_TBL;
CALL DM_PAL.PAL_LRMCTE_PROC(PAL_LRMCTE_DATA_TBL, PAL_CONTROL_TBL,
PAL_LRMCTE_MODEL_TBL, PAL_LRMCTE_RESULT_TBL) with overview;
select * from PAL_LRMCTE_RESULT_TBL;

```

Expected Result

	ID	Y	Lik
1	0	1	0.9767308528049194
2	1	1	0.9242296435114745
3	2	1	0.9598477290100679
4	3	0	0.8880419512524992
5	4	0	0.9507045537803743
6	5	0	0.5827042429473995
7	6	0	0.9470443651488033
8	7	0	0.6985289397446098
9	8	0	0.9348291224435037
10	9	0	0.558963550431078
11	10	0	0.7584917657041215
12	11	0	0.9991935239560293
13	12	0	0.974896249703052
14	13	0	0.9758652664111728
15	14	0	0.9999800819710587
16	15	0	0.999998859498342
17	16	0	0.9999745498003659
18	17	1	0.9923479091745743
19	18	1	0.9988793903523769
20	19	1	0.9983934626247112
21	20	1	0.8539418548285488
22	21	0	0.9334896295164925
23	22	0	0.7895403871184126
24	23	0	0.9818444372597636
25	24	0	0.9927692784684913
26	25	1	0.9935676083201597
27	26	1	0.9976788064038103
28	27	1	0.9997239257392854
29	28	0	0.9404912421157877
30	29	1	0.95302942104371
31	30	0	0.7041184524795155
32	31	0	0.7047845792901893

Prediction with PMML input:

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_LRMCTE_DATA_T;
CREATE TYPE PAL_LRMCTE_DATA_T
AS TABLE
(
    "ID" INTEGER,
    "V1" VARCHAR (50),
    "V2" DOUBLE,
    "V3" INTEGER
);

DROP TYPE PAL_LRMCTE_RESULT_T;
CREATE TYPE PAL_LRMCTE_RESULT_T
AS TABLE
(
    "ID" INTEGER,
    "Y" varchar(100),
    "Lik" DOUBLE
);
DROP table PAL_LRMCTEPMMI_INIT_TBL;
CREATE column table PAL_LRMCTEPMMI_INIT_TBL ("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
insert into PAL_LRMCTEPMMI_INIT_TBL values (1,'DM_PAL',
'PAL_LRMCTE_DATA_T','IN');
insert into PAL_LRMCTEPMMI_INIT_TBL values (2,'DM_PAL', 'PAL_CONTROL_T','IN');
insert into PAL_LRMCTEPMMI_INIT_TBL values (3,'DM_PAL', 'PAL_LRMCTE_PMML_T','IN');
insert into PAL_LRMCTEPMMI_INIT_TBL values (4,'DM_PAL',
'PAL_LRMCTE_RESULT_T','OUT');
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL like PAL_CONTROL_T;
INSERT INTO PAL_CONTROL_TBL VALUES ('MODEL_TYPE', 2, null, null);
INSERT INTO PAL_CONTROL_TBL VALUES ('VERBOSE_OUTPUT', 0, null, null);
DROP TABLE PAL_LRMCTE_DATA_TBL;
CREATE COLUMN TABLE PAL_LRMCTE_DATA_TBL like PAL_LRMCTE_DATA_T;
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (0,'B',2.62,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (1,'B',2.875,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (2,'A',2.32,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (3,'A',3.215,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (4,'B',3.44,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (5,'B',3.46,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (6,'A',3.57,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (7,'B',3.19,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (8,'A',3.15,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (9,'B',3.44,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (10,'B',3.44,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (11,'A',4.07,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (12,'A',3.73,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (13,'B',3.78,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (14,'B',5.25,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (15,'A',5.424,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (16,'A',5.345,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (17,'B',2.2,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (18,'B',1.615,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (19,'A',1.835,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (20,'B',2.465,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (21,'A',3.52,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (22,'A',3.435,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (23,'B',3.84,2);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (24,'B',3.845,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (25,'A',1.935,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (26,'B',2.14,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (27,'B',1.513,1);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (28,'A',3.17,3);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (29,'B',2.77,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (30,'B',3.57,0);
INSERT INTO PAL_LRMCTE_DATA_TBL VALUES (31,'A',2.78,3);
```

```
CALL SYS.AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_LRMCTEPMMI_PROC');
CALL SYS.AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'LRMCTE', 'DM_PAL',
'PAL_LRMCTEPMMI_PROC', 'PAL_LRMCTEPMMI_INIT_TBL');
delete from PAL_LRMCTE_RESULT_TBL;
CALL DM_PAL.PAL_LRMCTEPMMI_PROC(PAL_LRMCTE_DATA_TBL, PAL_CONTROL_TBL,
PAL_LRMCTE_PMML_TBL, PAL_LRMCTE_RESULT_TBL) with overview;
select * from PAL_LRMCTE_RESULT_TBL;
```

Expected Result

	ID	Y	Lik
1	0	1	0.976731116390336
2	1	1	0.9242306286840494
3	2	1	0.9598481816670585
4	3	0	0.8880398585803004
5	4	0	0.9507035491045562
6	5	0	0.5826994440004032
7	6	0	0.9470431687528854
8	7	0	0.6985250684310242
9	8	0	0.9348278399931155
10	9	0	0.5589587328768818
11	10	0	0.7584880702207132
12	11	0	0.9991934998106231
13	12	0	0.9748956279308331
14	13	0	0.9758646988073268
15	14	0	0.999980081207378
16	15	0	0.999998859449207
17	16	0	0.9999745487716664
18	17	1	0.9923479711399859
19	18	1	0.9988793938497558
20	19	1	0.9983934729134211
21	20	1	0.8539433504309744
22	21	0	0.9334881784695978
23	22	0	0.7895367454298755
24	23	0	0.9818439972962708
25	24	0	0.9927690964138418
26	25	1	0.993567659545192
27	26	1	0.9976788224922459
28	27	1	0.9997239261545239
29	28	0	0.9404900531485291
30	29	1	0.9530300052526701
31	30	0	0.7041141187812346
32	31	0	0.7047809453918248

3.2.10 Naive Bayes

Naive Bayes is a classification algorithm based on Bayes theorem. It estimates the class-conditional probability by assuming that the attributes are conditionally independent of one another.

Given the class label y and a dependent feature vector x_1 through x_n , the conditional independence assumption can be formally stated as follows:

$$P(y|x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots, x_n | y)}{P(x_1, \dots, x_n)}$$

Using the naive independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y)$$

for all i , this relationship is simplified to

$$P(y|x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i | y)}{P(x_1, \dots, x_n)}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$\begin{aligned} P(y|x_1, \dots, x_n) &\propto P(y) \prod_{i=1}^n P(x_i | y) \\ \Rightarrow \hat{y} &= \arg \max_y P(y) \prod_{i=1}^n P(x_i | y) \end{aligned}$$

We can use Maximum a posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i | y)$. The former is then the relative frequency of class y in the training set.

The different Naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i | y)$.

For continuous attributes, the attribute data are fitted to a Gaussian distribution and get the $P(x_i | y)$.

For discrete attributes, the count number ratio is used as $P(x_i | y)$. However, if there are categories that did not occur in the training set, $P(x_i | y)$ will become 0, while the actual probability is merely small instead of 0. This will bring errors to the prediction. To handle this issue, PAL introduces Laplace smoothing. The $P(x_i | y)$ is then denoted as:

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d} \quad (i = 1, \dots, d),$$

This is a type of shrinkage estimator, as the resulting estimate is between the empirical estimate x_i / N , and the uniform probability $1/d$. $\alpha > 0$ is the smoothing parameter, also called Laplace control value in the following discussion.

Despite its simplicity, Naive Bayes works quite well in areas like document classification and spam filtering, and it only requires a small amount of training data to estimate the parameters necessary for classification.

The Naive Bayes algorithm in PAL includes two functions: NBCTRAIN for generating training model; and NBCPREDICT for making prediction based on the training model.

Prerequisites

- The input data is of any data type, but the last column cannot be of double type.
- The input data does not contain null value.

NBCTRAIN

This function reads input data and generates training model with the Naive Bayes algorithm.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'NBCTRAIN',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<training input table>, <parameter table>,
<result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Training / Historical Data	Other columns	Varchar, nvarchar, integer, or double	Attribute columns	Discrete value: integer, varchar, or nvarchar Continuous value: double
	Last column	Varchar, nvarchar, or integer	Class column	Discrete value: integer, varchar, or nvarchar

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads.
MODEL_FORMAT	Integer	0	Specifies model format. <ul style="list-style-type: none">• 0: Deserializes the tree model from JSON format.• Other values: Deserializes the tree model from PMML format.
IS_SPLIT_MODEL	Integer	0	Indicates whether to split the string of the model. <ul style="list-style-type: none">• 0: Does not split the model• Other value: Splits the model. The maximum length of each unit is 5000.
LAPLACE	Double	0	Enables or disables Laplace smoothing. <ul style="list-style-type: none">• 0: Disables Laplace smoothing• Positive value: Enables Laplace smoothing for discrete values <p>Note: The LAPLACE value is only stored by JSON format models. If the PMML format is chosen, you may need to set the LAPLACE value again in the predicting phase.</p>

Name	Data Type	Default Value	Description
DISCRETIZATION	Integer	0	<ul style="list-style-type: none"> • 0: Disables discretization. • Other values: Uses supervised discretization to all the continuous attributes.

Output Table

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer	ID	
	2nd column	Varchar or nvarchar	Model saved as a JSON string.	<p>The table must be a column table.</p> <p>The maximum length is 5000.</p> <p>If the maximum length of the model is predicted to exceed 5000, set IS_SPLIT_MODEL to a value not equal to 0.</p>

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_NBCTRAIN_TRAININGSET_T;
CREATE TYPE PAL_NBCTRAIN_TRAININGSET_T AS TABLE(
    "HomeOwner" VARCHAR (100),
    "MaritalStatus" VARCHAR (100),
    "AnnualIncome" DOUBLE,
    "DefaultedBorrower" VARCHAR (100)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);

```

```

DROP TYPE PAL_NBC_MODEL_T;
CREATE TYPE PAL_NBC_MODEL_T AS TABLE(
    "ID" INTEGER,
    "JSONSTRING" VARCHAR(5000)
);
DROP TABLE PAL_NBCTRAIN_PDATA_TBL;
CREATE COLUMN TABLE PAL_NBCTRAIN_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_NBCTRAIN_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_NBCTRAIN_TRAININGSET_T', 'IN');
INSERT INTO PAL_NBCTRAIN_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_NBCTRAIN_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_NBC_MODEL_T',
'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_NBCTRAIN_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'NBCTRAIN', 'DM_PAL',
'PAL_NBCTRAIN_PROC', PAL_NBCTRAIN_PDATA_TBL);
DROP TABLE PAL_NBCTRAIN_TRAININGSET_TBL;
CREATE COLUMN TABLE PAL_NBCTRAIN_TRAININGSET_TBL LIKE PAL_NBCTRAIN_TRAININGSET_T;
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('YES','Single',125,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('NO','Married',100,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('NO','Single',70,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('YES','Married',120,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('NO','Divorced',95,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('NO','Married',60,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('YES','Divorced',220,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('NO','Single',85,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('NO','Married',75,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('NO','Single',90,'YES');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD NUMBER',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('IS_SPLIT_MODEL',0,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LAPLACE', null,1.0,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MODEL_FORMAT', 1,null,null);
DROP TABLE NBC_PAL_NBC_MODEL_TBL;
CREATE COLUMN TABLE NBC_PAL_NBC_MODEL_TBL LIKE PAL_NBC_MODEL_T;
CALL "DM_PAL".PAL_NBCTRAIN_PROC(PAL_NBCTRAIN_TRAININGSET_TBL,
"#PAL_CONTROL_TBL", NBC_PAL_NBC_MODEL_TBL) with overview;
SELECT * FROM NBC_PAL_NBC_MODEL_TBL;

```

Expected Result

ID	JsonString
0	<PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0" xm...

NBCPREDICT

This function uses the training model generated by NBCTRAIN to make predictive analysis.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'NBCPREDICT',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Model INPUT table type>	IN
4	<schema_name>	<Result Output table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<predict input table>, <parameter table>,
  <model input table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Predicted Data	1st column	Integer	ID
	Other columns	Integer, varchar, nvarchar, or double	Data to be classified (predicted)
Predictive Model	1st column	Integer	ID
	2nd column	Varchar or nvarchar	JSON string predictive model

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads
LAPLACE	Double	0	<ul style="list-style-type: none"> Enables or disables Laplace smoothing. • 0: Disables Laplace smoothing. • Positive value: Enables Laplace smoothing for discrete values. <p>Note: Non-zero LAPLACE value is required if there exist discrete category values that only occur in the test set. The LAPLACE value you set here takes precedence over the values read from JSON models.</p>
MODEL_FORMAT	Integer	0	<ul style="list-style-type: none"> Specifies model format. • 0: Deserializes the tree model from JSON format. • Other values: Deserializes the tree model from PMML format.
VERBOSE_OUTPUT	Integer	0	<ul style="list-style-type: none"> • 1: Outputs the probability of all label categories. • 0: Outputs the category of the highest probability only.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	2nd column	Varchar or nvarchar	Predictive result
	3rd column (optional)	Double	Confidence for the prediction of the sample, which is a logarithmic value of the a-posterior probabilities.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_NBCPREDICT_PREDICTION_T;
CREATE TYPE PAL_NBCPREDICT_PREDICTION_T AS TABLE(
    "ID" INTEGER,
    "HomeOwner" VARCHAR(100),
    "MaritalStatus" VARCHAR(100),
    "AnnualIncome" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_NBC_MODEL_T;
CREATE TYPE PAL_NBC_MODEL_T AS TABLE(
    "ID" INTEGER,
    "JSONSTRING" VARCHAR(5000)
);
DROP TYPE PAL_NBCPREDICT_RESULT_T;
CREATE TYPE PAL_NBCPREDICT_RESULT_T AS TABLE(
    "ID" INTEGER,
    "CLASS" VARCHAR(100)
);
DROP TABLE PAL_NBCPREDICT_PDATA_TBL;
CREATE COLUMN TABLE PAL_NBCPREDICT_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_NBCPREDICT_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_NBCPREDICT_PREDICTION_T', 'IN');
INSERT INTO PAL_NBCPREDICT_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_NBCPREDICT_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_NBC_MODEL_T',
'IN');
INSERT INTO PAL_NBCPREDICT_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_NBCPREDICT_RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_NBCPREDICT_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'NBCPREDICT', 'DM_PAL',
'PAL_NBCPREDICT_PROC', 'PAL_NBCPREDICT_PDATA_TBL');
DROP TABLE PAL_NBCPREDICT_PREDICTION_TBL;
CREATE COLUMN TABLE PAL_NBCPREDICT_PREDICTION_TBL LIKE
PAL_NBCPREDICT_PREDICTION_T;
INSERT INTO PAL_NBCPREDICT_PREDICTION_TBL VALUES (0,'NO','Married',120);
INSERT INTO PAL_NBCPREDICT_PREDICTION_TBL VALUES (1,'YES','Married',180);
INSERT INTO PAL_NBCPREDICT_PREDICTION_TBL VALUES (2,'NO','Single',90);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
```

```

INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD NUMBER',1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LAPLACE', null,1.0,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MODEL FORMAT', 1,null,null);
DROP TABLE PAL_NBCPREDICT_RESULTS_TBL;
CREATE COLUMN TABLE PAL_NBCPREDICT_RESULTS_TBL LIKE PAL_NBCPREDICT_RESULT_T;
CALL "DM_PAL".PAL_NBCPREDICT_PROC(PAL_NBCPREDICT_PREDICTION_TBL,
"#PAL_CONTROL_TBL", NBC_PAL_NBC_MODEL_TBL, PAL_NBCPREDICT_RESULTS_TBL) with
overview;
SELECT * FROM PAL_NBCPREDICT_RESULTS_TBL;

```

Expected Result

When VERBOSE_OUTPUT is set to 0:

ID	Class
0	NO
1	NO
2	YES

When VERBOSE_OUTPUT is set to 1:

	ID	CLASS	LOG_PROB
1	0	NO	-6.572352631131612
2	0	YES	-23.74725227057226
3	1	NO	-7.602220663331096
4	1	YES	-169.13354663169...
5	2	NO	-7.133598534952933
6	2	YES	-4.64863998190415

3.2.11 Parameter Selection and Model Evaluation (PSME)

Parameter selection and model evaluation (PSME) is used to enable cross validation and parameter selection for some PAL functions.

To avoid over fitting, it is a common practice to take use of cross validation to evaluate model performance and perform model selection for the optimal parameters of the model. This algorithm is an envelope for different classification algorithms to provide automatic parameter selection and model evaluation facilities during training phase. Logistic regression, naive Bayes, support vector machine (SVM), and random forest are supported.

PSME

This function performs parameter selection and model evaluation for classification algorithms including logistic regression, naive Bayes, and support vector machine (SVM).

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'PSME', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Function name table type>	IN
2	<schema_name>	<Input table type>	IN
3	<schema_name>	<Parameter table type>	IN
4	<schema_name>	<Evaluation results table type>	OUT
5	<schema_name>	<Selected parameters table type>	OUT
6	<schema_name>	<Function specific output table type>	OUT
7	<schema_name>	<Function specific output table type>	OUT (optional)
8	<schema_name>	<Function specific output table type>	OUT (optional)
9	<schema_name>	<Function specific output table type>	OUT (optional)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<function name tables>, <input tables>,  
<parameter table>, <evaluation results table>, <selected parameters table>,  
<function specific output table>, ...) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Function Name	1st column	Integer	ID.

Table	Column	Column Data Type	Description
	2nd column	Varchar or nvarchar	<p>Function name.</p> <p>The function name table specifies a single wrapped classification function to use. Supported function names are:</p> <ul style="list-style-type: none"> • LOGISTICREGRESSION: Logistic Regression • NBCTRAIN: Naive Bayes • SVMTRAIN: Support Vector Machine • RANDOMFOREST: Random Forest
Data	1st column	Integer, varchar, or nvarchar	<p>ID.</p> <p>Every data must have a unique ID specified in the 1st column. Thus for wrapped function that does not require an ID column, this column can be added.</p>
	Other columns	Function specific	Function specific columns must be exactly the same as the format defined by the functions.

Parameter Table

Parameters supported by wrapped functions are also supported. Besides, there are some additional general parameters and parameters specific for wrapped functions.

General Parameters

Name	Data Type	Default Value	Description
PARAM_SELECTION	Integer	0	<p>Determines whether automatic parameter selection is enabled.</p> <ul style="list-style-type: none"> • 0: Disabled • 1: Enabled

Name	Data Type	Default Value	Description
PARAM_SELECTION_MEASURE	String	'ACCURACY'	<p>Indicates the criteria to select the optimal parameters.</p> <ul style="list-style-type: none"> • 'ACCURACY': Accuracy is used as measure • 'F1_SCORE': The F1 score is used as measure <p>Note: The 'F1_SCORE' option is not available for random forest.</p>
NR_FOLD	Integer	10 (for LOGISTICREGRESSION) 3 (for SVMTRAIN) No default value for RANDOMFOREST	<p>Specifies how many portions the training data will be divided into in cross validation. The value must be equal to or greater than 2.</p> <p>This is an optional parameter in LOGISTICREGRESSION or SVMTRAIN, but a mandatory parameter in NBCTRAIN. The parameter is not used in RANDOMFOREST.</p> <p>Note: SVMTRAIN already supports this parameter, so you do not need to specify this twice.</p>
EVALUATION_SEED	Integer	0	<p>Indicates the seed used to initialize the random number generator.</p> <ul style="list-style-type: none"> • 0: Uses the system time • Not 0: Uses the specified seed

Additional Parameters for LOGISTICREGRESSION

Name	Data Type	Description
MIN_LAMBDA	Double	(Mandatory) Lower bound of the searching range for ENET_LAMBDA. This is a mandatory parameter. The value must be equal to or greater than 0.

Name	Data Type	Description
LAMBDA_STEP	Double	(Mandatory) Step size for searching. This is a mandatory parameter. The value must be greater than 0.
MAX_LAMBDA	Double	(Mandatory) Upper bound of the searching range for ENET_LAMBDA. This is a mandatory parameter. The value must be greater than the value of MIN_LAMBDA.

Additional Parameters for NBCTRAIN

Name	Data Type	Default Value	Description
MIN_LAPLACE	Double	0	Lower bound of the Laplace value where the search for optimal Laplace value will start. The value must be equal to or greater than 0.
LAPLACE_STEP	Double	0.1	Step size for searching. The value must be greater than 0.
MAX_LAPLACE	Double	100	Upper bound of Laplace value where the search for optimal Laplace value will stop. The value must be greater than the value of MIN_LAPLACE.

Additional Parameters for RANDOMFOREST

Name	Data Type	Default Value	Description
TRY_NUM_MIN	Integer	1	Lower bound of the try number where the search for optimal try number will start. The value must be equal to or greater than 1.
TRY_NUM_STEP	Integer	1	Step size for searching. The value must be equal or greater than 1.

Name	Data Type	Default Value	Description
TRY_NUM_MAX	Integer	Variables number	Upper bound of try number where the search for optimal try number will stop. The value must be greater than or equal to the value of TRY_NUM_MIN and less than or equal to variable number of the data.

Output Tables

Table	Column	Column Data Type	Description
Evaluation Results	1st column	Integer	ID
	2nd column	Varchar or nvarchar	Evaluation measure name
	3rd column	Varchar or nvarchar	Evaluation measure content
Selected Parameters	1st column	Integer	ID
	2nd column	Varchar or nvarchar	Parameter name
	3rd column	Varchar or nvarchar	Parameter value
Function Specific Output Table(s)	Function specific	Function specific	Function specific

i Note

Serialized confusion matrix is output in the evaluation results table in JSON format as shown below. The "Count" array is filled by row. It may be split into more rows if the length exceeds 5000.

```
{
  ConfusionMatrix: {
    ActualCategories:[],
    Count:[],
    PredictedCategories:[]
  }
}
```

For example,

```
{"ConfusionMatrix": {"ActualCategories": ["setosa", "versicolor", "virginica"], "Count": [50, 0, 0, 0, 47, 3, 0, 4, 46], "PredictedCategories": ["setosa", "versicolor", "virginica"]}} represents
```

		Predicted Categories		
ActualCategories		setosa	versicolor	virginica
setosa	50	0	0	0
versicolor	0	47	3	0
virginica	0	4	46	0

Examples

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1: Logistic regression

```

SET SCHEMA DM_PAL;
DROP TYPE CV_LR_DATA_T;
CREATE TYPE CV_LR_DATA_T AS TABLE("ROWID" INTEGER, "FACTOR1" INT ,
    "FACTOR2" DOUBLE ,
    "FACTOR3" DOUBLE ,
    "FACTOR4" VARCHAR(20),
    "LABEL" INTEGER );
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(50), "INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE CV_LR_FUNC_T;
CREATE TYPE CV_LR_FUNC_T AS TABLE( "ID" INTEGER, "FUNCNAME" VARCHAR(100));
DROP TYPE CV_LR_RESULT_T;
CREATE TYPE CV_LR_RESULT_T AS TABLE("ID" INTEGER, "AI" DOUBLE, "Z_score" DOUBLE,
"P_value" DOUBLE);
DROP TYPE CV_LR_MODEL_T;
CREATE TYPE CV_LR_MODEL_T AS TABLE( "ID" INTEGER, "PMMLMODEL" VARCHAR(5000));
DROP TYPE CV_LR_EVALUATION_T;
CREATE TYPE CV_LR_EVALUATION_T AS TABLE( "ID" INTEGER, "NAME" VARCHAR(5000),
"CONTENT" VARCHAR(5000));
DROP TYPE CV_LR PARA_SEL_T;
CREATE TYPE CV_LR PARA_SEL_T AS TABLE( "ID" INTEGER, "NAME" VARCHAR(5000),
"CONTENT" VARCHAR(5000));
DROP TABLE CV_LR_PDATA_TBL;
CREATE COLUMN TABLE CV_LR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO CV_LR_PDATA_TBL VALUES (1,'DM_PAL', 'CV_LR_FUNC_T','IN');
INSERT INTO CV_LR_PDATA_TBL VALUES (2,'DM_PAL', 'CV_LR_DATA_T','IN');
INSERT INTO CV_LR_PDATA_TBL VALUES (3,'DM_PAL', 'PAL_CONTROL_T','IN');
INSERT INTO CV_LR_PDATA_TBL VALUES (4,'DM_PAL', 'CV_LR PARA SEL_T','OUT');
INSERT INTO CV_LR_PDATA_TBL VALUES (5,'DM_PAL', 'CV_LR EVALUATION_T','OUT');
INSERT INTO CV_LR_PDATA_TBL VALUES (6,'DM_PAL', 'CV_LR RESULT T','OUT');
INSERT INTO CV_LR_PDATA_TBL VALUES (7,'DM_PAL', 'CV_LR MODEL T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_CV_LR_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PSME', 'DM_PAL',
'PAL_CV_LR_PROC', CV_LR_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(50),
"INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TABLE CV_LR_FUNC_TBL;
```

```

CREATE COLUMN TABLE CV_LR_FUNC_TBL LIKE CV_LR_FUNC_T;
DROP TABLE CV_LR_DATA_TBL;
CREATE COLUMN TABLE CV_LR_DATA_TBL LIKE CV_LR_DATA_T;
DROP TABLE CV_LR_RESULT_TBL;
CREATE COLUMN TABLE CV_LR_RESULT_TBL LIKE CV_LR_RESULT_T;
DROP TABLE CV_LR_MODEL_TBL;
CREATE COLUMN TABLE CV_LR_MODEL_TBL LIKE CV_LR_MODEL_T;
DROP TABLE CV_LR_EVALUATION_TBL;
CREATE COLUMN TABLE CV_LR_EVALUATION_TBL LIKE CV_LR_EVALUATION_T;
DROP TABLE CV_LR_PARA_SEL_TBL;
CREATE COLUMN TABLE CV_LR_PARA_SEL_TBL LIKE CV_LR_PARA_SEL_T;
INSERT INTO #PAL_CONTROL_TBL VALUES ('METHOD', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('STAT_INF', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PARAM_SELECTION', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PARAM_SELECTION_MEASURE', null, null, 'F1_SCORE');
INSERT INTO #PAL_CONTROL_TBL VALUES ('NR_FOLD', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_LAMBDA', null, 0.01, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_LAMBDA', null, 0.03, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LAMBDA_STEP', null, 0.001, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EVALUATION_SEED', 1, null, null);
INSERT INTO CV_LR_FUNC_TBL VALUES (1, 'LOGISTICREGRESSION');
INSERT INTO CV_LR_DATA_TBL VALUES (1, 1, 3.4, 14.5, 'A', 0);
INSERT INTO CV_LR_DATA_TBL VALUES (2, 2, 3.5, 42.5, 'B', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (3, 2, 3.6, 41.5, 'B', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (4, 1, 3.7, 42.5, 'B', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (5, 2, 3.8, 41.5, 'B', 0);
INSERT INTO CV_LR_DATA_TBL VALUES (6, 1, 3.4, 14.5, 'B', 0);
INSERT INTO CV_LR_DATA_TBL VALUES (7, 2, 3.5, 42.5, 'B', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (8, 1, 3.6, 41.5, 'B', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (9, 1, 3.7, 42.5, 'B', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (10, 2, 3.8, 41.5, 'B', 0);
INSERT INTO CV_LR_DATA_TBL VALUES (11, 1, 3.4, 14.5, 'B', 0);
INSERT INTO CV_LR_DATA_TBL VALUES (12, 2, 3.5, 42.5, 'B', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (13, 1, 3.6, 41.5, 'A', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (14, 2, 5.7, 42.5, 'A', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (15, 2, 6.8, 41.5, 'A', 0);
INSERT INTO CV_LR_DATA_TBL VALUES (16, 1, 13.4, 1.5, 'C', 0);
INSERT INTO CV_LR_DATA_TBL VALUES (17, 1, 13.5, 2.5, 'C', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (18, 1, 13.6, 4.5, 'D', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (19, 1, 13.7, 4.5, 'D', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (20, 1, 13.8, 4.5, 'B', 0);
INSERT INTO CV_LR_DATA_TBL VALUES (21, 0, 12.4, 1.5, 'B', 0);
INSERT INTO CV_LR_DATA_TBL VALUES (22, 1, 136.5, 12.5, 'B', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (23, 0, 153.6, 42.5, 'A', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (24, 0, 133.7, 54.5, 'A', 1);
INSERT INTO CV_LR_DATA_TBL VALUES (25, 0, 123.8, 14.5, 'C', 0);
CALL DM_PAL.PAL_CV_LR_PROC(CV_LR_FUNC_TBL, CV_LR_DATA_TBL, "#PAL_CONTROL_TBL",
CV_LR_EVALUATION_TBL, CV_LR_PARA_SEL_TBL, CV_LR_RESULT_TBL, CV_LR_MODEL_TBL)
WITH OVERVIEW;
SELECT * FROM CV_LR_EVALUATION_TBL;
SELECT * FROM CV_LR_PARA_SEL_TBL;

```

Expected Results

CV_LR_EVALUATION_TBL:

	ID	NAME	CONTENT
1	0	ACCURACY	0.761904
2	1	F1_SCORE	0.739583
3	2	CONFUSIONMATRIX	{"ConfusionMatrix":{"ActualC...

CV_LR PARA_SEL_TBL:

	ID	NAME	CONTENT
1	0	ENET_LAMBDA	0.03

Example 2: Naive Bayes

```
SET SCHEMA DM_PAL;
DROP TYPE FUNC_T;
CREATE TYPE FUNC_T AS TABLE(
    "ID" INTEGER,
    "Function" VARCHAR(50)
);
DROP TYPE EVALUATION_RESULT_T;
CREATE TYPE EVALUATION_RESULT_T AS TABLE(
    "ID" INTEGER,
    "Name" VARCHAR(50),
    "Value" VARCHAR(5000)
);
DROP TYPE SELECTED_PARAMETER_T;
CREATE TYPE SELECTED_PARAMETER_T AS TABLE(
    "ID" INTEGER,
    "Name" VARCHAR(50),
    "Value" VARCHAR(50)
);
DROP TYPE PAL_NBCTRAIN_TRAININGSET_T;
CREATE TYPE PAL_NBCTRAIN_TRAININGSET_T AS TABLE(
    "ID" VARCHAR(10),
    "HomeOwner" VARCHAR(100),
    "MaritalStatus" VARCHAR(100),
    "AnnualIncome" DOUBLE,
    "DefaultedBorrower" VARCHAR(100)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_NBC_MODEL_T;
CREATE TYPE PAL_NBC_MODEL_T AS TABLE(
    "ID" INTEGER,
    "JSONSTRING" VARCHAR(5000)
);
DROP TABLE PAL_NBCTRAIN_PDATA_TBL;
CREATE COLUMN TABLE PAL_NBCTRAIN_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_NBCTRAIN_PDATA_TBL VALUES (1, 'DM_PAL', 'FUNC_T', 'IN');
INSERT INTO PAL_NBCTRAIN_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_NBCTRAIN_TRAININGSET_T', 'IN');
INSERT INTO PAL_NBCTRAIN_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_NBCTRAIN_PDATA_TBL VALUES (4, 'DM_PAL', 'EVALUATION_RESULT_T',
'OUT');
INSERT INTO PAL_NBCTRAIN_PDATA_TBL VALUES (5, 'DM_PAL', 'SELECTED_PARAMETER_T',
'OUT');
INSERT INTO PAL_NBCTRAIN_PDATA_TBL VALUES (6, 'DM_PAL', 'PAL_NBC_MODEL_T',
'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_PSME_NBC_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PSME', 'DM_PAL',
'PAL_PSME_NBC_PROC', PAL_NBCTRAIN_PDATA_TBL);
DROP TABLE FUNC_TBL;
```

```

CREATE COLUMN TABLE FUNC_TBL LIKE FUNC_T;
INSERT INTO FUNC_TBL VALUES (0, 'NBCTRAIN');
DROP TABLE PAL_NBCTRAIN_TRAININGSET_TBL;
CREATE COLUMN TABLE PAL_NBCTRAIN_TRAININGSET_TBL LIKE PAL_NBCTRAIN_TRAININGSET_T;
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('2','YES','Single',125,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('1','NO','Married',100,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('3','NO','Single',70,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('4','YES','Married',120,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('5','NO','Divorced',95,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('6','NO','Married',60,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('7','YES','Divorced',220,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('8','NO','Single',85,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('9','NO','Married',75,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('10','NO','Single',62,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('11','NO','Single',65,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('12','NO','Married',130,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('13','NO','Single',190,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('14','NO','Divorced',100,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('15','NO','Divorced',105,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('16','NO','Married',70,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('17','NO','Married',90,'NO');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('18','NO','Married',160,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('20','NO','Married',50,'YES');
INSERT INTO PAL_NBCTRAIN_TRAININGSET_TBL VALUES ('19','NO','Single',90,'NO');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('IS_SPLIT_MODEL',0,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PARAM_SELECTION', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PARAM_SELECTION_MEASURE',
null,null,'ACCURACY');
INSERT INTO #PAL_CONTROL_TBL VALUES ('NR_FOLD', 5,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_LAPLACE', null,0.0,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_LAPLACE', null,10.0,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LAPLACE_STEP', null,0.2,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MODEL_FORMAT', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EVALUATION_SEED', 33,null,null);
DROP TABLE EVALUATION_RESULT_TBL;
CREATE COLUMN TABLE EVALUATION_RESULT_TBL LIKE EVALUATION_RESULT_T;
DROP TABLE SELECTED_PARAMETER_TBL;
CREATE COLUMN TABLE SELECTED_PARAMETER_TBL LIKE SELECTED_PARAMETER_T;
DROP TABLE NBC_PAL_NBC_MODEL_TBL;
CREATE COLUMN TABLE NBC_PAL_NBC_MODEL_TBL LIKE PAL_NBC_MODEL_T;
CALL "DM_PAL".PAL_PSME_NBC_PROC(FUNC_TBL, PAL_NBCTRAIN_TRAININGSET_TBL,
"#PAL_CONTROL_TBL", EVALUATION_RESULT_TBL, SELECTED_PARAMETER_TBL,
NBC_PAL_NBC_MODEL_TBL) with overview;
SELECT * FROM EVALUATION_RESULT_TBL;
SELECT * FROM SELECTED_PARAMETER_TBL;

```

Expected Results

EVALUATION_RESULT_TBL:

	ID	Name	Value
1	0	ACCURACY	0.65
2	1	F1_SCORE	0.601139
3	2	CONFUSIONMATRIX	{"ConfusionMatrix":{"ActualCatego...

SELECTED_PARAMETER_TBL:

	ID	Name	Value
1	0	LAPLACE	0.2
2	1	DISCRETIZATION	1

Example 3: SVM

```

SET SCHEMA DM_PAL;
DROP TYPE FUNC_T;
CREATE TYPE FUNC_T AS TABLE("ID" INTEGER,"Function" VARCHAR(50));
DROP TYPE EVALUATION_RESULT_T;
CREATE TYPE EVALUATION_RESULT_T AS TABLE("ID" INTEGER,"Name" VARCHAR(50),"Value"
VARCHAR(5000));
DROP TYPE SELECTED_PARAMETER_T;
CREATE TYPE SELECTED_PARAMETER_T AS TABLE("ID" INTEGER,"Name"
VARCHAR(50),"Value" VARCHAR(50));
--prepare input training data table type--
DROP TYPE PAL_SVM_TRAININGSET_T;
CREATE TYPE PAL_SVM_TRAININGSET_T AS TABLE ( ID integer, VALUEEE double,
ATTRIBUTE1 double, ATTRIBUTE2 double, ATTRIBUTE3 double);
--prepare argument table type--
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE( NAME varchar(50), INT#PAL_CONTROL_TBL
integer, DOUBLE#PAL_CONTROL_TBL double, STRING#PAL_CONTROL_TBL varchar(100));
--prepare result table type--
DROP TYPE PAL_SVM_MODELPART1_T;
CREATE TYPE PAL_SVM_MODELPART1_T AS TABLE( ID varchar(50), VALUEEE double);
--prepare result table type--
DROP TYPE PAL_SVM_MODELPART2_T;
CREATE TYPE PAL_SVM_MODELPART2_T AS TABLE( ID integer, ALPHA double, ATTRIBUTE1
double, ATTRIBUTE2 double, ATTRIBUTE3 double);
----create PAL procedure for training----
DROP TABLE PAL_PSME_PDATA_TBL;
CREATE TABLE PAL_PSME_PDATA_TBL("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_PSME_PDATA_TBL VALUES (1,'DM_PAL','FUNC_T','IN');
INSERT INTO PAL_PSME_PDATA_TBL VALUES (2,'DM_PAL','PAL_SVM_TRAININGSET_T','IN');
INSERT INTO PAL_PSME_PDATA_TBL VALUES (3,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_PSME_PDATA_TBL VALUES (4,'DM_PAL','EVALUATION_RESULT_T','OUT');
INSERT INTO PAL_PSME_PDATA_TBL VALUES (5,'DM_PAL','SELECTED_PARAMETER_T','OUT');
INSERT INTO PAL_PSME_PDATA_TBL VALUES (6,'DM_PAL','PAL_SVM_MODELPART1_T','OUT');
INSERT INTO PAL_PSME_PDATA_TBL VALUES (7,'DM_PAL','PAL_SVM_MODELPART2_T','OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_PSME_SVM_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PSME', 'DM_PAL',
'PAL_PSME_SVM_PROC', PAL_PSME_PDATA_TBL);
--create input training data table--
DROP TABLE PAL_SVM_TRAININGSET_TBL;
CREATE COLUMN TABLE PAL_SVM_TRAININGSET_TBL ( ID integer, VALUEEE double,
ATTRIBUTE1 double, ATTRIBUTE2 double, ATTRIBUTE3 double);
DROP TABLE FUNC_TBL;
CREATE COLUMN TABLE FUNC_TBL LIKE FUNC_T;
INSERT INTO FUNC_TBL VALUES (0, 'SVMTRAIN');
--create training argument table--
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (NAME varchar(50),
INT#PAL_CONTROL_TBL integer, DOUBLE#PAL_CONTROL_TBL double,
STRING#PAL_CONTROL_TBL varchar(100));
--create result table--
DROP TABLE PAL_SVM_MODELPART1_TBL;
CREATE COLUMN TABLE PAL_SVM_MODELPART1_TBL( ID varchar(50), VALUEEE double);
--create result table--
DROP TABLE PAL_SVM_MODELPART2_TBL;
CREATE COLUMN TABLE PAL_SVM_MODELPART2_TBL( ID integer, ALPHA double, ATTRIBUTE1
double, ATTRIBUTE2 double, ATTRIBUTE3 double);

```

```

DROP TABLE EVALUATION_RESULT_TBL;
CREATE COLUMN TABLE EVALUATION_RESULT_TBL LIKE EVALUATION_RESULT_T;
DROP TABLE SELECTED_PARAMETER_TBL;
CREATE COLUMN TABLE SELECTED_PARAMETER_TBL LIKE SELECTED_PARAMETER_T;
---insert data into input training data table---
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(0,1,1,10,100);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(1,1,1.1,10.1,100);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(2,1,1.2,10.2,100);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(3,1,1.3,10.4,100);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(4,1,1.2,10.3,100);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(5,2,4,40,400);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(6,2,4.1,40.1,400);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(7,2,4.2,40.2,400);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(8,2,4.3,40.4,400);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(9,2,4.2,40.3,400);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(10,3,9,90,900);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(11,3,9.1,90.1,900);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(12,3,9.2,90.2,900);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(13,3,9.3,90.4,900);
INSERT INTO PAL_SVM_TRAININGSET_TBL VALUES(14,3,9.2,90.3,900);
---insert data into input training argument---
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('KERNEL_TYPE',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('TYPE',1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('PARAM_SELECTION',1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('NR_FOLD',5,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PARAM_SELECTION_MEASURE',
null,null,'ACCURACY');
INSERT INTO #PAL_CONTROL_TBL VALUES ('EVALUATION_SEED', 1, null, null);
CALL "DM_PAL".PAL_PSME_SVM_PROC(FUNC_TBL, PAL_SVM_TRAININGSET_TBL,
"#PAL_CONTROL_TBL", EVALUATION_RESULT_TBL,
SELECTED_PARAMETER_TBL,PAL_SVM_MODELPART1_TBL,PAL_SVM_MODELPART2_TBL ) with
overview;
--check the result--
SELECT * FROM EVALUATION_RESULT_TBL;
SELECT * FROM SELECTED_PARAMETER_TBL;

```

Expected Results

EVALUATION_RESULT_TBL:

	ID	Name	Value
1	0	ACCURACY	1
2	1	F1_SCORE	1
3	2	CONFUSIONMATRIX	{"ConfusionMatrix":{"ActualCategories":["1","2...}}

SELECTED_PARAMETER_TBL:

	ID	Name	Value
1	0	SVM_C	0.5
2	1	RBF_GAMMA	0.8

Example 4: Random Forest

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_FUNC_TYPE;
DROP TYPE PAL_DATA_TYPE;
DROP TYPE PAL_PARA_TYPE;
DROP TYPE PAL_EVAL_TYPE;
DROP TYPE PAL_PARASEL_TYPE;
DROP TYPE PAL_MODEL_TYPE;

```

```

DROP TYPE PAL_VARIMPL_TYPE;
DROP TYPE PAL_ERROR_TYPE;
DROP TYPE PAL_CONFUSION_TYPE;
DROP TABLE PAL_RFPS_PDATA_TBL;
DROP TABLE PAL_FUNC_TBL;
DROP TABLE PAL_DATA_TBL;
DROP TABLE PAL PARA_TBL;
DROP TABLE PAL_EVAL_TBL;
DROP TABLE PAL_PARASEL_TBL;
DROP TABLE PAL_MODEL_TBL;
DROP TABLE PAL_VARIMPL_TBL;
DROP TABLE PAL_ERROR_TBL;
DROP TABLE PAL_CONFUSION_TBL;
CREATE TYPE PAL_FUNC_TYPE AS TABLE("ID" INTEGER, "FUNCNAME" VARCHAR(5000));
CREATE TYPE PAL_DATA_TYPE AS TABLE("ID" INTEGER, "C0" VARCHAR(5000), "C1"
INTEGER, "C2" INTEGER, "C3" VARCHAR(5000), "C8" VARCHAR(5000));
CREATE TYPE PAL PARA_TYPE AS TABLE("NAME" VARCHAR(5000), "INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(5000));
CREATE TYPE PAL_EVAL_TYPE AS TABLE("ID" INTEGER, "NAME" VARCHAR(5000),
"CONTENT" VARCHAR(5000));
CREATE TYPE PAL_PARASEL_TYPE AS TABLE("ID" INTEGER, "NAME" VARCHAR(5000),
"CONTENT" VARCHAR(5000));
CREATE TYPE PAL_MODEL_TYPE AS TABLE("ID" INTEGER, "TREEID" INTEGER,
"JSON_STRING" VARCHAR(5000));
CREATE TYPE PAL_VARIMPL_TYPE AS TABLE("ID" VARCHAR(5000), "IMPVAL" DOUBLE);
CREATE TYPE PAL_ERROR_TYPE AS TABLE("ID" INTEGER, "ERRORTYPES" DOUBLE);
CREATE TYPE PAL_CONFUSION_TYPE AS TABLE("ID" INTEGER, "JSON_STRING"
VARCHAR(500));
CREATE COLUMN TABLE PAL_RFPS_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_RFPS_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_FUNC_TYPE', 'IN');
INSERT INTO PAL_RFPS_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_DATA_TYPE', 'IN');
INSERT INTO PAL_RFPS_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL PARA_TYPE', 'IN');
INSERT INTO PAL_RFPS_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_EVAL_TYPE', 'OUT');
INSERT INTO PAL_RFPS_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_PARASEL_TYPE', 'OUT');
INSERT INTO PAL_RFPS_PDATA_TBL VALUES (6, 'DM_PAL', 'PAL_MODEL_TYPE', 'OUT');
INSERT INTO PAL_RFPS_PDATA_TBL VALUES (7, 'DM_PAL', 'PAL_VARIMPL_TYPE', 'OUT');
INSERT INTO PAL_RFPS_PDATA_TBL VALUES (8, 'DM_PAL', 'PAL_ERROR_TYPE', 'OUT');
INSERT INTO PAL_RFPS_PDATA_TBL VALUES (9, 'DM_PAL', 'PAL_CONFUSION_TYPE', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_RFPS_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PSME', 'DM_PAL',
'PAL_RFPS_PROC', 'PAL_RFPS_PDATA_TBL');
CREATE COLUMN TABLE PAL_FUNC_TBL LIKE PAL_FUNC_TYPE;
CREATE COLUMN TABLE PAL_DATA_TBL LIKE PAL_DATA_TYPE;
CREATE COLUMN TABLE PAL PARA_TBL LIKE PAL PARA_TYPE;
CREATE COLUMN TABLE PAL_EVAL_TBL LIKE PAL_EVAL_TYPE;
CREATE COLUMN TABLE PAL_PARASEL_TBL LIKE PAL_PARASEL_TYPE;
CREATE COLUMN TABLE PAL_MODEL_TBL LIKE PAL_MODEL_TYPE;
CREATE COLUMN TABLE PAL_VARIMPL_TBL LIKE PAL_VARIMPL_TYPE;
CREATE COLUMN TABLE PAL_ERROR_TBL LIKE PAL_ERROR_TYPE;
CREATE COLUMN TABLE PAL_CONFUSION_TBL LIKE PAL_CONFUSION_TYPE;
INSERT INTO PAL_FUNC_TBL VALUES (1, 'RANDOMFOREST');
INSERT INTO PAL_DATA_TBL VALUES (1, 'sunny', 75, 70, 'YES', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (2, 'sunny', 76, 90, 'YES', 'NOT', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (3, 'sunny', 83, 85, 'NO', 'NOT', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (4, 'sunny', 79, 95, 'NO', 'NOT', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (5, 'sunny', 68, 70, 'NO', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (6, 'overcast', 78, 90, 'YES', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (7, 'overcast', 70, 78, 'NO', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (8, 'overcast', 69, 65, 'YES', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (9, 'overcast', 99, 75, 'NO', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (10, 'rain', 100, 80, 'YES', 'NOT', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (11, 'rain', 98, 70, 'YES', 'NOT', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (12, 'rain', 97, 80, 'NO', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (13, 'rain', 96, 80, 'NO', 'PLAY');
INSERT INTO PAL_DATA_TBL VALUES (14, 'rain', 95, 96, 'NO', 'PLAY');
INSERT INTO PAL PARA_TBL VALUES ('TRY_NUM_MIN', 1, null, null);
INSERT INTO PAL PARA_TBL VALUES ('TRY_NUM_STEP', 1, null, null);

```

```

INSERT INTO PAL PARA TBL VALUES('TRY_NUM_MAX', 4, null, null);
INSERT INTO PAL PARA TBL VALUES('SEED', 1, null, null);
INSERT INTO PAL PARA TBL VALUES('CONTINUOUS_COL', 2, null, null);
INSERT INTO PAL PARA TBL VALUES('CONTINUOUS_COL', 3, null, null);
INSERT INTO PAL PARA TBL VALUES('PARAM_SELECTION', 1, null, null);
CALL DM PAL.PAL_RFPS PROC(PAL FUNC TBL, PAL DATA TBL, PAL PARA TBL,
PAL EVAL TBL, PAL PARASEL TBL, PAL MODEL TBL, PAL VARIMPL TBL, PAL ERROR TBL,
PAL CONFUSION TBL) WITH OVERVIEW;
SELECT * FROM PAL EVAL TBL;
SELECT * FROM PAL PARASEL TBL;

```

Expected Results

PAL_EVAL_TBL:

ID	VALUEE
0	{"C":0.031250,"attr_number":3,"bias":[1.0],"categorical_attr_number":0,"class_number":2,"c...

PAL_PARASEL_TBL:

	ID	NAME	CONTENT
1	0	TRY_NUM	3

3.2.12 Predict with Tree Model

This algorithm uses tree model to perform scoring.

Prerequisites

- The input data must contain an ID column.

PREDICTWITHDT

Procedure Generation

```

CALL SYS.AFLLANG_WRAPPER PROCEDURE_CREATE ('AFLPAL', 'PREDICTWITHDT',
'<schema_name>', '<procedure_name>', <signature_table>);

```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN

Position	Schema Name	Table Type Name	Parameter Type
3	<schema_name>	<Tree Model INPUT table type>	IN
4	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <parameter table>, <tree model input table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar	Data fields	
Tree Model	1st column	Integer	ID	
	2nd column	VARCHAR or nvarchar	Tree model string	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
MODEL_FORMAT	Integer	0	<ul style="list-style-type: none"> • 0: Deserializes the tree model from JSON format. • 1: Deserializes the tree model from PMML format.
THREAD_NUMBER	Integer	1	Number of threads.

Name	Data Type	Default Value	Description
IS_OUTPUT_PROBABILITY	Integer	0	<ul style="list-style-type: none"> • 0: Does not output the probability in the result table. • 1: Outputs the probability in the result table. The result table will thus contain a third column to store the probability.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer, varchar, or nvarchar	ID
	2nd column	Varchar or nvarchar	Scoring result
	3rd column (optional)	Double	Add this column if you have set the IS_OUTPUT_PROBABILITY parameter to 1 to output the probability.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DT_SCORING_DATA_T;
CREATE TYPE PAL_DT_SCORING_DATA_T AS TABLE(
    "ID" INTEGER,
    "OUTLOOK" VARCHAR(20),
    "TEMP" INTEGER,
    "HUMIDITY" DOUBLE,
    "WINDY" VARCHAR(10)
);
DROP TYPE PAL_DT_SCORING_TREEMODEL_T;
CREATE TYPE PAL_DT_SCORING_TREEMODEL_T AS TABLE(
    "ID" INTEGER,
    "TREEMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_DT_SCORING_RESULT_T;
CREATE TYPE PAL_DT_SCORING_RESULT_T AS TABLE("ID" INTEGER, "SCORING"
VARCHAR(50), "PROB" DOUBLE);
DROP TABLE PAL_DT_SCORING_PDATA_TBL;
CREATE COLUMN TABLE PAL_DT_SCORING_PDATA_TBL (
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),

```

```

"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_DT_SCORING_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_DT_SCORING_DATA_T', 'IN');
INSERT INTO PAL_DT_SCORING_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_DT_SCORING_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_DT_SCORING TREEMODEL_T', 'IN');
INSERT INTO PAL_DT_SCORING_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_DT_SCORING_RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_DT_SCORING PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'PREDICTWITHDT', 'DM_PAL',
'PAL_DT_SCORING PROC', PAL_DT_SCORING_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('IS_OUTPUT_PROBABILITY', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MODEL_FORMAT', 1, null, null);
DROP TABLE PAL_DT_SCORING_DATA_TBL;
CREATE COLUMN TABLE PAL_DT_SCORING DATA_TBL LIKE PAL_DT_SCORING DATA_T;
INSERT INTO PAL_DT_SCORING_DATA_TBL VALUES (0, 'Overcast', 75, 70, 'Yes');
INSERT INTO PAL_DT_SCORING_DATA_TBL VALUES (1, 'Rain', 78, 70, 'Yes');
INSERT INTO PAL_DT_SCORING_DATA_TBL VALUES (2, 'Sunny', 66, 70, 'Yes');
INSERT INTO PAL_DT_SCORING_DATA_TBL VALUES (3, 'Sunny', 69, 70, 'Yes');
INSERT INTO PAL_DT_SCORING_DATA_TBL VALUES (4, 'Rain', null, 70, 'Yes');
INSERT INTO PAL_DT_SCORING_DATA_TBL VALUES (5, null, 70, 70, 'Yes');
INSERT INTO PAL_DT_SCORING DATA_TBL VALUES (6, '***', 70, 70, 'Yes');
DROP TABLE PAL_DT_SCORING_RESULT_TBL;
CREATE COLUMN TABLE PAL_DT_SCORING_RESULT_TBL LIKE PAL_DT_SCORING_RESULT_T;
CALL "DM_PAL".PAL_DT_SCORING_PROC(PAL_DT_SCORING_DATA_TBL, #PAL_CONTROL_TBL,
PAL_DT_SCORING TREEMODEL_TBL, PAL_DT_SCORING_RESULT_TBL) with OVERVIEW;
SELECT * FROM PAL_DT_SCORING_RESULT_TBL;

```

Expected Result

ID	SCORING	PROB
0	Play	1
1	Do not Play	1
2	Do not Play	1
3	Play	1
4	Play	0.5
5	Play	0.642857142...
6	Play	0.642857142...

3.2.13 Random Forest

The random forests algorithm is an ensemble learning method for classification or regression. It grows many CART decision trees and outputs the class (classification) that is voted by majority of individual trees or mean prediction (regression) of the individual trees.

The algorithm uses both bagging and random feature selection techniques. Each new training set is drawn with replacement from the original training set, and then a tree is grown on the new training set using random feature selection.

The random forest algorithm generates an internal unbiased estimate (out-of-bag error) of the generalization error as the forest building processes. It gives estimates of what variables are important from nodes' splitting process. It also has an effective method for estimating missing data. If the m_{th} variable is numerical, the method computes the median of all values of this variable in class j or computes the most frequent non-missing value in class j , and then it uses this value to replace all missing values of the m_{th} variable in class j .

Prerequisite

The target column of the training data must not have null values, and other columns should have at least one valid value (not null).

RANDOMFORESTTRAIN

This function is used for classification or regression.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'RANDOMFORESTTRAIN',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Training data INPUT table type>	IN
2	<schema_name>	<Parameter table type>	IN
3	<schema_name>	<Random forest model OUTPUT table type>	OUT
4	<schema_name>	<Variable importance OUTPUT table type>	OUT
5	<schema_name>	<Error rate OUTPUT table type>	OUT
5	<schema_name>	<Confusion matrix OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<training data input table>, <parameter table>, <random forest model output table>, <variable importance output table>, <error rate output table>, <confusion matrix output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Training Data	Columns	Varchar, nvarchar, double, or integer	Independent fields.	
	Last column	Varchar, nvarchar, double, or integer	Dependent field. The varchar type is used for classification, and double type is for regression. The integer type is by default used for classification, but if intentionally treated as continuous variable, then it is solved as regression.	Null values are not allowed.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
TREES_NUM	Integer	500	Specifies the number of trees in the random forest.
TRY_NUM	Integer	$\text{sqrt}(p)$ (for classification) or $p/3$ (for regression), where p is the number of independent variables	Specifies the number of randomly selected splitting variables.

Name	Data Type	Default Value	Description
CONTINUOUS_COL	Integer	Detected from input data	Indicates which columns are continuous attributes. The default behavior is: <ul style="list-style-type: none"> String or integer: categorical Double: continuous

Output Tables

Table	Column	Column Data Type	Description
Model	1st column	Integer	ID
	2nd column	Integer	Tree index
	3rd column	Varchar or nvarchar	Model content. The length should be 5000.
Variable Importance	1st column	Varchar or nvarchar	Independent field name
	2nd column	Double	Variable importance
Error Rate (for classification) or Mean Squared Error (for regression)	1st column	Integer	Tree index
	2nd column	Double	Out-of-bag error rate or mean squared error for random forest up to indexed tree
Confusion Matrix (for classification only)	1st column	Integer	ID
	2nd column	Varchar or nvarchar	Confusion matrix content

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_RF_DATA_T;
CREATE TYPE PAL_RF_DATA_T AS TABLE(
    "OUTLOOK" VARCHAR(20),
    "TEMP" DOUBLE,
    "HUMIDITY" DOUBLE,
    "WINDY" VARCHAR(10),
    "CLASS" VARCHAR(20)
```

```

);
DROP TYPE PAL_RF_MODEL_T;
CREATE TYPE PAL_RF_MODEL_T AS TABLE(
    "ID" INTEGER,
    "TREEINDEX" INTEGER,
    "MODEL" VARCHAR(5000)
);
DROP TYPE PAL_RF_VAR_IMP_T;
CREATE TYPE PAL_RF_VAR_IMP_T AS TABLE(
    "VAR" VARCHAR(100),
    "IMP" DOUBLE
);
DROP TYPE PAL_RF_ERR_RATE_T;
CREATE TYPE PAL_RF_ERR_RATE_T AS TABLE(
    "TREEINDEX" INTEGER,
    "ERR" DOUBLE
);
DROP TYPE PAL_RF_CONFUSION_T;
CREATE TYPE PAL_RF_CONFUSION_T AS TABLE(
    "ID" INTEGER,
    "CONTENT" VARCHAR(1000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_RF_PDATA_TBL;
CREATE COLUMN TABLE PAL_RF_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_RF_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_RF_DATA_T', 'in');
INSERT INTO PAL_RF_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'in');
INSERT INTO PAL_RF_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_RF_MODEL_T', 'out');
INSERT INTO PAL_RF_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_RF_VAR_IMP_T', 'out');
INSERT INTO PAL_RF_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_RF_ERR_RATE_T', 'out');
INSERT INTO PAL_RF_PDATA_TBL VALUES (6, 'DM_PAL', 'PAL_RF_CONFUSION_T', 'out');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_RF_TRAINING_PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'RANDOMFORESTTRAIN',
'DM_PAL', 'PAL_RF_TRAINING_PROC', PAL_RF_PDATA_TBL);
DROP TABLE PAL_RF_DATA_TBL;
CREATE COLUMN TABLE PAL_RF_DATA_TBL LIKE PAL_RF_DATA_T;
INSERT INTO PAL_RF_DATA_TBL VALUES ('Sunny', 75, 70, 'Yes', 'Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Sunny', null, 90, 'Yes', 'Do not Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Sunny', 85, null, 'No', 'Do not Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Sunny', 72, 95, 'No', 'Do not Play');
INSERT INTO PAL_RF_DATA_TBL VALUES (null, null, 70, null, 'Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Overcast', 72, 90, 'Yes', 'Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Overcast', 83, 78, 'No', 'Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Overcast', 64, 65, 'Yes', 'Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Overcast', 81, 75, 'No', 'Play');
INSERT INTO PAL_RF_DATA_TBL VALUES (null, 71, 80, 'Yes', 'Do not Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Rain', 65, 70, 'Yes', 'Do not Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Rain', 75, 80, 'No', 'Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Rain', 68, 80, 'No', 'Play');
INSERT INTO PAL_RF_DATA_TBL VALUES ('Rain', 70, 96, 'No', 'Play');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);

```

```

INSERT INTO #PAL_CONTROL_TBL VALUES ('TREES_NUM', 300, null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TRY_NUM', 3, null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEED', 2, null,null);
DROP TABLE PAL_RF_MODEL_TBL;
CREATE COLUMN TABLE PAL_RF_MODEL_TBL LIKE PAL_RF_MODEL_T;
DROP TABLE PAL_RF_VAR_IMP_TBL;
CREATE COLUMN TABLE PAL_RF_VAR_IMP_TBL LIKE PAL_RF_VAR_IMP_T;
DROP TABLE PAL_RF_ERR_RATE_TBL;
CREATE COLUMN TABLE PAL_RF_ERR_RATE_TBL LIKE PAL_RF_ERR_RATE_T;
DROP TABLE PAL_RF_CONFUSION_TBL;
CREATE COLUMN TABLE PAL_RF_CONFUSION_TBL LIKE PAL_RF_CONFUSION_T;
CALL "DM_PAL".PAL_RF_TRAINING_PROC(PAL_RF_DATA_TBL, #PAL_CONTROL_TBL,
PAL_RF_MODEL_TBL, PAL_RF_VAR_IMP_TBL, PAL_RF_ERR_RATE_TBL, PAL_RF_CONFUSION_TBL)
WITH OVERVIEW;
SELECT * FROM PAL_RF_MODEL_TBL;
SELECT * FROM PAL_RF_VAR_IMP_TBL;
SELECT * FROM PAL_RF_ERR_RATE_TBL;
SELECT * FROM PAL_RF_CONFUSION_TBL;

```

Expected Result

PAL_RF_MODEL_TBL:

ID	TREEINDEX	MODEL
0	-1	<PMML version="4.0" xmlns="http://www.dmg.org/...>
1	0	<PMML version="4.0" xmlns="http://www.dmg.org/...>
2	1	<PMML version="4.0" xmlns="http://www.dmg.org/...>

PAL_RF_VAR_IMP_TBL:

VAR	IMP
OUTLOOK	0.17467931142931148
TEMP	0.13924695278266705
HUMIDITY	0.08958790283790281
WINDY	0.01699603703175132

PAL_RF_ERR_RATE_TBL:

TREEINDEX	ERR
0	0.6666666666666666
1	0.625
2	0.4
3	0.3076923076923077

PAL_RF_CONFUSION_TBL:

ID	CONTENT
0	{"ConfusionMatrix":{"ActualCategories":["Play","Do not Play"],"Count":[6,3,2,3],...}}

RANDOMFORESTSCORING

This function is used for classification or regression scoring.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'RANDOMFORESTSCORING',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Scoring data INPUT table type>	IN
2	<schema_name>	<Parameter table type>	IN
3	<schema_name>	<Random forest model INPUT table type>	IN
4	<schema_name>	<Results OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<scoring data input table>, <parameter table>, <random forest model input table>, <results output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Scoring Data	1st column	Integer, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar	Dependent field	
Tree Model	1st column	Integer	ID	
	2nd column	Integer	Tree index	
	3rd column	VARCHAR or NVARCHAR	Model content	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

None.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	2nd column	Varchar or nvarchar	Scoring result
	3rd column	Double	Output nulls for regression

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_RF_SCORING_DATA_T;
CREATE TYPE PAL_RF_SCORING_DATA_T AS TABLE(
    "ID" INTEGER,
    "OUTLOOK" VARCHAR(20),
    "TEMP" INTEGER,
    "HUMIDITY" DOUBLE,
    "WINDY" VARCHAR(10)
);
DROP TYPE PAL_RF_SCORING_MODEL_T;
CREATE TYPE PAL_RF_SCORING_MODEL_T AS TABLE(
    "ID" INTEGER,
    "TREEINDEX" INTEGER,
    "MODEL" VARCHAR(5000)
);
DROP TYPE PAL_RF_SCORING_RESULT_T;
CREATE TYPE PAL_RF_SCORING_RESULT_T AS TABLE("ID" INTEGER, "SCORING"
VARCHAR(50), "PROB" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_RF_SCORING_PDATA_TBL;
CREATE COLUMN TABLE PAL_RF_SCORING_PDATA_TBL (
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_RF_SCORING_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_RF_SCORING_DATA_T', 'IN');
INSERT INTO PAL_RF_SCORING_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_RF_SCORING_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_RF_SCORING_MODEL_T', 'IN');
```

```

INSERT INTO PAL_RF_SCORING_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_RF_SCORING_RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_RF_SCORING_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'RANDOMFORESTSCORING',
'DM_PAL', 'PAL_RF_SCORING_PROC', PAL_RF_SCORING_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('IS_OUTPUT_PROBABILITY', 0, null, null);
DROP TABLE PAL_RF_SCORING_DATA_TBL;
CREATE COLUMN TABLE PAL_RF_SCORING_DATA_TBL LIKE PAL_RF_SCORING_DATA_T;
INSERT INTO PAL_RF_SCORING_DATA_TBL VALUES (0, 'Overcast', 75, -10000, 'Yes');
INSERT INTO PAL_RF_SCORING_DATA_TBL VALUES (1, 'Rain', 78, 70, 'Yes');
INSERT INTO PAL_RF_SCORING_DATA_TBL VALUES (2, 'Sunny', -10000, null, 'Yes');
INSERT INTO PAL_RF_SCORING_DATA_TBL VALUES (3, 'Sunny', 69, 70, 'Yes');
INSERT INTO PAL_RF_SCORING_DATA_TBL VALUES (4, 'Rain', null, 70, 'Yes');
INSERT INTO PAL_RF_SCORING_DATA_TBL VALUES (5, null, 70, 70, 'Yes');
INSERT INTO PAL_RF_SCORING_DATA_TBL VALUES (6, '***', 70, 70, 'Yes');
DROP TABLE PAL_RF_SCORING_RESULT_TBL;
CREATE COLUMN TABLE PAL_RF_SCORING_RESULT_TBL LIKE PAL_RF_SCORING_RESULT_T;
CALL "DM_PAL".PAL_RF_SCORING_PROC(PAL_RF_SCORING_DATA_TBL, #PAL_CONTROL_TBL,
PAL_RF_MODEL_TBL, PAL_RF_SCORING_RESULT_TBL) with OVERVIEW;
SELECT * FROM PAL_RF_SCORING_RESULT_TBL;

```

Expected Result

PAL_RF_SCORING_RESULT_TBL:

ID	SCORING	PROB
0	Play	0.99
1	Play	0.8966666666666666
2	Do not Play	0.9133333333333333
3	Do not Play	0.6233333333333333

3.2.14 Support Vector Machine

Support Vector Machines (SVMs) refer to a family of supervised learning models using the concept of support vector. Compared with many other supervised learning models, SVMs have the advantages in that the models produced by SVMs can be either linear or non-linear, where the latter is realized by a technique called Kernel Trick.

Like most supervised models, there are training phase and testing phase for SVMs. In the training phase, a function $f(x) \rightarrow y$ where $f(\cdot)$ is a function (can be non-linear) mapping a sample onto a TARGET, is learnt. The training set consists of pairs denoted by $\{x_i, y_i\}$, where x denotes a sample represented by several attributes, and y denotes a TARGET (supervised information). In the testing phase, the learnt $f(\cdot)$ is further used to map a sample with unknown TARGET onto its predicted TARGET.

In the current implementation in PAL, SVMs can be used for the following three tasks:

- **Support Vector Classification (SVC)**

Classification is one of the most frequent tasks in many fields including machine learning, data mining, computer vision, and business data analysis. Compared with linear classifiers like logistic regression, SVC

is able to produce non-linear decision boundary, which leads to better accuracy on some real world dataset. In classification scenario, $f(\cdot)$ refers to decision function, and a TARGET refers to a "label" represented by a real number.

- **Support Vector Regression (SVR)**

SVR is another method for regression analysis. Compared with classical linear regression methods like least square regression, the regression function in SVR can be non-linear. In regression scenario, $f(\cdot)$ refers to regression function, and TARGET refers to "response" represented by a real number.

- **Support Vector Ranking**

This implements a pairwise "learning to rank" algorithm which learns a ranking function from several sets (distinguished by Query ID) of ranked samples. In the scenario of ranking, $f(\cdot)$ refers to ranking function, and TARGET refers to score, according to which the final ranking is made. For pairwise ranking, $f(\cdot)$ is learnt so that the pairwise relationship expressing the rank of the samples within each set is considered.

Because non-linearity is realized by Kernel Trick, besides the datasets, the kernel type and parameters should be specified as well.

Prerequisite

No missing or null data in the inputs.

SVMTRAIN

This function reads the input data and generates training model.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'SVMTRAIN',
'<schema_name>', '<training_procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT model table type>	OUT

Procedure Calling

```
CALL <schema_name>.<training_procedure_name>(<input table>, <parameter table>,
<model table>) with overview;
```

The training procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Attributes columns	Integer, double, varchar, or nvarchar	Attributes data	
	Query ID column	Varchar or nvarchar	Query ID	Only needed for SVM ranking.
	Dependent variable columns	Integer, double, or nvarchar	Dependent variable	

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
TYPE	Integer	SVM type: <ul style="list-style-type: none">• 1: SVC (for classification)• 2: SVR (for regression)• 3: Support Vector Ranking (for ranking)

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
THREAD_NUMBER	Integer	1	Number of threads.	
KERNEL_TYPE	Integer	2	Kernel type: <ul style="list-style-type: none">• 0: LINEAR KERNEL• 1: POLY KERNEL• 2: RBF KERNEL• 3: SIGMOID KERNEL	
POLY_DEGREE	Integer	3	Coefficient for the PLOY KERNEL type. Value range: ≥ 1	Only valid when KERNEL_TYPE is 1.

Name	Data Type	Default Value	Description	Dependency
RBF_GAMMA	Double	0.005	Coefficient for the RBF KERNEL type. Value range: > 0	Only valid when KERNEL_TYPE is 2.
COEF_LIN	Double	0.0	Coefficient for the POLY/SIGMOID KERNEL type.	Only valid when KERNEL_TYPE is 1 or 3.
COEF_CONST	Double	0.0	Coefficient for the POLY/SIGMOID KERNEL type.	Only valid when KERNEL_TYPE is 1 or 3.
SVM_C	Double	100.0	Trade-off between training error and margin. Value range: > 0	
REGRESSION_EPS	Double	0.1	Epsilon width of tube for regression. Value range: > 0	Only valid when TYPE is 2.
SCALE_INFO	Integer	1	<ul style="list-style-type: none"> • 0: No scale • 1: Normalizes the data • 2: Scales the data into [-1,1] 	
SHRINK	Integer	1	Decides whether to use shrink strategy or not: <ul style="list-style-type: none"> • 0: Does not use shrink strategy • 1: Uses shrink strategy 	Using shrink strategy may accelerate the training process.
CATEGORY_COL	Integer	-1	Indicates whether the column is category variable. By default, 'string' is category variable, and 'integer' or 'double' is continuous variable. The value of -1 means the default will be used.	

Name	Data Type	Default Value	Description	Dependency
CATEGORY_WEIGHT	Double	0.707	Represents the weight of category attributes (γ). The value must be greater than 0.	
ERROR_TOL	Double	0.001	Specifies the error tolerance in the training process. The value must be greater than 0.	
EVALUATION_SEED	Integer	0	The random seed in parameter selection. The value must be greater than 0.	
PROBABILITY	Integer	0	If you want to output probability when scoring, set this to 1. Note: Setting this to 1 will severely degrade the training performance. Use the default value (0) unless you want to output probabilities when scoring.	

Output Tables

Table	Column	Column Data Type	Description
Model Result	1st column	Varchar, nvarchar, or integer	ID
	2nd column	Varchar	Model content

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1: Support vector classification

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_SVM_TRAINING_T;
CREATE TYPE PAL_SVM_TRAINING_T AS TABLE (
```

```

        ID INTEGER,
        ATTRIBUTE1 DOUBLE,
        ATTRIBUTE2 DOUBLE,
        ATTRIBUTE3 DOUBLE,
        ATTRIBUTE4 VARCHAR(10),
        LABEL INTEGER
    );
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    NAME VARCHAR(50),
    INTARGS INTEGER,
    DOUBLEARGS DOUBLE,
    STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_SVM_MODEL_T;
CREATE TYPE PAL_SVM_MODEL_T AS TABLE( ID VARCHAR(50), MODEL VARCHAR(5000));
DROP TABLE PAL_SVM_PDATA_TBL;
CREATE TABLE PAL_SVM_PDATA_TBL("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_SVM_PDATA_TBL VALUES (1,'DM_PAL','PAL_SVM_TRAINING_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (3,'DM_PAL','PAL_SVM_MODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_SVM_TRAIN');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','SVMTRAIN','DM_PAL','PAL_SVM_TRAIN',
'PAL_SVM_PDATA_TBL');
DROP TABLE PAL_SVM_TRAINING_TBL;
CREATE COLUMN TABLE PAL_SVM_TRAINING_TBL LIKE PAL_SVM_TRAINING_T;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
DROP TABLE PAL_SVM_MODEL_TBL_EX1;
CREATE COLUMN TABLE PAL_SVM_MODEL_TBL_EX1 LIKE PAL_SVM_MODEL_T;
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(0,1,10,100,'A',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(1,1.1,10.1,100,'A',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(2,1.2,10.2,100,'A',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(3,1.3,10.4,100,'A',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(4,1.2,10.3,100,'AB',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(5,4,40,400,'AB',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(6,4.1,40.1,400,'AB',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(7,4.2,40.2,400,'AB',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(8,4.3,40.4,400,'AB',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(9,4.2,40.3,400,'AB',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(10,9,90,900,'B',3);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(11,9.1,90.1,900,'A',3);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(12,9.2,90.2,900,'B',3);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(13,9.3,90.4,900,'A',3);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(14,9.2,90.3,900,'A',3);
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('KERNEL_TYPE',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('TYPE',1,null,null);
CALL
DM_PAL.PAL_SVM_TRAIN(PAL_SVM_TRAINING_TBL,#PAL_CONTROL_TBL,PAL_SVM_MODEL_TBL_EX1)
WITH OVERVIEW;
SELECT * FROM PAL_SVM_MODEL_TBL_EX1;

```

Expected Result

ID	MODEL
0	{"attr_number":4,"bias":[-0.03010865044869730,0.01642110788275344,0.2088123843101358],"categoric...

Example 2: Support vector ranking

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SVM_TRAINING_T;
CREATE TYPE PAL_SVM_TRAINING_T AS TABLE (

```

```

        ID INTEGER,
        ATTRIBUTE1 DOUBLE,
        ATTRIBUTE2 DOUBLE,
        ATTRIBUTE3 DOUBLE,
        ATTRIBUTE4 DOUBLE,
        ATTRIBUTE5 DOUBLE,
        QID VARCHAR(50),
        LABEL INTEGER
    );
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    NAME VARCHAR(50),
    INTARGS INTEGER,
    DOUBLEARGS DOUBLE,
    STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_SVM_MODEL_T;
CREATE TYPE PAL_SVM_MODEL_T AS TABLE( ID VARCHAR(50), MODEL VARCHAR(5000));
DROP TABLE PAL_SVM_PDATA_TBL;
CREATE TABLE PAL_SVM_PDATA_TBL("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_SVM_PDATA_TBL VALUES (1,'DM_PAL','PAL_SVM_TRAINING_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (3,'DM_PAL','PAL_SVM_MODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_SVM_TRAIN');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','SVMTRAIN','DM_PAL','PAL_SVM_TRAIN',
,PAL_SVM_PDATA_TBL);
DROP TABLE PAL_SVM_TRAINING_TBL;
CREATE COLUMN TABLE PAL_SVM_TRAINING_TBL LIKE PAL_SVM_TRAINING_T;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
DROP TABLE PAL_SVM_MODEL_TBL_EX2;
CREATE COLUMN TABLE PAL_SVM_MODEL_TBL_EX2 LIKE PAL_SVM_MODEL_T;
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(0,1,1,0,0.2,0,'qid:1',3);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(1,0,0,1,0.1,1,'qid:1',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(2,0,0,1,0.3,0,'qid:1',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(3,2,1,1,0.2,0,'qid:1',4);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(4,3,1,1,0.4,1,'qid:1',5);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(5,4,1,1,0.7,0,'qid:1',6);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(6,0,0,1,0.2,0,'qid:2',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(7,1,0,1,0.4,0,'qid:2',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(8,0,0,1,0.2,0,'qid:2',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(9,1,1,1,0.2,0,'qid:2',3);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(10,2,2,1,0.4,0,'qid:2',4);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(11,3,3,1,0.2,0,'qid:2',5);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(12,0,0,1,0.1,1,'qid:3',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(13,1,0,0,0.4,1,'qid:3',4);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(14,0,1,1,0.5,0,'qid:3',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(15,1,1,1,0.5,1,'qid:3',3);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(16,2,2,0,0.7,1,'qid:3',5);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(17,1,3,1,1.5,0,'qid:3',6);
INSERT INTO #PAL_CONTROL_TBL VALUES('KERNEL_TYPE',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('TYPE',3,null,null);
CALL
DM_PAL.PAL_SVM_TRAIN(PAL_SVM_TRAINING_TBL,#PAL_CONTROL_TBL,PAL_SVM_MODEL_TBL_EX2)
    WITH OVERVIEW;
SELECT * FROM PAL_SVM_MODEL_TBL_EX2;

```

Expected Result

ID	MODEL
0	{"attr_number":5,"bias":[0.0],"categorical_attr_number":0,"class_number":2,"converted_attributes_number":5,...}

Example 3: Support vector classification (training a model with additional information that can be used when calculating scoring probability)

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SVM_TRAINING_T;
CREATE TYPE PAL_SVM_TRAINING_T AS TABLE (
    ID INTEGER,
    ATTRIBUTE1 DOUBLE,
    ATTRIBUTE2 DOUBLE,
    ATTRIBUTE3 DOUBLE,
    ATTRIBUTE4 VARCHAR(10),
    LABEL INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    NAME VARCHAR(50),
    INTARGS INTEGER,
    DOUBLEARGS DOUBLE,
    STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_SVM_MODEL_T;
CREATE TYPE PAL_SVM_MODEL_T AS TABLE( ID VARCHAR(50), MODEL VARCHAR(5000));
DROP TABLE PAL_SVM_PDATA_TBL;
CREATE TABLE PAL_SVM_PDATA_TBL("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_SVM_PDATA_TBL VALUES (1,'SYSTEM','PAL_SVM_TRAINING_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (2,'SYSTEM','PAL_CONTROL_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (3,'SYSTEM','PAL_SVM_MODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('SYSTEM','PAL_SVM_TRAIN');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','SVMTRAIN','SYSTEM','PAL_SVM_TRAIN',
,PAL_SVM_PDATA_TBL);
DROP TABLE PAL_SVM_TRAINING_TBL;
CREATE COLUMN TABLE PAL_SVM_TRAINING_TBL LIKE PAL_SVM_TRAINING_T;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
DROP TABLE PAL_SVM_MODEL_TBL_EX3;
CREATE COLUMN TABLE PAL_SVM_MODEL_TBL_EX3 LIKE PAL_SVM_MODEL_T;
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(0,1,10,100,'A',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(1,1.1,10.1,100,'A',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(2,1.2,10.2,100,'A',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(3,1.3,10.4,100,'A',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(4,1.2,10.3,100,'AB',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(5,4,40,400,'AB',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(6,4.1,40.1,400,'AB',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(7,4.2,40.2,400,'AB',1);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(8,4.3,40.4,400,'AB',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(9,4.2,40.3,400,'AB',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(10,9,90,900,'B',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(11,9.1,90.1,900,'A',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(12,9.2,90.2,900,'B',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(13,9.3,90.4,900,'A',2);
INSERT INTO PAL_SVM_TRAINING_TBL VALUES(14,9.2,90.3,900,'A',2);
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('KERNEL_TYPE',2,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('TYPE',1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES('PROBABILITY',1,null,null);
CALL
SYSTEM.PAL_SVM_TRAIN(PAL_SVM_TRAINING_TBL,#PAL_CONTROL_TBL,PAL_SVM_MODEL_TBL_EX3)
WITH OVERVIEW;
SELECT * FROM PAL_SVM_MODEL_TBL_EX3;

```

Expected Result

ID	MODEL
0	{"attr_number":4,"bias":[-0.01919798419693774],"categorical_attr_number":1,"class_numbe...}

SVMPREDICT

This function uses the training model generated by SVMTRAIN to make predictive analysis.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'SVMPREDICT',
'<schema_name>', '<predicting_procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<INPUT model table type>	IN
4	<schema_name>	<OUTPUT predict result table type>	OUT

Procedure Calling

```
CALL <schema_name>.<predicting_procedure_name>(<input table>, <parameter table>,
<model table>, <predict result table>) with overview;
```

The predicting procedure name is the same as specified in the procedure generation.

The tables must of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, double, varchar, or nvarchar	Attributes data	The varchar or nvarchar data type is available only when the input model table is not null.

Table	Column	Column Data Type	Description	Constraint
	Last column	Varchar or nvarchar	Query ID	Only needed for SVM ranking.

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads
VERBOSE_OUTPUT	Integer	0	Output scoring probabilities for each class

Input Model Table

Table	Column	Column Data Type	Description
Model Result	1st column	Integer, varchar, or nvarchar	ID
	2nd column	Varchar or nvarchar	Model content

Output Table

Table	Column	Column Data Type	Description
Predict Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer, varchar, nvarchar, or double	Prediction value
	3rd column	Double	Prediction probability. It is necessary when the PROBABILITY parameter is set to 1 in SVMTRAIN for SVC.

Examples

Assume that:

- DM_PAL is a schema belonging to USER1; and

- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1: Support vector classification

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SVM_SCORING_T;
CREATE TYPE PAL_SVM_SCORING_T AS TABLE ( ID INTEGER, ATTRIBUTE1 DOUBLE,
ATTRIBUTE2 DOUBLE, ATTRIBUTE3 DOUBLE, ATTRIBUTE4 VARCHAR(10));
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    NAME VARCHAR(50),
    INTARGS INTEGER,
    DOUBLEARGS DOUBLE,
    STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_SVM_MODEL_T;
CREATE TYPE PAL_SVM_MODEL_T AS TABLE( ID VARCHAR(50), MODEL VARCHAR(5000) );
DROP TYPE PAL_SVM_RESULT_T;
CREATE TYPE PAL_SVM_RESULT_T AS TABLE( ID INTEGER, SCORING INTEGER);
DROP TABLE PAL_SVM_PDATA_TBL;
CREATE COLUMN TABLE PAL_SVM_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_SVM_PDATA_TBL VALUES (1,'DM_PAL','PAL_SVM_SCORING_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (3,'DM_PAL','PAL_SVM_MODEL_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (4,'DM_PAL','PAL_SVM_RESULT_T','OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_SVM_PREDICT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','SVMPREDICT','DM_PAL','PAL_SVM_PREDICT',PAL_SVM_PDATA_TBL);
DROP TABLE PAL_SVM_SCORING_TBL;
CREATE COLUMN TABLE PAL_SVM_SCORING_TBL LIKE PAL_SVM_SCORING_T;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
DROP TABLE PAL_SVM_RESULT_TBL;
CREATE COLUMN TABLE PAL_SVM_RESULT_TBL LIKE PAL_SVM_RESULT_T;
INSERT INTO PAL_SVM_SCORING_TBL VALUES(0,1,10,100,'A');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(1,1.2,10.2,100,'A');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(2,4.1,40.1,400,'AB');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(3,4.2,40.3,400,'AB');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(4,9.1,90.1,900,'A');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(5,9.2,90.2,900,'A');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(6,4,40,400,'A');
INSERT INTO #PAL_CONTROL_TBL VALUES('THREAD_NUMBER',8,null,null);
CALL
DM_PAL.PAL_SVM_PREDICT(PAL_SVM_SCORING_TBL,#PAL_CONTROL_TBL,PAL_SVM_MODEL_TBL_EX1
,PAL_SVM_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_SVM_RESULT_TBL;

```

Expected Result

ID	SCORING
0	1
1	1
2	2
3	2
4	3
5	3
6	2

Example 2: Support vector ranking

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_SVM_SCORING_T;
CREATE TYPE PAL_SVM_SCORING_T AS TABLE (
    ID INTEGER,
    ATTRIBUTE1 DOUBLE,
    ATTRIBUTE2 DOUBLE,
    ATTRIBUTE3 DOUBLE,
    ATTRIBUTE4 DOUBLE,
    ATTRIBUTE5 DOUBLE,
    QID VARCHAR(50)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    NAME VARCHAR(50),
    INTARGS INTEGER,
    DOUBLEARGS DOUBLE,
    STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_SVM_MODEL_T;
CREATE TYPE PAL_SVM_MODEL_T AS TABLE( ID VARCHAR(50), MODEL VARCHAR(5000));
DROP TYPE PAL_SVM_RESULT_T;
CREATE TYPE PAL_SVM_RESULT_T AS TABLE( ID INTEGER, DECISION_VAL DOUBLE);
DROP TABLE PAL_SVM_PDATA_TBL;
CREATE COLUMN TABLE PAL_SVM_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_SVM_PDATA_TBL VALUES (1,'DM_PAL','PAL_SVM_SCORING_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (3,'DM_PAL','PAL_SVM_MODEL_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (4,'DM_PAL','PAL_SVM_RESULT_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_SVM_PREDICT');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','SVMPREDICT','DM_PAL','PAL_SVM_PRED
ICT',PAL_SVM_PDATA_TBL);
DROP TABLE PAL_SVM_SCORING_TBL;
CREATE COLUMN TABLE PAL_SVM_SCORING_TBL LIKE PAL_SVM_SCORING_T;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
DROP TABLE PAL_SVM_RESULT_TBL;
CREATE COLUMN TABLE PAL_SVM_RESULT_TBL LIKE PAL_SVM_RESULT_T;
INSERT INTO PAL_SVM_SCORING_TBL VALUES(0,1,1,0,0.2,0,'qid:1');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(1,0,0,1,0.1,1,'qid:1');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(2,0,0,1,0.3,0,'qid:1');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(3,2,1,1,0.2,0,'qid:1');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(4,3,1,1,0.4,1,'qid:1');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(5,4,1,1,0.7,0,'qid:1');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(6,0,0,1,0.2,0,'qid:4');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(7,1,0,1,0.4,0,'qid:4');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(8,0,0,1,0.2,0,'qid:4');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(9,1,1,1,0.2,0,'qid:4');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(10,2,2,1,0.4,0,'qid:4');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(11,3,3,1,0.2,0,'qid:4');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(12,0,0,1,0.1,1,'qid:5');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(13,1,0,0,0.4,1,'qid:5');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(14,0,1,1,0.5,0,'qid:5');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(15,1,1,1,0.5,1,'qid:5');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(16,2,2,0,0.7,1,'qid:5');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(17,1,3,1,1.5,0,'qid:5');
CALL
DM_PAL.PAL_SVM_PREDICT(PAL_SVM_SCORING_TBL,#PAL_CONTROL_TBL,PAL_SVM_MODEL_TBL_EX2
,PAL_SVM_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_SVM_RESULT_TBL;
```

Expected Result

ID	DECISION_VAL
0	-9.85138781348735
1	-10.865705861856142
2	-11.674107955799869
3	-9.339858081462978
4	-7.888395858857351
5	-6.884207129029804
6	-11.708123470415686
7	-10.800391412029356
8	-11.708123470415686
9	-10.2583824933903
10	-8.38342259491742
11	-6.640574243662741
12	-10.865705861856142
13	-9.435329678678514
14	-10.880256607339817
15	-9.531995558790442
16	-7.617642390305434
17	-6.117245878259126

Example 3: Support vector classification (training a model with additional information that can be used when calculating scoring probability)

```

SET SCHEMA DM PAL;
DROP TYPE PAL_SVM_SCORING_T;
CREATE TYPE PAL_SVM_SCORING_T AS TABLE ( ID INTEGER, ATTRIBUTE1 DOUBLE,
ATTRIBUTE2 DOUBLE, ATTRIBUTE3 DOUBLE, ATTRIBUTE4 VARCHAR(10));
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    NAME VARCHAR(50),
    INTARGS INTEGER,
    DOUBLEARGS DOUBLE,
    STRINGARGS VARCHAR(100)
);
DROP TYPE PAL_SVM_MODEL_T;
CREATE TYPE PAL_SVM_MODEL_T AS TABLE( ID VARCHAR(50), MODEL VARCHAR(5000) );
DROP TYPE PAL_SVM_RESULT_T;
CREATE TYPE PAL_SVM_RESULT_T AS TABLE( ID INTEGER, SCORING INTEGER, PROBABILITY
DOUBLE );
DROP TABLE PAL_SVM_PDATA_TBL;
CREATE COLUMN TABLE PAL_SVM_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_SVM_PDATA_TBL VALUES (1,'SYSTEM','PAL_SVM_SCORING_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (2,'SYSTEM','PAL_CONTROL_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (3,'SYSTEM','PAL_SVM_MODEL_T','IN');
INSERT INTO PAL_SVM_PDATA_TBL VALUES (4,'SYSTEM','PAL_SVM_RESULT_T','OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('SYSTEM','PAL_SVM_PREDICT');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','SVMPREDICT','SYSTEM','PAL_SVM_PRED
ICT',PAL_SVM_PDATA_TBL);
DROP TABLE PAL_SVM_SCORING_TBL;
CREATE COLUMN TABLE PAL_SVM_SCORING_TBL LIKE PAL_SVM_SCORING_T;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
DROP TABLE PAL_SVM_RESULT_TBL;
CREATE COLUMN TABLE PAL_SVM_RESULT_TBL LIKE PAL_SVM_RESULT_T;
INSERT INTO PAL_SVM_SCORING_TBL VALUES(0,1,10,100,'A');

```

```

INSERT INTO PAL_SVM_SCORING_TBL VALUES(1,1.2,10.2,100,'A');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(2,4.1,40.1,400,'AB');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(3,4.2,40.3,400,'AB');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(4,9.1,90.1,900,'A');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(5,9.2,90.2,900,'A');
INSERT INTO PAL_SVM_SCORING_TBL VALUES(6,4,40,400,'A');
INSERT INTO #PAL_CONTROL_TBL VALUES('VERBOSE_OUTPUT',1,null,null);
CALL
SYSTEM.PAL_SVM_PREDICT(PAL_SVM_SCORING_TBL,#PAL_CONTROL_TBL,PAL_SVM_MODEL_TBL_EX3
,PAL_SVM_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_SVM_RESULT_TBL;

```

Expected Result

ID	SCORING	PROBABILITY
0	1	0.8932019000826287
0	2	0.10679809991737166
1	1	0.890176378374231
1	2	0.10982362162576906
2	1	0.7766171907086251
2	2	0.22338280929137477
3	1	0.7734522772613809
3	2	0.226547722738619
4	1	0.4363141158466445
4	2	0.5636858841533554
5	1	0.43215038213638096
5	2	0.5678496178636192
6	1	0.7795014025835508
6	2	0.2204985974164494

Related Information

[Confusion Matrix \[page 154\]](#)

3.2.15 Incremental Classification on SAP HANA Smart Data Streaming

For information on incremental classification on SAP HANA Smart Data Streaming, see the example of Hoeffding tree training and scoring in the *SAP HANA Smart Data Streaming: Developer Guide*.

i Note

SAP HANA Smart Data Streaming is only supported on Intel-based hardware platforms.

3.3 Regression Algorithms

This section describes the regression algorithms that are provided by the Predictive Analysis Library.

3.3.1 Bi-Variate Geometric Regression

Geometric regression is an approach used to model the relationship between a scalar variable y and a variable denoted X . In geometric regression, data is modeled using geometric functions, and unknown model parameters are estimated from the data. Such models are called geometric models.

In PAL, the implementation of geometric regression is to transform to linear regression and solve it:

$$y = \beta_0 \times x^{\beta_1}$$

Where β_0 and β_1 are parameters that need to be calculated.

The steps are:

1. Put natural logarithmic operation on both sides: $\ln(y) = \ln(\beta_0 \times x^{\beta_1})$
2. Transform it into: $\ln(y) = \ln(\beta_0) + \beta_1 \times \ln(x)$
3. Let $y' = \ln(y)$, $x' = \ln(x)$, $\beta_0' = \ln(\beta_0)$
 $y' = \beta_0' + \beta_1 \times x'$

Thus, y' and x' is a linear relationship and can be solved with the linear regression method.

The implementation also supports calculating the F value and R² to determine statistical significance.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

GOREGRESSION

This is a geometric regression function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'GOREGRESSION',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Fitted OUTPUT table type>	OUT
5	<schema_name>	<Significance OUTPUT table type>	OUT
6	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>, <fitted output table>, <significance output table>, <PMML output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Variable y
	3rd column	Integer or double	Variable x

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads.

Name	Data Type	Default Value	Description
ALG	Integer	0	<p>Specifies decomposition method:</p> <ul style="list-style-type: none"> • 0: Doolittle decomposition (LU) • 2: Singular value decomposition (SVD)
ADJUSTED_R2	Integer	0	<ul style="list-style-type: none"> • 0: Does not output adjusted R square • 1: Outputs adjusted R square
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export geometric regression model in PMML. • 1: Exports geometric regression model in PMML in single row. • 2: Exports geometric regression model in several rows, and the minimum length of each row is 5000 characters.
SELECTED_FEATURES	Varchar	No default value	<p>A string to specify the features that will be processed. The pattern is “X_1, \dots, X_n”, where X_i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed, and the third column data will be processed as independent variables.</p>
DEPENDENT_VARIABLE	Varchar	No default value	(Optional) Column name in the data table used as dependent variable. If this parameter is not specified, the second column data will be processed as dependent variables.

Output Tables

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, varchar, or nvarchar	ID Note: If the SELECTED_FEATURE parameter is specified, the column data type must be varchar or nvarchar.	
	2nd column	Integer or double	Value A_i <ul style="list-style-type: none"> A_0: intercept A_1: beta coefficient for X_1 	
Fitted Data	1st column	Integer, bigint, varchar, or nvarchar	ID	
	2nd column	Integer or double	Value Y_i	
Significance	1st column	VARCHAR or nvarchar	Name	(R^2 / F)
	2nd column	Double	Value	
PMML Result	1st column	Integer	ID	
	2nd column	CLOB, varchar, or nvarchar	Geometric regression model in PMML format	

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_GR_DATA_T;
CREATE TYPE PAL_GR_DATA_T AS TABLE( "ID" INT, "Y" DOUBLE, "X1" DOUBLE);
DROP TYPE PAL_GR_RESULT_T;
CREATE TYPE PAL_GR_RESULT_T AS TABLE("ID" INT, "Ai" DOUBLE);
DROP TYPE PAL_GR_FITTED_T;
CREATE TYPE PAL_GR_FITTED_T AS TABLE("ID" INT, "Fitted" DOUBLE);
DROP TYPE PAL_GR_SIGNIFICANCE_T;
CREATE TYPE PAL_GR_SIGNIFICANCE_T AS TABLE("NAME" VARCHAR(50), "VALUE" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_GR_PMMODEL_T;
CREATE TYPE PAL_GR_PMMODEL_T AS TABLE("ID" INT, "Model" VARCHAR(5000));
DROP table PAL_GR_PDATA_TBL;
CREATE column table PAL_GR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_GR_PDATA_TBL values (1,'DM_PAL','PAL_GR_DATA_T','IN');
INSERT INTO PAL_GR_PDATA_TBL values (2,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_GR_PDATA_TBL values (3,'DM_PAL','PAL_GR_RESULT_T','OUT');
INSERT INTO PAL_GR_PDATA_TBL values (4,'DM_PAL','PAL_GR_FITTED_T','OUT');
INSERT INTO PAL_GR_PDATA_TBL values (5,'DM_PAL','PAL_GR_SIGNIFICANCE_T','OUT');
INSERT INTO PAL_GR_PDATA_TBL values (6,'DM_PAL','PAL_GR_PMMODEL_T','OUT');
```

```

CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_GEOREGRESSION_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'GEOREGRESSION', 'DM_PAL', 'PAL_GEOREGRESSION_PROC', 'PAL_GR_PDATA_TBL');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 8, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT', 2, null, null);
DROP TABLE PAL_GR_DATA_TBL;
CREATE COLUMN TABLE PAL_GR_DATA_TBL ( "ID" INT, "Y" DOUBLE, "X1" DOUBLE);
INSERT INTO PAL_GR_DATA_TBL VALUES (0, 1.1, 1);
INSERT INTO PAL_GR_DATA_TBL VALUES (1, 4.2, 2);
INSERT INTO PAL_GR_DATA_TBL VALUES (2, 8.9, 3);
INSERT INTO PAL_GR_DATA_TBL VALUES (3, 16.3, 4);
INSERT INTO PAL_GR_DATA_TBL VALUES (4, 24.5);
INSERT INTO PAL_GR_DATA_TBL VALUES (5, 36.6);
INSERT INTO PAL_GR_DATA_TBL VALUES (6, 48.7);
INSERT INTO PAL_GR_DATA_TBL VALUES (7, 64.8);
INSERT INTO PAL_GR_DATA_TBL VALUES (8, 80.9);
INSERT INTO PAL_GR_DATA_TBL VALUES (9, 101.10);
DROP TABLE PAL_GR_RESULTS_TBL;
CREATE COLUMN TABLE PAL_GR_RESULTS_TBL ("ID" INT, "Ai" DOUBLE);
DROP TABLE PAL_GR_FITTED_TBL;
CREATE COLUMN TABLE PAL_GR_FITTED_TBL ("ID" INT, "Fitted" DOUBLE);
DROP TABLE PAL_GR_SIGNIFICANCE_TBL;
CREATE COLUMN TABLE PAL_GR_SIGNIFICANCE_TBL ("NAME" varchar(50), "VALUE" DOUBLE);
DROP TABLE PAL_GR_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_GR_PMMODEL_TBL ("ID" INT, "PMMODEL" VARCHAR(5000));
CALL DM_PAL.PAL_GEOREGRESSION_PROC(PAL_GR_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_GR_RESULTS_TBL, PAL_GR_FITTED_TBL, PAL_GR_SIGNIFICANCE_TBL,
PAL_GR_PMMODEL_TBL) with overview;
SELECT * FROM PAL_GR_RESULTS_TBL;
SELECT * FROM PAL_GR_FITTED_TBL;
SELECT * FROM PAL_GR_SIGNIFICANCE_TBL;
SELECT * FROM PAL_GR_PMMODEL_TBL;

```

Expected Result

PAL_GR_RESULTS_TBL:

ID	Ai
0	1.0722194976977562
1	1.9596331198574095

PAL_GR_FITTED_TBL:

ID	Fitted
0	1.0722194976977562
1	4.170537685916355
2	9.231373297672446
3	16.221850681688966
4	25.119354247561105
5	35.90663135054967
6	48.56974340546393
7	63.097005555817994
8	79.47836533840173
9	97.70500700675326

PAL_GR_SIGNIFICANCE_TBL:

NAME	VALUE
R2	0.9996968130780065
F	26378.362404439384

PAL_GR_PMMODEL_TBL:

ID	PMMODEL
1	<PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.dmg.org/PMML-4_0 PMML-4_0.xsd">

FORECASTWITHGEOR

This function performs prediction with the geometric regression result.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'FORECASTWITHGEOR',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Predictive INPUT table type>	IN
2	<schema_name>	<Coefficient INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<predictive input table>, <coefficient input table>, <parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Predictive Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Variable X

Table	Column	Column Data Type	Description
Coefficient	1st column	Integer, varchar, or nvarchar	ID (start from 0)
	2nd column	Integer, double, varchar, nvarchar, or CLOB	Value A_i or PMML model. Varchar, nvarchar, and CLOB types are only valid for PMML model.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads
MODEL_FORMAT	Integer	0	<ul style="list-style-type: none"> 0: Coefficients in table 1: PMML format

Output Table

Table	Column	Column Data Type	Description
Fitted Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Value Y_i

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_FGR_PREDICT_T;
CREATE TYPE PAL_FGR_PREDICT_T AS TABLE( "ID" INT, "X1" DOUBLE);
DROP TYPE PAL_FGR_COEFFICIENT_T;
CREATE TYPE PAL_FGR_COEFFICIENT_T AS TABLE("ID" INT, "Ai" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_FGR_FITTED_T;
CREATE TYPE PAL_FGR_FITTED_T AS TABLE("ID" INT, "Fitted" DOUBLE);
DROP table PAL_FGR_FDTA_TBL;

```

```

CREATE column table PAL_FGR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
insert into PAL_FGR_PDATA_TBL values (1,'DM_PAL','PAL_FGR_PREDICT_T','IN');
insert into PAL_FGR_PDATA_TBL values (2,'DM_PAL','PAL_FGR_COEFFICIENT_T','IN');
insert into PAL_FGR_PDATA_TBL values (3,'DM_PAL','PAL_CONTROL_T','IN');
insert into PAL_FGR_PDATA_TBL values (4,'DM_PAL','PAL_FGR_FITTED_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_FORECAST_GEOG_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','FORECASTWITHGEOG','DM_PAL','PAL_FO
RECAST_GEOG_PROC',PAL_FGR_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',8,null,null);
DROP TABLE PAL_FGR_PREDICTDATA_TBL;
CREATE COLUMN TABLE PAL_FGR_PREDICTDATA_TBL ( "ID" INT,"X1" DOUBLE);
INSERT INTO PAL_FGR_PREDICTDATA_TBL VALUES (0,1);
INSERT INTO PAL_FGR_PREDICTDATA_TBL VALUES (1,2);
INSERT INTO PAL_FGR_PREDICTDATA_TBL VALUES (2,3);
INSERT INTO PAL_FGR_PREDICTDATA_TBL VALUES (3,4);
INSERT INTO PAL_FGR_PREDICTDATA_TBL VALUES (4,5);
DROP TABLE PAL_FGR_COEFFICIENT_TBL;
CREATE COLUMN TABLE PAL_FGR_COEFFICIENT_TBL ("ID" INT,"Ai" DOUBLE);
INSERT INTO PAL_FGR_COEFFICIENT_TBL VALUES (0,1);
INSERT INTO PAL_FGR_COEFFICIENT_TBL VALUES (1,1.99);
DROP TABLE PAL_FGR_FITTED_TBL;
CREATE COLUMN TABLE PAL_FGR_FITTED_TBL ("ID" INT,"Fitted" DOUBLE);
CALL DM_PAL.PAL_FORECAST_GEOG_PROC(PAL_FGR_PREDICTDATA_TBL,
PAL_FGR_COEFFICIENT_TBL,"#PAL_CONTROL_TBL", PAL_FGR_FITTED_TBL) with overview;
SELECT * FROM PAL_FGR_FITTED_TBL;

```

Expected Result

PAL_FGR_FITTED_TBL:

ID	Fitted
0	1.0
1	3.972369...
2	8.901666...
3	15.77972...
4	24.60086...

3.3.2 Bi-Variate Natural Logarithmic Regression

Bi-variate natural logarithmic regression is an approach to modeling the relationship between a scalar variable y and one variable denoted X . In natural logarithmic regression, data is modeled using natural logarithmic functions, and unknown model parameters are estimated from the data. Such models are called natural logarithmic models.

In PAL, the implementation of natural logarithmic regression is to transform to linear regression and solve it:

$$y = \beta_1 \ln(x) + \beta_0$$

Where β_0 and β_1 are parameters that need to be calculated.

Let $x' = \ln(x)$

Then $y = \beta_0 + \beta_1 \times x'$

Thus, y and x' is a linear relationship and can be solved with the linear regression method.

The implementation also supports calculating the F value and R² to determine statistical significance.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.
- Given the structure as Y and X, there are more than 2 records available for analysis.

LNREGRESSION

This is a logarithmic regression function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LNREGRESSION',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Fitted OUTPUT table type>	OUT
5	<schema_name>	<Significance OUTPUT table type>	OUT
6	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>, <fitted output table>, <significance output table>, <PMML output
table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Variable y
	3rd column	Integer or double	Variable X

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads
ALG	Integer	0	Specifies decomposition method: <ul style="list-style-type: none"> • 0: Doolittle decomposition (LU) • 2: Singular value decomposition (SVD)
ADJUSTED_R2	Integer	0	<ul style="list-style-type: none"> • 0: Does not output adjusted R square • 1: Outputs adjusted R square
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export logarithmic regression model in PMML. • 1: Exports logarithmic regression model in PMML in single row. • 2: Exports logarithmic regression model in PMML in several rows, and the minimum length of each row is 5000 characters.

Name	Data Type	Default Value	Description
SELECTED_FEATURES	Varchar	No default value	A string to specify the features that will be processed. The pattern is "X ₁,X _n ", where X _i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed, and the third column data will be processed as independent variables.
DEPENDENT_VARIABLE	Varchar	No default value	Column name in the data table used as dependent variable. If this parameter is not specified, the second column data will be processed as dependent variables.

Output Tables

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, varchar, or nvarchar	ID Note: If the SELECTED_FEATURES parameter is specified, the column data type must be varchar or nvarchar.	
	2nd column	Integer or double	Value A _i <ul style="list-style-type: none">• A₀: intercept• A₁: beta coefficient for X₁• A₂: beta coefficient for X₂• ...	
Fitted Data	1st column	Integer, bigint, varchar, or nvarchar	ID	
	2nd column	Integer or double	Value Y _i	
Significance	1st column	Varchar or nvarchar	Name	(R ² / F)
	2nd column	Double	Value	
PMML Result	1st column	Integer	ID	
	2nd column	CLOB, varchar, or nvarchar	Logarithmic regression model in PMML format	

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_NLR_DATA_T;
CREATE TYPE PAL_NLR_DATA_T AS TABLE( "ID" INT,"Y" DOUBLE,"X1" DOUBLE);
DROP TYPE PAL_NLR_RESULT_T;
CREATE TYPE PAL_NLR_RESULT_T AS TABLE("ID" INT,"Ai" DOUBLE);
DROP TYPE PAL_NLR_FITTED_T;
CREATE TYPE PAL_NLR_FITTED_T AS TABLE("ID" INT,"Fitted" DOUBLE);
DROP TYPE PAL_NLR_SIGNIFICANCE_T;
CREATE TYPE PAL_NLR_SIGNIFICANCE_T AS TABLE("NAME" varchar(50),"VALUE" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
DROP TYPE PAL_NLR_PMMODEL_T;
CREATE TYPE PAL_NLR_PMMODEL_T AS TABLE("ID" INT,"Model" varchar(5000));
DROP table PAL_NLR_PDATA_TBL;
CREATE column table PAL_NLR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
insert into PAL_NLR_PDATA_TBL values (1,'DM_PAL','PAL_NLR_DATA_T','IN');
insert into PAL_NLR_PDATA_TBL values (2,'DM_PAL','PAL_CONTROL_T','IN');
insert into PAL_NLR_PDATA_TBL values (3,'DM_PAL','PAL_NLR_RESULT_T','OUT');
insert into PAL_NLR_PDATA_TBL values (4,'DM_PAL','PAL_NLR_FITTED_T','OUT');
insert into PAL_NLR_PDATA_TBL values (5,'DM_PAL','PAL_NLR_SIGNIFICANCE_T','OUT');
insert into PAL_NLR_PDATA_TBL values (6,'DM_PAL','PAL_NLR_PMMODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_NLR_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','LNREGRESSION','DM_PAL','PAL_NLR_PR
OC',PAL_NLR_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT',2,null,null);
DROP TABLE PAL_NLR_DATA_TBL;
CREATE COLUMN TABLE PAL_NLR_DATA_TBL ( "ID" INT,"Y" DOUBLE,"X1" DOUBLE);
INSERT INTO PAL_NLR_DATA_TBL VALUES (0,10,1);
INSERT INTO PAL_NLR_DATA_TBL VALUES (1,80,2);
INSERT INTO PAL_NLR_DATA_TBL VALUES (2,130,3);
INSERT INTO PAL_NLR_DATA_TBL VALUES (3,160,4);
INSERT INTO PAL_NLR_DATA_TBL VALUES (4,180,5);
INSERT INTO PAL_NLR_DATA_TBL VALUES (5,190,6);
INSERT INTO PAL_NLR_DATA_TBL VALUES (6,192,7);
DROP TABLE PAL_NLR_RESULTS_TBL;
CREATE COLUMN TABLE PAL_NLR_RESULTS_TBL ("ID" INT,"Ai" DOUBLE);
DROP TABLE PAL_NLR_FITTED_TBL;
CREATE COLUMN TABLE PAL_NLR_FITTED_TBL ("ID" INT,"Fitted" DOUBLE);
DROP TABLE PAL_NLR_SIGNIFICANCE_TBL;
CREATE COLUMN TABLE PAL_NLR_SIGNIFICANCE_TBL ("NAME" varchar(50),"VALUE" DOUBLE);
DROP TABLE PAL_NLR_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_NLR_PMMODEL_TBL("ID" INT, "PMMODEL" VARCHAR(5000));
CALL DM_PAL.PAL_NLR_PROC(PAL_NLR_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_NLR_RESULTS_TBL, PAL_NLR_FITTED_TBL, PAL_NLR_SIGNIFICANCE_TBL,
PAL_NLR_PMMODEL_TBL) with overview;
SELECT * FROM PAL_NLR_RESULTS_TBL;
SELECT * FROM PAL_NLR_FITTED_TBL;
SELECT * FROM PAL_NLR_SIGNIFICANCE_TBL;
SELECT * FROM PAL_NLR_PMMODEL_TBL;
```

Expected Result

PAL_NLR_RESULTS_TBL:

ID	Ai
0	14.861602985473382
1	98.29359745948102

PAL_NLR_FITTED_TBL:

ID	Fitted
0	14.861602985473382
1	82.99353293160685
2	122.84815705185572
3	151.12546287774032
4	173.0590452862983
5	190.9800869979892
6	206.13211186903504

PAL_NLR_SIGNIFICANCE_TBL:

NAME	VALUE
R2	0.9850945373891545
F	330.4474886517365

PAL_NLR_PMMODEL_TBL:

ID	PMMLMODEL
1	<PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.dmg.org/PMML-4_0 PMML-4_0.xsd">

FORECASTWITHLNR

This function performs prediction with the natural logarithmic regression result.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'FORECASTWITHLNR',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Predictive INPUT table type>	IN
2	<schema_name>	<Coefficient INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN

Position	Schema Name	Table Type Name	Parameter Type
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<predictive input table>, <coefficient input table>, <parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Predictive Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Variable X
Coefficient	1st column	Integer, varchar, or nvarchar	ID (start from 0)
	2nd column	Integer, double, varchar, nvarchar, or CLOB	Value A_i or PMML model. Varchar, nvarchar, and CLOB types are only valid for PMML model.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads
MODEL_FORMAT	Integer	0	<ul style="list-style-type: none"> • 0: Coefficients in table • 1: PMML format

Output Table

Table	Column	Column Data Type	Description
Fitted Result	1st column	Integer, bigint, varchar, or nvarchar	ID

Table	Column	Column Data Type	Description
	2nd column	Integer or double	Value Y_i

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_FNLR_PREDICT_T;
CREATE TYPE PAL_FNLR_PREDICT_T AS TABLE( "ID" INT,"X1" DOUBLE);
DROP TYPE PAL_FNLR_COEFFICIENT_T;
CREATE TYPE PAL_FNLR_COEFFICIENT_T AS TABLE("ID" INT,"Ai" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("Name" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
DROP TYPE PAL_FNLR_FITTED_T;
CREATE TYPE PAL_FNLR_FITTED_T AS TABLE("ID" INT,"Fitted" DOUBLE);
DROP table PAL_FNLR_PDATA_TBL;
CREATE column_table PAL_FNLR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_FNLR_PDATA_TBL values (1,'DM_PAL','PAL_FNLR_PREDICT_T','IN');
INSERT INTO PAL_FNLR_PDATA_TBL values (2,'DM_PAL','PAL_FNLR_COEFFICIENT_T','IN');
INSERT INTO PAL_FNLR_PDATA_TBL values (3,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_FNLR_PDATA_TBL values (4,'DM_PAL','PAL_FNLR_FITTED_T','OUT');

CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_FORECAST_LNR_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','FORECASTWITHLNR','DM_PAL','PAL_FOR
ECAST_LNR_PROC',PAL_FNLR_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("Name" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',8,null,null);
DROP TABLE PAL_FNLR_PREDICTDATA_TBL;
CREATE COLUMN TABLE PAL_FNLR_PREDICTDATA_TBL ( "ID" INT,"X1" DOUBLE);
INSERT INTO PAL_FNLR_PREDICTDATA_TBL VALUES (0,1);
INSERT INTO PAL_FNLR_PREDICTDATA_TBL VALUES (1,2);
INSERT INTO PAL_FNLR_PREDICTDATA_TBL VALUES (2,3);
INSERT INTO PAL_FNLR_PREDICTDATA_TBL VALUES (3,4);
INSERT INTO PAL_FNLR_PREDICTDATA_TBL VALUES (4,5);
INSERT INTO PAL_FNLR_PREDICTDATA_TBL VALUES (5,6);
INSERT INTO PAL_FNLR_PREDICTDATA_TBL VALUES (6,7);
DROP TABLE PAL_FNLR_COEFFICIENT_TBL;
CREATE COLUMN TABLE PAL_FNLR_COEFFICIENT_TBL ("ID" INT,"Ai" DOUBLE);
INSERT INTO PAL_FNLR_COEFFICIENT_TBL VALUES (0,14.86160299);
INSERT INTO PAL_FNLR_COEFFICIENT_TBL VALUES (1,98.29359746);
DROP TABLE PAL_FNLR_FITTED_TBL;
CREATE COLUMN TABLE PAL_FNLR_FITTED_TBL ("ID" INT,"Fitted" DOUBLE);
CALL DM_PAL.PAL_FORECAST_LNR_PROC(PAL_FNLR_PREDICTDATA_TBL,
PAL_FNLR_COEFFICIENT_TBL, "#PAL_CONTROL_TBL", PAL_FNLR_FITTED_TBL) with overview;
SELECT * FROM PAL_FNLR_FITTED_TBL;

```

Expected Result

PAL_FNLR_FITTED_TBL:

ID	Fitted
0	14.86160299
1	82.9935329364932
2	122.8481570569525
3	151.1254628829864
4	173.05904529166017
5	190.98008700344568
6	206.13211187457156

3.3.3 Exponential Regression

Exponential regression is an approach to modeling the relationship between a scalar variable y and one or more variables denoted X . In exponential regression, data is modeled using exponential functions, and unknown model parameters are estimated from the data. Such models are called exponential models.

In PAL, the implementation of exponential regression is to transform to linear regression and solve it:

$$y = \beta_0 \times \exp(\beta_1 \times x_1 + \beta_2 \times x_2 + \dots + \beta_n \times x_n)$$

Where $\beta_0 \dots \beta_n$ are parameters that need to be calculated.

The steps are:

1. Put natural logarithmic operation on both sides:
$$\ln(y) = \ln(\beta_0 \times \exp(\beta_1 \times x_1 + \beta_2 \times x_2 + \dots + \beta_n \times x_n))$$
2. Transform it into: $\ln(y) = \ln(\beta_0) + \beta_1 \times x_1 + \beta_2 \times x_2 + \dots + \beta_n \times x_n$
3. Let $y' = \ln(y)$, $\beta_0' = \ln(\beta_0)$
$$y' = \beta_0' + \beta_1 \times x_1 + \beta_2 \times x_2 + \dots + \beta_n \times x_n$$

Thus, y' and $x_1 \dots x_n$ is a linear relationship and can be solved using the linear regression method.

The implementation also supports calculating the F value and R^2 to determine statistical significance.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.
- Given the structure as Y and X1...Xn, there are more than n+1 records available for analysis.

EXPREGRESSION

This is an exponential regression function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'EXPREGRESSION',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Fitted OUTPUT table type>	OUT
5	<schema_name>	<Significance OUTPUT table type>	OUT
6	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>, <fitted output table>, <significance output table>, <PMML output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Variable y
	Other columns	Integer or double	Variable Xn

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads
ALG	Integer	0	<p>Specifies decomposition method:</p> <ul style="list-style-type: none"> • 0: Doolittle decomposition (LU) • 2: Singular value decomposition (SVD)
ADJUSTED_R2	Integer	0	<ul style="list-style-type: none"> • 0: Does not output adjusted R square • 1: Outputs adjusted R square
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export exponential regression model in PMML. • 1: Exports exponential regression model in PMML in single row. • 2: Exports exponential regression model in PMML in several rows, and the minimum length of each row is 5000 characters.
SELECTED_FEATURES	Varchar	No default value	<p>A string to specify the features that will be processed. The pattern is “X₁....X_n”, where X_i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed, and all columns except for the first and second column data will be processed as independent variables.</p>
DEPENDENT_VARIABLE	Varchar	No default value	<p>Column name in the data table used as dependent variable. If this parameter is not specified, the second column data will be processed as dependent variables.</p>

Output Tables

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, varchar, or nvarchar	ID Note: If the SELECTED_FEATURES parameter is specified, the column data type must be varchar or nvarchar.	
	2nd column	Integer or double	Value A_i <ul style="list-style-type: none"> • A_0: the intercept • A_1: the beta coefficient for X_1 • A_2: the beta coefficient for X_2 • ... 	
Fitted Data	1st column	Integer, bigint, varchar, or nvarchar	ID	
	2nd column	Integer or double	Value Y_i	
Significance	1st column	Varchar or nvarchar	Name	(R^2 / F)
	2nd column	Double	Value	
PMML Result	1st column	Integer	ID	
	2nd column	CLOB, varchar, or nvarchar	Exponential regression model in PMML format	

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ER_DATA_T;
CREATE TYPE PAL_ER_DATA_T AS TABLE( "ID" INT,"Y" DOUBLE,"X1" DOUBLE, "X2"
DOUBLE);
DROP TYPE PAL_ER_RESULT_T;
CREATE TYPE PAL_ER_RESULT_T AS TABLE("ID" INT,"Ai" DOUBLE);
DROP TYPE PAL_ER_FITTED_T;
CREATE TYPE PAL_ER_FITTED_T AS TABLE("ID" INT,"Fitted" DOUBLE);
DROP TYPE PAL_ER_SIGNIFICANCE_T;
CREATE TYPE PAL_ER_SIGNIFICANCE_T AS TABLE("NAME" varchar(50), "VALUE" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_ER_PMMODEL_T;
CREATE TYPE PAL_ER_PMMODEL_T AS TABLE("ID" INT,"Model" varchar(5000));
DROP table PAL_ER_PDATA_TBL;
CREATE column table PAL_ER_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ER_PDATA_TBL values (1,'DM_PAL','PAL_ER_DATA_T','in');

```

```

INSERT INTO PAL_ER_PDATA_TBL values (2,'DM_PAL','PAL_CONTROL_T','in');
INSERT INTO PAL_ER_PDATA_TBL values (3,'DM_PAL','PAL_ER_RESULT_T','out');
INSERT INTO PAL_ER_PDATA_TBL values (4,'DM_PAL','PAL_ER_FITTED_T','out');
INSERT INTO PAL_ER_PDATA_TBL values (5,'DM_PAL','PAL_ER_SIGNIFICANCE_T','out');
INSERT INTO PAL_ER_PDATA_TBL values (6,'DM_PAL','PAL_ER_PMMODEL_T','out');

CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_EXPR_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','EXPREGRESSION','DM_PAL','PAL_EXPR_
PROC',PAL_ER_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT',2,null,null);
DROP TABLE PAL_ER_DATA_TBL;
CREATE COLUMN TABLE PAL_ER_DATA_TBL ( "ID" INT, "Y" DOUBLE, "X1" DOUBLE, "X2"
DOUBLE );
INSERT INTO PAL_ER_DATA_TBL VALUES (0,0.5,0.13,0.33);
INSERT INTO PAL_ER_DATA_TBL VALUES (1,0.15,0.14,0.34);
INSERT INTO PAL_ER_DATA_TBL VALUES (2,0.25,0.15,0.36);
INSERT INTO PAL_ER_DATA_TBL VALUES (3,0.35,0.16,0.35);
INSERT INTO PAL_ER_DATA_TBL VALUES (4,0.45,0.17,0.37);
INSERT INTO PAL_ER_DATA_TBL VALUES (5,0.55,0.18,0.38);
INSERT INTO PAL_ER_DATA_TBL VALUES (6,0.65,0.19,0.39);
INSERT INTO PAL_ER_DATA_TBL VALUES (7,0.75,0.19,0.31);
INSERT INTO PAL_ER_DATA_TBL VALUES (8,0.85,0.11,0.32);
INSERT INTO PAL_ER_DATA_TBL VALUES (9,0.95,0.12,0.33);
DROP TABLE PAL_ER_RESULTS_TBL;
CREATE COLUMN TABLE PAL_ER_RESULTS_TBL LIKE PAL_ER_RESULT_T;
DROP TABLE PAL_ER_FITTED_TBL;
CREATE COLUMN TABLE PAL_ER_FITTED_TBL LIKE PAL_ER_FITTED_T;
DROP TABLE PAL_ER_SIGNIFICANCE_TBL;
CREATE COLUMN TABLE PAL_ER_SIGNIFICANCE_TBL LIKE PAL_ER_SIGNIFICANCE_T;
DROP TABLE PAL_ER_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_ER_PMMODEL_TBL LIKE PAL_ER_PMMODEL_T;
CALL DM_PAL.PAL_EXPR_PROC(PAL_ER_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_ER_RESULTS_TBL, PAL_ER_FITTED_TBL, PAL_ER_SIGNIFICANCE_TBL,
PAL_ER_PMMODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ER_RESULTS_TBL;
SELECT * FROM PAL_ER_FITTED_TBL;
SELECT * FROM PAL_ER_SIGNIFICANCE_TBL;
SELECT * FROM PAL_ER_PMMODEL_TBL;

```

Expected Result

PAL_ER_RESULTS_TBL:

ID	Ai
0	2.72773...
1	2.67414...
2	-6.1804...

PAL_ER_FITTED_TBL:

ID	Fitted
0	0.502363029...
1	0.485053971...
2	0.440272142...
3	0.481034365...
4	0.436623640...
5	0.421579651...
6	0.407054007...
7	0.667393613...
8	0.506560860...
9	0.489107165...

PAL_ER_SIGNIFICANCE_TBL:

NAME	VALUE
R2	0.0584344...
F	0.2172133...

PAL_ER_PMMODEL_TBL:

ID	PMMLMODEL
1	<PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.dmg.org/PMML-4_0 PMML-v4_0.xsd">

FORECASTWITHEXPR

This function performs prediction with the exponential regression result.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'FORECASTWITHEXPR',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Coefficient INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <coefficient input table>, <parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Predictive Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	Other columns	Integer or double	Variable X_n
Coefficient	1st column	Integer, varchar, or nvarchar	ID (start from 0)
	2nd column	Integer, double, varchar, nvarchar, or CLOB	Value A_i or PMML model. Varchar, nvarchar and CLOB types are only valid for PMML model.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads
MODEL_FORMAT	Integer	0	<ul style="list-style-type: none">0: Coefficients in table1: PMML format

Output Table

Table	Column	Column Data Type	Description
Fitted Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Value Y_i

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and

- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_FER_PREDICT_T;
CREATE TYPE PAL_FER_PREDICT_T AS TABLE( "ID" INT, "X1" DOUBLE, "X2" DOUBLE);
DROP TYPE PAL_FER_COEFFICIENT_T;
CREATE TYPE PAL_FER_COEFFICIENT_T AS TABLE("ID" INT, "Ai" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_FER_FITTED_T;
CREATE TYPE PAL_FER_FITTED_T AS TABLE("ID" INT, "Fitted" DOUBLE);
DROP table PAL_FER_PDATA_TBL;
CREATE column table PAL_FER_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
insert into PAL_FER_PDATA_TBL values (1,'DM_PAL','PAL_FER_PREDICT_T','in');
insert into PAL_FER_PDATA_TBL values (2,'DM_PAL','PAL_FER_COEFFICIENT_T','in');
insert into PAL_FER_PDATA_TBL values (3,'DM_PAL','PAL_CONTROL_T','in');
insert into PAL_FER_PDATA_TBL values (4,'DM_PAL','PAL_FER_FITTED_T','out');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_FORECAST_EXPR_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','FORECASTWITHEXPR','DM_PAL','PAL_FO
RECAST_EXPR_PROC',PAL_FER_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',8,null,null);
DROP TABLE PAL_FER_PREDICTDATA_TBL;
CREATE COLUMN TABLE PAL_FER_PREDICTDATA_TBL ("ID" INT, "X1" DOUBLE, "X2" DOUBLE);
INSERT INTO PAL_FER_PREDICTDATA_TBL VALUES (0,0.5,0.3);
INSERT INTO PAL_FER_PREDICTDATA_TBL VALUES (1,4,0.4);
INSERT INTO PAL_FER_PREDICTDATA_TBL VALUES (2,0,1.6);
INSERT INTO PAL_FER_PREDICTDATA_TBL VALUES (3,0.3,0.45);
INSERT INTO PAL_FER_PREDICTDATA_TBL VALUES (4,0.4,1.7);
DROP TABLE PAL_FER_COEFFICIENT_TBL;
CREATE COLUMN TABLE PAL_FER_COEFFICIENT_TBL ("ID" INT, "Ai" DOUBLE);
INSERT INTO PAL_FER_COEFFICIENT_TBL VALUES (0,1.7120914258645001);
INSERT INTO PAL_FER_COEFFICIENT_TBL VALUES (1,0.2652771198483208);
INSERT INTO PAL_FER_COEFFICIENT_TBL VALUES (2,-3.471103742302148);
DROP TABLE PAL_FER_FITTED_TBL;
CREATE COLUMN TABLE PAL_FER_FITTED_TBL ("ID" INT, "Fitted" DOUBLE);
CALL DM_PAL.PAL_FORECAST_EXPR_PROC(PAL_FER_PREDICTDATA_TBL,
PAL_FER_COEFFICIENT_TBL, "#PAL_CONTROL_TBL", PAL_FER_FITTED_TBL) with overview;
SELECT * FROM PAL_FER_FITTED_TBL;

```

Expected Result

PAL_FER_FITTED_TBL:

ID	Fitted
0	0.6900598931338715
1	1.2341502316656843
2	0.00663066413618...
3	0.3887970208571841
4	0.00521065435714...

3.3.4 Multiple Linear Regression

Linear regression is an approach to modeling the linear relationship between a variable $y \in \mathbb{R}$, usually referred to as dependent variable, and one or more variables, usually referred to as independent variables, denoted as predictor vector $x \in \mathbb{R}^n$. In linear regression, data are modeled using linear functions, and unknown model parameters are estimated from the data. Such models are called linear models.

$$\begin{bmatrix} 1 & x_1^T \\ \dots & \dots \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta \end{bmatrix} = y,$$

Assume we have m observation pairs (x_i, y_i) . Then we obtain an overdetermined linear system

with $\begin{bmatrix} 1 & x_1^T \\ \dots & \dots \end{bmatrix}$ is $m \times (n+1)$ matrix, $\begin{bmatrix} \beta_0 \\ \beta \end{bmatrix}$ is $(n+1) \times 1$ matrix, and y is $m \times 1$ matrix, where $m > n+1$. Since equality is

usually not exactly satisfiable, when $m > n+1$, the least squares solution $\begin{bmatrix} \beta_0 \\ \beta \end{bmatrix}$ minimizes the squared Euclidean norm of the residual vector $r(\begin{bmatrix} \beta_0 \\ \beta \end{bmatrix})$ so that

$$\min_{(\beta_0, \beta) \in \mathbb{R}^{n+1}} \|r(\begin{bmatrix} \beta_0 \\ \beta \end{bmatrix})\|_2^2 = \min_{(\beta_0, \beta) \in \mathbb{R}^{n+1}} (\sum_{i=1}^m (y_i - \beta_0 - x_i^T \beta)^2)$$

Elastic net regularization for multiple linear regression seeks to find $\begin{bmatrix} \beta_0 \\ \beta \end{bmatrix}$ that minimizes:

$$\min_{(\beta_0, \beta) \in \mathbb{R}^{n+1}} (\frac{1}{2m} \sum_{i=1}^m (y_i - \beta_0 - x_i^T \beta)^2 + \lambda * P_\alpha(\beta))$$

Where $P_\alpha(\beta) = (1-\alpha) \frac{1}{2} \|\beta\|_2^2 + \alpha \|\beta\|_1$.

Here $\alpha \in [0, 1]$ and $\lambda \geq 0$. If $\alpha=0$, we have the ridge regularization; if $\alpha=1$, we have the LASSO regularization.

The implementation also supports calculating F and R^2 to determine statistical significance.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.
- Given n independent variables, there must be at least $n+1$ records available for analysis.

LRREGRESSION

This is a multiple linear regression function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LRREGRESSION',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Fitted OUTPUT table type>	OUT
5	<schema_name>	<Significance OUTPUT table type>	OUT
6	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>, <fitted output table>, <significance output table>, <PMML output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Variable y
	Other columns	Integer or double	Variable Xm

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
THREAD_NUMBER	Integer	1	Specifies the number of threads.	
ALG	Integer	1	<p>Specifies algorithms for solving the least square problem:</p> <ul style="list-style-type: none"> • 0: LU (fast but numerically unstable) • 1: QR decomposition (numerically stable, but fails when A is rank-deficient) • 2: SVD (numerically stable and can handle rank deficiency but computationally expensive) • 4: Cyclical coordinate descent method to solve elastic net regularized multiple linear regression 	The value 4 is supported only when VARIABLE_SELECTION is 0.
VARIABLE_SELECTION	Integer	0	<ul style="list-style-type: none"> • 0: All variables are included • 1: Forward selection • 2: Backward selection 	The value 1 or 2 are supported only when ALG is not 4.
ALPHA_TO_ENTER	Double	0.05	P-value for forward selection.	Only valid when VARIABLE_SELECTION is 1.
ALPHA_TO_REMOVE	Double	0.1	P-value for backward selection.	Only valid when VARIABLE_SELECTION is 2.
ENET_LAMBDA	Double	No default value	Penalized weight. The value should be equal to or greater than 0.	Only valid when ALG is 4.

Name	Data Type	Default Value	Description	Dependency
ENET_ALPHA	Double	1.0	The elastic net mixing parameter. The value range is between 0 and 1 inclusively. <ul style="list-style-type: none">• 0: Ridge penalty• 1: LASSO penalty	Only valid when ALG is 4.
MAX_ITERATION	Double	1e5	Maximum number of passes over training data. If convergence is not reached after the specified number of iterations, an error will be generated.	Only valid when ALG is 4.
THRESHOLD	Double	1.0e-7	Convergence threshold for coordinate descent.	Only valid when ALG is 4.
STAT_INF	Integer	0	Specifies whether to output t-value and Pr(> t) of coefficients in the Result table or not. <ul style="list-style-type: none">• 0: No• 1: Yes	
ADJUSTED_R2	Integer	0	Specifies whether to output adjusted R square or not. <ul style="list-style-type: none">• 0: No• 1: Yes	

Name	Data Type	Default Value	Description	Dependency
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export multiple linear regression model in PMML. • 1: Exports multiple linear regression model in PMML in single row. • 2: Exports multiple linear regression model in PMML in several rows, and the minimum length of each row is 5000 characters. 	
SELECTED_FEATURES	Varchar	No default value	A string to specify the features that will be processed. The pattern is " X_1, \dots, X_n ", where X_i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed, and all columns except for the first and second column data will be processed as independent variables.	
DEPENDENT_VARIABLE	Varchar	No default value	Column name in the data table used as dependent variable. If this parameter is not specified, the second column data will be processed as dependent variables.	

Output Tables

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, varchar, or nvarchar	ID Note: If the SELECTED_FEATURES parameter is specified, the column data type must be varchar or nvarchar.	
	2nd column	Integer or double	Value A_i <ul style="list-style-type: none"> • A_0: the intercept • A_1: the beta coefficient for X_1 • A_2: the beta coefficient for X_2 • ... 	
	3rd column	Double	t-value of coefficients. Note: This column is available only when the STAT_INF parameter is set to 1.	
	4th column	Double	$Pr(> t)$ of coefficients. Note: This column is available only when the STAT_INF parameter is set to 1.	
Fitted Data	1st column	Integer, bigint, varchar, or nvarchar	ID	
	2nd column	Integer or double	Value Y_i	
Significance	1st column	VARCHAR or nvarchar	Name	(R^2 / F)
	2nd column	Double	Value	
PMML Result	1st column	Integer	ID	
	2nd column	CLOB, varchar, or nvarchar	Multiple linear regression model in PMML format	

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1: Fitting multiple linear regression model without penalties

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_MLR_DATA_T;
CREATE TYPE PAL_MLR_DATA_T AS TABLE( "ID" INT, "Y" DOUBLE, "V1" DOUBLE, "V2" DOUBLE);
DROP TYPE PAL_MLR_RESULT_T;
CREATE TYPE PAL_MLR_RESULT_T AS TABLE("Coefficient" varchar(50), "CoefficientValue" DOUBLE);
DROP TYPE PAL_MLR_FITTED_T;
CREATE TYPE PAL_MLR_FITTED_T AS TABLE("ID" INT, "Fitted" DOUBLE);
DROP TYPE PAL_MLR_SIGNIFICANCE_T;
CREATE TYPE PAL_MLR_SIGNIFICANCE_T AS TABLE("NAME" varchar(50), "VALUE" DOUBLE);
DROP TYPE PAL_MLR_PMMODEL_T;
CREATE TYPE PAL_MLR_PMMODEL_T AS TABLE("ID" INT, "Model" varchar(5000));
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TABLE PAL_MLR_PDATA_TBL;
CREATE COLUMN TABLE PAL_MLR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_MLR_PDATA_TBL values (1,'DM_PAL','PAL_MLR_DATA_T','IN');
INSERT INTO PAL_MLR_PDATA_TBL values (2,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_MLR_PDATA_TBL values (3,'DM_PAL','PAL_MLR_RESULT_T','OUT');
INSERT INTO PAL_MLR_PDATA_TBL values (4,'DM_PAL','PAL_MLR_FITTED_T','OUT');
INSERT INTO PAL_MLR_PDATA_TBL values (5,'DM_PAL','PAL_MLR_SIGNIFICANCE_T','OUT');
INSERT INTO PAL_MLR_PDATA_TBL values (6,'DM_PAL','PAL_MLR_PMMODEL_T','OUT');

CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_LR_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'LRREGRESSION', 'DM_PAL', 'PAL_LR_PROC',
'PAL_MLR_PDATA_TBL');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT',0,null,null);
DROP TABLE PAL_MLR_DATA_TBL;
CREATE COLUMN TABLE PAL_MLR_DATA_TBL ( "ID" INT, "Y" DOUBLE, "V1" DOUBLE, "V2" DOUBLE);
INSERT INTO PAL_MLR_DATA_TBL VALUES (0,0.5,0.13,0.33);
INSERT INTO PAL_MLR_DATA_TBL VALUES (1,0.15,0.14,0.34);
INSERT INTO PAL_MLR_DATA_TBL VALUES (2,0.25,0.15,0.36);
INSERT INTO PAL_MLR_DATA_TBL VALUES (3,0.35,0.16,0.35);
INSERT INTO PAL_MLR_DATA_TBL VALUES (4,0.45,0.17,0.37);
INSERT INTO PAL_MLR_DATA_TBL VALUES (5,0.55,0.18,0.38);
INSERT INTO PAL_MLR_DATA_TBL VALUES (6,0.65,0.19,0.39);
INSERT INTO PAL_MLR_DATA_TBL VALUES (7,0.75,0.19,0.31);
INSERT INTO PAL_MLR_DATA_TBL VALUES (8,0.85,0.11,0.32);
INSERT INTO PAL_MLR_DATA_TBL VALUES (9,0.95,0.12,0.33);
DROP TABLE PAL_MLR_RESULTS_TBL;
CREATE COLUMN TABLE PAL_MLR_RESULTS_TBL ("Coefficient" varchar(50), "CoefficientValue" DOUBLE);
DROP TABLE PAL_MLR_FITTED_TBL;
CREATE COLUMN TABLE PAL_MLR_FITTED_TBL ("ID" INT, "Fitted" DOUBLE);
DROP TABLE PAL_MLR_SIGNIFICANCE_TBL;
CREATE COLUMN TABLE PAL_MLR_SIGNIFICANCE_TBL ("NAME" varchar(50), "VALUE" DOUBLE);
DROP TABLE PAL_MLR_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_MLR_PMMODEL_TBL ("ID" INT, "PMMODEL" VARCHAR(5000));
CALL DM_PAL.PAL_LR_PROC(PAL_MLR_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_MLR_RESULTS_TBL, PAL_MLR_FITTED_TBL, PAL_MLR_SIGNIFICANCE_TBL,
PAL_MLR_PMMODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_MLR_RESULTS_TBL;
SELECT * FROM PAL_MLR_FITTED_TBL;
SELECT * FROM PAL_MLR_SIGNIFICANCE_TBL;
```

Expected Results

PAL_MLR_RESULTS_TBL:

Coefficient	CoefficientValue
PAL_INTERCEPT	1.7120914258645206
V1	0.26527711984840835
V2	-3.4711037423022284

PAL_MLR_FITTED_TBL:

ID	Fitted
0	0.601113216...
1	0.569054950...
2	0.502285646...
3	0.539649455...
4	0.472880151...
5	0.440821885...
6	0.408763619...
7	0.686451918...
8	0.630518711...
9	0.598460445...

PAL_MLR_SIGNIFICANCE_TBL:

NAME	VALUE
R2	0.11685395777203476
F	0.4631044387294608
AIC	7.039446372412275
BIC	8.249786744388459

Example 2: Fitting multiple linear regression model with elastic net penalties

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_ENET_MLR_DATA_T;
CREATE TYPE PAL_ENET_MLR_DATA_T AS TABLE( "ID" INT, "Y" DOUBLE, "V1" DOUBLE, "V2"
DOUBLE, "V3" DOUBLE);
DROP TYPE PAL_ENET_MLR_RESULT_T;
CREATE TYPE PAL_ENET_MLR_RESULT_T AS TABLE("Coefficient" varchar(50),
"CoefficientValue" DOUBLE);
DROP TYPE PAL_ENET_MLR_FITTED_T;
CREATE TYPE PAL_ENET_MLR_FITTED_T AS TABLE("ID" INT, "Fitted" DOUBLE);
DROP TYPE PAL_ENET_MLR_SIGNIFICANCE_T;
CREATE TYPE PAL_ENET_MLR_SIGNIFICANCE_T AS TABLE("NAME" varchar(50), "VALUE"
DOUBLE);
DROP TYPE PAL_ENET_MLR_PMMODEL_T;
CREATE TYPE PAL_ENET_MLR_PMMODEL_T AS TABLE("ID" INT, "Model" VARCHAR(5000));
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS"
INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TABLE PAL_ENET_MLR_PDATA_TBL;
```

```

CREATE COLUMN TABLE PAL_ENET_MLR_PDATA_TBL("POSITION" INT,"SCHEMA_NAME"
NVARCHAR(256),"TYPE_NAME" NVARCHAR(256),"PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ENET_MLR_PDATA_TBL values (1,'DM_PAL','PAL_ENET_MLR_DATA_T',
'IN');
INSERT INTO PAL_ENET_MLR_PDATA_TBL values (2,'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_ENET_MLR_PDATA_TBL values (3,'DM_PAL','PAL_ENET_MLR_RESULT_T',
'OUT');
INSERT INTO PAL_ENET_MLR_PDATA_TBL values (4,'DM_PAL','PAL_ENET_MLR_FITTED_T',
'OUT');
INSERT INTO PAL_ENET_MLR_PDATA_TBL values
(5,'DM_PAL','PAL_ENET_MLR_SIGNIFICANCE_T', 'OUT');
INSERT INTO PAL_ENET_MLR_PDATA_TBL values
(6,'DM_PAL','PAL_ENET_MLR_PMMODEL_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_ENET_MLR_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'LRREGRESSION', 'DM_PAL',
'PAL_ENET_MLR_PROC', PAL_ENET_MLR_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME"
VARCHAR(100),"INTARGS" INT,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALG', 4, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ENET_LAMBDA', null, 0.003194, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ENET_ALPHA', null, 0.95, null);
DROP TABLE PAL_ENET_MLR_DATA_TBL;
CREATE COLUMN TABLE PAL_ENET_MLR_DATA_TBL ( "ID" INT,"Y" DOUBLE,"V1" DOUBLE,"V2"
DOUBLE,"V3" DOUBLE);
INSERT INTO PAL_ENET_MLR_DATA_TBL VALUES (0, 1.2, 0.1, 0.205, 0.9);
INSERT INTO PAL_ENET_MLR_DATA_TBL VALUES (1, 0.2, -1.705, -3.4, 1.7);
INSERT INTO PAL_ENET_MLR_DATA_TBL VALUES (2, 1.1, 0.4, 0.8, 0.5);
INSERT INTO PAL_ENET_MLR_DATA_TBL VALUES (3, 1.1, 0.1, 0.201, 0.8);
INSERT INTO PAL_ENET_MLR_DATA_TBL VALUES (4, 0.3, -0.306, -0.6, 0.2);
DROP TABLE PAL_ENET_MLR_RESULTS_TBL;
CREATE COLUMN TABLE PAL_ENET_MLR_RESULTS_TBL ("Coefficient"
varchar(50),"CoefficientValue" DOUBLE);
DROP TABLE PAL_ENET_MLR_FITTED_TBL;
CREATE COLUMN TABLE PAL_ENET_MLR_FITTED_TBL ("ID" INT,"Fitted" DOUBLE);
DROP TABLE PAL_ENET_MLR_SIGNIFICANCE_TBL;
CREATE COLUMN TABLE PAL_ENET_MLR_SIGNIFICANCE_TBL ("NAME" varchar(50),"VALUE"
DOUBLE);
DROP TABLE PAL_ENET_MLR_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_ENET_MLR_PMMODEL_TBL ("ID" INT, "PMMODEL"
VARCHAR(5000));
CALL DM_PAL.PAL_ENET_MLR_PROC(PAL_ENET_MLR_DATA_TBL,"#PAL_CONTROL_TBL",
PAL_ENET_MLR_RESULTS_TBL, PAL_ENET_MLR_FITTED_TBL,
PAL_ENET_MLR_SIGNIFICANCE_TBL, PAL_ENET_MLR_PMMODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ENET_MLR_RESULTS_TBL;
SELECT * FROM PAL_ENET_MLR_FITTED_TBL;
SELECT * FROM PAL_ENET_MLR_SIGNIFICANCE_TBL;

```

Expected Results

PAL_ENET_MLR_RESULTS_TBL:

Coefficient	CoefficientValue
_PAL_INTERCEPT_	0.4268381704611707
V1	0
V2	0.4254020093651538
V3	0.7205810638683869

PAL_ENET_MLR_FITTED_TBL:

ID	Fitted
0	1.1625685398625754
1	0.2054591471959053
2	1.1274503098874873
3	1.0888088254382762
4	0.3157131776157558

PAL_ENET_MLR_SIGNIFICANCE_TBL:

NAME	VALUE
R2	0.9973031831773201
F	355.20183495294776
AIC	-15.703223893349...
BIC	-17.265472243612...

FORECASTWITHLR

This function performs prediction with the linear regression result.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'FORECASTWITHLR',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Coefficient INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <coefficient input table>, <parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Predictive Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	Other columns	Integer or double	Variable X_n
Coefficient	1st column	Integer, varchar, or nvarchar	ID (start from 0)
	2nd column	Integer, double, varchar, nvarchar, or CLOB	Value A_i or PMML model. Varchar, nvarchar, and CLOB types are only valid for PMML model.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads
MODEL_FORMAT	Integer	0	<ul style="list-style-type: none"> 0: Coefficients in table 1: PMML format

Output Table

Table	Column	Column Data Type	Description
Fitted Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Value Y_i

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_FMLR_PREDICT_T;
CREATE TYPE PAL_FMLR_PREDICT_T AS TABLE( "ID" INT, "V1" DOUBLE, "V2" DOUBLE);
DROP TYPE PAL_FMLR_COEFFICIENT_T;
CREATE TYPE PAL_FMLR_COEFFICIENT_T AS TABLE("Coefficient" varchar(50),
"CoefficientValue" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
DROP TYPE PAL_FMLR_FITTED_T;
CREATE TYPE PAL_FMLR_FITTED_T AS TABLE("ID" INT,"Fitted" DOUBLE);
```

```

DROP table PAL_FMLR_PDATA_TBL;
CREATE column table PAL_FMLR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_FMLR_PDATA_TBL values (1,'DM_PAL','PAL_FMLR_PREDICT_T','IN');
INSERT INTO PAL_FMLR_PDATA_TBL values (2,'DM_PAL','PAL_FMLR_COEFFICIENT_T','IN');
INSERT INTO PAL_FMLR_PDATA_TBL values (3,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_FMLR_PDATA_TBL values (4,'DM_PAL','PAL_FMLR_FITTED_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_FORECAST_LR_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','FORECASTWITHLR','DM_PAL','PAL_FORE
CAST_LR_PROC',PAL_FMLR_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',2,null,null);
DROP TABLE PAL_FMLR_PREDICTDATA_TBL;
CREATE COLUMN TABLE PAL_FMLR_PREDICTDATA_TBL ("ID" INT, "V1" DOUBLE, "V2"
DOUBLE);
INSERT INTO PAL_FMLR_PREDICTDATA_TBL VALUES (0,0.5,0.3);
INSERT INTO PAL_FMLR_PREDICTDATA_TBL VALUES (1,4,0.4);
INSERT INTO PAL_FMLR_PREDICTDATA_TBL VALUES (2,0,1.6);
INSERT INTO PAL_FMLR_PREDICTDATA_TBL VALUES (3,0.3,0.45);
INSERT INTO PAL_FMLR_PREDICTDATA_TBL VALUES (4,0.4,1.7);
DROP TABLE PAL_FMLR_COEFICIENT_TBL;
CREATE COLUMN TABLE PAL_FMLR_COEFICIENT_TBL ("Coefficient" varchar(50),
"CoefficientValue" DOUBLE);
INSERT INTO PAL_FMLR_COEFICIENT_TBL VALUES ('__PAL_INTERCEPT__', 1.7120914258645001);
INSERT INTO PAL_FMLR_COEFICIENT_TBL VALUES ('V1', 0.2652771198483208);
INSERT INTO PAL_FMLR_COEFICIENT_TBL VALUES ('V2', -3.471103742302148);
DROP TABLE PAL_FMLR_FITTED_TBL;
CREATE COLUMN TABLE PAL_FMLR_FITTED_TBL ("ID" INT,"Fitted" DOUBLE);
CALL DM.PAL.PAL_FORECAST_LR_PROC(PAL_FMLR_PREDICTDATA_TBL,
PAL_FMLR_COEFICIENT_TBL, "#PAL_CONTROL_TBL", PAL_FMLR_FITTED_TBL) with overview;
SELECT * FROM PAL_FMLR_FITTED_TBL;

```

Expected Result

PAL_FMLR_FITTED_TBL:

ID	Fitted
0	0.803398...
1	1.384758...
2	-3.84167...
3	0.229677...
4	-4.08267...

3.3.5 Polynomial Regression

Polynomial regression is an approach to modeling the relationship between a scalar variable y and a variable denoted X . In polynomial regression, data is modeled using polynomial functions, and unknown model parameters are estimated from the data. Such models are called polynomial models.

In PAL, the implementation of exponential regression is to transform to linear regression and solve it:

$$y = \beta_0 + \beta_1 \times x + \beta_2 \times x^2 + \dots + \beta_n \times x^n$$

Where $\beta_0 \dots \beta_n$ are parameters that need to be calculated.

Let $x = x_1', x^2 = x_2', \dots, x^n = x_n'$, and then

$$y' = \beta_0' + \beta_1 \times x_1 + \beta_2 \times x_2 + \dots + \beta_n \times x_n$$

So, y' and $x_1 \dots x_n$ is a linear relationship and can be solved using the linear regression method.

The implementation also supports calculating the F value and R² to determine statistical significance.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.
- Given the structure as Y and X1...Xn, there are more than n+1 records available for analysis.

POLYNOMIALREGRESSION

This is a polynomial regression function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'POLYNOMIALREGRESSION',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Fitted OUTPUT table type>	OUT
5	<schema_name>	<Significance OUTPUT table type>	OUT
6	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name> (<input table>, <parameter table>, <result
output table>, <fitted output table>, <significance output table>, <PMML output
table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Variable y
	3rd column	Integer or double	Variable X

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
POLYNOMIAL_NUM	Integer	This is a mandatory parameter to create a polynomial of degree POLYNOMIAL_NUM model. Note: POLYNOMIAL_NUM replaces VARIABLE_NUM.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads.
ALG	Integer	0	Specifies decomposition method: <ul style="list-style-type: none">• 0: Doolittle decomposition (LU)• 2: Singular value decomposition (SVD)
ADJUSTED_R2	Integer	0	<ul style="list-style-type: none">• 0: Does not output adjusted R square• 1: Outputs adjusted R square

Name	Data Type	Default Value	Description
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export polynomial regression model in PMML. • 1: Exports polynomial regression model in PMML in single row. • 2: Exports polynomial regression model in PMML in several rows, and the minimum length of each row is 5000 characters.
SELECTED_FEATURES	Varchar	No default value	A string to specify the features that will be processed. The pattern is " X_1, \dots, X_n ", where X_i is the corresponding column name in the data table. If this parameter is not specified, all the features will be processed, and the third column data will be processed as independent variables.
DEPENDENT_VARIABLE	Varchar	No default value	Column name in the data table used as dependent variable. If this parameter is not specified, the second column data will be processed as dependent variables.

Output Tables

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, varchar, or nvarchar	ID Note: If the SELECTED_FEATURES parameter is specified, the column data type must be varchar or nvarchar.	

Table	Column	Column Data Type	Description	Constraint
	2nd column	Integer or double	Value A_i <ul style="list-style-type: none"> A_0: the intercept A_1: the beta coefficient for X_1 A_2: the beta coefficient for X_2 ... 	
Fitted Data	1st column	Integer, bigint, varchar, or nvarchar	ID	
	2nd column	Integer or double	Value Y_i	
Significance	1st column	Varchar or nvarchar	Name	(R^2 / F)
	2nd column	Double	Value	
PMML Result	1st column	Integer	ID	
	2nd column	CLOB, varchar, or nvarchar	Polynomial regression model in PMML format	

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_PR_DATA_T;
CREATE TYPE PAL_PR_DATA_T AS TABLE( "ID" INT,"Y" DOUBLE,"X1" DOUBLE);
DROP TYPE PAL_PR_RESULT_T;
CREATE TYPE PAL_PR_RESULT_T AS TABLE("ID" INT,"Ai" DOUBLE);
DROP TYPE PAL_PR_FITTED_T;
CREATE TYPE PAL_PR_FITTED_T AS TABLE("ID" INT,"Fitted" DOUBLE);
DROP TYPE PAL_PR_SIGNIFICANCE_T;
CREATE TYPE PAL_PR_SIGNIFICANCE_T AS TABLE("NAME" varchar(50),"VALUE" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
DROP TYPE PAL_PR_PMMODEL_T;
CREATE TYPE PAL_PR_PMMODEL_T AS TABLE("ID" INT,"Model" varchar(5000));
DROP table PAL_PR_PDATA_TBL;
CREATE column_table PAL_PR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
insert into PAL_PR_PDATA_TBL values (1,'DM_PAL','PAL_PR_DATA_T','IN');
insert into PAL_PR_PDATA_TBL values (2,'DM_PAL','PAL_CONTROL_T','IN');
insert into PAL_PR_PDATA_TBL values (3,'DM_PAL','PAL_PR_RESULT_T','OUT');
insert into PAL_PR_PDATA_TBL values (4,'DM_PAL','PAL_PR_FITTED_T','OUT');
insert into PAL_PR_PDATA_TBL values (5,'DM_PAL','PAL_PR_SIGNIFICANCE_T','OUT');
insert into PAL_PR_PDATA_TBL values (6,'DM_PAL','PAL_PR_PMMODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_POLYNOMIALR_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','POLYNOMIALREGRESSION','DM_PAL','PA
L_POLYNOMIALR_PROC','PAL_PR_PDATA_TBL');

DROP TABLE #PAL_CONTROL_TBL;

```

```

CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('POLYNOMIAL_NUM',3,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',8,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT',2,null,null);
DROP TABLE PAL_PR_DATA_TBL;
CREATE COLUMN TABLE PAL_PR_DATA_TBL ( "ID" INT, "Y" DOUBLE, "X1" DOUBLE);
INSERT INTO PAL_PR_DATA_TBL VALUES (0,5,1);
INSERT INTO PAL_PR_DATA_TBL VALUES (1,20,2);
INSERT INTO PAL_PR_DATA_TBL VALUES (2,43,3);
INSERT INTO PAL_PR_DATA_TBL VALUES (3,89,4);
INSERT INTO PAL_PR_DATA_TBL VALUES (4,166,5);
INSERT INTO PAL_PR_DATA_TBL VALUES (5,247,6);
INSERT INTO PAL_PR_DATA_TBL VALUES (6,403,7);
DROP TABLE PAL_PR_RESULTS_TBL;
CREATE COLUMN TABLE PAL_PR_RESULTS_TBL ("ID" INT, "Ai" DOUBLE);
DROP TABLE PAL_PR_FITTED_TBL;
CREATE COLUMN TABLE PAL_PR_FITTED_TBL ("ID" INT, "Fitted" DOUBLE);
DROP TABLE PAL_PR_SIGNIFICANCE_TBL;
CREATE COLUMN TABLE PAL_PR_SIGNIFICANCE_TBL ("NAME" varchar(50), "VALUE" DOUBLE);
DROP TABLE PAL_PR_MODEL_TBL;
CREATE COLUMN TABLE PAL_PR_MODEL_TBL ("ID" INT, "PMMLMODEL" VARCHAR(5000));
CALL DM.PAL.PAL_POLYNOMIALR_PROC(PAL_PR_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_PR_RESULTS_TBL, PAL_PR_FITTED_TBL, PAL_PR_SIGNIFICANCE_TBL,
PAL_PR_MODEL_TBL) with overview;
SELECT * FROM PAL_PR_RESULTS_TBL;
SELECT * FROM PAL_PR_FITTED_TBL;
SELECT * FROM PAL_PR_SIGNIFICANCE_TBL;
SELECT * FROM PAL_PR_MODEL_TBL;

```

Expected Result

PAL_PR_RESULTS_TBL:

ID	Ai
0	-10.99999999985448
1	17.24999999985448
2	-3.416666666642413
3	1.333333333333712

PAL_PR_FITTED_TBL:

ID	Fitted
0	4.16666666666913
1	20.49999999995453
2	45.9999999999375
3	88.6666666666424
4	156.5000000000716
5	257.5000000002274
6	399.666666667112

PAL_PR_SIGNIFICANCE_TBL:

NAME	VALUE
R2	0.9982396510011277
F	567.0691729324692

PAL_PR_MODEL_TBL:

ID	PMMLMODEL
1	<PMML version="4.0" xmlns="http://www.dmg.org/PMML-4_0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.dmg.org/PMML-4_0 PMML-4_0.xsd">

FORECASTWITHPOLYNOMIALR

This function performs prediction with the polynomial regression result.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'FORECASTWITHPOLYNOMIALR',  
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Coefficient INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <coefficient input table>, <parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Predictive Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Variable X
Coefficient	1st column	Integer, varchar, or nvarchar	ID (start from 0)
	2nd column	Integer, double, varchar, nvarchar, or CLOB	Value A_i or PMML model. Varchar, nvarchar, and CLOB types are only valid for PMML model.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads
MODEL_FORMAT	Integer	0	<ul style="list-style-type: none">• 0: Coefficients in table• 1: PMML format

Output Table

Table	Column	Column Data Type	Description
Fitted Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Value Y_i

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_FPR_PREDICT_T;
CREATE TYPE PAL_FPR_PREDICT_T AS TABLE( "ID" INT,"X1" DOUBLE);
DROP TYPE PAL_FPR_COEFFICIENT_T;
CREATE TYPE PAL_FPR_COEFFICIENT_T AS TABLE("ID" INT,"Ai" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("Name" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
DROP TYPE PAL_FPR_FITTED_T;
CREATE TYPE PAL_FPR_FITTED_T AS TABLE("ID" INT,"Fitted" DOUBLE);
DROP table PAL_FPR_PDATA_TBL;
CREATE column table PAL_FPR_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_FPR_PDATA_TBL values (1,'DM_PAL','PAL_FPR_PREDICT_T','IN');
INSERT INTO PAL_FPR_PDATA_TBL values (2,'DM_PAL','PAL_FPR_COEFFICIENT_T','IN');
INSERT INTO PAL_FPR_PDATA_TBL values (3,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_FPR_PDATA_TBL values (4,'DM_PAL','PAL_FPR_FITTED_T','OUT');

CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_FORECAST_POLYNOMIALR_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','FORECASTWITHPOLYNOMIALR','DM_PAL',
'PAL_FORECAST_POLYNOMIALR_PROC',PAL_FPR_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
```

```

CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("Name" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 8, null, null);
DROP TABLE PAL_FPR_PREDICTDATA_TBL;
CREATE COLUMN TABLE PAL_FPR_PREDICTDATA_TBL ( "ID" INT, "X1" DOUBLE);
INSERT INTO PAL_FPR_PREDICTDATA_TBL VALUES (0, 0.3);
INSERT INTO PAL_FPR_PREDICTDATA_TBL VALUES (1, 4.0);
INSERT INTO PAL_FPR_PREDICTDATA_TBL VALUES (2, 1.6);
INSERT INTO PAL_FPR_PREDICTDATA_TBL VALUES (3, 0.45);
INSERT INTO PAL_FPR_PREDICTDATA_TBL VALUES (4, 1.7);
DROP TABLE PAL_FPR_COEEFICIENT_TBL;
CREATE COLUMN TABLE PAL_FPR_COEEFICIENT_TBL ("ID" INT, "Ai" DOUBLE);
INSERT INTO PAL_FPR_COEEFICIENT_TBL VALUES (0, 4.0);
INSERT INTO PAL_FPR_COEEFICIENT_TBL VALUES (1, 3.0);
INSERT INTO PAL_FPR_COEEFICIENT_TBL VALUES (2, 2.0);
INSERT INTO PAL_FPR_COEEFICIENT_TBL VALUES (3, 1.0);
DROP TABLE PAL_FPR_FITTED_TBL;
CREATE COLUMN TABLE PAL_FPR_FITTED_TBL ("ID" INT, "Fitted" DOUBLE);
CALL DM_PAL.PAL_FORECAST_POLYNOMIALR_PROC(PAL_FPR_PREDICTDATA_TBL,
PAL_FPR_COEEFICIENT_TBL, "#PAL_CONTROL_TBL", PAL_FPR_FITTED_TBL) with overview;
SELECT * FROM PAL_FPR_FITTED_TBL;

```

Expected Result

PAL_FPR_FITTED_TBL:

ID	Fitted
0	5.107
1	112.0
2	18.016000...
3	5.846125
4	19.793

3.4 Association Algorithms

This section describes the association algorithms that are provided by the Predictive Analysis Library.

3.4.1 Apriori

Apriori is a classic predictive analysis algorithm for finding association rules used in association analysis. Association analysis uncovers the hidden patterns, correlations or causal structures among a set of items or objects. For example, association analysis enables you to understand what products and services customers tend to purchase at the same time. By analyzing the purchasing trends of your customers with association analysis, you can predict their future behavior.

Apriori is designed to operate on databases containing transactions. As is common in association rule mining, given a set of items, the algorithm attempts to find subsets which are common to at least a minimum number of the item sets. Apriori uses a “bottom up” approach, where frequent subsets are extended one item at a time, a step known as candidate generation, and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found. Apriori uses breadth-first search and a tree

structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length $k-1$, and then prunes the candidates which have an infrequent sub pattern. The candidate set contains all frequent k -length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

The Apriori function in PAL uses vertical data format to store the transaction data in memory. The function can take varchar/nvarchar or integer transaction ID and item ID as input. It supports the output of confidence, support, and lift value, but does not limit the number of output rules. However, you can use SQL script to select the number of output rules, for example:

```
SELECT TOP 2000 FROM RULE_RESULTS where lift > 0.5
```

Prerequisites

- The input data does not contain null value.
- There are no duplicated items in each transaction.

APRIORIRULE

This function reads input transaction data and generates association rules by the Apriori algorithm.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'APRIORIRULE',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>, <PMML output table>) WITH overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Dataset/ Historical Data	1st column	Integer, varchar, or nvarchar	Transaction ID
	2nd column	Integer, varchar, or nvarchar	Item ID

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description
MIN_SUPPORT	Double	User-specified minimum support (actual value).
MIN_CONFIDENCE	Double	User-specified minimum confidence (actual value).

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
MIN_LIFT	Double	0.0	User-specified minimum lift.
MAX_CONSEQUENT	Integer	100	Maximum length of dependent items.
MAXITEMLENGTH	Integer	5	Total length of leading items and dependent items in the output.
UBIQUITOUS	Double	1.0	Ignores items whose support values are greater than the UBIQUITOUS value during the frequent items mining phase.

Name	Data Type	Default Value	Description
IS_USE_PREFIX_TREE	Integer	0	<p>Indicates whether to use the prefix tree, which can save memory.</p> <ul style="list-style-type: none"> • 0: Does not use the prefix tree. • 1: Uses the prefix tree.
LHS_RESTRICT	Varchar	No default value	Specifies that some items are only allowed on the left-hand side of the association rules.
RHS_RESTRICT	Varchar	No default value	Specifies that some items are only allowed on the right-hand side of the association rules.

Name	Data Type	Default Value	Description
LHS_IS_COMPLEMENTARY_RHS	Integer	0	<p>If you use RHS_RESTRICT to restrict some items to the right-hand side of the association rules, you can set this parameter to 1 to restrict the consequent items to the left-hand side.</p> <p>For example, if you have 1000 items (i1, i2, ..., i1000) and want to restrict i1 and i2 to the right-hand side, and i3, i4, ..., i1000 to the left-hand side, you can set the parameters similar to the following:</p> <pre>INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i1'); INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i2'); INSERT INTO PAL_CONTROL_TBL VALUES ('LHS_IS_COMPLEMENTARY_RHS', 1, NULL, NULL);</pre>
RHS_IS_COMPLEMENTARY_LHS	Integer	0	If you use LHS_RESTRICT to restrict some items to the left-hand side of the association rules, you can set this parameter to 1 to restrict the consequent items to the right-hand side.
THREAD_NUMBER	Integer	1	Number of threads.

Name	Data Type	Default Value	Description
TIMEOUT	Integer	3600	Specifies the maximum run time in seconds. The algorithm will stop running when the specified timeout is reached.
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export Apriori model in PMML. • 1: Exports Apriori model in PMML in single row. • 2: Exports Apriori model in PMML in several rows, and the minimum length of each row is 5000 characters.

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Leading items
	2nd column	Varchar or nvarchar	Dependent items
	3rd column	Double	Support value
	4th column	Double	Confidence value
	5th column	Double	Lift value
PMML Result	1st column	Integer	ID
	2nd column	CLOB, varchar, or nvarchar	Apriori model in PMML format

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_APRIORI_DATA_T;
CREATE TYPE PAL_APRIORI_DATA_T AS TABLE(
    "CUSTOMER" INTEGER,
    "ITEM" VARCHAR(20)
);
```

```

DROP TYPE PAL_APRIORI_RESULT_T;
CREATE TYPE PAL_APRIORI_RESULT_T AS TABLE(
    "PRERULE" VARCHAR(500),
    "POSTRULE" VARCHAR(500),
    "SUPPORT" DOUBLE,
    "CONFIDENCE" DOUBLE,
    "LIFT" DOUBLE
);
DROP TYPE PAL_APRIORI_PMMILMODEL_T;
CREATE TYPE PAL_APRIORI_PMMILMODEL_T AS TABLE(
    "ID" INTEGER,
    "PMMLMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_APRIORI_PDATA_TBL;
CREATE COLUMN TABLE PAL_APRIORI_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_APRIORI_DATA_T', 'IN');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_APRIORI_RESULT_T', 'OUT');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_APRIORI_PMMILMODEL_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_APRIORI_RULE_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'APRIORIRULE', 'DM_PAL', 'PAL_APRIORI_RULE_PROC', PAL_APRIORI_PDATA_TBL);
DROP TABLE PAL_APRIORI_TRANS_TBL;
CREATE COLUMN TABLE PAL_APRIORI_TRANS_TBL LIKE PAL_APRIORI_DATA_T;
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (2, 'item2');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (2, 'item3');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (3, 'item1');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (3, 'item2');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (3, 'item4');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (4, 'item1');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (4, 'item3');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (5, 'item2');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (5, 'item3');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (6, 'item1');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (6, 'item3');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (0, 'item1');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (0, 'item2');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (0, 'item5');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (1, 'item2');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (1, 'item4');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (7, 'item1');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (7, 'item2');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (7, 'item3');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (7, 'item5');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (8, 'item1');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (8, 'item2');
INSERT INTO PAL_APRIORI_TRANS_TBL VALUES (8, 'item3');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);

```

```

);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_SUPPORT', null, 0.1, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_CONFIDENCE', null, 0.3, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_LIFT', null, 1.1, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_CONSEQUENT', 1, null, null);
DROP TABLE PAL_APRIORI_RESULT_TBL;
CREATE COLUMN TABLE PAL_APRIORI_RESULT_TBL LIKE PAL_APRIORI_RESULT_T;
DROP TABLE PAL_APRIORI_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_APRIORI_PMMODEL_TBL LIKE PAL_APRIORI_PMMODEL_T;
CALL "DM_PAL".PAL_APRIORI_RULE_PROC(PAL_APRIORI_TRANS_TBL, #PAL_CONTROL_TBL,
PAL_APRIORI_RESULT_TBL, PAL_APRIORI_PMMODEL_TBL) WITH overview;
SELECT * FROM PAL_APRIORI_RESULT_TBL;
SELECT * FROM PAL_APRIORI_PMMODEL_TBL;

```

Expected Result:

PAL_APRIORI_RESULT_TBL:

PRERULE	POSTRULE	SUPPORT	CONFIDE...	LIFT
item5	item2	0.2222222222...	1	1.285714285...
item5	item1	0.2222222222...	1	1.5
item1	item5	0.2222222222...	0.3333333...	1.5
item4	item2	0.2222222222...	1	1.285714285...
item2&item1	item5	0.2222222222...	0.5	2.25
item5&item1	item2	0.2222222222...	1	1.285714285...
item5&item2	item1	0.2222222222...	1	1.5
item5&item3	item2	0.1111111111...	1	1.285714285...
item5&item3	item1	0.1111111111...	1	1.5
item1&item4	item2	0.1111111111...	1	1.285714285...
item2&item1&item3	item5	0.1111111111...	0.5	2.25
item5&item1&item3	item2	0.1111111111...	1	1.285714285...
item5&item2&item3	item1	0.1111111111...	1	1.5

APRIORIRULE2

This function has the same logic with APRIORIRULE, but it splits the result table into three tables.

Procedure Generation

```

CALL SYS.AFLLANG_WRAPPER PROCEDURE CREATE ('AFLPAL', 'APRIORIRULE2',
'<schema_name>', '<procedure_name>', <signature_table>);

```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN

Position	Schema Name	Table Type Name	Parameter Type
3	<schema_name>	<Statistical OUTPUT table type>	OUT
4	<schema_name>	<Antecedent OUTPUT table type>	OUT
5	<schema_name>	<Consequent OUTPUT table type>	OUT
6	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>,
<statistical output table>, <antecedent output table>, <consequent output
table>, <PMML output table>) WITH overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, varchar, or nvarchar	Transaction ID
	2nd column	Integer, varchar, or nvarchar	Item ID

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description
MIN_SUPPORT	Double	User-specified minimum support (actual value).
MIN_CONFIDENCE	Double	User-specified minimum confidence (actual value).

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
MIN_LIFT	Double	0.0	User-specified minimum lift.
MAX_CONSEQUENT	Integer	100	Maximum length of dependent items.
MAXITEMLENGTH	Integer	5	Total length of leading items and dependent items in the output.
UBIQUITOUS	Double	1.0	Ignores items whose support values are greater than the UBIQUITOUS value during the frequent items mining phase.
IS_USE_PREFIX_TREE	Integer	0	Indicates whether to use the prefix tree, which can save memory. <ul style="list-style-type: none"> • 0: Does not use the prefix tree. • 1: Uses the prefix tree.
LHS_RESTRICT	Varchar	No default value	Specifies that some items are only allowed on the left-hand side of the association rules.
RHS_RESTRICT	Varchar	No default value	Specifies that some items are only allowed on the right-hand side of the association rules.

Name	Data Type	Default Value	Description
LHS_IS_COMPLEMENTARY_RHS	Integer	0	<p>If you use RHS_RESTRICT to restrict some items to the right-hand side of the association rules, you can set this parameter to 1 to restrict the consequent items to the left-hand side.</p> <p>For example, if you have 1000 items (i1, i2, ..., i1000) and want to restrict i1 and i2 to the right-hand side, and i3, i4, ..., i1000 to the left-hand side, you can set the parameters similar to the following:</p> <pre>INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i1'); INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i2'); INSERT INTO PAL_CONTROL_TBL VALUES ('LHS_IS_COMPLEMENTARY_RHS', 1, NULL, NULL);</pre>
RHS_IS_COMPLEMENTARY_LHS	Integer	0	If you use LHS_RESTRICT to restrict some items to the left-hand side of the association rules, you can set this parameter to 1 to restrict the consequent items to the right-hand side.
THREAD_NUMBER	Integer	1	Number of threads.

Name	Data Type	Default Value	Description
TIMEOUT	Integer	3600	Specifies the maximum run time in seconds. The algorithm will stop running when the specified timeout is reached.
PMML_EXPORT	Double	0	<ul style="list-style-type: none"> • 0: Does not export the PMML model. • 1: Exports Apriori model in PMML in single row. • 2: Exports Apriori model in PMML in several rows, and the minimum length of each row is 5000 characters.

Output Tables

Table	Column	Column Data Type	Description
Statistical	1st column	Integer	ID
	2nd column	Double	Support
	3rd column	Double	Confidence
	4th column	Double	Lift
Antecedent	1st column	Integer	ID
	2nd column	Varchar, nvarchar, or integer	Antecedent item
Consequent	1st column	Integer	ID
	2nd column	Varchar, nvarchar, or integer	Consequent item
PMML Result	1st column	Integer	ID
	2nd column	Varchar or nvarchar	PMML model

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and

- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_APRIORI_DATA_T;
CREATE TYPE PAL_APRIORI_DATA_T AS TABLE(
    "CUSTOMER" INTEGER,
    "ITEM" VARCHAR(20)
);
DROP TYPE PAL_APRIORI_RULES_T;
CREATE TYPE PAL_APRIORI_RULES_T AS TABLE(
    "ID" INTEGER,
    "SUPPORT" DOUBLE,
    "CONFIDENCE" DOUBLE,
    "LIFT" DOUBLE
);
DROP TYPE PAL_APRIORI_ANTE_ITEMS_T;
CREATE TYPE PAL_APRIORI_ANTE_ITEMS_T AS TABLE(
    "ID" INTEGER,
    "ANTECEDENT" VARCHAR(20)
);
DROP TYPE PAL_APRIORI_CONS_ITEMS_T;
CREATE TYPE PAL_APRIORI_CONS_ITEMS_T AS TABLE(
    "ID" INTEGER,
    "CONSEQUENT" VARCHAR(20)
);
DROP TYPE PAL_APRIORI_PMMODEL_T;
CREATE TYPE PAL_APRIORI_PMMODEL_T AS TABLE(
    "ID" INTEGER,
    "PMMLMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);

DROP TABLE PAL_APRIORI_PDATA_TBL;
CREATE COLUMN TABLE PAL_APRIORI_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_APRIORI_DATA_T', 'IN');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_APRIORI_RULES_T', 'OUT');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_APRIORI_ANTE_ITEMS_T', 'OUT');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_APRIORI_CONS_ITEMS_T', 'OUT');
INSERT INTO PAL_APRIORI_PDATA_TBL VALUES (6, 'DM_PAL', 'PAL_APRIORI_PMMODEL_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_APRIORI2_PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'APRIORIRULE2', 'DM_PAL',
'PAL_APRIORI2_PROC', PAL_APRIORI_PDATA_TBL);
DROP TABLE PAL_APRIORI_DATA_TBL;
CREATE COLUMN TABLE PAL_APRIORI_DATA_TBL LIKE PAL_APRIORI_DATA_T;
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (2, 'item2');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (2, 'item3');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (3, 'item1');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (3, 'item2');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (3, 'item4');

```

```

INSERT INTO PAL_APRIORI_DATA_TBL VALUES (4, 'item1');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (4, 'item3');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (5, 'item2');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (5, 'item3');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (6, 'item1');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (6, 'item3');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (0, 'item1');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (0, 'item2');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (0, 'item5');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (1, 'item2');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (1, 'item4');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (7, 'item1');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (7, 'item2');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (7, 'item3');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (7, 'item5');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (8, 'item1');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (8, 'item2');
INSERT INTO PAL_APRIORI_DATA_TBL VALUES (8, 'item3');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_SUPPORT', null, 0.1, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_CONFIDENCE', null, 0.6, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_LIFT', null, 1.2, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_CONSEQUENT', 1, null, null);
DROP TABLE PAL_APRIORI_RULES_TBL;
CREATE COLUMN TABLE PAL_APRIORI_RULES_TBL LIKE PAL_APRIORI_RULES_T;
DROP TABLE PAL_APRIORI_ANTE_ITEMS_TBL;
CREATE COLUMN TABLE PAL_APRIORI_ANTE_ITEMS_TBL LIKE PAL_APRIORI_ANTE_ITEMS_T;
DROP TABLE PAL_APRIORI_CONS_ITEMS_TBL;
CREATE COLUMN TABLE PAL_APRIORI_CONS_ITEMS_TBL LIKE PAL_APRIORI_CONS_ITEMS_T;

DROP TABLE PAL_APRIORI_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_APRIORI_PMMODEL_TBL LIKE PAL_APRIORI_PMMODEL_T;
CALL "DM_PAL".PAL_APRIORI2_PROC(PAL_APRIORI_DATA_TBL, #PAL_CONTROL_TBL,
PAL_APRIORI_RULES_TBL, PAL_APRIORI_ANTE_ITEMS_TBL, PAL_APRIORI_CONS_ITEMS_TBL,
PAL_APRIORI_PMMODEL_TBL) with OVERVIEW;
SELECT * FROM PAL_APRIORI_RULES_TBL;
SELECT * FROM PAL_APRIORI_ANTE_ITEMS_TBL;
SELECT * FROM PAL_APRIORI_CONS_ITEMS_TBL;
SELECT * FROM PAL_APRIORI_PMMODEL_TBL;

```

Expected Result:

PAL_APRIORI_RULES_TBL:

ID	SUPPORT	CONFIDENCE	LIFT
0	0.2222222222222222	1	1.2857142857142856
1	0.2222222222222222	1	1.5
2	0.2222222222222222	1	1.2857142857142856
3	0.2222222222222222	1	1.2857142857142856
4	0.2222222222222222	1	1.5
5	0.1111111111111111	1	1.2857142857142856
6	0.1111111111111111	1	1.5
7	0.1111111111111111	1	1.2857142857142856
8	0.1111111111111111	1	1.2857142857142856
9	0.1111111111111111	1	1.5

PAL_APRIORI_ANTE_ITEMS_TBL:

ID	ANTECEDENT
0	item5
1	item5
2	item4
3	item5
3	item1
4	item5
4	item2
5	item5
5	item3
6	item5
6	item3
7	item1
7	item4
8	item5
8	item1
8	item3
9	item5
9	item2
9	item3

PAL_APRIORI_CONS_ITEMS_TBL:

ID	CONSEQUENT
0	item2
1	item1
2	item2
3	item2
4	item1
5	item2
6	item1
7	item2
8	item2
9	item1

LITEAPRIORIRULE

This is a light association rule mining algorithm to realize the Apriori algorithm. It only calculates two large item sets.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LITEAPRIORIRULE',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<PMML OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>, <PMML output table>) WITH overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Dataset/ Historical Data	1st column	Integer, varchar, or nvarchar	Transaction ID
	2nd column	Integer, varchar, or nvarchar	Item ID

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description
MIN_SUPPORT	Double	User-specified minimum support (actual value).
MIN_CONFIDENCE	Double	User-specified minimum confidence (actual value).

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads.
OPTIMIZATION_TYPE	Integer or double	0	If you want to use the entire data, set the integer value to 0. If you want to sample the source input data, specify a double value as the sampling percentage.
IS_RECALCULATE	Integer	1	If you sample the input data, this parameter controls whether to use the remaining data or not. <ul style="list-style-type: none"> • 1: Uses the remaining data to update the support, confidence, and lift. • 0: Does not use the remaining data

Name	Data Type	Default Value	Description
TIMEOUT	Integer	3600	Specifies the maximum run time in seconds. The algorithm will stop running when the specified timeout is reached.
PMML_EXPORT	Integer	0	<ul style="list-style-type: none"> • 0: Does not export liteApriori model in PMML. • 1: Exports liteApriori model in PMML in single row. • 2: Exports liteApriori model in PMML in several rows, and the minimum length of each row is 5000 characters.

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Leading items
	2nd column	Varchar or nvarchar	Dependent items
	3rd column	Double	Support value
	4th column	Double	Confidence value
	5th column	Double	Lift value
PMML Result	1st column	Integer	ID
	2nd column	CLOB, varchar, or nvarchar	liteApriori model in PMML format

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_LITEAPRIORI_DATA_T;
```

```

CREATE TYPE PAL_LITEAPRIORI_DATA_T AS TABLE(
    "CUSTOMER" INTEGER,
    "ITEM" VARCHAR(20)
);
DROP TYPE PAL_LITEAPRIORI_RESULT_T;
CREATE TYPE PAL_LITEAPRIORI_RESULT_T AS TABLE(
    "PRERULE" VARCHAR(500),
    "POSTRULE" VARCHAR(500),
    "SUPPORT" DOUBLE,
    "CONFIDENCE" DOUBLE,
    "LIFT" DOUBLE
);
DROP TYPE PAL_LITEAPRIORI_PMMODEL_T;
CREATE TYPE PAL_LITEAPRIORI_PMMODEL_T AS TABLE(
    "ID" INTEGER,
    "PMMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_LITEAPRIORI_PDATA_TBL;
CREATE COLUMN_TABLE PAL_LITEAPRIORI_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_LITEAPRIORI_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_LITEAPRIORI_DATA_T', 'IN');
INSERT INTO PAL_LITEAPRIORI_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T',
'IN');
INSERT INTO PAL_LITEAPRIORI_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_LITEAPRIORI_RESULT_T', 'OUT');
INSERT INTO PAL_LITEAPRIORI_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_LITEAPRIORI_PMMODEL_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL',
'PAL_LITE_APRIORI_RULE_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'LITEAPRIORIRULE',
'DM_PAL', 'PAL_LITE_APRIORI_RULE_PROC', PAL_LITEAPRIORI_PDATA_TBL);
DROP TABLE PAL_LITEAPRIORI_DATA_TBL;
CREATE COLUMN_TABLE PAL_LITEAPRIORI_DATA_TBL LIKE PAL_LITEAPRIORI_DATA_T;
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (2, 'item2');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (2, 'item3');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (3, 'item1');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (3, 'item2');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (3, 'item4');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (4, 'item1');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (4, 'item3');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (5, 'item2');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (5, 'item3');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (6, 'item1');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (6, 'item3');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (0, 'item1');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (0, 'item2');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (0, 'item5');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (1, 'item2');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (1, 'item4');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (7, 'item1');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (7, 'item2');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (7, 'item3');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (7, 'item5');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (8, 'item1');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (8, 'item2');
INSERT INTO PAL_LITEAPRIORI_DATA_TBL VALUES (8, 'item3');

```

```

DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_SUPPORT', null, 0.3, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_CONFIDENCE', null, 0.4, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('OPTIMIZATION_TYPE', 0, 0.7, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('IS_RECALCULATE', 1, null, null);
DROP TABLE PAL_LITEAPRIORI_RESULT_TBL;
CREATE COLUMN TABLE PAL_LITEAPRIORI_RESULT_TBL LIKE PAL_LITEAPRIORI_RESULT_T;
DROP TABLE PAL_LITEAPRIORI_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_LITEAPRIORI_PMMODEL_TBL LIKE
PAL_LITEAPRIORI_PMMODEL_T;
CALL "DM_PAL".PAL_LITE_APRIORI_RULE_PROC(PAL_LITEAPRIORI_DATA_TBL,
#PAL_CONTROL_TBL, PAL_LITEAPRIORI_RESULT_TBL, PAL_LITEAPRIORI_PMMODEL_TBL)
WITH OVERVIEW;
SELECT * FROM PAL_LITEAPRIORI_RESULT_TBL;
SELECT * FROM PAL_LITEAPRIORI_PMMODEL_TBL;

```

Expected Result

PAL_LITEAPRIORI_RESULT_TBL:

PRERULE	POSTRULE	SUPPORT	CONFIDENCE	LIFT
item2	item1	0.444444444...	0.5714285714285714	0.85714285714...
item1	item2	0.444444444...	0.6666666666666666	0.85714285714...
item2	item3	0.444444444...	0.5714285714285714	0.85714285714...
item3	item2	0.444444444...	0.6666666666666666	0.85714285714...
item1	item3	0.444444444...	0.6666666666666666	1
item3	item1	0.444444444...	0.6666666666666666	1

3.4.2 FP-Growth

FP-Growth is an algorithm to find frequent patterns from transactions without generating a candidate itemset.

In PAL, the FP-Growth algorithm is extended to find association rules in three steps:

1. Converts the transactions into a compressed frequent pattern tree (FP-Tree);
2. Recursively finds frequent patterns from the FP-Tree;
3. Generates association rules based on the frequent patterns found in Step 2.

FP-Growth with relational output is also supported.

Prerequisites

- The input data does not contain null value.
- There are no duplicated items in each transaction.

FPGROWTH

This function reads input transaction data and generates association rules by the FP-Growth algorithm.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'FPGROWTH',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Dataset / Historical Data	1st column	Integer, varchar, or nvarchar	Transaction ID
	2nd column	Integer, varchar, or nvarchar	Item ID

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
MIN_SUPPORT	Double	0	Minimum support. Value range: between 0 and 1
MIN_CONFIDENCE	Double	0	Minimum confidence. Value range: between 0 and 1
MIN_LIFT	Double	0	Minimum lift.
MAXITEMLENGTH	Integer	10	Maximum length of leading items and dependent items in the output.
MAX_CONSEQUENT	Integer	10	Maximum length of right-hand side.
UBIQUITOUS	Double	1.0	Ignores items whose support values are greater than the UBIQUITOUS value during the frequent items mining phase.
LHS_RESTRICT	Varchar	No default value	Specifies which items are allowed in the left-hand side of the association rule.
RHS_RESTRICT	Varchar	No default value	Specifies which items are allowed in the right-hand side of the association rule.

Name	Data Type	Default Value	Description
LHS_IS_COMPLEMENTARY_RHS	Integer	0	<p>If you use RHS_RESTRICT to restrict some items to the right-hand side of the association rules, you can set this parameter to 1 to restrict the consequent items to the left-hand side.</p> <p>For example, if you have 1000 items (i1, i2, ..., i1000) and want to restrict i1 and i2 to the right-hand side, and i3, i4, ..., i1000 to the left-hand side, you can set the parameters similar to the following:</p> <pre>INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i1'); INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i2'); INSERT INTO PAL_CONTROL_TBL VALUES ('LHS_IS_COMPLEMENTARY_RHS', 1, NULL, NULL);</pre>
RHS_IS_COMPLEMENTARY_LHS	Integer	0	If you use LHS_RESTRICT to restrict some items to the left-hand side of the association rules, you can set this parameter to 1 to restrict the consequent items to the right-hand side.
THREAD_NUMBER	Integer	1	Number of threads.

Name	Data Type	Default Value	Description
TIMEOUT	Integer	3600	The function will automatically terminate if its running time is longer than the TIMEOUT value (unit: second).

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Right-hand side items
	2nd column	Varchar or nvarchar	Left-hand side items
	3rd column	Double	Support value
	4th column	Double	Confidence value
	5th column	Double	Lift value

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_FPGROWTH_DATA_T;
CREATE TYPE PAL_FPGROWTH_DATA_T AS TABLE(
    "TRANS" INTEGER,
    "ITEM" INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_FPGROWTH_RESULT_T;
CREATE TYPE PAL_FPGROWTH_RESULT_T AS TABLE(
    "PRERULE" VARCHAR(500),
    "POSTRULE" VARCHAR(500),
    "SUPPORT" DOUBLE,
    "CONFIDENCE" DOUBLE,
    "LIFT" DOUBLE
);
DROP TABLE PAL_FPGROWTH_PDATA_TBL;
CREATE COLUMN TABLE PAL_FPGROWTH_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    ...
);
  
```

```

    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_FPGROWTH_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_FPGROWTH_DATA_T',
'in');
INSERT INTO PAL_FPGROWTH_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'in');
INSERT INTO PAL_FPGROWTH_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_FPGROWTH_RESULT_T',
'out');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_FPGROWTH_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'FPGROWTH', 'DM_PAL',
'PAL_FPGROWTH_PROC', 'PAL_FPGROWTH_PDATA_TBL');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_SUPPORT', null, 0.2, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_CONFIDENCE', null, 0.5, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_LIFT', null, 1, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAXITEMLENGTH', 5, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_CONSEQUENT', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LHS_RESTRICT', null, null, '1');
INSERT INTO #PAL_CONTROL_TBL VALUES ('LHS_RESTRICT', null, null, '2');
INSERT INTO #PAL_CONTROL_TBL VALUES ('LHS_RESTRICT', null, null, '3');
INSERT INTO #PAL_CONTROL_TBL VALUES ('TIMEOUT', 60, null, null);
DROP TABLE PAL_FPGROWTH_RESULT_TBL;
CREATE COLUMN TABLE PAL_FPGROWTH_RESULT_TBL LIKE PAL_FPGROWTH_RESULT_T;
DROP TABLE PAL_FPGROWTH_DATA_TBL;
CREATE COLUMN TABLE PAL_FPGROWTH_DATA_TBL LIKE PAL_FPGROWTH_DATA_T;
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (1, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (1, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (2, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (2, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (2, 4);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (3, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (3, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (3, 4);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (3, 5);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (4, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (4, 4);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (4, 5);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (5, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (5, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (6, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (6, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (6, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (6, 4);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (7, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (8, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (8, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (8, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (9, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (9, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (9, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (10, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (10, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (10, 5);
DELETE FROM PAL_FPGROWTH_RESULT_TBL;
CALL DM_PAL.PAL_FPGROWTH_PROC(PAL_FPGROWTH_DATA_TBL, #PAL_CONTROL_TBL,
PAL_FPGROWTH_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_FPGROWTH_RESULT_TBL ORDER BY PRERULE, POSTRULE, SUPPORT,
CONFIDENCE, LIFT;

```

Expected Result

	PRERULE	POSTRULE	SUPPORT	CONFIDENCE	LIFT
1	1&2	3	0.3	0.6	1
2	1&3	2	0.3	0.75	1.0714285714285714
3	1&3	4	0.2	0.5	1.25
4	2	3	0.5	0.714285714...	1.1904761904761905
5	3	2	0.5	0.833333333...	1.1904761904761905
6	3	4	0.3	0.5	1.25

FPGROWTH (Relational Output)

FP-Growth with relational output uses the same algorithm as FP-Growth. The only difference is the output format. The relational output version of FP-Growth separates the result into three tables.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'FPGROWTH',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<PRE-RULE OUTPUT table type>	OUT
4	<schema_name>	<POST-RULE OUTPUT table type>	OUT
5	<schema_name>	<STATISTICS OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <pre-rule output table>, <post-rule output table>, <statistics output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Dataset / Historical Data	1st column	Integer, varchar, or nvarchar	Transaction ID
	2nd column	Integer, varchar, or nvarchar	Item ID

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
MIN_SUPPORT	Double	0	Minimum support. Value range: between 0 and 1
MIN_CONFIDENCE	Double	0	Minimum confidence. Value range: between 0 and 1
MIN_LIFT	Double	0	Minimum lift.
MAXITEMLENGTH	Integer	10	Maximum length of leading items and dependent items in the output.
MAX_CONSEQUENT	Integer	10	Maximum length of right-hand side.
UBIQUITOUS	Double	1.0	Ignores items whose support values are greater than the UBIQUITOUS value during the frequent items mining phase.
LHS_RESTRICT	Varchar	No default value	Specifies which items are allowed in the left-hand side of the association rule.
RHS_RESTRICT	Varchar	No default value	Specifies which items are allowed in the right-hand side of the association rule.

Name	Data Type	Default Value	Description
LHS_IS_COMPLEMENTARY_RHS	Integer	0	<p>If you use RHS_RESTRICT to restrict some items to the right-hand side of the association rules, you can set this parameter to 1 to restrict the consequent items to the left-hand side.</p> <p>For example, if you have 1000 items (i1, i2, ..., i1000) and want to restrict i1 and i2 to the right-hand side, and i3, i4, ..., i1000 to the left-hand side, you can set the parameters similar to the following:</p> <pre>INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i1'); INSERT INTO PAL_CONTROL_TBL VALUES ('RHS_RESTRICT', NULL, NULL, 'i2'); INSERT INTO PAL_CONTROL_TBL VALUES ('LHS_IS_COMPLEMENTARY_RHS', 1, NULL, NULL);</pre>
RHS_IS_COMPLEMENTARY_LHS	Integer	0	If you use LHS_RESTRICT to restrict some items to the left-hand side of the association rules, you can set this parameter to 1 to restrict the consequent items to the right-hand side.
THREAD_NUMBER	Integer	1	Number of threads.

Name	Data Type	Default Value	Description
TIMEOUT	Integer	3600	The function will automatically terminate if its running time is longer than the TIMEOUT value (unit: second).

Output Tables

Table	Column	Column Data Type	Description
Prerule Result	1st column	Integer	Rule ID
	2nd column	Varchar or nvarchar	Item
Postrule Result	1st column	Integer	Rule ID
	2nd column	Varchar or nvarchar	Item
Statistics Result	1st column	Integer	Rule ID
	2nd column	Double	Support
	3rd column	Double	Confidence
	4th column	Double	Lift

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_FPGROWTH_DATA_T;
CREATE TYPE PAL_FPGROWTH_DATA_T AS TABLE(
    "TRANS" INTEGER,
    "ITEM" INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_FPGROWTH_PRERULE_T;
CREATE TYPE PAL_FPGROWTH_PRERULE_T AS TABLE(
    "ID" INTEGER,
    "PRERULE" INTEGER
);

```

```

DROP TYPE PAL_FPGROWTH_POSTRULE_T;
CREATE TYPE PAL_FPGROWTH_POSTRULE_T AS TABLE(
    "ID" INTEGER,
    "POSTRULE" INTEGER
);
DROP TYPE PAL_FPGROWTH_VALUES_T;
CREATE TYPE PAL_FPGROWTH_VALUES_T AS TABLE(
    "ID" INTEGER,
    "SUPPORT" DOUBLE,
    "CONFIDENCE" DOUBLE,
    "LIFT" DOUBLE
);
DROP TABLE PAL_FPGROWTH_PDATA_TBL;
CREATE COLUMN TABLE PAL_FPGROWTH_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_FPGROWTH_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_FPGROWTH_DATA_T',
'in');
INSERT INTO PAL_FPGROWTH_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'in');
INSERT INTO PAL_FPGROWTH_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_FPGROWTH_PRERULE_T', 'out');
INSERT INTO PAL_FPGROWTH_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_FPGROWTH_POSTRULE_T', 'out');
INSERT INTO PAL_FPGROWTH_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_FPGROWTH_VALUES_T',
'out');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_FPGROWTH_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'FPGROWTH', 'DM_PAL',
'PAL_FPGROWTH_PROC', 'PAL_FPGROWTH_PDATA_TBL');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_SUPPORT', null, 0.2, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_CONFIDENCE', null, 0.5, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_LIFT', null, 1, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAXITEMLENGTH', 5, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_CONSEQUENT', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LHS_RESTRICT', null, null, '1');
INSERT INTO #PAL_CONTROL_TBL VALUES ('LHS_RESTRICT', null, null, '2');
INSERT INTO #PAL_CONTROL_TBL VALUES ('LHS_RESTRICT', null, null, '3');
INSERT INTO #PAL_CONTROL_TBL VALUES ('TIMEOUT', 60, null, null);
DROP TABLE PAL_FPGROWTH_DATA_TBL;
CREATE COLUMN TABLE PAL_FPGROWTH_DATA_TBL LIKE PAL_FPGROWTH_DATA_T;
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (1, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (1, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (2, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (2, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (2, 4);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (3, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (3, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (3, 4);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (3, 5);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (4, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (4, 4);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (4, 5);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (5, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (5, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (6, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (6, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (6, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (6, 4);

```

```

INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (7, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (8, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (8, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (8, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (9, 1);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (9, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (9, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (10, 2);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (10, 3);
INSERT INTO PAL_FPGROWTH_DATA_TBL VALUES (10, 5);
DROP TABLE PAL_FPGROWTH_PRERULE_TBL;
CREATE COLUMN TABLE PAL_FPGROWTH_PRERULE_TBL LIKE PAL_FPGROWTH_PRERULE_T;
DROP TABLE PAL_FPGROWTH_POSTRULE_TBL;
CREATE COLUMN TABLE PAL_FPGROWTH_POSTRULE_TBL LIKE PAL_FPGROWTH_POSTRULE_T;
DROP TABLE PAL_FPGROWTH_VALUES_TBL;
CREATE COLUMN TABLE PAL_FPGROWTH_VALUES_TBL LIKE PAL_FPGROWTH_VALUES_T;
CALL DM_PAL.PAL_FPGROWTH PROC(PAL_FPGROWTH_DATA_TBL, #PAL_CONTROL_TBL,
PAL_FPGROWTH_PRERULE_TBL, PAL_FPGROWTH_POSTRULE_TBL, PAL_FPGROWTH_VALUES_TBL)
WITH OVERVIEW;
SELECT * FROM PAL_FPGROWTH_PRERULE_TBL;
SELECT * FROM PAL_FPGROWTH_POSTRULE_TBL;
SELECT * FROM PAL_FPGROWTH_VALUES_TBL;

```

Expected Results

PAL_FPGROWTH_PRERULE_TBL:

	ID	PRERULE
1	0	2
2	1	3
3	2	3
4	3	1
5	3	2
6	4	1
7	4	3
8	5	1
9	5	3

PAL_FPGROWTH_POSTRULE_TBL:

	ID	POSTRULE
1	0	3
2	1	2
3	2	4
4	3	3
5	4	2
6	5	4

PAL_FPGROWTH_VALUES_TBL:

	ID	SUPPORT	CONFIDENCE	LIFT
1	0	0.5	0.7142857142857143	1.1904761904761905
2	1	0.5	0.8333333333333334	1.1904761904761905
3	2	0.3	0.5	1.25
4	3	0.3	0.6	1
5	4	0.3	0.75	1.0714285714285714
6	5	0.2	0.5	1.25

3.4.3 K-Optimal Rule Discovery (KORD)

K-optimal rule discovery (KORD) follows the idea of generating association rules with respect to a well-defined measure, instead of first finding all frequent itemsets and then generating all possible rules. The algorithm only calculates the top-k rules according to that measure. The size of the right hand side (RHS) of those rules is restricted to one. Furthermore, the KORD implementation generates only non-redundant rules.

The algorithm's search strategy is based on the so-called OPUS search. While the search space of all possible LHSs is traversed in a depth-first manner, the information about all qualified RHSs of the rules for a given LHS is propagated further to the deeper search levels. KORD does not build a real tree search structure; instead it traverses the LHSs in a specific order, which allows the pruning of the search space by simply not visiting those itemsets subsequently. In this way it is possible to use pruning rules which restrict the possible LHSs and RHSs at different rule generation stages.

Prerequisites

- There are no duplicated items in each transaction.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

KORD

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'KORD', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<TRANSACTION_table_type>	IN

Position	Schema Name	Table Type Name	Parameter Type
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Rules OUTPUT table type>	OUT
4	<schema_name>	<Antecedent table type>	OUT
5	<schema_name>	<Consequent table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<transaction_table>, <parameter_table>,
<rules_output_table>, <antecedent_output_table>, <consequent_output_table>) with
overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Transaction Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, varchar, or nvarchar	Transaction ID
	2nd column	Integer, varchar, or nvarchar	Item ID

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
TOPK	Integer	10	Specifies the number (k) of top rules.	
MAX_ANTECEDENT	Integer	4	Specifies the maximum length of antecedent rules.	
MEASURE_TYPE	Integer	0	Specifies the measure that will be used to define the priority of the rules. <ul style="list-style-type: none"> • 0: Leverage • 1: Lift 	

Name	Data Type	Default Value	Description	Dependency
IS_USE_EPSILON	Integer	0	Controls whether to use epsilon to punish the length of rules: <ul style="list-style-type: none">• 0: Does not use epsilon• Others: Uses epsilon	
THREAD_NUMBER	Integer	1	Specifies the number of threads.	
MIN_SUPPORT	Double	0.0	Specifies the minimum support.	
MIN_CONFIDENCE	Double	0.0	Specifies the minimum confidence.	
MIN_COVERAGE	Double	The value of MIN_SUPPORT	Specifies the minimum coverage. Default: T	
MIN_MEASURE	Double	0.0	Specifies the minimum measure value for leverage or lift, dependent on the MEASURE_TYPE setting.	
EPSILON	Double	0.0	Epsilon value.	Only valid when IS_USE_EPSILON is not 0.

Output Tables

Table	Column	Column Data Type	Description
Rules	1st column	Integer	ID
	2nd column	Double	Support
	3rd column	Double	Confidence
	4th column	Double	Lift
	5th column	Double	Leverage
	6th column	Double	Measure value
Antecedent	1st column	Integer	ID
	2nd column	Varchar or nvarchar	Antecedent items
Consequent	1st column	Integer	ID
	2nd column	Varchar or nvarchar	Consequent items

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and

- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_KORD_DATA_T;
CREATE TYPE PAL_KORD_DATA_T AS TABLE(
  "CUSTOMER" INTEGER,
  "ITEM" VARCHAR(20)
);
DROP TYPE PAL_KORD_RULES_T;
CREATE TYPE PAL_KORD_RULES_T AS TABLE(
  "ID" INTEGER,
  "SUPPORT" DOUBLE,
  "CONFIDENCE" DOUBLE,
  "LIFT" DOUBLE,
  "LEVERAGE" DOUBLE,
  "MEASURE" DOUBLE
);
DROP TYPE PAL_KORD_ANTE_ITEMS_T;
CREATE TYPE PAL_KORD_ANTE_ITEMS_T AS TABLE(
  "ID" INTEGER,
  "ANTECEDENT" VARCHAR(20)
);
DROP TYPE PAL_KORD_CONS_ITEMS_T;
CREATE TYPE PAL_KORD_CONS_ITEMS_T AS TABLE(
  "ID" INTEGER,
  "CONSEQUENT" VARCHAR(20)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
  "NAME" VARCHAR(100),
  "INTARGS" INTEGER,
  "DOUBLEARGS" DOUBLE,
  "STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_KORD_PDATA_TBL;
CREATE COLUMN TABLE PAL_KORD_PDATA_TBL(
  "POSITION" INT,
  "SCHEMA_NAME" NVARCHAR(256),
  "TYPE_NAME" NVARCHAR(256),
  "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_KORD_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_KORD_DATA_T', 'IN');
INSERT INTO PAL_KORD_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_KORD_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_KORD_RULES_T', 'OUT');
INSERT INTO PAL_KORD_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_KORD_ANTE_ITEMS_T',
'OUT');
INSERT INTO PAL_KORD_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL_KORD_CONS_ITEMS_T',
'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_KORD_PROC');
call "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'KORD', 'DM_PAL',
'PAL_KORD_PROC', PAL_KORD_PDATA_TBL);
DROP TABLE PAL_KORD_DATA_TBL;
CREATE COLUMN TABLE PAL_KORD_DATA_TBL LIKE PAL_KORD_DATA_T;
INSERT INTO PAL_KORD_DATA_TBL VALUES (2, 'item2');
INSERT INTO PAL_KORD_DATA_TBL VALUES (2, 'item3');
INSERT INTO PAL_KORD_DATA_TBL VALUES (3, 'item1');
INSERT INTO PAL_KORD_DATA_TBL VALUES (3, 'item2');
INSERT INTO PAL_KORD_DATA_TBL VALUES (3, 'item4');
INSERT INTO PAL_KORD_DATA_TBL VALUES (4, 'item1');
INSERT INTO PAL_KORD_DATA_TBL VALUES (4, 'item3');
INSERT INTO PAL_KORD_DATA_TBL VALUES (5, 'item2');
INSERT INTO PAL_KORD_DATA_TBL VALUES (5, 'item3');
INSERT INTO PAL_KORD_DATA_TBL VALUES (6, 'item1');
INSERT INTO PAL_KORD_DATA_TBL VALUES (6, 'item3');

```

```

INSERT INTO PAL_KORD_DATA_TBL VALUES (0, 'item1');
INSERT INTO PAL_KORD_DATA_TBL VALUES (0, 'item2');
INSERT INTO PAL_KORD_DATA_TBL VALUES (0, 'item5');
INSERT INTO PAL_KORD_DATA_TBL VALUES (1, 'item2');
INSERT INTO PAL_KORD_DATA_TBL VALUES (1, 'item4');
INSERT INTO PAL_KORD_DATA_TBL VALUES (7, 'item1');
INSERT INTO PAL_KORD_DATA_TBL VALUES (7, 'item2');
INSERT INTO PAL_KORD_DATA_TBL VALUES (7, 'item3');
INSERT INTO PAL_KORD_DATA_TBL VALUES (7, 'item5');
INSERT INTO PAL_KORD_DATA_TBL VALUES (8, 'item1');
INSERT INTO PAL_KORD_DATA_TBL VALUES (8, 'item2');
INSERT INTO PAL_KORD_DATA_TBL VALUES (8, 'item3');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TOPK', 5, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MEASURE_TYPE', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_SUPPORT', null, 0.1, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MIN_CONFIDENCE', null, 0.2, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('IS_USE_EPSILON', 0, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EPSILON', null, 0.1, null);
DROP TABLE PAL_KORD_RULES_TBL;
CREATE COLUMN TABLE PAL_KORD_RULES_TBL LIKE PAL_KORD_RULES_T;
DROP TABLE PAL_KORD_ANTE_ITEMS_TBL;
CREATE COLUMN TABLE PAL_KORD_ANTE_ITEMS_TBL LIKE PAL_KORD_ANTE_ITEMS_T;
DROP TABLE PAL_KORD_CONS_ITEMS_TBL;
CREATE COLUMN TABLE PAL_KORD_CONS_ITEMS_TBL LIKE PAL_KORD_CONS_ITEMS_T;

CALL "DM_PAL".PAL_KORD_PROC(PAL_KORD_DATA_TBL, #PAL_CONTROL_TBL,
PAL_KORD_RULES_TBL, PAL_KORD_ANTE_ITEMS_TBL, PAL_KORD_CONS_ITEMS_TBL) with
OVERVIEW;
SELECT * FROM PAL_KORD_RULES_TBL ORDER BY "MEASURE" DESC;
SELECT * FROM PAL_KORD_ANTE_ITEMS_TBL;
SELECT * FROM PAL_KORD_CONS_ITEMS_TBL;

```

Expected Result

PAL_KORD_RULES_TBL

ID	SUPPORT	CONFIDENCE	LIFT	LEVERAGE	MEASURE
2	0.2222222222...	0.5	2.25	0.12345679012...	2.25
3	0.2222222222...	1	1.5	0.07407407407...	1.5
1	0.2222222222...	0.333333333333...	1.5	0.07407407407...	1.5
4	0.2222222222...	0.2857142857142...	1.2857142...	0.04938271604...	1.28571428571...
0	0.2222222222...	0.2857142857142...	1.2857142...	0.04938271604...	1.28571428571...

PAL_KORD_ANTE_ITEMS_TBL

ID	ANTECEDENT
0	item2
1	item1
2	item2
2	item1
3	item5
4	item2

PAL_KORD_CONS_ITEMS_TBL

ID	CONSEQUENT
0	item5
1	item5
2	item5
3	item1
4	item4

3.5 Time Series Algorithms

Financial market data or economic data usually comes with time stamps. Predicting the future values, such as stock value for tomorrow, is of great interest in many business scenarios. Quantity over time is called time series, and predicting the future value based on existing time series is also known as forecasting. In this release of PAL, three smoothing based time series models are implemented. These models can be used to smooth the existing time series and forecast. In the time series algorithms, let x_t be the observed values for the t -th time period, and T be the total number of time periods.

3.5.1 ARIMA

The auto regressive integrated moving average (ARIMA) algorithm is famous in econometrics, statistics and time series analysis. An ARIMA model can be written as ARIMA (p, d, q), where p refers to the auto regressive order, d refers to integrated order, and q refers to the moving average order. This algorithm helps you understand the time series data better and predict future data in the series.

The auto regressive integrated moving average with intervention (ARIMAX) algorithm is an extension for ARIMA. Compared with ARIMA, an ARIMAX model can not only get the internal relationship with former data, but also take external factor into consideration.

Prerequisite

No missing or null data in the inputs.

ARIMATRAIN

This function generates ARIMA or ARIMAX training model.

ARIMA Model

An ARIMA model is a universalization of an auto regressive moving average (ARMA) model. The integrated part is mainly applied to induce stationary when data show evidence of non-stationary.

An ARIMA (p, d, q) model can be expressed as:

$$\Phi(B)(1-B)^d(Y_t - c) = \Theta(B)\varepsilon_t, \quad t \in \mathbb{Z}$$

An ARMA (p, q) model can be expressed as:

$$\Phi(B)(Y_t - c) = \Theta(B)\varepsilon_t, \quad t \in \mathbb{Z}$$

$$\varepsilon_t \sim i.i.d.N(0, \sigma^2)$$

Where B is lag operator (backward shift operator), c is the mean of the series data,

$$\Phi(B) = 1 - \varphi_1 B - \varphi_2 B^2 - \dots - \varphi_p B^p, \quad p \geq 0$$

$$\Theta(B) = 1 + \theta_1 B + \theta_2 B^2 + \dots + \theta_q B^q, \quad q \geq 0$$

ARIMAX Model

An ARIMAX model is a universalization of an ARMAX model. The integrated part is mainly applied to induce stationary when data show evidence of non-stationary.

An ARIMAX (p, d, q) model can be expressed as:

$$\Phi(B)(1-B)^d(Y_t - c) = H^T(1-B)^dX_t + \Theta(B)\varepsilon_t, \quad t \in \mathbb{Z}$$

An ARMAX (p, q) model can be expressed as:

$$\Phi(B)Y_t = H^TX_t + \Theta(B)\varepsilon_t, \quad t \in \mathbb{Z}$$

$$\varepsilon_t \sim i.i.d.N(0, \sigma^2)$$

Where B is lag operator (backward shift operator), X_t is a covariate vector at time t, and H is its coefficient vector.

In PAL, the ARIMATRAIN algorithm first converts the original non-stationary time series data to a new stationary time series data by the integrated step, and then ARIMA fits the stationary time series data to an ARMA model, and ARIMAX fits the stationary time series data to an ARMAX model.

PAL provides two parameter estimation methods: conditional sum of squares (CSS or conditional maximum likelihood estimation) and maximum likelihood estimation (MLE).

SARIMA Model

The ARIMA (p,d,q) model is used to model non-seasonal data. To treat with seasonal data, a seasonal ARIMA (SARIMA) model, written as ARIMA (p,d,q)(P,D,Q)_s, is introduced. The definition of SARIMA is as follows:

$$\Phi(B^s)\phi(B)(1-B)^d(1-B^s)^D Y_t = \Theta(B^s)\theta(B)\varepsilon_t$$

Where

$$\varepsilon_t \sim N(0, \sigma^2)$$

$$\Phi(B^s) = 1 - \Phi_1 B^s - \Phi_2 B^{2s} - \dots - \Phi_p B^{ps}$$

$$\phi(B) = 1 - \phi_1 B^1 - \phi_2 B^2 - \dots - \phi_p B^p$$

$$\Theta(B^s) = 1 + \Theta_1 B^s + \Theta_2 B^{2s} + \dots + \Theta_q B^{qs}$$

$$\theta(B) = 1 + \theta_1 B^1 + \theta_2 B^2 + \dots + \theta_q B^q$$

CSS Estimation

An ARMA (p, q) model can also be expressed as:

$$Y_t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \dots + \phi_p Y_{t-p} + \dots + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q}$$

$$\text{Where } \Phi_0 = C(1 - \sum_{i=1}^p \Phi_i).$$

Let r = max (p, q), y_i, i = 1, 2, ..., N denote the observed series data, with the length N,

$$Y_{r+1} = \phi_0 + \phi_1 Y_r + \phi_2 Y_{r-1} + \dots + \phi_p Y_{r-p+1} + \dots + \varepsilon_{r+1} + \theta_1 \varepsilon_r + \theta_2 \varepsilon_{r-1} + \dots + \theta_q \varepsilon_{r-q+1}$$

Conditional on Y_r = Y_r, Y_{r-1} = Y_{r-1}, ..., Y_{r-p+1} = Y_{r-p+1}, ε_r = ε_{r-1} = ... = ε_{r-q+1} = 0 we have

$$Y_{r+1} \sim N((\phi_0 + \phi_1 Y_r + \phi_2 Y_{r-1} + \dots + \phi_p Y_{r-p+1}), \sigma^2)$$

Where the sequence {ε_{r+1}, ε_{r+2}, ..., ε_N} can be calculated by iterations:

$$\varepsilon_t = Y_t - \phi_0 - \phi_1 Y_{t-1} - \dots - \phi_p Y_{t-p} - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \dots - \theta_q \varepsilon_{t-q}, \quad t = r+1, r+2, \dots, N$$

Therefore the log likelihood is

$$L(\phi, \theta, \sigma^2) = \log f(Y_N, Y_{N-1}, \dots, Y_{r+1} | Y_r, \dots, Y_1, \varepsilon_r = \varepsilon_{r-1} = \dots = \varepsilon_{r-q+1} = 0, \phi, \theta, \sigma^2)$$

$$L(\phi, \theta, \sigma^2) = -\frac{N-r}{2} \log(2\pi) - \frac{N-r}{2} \log(\sigma^2) - \sum_{t=r+1}^N \frac{\varepsilon_t^2}{2\sigma^2}$$

MLE Estimation

Kalman filtering is applied to calculate MLE. An ARMA (p, q) model can be expressed as a Kalman state space model.

$$\begin{cases} Y_t = c + H\alpha_t + w_t \\ \alpha_{t+1} = F\alpha_t + v_{t+1} \end{cases}$$

Where

$$H = [1 \ \theta_1 \ \theta_2 \dots \ \theta_q], F = \begin{bmatrix} \varphi_1 & \varphi_2 & \cdots & \varphi_{r-1} & \varphi_r \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix}, r = \max(p, q+1)$$

$$\alpha_t \sim N(a_t, P_t), v_t \sim N(0, Q), Q = \begin{bmatrix} \sigma^2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

Initial P_0 can be calculated by

$$(I_{r^2} - F \otimes F) \text{Vec}(P_0) = \text{Vec}(Q)$$

Let d_t denote the first element of P_t , therefore the log likelihood is

$$L(\varphi, \theta, \sigma^2) = \log f(Y_N, Y_{N-1}, \dots, Y_1 | y_N, \dots, y_1, \varphi, \theta, \sigma^2)$$

$$L(\varphi, \theta, \sigma^2) = -\frac{N}{2} \log(2\pi) - \frac{1}{2} \sum_{t=1}^N \log(d_t) - \frac{1}{2} \sum_{t=r+1}^N \frac{v_t^2}{d_t}$$

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'ARIMATRAIN',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	Time stamp
	2nd column	Integer or double	Raw data
	(For ARIMAX only) Other columns	Integer or double	External (Intervention) data

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
D	Integer	0	Value of the differentiation order
P	Integer	0	Value of the auto regression order
Q	Integer	0	Value of the moving average order
METHOD	Integer	1	<ul style="list-style-type: none">• 0: Uses the CSS maximized likelihood method• 1: Uses the MLE maximized likelihood method
SEASONAL_P	Integer	0	Value of the auto regression order for seasonal part
SEASONAL_Q	Integer	0	Value of the moving average order for seasonal part
SEASONAL_D	Integer	0	Value of the differentiation order for seasonal part
SEASONAL_PERIOD	Integer	0	Value of the seasonal period

Output Table (Model Table)

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Varchar or nvarchar	Name	The minimum length is 50.
	2nd column	Varchar or nvarchar	Value	The table must be a column table. The minimum length of each unit (row) is 1024.

Examples

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1: ARIMA

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ARIMA_DATA_T;
CREATE TYPE PAL_ARIMA_DATA_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "X1" DOUBLE);
DROP TYPE PAL_ARIMA_CONTROL_T;
CREATE TYPE PAL_ARIMA_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100));
DROP TYPE PAL_ARIMA_MODEL_T;
CREATE TYPE PAL_ARIMA_MODEL_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" VARCHAR (5000));

DROP TABLE PAL_ARIMA_PDATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_ARIMA_DATA_T', 'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_ARIMA_CONTROL_T',
'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_ARIMA_MODEL_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_ARIMATRAIN PROC');
CALL SYS.AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'ARIMATRAIN', 'DM_PAL',
'PAL_ARIMATRAIN PROC', PAL_ARIMA_PDATA_TBL);
DROP TABLE PAL_ARIMA_DATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_DATA_TBL LIKE PAL_ARIMA_DATA_T;
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(1, 0.8);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(2, 1.2);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(3, 1.34845613096197);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(4, 1.32261090809898);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(5, 1.38095306748554);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(6, 1.54066648969168);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(7, 1.50920806756785);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(8, 1.48461408893443);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(9, 1.43784887380224);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(10, 1.64251548718992);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(11, 1.74292337447476);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(12, 1.91137546943257);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(13, 2.07735796176367);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(14, 2.01741246166924);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(15, 1.87176938196573);

```

```

INSERT INTO PAL_ARIMA_DATA_TBL VALUES(16, 1.83354723357744);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(17, 1.66104978144571);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(18, 1.65115984070812);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(19, 1.69470966154593);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(20, 1.70459802935728);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(21, 1.61246059980916);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(22, 1.53949706614636);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(23, 1.59231354902055);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(24, 1.81741927705578);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(25, 1.80224252773564);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(26, 1.81881576781466);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(27, 1.78089755157948);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(28, 1.61473635574416);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(29, 1.42002147867225);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(30, 1.49971641345022);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR (50), "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('P', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('Q', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('D', 0, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('METHOD', 1, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('STATIONARY', 1, null, null);
DROP TABLE PAL_ARIMA_MODEL_TBL;
CREATE COLUMN TABLE PAL_ARIMA_MODEL_TBL LIKE PAL_ARIMA_MODEL_T;
CALL DM_PAL.PAL_ARIMATRAIN_PROC(PAL_ARIMA_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_ARIMA_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ARIMA_MODEL_TBL;

```

Expected Result

	NAME	VALUE
1	p	1
2	ARParameters	0.888972;
3	d	0
4	q	1
5	MAParamet...	0.463126;
6	Intercept	1.43046
7	Sigma2	0.0145949
8	logLikelihood	19.5916
9	SeriesData	
10	DeltaSeriesD...	1.49971;
11	Eps	0.135558;
12	FORECASTI...	1
13	AllDeltaSeri...	0.8;1.2;1.348...
14	AIC	-33.1832
15	AICC	-32.2601
16	BIC	-28.9796

Example 2: ARIMAX

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ARIMAX_DATA_T;
CREATE TYPE PAL_ARIMAX_DATA_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "X1" DOUBLE,
    "Xreg" DOUBLE);
DROP TYPE PAL_ARIMAX_CONTROL_T;

```

```

CREATE TYPE PAL_ARIMAX_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100) );
DROP TYPE PAL_ARIMAX_MODEL_T;
CREATE TYPE PAL_ARIMAX_MODEL_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" VARCHAR (5000) );

DROP TABLE PAL_ARIMAX_PDATA_TBL;
CREATE COLUMN TABLE PAL_ARIMAX_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES (1,'DM_PAL','PAL_ARIMAX_DATA_T','IN');
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES(2,'DM_PAL', 'PAL_ARIMAX_CONTROL_T','IN');
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES(3,'DM_PAL', 'PAL_ARIMAX_MODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_ARIMAXTRAIN PROC');
CALL SYS.AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'ARIMATRAIN', 'DM_PAL',
'PAL_ARIMAXTRAIN PROC',PAL_ARIMAX_PDATA_TBL);
DROP TABLE PAL_ARIMAX_DATA_TBL;
CREATE COLUMN TABLE PAL_ARIMAX_DATA_TBL LIKE PAL_ARIMAX_DATA_T;
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(1, 1.2, 0.8);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(2, 1.34845613096197, 1.2);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(3, 1.32261090809898, 1.34845613096197);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(4, 1.38095306748554, 1.32261090809898);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(5, 1.54066648969168, 1.38095306748554);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(6, 1.50920806756785, 1.54066648969168);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(7, 1.48461408893443, 1.50920806756785);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(8, 1.43784887380224, 1.48461408893443);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(9, 1.64251548718992, 1.43784887380224);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(10, 1.74292337447476, 1.64251548718992);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(11, 1.91137546943257, 1.74292337447476);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(12, 2.07735796176367, 1.91137546943257);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(13, 2.01741246166924, 2.07735796176367);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(14, 1.87176938196573, 2.01741246166924);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(15, 1.83354723357744, 1.87176938196573);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(16, 1.66104978144571, 1.83354723357744);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(17, 1.65115984070812, 1.66104978144571);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(18, 1.69470966154593, 1.65115984070812);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(19, 1.70459802935728, 1.69470966154593);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(20, 1.61246059980916, 1.70459802935728);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(21, 1.53949706614636, 1.61246059980916);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(22, 1.59231354902055, 1.53949706614636);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(23, 1.81741927705578, 1.59231354902055);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(24, 1.80224252773564, 1.81741927705578);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(25, 1.81881576781466, 1.80224252773564);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(26, 1.78089755157948, 1.81881576781466);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(27, 1.61473635574416, 1.78089755157948);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(28, 1.42002147867225, 1.61473635574416);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(29, 1.49971641345022, 1.42002147867225);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR
(50),"INTARGS" INTEGER,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('P', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('Q', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('D', 0,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('METHOD', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('STATIONARY', 1,null,null);
DROP TABLE PAL_ARIMAX_MODEL_TBL;
CREATE COLUMN TABLE PAL_ARIMAX_MODEL_TBL LIKE PAL_ARIMAX_MODEL_T;
CALL DM_PAL.PAL_ARIMAXTRAIN PROC(PAL_ARIMAX_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_ARIMAX_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ARIMAX_MODEL_TBL;

```

Expected Result

	NAME	VALUE
1	p	1
2	ARParameters	0.302187;
3	d	0
4	q	1
5	MAParamet...	0.291634;
6	Intercept	0.732762
7	Sigma2	0.00874418
8	logLikelihood	27.3948
9	SeriesData	
10	DeltaSeriesD...	0.701638;
11	Eps	0.0788653;
12	RegCoeffNa...	Xreg;
13	RegCoeffVal...	0.562018;
14	FORECASTI...	1
15	AllDeltaSeri...	0.750385;0.6...
16	AIC	-48.7896
17	AICC	-47.8296
18	BIC	-44.6877

Example 3: SARIMA

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ARIMA_DATA_T;
CREATE TYPE PAL_ARIMA_DATA_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "SERIES" DOUBLE
);
DROP TYPE PAL_ARIMA_CONTROL_T;
CREATE TYPE PAL_ARIMA_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_ARIMA_MODEL_T;
CREATE TYPE PAL_ARIMA_MODEL_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" VARCHAR (5000)
);

DROP TABLE PAL_ARIMA_PDATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_ARIMA_DATA_T', 'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_ARIMA_CONTROL_T',
'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_ARIMA_MODEL_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_ARIMATRAIN_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'ARIMATRAIN', 'DM_PAL',
'PAL_ARIMATRAIN_PROC', PAL_ARIMA_PDATA_TBL);
DROP TABLE PAL_ARIMA_DATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_DATA_TBL LIKE PAL_ARIMA_DATA_T;

```

```

INSERT INTO PAL_ARIMA_DATA_TBL VALUES (1, -0.636126431 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (2, 3.092508651 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (3, -0.73733556 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (4, -3.142190983 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (5, 2.088819813 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (6, 3.179302734 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (7, -0.871376102 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (8, -3.475633275 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (9, 1.779244219 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (10, 3.609159416 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (11, 0.082170143 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (12, -4.42439631 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (13, 0.499210261 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (14, 4.514017351 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (15, -0.320607187);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (16, -3.70219307 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (17, 0.100228116 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (18, 4.553625233 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (19, 0.261489853 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (20, -4.474116429);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (21, -0.372574233);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (22, 2.872305281 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (23, 1.289850031 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (24, -3.662763983);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (25, -0.168962933);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (26, 4.018728154 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (27, -1.306247869);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (28, -2.182690245);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (29, -0.845114493);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (30, 0.99806763 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (31, -0.641201109);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (32, -2.640777923);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (33, 1.493840358 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (34, 4.326449202 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (35, -0.653797151);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES (36, -4.165384227);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEASONAL_P', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEASONAL_Q', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEASONAL_D', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEASONAL_PERIOD', 4, null, null); ;
DROP TABLE PAL_ARIMA_MODEL_TBL;
CREATE COLUMN TABLE PAL_ARIMA_MODEL_TBL LIKE PAL_ARIMA_MODEL_T;
CALL "DM_PAL".PAL_ARIMATRAIN_PROC(PAL_ARIMA_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_ARIMA_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ARIMA_MODEL_TBL;

```

Expected Result

	NAME	VALUE
1	p	0
2	ARParameters	
3	d	0
4	q	0
5	MAParamet...	
6	Intercept	-0.0163595
7	Sigma2	0.919718
8	logLikelihood	-48.285
9	SeriesData	1.49384;4.3264...
10	DeltaSeriesD...	2.33895;3.3283...
11	Eps	1.32569;1.1691...
12	Period	4
13	SeasonalP	1
14	SARParamet...	0.047218;
15	SeasonalD	1
16	SeasonalQ	1
17	SMAParamete...	-0.999416;
18	FORECASTI...	1
19	AllDeltaSeri...	2.72494;0.0867...
20	AIC	102.57
21	AICC	103.32
22	BIC	107.32

ARIMAFORECAST

An ARMA (p, q) model can be transformed to a MA (∞) model. More generally, an ARIMA (p, d, q) can also be changed to a MA (∞) model.

$$Y_t = \Psi(B) \varepsilon_t = \varepsilon_t + \psi_1 \varepsilon_{t-1} + \psi_2 \varepsilon_{t-2} + \dots$$

$\hat{Y}_N(1)$ denotes the 'l-step ahead' of series Y.

At $t=N+1$

$$Y_{N+1} = \sum_{j=0}^{\infty} \psi_j \varepsilon_{N+1-j}$$

$$\hat{Y}_N(1) = \psi_1^+ \varepsilon_N + \psi_{1+1}^+ \varepsilon_{N-1} + \psi_{1+2}^+ \varepsilon_{N-2} + \dots$$

Minimize the mean square forecast error

$$\min_{\psi} E (Y_{N+1} - \hat{Y}_N(1))^2$$

We get

$$\hat{Y}_N(1) = \psi_1 \varepsilon_N + \psi_{1+1} \varepsilon_{N-1} + \psi_{1+2} \varepsilon_{N-2} + \dots$$

And variance of the forecast error

$$\text{Var}(e_N(1)) = \sigma^2 \sum_{i=1}^1 \psi_i^2$$

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'ARIMAFORECAST',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output
table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Varchar or nvarchar	Name
	2nd column	Varchar or nvarchar	Value

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
ForecastLength	Integer	1	Length of the final forecast results

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	Time stamp
	2nd column	Double	Expected value
	3rd column	Double	Low80% value
	4th column	Double	Hi80% value
	5th column	Double	Low95% value
	6th column	Double	Hi95% value

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ARIMA_MODEL_T;
CREATE TYPE PAL_ARIMA_MODEL_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" VARCHAR (5000)
);
DROP TYPE PAL_ARIMA_CONTROL_T;
CREATE TYPE PAL_ARIMA_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100) );
DROP TYPE PAL_ARIMA_RESULT_T;
CREATE TYPE PAL_ARIMA_RESULT_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "MEAN" DOUBLE,
    "LOW80%" DOUBLE,
    "HIGH80%" DOUBLE,
    "LOW95%" DOUBLE,
    "HIGH95%" DOUBLE
);
DROP TABLE PAL_ARIMA_PDATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_ARIMA_MODEL_T', 'IN');

```

```

INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_ARIMA_CONTROL_T',
'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_ARIMA_RESULT_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_ARIMAFORECAST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'ARIMAFORECAST', 'DM_PAL',
'PAL_ARIMAFORECAST_PROC', 'PAL_ARIMA_PDATA_TBL');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR
(50), "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('ForecastLength', 30, null, null);
DROP TABLE PAL_ARIMA_RESULT_TBL;
CREATE COLUMN TABLE PAL_ARIMA_RESULT_TBL LIKE PAL_ARIMA_RESULT_T;
SELECT * FROM PAL_ARIMA_MODEL_TBL;
CALL DM_PAL.PAL_ARIMAFORECAST_PROC(PAL_ARIMA_MODEL_TBL, "#PAL_CONTROL_TBL",
PAL_ARIMA_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ARIMA_RESULT_TBL;

```

Expected Result

	TIMESTAMP	MEAN	LOW80%	HIGH80%	LOW95%	HIGH95%
1	0	1.5548362553700001	1.400014377422394	1.7096581333176062	1.3180567043336773	1.791615806406323
2	1	1.541084288474145	1.2807132398532222	1.8014553370950677	1.1428812643758186	1.9392873125724712
3	2	1.5288582398250539	1.2088103534476486	1.848906126202459	1.0393873985515008	2.018329081098607
4	3	1.517988793534066	1.1577001462826058	1.878277440785526	0.966975039724965	2.0690025473431666
5	4	1.508325421003526	1.119163912415553	1.897486929591499	0.9131544509468152	2.1034963910602364
6	5	1.4997342962889748	1.0891861341520443	1.9102824584259053	0.8718552729995986	2.127613319578351
7	6	1.49209644277275	1.0654022200187179	1.9187906655267823	0.8395241588499096	2.1446687266955906
8	7	1.4853060854826858	1.0462700825491975	1.9243420884161742	0.8138586838528407	2.156753487112531
9	8	1.4792691862375271	1.0307185461249284	1.9278198263501258	0.7932704075709966	2.1652679649040576
10	9	1.4739021413326112	1.017971675704438	1.9298326069607845	0.7766168972422763	2.171187385422946
11	10	1.4691306237303448	1.0074506610592722	1.9308105864014173	0.7630522856241688	2.1752089618365207
12	11	1.4648885537212257	0.9987144044408502	1.9310627030016012	0.7519369525999589	2.1778401548424924
13	12	1.4611171838003185	0.9914212899396596	1.9308130776609773	0.7427795409151976	2.1794548266854394
14	13	1.4577642850858352	0.9853034059110768	1.9302251642605937	0.7351979634766644	2.180330606695006
15	14	1.4547834240127109	0.980148518992835	1.9294183290325868	0.7288922178951472	2.1806746301302744
16	15	1.4521333192842605	0.9757871018029123	1.9284795367656087	0.7236248874768791	2.180641751091642
17	16	1.4497772701764788	0.9720827825846626	1.927471757768295	0.7192068378581598	2.1803477024947977
18	17	1.4476826482776968	0.9689251851805663	1.9264401113748273	0.7154865357503046	2.1798787608050887
19	18	1.4458204456248036	0.9662244820451821	1.9254164092044252	0.7123419576965968	2.17929893355301
20	19	1.4441648729782754	0.9639072010631957	1.9244225448933552	0.7096743898161664	2.1786553561403847
21	20	1.442693002672606	0.9619129660901078	1.923473039255104	0.7074036319561783	2.1779823733890336
22	21	1.4413844510960536	0.9601919428220533	1.922576959370054	0.7054642595730075	2.1773046426190996
23	22	1.4402210964024356	0.9587028237508823	1.9217393690539888	0.7038026914222014	2.17663950138267
24	23	1.4391868275456212	0.9574112291523921	1.9209624259388502	0.7023748769348095	2.175998778156433
25	24	1.4382673211611592	0.9562884317363896	1.9202462105859288	0.7011444638502353	2.175390178472083
26	25	1.437449843205117	0.9553103347997394	1.9195893516104947	0.7000813404426559	2.1748183459675783
27	26	1.4367230726030773	0.9544566500709843	1.9189894951351703	0.6991604714977838	2.1742856737083707
28	27	1.4360769444670398	0.9537102336357871	1.9184436552982924	0.6983609656946878	2.1737929232393918
29	28	1.4355025107089772	0.9530565475468795	1.9179484738710748	0.6976653259961932	2.173339695421761
30	29	1.4349918161207091	0.9524832217511207	1.9175004104902977	0.6970588452736455	2.172924786967773

ARIMAXFORECAST

The forecast procedure of this algorithm is similar to that of ARIMAFORECAST. However, this algorithm requires one more table to restore the future external data. The mean of ARIMAX forecast is the sum of external value and ARIMA forecast.

$$Y_{t+1} = H^T X_{t+1} + \varphi_0 + \varphi_1 Y_t + \varphi_2 Y_{t-1} + \dots + \varphi_p Y_{t-p+1} + \dots + \varepsilon_{t+1} + \theta_1 \varepsilon_t + \theta_2 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q+1}$$

The variance of the ARIMAX forecast error is equal to the ARIMA forecast. Refer to ARIMAFORECAST for more information.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'ARIMAXFORECAST',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter
1	<schema_name>	<External INPUT table type>	IN
2	<schema_name>	<Model table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<external input table>, <model table>,
<parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	Time stamp
	Other columns	Integer or double	External (Intervention) data Note: The count and the names of the intervention columns should be consistent with the input data of the ARIMAX model.

Model Table

Table	Column	Column Data Type	Description
Model	1st column	Varchar or nvarchar	Name
	2nd column	Varchar or nvarchar	Value

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
ForecastLength	Integer	1	Length of the final forecast results

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	Time stamp
	2nd column	Double	Expected value
	3rd column	Double	Low80% value
	4th column	Double	Hi80% value
	5th column	Double	Low95% value
	6th column	Double	Hi95% value

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_ARIMAX_DATA_T;
CREATE TYPE PAL_ARIMAX_DATA_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "Xreg" DOUBLE);
```

```

DROP TYPE PAL_ARIMAX_MODEL_T;
CREATE TYPE PAL_ARIMAX_MODEL_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" VARCHAR (5000)
);
DROP TYPE PAL_ARIMAX_CONTROL_T;
CREATE TYPE PAL_ARIMAX_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100) );
DROP TYPE PAL_ARIMAX_RESULT_T;
CREATE TYPE PAL_ARIMAX_RESULT_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "MEAN" DOUBLE,
    "LOW80%" DOUBLE,
    "HIGH80%" DOUBLE,
    "LOW95%" DOUBLE,
    "HIGH95%" DOUBLE
);
DROP TABLE PAL_ARIMAX_PDATA_TBL;
CREATE COLUMN TABLE PAL_ARIMAX_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES (1,'DM_PAL','PAL_ARIMAX_MODEL_T','IN');
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES (2,'DM_PAL','PAL_ARIMAX_MODEL_T','IN');
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES (3,'DM_PAL','PAL_ARIMAX_CONTROL_T','IN');
INSERT INTO PAL_ARIMAX_PDATA_TBL VALUES (4,'DM_PAL','PAL_ARIMAX_RESULT_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_ARIMAXFORECAST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'ARIMAXFORECAST', 'DM_PAL',
'PAL_ARIMAXFORECAST_PROC',PAL_ARIMAX_PDATA_TBL);
DROP TABLE PAL_ARIMAX_DATA_TBL;
CREATE COLUMN TABLE PAL_ARIMAX_DATA_TBL LIKE PAL_ARIMAX_DATA_T;
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(1, 0.8);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(2, 1.2);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(3, 1.34845613096197);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(4, 1.32261090809898);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(5, 1.38095306748554);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(6, 1.54066648969168);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(7, 1.50920806756785);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(8, 1.48461408893443);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(9, 1.43784887380224);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(10, 1.64251548718992);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(11, 1.74292337447476);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(12, 1.91137546943257);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(13, 2.07735796176367);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(14, 2.01741246166924);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(15, 1.87176938196573);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(16, 1.83354723357744);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(17, 1.66104978144571);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(18, 1.65115984070812);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(19, 1.69470966154593);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(20, 1.70459802935728);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(21, 1.61246059980916);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(22, 1.53949706614636);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(23, 1.59231354902055);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(24, 1.81741927705578);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(25, 1.80224252773564);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(26, 1.81881576781466);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(27, 1.78089755157948);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(28, 1.61473635574416);
INSERT INTO PAL_ARIMAX_DATA_TBL VALUES(29, 1.42002147867225);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR
(50),"INTARGS" INTEGER,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('ForecastLength', 29, null,null);
DROP TABLE PAL_ARIMAX_RESULT_TBL;

```

```

CREATE COLUMN TABLE PAL_ARIMAX_RESULT_TBL LIKE PAL_ARIMAX_RESULT_T;
SELECT * FROM PAL_ARIMAX_MODEL_TBL;
CALL DM_PAL.PAL_ARIMAXFORECAST_PROC(PAL_ARIMAX_DATA_TBL, PAL_ARIMAX_MODEL_TBL,
"#PAL_CONTROL_TBL", PAL_ARIMAX_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ARIMAX_RESULT_TBL;

```

Expected Result

TIMESTA...	MEAN	LOW80%	HIGH80%	LOW95%	HIGH95%
0	1.1959558075853	1.0761174...	1.31579...	1.012678...	1.37923277...
1	1.411286001927...	1.2719132...	1.55065...	1.198133...	1.62443812...
2	1.491857284885...	1.3508352...	1.63287...	1.276182...	1.70753187...
3	1.476465421551...	1.3352936...	1.61763...	1.260561...	1.69236893...
4	1.508993791375...	1.3678083...	1.65017...	1.293069...	1.72491819...
5	1.598678586326...	1.4574919...	1.73986...	1.382752...	1.81460489...
6	1.580974072843...	1.4397872...	1.72216...	1.365047...	1.79690056...
7	1.567144269976...	1.4259574...	1.70833...	1.351217...	1.78307077...
8	1.540858586010...	1.3996717...	1.68204...	1.324932...	1.75678509...
9	1.655886907560...	1.5147001...	1.79707...	1.439960...	1.87181341...
10	1.712319053446...	1.5711322...	1.85350...	1.496392...	1.92824555...
11	1.806994292562...	1.6658074...	1.94818...	1.591067...	2.02292079...
12	1.900281580496...	1.7590947...	2.04146...	1.684355...	2.11620808...
13	1.866590345628...	1.7254035...	2.00777...	1.650663...	2.08251685...
14	1.784734418236...	1.6435476...	1.92592...	1.568807...	2.00066092...
15	1.763252385452...	1.6220655...	1.90443...	1.547325...	1.97917889...
16	1.666303469781...	1.5251166...	1.80749...	1.450376...	1.88222997...
17	1.660745016452...	1.5195582...	1.80193...	1.444818...	1.87667152...
18	1.685221365794...	1.5440345...	1.82640...	1.469294...	1.90114787...
19	1.690778935039...	1.5495921...	1.83196...	1.474852...	1.90670543...
20	1.638994843371...	1.4978080...	1.78018...	1.423068...	1.85492134...
21	1.597987075583...	1.4568002...	1.73917...	1.382060...	1.81391358...
22	1.627671576269...	1.4864847...	1.76885...	1.411745...	1.84359808...
23	1.754187973702...	1.6130011...	1.89537...	1.538261...	1.97011447...
24	1.745658170105...	1.6044713...	1.88684...	1.529731...	1.96158467...
25	1.754972844800...	1.6137860...	1.89615...	1.539046...	1.97089934...
26	1.733661631811...	1.5924748...	1.87484...	1.517735...	1.94958813...
27	1.640273888755...	1.4990870...	1.78146...	1.424347...	1.85620039...
28	1.530838091679...	1.3896512...	1.67202...	1.314911...	1.74676459...

Related Information

[Seasonality Test \[page 409\]](#)

3.5.2 Auto ARIMA

This function automatically identifies the orders of an ARIMA model, that is, $(p,d,q)(P,D,Q)_m$, where m is the seasonal period according to some information criterion such as AICC, AIC, and BIC. If order selection succeeds, the function gives the optimal model as in the ARIMATRAIN function.

Successively, you can use functions such as ARIMAFORECAST and ARIMAXFORECAST, which are described in the ARIMA topic, to make forecast.

Prerequisite

No missing or null data in the inputs.

AUTOARIMA

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'AUTOARIMA',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output
table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer	Time stamp	No identical value is allowed.
	2nd column	Integer or double	Raw data	
	(For auto ARIMAX only) Other columns	Integer or double	External (Intervention) data	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
SEASONAL_PERIOD	Integer	-1	<p>Value of the seasonal period.</p> <ul style="list-style-type: none"> • Negative: Automatically identify seasonality by means of auto-correlation scheme. • 0 or 1: Non-seasonal. • Others: Seasonal period. 	
SEASONALITY_CRITERION	Double	0.5	<p>The criterion of the auto-correlation coefficient for accepting seasonality, in the range of (0, 1). The larger it is, the less probable a time series is regarded to be seasonal.</p>	Valid only when SEASONAL_PERIOD is negative. Refer to <i>Seasonality Test</i> for more information.

Name	Data Type	Default Value	Description	Dependency
D	Integer	-1	<p>Order of first-differencing.</p> <ul style="list-style-type: none"> • Negative: Automatically identifies first-differencing order with KPSS test. • Others: Uses the specified value as the first-differencing order. 	
KPSS_SIGNIFICANCE_LEVEL	double	0.05	<p>The significance level for KPSS test. Supported values are 0.01, 0.025, 0.05, and 0.1. The smaller it is, the larger probable a time series is considered as first-stationary, that is, the less probable it needs first-differencing.</p>	Valid only when D is negative.
MAX_D	Integer	2	The maximum value of D when KPSS test is applied.	
SEASONAL_D	Integer	-1	<p>Order of seasonal-differencing.</p> <ul style="list-style-type: none"> • Negative: Automatically identifies seasonal-differencing order Canova-Hansen test • Others: Uses the specified value as the seasonal-differencing order. 	

Name	Data Type	Default Value	Description	Dependency
CH_SIGNIFI-CANCE_LEVEL	Double	0.05	The significance level for Canova-Hansen test. Supported values are 0.01, 0.025, 0.05, 0.1, and 0.2. The smaller it is, the larger probable a time series is considered seasonal-stationary, that is, the less probable it needs seasonal-differencing.	Valid only when SEASONAL_D is negative.
MAX_SEASONAL_D	Integer	1	The maximum value of SEASONAL_D when Canova-Hansen test is applied.	
MAX_P	Integer	5	The maximum value of AR order p.	
MAX_Q	Integer	5	The maximum value of MA order q.	
MAX_SEASONAL_P	Integer	2	The maximum value of AR order P.	
MAX_SEASONAL_Q	Integer	2	The maximum value of MA order Q.	
INFORMATION_CRITERION	Integer	0	The information criterion for order selection. <ul style="list-style-type: none"> • 0: AICC • 1: AIC • 2: BIC 	

Name	Data Type	Default Value	Description	Dependency
SEARCH_STRATEGY	Integer	1	<p>The search strategy for optimal ARMA model.</p> <ul style="list-style-type: none"> • 0: Exhaustive. Traverse all models specified by MAX_P, MAX_Q, MAX_SEASONAL_P, MAX_SEASONAL_Q, and MAX_ORDER. • 1: Stepwise. Changes one or two orders of (p, q, P, Q) by 1 from the current optimal model each time, until no better model is found. This is more time-efficient but possibly less accuracy-effective. 	
MAX_ORDER	integer	15	The maximum value of $(p + q + P + Q)$.	Valid only when SEARCH_STRATEGY is 0.
INITIAL_P	integer	0	Order p of user-defined initial model.	Valid only when SEARCH_STRATEGY is 1.
INITIAL_Q	integer	0	Order q of user-defined initial model.	Valid only when SEARCH_STRATEGY is 1.
INITIAL_SEASONAL_P	integer	0	Order P of user-defined initial model.	Valid only when SEARCH_STRATEGY is 1.
INITIAL_SEASONAL_Q	integer	0	Order Q of user-defined initial model.	Valid only when SEARCH_STRATEGY is 1.

Name	Data Type	Default Value	Description	Dependency
GUESS_STATES	integer	1	If employing ACF/PACF to guess initial ARMA models, besides user-defined model: <ul style="list-style-type: none">• 0: No guess. Besides user-defined model, uses states (2, 2) (1, 1)_m, (1, 0) (1, 0)_m, and (0, 1) (0, 1)_m meanwhile as starting states.• 1: Guesses starting states taking advantage of ACF/PACF.	Valid only when SEARCH_STRATEGY is 1.
MAX_SEARCH_ITERATIONS	integer	(MAX_P+1)* (MAX_Q+1)* (MAX_SEASONAL_P+1)* (MAX_SEASONAL_Q+1)	The maximum iterations for searching optimal ARMA states.	Valid only when SEARCH_STRATEGY is 1.

i Note

In practice, the ARMA model (0,0) (0,0)_m, that is, white noise, is not actually calculated.

Output Table (Model Table)

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Varchar or nvarchar	Name	The minimum length is 50.
	2nd column	Varchar or nvarchar	Value	The table must be a column table. The minimum length of each unit (row) is 1024.

Examples

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and

- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1: Auto ARIMA

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ARIMA_DATA_T;
CREATE TYPE PAL_ARIMA_DATA_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "X1" DOUBLE);
DROP TYPE PAL_ARIMA_CONTROL_T;
CREATE TYPE PAL_ARIMA_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100) );
DROP TYPE PAL_ARIMA_MODEL_T;
CREATE TYPE PAL_ARIMA_MODEL_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" VARCHAR (5000) );

DROP TABLE PAL_ARIMA_PDATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_ARIMA_DATA_T', 'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_ARIMA_CONTROL_T',
'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_ARIMA_MODEL_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_AUTOARIMA_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'AUTOARIMA', 'DM_PAL',
'PAL_AUTOARIMA_PROC', 'PAL_ARIMA_PDATA_TBL');
DROP TABLE PAL_ARIMA_DATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_DATA_TBL LIKE PAL_ARIMA_DATA_T;
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(1, 88);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(2, 84);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(3, 85);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(4, 85);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(5, 84);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(6, 85);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(7, 83);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(8, 85);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(9, 88);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(10, 89);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(11, 91);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(12, 99);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(13, 104);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(14, 112);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(15, 126);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(16, 138);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(17, 146);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(18, 151);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(19, 150);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(20, 148);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(21, 147);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(22, 149);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(23, 143);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(24, 132);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(25, 131);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(26, 139);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(27, 147);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(28, 150);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(29, 148);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(30, 145);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(31, 140);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(32, 134);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(33, 131);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(34, 131);

```

```

INSERT INTO PAL_ARIMA_DATA_TBL VALUES(35, 129);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(36, 126);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(37, 126);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(38, 132);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(39, 137);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(40, 140);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(41, 142);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(42, 150);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(43, 159);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(44, 167);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(45, 170);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(46, 171);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(47, 172);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(48, 172);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(49, 174);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(50, 175);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(51, 172);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(52, 172);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(53, 174);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(54, 174);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(55, 169);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(56, 165);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(57, 156);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(58, 142);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(59, 131);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(60, 121);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(61, 112);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(62, 104);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(63, 102);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(64, 99);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(65, 99);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(66, 95);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(67, 88);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(68, 84);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(69, 84);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(70, 87);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(71, 89);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(72, 88);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(73, 85);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(74, 86);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(75, 89);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(76, 91);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(77, 91);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(78, 94);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(79, 101);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(80, 110);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(81, 121);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(82, 135);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(83, 145);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(84, 149);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(85, 156);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(86, 165);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(87, 171);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(88, 175);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(89, 177);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(90, 182);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(91, 193);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(92, 204);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(93, 208);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(94, 210);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(95, 215);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(96, 222);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(97, 228);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(98, 226);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(99, 222);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(100, 220);
DROP TABLE #PAL_CONTROL_TBL;

```

```

CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR (50), "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEARCH_STRATEGY', 1, null, null);
DROP TABLE PAL_ARIMA_MODEL_TBL;
CREATE COLUMN TABLE PAL_ARIMA_MODEL_TBL LIKE PAL_ARIMA_MODEL_T;
CALL DM_PAL.PAL_AUTOARIMA_PROC(PAL_ARIMA_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_ARIMA_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ARIMA_MODEL_TBL;

```

Expected Result

	NAME	VALUE
1	p	3
2	ARParameters	1.14608;-0.659098;0.334361;
3	d	1
4	q	0
5	MAParamet...	
6	Intercept	0.977855
7	Sigma2	9.33618
8	logLikelihood	-251.832
9	SeriesData	220;
10	DeltaSeriesD...	-2;-4;-2;
11	Eps	
12	FORECASTI...	1
13	AllDeltaSeri...	-4;1;0;-1;1;-2;2;3;1;2;8;5;8;...
14	AIC	511.664
15	AICC	512.09
16	BIC	522.045

Example 2: Auto ARIMAX

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ARIMA_DATA_T;
CREATE TYPE PAL_ARIMA_DATA_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "X1" DOUBLE,
    "Xreg" DOUBLE);
DROP TYPE PAL_ARIMA_CONTROL_T;
CREATE TYPE PAL_ARIMA_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100) );
DROP TYPE PAL_ARIMA_MODEL_T;
CREATE TYPE PAL_ARIMA_MODEL_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" VARCHAR (5000) );

DROP TABLE PAL_ARIMA_PDATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_ARIMA_DATA_T', 'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_ARIMA_CONTROL_T',
'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_ARIMA_MODEL_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_AUTOARIMA_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'AUTOARIMA', 'DM_PAL',
'PAL_AUTOARIMA_PROC', 'PAL_ARIMA_PDATA_TBL');

```

```

DROP TABLE PAL_ARIMA_DATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_DATA_TBL LIKE PAL_ARIMA_DATA_T;
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(1, 1.2, 0.8);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(2, 1.34845613096197, 1.2);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(3, 1.32261090809898, 1.34845613096197);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(4, 1.38095306748554, 1.32261090809898);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(5, 1.54066648969168, 1.38095306748554);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(6, 1.50920806756785, 1.54066648969168);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(7, 1.48461408893443, 1.50920806756785);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(8, 1.43784887380224, 1.48461408893443);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(9, 1.64251548718992, 1.43784887380224);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(10, 1.74292337447476, 1.64251548718992);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(11, 1.91137546943257, 1.74292337447476);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(12, 2.07735796176367, 1.91137546943257);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(13, 2.01741246166924, 2.07735796176367);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(14, 1.87176938196573, 2.01741246166924);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(15, 1.83354723357744, 1.87176938196573);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(16, 1.66104978144571, 1.83354723357744);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(17, 1.65115984070812, 1.66104978144571);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(18, 1.69470966154593, 1.65115984070812);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(19, 1.70459802935728, 1.69470966154593);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(20, 1.61246059980916, 1.70459802935728);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(21, 1.53949706614636, 1.61246059980916);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(22, 1.59231354902055, 1.53949706614636);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(23, 1.81741927705578, 1.59231354902055);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(24, 1.80224252773564, 1.81741927705578);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(25, 1.81881576781466, 1.80224252773564);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(26, 1.78089755157948, 1.81881576781466);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(27, 1.61473635574416, 1.78089755157948);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(28, 1.42002147867225, 1.61473635574416);
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(29, 1.49971641345022, 1.42002147867225);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR (50), "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEARCH_STRATEGY', 1, null, null);
DROP TABLE PAL_ARIMA_MODEL_TBL;
CREATE COLUMN TABLE PAL_ARIMA_MODEL_TBL LIKE PAL_ARIMA_MODEL_T;
CALL DM_PAL.PAL_AUTOARIMA_PROC(PAL_ARIMA_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_ARIMA_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ARIMA_MODEL_TBL;

```

Expected Result

	NAME	VALUE
1	p	2
2	ARParameters	1.33564;-0.60283;
3	d	0
4	q	1
5	MAParamet...	-0.999728;
6	Intercept	0.656911
7	Sigma2	0.00688968
8	logLikelihood	29.5906
9	SeriesData	
10	DeltaSeriesD...	0.423875;0.623...
11	Eps	0.0504612;
12	RegCoeffNa...	Xreg;
13	RegCoeffVal...	0.616909;
14	FORECASTI...	1
15	AllDeltaSeri...	0.706472;0.608...
16	AIC	-51.1813
17	AICC	-49.5147
18	BIC	-45.7121

Example 3: Auto Seasonal ARIMA

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ARIMA_DATA_T;
CREATE TYPE PAL_ARIMA_DATA_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "X1" DOUBLE);
DROP TYPE PAL_ARIMA_CONTROL_T;
CREATE TYPE PAL_ARIMA_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100) );
DROP TYPE PAL_ARIMA_MODEL_T;
CREATE TYPE PAL_ARIMA_MODEL_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" VARCHAR (5000) );

DROP TABLE PAL_ARIMA_PDATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_ARIMA_DATA_T', 'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_ARIMA_CONTROL_T',
'IN');
INSERT INTO PAL_ARIMA_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_ARIMA_MODEL_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_AUTOARIMA_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'AUTOARIMA', 'DM_PAL',
'PAL_AUTOARIMA_PROC', 'PAL_ARIMA_PDATA_TBL');
DROP TABLE PAL_ARIMA_DATA_TBL;
CREATE COLUMN TABLE PAL_ARIMA_DATA_TBL LIKE PAL_ARIMA_DATA_T;
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(1, -24.525 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(2, 34.72 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(3, 57.325 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(4, 10.34 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(5, -12.89 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(6, 39.045 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(7, 57.3 );

```

```

INSERT INTO PAL_ARIMA_DATA_TBL VALUES(8 , 6.735 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(9 , -19.365 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(10 , 34.085 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(11 , 52.455 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(12 , 8.445 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(13 , -13.595 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(14 , 36.73 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(15 , 54.81 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(16 , 4.625 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(17 , -15.595 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(18 , 36.61 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(19 , 58.51 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(20 , 6.725 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(21 , -9.815 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(22 , 38.65 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(23 , 55.5 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(24 , 12.415 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(25 , -17.28 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(26 , 36.105 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(27 , 50.43 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(28 , 7.17 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(29 , -18.97 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(30 , 39.775 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(31 , 57.825 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(32 , 0.49 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(33 , -19.475 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(34 , 31.53 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(35 , 50.025 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(36 , 6.47 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(37 , -20.585 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(38 , 36.94 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(39 , 54.37 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(40 , 10.705 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(41 , -15.965 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(42 , 33.415 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(43 , 55.41 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(44 , -0.62 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(45 , -24.52 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(46 , 37.345 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(47 , 59.44 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(48 , 3.91 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(49 , -19.305 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(50 , 39.525 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(51 , 55.545 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(52 , 3.96 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(53 , -21.69 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(54 , 33.255 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(55 , 57.795 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(56 , 7.535 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(57 , -21.865 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(58 , 36.89 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(59 , 52.17 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(60 , 0.94 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(61 , -23.82 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(62 , 35.025 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(63 , 50.5 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(64 , 4.29 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(65 , -15.27 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(66 , 36.335 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(67 , 51.435 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(68 , 3.365 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(69 , -25.535 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(70 , 33.425 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(71 , 52.785 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(72 , 3.95 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(73 , -17.735 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(74 , 31.95 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(75 , 53.555 );

```

```

INSERT INTO PAL_ARIMA_DATA_TBL VALUES(76 , 6.745 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(77 , -20 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(78 , 31.355 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(79 , 54.71 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(80 , 3.835 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(81 , -23.145 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(82 , 35.23 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(83 , 55.1 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(84 , 3.73 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(85 , -17.605 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(86 , 33.9 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(87 , 55.42 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(88 , 3.505 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(89 , -23.895 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(90 , 34.445 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(91 , 55.635 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(92 , 3.595 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(93 , -21.8 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(94 , 33.45 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(95 , 56.54 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(96 , 5.775 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(97 , -21.27 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(98 , 32.14 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(99 , 53.46 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(100 , -1.935 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(101 , -14.275 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(102 , 35.485 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(103 , 57.305 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(104 , 7.325 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(105 , -21.545 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(106 , 34.11 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(107 , 53.885 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(108 , 1.625 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(109 , -15.75 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(110 , 36.58 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(111 , 54.89 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(112 , 3.8 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(113 , -18.035 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(114 , 37.495 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(115 , 51.875 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(116 , -6.58 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(117 , -18.17 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(118 , 33.565 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(119 , 52.605 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(120 , 3.665 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(121 , -26.47 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(122 , 31.495 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(123 , 52.375 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(124 , 2.135 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(125 , -19.87 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(126 , 36.46 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(127 , 53.11 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(128 , 6.64 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(129 , -21.835 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(130 , 34.625 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(131 , 54.61 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(132 , -5.11 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(133 , -17.395 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(134 , 36.685 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(135 , 52.68 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(136 , 1.105 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(137 , -25.42 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(138 , 32.595 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(139 , 51.615 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(140 , 8.525 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(141 , -15.215 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(142 , 34.455 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(143 , 56.165 );

```

```

INSERT INTO PAL_ARIMA_DATA_TBL VALUES(144 , 2.335 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(145 , -19.575 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(146 , 34.445 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(147 , 53.48 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(148 , 7.025 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(149 , -19.655 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(150 , 32.355 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(151 , 53.41 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(152 , 1.64 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(153 , -17.325 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(154 , 37.505 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(155 , 50.875 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(156 , 1.555 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(157 , -16.195 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(158 , 38.485 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(159 , 53.795 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(160 , 2.23 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(161 , -17.235 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(162 , 36.255 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(163 , 55.895 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(164 , 2.025 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(165 , -19.19 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(166 , 35.215 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(167 , 54.215 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(168 , -5.73 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(169 , -9.995 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(170 , 35.27 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(171 , 54.665 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(172 , 7.615 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(173 , -14.625 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(174 , 40.195 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(175 , 54.13 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(176 , 3.225 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(177 , -19.245 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(178 , 32.695 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(179 , 51.16 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(180 , 1.265 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(181 , -22.03 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(182 , 35.295 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(183 , 55.21 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(184 , 1.97 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(185 , -27.47 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(186 , 35.83 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(187 , 53.405 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(188 , 6.855 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(189 , -15.515 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(190 , 31.675 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(191 , 54.205 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(192 , 1.505 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(193 , -23.06 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(194 , 30.915 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(195 , 54.07 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(196 , 4.785 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(197 , -18.9 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(198 , 31.5 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(199 , 52.565 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(200 , 2.895 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(201 , -10.22 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(202 , 39.195 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(203 , 56.27 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(204 , 9.155 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(205 , -17.245 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(206 , 37.905 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(207 , 59.035 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(208 , 7.17 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(209 , -15.815 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(210 , 31.77 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(211 , 51.58 );

```

```

INSERT INTO PAL_ARIMA_DATA_TBL VALUES(212 , 1.74 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(213 , -18.805 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(214 , 35.875 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(215 , 56.17 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(216 , 14.525 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(217 , -10.39 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(218 , 30.98 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(219 , 58.91 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(220 , 1.465 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(221 , -25.78 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(222 , 29.75 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(223 , 56.385 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(224 , 7.5 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(225 , -22.755 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(226 , 31.735 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(227 , 53.655 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(228 , 4.825 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(229 , -20.685 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(230 , 35.48 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(231 , 58.655 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(232 , 7.605 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(233 , -11.89 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(234 , 36.8 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(235 , 55.04 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(236 , 12.16 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(237 , -19.905 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(238 , 34.95 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(239 , 55.69 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(240 , 7.225 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(241 , -15.38 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(242 , 35.365 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(243 , 54.855 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(244 , 5.235 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(245 , -19.81 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(246 , 33.12 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(247 , 53.27 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(248 , 6.525 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(249 , -8.875 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(250 , 39.43 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(251 , 58.28 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(252 , 5.665 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(253 , -19.31 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(254 , 34.25 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(255 , 58.675 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(256 , 12.91 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(257 , -7 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(258 , 38.5 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(259 , 58.895 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(260 , 8.65 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(261 , -13.97 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(262 , 35.015 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(263 , 56.47 );
INSERT INTO PAL_ARIMA_DATA_TBL VALUES(264 , 3.535 );
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR (50), "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEARCH_STRATEGY', 0,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ORDER', 8,null,null);
DROP TABLE PAL_ARIMA_MODEL_TBL;
CREATE COLUMN TABLE PAL_ARIMA_MODEL_TBL LIKE PAL_ARIMA_MODEL_T;
CALL DM_PAL.PAL_AUTOARIMA_PROC(PAL_ARIMA_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_ARIMA_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ARIMA_MODEL_TBL;

```

Expected Result

	NAME	VALUE
1	p	1
2	ARParameters	0.255404;
3	d	0
4	q	0
5	MAParamet...	
6	Intercept	0.0151056
7	Sigma2	10.8736
8	logLikelihood	-682.36
9	SeriesData	-13.97;35.015;56.47;3.5...
10	DeltaSeriesD...	-4.26;-6.97;-3.48499;-2...
11	Eps	7.05249;0.709877;2.165...
12	Period	4
13	SeasonalP	1
14	SARParamet...	-0.748652;
15	SeasonalD	1
16	SeasonalQ	2
17	SMAParamete...	-0.24905;-0.534333;
18	FORECASTI...	1
19	AllDeltaSeri...	11.6349;4.325;-0.025;-...
20	AllDeltaSeri...	.69;-0.08;-2.09;-0.07;-5...
21	AIC	1374.72
22	AICC	1374.95
23	BIC	1392.52

Related Information

[ARIMA \[page 314\]](#)

3.5.3 Brown Exponential Smoothing

The brown exponential smoothing model is suitable to model the time series with trend but without seasonality. In PAL, both non-adaptive and adaptive brown linear exponential smoothing are provided.

For non-adaptive brown exponential smoothing, let S_t and T_t be the simply smoothed value and doubly smoothed value for the $(t+1)$ -th time period, respectively. Let a_t and b_t be the intercept and the slope. The procedure is as follows:

1. Initialization:

$$S_0 = x_0$$

$$T_0 = x_0$$

$$a_0 = 2S_0 - T_0$$

$$b_0 = \frac{\alpha}{1-\alpha} \times (S_0 - T_0)$$

$$F_1 = a_0 + b_0$$

2. Calculation:

$$S_t = \alpha x_t + (1 - \alpha) S_{t-1}$$

$$T_t = \alpha S_t + (1 - \alpha) T_{t-1}$$

$$a_t = 2S_t - T_t$$

$$b_t = \frac{\alpha}{1-\alpha} \times (S_t - T_t)$$

$$F_{t+1} = a_t + b_t$$

For adaptive brown exponential smoothing, you need to update the parameter of α for every forecasting. The following rules must be satisfied.

1. Initialization:

$$S_0 = x_0$$

$$T_0 = x_0$$

$$a_0 = 2S_0 - T_0$$

$$b_0 = \frac{\alpha}{1-\alpha} \times (S_0 - T_0)$$

$$F_1 = a_0 + b_0$$

$$A_0 = M_0 = 0$$

$$\alpha_1 = \alpha_2 = \alpha_3 = \delta = 0.2$$

2. Calculation:

$$E_t = x_t - F_t$$

$$A_t = \delta E_t + (1 - \delta) A_{t-1}$$

$$M_t = \delta |E_t| + (1 - \delta) M_{t-1}$$

$$\alpha_t = \lfloor \frac{A_t}{M_t} \rfloor \quad (t \geq 4)$$

$$S_t = \alpha_t x_t + (1 - \alpha_t) S_{t-1}$$

$$T_t = \alpha_t S_t + (1 - \alpha_t) T_{t-1}$$

$$a_t = 2S_t - T_t$$

$$b_t = \frac{\alpha}{1-\alpha} \times (S_t - T_t)$$

$$F_{t+1} = a_t + b_t$$

Where $\alpha, \delta \in (0,1)$ are two user specified parameters. The model can be viewed as two coupled single exponential smoothing models, and thus forecast can be made by the following equation:

$$F_{T+m} = a_T + m b_T$$

i Note

F_0 is not defined because there is no estimation for the time slot 0. According to the definition, you can get $F_1 = a_0 + b_0$ and so on.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

BROWNEXPSSMOOTH

This is a brown exponential smoothing function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'BROWNEXPSSMOOTH',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT
4	<schema_name>	<Statistics OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output table>, <statistics output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	ID
	2nd column	Integer or double	Raw data

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
ALPHA	Double	0.1 (Brown exponential smoothing) 0.2 (Adaptive brown exponential smoothing)	The smoothing constant alpha for brown exponential smoothing or the initialization value for adaptive brown exponential smoothing ($0 < \alpha < 1$).	
DELTA	Double	0.2	Value of weighted for A_t and M_t .	Only valid when ADAPTIVE_METHOD is 1.
FORECAST_NUM	Integer	0	Number of values to be forecast.	
ADAPTIVE_METHOD	Integer	0	<ul style="list-style-type: none"> 0: Brown exponential smoothing 1: Adaptive brown exponential smoothing 	
MEASURE_NAME	Varchar	No default value	<p>Measure name. Supported measures are MPE, MSE, RMSE, ET, MAD, MASE, WMAPE, SMAPE, and MAPE.</p> <p>For detailed information on measures, see <i>Forecast Accuracy Measures</i>.</p>	
IGNORE_ZERO	Integer	0	<ul style="list-style-type: none"> 0: Uses zero values in the input dataset when calculating MPE or MAPE. 1: Ignores zero values in the input dataset when calculating MPE or MAPE. 	Only valid when MEASURE_NAME is MPE or MAPE.
EXPOST_FLAG	Integer	1	<ul style="list-style-type: none"> 0: Does not output the expost forecast, and just outputs the forecast values. 1: Outputs the expost forecast and the forecast values. 	

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	2nd column	Integer or double	Output result
Statistics	1st column	Varchar or nvarchar	Name of statistics
	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_BROWNSMOOTH_DATA_T;
CREATE TYPE PAL_BROWNSMOOTH_DATA_T AS TABLE("ID" INT, "RAWDATA" DOUBLE);
DROP TYPE PAL_BROWNSMOOTH_RESULT_T;
CREATE TYPE PAL_BROWNSMOOTH_RESULT_T AS TABLE("TIME" INT, "OUTPUT" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_BROWNSMOOTH_STATISTIC_T;
CREATE TYPE PAL_BROWNSMOOTH_STATISTIC_T AS TABLE("NAME" VARCHAR(100), "VALUE"
DOUBLE);
DROP TABLE PAL_BROWNSMOOTH_PDATA_TBL;
CREATE COLUMN TABLE PAL_BROWNSMOOTH_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_BROWNSMOOTH_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_BROWNSMOOTH_DATA_T','IN');
INSERT INTO PAL_BROWNSMOOTH_PDATA_TBL VALUES (2,'DM_PAL', 'PAL_CONTROL_T','IN');
INSERT INTO PAL_BROWNSMOOTH_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_BROWNSMOOTH_RESULT_T','OUT');
INSERT INTO PAL_BROWNSMOOTH_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_BROWNSMOOTH_STATISTIC_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'BROWNSMOOTH_TEST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'BROWNEXPSMOOTH', 'DM_PAL',
'BROWNSMOOTH_TEST_PROC',PAL_BROWNSMOOTH_PDATA_TBL);
DROP TABLE PAL_BROWNSMOOTH_DATA_TBL;
CREATE COLUMN TABLE PAL_BROWNSMOOTH_DATA_TBL LIKE PAL_BROWNSMOOTH_DATA_T;
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (1,143.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (2,152.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (3,161.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (4,139.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (5,137.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (6,174.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (7,142.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (8,141.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (9,162.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (10,180.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (11,164.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (12,171.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (13,206.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (14,193.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (15,207.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (16,218.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (17,229.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (18,225.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (19,204.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (20,227.0);

```

```

INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (21,223.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (22,242.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (23,239.0);
INSERT INTO PAL_BROWNSMOOTH_DATA_TBL VALUES (24,266.0);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL, 0.1, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('DELTA',NULL, 0.2, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM',6, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXPOST_FLAG',1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ADAPTIVE_METHOD',0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'MSE');
DROP TABLE PAL_BROWNSMOOTH_RESULT_TBL;
CREATE COLUMN TABLE PAL_BROWNSMOOTH_RESULT_TBL  LIKE PAL_BROWNSMOOTH_RESULT_T ;
DROP TABLE PAL_BROWNSMOOTH_STATISTIC_TBL;
CREATE COLUMN TABLE PAL_BROWNSMOOTH_STATISTIC_TBL  LIKE
PAL_BROWNSMOOTH_STATISTIC_T;
CALL DM_PAL.BROWNSMOOTH_TEST_PROC(PAL_BROWNSMOOTH_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_BROWNSMOOTH_RESULT_TBL, PAL_BROWNSMOOTH_STATISTIC_TBL)WITH OVERVIEW;
SELECT * FROM PAL_BROWNSMOOTH_RESULT_TBL;
SELECT * FROM PAL_BROWNSMOOTH_STATISTIC_TBL;

```

Expected Result

PAL_BROWNSMOOTH_RESULT_TBL:

TIME	OUTPUT
2	143
3	144.799...
4	148.13
5	146.555...
6	144.805...
7	150.709...
8	149.324...
9	147.929...
10	150.930...
11	157.071...
12	159.075...
13	162.147...
14	171.725...
15	177.225...
16	184.638...
17	193.066...
18	202.342...
19	209.322...
20	210.933...
21	216.768...
22	220.797...
23	227.882...
24	233.163...
25	242.898...
26	246.395...
27	249.891...
28	253.388...
29	256.885...
30	260.381...

PAL_BROWNSMOOTH_STATISTIC_TBL:

NAME	VALUE
MSE	474.14200407762...

Related Information

[Forecast Accuracy Measures \[page 358\]](#)

3.5.4 Croston's Method

The Croston's method is a forecast strategy for products with intermittent demand. The Croston's method consists of two steps. First, separate exponential smoothing estimates are made of the average size of a demand. Second, the average interval between demands is calculated. This is then used in a form of the constant model to predict the future demand.

Initialization

The system checks the first time bucket of the historical values. If it finds a value (not zero), it is set as the Z's initial value and X is set to 1. Otherwise, Z is set to 1 and X to 2.

```
V(t) = Historical value  
P(t) = Forecasted value  
q = Interval between last two periods with demand  
α = Smoothing factor for the estimates  
Z = Estimate of demand volume  
X = Estimate of intervals between demand
```

```
If 1st value ≠ 0  
Z(0) = V(1), X(0) = 1  
If 1st value = 0  
Z(0) = 1, X(0) = 2
```

Calculation of Forecast Parameters

The forecast is made using a modified constant model. The forecast parameters P and X are determined as follows:

```
If V(t) = 0  
q = q + 1  
Else  
Z(t) = Z(t-1) + α[V(t) - Z(t-1)]  
X(t) = X(t-1) + α[q - X(t-1)]  
Endif
```

In the last iteration, the parameters Z(f) and X(f) will be delivered for the forecast.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

CROSTON

This is a Croston's method function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'CROSTON', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT
4	<schema_name>	<Statistics OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output table>, <statistics output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	ID
	2nd column	Integer or double	Raw data

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
RAW_DATA_COL	Integer	1	Column number of the column that contains the raw data.	
ALPHA	Double	0.1	Value of the smoothing constant alpha (0 < a < 1).	

Name	Data Type	Default Value	Description	Dependency
FORECAST_NUM	Integer	0	Number of values to be forecast. When it is set to 1, the algorithm only forecasts one value.	
METHOD	Integer	0	<ul style="list-style-type: none"> • 0: Uses the sporadic method • 1: Uses the constant method 	
MEASURE_NAME	Varchar	No default value	<p>Measure name. Supported measures are MPE, MSE, RMSE, ET, MAD, MASE, WMAPE, SMAPE, and MAPE.</p> <p>For detailed information on measures, see <i>Forecast Accuracy Measures</i>.</p>	
EXPOST_FLAG	Integer	1	<ul style="list-style-type: none"> • 0: Does not output the expost forecast, and just outputs the forecast values. • 1: Outputs the expost forecast and the forecast values. 	
IGNORE_ZERO	Integer	0	<ul style="list-style-type: none"> • 0: Uses zero values in the input dataset when calculating MPE or MAPE. • 1: Ignores zero values in the input dataset when calculating MPE or MAPE. 	Only valid when MEASURE_NAME is MPE or MAPE.

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	2nd column	Integer or double	Output result
Statistics	1st column	Varchar or nvarchar	Name of statistics

Table	Column	Column Data Type	Description
	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_CROSTON_DATA_T;
CREATE TYPE PAL_CROSTON_DATA_T AS TABLE("ID" INT, "RAWDATA" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_CROSTON_RESULT_T;
CREATE TYPE PAL_CROSTON_RESULT_T AS TABLE("TIME" INT, "OUTPUT" DOUBLE);
DROP TYPE PAL_CROSTON_STATISTICS_T;
CREATE TYPE PAL_CROSTON_STATISTICS_T AS TABLE ("NAME" VARCHAR (50), "VALUE"
DOUBLE);
DROP TABLE PAL_CROSTON_PDATA_TBL;
CREATE COLUMN TABLE PAL_CROSTON_PDATA_TBL ("POSITION" INT,"SCHEMA_NAME"
NVARCHAR(256),"TYPE_NAME" NVARCHAR (256),"PARAMETER_TYPE" VARCHAR (7));
INSERT INTO PAL_CROSTON_PDATA_TBL VALUES (1,'DM_PAL','PAL_CROSTON_DATA_T','IN');
INSERT INTO PAL_CROSTON_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_CROSTON_PDATA_TBL VALUES
(3,'DM_PAL','PAL_CROSTON_RESULT_T','OUT');
INSERT INTO PAL_CROSTON_PDATA_TBL VALUES
(4,'DM_PAL','PAL_CROSTON_STATISTICS_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','CROSTON_TEST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','CROSTON', 'DM_PAL',
'CROSTON_TEST_PROC', PAL_CROSTON_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('RAW_DATA_COL',1,NULL,NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL,0.1,NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM',1, NULL,NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('METHOD',0, NULL,NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MEASURE_NAME',NULL,NULL,'MAPE');
DROP TABLE PAL_CROSTON_DATA_TBL;
CREATE COLUMN TABLE PAL_CROSTON_DATA_TBL LIKE PAL_CROSTON_DATA_T ;
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (0, 0.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (1, 1.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (2, 4.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (3, 0.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (4, 0.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (5, 0.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (6, 5.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (7, 3.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (8, 0.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (9, 0.0);
INSERT INTO PAL_CROSTON_DATA_TBL VALUES (10, 0.0);
DROP TABLE PAL_CROSTON_RESULT_TBL;
CREATE COLUMN TABLE PAL_CROSTON_RESULT_TBL LIKE PAL_CROSTON_RESULT_T;
DROP TABLE PAL_CROSTON_STATISTICS_TBL;
CREATE COLUMN TABLE PAL_CROSTON_STATISTICS_TBL LIKE PAL_CROSTON_STATISTICS_T;
CALL DM_PAL.CROSTON_TEST_PROC(PAL_CROSTON_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_CROSTON_RESULT_TBL, PAL_CROSTON_STATISTICS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_CROSTON_RESULT_TBL;

```

```
SELECT * FROM PAL_CROSTON_STATISTICS_TBL;
```

Expected Results

PAL_CROSTON_RESULT_TBL:

ID	RESULT
0	0
1	1
2	1.3
3	0
4	0
5	0
6	1.6700000000000002
7	1.8030000000000002
8	0
9	0
10	0
11	1.8030000000000002

PAL_CROSTON_STATISTICS_TBL:

NAME	VALUE
MAPE	0.1581818181818182

Related Information

[Forecast Accuracy Measures \[page 358\]](#)

3.5.5 Forecast Accuracy Measures

Measures are used to check the accuracy of the forecast made by PAL algorithms. They are calculated based on the difference between the historical values and the forecasted values of the fitted model. The measures supported in PAL are MPE, MSE, RMSE, ET, MAD, MASE, WMAPE, SMAPE, and MAPE.

Prerequisite

The input data is numeric, not categorical.

ACCURACYMEASURES

This is a forecast accuracy measures function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'ACCURACYMEASURES',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Double	Actual data
	2nd column	Double	Forecasted data

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
MEASURE_NAME	Varchar	<p>Specifies measure name:</p> <ul style="list-style-type: none"> • MPE: Mean percentage error • MSE: Mean squared error • RMSE: Root mean squared error • ET: Error total • MAD: Mean absolute deviation • MASE: Out-of-sample mean absolute scaled error • WMAPE: Weighted mean absolute percentage error • SMAPE: Symmetric mean absolute percentage error • MAPE: Mean absolute percentage error

Optional Parameters

None.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Name of accuracy measures
	2nd column	Double	Value of accuracy measures

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_FORECASTACCURACYMEASURES_DATA_T;
CREATE TYPE PAL_FORECASTACCURACYMEASURES_DATA_T AS TABLE (
    "ACTUALCOL" DOUBLE,
    "FORECASTCOL" DOUBLE
);

DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE (
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_FORECASTACCURACYMEASURES_RESULT_T;
CREATE TYPE PAL_FORECASTACCURACYMEASURES_RESULT_T AS TABLE (
    "NAME" VARCHAR(50),
    "VALUE" DOUBLE
);
DROP TABLE PAL_FORECASTACCURACYMEASURES_DATA_TBL;

```

```

CREATE COLUMN TABLE PAL_FORECASTACCURACYMEASURES_DATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (1,'DM_PAL',
'PAL_FORECASTACCURACYMEASURES_DATA_T','IN');
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES(2,'DM_PAL',
'PAL_CONTROL_T','IN');
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES(3,'DM_PAL',
'PAL_FORECASTACCURACYMEASURES_RESULT_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL',
'PAL_FORECASTACCURACYMEASURES_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'ACCURACYMEASURES',
'DM_PAL',
'PAL_FORECASTACCURACYMEASURES_PROC',PAL_FORECASTACCURACYMEASURES_DATA_TBL);
DROP TABLE PAL_FORECASTACCURACYMEASURES_DATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTACCURACYMEASURES_DATA_T
LIKE PAL_FORECASTACCURACYMEASURES_DATA_T;
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (1130, 1270);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2410, 2340);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2210, 2310);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2500, 2340);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2432, 2348);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (1980, 1890);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2045, 2100);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2340, 2231);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2460, 2401);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2350, 2310);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2345, 2340);
INSERT INTO PAL_FORECASTACCURACYMEASURES_DATA_TBL VALUES (2650, 2560);
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'MSE');
INSERT INTO PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'RMSE');
INSERT INTO PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'MPE');
INSERT INTO PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'ET');
INSERT INTO PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'MAD');
INSERT INTO PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'MASE');
INSERT INTO PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'WMAPE');
INSERT INTO PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'SMAPE');
INSERT INTO PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'MAPE');
INSERT INTO PAL_CONTROL_TBL VALUES ('IGNORE_ZERO', 1, NULL, NULL);
DROP TABLE PAL_FORECASTACCURACYMEASURES_RESULT_TBL;
CREATE COLUMN TABLE PAL_FORECASTACCURACYMEASURES_RESULT_TBL
LIKE PAL_FORECASTACCURACYMEASURES_RESULT_T;

CALL
DM_PAL.PAL_FORECASTACCURACYMEASURES_PROC(PAL_FORECASTACCURACYMEASURES_DATA_TBL,
PAL_CONTROL_TBL, PAL_FORECASTACCURACYMEASURES_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_FORECASTACCURACYMEASURES_RESULT_TBL;

```

Expected Result

PAL_FORECASTACCURACYMEASURES_RESULT_TBL:

NAME	VALUE
ET	412
MAD	83.5
MAPE	0.0410631...
MASE	0.2879310...
MPE	0.0083902...
MSE	8,614
RMSE	92.811637...
SMAPE	0.0408764...
WMAPE	0.0373156...

3.5.6 Forecast Smoothing

Forecast smoothing is used to calculate optimal parameters of a set of smoothing functions in PAL, including Single Exponential Smoothing, Double Exponential Smoothing, and Triple Exponential Smoothing.

This function also outputs the forecasting results based on these optimal parameters. This optimization is computed by exploring of the parameter space which includes all possible parameter combinations. The quality assessment is done by comparing historic and forecast values. In PAL, MSE (mean squared error) or MAPE (mean absolute percentage error) is used to evaluate the quality of the parameters.

The parameter optimization is based on global and local search algorithms. The global search algorithm used in this function is simulated annealing, whereas the local search algorithm is Nelder Mead. Those algorithms allow for efficient search processes.

To evaluate the flexibility of the function, a train-and-test scheme is carried out. In other words, a partition of the time series is allowed, of which the former one is used to train the parameters, whereas the latter one is applied to test.

The error of every forecast value is calculated as follows:

$$D_i = (1 + i * \alpha^2) * E$$

Where α is the weight of smoothing and E is the calculated MSE or MAPE value.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

FORECASTSMMOOTHING

This function is used to calculate optimal parameters and output forecast results.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'FORECASTSMMOOTHING',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT PARAMETER table type>	OUT
4	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output parameter table>, <result output table>) with overview;
```

The procedure name is the same as the name specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	Index of timestamp
	2nd column	Integer or double	Raw data

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
FORECAST_MODEL_NAME	Varchar	<p>Name of the statistical model used for calculating the forecast.</p> <ul style="list-style-type: none"> • SESM: Single Exponential Smoothing • DESM: Double Exponential Smoothing • TESM: Triple Exponential Smoothing

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
OPTIMIZER_TIME_BUDGET	Integer	5	Time budget for the whole optimization process. The time unit is second and the value should be larger than zero.	
OPTIMIZER_RANDOM_SEED	Integer	System time	Random seed for simulated annealing and Nelder Mead. The value should be larger than zero.	
THREAD_NUMBER	Integer	1	Number of threads.	
ALPHA	Double	0.1	Weight for smoothing. Value range: $0 < \alpha < 1$	
BETA	Double	0.1	Weight for the trend component. Value range: $0 < \beta < 1$	Only valid when FORECAST_MODEL_NAME is DESM or TESM.
GAMMA	Double	0.1	Weight for the seasonal component. Value range: $0 < \gamma < 1$	Only valid when FORECAST_MODEL_NAME is TESM.
FORECAST_NUM	Integer	0	Number of values to be forecast.	

Name	Data Type	Default Value	Description	Dependency
CYCLE	Integer	2	Length of a cycle ($L > 1$). For example, the cycle of quarterly data is 4, and the cycle of monthly data is 12.	Only valid when FORECAST_MODEL_NAME is TESM.
MODELSELECTION	Integer	0	<p>When this is set to 1, the algorithm will select the best model among Single/Double/Triple Exponential Smoothing models. The FORECAST_MODEL_NAME parameter will be ignored.</p> <p>When this is set to 0, the algorithm will not perform the model selection.</p> <p>Note: when model selection is enabled, set the CYCLE parameter if you do not want to use the default value of 2.</p>	
FORECAST_AUTOMATIC	Integer	1	<p>The value should be 0 or 1.</p> <ul style="list-style-type: none"> • 1: The optimal parameters will be computed. • 0: The parameter values will be set by the user. 	
SEASONAL	Integer	0	<ul style="list-style-type: none"> • 0: Multiplicative triple exponential smoothing • 1: Additive triple exponential smoothing 	Only valid when FORECAST_MODEL_NAME is TESM.

Name	Data Type	Default Value	Description	Dependency
INITIAL_METHOD	Integer	0	Initialization method for the trend and seasonal components. Refer to <i>Triple Exponential Smoothing</i> for detailed information on initialization method.	Only valid when FORECAST_MODEL_NAME is TESM.
GETPERIODS_AUTOMATIC	Integer	0	Specifies whether the value of CYCLE is automatically obtained. <ul style="list-style-type: none"> • 0: The value of CYCLE is set by user. • 1: The optimal value of CYCLE will be computed automatically. 	Only valid when FORECAST_MODEL_NAME is TESM.
MAX_ITERATION	Integer	100	Maximum number of iterations.	
TRAINING_RATIO	Double	1.0	The ratio of training data to the whole time series. Assuming the size of time series is N, and the training ratio is r, the first $N \cdot r$ time series is used to train, whereas only the latter $N \cdot (1-r)$ one is used to test.	If this parameter is set to 0.0 or 1.0, or the resulting training data ($N \cdot r$) is less than 1 or equal to the size of time series, no train-and-test procedure is carried out.
OPTIMIZATION_METHOD	Integer	0	<ul style="list-style-type: none"> • 0: Heuristic optimization method (simulated annealing and Nelder-mead) • 1: LBFGS-B optimization method 	

Name	Data Type	Default Value	Description	Dependency
DAMPED	Integer	0	<p>For DESM:</p> <ul style="list-style-type: none"> • 0: Uses the Holt's linear method. • 1: Uses the additive damped trend Holt's linear method. <p>For TESM:</p> <ul style="list-style-type: none"> • 0: Uses the Holt Winter method. • 1: Uses the additive damped seasonal Holt Winter method. 	
ACCURACY_MEASURE	Varchar	MSE	The criterion used for the optimization. Available values are MSE and MAPE.	

i Note

Cycle determines the seasonality within the time series data by considering the seasonal factor of a data point_{t-CYCLE+1} in the forecast calculation of data point_{t+1}. Additionally, the algorithm of TESM takes an entire CYCLE as the base to calculate the first forecast value for data point_{CYCLE+1}. The value range for CYCLE should be $2 \leq CYCLE \leq \text{total number of data points}/2$.

For example, there is one year of weekly data (52 data points) as input time series. The value for CYCLE should be within the range of $2 \leq CYCLE \leq 26$. If CYCLE is 4 then we get the first forecast value for data point 5 (e.g. week 201205) by considering the seasonal factor of data point 1 (e.g. week 201201). The second forecast value for data point 6 (e.g. week 201206) takes into account of the seasonal factor of data point 2 (e.g. week 201202), etc. If CYCLE is 2 then we get the first forecast value for data point 3 (e.g. week 201203) by considering the seasonal factor of data point 1 (e.g. week 201201). The second forecast value for data point 4 (e.g. week 201204) takes into account of the seasonal factor of data point 2 (e.g. week 201202), etc.

Output Tables

Table	Column	Column Data Type	Description
Output Parameter	1st column	Varchar or nvarchar	<p>Name of related statistics.</p> <ul style="list-style-type: none"> MSE: the statistics of MSE (Mean Squared Error) NUMBER_OF_ITERATIONS: the sum of numbers of iterations of both optimizers (simulated annealing and Nelder Mead) ALPHA/BETA/GAMMA: the optimal parameters or values set by user (For train-and-test scheme only) NUMBER_OF_TRAINING: the size of time series used to train (For train-and-test scheme only) NUMBER_OF_TESTING: the size of time series used to test (For train-and-test scheme only) TEST_MSE: the statistics of MSE calculated from the test data
	2nd column	Double, varchar, or nvarchar	<p>Parameter values.</p> <p>Note: If you set the MODEL-SELECTION parameter to 1, this column data type should be varchar or nvarchar to output the best model name.</p>
Result	1st column	Integer	Index of timestamp
	2nd column	Integer or double	Result of smoothing forecast
	3rd column	Double	Error of every forecast value

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

For Single Exponential Smoothing:

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_FORECASTSINGLESMOOTHING_DATA_T;
CREATE TYPE PAL_FORECASTSINGLESMOOTHING_DATA_T AS TABLE ("TIMESTAMP" INT,
"VALUE" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE ("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_OUTPARAMETER_T;
CREATE TYPE PAL_OUTPARAMETER_T AS TABLE ("NAME" VARCHAR(100), "VALUE" DOUBLE);
DROP TYPE PAL_FORECASTSINGLESMOOTHING_FORECAST_T;
CREATE TYPE PAL_FORECASTSINGLESMOOTHING_FORECAST_T AS TABLE ("TIMESTAMP" INT,
"VALUE" DOUBLE, "DIFFERENCE" DOUBLE);
DROP TABLE PAL_FORECASTSINGLESMOOTHING_PDATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTSINGLESMOOTHING_PDATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_FORECASTSINGLESMOOTHING_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_FORECASTSINGLESMOOTHING_DATA_T','IN');
INSERT INTO PAL_FORECASTSINGLESMOOTHING_PDATA_TBL VALUES(2,'DM_PAL',
'PAL_CONTROL_T','IN');
INSERT INTO PAL_FORECASTSINGLESMOOTHING_PDATA_TBL VALUES(3,'DM_PAL',
'PAL_OUTPARAMETER_T','OUT');
INSERT INTO PAL_FORECASTSINGLESMOOTHING_PDATA_TBL VALUES(4,'DM_PAL',
'PAL_FORECASTSINGLESMOOTHING_FORECAST_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PALFORECASTSMOOTHING_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'FORECASTSMOOTHING',
'DM_PAL', 'PALFORECASTSMOOTHING_PROC', PAL_FORECASTSINGLESMOOTHING_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST MODEL NAME', NULL, NULL,'SESM');
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD NUMBER',8, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL,0.1, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM',2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_AUTOMATIC',1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION',300, NULL, NULL);
DROP TABLE PAL_FORECASTSINGLESMOOTHING_DATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTSINGLESMOOTHING_DATA_TBL LIKE
PAL_FORECASTSINGLESMOOTHING_DATA_T ;
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (1,200.0);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (2,135.0);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (3,195.0);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (4,197.5);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (5,310.0);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (6,175.0);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (7,155.0);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (8,130.0);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (9,220.0);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (10,277.5);
INSERT INTO PAL_FORECASTSINGLESMOOTHING_DATA_TBL VALUES (11,235.0);
DROP TABLE PAL_OUTPARAMETER_TBL;
CREATE COLUMN TABLE PAL_OUTPARAMETER_TBL LIKE PAL_OUTPARAMETER_T ;
DROP TABLE PAL_FORECASTSINGLESMOOTHING_RESULT_TBL;
CREATE COLUMN TABLE PAL_FORECASTSINGLESMOOTHING_RESULT_TBL LIKE
PAL_FORECASTSINGLESMOOTHING_FORECAST_T ;
CALL DM_PAL.PALFORECASTSMOOTHING_PROC( PAL_FORECASTSINGLESMOOTHING_DATA_TBL,
"#PAL_CONTROL_TBL", PAL_OUTPARAMETER_TBL,
PAL_FORECASTSINGLESMOOTHING_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_OUTPARAMETER_TBL;
SELECT * FROM PAL_FORECASTSINGLESMOOTHING_RESULT_TBL;
```

Expected Result

PAL_OUTPARAMETER_TBL:

NAME	VALUE
MSE	3,153.7500003063724
NUMBER_OF_ITERATIONS	311
ALPHA	0.0000000010000017344274...

PAL_FORECASTSINGLESMOOTHING_RESULT_TBL:

TIMESTAMP	VALUE	DIFFERENCE
2	200	3,153.7500...
3	199.9999999935	3,153.7500...
4	199.99999999...	3,153.7500...
5	199.99999999...	3,153.7500...
6	200.00000000...	3,153.7500...
7	200.00000000...	3,153.7500...
8	199.99999999...	3,153.7500...
9	199.99999998...	3,153.7500...
10	199.99999999...	3,153.7500...
11	199.99999999...	3,153.7500...
12	200.00000000...	3,153.7500...
13	200.00000000...	3,153.7500...

For Double Exponential Smoothing:

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_FORECASTDOUBLESMOOTHING_DATA_T;
CREATE TYPE PAL_FORECASTDOUBLESMOOTHING_DATA_T AS TABLE ("TIMESTAMP" INT,
"VALUE" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE ("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_OUTPARAMETER_T;
CREATE TYPE PAL_OUTPARAMETER_T AS TABLE ("NAME" VARCHAR(100), "VALUE" DOUBLE);
DROP TYPE PAL_FORECASTDOUBLESMOOTHING_FORECAST_T;
CREATE TYPE PAL_FORECASTDOUBLESMOOTHING_FORECAST_T AS TABLE ("TIMESTAMP" INT,
"VALUE" DOUBLE, "DIFFERENCE" DOUBLE);
DROP TABLE PAL_FORECASTDOUBLESMOOTHING_PDATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTDOUBLESMOOTHING_PDATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_FORECASTDOUBLESMOOTHING_DATA_T','IN');
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_PDATA_TBL VALUES (2,'DM_PAL',
'PAL_CONTROL_T','IN');
```

```

INSERT INTO PAL_FORECASTDOUBLESMOOTHING_PDATA_TBL VALUES(3,'DM_PAL',
'PAL_OUTPARAMETER_T','OUT');
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_PDATA_TBL VALUES(4,'DM_PAL',
'PAL_FORECASTDOUBLESMOOTHING_FORECAST_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PALFORECASTSMOOTHING_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'FORECASTSMOOTHING',
'DM_PAL', 'PALFORECASTSMOOTHING_PROC', PAL_FORECASTDOUBLESMOOTHING_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_MODEL_NAME', NULL, NULL, 'DESM');
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 8, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL, 0.1, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BETA', NULL, 0.1, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM', 2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_AUTOMATIC', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 300, NULL, NULL);
DROP TABLE PAL_FORECASTDOUBLESMOOTHING_DATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTDOUBLESMOOTHING_DATA_TBL LIKE
PAL_FORECASTDOUBLESMOOTHING_DATA_T ;
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (1,143.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (2,152.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (3,161.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (4,139.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (5,137.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (6,174.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (7,142.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (8,141.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (9,162.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (10,180.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (11,164.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (12,171.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (13,206.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (14,193.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (15,207.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (16,218.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (17,229.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (18,225.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (19,204.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (20,227.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (21,223.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (22,242.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (23,239.0);
INSERT INTO PAL_FORECASTDOUBLESMOOTHING_DATA_TBL VALUES (24,266.0);
DROP TABLE PAL_OUTPARAMETER_TBL;
CREATE COLUMN TABLE PAL_OUTPARAMETER_TBL LIKE PAL_OUTPARAMETER_T ;
DROP TABLE PAL_FORECASTDOUBLESMOOTHING_RESULT_TBL;
CREATE COLUMN TABLE PAL_FORECASTDOUBLESMOOTHING_RESULT_TBL LIKE
PAL_FORECASTDOUBLESMOOTHING_FORECAST_T ;
CALL DM_PAL.PALFORECASTSMOOTHING_PROC( PAL_FORECASTDOUBLESMOOTHING_DATA_TBL,
"#PAL_CONTROL_TBL", PAL_OUTPARAMETER_TBL,
PAL_FORECASTDOUBLESMOOTHING_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_OUTPARAMETER_TBL;
SELECT * FROM PAL_FORECASTDOUBLESMOOTHING_RESULT_TBL;

```

Expected Results

PAL_OUTPARAMETER_TBL:

NAME	VALUE
MSE	274.8958818071007
NUMBER_OF_ITERATIONS	336
ALPHA	0.5008159910719763
BETA	0.07228928248089861

PAL_FORECASTDOUBLESMOOTHING_RESULT_TBL:

TIMESTAMP	VALUE	DIFFERENCE
2	152	274.895881807...
3	161	343.844347881...
4	170	412.792813955...
5	162.352391...	481.741280029...
6	156.615347...	550.689746103...
7	172.911085...	619.638212177...
8	163.900452...	688.586678251...
9	158.072593...	757.535144325...
10	165.822741...	826.483610400...
11	179.219447...	895.432076474...
12	177.342813...	964.380542548...
13	179.682107...	1,033.32900862...
14	199.331207...	1,102.27747469...
15	202.399904...	1,171.22594077...
16	211.109712...	1,240.17440684...
17	221.215938...	1,309.12287291...
18	232.051592...	1,378.07133899...
19	235.202020...	1,447.01980506...
20	225.127901...	1,515.96827114...
21	231.685606...	1,584.91673721...
22	232.641394...	1,653.86520328...
23	242.972827...	1,722.81366936...
24	246.483834...	1,791.76213543...
25	262.465061...	1,860.71060151...
26	268.672279...	1,929.65906758...

For Triple Exponential Smoothing:

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_FORECASTTRIPLESMOOTHING_DATA_T;
CREATE TYPE PAL_FORECASTTRIPLESMOOTHING_DATA_T AS TABLE ("TIMESTAMP" INT,
"VALUE" DOUBLE);
DROP TYPE PAL_CONTROL_T;
```

```

CREATE TYPE PAL_CONTROL_T AS TABLE ("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_OUTPARAMETER_T;
CREATE TYPE PAL_OUTPARAMETER_T AS TABLE ("NAME" VARCHAR(100), "VALUE" DOUBLE);
DROP TYPE PAL_FORECASTTRIPLESMOOTHING_FORECAST_T;
CREATE TYPE PAL_FORECASTTRIPLESMOOTHING_FORECAST_T AS TABLE ("TIMESTAMP" INT,
"VALUE" DOUBLE, "DIFFERENCE" DOUBLE);
DROP TABLE PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_FORECASTTRIPLESMOOTHING_DATA_T','IN');
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL VALUES (2,'DM_PAL',
'PAL_CONTROL_T','IN');
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_OUTPARAMETER_T','OUT');
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_FORECASTTRIPLESMOOTHING_FORECAST_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PALFORECASTSMOOTHING_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'FORECASTSMOOTHING',
'DM_PAL', 'PALFORECASTSMOOTHING_PROC', PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_MODEL_NAME', NULL, NULL, 'TESM');
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 8, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL, 0.4, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BETA', NULL, 0.4, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('GAMMA', NULL, 0.4, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CYCLE', 4, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM', 3, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_AUTOMATIC', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEASONAL', 0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INITIAL_METHOD', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 300, NULL, NULL);
DROP TABLE PAL_FORECASTTRIPLESMOOTHING_DATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTTRIPLESMOOTHING_DATA_TBL LIKE
PAL_FORECASTTRIPLESMOOTHING_DATA_T ;
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (1,362.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (2,385.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (3,432.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (4,341.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (5,382.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (6,409.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (7,498.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (8,387.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (9,473.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (10,513.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (11,582.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (12,474.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (13,544.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (14,582.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (15,681.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (16,557.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (17,628.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (18,707.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (19,773.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (20,592.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (21,627.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (22,725.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (23,854.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (24,661.0);
DROP TABLE PAL_OUTPARAMETER_TBL;
CREATE COLUMN TABLE PAL_OUTPARAMETER_TBL LIKE PAL_OUTPARAMETER_T ;
DROP TABLE PAL_FORECASTTRIPLESMOOTHING_RESULT_TBL;
CREATE COLUMN TABLE PAL_FORECASTTRIPLESMOOTHING_RESULT_TBL LIKE
PAL_FORECASTTRIPLESMOOTHING_FORECAST_T ;

```

```
CALL DM_PAL.PALFORECASTSMOOTHING_PROC(PAL_FORECASTTRIPLESMOOTHING_DATA_TBL,  
"#PAL_CONTROL_TBL", PAL_OUTPARAMETER_TBL,  
PAL_FORECASTTRIPLESMOOTHING_RESULT_TBL) WITH OVERVIEW;  
SELECT * FROM PAL_OUTPARAMETER_TBL;  
SELECT * FROM PAL_FORECASTTRIPLESMOOTHING_RESULT_TBL;
```

Expected Results

PAL_OUTPARAMETER_TBL:

NAME	VALUE
MSE	666.5105074278985
NUMBER_OF_ITERATIONS	477
ALPHA	0.630997010670435
BETA	0.07734374169741...
GAMMA	0.00000000010000...

PAL_FORECASTTRIPLESMOOTHING_RESULT_TBL:

TIMESTAMP	VALUE	DIFFERENCE
5	367.95605...	666.51050...
6	400.85305...	931.88648...
7	476.38466...	1,197.2624...
8	392.32203...	1,462.6384...
9	435.14952...	1,728.0144...
10	488.83760...	1,993.3903...
11	592.56741...	2,258.7663...
12	469.01165...	2,524.1423...
13	528.53716...	2,789.5183...
14	572.60799...	3,054.8942...
15	678.73720...	3,320.2702...
16	543.79522...	3,585.6462...
17	617.58885...	3,851.0222...
18	663.23296...	4,116.3981...
19	810.88163...	4,381.7741...
20	628.48270...	4,647.1501...
21	674.72130...	4,912.5261...
22	680.48942...	5,177.9020...
23	826.70210...	5,443.2780...
24	671.84384...	5,708.6540...
25	739.86327...	5,974.0300...
26	781.76469...	6,239.4059...
27	910.87397...	6,504.7819...

For Model Selection:

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_FORECASTMODELSELECTION_DATA_T;
CREATE TYPE PAL_FORECASTMODELSELECTION_DATA_T AS TABLE ("TIMESTAMP" INT, "VALUE"
DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE ("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_OUTPARAMETER_T;
```

```

CREATE TYPE PAL_OUTPARAMETER_T AS TABLE ("NAME" VARCHAR(100), "VALUE"
VARCHAR(100));
DROP TYPE PAL_FORECASTMODELSELECTION_FORECAST_T;
CREATE TYPE PAL_FORECASTMODELSELECTION_FORECAST_T AS TABLE ("TIMESTAMP" INT,
"VALUE" DOUBLE, "DIFFERENCE" DOUBLE);
DROP TABLE PAL_FORECASTMODELSELECTION_PDATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTMODELSELECTION_PDATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_FORECASTMODELSELECTION_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_FORECASTMODELSELECTION_DATA_T','IN');
INSERT INTO PAL_FORECASTMODELSELECTION_PDATA_TBL VALUES (2,'DM_PAL',
'PAL_CONTROL_T','IN');
INSERT INTO PAL_FORECASTMODELSELECTION_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_OUTPARAMETER_T','OUT');
INSERT INTO PAL_FORECASTMODELSELECTION_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_FORECASTMODELSELECTION_FORECAST_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PALFORECASTSMOOTHING_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'FORECASTSMOOTHING',
'DM_PAL', 'PALFORECASTSMOOTHING_PROC', PAL_FORECASTMODELSELECTION_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',8, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL, 0.1, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM',2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CYCLE', 2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_AUTOMATIC',1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MODELSELECTION',1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION',300, NULL, NULL);
DROP TABLE PAL_FORECASTMODELSELECTION_DATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTMODELSELECTION_DATA_TBL LIKE
PAL_FORECASTMODELSELECTION_DATA_T;
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (1,200.0);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (2,135.0);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (3,195.0);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (4,197.5);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (5,310.0);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (6,175.0);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (7,155.0);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (8,130.0);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (9,220.0);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (10,277.5);
INSERT INTO PAL_FORECASTMODELSELECTION_DATA_TBL VALUES (11,235.0);
DROP TABLE PAL_OUTPARAMETER_TBL;
CREATE COLUMN TABLE PAL_OUTPARAMETER_TBL LIKE PAL_OUTPARAMETER_T;
DROP TABLE PAL_FORECASTMODELSELECTION_RESULT_TBL;
CREATE COLUMN TABLE PAL_FORECASTMODELSELECTION_RESULT_TBL LIKE
PAL_FORECASTMODELSELECTION_FORECAST_T;
CALL DM_PAL.PALFORECASTSMOOTHING_PROC( PAL_FORECASTMODELSELECTION_DATA_TBL,
"#PAL_CONTROL_TBL", PAL_OUTPARAMETER_TBL, PAL_FORECASTMODELSELECTION_RESULT_TBL)
WITH OVERVIEW;
SELECT * FROM PAL_OUTPARAMETER_TBL;
SELECT * FROM PAL_FORECASTMODELSELECTION_RESULT_TBL;

```

Expected Result

PAL_OUTPARAMETER_TBL:

NAME	VALUE
FORECAST_MODEL_NAME	SESM
MSE	3153.75
NUMBER_OF_ITERATIONS	311
ALPHA	1e-10

PAL_FORECASTMODELSELECTION_RESULT_TBL:

TIMESTAMP	VALUE	DIFFERENCE
2	200	3,153.7500...
3	199.999999...	3,153.7500...
4	199.999999...	3,153.7500...
5	199.999999...	3,153.7500...
6	200.000000...	3,153.7500...
7	200.000000...	3,153.7500...
8	199.999999...	3,153.7500...
9	199.999999...	3,153.7500...
10	199.999999...	3,153.7500...
11	199.999999...	3,153.7500...
12	200.000000...	3,153.7500...
13	200.000000...	3,153.7500...

For Train-and-Test:

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_FORECASTTRIPLESMOOTHING_DATA_T;
CREATE TYPE PAL_FORECASTTRIPLESMOOTHING_DATA_T AS TABLE ("TIMESTAMP" INT,
"VALUE" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE ("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_OUTPARAMETER_T;
CREATE TYPE PAL_OUTPARAMETER_T AS TABLE ("NAME" VARCHAR(100), "VALUE" DOUBLE);
DROP TYPE PAL_FORECASTTRIPLESMOOTHING_FORECAST_T;
CREATE TYPE PAL_FORECASTTRIPLESMOOTHING_FORECAST_T AS TABLE ("TIMESTAMP" INT,
"VALUE" DOUBLE, "DIFFERENCE" DOUBLE);
DROP TABLE PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_FORECASTTRIPLESMOOTHING_DATA_T','IN');

```

```

INSERT INTO PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL VALUES(2,'DM_PAL',
'PAL_CONTROL_T','IN');
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL VALUES(3,'DM_PAL',
'PAL_OUTPARAMETER_T','OUT');
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL VALUES(4,'DM_PAL',
'PAL_FORECASTTRIPLESMOOTHING_FORECAST_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PALFORECASTSMOOTHING_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'FORECASTSMOOTHING',
'DM_PAL', 'PALFORECASTSMOOTHING_PROC', PAL_FORECASTTRIPLESMOOTHING_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_MODEL_NAME', NULL, NULL, 'TESM');
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 8, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL, 0.4, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BETA', NULL, 0.4, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('GAMMA', NULL, 0.4, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CYCLE', 4, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM', 3, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_AUTOMATIC', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEASONAL', 0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INITIAL_METHOD', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 300, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TRAINING_RATIO', NULL, 0.75, NULL);
DROP TABLE PAL_FORECASTTRIPLESMOOTHING_DATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTTRIPLESMOOTHING_DATA_TBL LIKE
PAL_FORECASTTRIPLESMOOTHING_DATA_T ;
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (1,362.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (2,385.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (3,432.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (4,341.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (5,382.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (6,409.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (7,498.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (8,387.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (9,473.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (10,513.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (11,582.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (12,474.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (13,544.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (14,582.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (15,681.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (16,557.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (17,628.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (18,707.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (19,773.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (20,592.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (21,627.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (22,725.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (23,854.0);
INSERT INTO PAL_FORECASTTRIPLESMOOTHING_DATA_TBL VALUES (24,661.0);
DROP TABLE PAL_OUTPARAMETER_TBL;
CREATE COLUMN TABLE PAL_OUTPARAMETER_TBL LIKE PAL_OUTPARAMETER_T ;
DROP TABLE PAL_FORECASTTRIPLESMOOTHING_RESULT_TBL;
CREATE COLUMN TABLE PAL_FORECASTTRIPLESMOOTHING_RESULT_TBL LIKE
PAL_FORECASTTRIPLESMOOTHING_FORECAST_T ;
CALL DM_PAL.PALFORECASTSMOOTHING_PROC(PAL_FORECASTTRIPLESMOOTHING_DATA_TBL,
"#PAL_CONTROL_TBL", PAL_OUTPARAMETER_TBL,
PAL_FORECASTTRIPLESMOOTHING_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_OUTPARAMETER_TBL;
SELECT * FROM PAL_FORECASTTRIPLESMOOTHING_RESULT_TBL;

```

Expected Result

PAL_OUTPARAMETER_TBL:

	NAME	VALUE
1	MSE	268.90511352548185
2	NUMBER_OF_ITERATIONS	387
3	ALPHA	0.3688045371965867
4	BETA	0.9996846768727286
5	GAMMA	0.33532586333727...
6	NUMBER_OF_TRAINING	18
7	NUMBER_OF_TESTING	6
8	TEST_MSE	11,822.2286930518...

PAL_FORECASTMODELSELECTION_RESULT_TBL:

	TIMESTAMP	VALUE	DIFFERENCE
1	5	367.9560567478853	268.90511352548185
2	6	401.69115849436497	305.4807229827968
3	7	482.4750124884512	342.05633244011176
4	8	400.70287478855596	378.6319418974267
5	9	451.4467642966472	415.20755135474167
6	10	499.24245068028443	451.7831608120566
7	11	612.7163701588196	488.35877026937163
8	12	480.03023179395313	524.9343797266865
9	13	551.7044255607572	561.5099891840015
10	14	578.9195701259222	598.0855986413164
11	15	671.9953640164973	634.6612080986314
12	16	539.771250133854	671.2368175559463
13	17	635.9672650125581	707.8124270132614
14	18	676.3191098518699	744.3880364705763
15	19	815.3559980397869	780.9636459278912
16	20	664.2968165379058	817.5392553852062
17	21	768.928755828481	854.1148648425211
18	22	837.577247291799	890.6904742998361
19	23	975.580115846229	927.266083757151
20	24	788.7238388040668	963.8416932144659
21	25	906.5114765111293	1,000.41730267178...
22	26	981.0263337383763	1,036.99291212909...
23	27	1,135.80423365267...	1,073.568521586411

Related Information

[Triple Exponential Smoothing \[page 400\]](#)

3.5.7 Linear Regression with Damped Trend and Seasonal Adjust

Linear regression with damped trend and seasonal adjust is an approach for forecasting when a time series presents a trend. In PAL, it provides a damped smoothing parameter for smoothing the forecasted values. This dampening parameter avoids the over-casting due to the “indefinitely” increasing or decreasing trend. In addition, if the time series presents seasonality, you can deal with it by providing the length of the periods in order to adjust the forecasting results. On the other hand, it also helps you to detect the seasonality and to determine the periods.

i Note

Occasionally, there is probability that the average, the linear forecast, or the seasonal index is calculated to be 0, which gives rise to the issue of division by zero in the subsequent calculation. To address this, therefore, a tiny value, 1.0e-6 for example, is adopted as the divisor instead of 0. In the Result output table, an indicator named “HandleZero” is given, which represents if the substitution takes place (1) or not (0).

Prerequisites

- No missing or null data in the inputs. The algorithm will issue errors when encountering null values.
- The data is numeric, not categorical.

LRWITHSEASONALADJUST

This is the function for linear regression with damped trend and seasonal adjust.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LRWITHSEASONALADJUST',  
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT
4	<schema_name>	<Statistics OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output table>, <statistics output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	Time stamp
	2nd column	Integer or double	Raw data

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
FORECAST_LENGTH	Integer	1	Length of the final forecast results.	
TREND	Double	1	Damped trend factor. Value range is (0,1].	
AFFECT_FUTURE_ONLY	Integer	1	Specifies whether the damped trend affects the history. <ul style="list-style-type: none">• 0: Affects all.• 1: Affects the future only.	Only valid when TREND is not 1.

Name	Data Type	Default Value	Description	Dependency
SEASONALITY	Integer	0	<p>Specifies whether the data represents seasonality.</p> <ul style="list-style-type: none"> • 0: Non-seasonality. • 1: Seasonality exists and user inputs the value of periods. • 2: Automatically detects seasonality. 	
PERIODS	Integer	No default value	<p>Length of the periods.</p> <p>Only valid when SEASONALITY is 1.</p> <p>If this parameter is not specified, the SEASONALITY value will be changed from 1 to 2, that is, from user-defined to automatically-detected.</p>	
MEASURE_NAME	Varchar	No default value	<p>Measure name. Supported measures are MPE, MSE, RMSE, ET, MAD, MASE, WMAPE, SMAPE, and MAPE.</p> <p>For detailed information on measures, see <i>Forecast Accuracy Measures</i>.</p>	
SEASONAL_HANDLE_METHOD	Integer	0	<p>Method used for calculating the index value in the periods.</p> <ul style="list-style-type: none"> • 0: Average method. • 1: Fitting linear regression. 	Only valid when SEASONALITY is 2.

Name	Data Type	Default Value	Description	Dependency
EXPOST_FLAG	Integer	1	<ul style="list-style-type: none"> • 0: Does not output the expost forecast, and just outputs the forecast values. • 1: Outputs the expost forecast and the forecast values. 	
IGNORE_ZERO	Integer	0	<ul style="list-style-type: none"> • 0: Uses zero values in the input dataset when calculating MPE or MAPE. • 1: Ignores zero values in the input dataset when calculating MPE or MAPE. 	Only valid when MEASURE_NAME is MPE or MAPE.

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Name
	2nd column	Double	Value
Forecast	1st column	Integer	Time stamp
	2nd column	Double	Forecasted value

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_FORECASTSLR_DATA_T;
CREATE TYPE PAL_FORECASTSLR_DATA_T AS TABLE (
    "TIMESTAMP" INTEGER,
    "Y" DOUBLE);
DROP TYPE PAL_FORECASTSLR_CONTROL_T;
```

```

CREATE TYPE PAL_FORECASTSLR_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100) );
DROP TYPE PAL_FORECASTSLR_RESULT_T;
CREATE TYPE PAL_FORECASTSLR_RESULT_T AS TABLE(
    "NAME" VARCHAR (50),
    "VALUE" DOUBLE );
DROP TYPE PAL_FORECASTSLR_FORECAST_T;
CREATE TYPE PAL_FORECASTSLR_FORECAST_T AS TABLE(
    "TIMESTAMP" INTEGER,
    "VALUE" DOUBLE );

DROP TABLE PAL_FORECASTSLR_PDATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTSLR_PDATA_TBL("POSITION" INT,"SCHEMA_NAME"
NVARCHAR(256),"TYPE_NAME" NVARCHAR (256),"PARAMETER_TYPE" VARCHAR (7));
INSERT INTO PAL_FORECASTSLR_PDATA_TBL VALUES
(1,'DM_PAL','PAL_FORECASTSLR_DATA_T','in');
INSERT INTO PAL_FORECASTSLR_PDATA_TBL VALUES
(2,'DM_PAL','PAL_FORECASTSLR_CONTROL_T','in');
INSERT INTO PAL_FORECASTSLR_PDATA_TBL VALUES
(3,'DM_PAL','PAL_FORECASTSLR_RESULT_T','out');
INSERT INTO PAL_FORECASTSLR_PDATA_TBL VALUES
(4,'DM_PAL','PAL_FORECASTSLR_FORECAST_T','out');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_FORECASTSLR_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL','LRWITHSEASONALADJUST',
'DM_PAL','PAL_FORECASTSLR_PROC',PAL_FORECASTSLR_PDATA_TBL);
DROP TABLE PAL_FORECASTSLR_DATA_TBL;
CREATE COLUMN TABLE PAL_FORECASTSLR_DATA_TBL LIKE PAL_FORECASTSLR_DATA_T;
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(1, 5384);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(2, 8081);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(3, 10282);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(4, 9156);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(5, 6118);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(6, 9139);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(7, 12460);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(8, 10717);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(9, 7825);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(10, 9693);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(11, 15177);
INSERT INTO PAL_FORECASTSLR_DATA_TBL VALUES(12, 10990);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR
(50),"INTARGS" INTEGER,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_LENGTH', 10,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TREND', null,0.9,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('AFFECT_FUTURE_ONLY', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEASONALITY', 1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PERIODS', 4,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MEASURE_NAME', null, null, 'MSE');
DROP TABLE PAL_FORECASTSLR_RESULT_TBL;
CREATE COLUMN TABLE PAL_FORECASTSLR_RESULT_TBL LIKE PAL_FORECASTSLR_RESULT_T;
DROP TABLE PAL_FORECASTSLR_FORECAST_TBL;
CREATE COLUMN TABLE PAL_FORECASTSLR_FORECAST_TBL LIKE PAL_FORECASTSLR_FORECAST_T;
CALL DM_PAL.PAL_FORECASTSLR_PROC(PAL_FORECASTSLR_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_FORECASTSLR_RESULT_TBL, PAL_FORECASTSLR_FORECAST_TBL) WITH OVERVIEW;
SELECT * FROM PAL_FORECASTSLR_RESULT_TBL;
SELECT * FROM PAL_FORECASTSLR_FORECAST_TBL;

```

Expected Result

PAL_FORECASTSLR_RESULT_TBL:

	NAME	VALUE
1	Intercept	7,626.072427992673
2	Slope	301.39911364215277
3	Periods	4
4	Index0	0.6721148997583071
5	Index1	0.9359253012467181
6	Index2	1.3186694719271097
7	Index3	1.0732903270678653
8	MSE	332,202.479081552...
9	HandleZero	0

PAL_FORECASTSLR_FORECAST_TBL:

	TIMESTAMP	VALUE
1	1	5,328.171740542723
2	2	7,701.608246960384
3	3	11,248.606331576377
4	4	9,478.94478351929
5	5	6,138.471080754076
6	6	8,829.956471884489
7	7	12,838.389571679569
8	8	10,772.899796555095
9	9	6,948.77042096543
10	10	9,958.304696808595
11	11	14,428.172811782759
12	12	12,066.854809590903
13	13	7,738.812277671501
14	14	11,004.8476754257
15	15	15,794.988952461474
16	16	13,068.084849802683
17	17	8,303.102712446336
18	18	11,712.048670095626
19	19	16,691.75702236078
20	20	13,724.991879185629
21	21	8,673.333666702107
22	22	12,176.043242698564

Related Information

[Forecast Accuracy Measures \[page 358\]](#)

3.5.8 Single Exponential Smoothing

Single Exponential Smoothing model is suitable to model the time series without trend and seasonality. In the model, the smoothed value is the weighted sum of previous smoothed value and previous observed value.

PAL provides two simple exponential smoothing algorithms: single exponential smoothing and adaptive-response-rate simple exponential smoothing. The adaptive-response-rate single exponential smoothing algorithm may have an advantage over single exponential smoothing in that it allows the value of alpha to be modified.

For single exponential smoothing, let s_t be the smoothed value for the t-th time period. Mathematically:

$$s_1 = x_0$$

$$s_t = \alpha x_{t-1} + (1-\alpha) s_{t-1}$$

Where $\alpha \in (0, 1)$ is a user specified parameter. Forecast is made by:

$$s_{T+1} = \alpha x_T + (1-\alpha) s_T$$

For adaptive-response-rate single exponential smoothing, let s_t be the smoothed value for the t-th time period. Initialize for adaptive-response-rate single exponential smoothing as follows:

$$s_1 = x_0$$

$$\alpha_1 = \alpha_2 = \alpha_3 = \delta = 0.2$$

$$A_0 = M_0 = 0$$

Update the parameter of α as follows:

$$E_t = X_t - S_t$$

$$A_t = \delta E_t + (1 - \delta) A_{t-1}$$

$$M_t = \delta |E_t| + (1 - \delta) M_{t-1}$$

$$\alpha_t = \frac{A_{t-1}}{|M_{t-1}|} \quad (t \geq 4)$$

The calculation of the smoothed value as follows:

$$s_{t+1} = \alpha_t x_t + (1 - \alpha_t) s_t$$

Where $\alpha, \delta \in (0, 1)$ is a user specified parameter, and $| |$ denotes absolute values.

It is worth noting that when $t \geq T+2$, the smoothed value s_t , that is, the forecast value, is always s_{T+1} (x_{t-1} is not available and s_{t-1} is used instead).

PAL calculates the prediction interval to get the idea of likely variation. Assume that the forecast data is normally distributed. The mean value is s_t and the variance is σ^2 . Let U_t be the upper bound of prediction interval for s_t and L_t be the lower bound. Then they are calculated as follows:

$$U_t = s_t + z\sigma$$

$$L_t = s_t - z\sigma$$

Here z is the one-tailed value of a standard normal distribution. It is derived from the input parameters PREDICTION_CONFIDENCE_1 and PREDICTION_CONFIDENCE_2.

i Note

The algorithm is backward compatible. You can still work in the SAP HANA SPS 11 or older versions where the prediction interval feature is not available. In that case, only point forecasts are calculated.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

SINGLESMOOTH

This is a single exponential smoothing function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'SINGLESMOOTH',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT
4	<schema_name>	<Statistics OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output table>, <statistics output table>) with overview;
```

i Note

The statistics output table is optional.

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	ID
	2nd column	Integer or double	Raw data

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
ALPHA	Double	0.1 (Single exponential smoothing) 0.2 (Adaptive-response-rate single exponential smoothing)	The smoothing constant alpha for single exponential smoothing or the initialization value for adaptive-response-rate single exponential smoothing ($0 < \alpha < 1$).	
DELTA	Double	0.2	Value of weighted for A_t and M_t .	Only valid when ADAPTIVE_METHOD is 1.
FORECAST_NUM	Integer	0	Number of values to be forecast. When it is set to 1, the algorithm only forecasts one value.	
ADAPTIVE_METHOD	Integer	0	<ul style="list-style-type: none"> • 0: Single exponential smoothing • 1: Adaptive-response-rate single exponential smoothing 	

Name	Data Type	Default Value	Description	Dependency
MEASURE_NAME	Varchar	No default value	<p>Measure name. Supported measures are MPE, MSE, RMSE, ET, MAD, MASE, WMAPE, SMAPE, and MAPE.</p> <p>For detailed information on measures, see <i>Forecast Accuracy Measures</i>.</p>	
IGNORE_ZERO	Integer	0	<ul style="list-style-type: none"> • 0: Uses zero values in the input dataset when calculating MPE or MAPE. • 1: Ignores zero values in the input dataset when calculating MPE or MAPE. 	Only valid when MEASURE_NAME is MPE or MAPE.
EXPOST_FLAG	Integer	1	<ul style="list-style-type: none"> • 0: Does not output the expost forecast, and just outputs the forecast values. • 1: Outputs the expost forecast and the forecast values. 	
PREDICTION_CONFIDENCE_1	Double	0.8	Prediction confidence for interval 1	Only valid when the upper and lower columns are provided in the result table.
PREDICTION_CONFIDENCE_2	Double	0.95	Prediction confidence for interval 2	Only valid when the upper and lower columns are provided in the result table.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID

Table	Column	Column Data Type	Description
	2nd column	Integer or double	Output result
	3rd column	Integer or double	Lower bound of prediction interval 1
	4th column	Integer or double	Upper bound of prediction interval 1
	5th column	Integer or double	Lower bound of prediction interval 2
	6th column	Integer or double	Upper bound of prediction interval 2
	1st column	Varchar or nvarchar	Name of statistics
Statistics (optional)	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SMOOTH_DATA_T;
CREATE TYPE PAL_SMOOTH_DATA_T AS TABLE("ID" INT, "RAWDATA" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE ("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_SMOOTH_RESULT_T;
CREATE TYPE PAL_SMOOTH_RESULT_T AS TABLE ("TIME" INT, "OUTPUT" DOUBLE,
"LOWER1" DOUBLE, "UPPER1" DOUBLE, "LOWER2" DOUBLE, "UPPER2" DOUBLE);
DROP TYPE PAL_SMOOTH_STATISTICS_T;
CREATE TYPE PAL_SMOOTH_STATISTICS_T AS TABLE ("NAME" VARCHAR (50), "VALUE"
DOUBLE);
DROP TABLE PAL_SMOOTH_PDATA_TBL;
CREATE COLUMN TABLE PAL_SMOOTH_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_SMOOTH_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_SMOOTH_DATA_T','IN');
INSERT INTO PAL_SMOOTH_PDATA_TBL VALUES (2,'DM_PAL', 'PAL_CONTROL_T','IN');
INSERT INTO PAL_SMOOTH_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_SMOOTH_RESULT_T','OUT');
INSERT INTO PAL_SMOOTH_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_SMOOTH_STATISTICS_T','OUT');
CALL_SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'SMOOTH_TEST_PROC');
CALL_SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'SMOOTH', 'DM_PAL',
'SMOOTH_TEST_PROC',PAL_SMOOTH_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('ADAPTIVE_METHOD',0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'MSE');

```

```

INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL,0.1, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('DELTA', NULL,0.2, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM',6, NULL,NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXPOST_FLAG',1, NULL,NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PREDICTION_CONFIDENCE_1', NULL, 0.8, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PREDICTION_CONFIDENCE_2', NULL, 0.95,
NULL);
DROP TABLE PAL_SMOOTHING_DATA_TBL;
CREATE COLUMN TABLE PAL_SMOOTHING_DATA_TBL LIKE PAL_SMOOTHING_DATA_T ;
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (1,200.0);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (2,135.0);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (3,195.0);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (4,197.5);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (5,310.0);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (6,175.0);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (7,155.0);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (8,130.0);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (9,220.0);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (10,277.5);
INSERT INTO PAL_SMOOTHING_DATA_TBL VALUES (11,235.0);
DROP TABLE PAL_SMOOTHING_RESULT_TBL;
CREATE COLUMN TABLE PAL_SMOOTHING_RESULT_TBL LIKE PAL_SMOOTHING_RESULT_T ;
DROP TABLE PAL_SMOOTHING_STATISTICS_TBL;
CREATE COLUMN TABLE PAL_SMOOTHING_STATISTICS_TBL LIKE
PAL_SMOOTHING_STATISTICS_T;
CALL DM_PAL.SMOOTHING_TEST_PROC(PAL_SMOOTHING_DATA_TBL,
"#PAL_CONTROL_TBL", PAL_SMOOTHING_RESULT_TBL,
PAL_SMOOTHING_STATISTICS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_SMOOTHING_RESULT_TBL;
SELECT * FROM PAL_SMOOTHING_STATISTICS_TBL;

```

Expected Result

PAL_SMOOTHING_RESULT_TBL:

	TIME	OUTPUT	LOWER1	UPPER1	LOWER2	UPPER2
1	2	200	?	?	?	?
2	3	193.5	?	?	?	?
3	4	193.65	?	?	?	?
4	5	194.035	?	?	?	?
5	6	205.6315	?	?	?	?
6	7	202.5683499...	?	?	?	?
7	8	197.8115149...	?	?	?	?
8	9	191.0303635	?	?	?	?
9	10	193.92732715	?	?	?	?
10	11	202.2845944...	?	?	?	?
11	12	205.5561349...	126.70...	284.4...	84.957...	326.1...
12	13	205.5561349...	126.30...	284.8...	84.356...	326.7...
13	14	205.5561349...	125.91...	285.1...	83.757...	327.3...
14	15	205.5561349...	125.52...	285.5...	83.161...	327.9...
15	16	205.5561349...	125.13...	285.9...	82.569...	328.5...
16	17	205.5561349...	124.75...	286.3...	81.979...	329.1...

PAL_SINGLESMOOTH_STATISTICS_TBL:

NAME	VALUE
MSE	3,438.3321253085405

Related Information

[Forecast Accuracy Measures \[page 358\]](#)

3.5.9 Double Exponential Smoothing

Double Exponential Smoothing model is suitable to model the time series with trend but without seasonality. In the model there are two kinds of smoothed quantities: smoothed signal and smoothed trend.

PAL provides two methods of double exponential smoothing: Holt's linear exponential smoothing and additive damped trend Holt's linear exponential smoothing. The Holt's linear exponential smoothing displays a constant trend indefinitely into the future. Empirical evidence shows that the Holt's linear method tends to over-forecast. A parameter that is used to damp the trend could improve the situation.

Holt's Linear Exponential Smoothing

Let s_t and b_t be the smoothed value and smoothed trend for the $(t+1)$ -th time period, respectively. The following rules are satisfied:

$$s_0 = x_0$$

$$b_0 = x_1 - x_0$$

$$s_t = \alpha x_t + (1 - \alpha) (s_{t-1} + b_{t-1})$$

$$b_t = \beta (s_t - s_{t-1}) + (1 - \beta) b_{t-1}$$

Where $\alpha, \beta \in (0, 1)$ are two user specified parameters. The model can be understood as two coupled Single Exponential Smoothing models, and forecast can be made by the following equation:

$$F_{T+m} = s_T + mb_T$$

Additive Damped Trend Holt's Linear Exponential Smoothing

Let s_t and b_t be the smoothed value and smoothed trend for the $(t+1)$ -th time period, respectively. The following rules are satisfied:

$$s_0 = x_0$$

$$b_0 = x_1 - x_0$$

$$s_t = \alpha x_t + (1 - \alpha) (s_{t-1} + \phi b_{t-1})$$

$$b_t = \beta (s_t - s_{t-1}) + (1 - \beta) \phi b_{t-1}$$

Where $\alpha, \beta, \phi \in (0, 1)$ are three user specified parameters. Forecast can be made by the following equation:

$$F_{T+m} = s_T + (\phi + \phi^2 + \dots + \phi^m) b_T$$

i Note

F_0 is not defined because there is no estimation for time 0. According to the definition, you can get $F_1 = s_0 + b_0$ and so on.

PAL calculates the prediction interval to get the idea of likely variation. Assume that the forecast data is normally distributed. The mean value is s_t and the variance is σ^2 . Let U_t be the upper bound of prediction interval for s_t and L_t be the lower bound. Then they are calculated as follows:

$$U_t = s_t + z\sigma$$

$$L_t = s_t - z\sigma$$

Here z is the one-tailed value of a standard normal distribution. It is derived from the input parameters PREDICTION_CONFIDENCE_1 and PREDICTION_CONFIDENCE_2.

i Note

The algorithm is backward compatible. You can still work in the SAP HANA SPS 11 or older versions where the prediction interval feature is not available. In that case, only point forecasts are calculated.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

DOUBLESMOOTH

This is a double exponential smoothing function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'DOUBLESMOOTH',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT

Position	Schema Name	Table Type Name	Parameter Type
4	<schema_name>	<Statistics OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output table>, <statistics output table>) with overview;
```

i Note

The statistics output table is optional.

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	ID
	2nd column	Integer or double	Raw data

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
ALPHA	Double	0.1	Value of the smoothing constant alpha ($0 < \alpha < 1$).	
BETA	Double	0.1	Value of the smoothing constant beta ($0 < \beta < 1$).	
FORECAST_NUM	Integer	0	Number of values to be forecast.	

Name	Data Type	Default Value	Description	Dependency
PHI	Double	0.1	Value of the damped smoothing constant ϕ ($0 < \phi < 1$).	
DAMPED	Integer	0	<ul style="list-style-type: none"> • 0: Uses the Holt's linear method. • 1: Uses the additive damped trend Holt's linear method. 	
MEASURE_NAME	Varchar	No default value	<p>Measure name. Supported measures are MPE, MSE, RMSE, ET, MAD, MASE, WMAPE, SMAPE, and MAPE.</p> <p>For detailed information on measures, see <i>Forecast Accuracy Measures</i>.</p>	
IGNORE_ZERO	Integer	0	<ul style="list-style-type: none"> • 0: Uses zero values in the input dataset when calculating MPE or MAPE. • 1: Ignores zero values in the input dataset when calculating MPE or MAPE. 	Only valid when MEASURE_NAME is MPE or MAPE.
EXPOST_FLAG	Integer	1	<ul style="list-style-type: none"> • 0: Does not output the expost forecast, and just outputs the forecast values. • 1: Outputs the expost forecast and the forecast values. 	
PREDICTION_CONFIDENCE_1	Double	0.8	Prediction confidence for interval 1	Only valid when the upper and lower columns are provided in the result table.

Name	Data Type	Default Value	Description	Dependency
PREDICTION_CONFIDENCE_2	Double	0.95	Prediction confidence for interval 2	Only valid when the upper and lower columns are provided in the result table.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	2nd column	Integer or double	Output result
	3rd column	Integer or double	Lower bound of prediction interval 1
	4th column	Integer or double	Upper bound of prediction interval 1
	5th column	Integer or double	Lower bound of prediction interval 2
	6th column	Integer or double	Upper bound of prediction interval 2
Statistics (optional)	1st column	Varchar or nvarchar	Name of statistics
	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_DOBLESMOOTH_DATA_T;
CREATE TYPE PAL_DOBLESMOOTH_DATA_T AS TABLE("ID" INT, "RAWDATA" DOUBLE);
DROP TYPE PAL_DOBLESMOOTH_RESULT_T;
CREATE TYPE PAL_DOBLESMOOTH_RESULT_T AS TABLE("TIME" INT, "OUTPUT" DOUBLE,
"LOWER1" DOUBLE, "UPPER1" DOUBLE, "LOWER2" DOUBLE, "UPPER2" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TYPE PAL_DOBLESMOOTH_STATISTIC_T;
CREATE TYPE PAL_DOBLESMOOTH_STATISTIC_T AS TABLE("NAME" VARCHAR(100), "VALUE"
DOUBLE);
DROP TABLE PAL_DOBLESMOOTH_STATISTIC_TBL;
CREATE COLUMN TABLE PAL_DOBLESMOOTH_STATISTIC_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
```

```

INSERT INTO PAL_DOUBLESMOOTH_STATISTIC_TBL VALUES (1,'DM_PAL',
'PAL_DOUBLESMOOTH_DATA_T','IN');
INSERT INTO PAL_DOUBLESMOOTH_STATISTIC_TBL VALUES (2,'DM_PAL',
'PAL_CONTROL_T','IN');
INSERT INTO PAL_DOUBLESMOOTH_STATISTIC_TBL VALUES (3,'DM_PAL',
'PAL_DOUBLESMOOTH_RESULT_T','OUT');
INSERT INTO PAL_DOUBLESMOOTH_STATISTIC_TBL VALUES (4,'DM_PAL',
'PAL_DOUBLESMOOTH_STATISTIC_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'DOUBLESMOOTH_TEST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'DOUBLESMOOTH', 'DM_PAL',
'DOUBLESMOOTH_TEST_PROC',PAL_DOUBLESMOOTH_STATISTIC_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL, 0.501, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BETA', NULL, 0.072, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM', 6, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXPOST_FLAG', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'MSE');
INSERT INTO #PAL_CONTROL_TBL VALUES ('PREDICTION_CONFIDENCE_1', NULL, 0.8, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PREDICTION_CONFIDENCE_2', NULL, 0.95,
NULL);
DROP TABLE PAL_DOUBLESMOOTH_DATA_TBL;
CREATE COLUMN TABLE PAL_DOUBLESMOOTH_DATA_TBL LIKE PAL_DOUBLESMOOTH_DATA_T;
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (1,143.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (2,152.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (3,161.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (4,139.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (5,137.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (6,174.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (7,142.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (8,141.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (9,162.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (10,180.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (11,164.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (12,171.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (13,206.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (14,193.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (15,207.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (16,218.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (17,229.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (18,225.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (19,204.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (20,227.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (21,223.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (22,242.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (23,239.0);
INSERT INTO PAL_DOUBLESMOOTH_DATA_TBL VALUES (24,266.0);
DROP TABLE PAL_DOUBLESMOOTH_RESULT_TBL;
CREATE COLUMN TABLE PAL_DOUBLESMOOTH_RESULT_TBL  LIKE PAL_DOUBLESMOOTH_RESULT_T ;
DROP TABLE PAL_DOUBLESMOOTH_STATISTIC_TBL;
CREATE COLUMN TABLE PAL_DOUBLESMOOTH_STATISTIC_TBL  LIKE
PAL_DOUBLESMOOTH_STATISTIC_T;
CALL DM_PAL.DOUBLESMOOTH_TEST_PROC(PAL_DOUBLESMOOTH_DATA_TBL,
"#PAL_CONTROL_TBL", PAL_DOUBLESMOOTH_RESULT_TBL,
PAL_DOUBLESMOOTH_STATISTIC_TBL)WITH OVERVIEW;
SELECT * FROM PAL_DOUBLESMOOTH_RESULT_TBL;
SELECT * FROM PAL_DOUBLESMOOTH_STATISTIC_TBL;

```

Expected Result

PAL_DOUBLESMOOTH_RESULT_TBL:

	TIME	OUTPUT	LOWER1	UPPER1	LOWER2	UPPER2
1	2	152	?	?	?	?
2	3	161	?	?	?	?
3	4	170	?	?	?	?
4	5	162.35...	?	?	?	?
5	6	156.61...	?	?	?	?
6	7	172.92...	?	?	?	?
7	8	163.90...	?	?	?	?
8	9	158.08...	?	?	?	?
9	10	165.83...	?	?	?	?
10	11	179.23...	?	?	?	?
11	12	177.35...	?	?	?	?
12	13	179.69...	?	?	?	?
13	14	199.35...	?	?	?	?
14	15	202.41...	?	?	?	?
15	16	211.12...	?	?	?	?
16	17	221.22...	?	?	?	?
17	18	232.06...	?	?	?	?
18	19	235.20...	?	?	?	?
19	20	225.13...	?	?	?	?
20	21	231.69...	?	?	?	?
21	22	232.65...	?	?	?	?
22	23	242.98...	?	?	?	?
23	24	246.49...	?	?	?	?
24	25	262.47...	240.72...	284.2...	229.20...	295.7...
25	26	268.68...	243.99...	293.3...	230.92...	306.4...
26	27	274.89...	247.23...	302.5...	232.59...	317.2...
27	28	281.10...	250.43...	311.7...	234.19...	328.0...
28	29	287.31...	253.58...	321.0...	235.72...	338.9...
29	30	293.52...	256.68...	330.3...	237.17...	349.8...

PAL_DOUBLESMOOTH_STATISTIC_TBL:

NAME	VALUE
MSE	274.8960228191495

Related Information

[Forecast Accuracy Measures \[page 358\]](#)

3.5.10 Triple Exponential Smoothing

Triple exponential smoothing is used to handle the time series data containing a seasonal component. This method is based on three smoothing equations: stationary component, trend, and seasonal. Both seasonal and trend can be additive or multiplicative. PAL supports multiplicative triple exponential smoothing and additive triple exponential smoothing. For additive triple exponential smoothing, an additive damped method is also supported.

Multiplicative triple exponential smoothing is given by the below formula:

$$\begin{aligned} S_t &= \alpha \times \frac{x_t}{c_{t-L}} + (1 - \alpha) \times (S_{t-1} + B_{t-1}) \\ B_t &= \beta \times (S_t - S_{t-1}) + (1 - \beta) \times B_{t-1} \\ C_t &= \gamma \times \frac{x_t}{S_t} + (1 - \gamma) \times C_{t-L} \\ F_{t+m} &= (S_t + m \times B_t) \times C_{t-L+1+(m-1) \bmod L} \end{aligned}$$

Additive triple exponential smoothing is given by the below formula:

$$\begin{aligned} S_t &= \alpha \times (x_t - c_{t-L}) + (1 - \alpha) \times (S_{t-1} + B_{t-1}) \\ B_t &= \beta \times (S_t - S_{t-1}) + (1 - \beta) \times B_{t-1} \\ C_t &= \gamma \times (x_t - S_t) + (1 - \gamma) \times C_{t-L} \\ F_{t+m} &= S_t + m \times B_t + C_{t-L+1+((m-1) \bmod L)} \end{aligned}$$

The additive damped method of additive triple exponential smoothing is given by the below formula:

$$\begin{aligned} S_t &= \alpha \times (x_t - c_{t-L}) + (1 - \alpha) \times (S_{t-1} + \phi \times B_{t-1}) \\ B_t &= \beta \times (S_t - S_{t-1}) + (1 - \beta) \times \phi \times B_{t-1} \\ C_t &= \gamma \times (x_t - S_t) + (1 - \gamma) \times C_{t-L} \\ F_{t+m} &= S_t + (\phi + \phi^2 + \dots + \phi^m) \times B_t + C_{t-L+1+((m-1) \bmod L)} \end{aligned}$$

Where:

α	Data smoothing factor. The range is $0 < \alpha < 1$.
β	Trend smoothing factor. The range is $0 < \beta < 1$.
γ	Seasonal change smoothing factor. The range is $0 < \gamma < 1$.
ϕ	Damped smoothing factor. Than range is $0 < \phi < 1$.
x	Observation
S	Smoothed observation

B	Trend factor
C	Seasonal index
F	The forecast at m periods ahead
t	The index that denotes a time period

i Note

α , β , and γ are the constants that must be estimated in such a way that the MSE of the error is minimized.

PAL uses two methods for initialization. The first is the following formula:

To initialize trend estimate B_{L-1} :

$$B_{L-1} = \frac{1}{L} \times \left(\sum_{i=0}^{L-1} \frac{x_{L+i} - x_i}{L} \right)$$

To initialize the seasonal indices c_i for $i = 0, 1, \dots, L-1$ for multiplicative triple exponential smoothing:

$$c_i = \frac{x_i}{s_{L-1}} \quad 0 \leq i \leq L-1$$

To initialize the seasonal indices c_i for $i = 0, 1, \dots, L-1$ for additive triple exponential smoothing:

$$c_i = x_i - s_{L-1} \quad 0 \leq i \leq L-1$$

Where

$$s_{L-1} = \frac{1}{L} \times \left(\sum_{i=0}^{L-1} x_i \right)$$

i Note

s_{L-1} is the average value of x in the L cycle of your data.

The second initialization method is as follows:

1. Get the trend component by using moving averages with the first two CYCLE observations.
2. The seasonal component is computed by removing the trend component from the observations. For additive, use the observations minus the trend component. For multiplicative, use the observations divide the trend component.
3. The start values of c_t are initialized by using the seasonal component calculated in Step 2. The start values of s_t and B_t are initialized by using a simple linear regressing on the trend component, s_t is initialized by intercept and B_t is initialized by slope.

PAL calculates the prediction interval to get the idea of likely variation. Assume that the forecast data is normally distributed. The mean value is s_t and the variance is σ^2 . Let U_t be the upper bound of prediction interval for s_t and L_t be the lower bound. Then they are calculated as follows:

$$U_t = s_t + z\sigma$$

$$L_t = s_t - z\sigma$$

Here z is the one-tailed value of a standard normal distribution. It is derived from the input parameters PREDICTION_CONFIDENCE_1 and PREDICTION_CONFIDENCE_2.

i Note

The algorithm is backward compatible. You can still work in the SAP HANA SPS 11 or older versions where the prediction interval feature is not available. In that case, only point forecasts are calculated.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

TRIPLESMOOTH

This is a triple exponential smoothing function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'TRIPLESMOOTH',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT
4	<schema_name>	<Statistics OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output table>, <statistics output table>) with overview;
```

i Note

The statistics output table is optional.

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	ID
	2nd column	Integer or double	Raw data

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
ALPHA	Double	0.1	Value of the smoothing constant alpha ($0 < \alpha < 1$).	
BETA	Double	0.1	Value of the smoothing constant beta ($0 < \beta < 1$).	
GAMMA	Double	0.1	Value of the smoothing constant gamma ($0 < \gamma < 1$).	
CYCLE	Integer	2	A cycle of length L ($L > 1$). For example, quarterly data cycle is 4, and monthly data cycle is 12.	
FORECAST_NUM	Integer	0	Number of values to be forecast.	

Name	Data Type	Default Value	Description	Dependency
SEASONAL	Integer	0	<ul style="list-style-type: none"> • 0: Multiplicative triple exponential smoothing • 1: Additive triple exponential smoothing 	When SEASONAL is set to 1, the default value of INITIAL_METHOD is 1; When SEASONAL is set to 0, the default value of INITIAL_METHOD is 0.
INITIAL_METHOD	Integer	0 or 1	<p>Initialization method for trend and seasonal</p> <ul style="list-style-type: none"> • 0: The first initialization method as the description part. • 1: The second initialization method as the description part. 	When SEASONAL is set to 1, the default value of INITIAL_METHOD is 1; When SEASONAL is set to 0, the default value of INITIAL_METHOD is 0.
PHI	Double	0.1	Value of the damped smoothing constant Φ ($0 < \Phi < 1$).	
DAMPED	Integer	0	<ul style="list-style-type: none"> • 0: Uses the Holt's linear method. • 1: Uses the additive damped trend Holt's linear method. 	
MEASURE_NAME	Varchar	No default value	<p>Measure name. Supported measures are MPE, MSE, RMSE, ET, MAD, MASE, WMAPE, SMAPE, and MAPE.</p> <p>For detailed information on measures, see <i>Forecast Accuracy Measures</i>.</p>	

Name	Data Type	Default Value	Description	Dependency
IGNORE_ZERO	Integer	0	<ul style="list-style-type: none"> • 0: Uses zero values in the input dataset when calculating MPE or MAPE. • 1: Ignores zero values in the input dataset when calculating MPE or MAPE. 	Only valid when MEASURE_NAME is MPE or MAPE.
EXPOST_FLAG	Integer	1	<ul style="list-style-type: none"> • 0: Does not output the expost forecast, and just outputs the forecast values. • 1: Outputs the ex-post forecast and the forecast values. 	
PREDICTION_CONFIDENCE_1	Double	0.8	Prediction confidence for interval 1	Only valid when the upper and lower columns are provided in the result table.
PREDICTION_CONFIDENCE_2	Double	0.95	Prediction confidence for interval 2	Only valid when the upper and lower columns are provided in the result table.

i Note

Cycle determines the seasonality within the time series data by considering the seasonal factor of a data point_{t-CYCLE+1} in the forecast calculation of data point_{t+1}. Additionally, the algorithm of TESM takes an entire CYCLE as the base to calculate the first forecasted value for data point_{CYCLE+1}. The value for CYCLE should be within the range of $2 \leq CYCLE \leq \text{entire number of data point}/2$.

For example, there is one year of weekly data (52 data points) as input time series. The value for CYCLE should range within $2 \leq CYCLE \leq 26$. If CYCLE is 4, we get the first forecast value for data point 5 (e.g. week 201205) which considers the seasonal factor of data point 1 (e.g. week 201201). The second forecast value for data point 6 (e.g. week 201206) considers the seasonal factor of data point 2 (e.g. week 201202), etc. If CYCLE is 2, then we get the first forecast value for data point 3 (e.g. week 201203) which considers the seasonal factor of data point 1 (e.g. week 201201). The second forecast value for data point 4 (e.g. week 201204) considers the seasonal factor of data point 2 (e.g. week 201202), etc.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	2nd column	Integer or double	Output result
	3rd column	Integer or double	Lower bound of prediction interval 1
	4th column	Integer or double	Upper bound of prediction interval 1
	5th column	Integer or double	Lower bound of prediction interval 2
	6th column	Integer or double	Upper bound of prediction interval 2
Statistics (optional)	1st column	Varchar or nvarchar	Name of statistics
	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_TRIPLESMOOTH_DATA_T;
CREATE TYPE PAL_TRIPLESMOOTH_DATA_T AS TABLE("ID" INT, "RAWDATA" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE ("NAME" VARCHAR (100), "INTARGS" INT,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
DROP TYPE PAL_TRIPLESMOOTH_RESULT_T;
CREATE TYPE PAL_TRIPLESMOOTH_RESULT_T AS TABLE ("TIME" INT, "OUTPUT" DOUBLE,
"LOWER1" DOUBLE, "UPPER1" DOUBLE, "LOWER2" DOUBLE, "UPPER2" DOUBLE);
DROP TYPE PAL_TRIPLESMOOTH_STATISTICS_T;
CREATE TYPE PAL_TRIPLESMOOTH_STATISTICS_T AS TABLE ("NAME" VARCHAR (50), "VALUE"
DOUBLE);
DROP TABLE PAL_TRIPLESMOOTH_PDATA_TBL;
CREATE COLUMN TABLE PAL_TRIPLESMOOTH_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_TRIPLESMOOTH_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_TRIPLESMOOTH_DATA_T','IN');
INSERT INTO PAL_TRIPLESMOOTH_PDATA_TBL VALUES (2,'DM_PAL', 'PAL_CONTROL_T','IN');
INSERT INTO PAL_TRIPLESMOOTH_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_TRIPLESMOOTH_RESULT_T','OUT');
INSERT INTO PAL_TRIPLESMOOTH_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_TRIPLESMOOTH_STATISTICS_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'TRIPLESMOOTH_TEST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'TRIPLESMOOTH', 'DM_PAL',
'TRIPLESMOOTH_TEST_PROC',PAL_TRIPLESMOOTH_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
```

```

CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ("NAME" VARCHAR(100),
"INTARGS" INT, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', NULL, 0.822, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BETA', NULL, 0.055, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('GAMMA', NULL, 0.055, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CYCLE', 4, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('FORECAST_NUM', 6, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEASONAL', 0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INITIAL_METHOD', 0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MEASURE_NAME', NULL, NULL, 'MSE');
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXPOST_FLAG', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PREDICTION_CONFIDENCE_1', NULL, 0.8, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PREDICTION_CONFIDENCE_2', NULL, 0.95,
NULL);
DROP TABLE PAL_TRIPLESMOOTH_DATA_TBL;
CREATE COLUMN TABLE PAL_TRIPLESMOOTH_DATA_TBL LIKE PAL_TRIPLESMOOTH_DATA_T ;
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (1,362.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (2,385.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (3,432.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (4,341.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (5,382.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (6,409.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (7,498.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (8,387.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (9,473.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (10,513.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (11,582.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (12,474.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (13,544.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (14,582.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (15,681.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (16,557.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (17,628.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (18,707.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (19,773.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (20,592.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (21,627.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (22,725.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (23,854.0);
INSERT INTO PAL_TRIPLESMOOTH_DATA_TBL VALUES (24,661.0);
DROP TABLE PAL_TRIPLESMOOTH_RESULT_TBL;
CREATE COLUMN TABLE PAL_TRIPLESMOOTH_RESULT_TBL LIKE PAL_TRIPLESMOOTH_RESULT_T;
DROP TABLE PAL_TRIPLESMOOTH_STATISTICS_TBL;
CREATE COLUMN TABLE PAL_TRIPLESMOOTH_STATISTICS_TBL LIKE
PAL_TRIPLESMOOTH_STATISTICS_T;
CALL DM_PAL.TRIPLESMOOTH_TEST_PROC(PAL_TRIPLESMOOTH_DATA_TBL,
"#PAL_CONTROL_TBL", PAL_TRIPLESMOOTH_RESULT_TBL,
PAL_TRIPLESMOOTH_STATISTICS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_TRIPLESMOOTH_RESULT_TBL;
SELECT * FROM PAL_TRIPLESMOOTH_STATISTICS_TBL;

```

Expected Result

PAL_TRIPLESMOOTH_RESULT_TBL:

	TIME	OUTPUT	LOWER1	UPPER1	LOWER2	UPPER2
1	5	371.28815789473686	?	?	?	?
2	6	414.63620709090577	?	?	?	?
3	7	471.4318077577633	?	?	?	?
4	8	399.2922358985797	?	?	?	?
5	9	423.22175605383677	?	?	?	?
6	10	506.39903682061964	?	?	?	?
7	11	589.5887382682944	?	?	?	?
8	12	471.56575268616933	?	?	?	?
9	13	515.8787903699542	?	?	?	?
10	14	586.9176998062271	?	?	?	?
11	15	670.2603541868965	?	?	?	?
12	16	548.6608417948656	?	?	?	?
13	17	605.1661021997458	?	?	?	?
14	18	678.4727594613225	?	?	?	?
15	19	807.7263193081459	?	?	?	?
16	20	628.86590524558	?	?	?	?
17	21	650.2559751722578	?	?	?	?
18	22	683.0454322452091	?	?	?	?
19	23	821.7673777439587	?	?	?	?
20	24	683.4132407295845	?	?	?	?
21	25	721.983642459322	695.64...	748.3...	681.70...	762.2...
22	26	782.2537878644209	742.95...	821.5...	722.14...	842.3...
23	27	894.3794445625507	841.20...	947.5...	813.05...	975.7...
24	28	718.2726402707948	667.03...	769.5...	639.91...	796.6...
25	29	778.9447542551759	715.87...	842.0...	682.48...	875.4...
26	30	842.7761971859832	767.84...	917.7...	728.18...	957.3...

PAL_TRIPLESMOOTH_STATISTICS_TBL:

NAME	VALUE
MSE	616.5415419997985

Related Information

[Forecast Accuracy Measures \[page 358\]](#)

3.5.11 Seasonality Test

This algorithm is used to test whether a time series has a seasonality or not. If it does, the corresponding additive or multiplicative seasonality model is identified, and the de-seasonalized series (both trend and seasonality are eliminated) is given.

Typically, there are two decomposition models for a time series :

1. Additive: $x_t = m_t + s_t + y_t$
2. Multiplicative: $x_t = m_t \times s_t \times y_t$

Where m_t , s_t , and y_t are trend, seasonality, and random components, respectively. They satisfy the properties:

$$E(y_t) = 0, s_{t+d} = s_t, \sum_{j=1}^d s_j = 0$$

Where d is the length of the seasonality cycle, that is, the period. It is believed that the additive model is useful when the seasonal variation is relatively constant over time, whereas the multiplicative model is useful when the seasonal variation increases over time.

Autocorrelation is employed to identify the seasonality. The autocorrelation coefficient at lag h is given by:

$$r_h = c_h / c_0$$

Where c_h is the autocovariance function

$$c_h = \frac{1}{n} \sum_{t=1}^{n-h} (x_t - \bar{x})(x_{t+h} - \bar{x})$$

The resulting r_h has a value in the range of -1 to 1 , and a larger value indicates more relevance.

For an n -element time series, the probable seasonality cycle is from 2 to $n/2$. The main procedure to determine the seasonality, therefore, is to calculate the autocorrelation coefficients of all possible lags ($d=2, 3, \dots, n$) in the case of both additive and multiplicative models. There is a user-specified threshold for the coefficient, for example, 0.2 , indicating that only if the autocorrelation is larger than the threshold is the tested seasonality considered. If there is no lag satisfying the requirement, the time series is regarded to have no seasonality. Otherwise, the one having the largest autocorrelation is the optimal seasonality.

Practically, the autocorrelation coefficient at lag of d is calculated as follows:

1. Estimate the trend \hat{m}_t ($t = q+1, \dots, n-q$). Here moving average is applied to estimate the trend, that is,

$$\hat{m}_t = \begin{cases} \frac{1}{2q+1} \sum_{j=-q}^q x_{t-j}, & d = 2q+1 \\ \frac{1}{2q} (0.5x_{t-q} + \sum_{j=-q+1}^{q-1} x_{t-j} + 0.5x_{t+q}), & d = 2q \end{cases}$$

2. De-trend the time series. For an additive model, this is done by subtracting the trend estimates from the series. For a multiplicative decomposition, likewise, this is done by dividing the series by the trend values. And then two sets (additive and multiplicative) of autocorrelation coefficients are calculated from the de-trended series.

Once the trend and seasonality are determined, you can eventually obtain the de-trended and de-seasonalized series, that is, the random. As illustrated before, the trend component is estimated as moving average \hat{m}_t . The seasonal component, correspondingly, can be estimated as

$$\begin{cases} w_k = \frac{1}{\text{count_of_summands}} \sum_j x_{k+jd} - \hat{m}_{k+jd}, q < k + jd \leq n - q \\ \hat{s}_k = w_k - \frac{1}{d} \sum_{i=1}^d w_i, k = 1, 2, \dots, d \end{cases}$$

, additive

decomposition,

$$\begin{cases} w_k = \frac{1}{\text{count_of_summands}} \sum_j x_{k+jd} / \hat{m}_{k+jd}, q < k + jd \leq n - q \\ \hat{s}_k = w_k / (\frac{1}{d} \sum_{i=1}^d w_i), k = 1, 2, \dots, d \end{cases}$$

, multiplicative
or, decomposition.

Consequently, the random component can be calculated straightforwardly.

Note that during the course of calculating the moving average, the element at t is determined by $x(t-q, \dots, t+q)$, where $d=2q$ or $d=2q+1$, d is the seasonality cycle. As a result, the trend and random series are valid only within the time range between q and $n-q$. Should there be no seasonality, the random series is just exactly the input time series.

Prerequisites

- No null data in the inputs. The time periods should be unique and equal sampling.
- The length of time series must be at least 1.
- The data type of time periods is integer. The data type of time series is integer or double.

SEASONALITYTEST

This function identifies the seasonality and calculates de-seasonalized series (random) of a time series.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'SEASONALITYTEST',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<SEASONALITY OUTPUT table type>	OUT
4	<schema_name>	<RANDOM OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>,
<seasonality output table>, <random output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	Time periods. Time periods do not need to be in order, but must be unique and equal sampling.
	2nd column	Integer or double	The corresponding raw data of time series.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
ALPHA	Double	0.2	The threshold for the auto-correlation coefficient. The value range is (0,1].

Output Tables

Table	Column	Column Data Type	Description
Seasonality	1st column	Integer	Cycle length of seasonality (1 for no-seasonality).
	2nd column	Varchar or nvarchar	Seasonality mode. The values are "additive", "multiplicative", or "no-seasonality".
Random	1st column	Integer	Time periods that are monotonically increasing sorted.
	2nd column	Double	The random component.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_TSSEASONALITY_DATA_T;
CREATE TYPE PAL_TSSEASONALITY_DATA_T AS TABLE(
    "PERIOD" INTEGER,
    "SERIES" DOUBLE
);
DROP TYPE PAL_TSSEASONALITY_CONTROL_T;
CREATE TYPE PAL_TSSEASONALITY_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_TSSEASONALITY_SEASONALITY_T;
CREATE TYPE PAL_TSSEASONALITY_SEASONALITY_T AS TABLE(
    "CYCLE" INTEGER,
    "MODE" VARCHAR(100)
);
DROP TYPE PAL_TSSEASONALITY_RANDOM_T;
CREATE TYPE PAL_TSSEASONALITY_RANDOM_T AS TABLE(
    "PERIOD" INTEGER,
    "RANDOM" DOUBLE
);
DROP TABLE PAL_TSSEASONALITY_PDATA_TBL;
CREATE COLUMN TABLE PAL_TSSEASONALITY_PDATA_TBL(

```

```

    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_TSSEASONALITY_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_TSSEASONALITY_DATA_T', 'IN');
INSERT INTO PAL_TSSEASONALITY_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_TSSEASONALITY_CONTROL_T', 'IN');
INSERT INTO PAL_TSSEASONALITY_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_TSSEASONALITY_SEASONALITY_T', 'OUT');
INSERT INTO PAL_TSSEASONALITY_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_TSSEASONALITY_RANDOM_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL',
'PAL_TIMESERIESSEASONALITY_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'SEASONALITYTEST',
'DM_PAL', 'PAL_TIMESERIESSEASONALITY_PROC', 'PAL_TSSEASONALITY_PDATA_TBL');
DROP TABLE PAL_TSSEASONALITY_DATA_TBL;
CREATE COLUMN TABLE PAL_TSSEASONALITY_DATA_TBL LIKE PAL_TSSEASONALITY_DATA_T;
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (1, 10);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (2, 7);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (3, 17);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (4, 34);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (5, 9);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (6, 7);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (7, 18);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (8, 40);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (9, 27);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (10, 7);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (11, 27);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (12, 100);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (13, 93);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (14, 29);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (15, 159);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (16, 614);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (17, 548);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (18, 102);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (19, 21);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (20, 238);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (21, 89);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (22, 292);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (23, 446);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (24, 689);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (25, 521);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (26, 155);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (27, 968);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (28, 1456);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (29, 936);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (30, 10);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (31, 83);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (32, 55);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (33, 207);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (34, 25);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (35, 0);
INSERT INTO PAL_TSSEASONALITY_DATA_TBL VALUES (36, 0);
DROP TABLE #PAL_TSSEASONALITY_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_TSSEASONALITY_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_TSSEASONALITY_CONTROL_TBL VALUES ('ALPHA', null, 0.05, null);
DROP TABLE PAL_TSSEASONALITY_SEASONALITY_TBL;
CREATE COLUMN TABLE PAL_TSSEASONALITY_SEASONALITY_TBL LIKE
PAL_TSSEASONALITY_SEASONALITY_T;
DROP TABLE PAL_TSSEASONALITY_RANDOM_TBL;
CREATE COLUMN TABLE PAL_TSSEASONALITY_RANDOM_TBL LIKE PAL_TSSEASONALITY_RANDOM_T;

```

```

CALL "DM_PAL".PAL_TIMESERIESSEASONALITY PROC(PAL_TSSEASONALITY_DATA_TBL,
"#PAL_TSSEASONALITY_CONTROL_TBL", PAL_TSSEASONALITY_SEASONALITY_TBL,
PAL_TSSEASONALITY_RANDOM_TBL) with overview;
SELECT * FROM PAL_TSSEASONALITY_SEASONALITY_TBL;
SELECT * FROM PAL_TSSEASONALITY_RANDOM_TBL;

```

Expected Result

PAL_TSSEASONALITY_SEASONALITY_TBL:

CYCLE	MODE
4	Multiplicative

PAL_TSSEASONALITY_RANDOM_TBL:

PERIOD	RANDOM
3	1.3452710573776516
4	1.2313043238363521
5	0.4257604880678705
6	1.1269162486811228
7	1.1584013711650227
8	1.0549538324429104
9	0.8934352210750649
10	0.6107714019569445
11	0.7434070655156976
12	1.0194931994196192
13	0.9427553664359989
14	0.5195522964698683
15	0.7566158324803169
16	1.0745083318974957
17	1.2923748936181296
18	1.0627851898617908
19	0.16508000898868...
20	1.05959950069587
21	0.33336671646369...
22	2.586287448680693
23	1.3754715793490764
24	0.8894840852738921
25	0.8029242795005811
26	0.6521897040170259
27	1.5632921771207287
28	1.0262394754059023
29	1.0209525623352522
30	0.06405239855610...
31	0.6161861667790781
32	0.36814250980421...
33	2.0121557312796616
34	1.1011705705527075

3.5.12 Trend Test

This algorithm is used to identify whether a time series has an upward or downward trend or not, and calculate the de-trended time series.

Two methods are provided for identifying the trend: difference-sign test and rank test.

Difference-Sign Test

The difference-sign test counts the number S of times that $x(t) - x(t-1)$ is positive, $t=1, 2, \dots, n$. For an IID (Independent and Identically Distributed) series, the theoretical expectation of s is

$$\mu_S = (n-1)/2,$$

and the corresponding standard deviation is

$$\sigma_S = \sqrt{(n+1)/2}$$

For a large n, s approximately behaves Gaussian distribution $N(\mu_S, \sigma_S^2)$. Hence, a large positive or negative value of $s - \mu_S$ indicates the presence of an increasing (or decreasing) trend. The data is considered to have a trend if $|s - \mu_S| > \sigma_S$, otherwise no trend exists. Nevertheless, the difference-sign test must be used with great caution. In the case of large proportion of tie data, for example, it may give a result of negative trend, while in fact it has no trend.

Rank Test

The second solution is the rank test, which is usually known as Mann-Kendall (MK) Test (Mann 1945, Kendall 1975, Gilbert 1987). It tests whether to reject the null hypothesis (H_0) and accept the alternative hypothesis (H_a), where

H_0 : No monotonic trend,

H_a : Monotonic trend is present,

a: the tolerance probability that falsely concludes a trend exists when there is none, $0 < a < 0.5$.

The MK test may give rise to different trends for an identical time series, given a distinct a. Anyway, for a very small a, for example, 0.05, MK test is expected to achieve a quite satisfactory estimation for trend. At the same time, MK test requires that the length of time series should be at least 4. Provided the length of data set is only 3, therefore, a linear regression strategy is applied.

The resulting trend indicator has three possible numeric values: 1 indicating upward trend, -1 indicating downward trend, and 0 for no trend.

Should there be a trend, a de-trended time series with first differencing approach is given:

$$w(t) = x(t) - x(t-1),$$

where $x(t)$ is the input time series, and $w(t)$ is the de-trended time series. Apparently, the length of de-trended series is exactly one less than the input's (lack of the first period). On the other hand, the output series is just the input one if no trend is identified. Note that the resulting time series is sorted by time periods.

Prerequisites

- No null data in the inputs. The time periods should be unique.
- The length of time series must be at least 3.
- The data type of time periods is integer. The data type of time series is integer or double.

TRENDTEST

This function identifies the seasonality and calculates de-seasonalized series (random) of a time series.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'TRENDTEST',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<TREND INDICATOR OUTPUT table type>	OUT
4	<schema_name>	<DE-TRENDED OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <trend
  indicator output table>, <de-trended output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	Time periods. Time periods do not need to be in order, but must be unique.

Table	Column	Column Data Type	Description
	2nd column	Integer or double	The corresponding raw data of time series.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
METHOD	Integer	1	<ul style="list-style-type: none"> • 1: MK test • 2: Difference-sign test
ALPHA	Double	0.05	Tolerance probability for MK test. The value range is (0,0.5).

Output Tables

Table	Column	Column Data Type	Description
Trend	1st column	Integer	Indicator of trend: <ul style="list-style-type: none"> • 1: Upward • -1: Downward • 0: No trend
De-trended Series	1st column	Integer	Time periods that are monotonically increasing sorted.
	2nd column	Integer or double	The corresponding de-trended time series.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and

- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_TSTREND_DATA_T;
CREATE TYPE PAL_TSTREND_DATA_T AS TABLE(
    "PERIOD" INTEGER,
    "SERIES" DOUBLE
);
DROP TYPE PAL_TSTREND_CONTROL_T;
CREATE TYPE PAL_TSTREND_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_TSTREND_TREND_T;
CREATE TYPE PAL_TSTREND_TREND_T AS TABLE(
    "TREND" INTEGER
);
DROP TYPE PAL_TSTREND_DETRENDED_T;
CREATE TYPE PAL_TSTREND_DETRENDED_T AS TABLE(
    "PERIOD" INTEGER,
    "DETRENDED" DOUBLE
);
DROP TABLE PAL_TSTREND_PDATA_TBL;
CREATE COLUMN TABLE PAL_TSTREND_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_TSTREND_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_TSTREND_DATA_T',
'IN');
INSERT INTO PAL_TSTREND_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_TSTREND_CONTROL_T',
'IN');
INSERT INTO PAL_TSTREND_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_TSTREND_TREND_T',
'OUT');
INSERT INTO PAL_TSTREND_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_TSTREND_DETRENDED_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE_DROP('DM_PAL', 'PAL_TIMESERIESTREND_PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE_CREATE('AFLPAL', 'TRENDTEST', 'DM_PAL',
'PAL_TIMESERIESTREND_PROC', PAL_TSTREND_PDATA_TBL);
DROP TABLE PAL_TSTREND_DATA_TBL;
CREATE COLUMN TABLE PAL_TSTREND_DATA_TBL LIKE PAL_TSTREND_DATA_T;
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (1, 1500);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (2, 1510);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (3, 1550);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (4, 1650);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (5, 1620);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (6, 1690);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (7, 1695);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (8, 1700);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (9, 1710);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (10, 1705);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (11, 1708);
INSERT INTO PAL_TSTREND_DATA_TBL VALUES (12, 1715);
DROP TABLE #PAL_TSTREND_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_TSTREND_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_TSTREND_CONTROL_TBL VALUES ('METHOD', 1, null, null);
INSERT INTO #PAL_TSTREND_CONTROL_TBL VALUES ('ALPHA', null, 0.05, null);
DROP TABLE PAL_TSTREND_TREND_TBL;

```

```

CREATE COLUMN TABLE PAL_TSTREND_TREND_TBL LIKE PAL_TSTREND_TREND_T;
DROP TABLE PAL_TSTREND_DETRENDED_TBL;
CREATE COLUMN TABLE PAL_TSTREND_DETRENDED_TBL LIKE PAL_TSTREND_DETRENDED_T;
CALL "DM_PAL".PAL_TIMESERIESTREND_PROC(PAL_TSTREND_DATA_TBL,
"#PAL_TSTREND_CONTROL_TBL", PAL_TSTREND_TREND_TBL, PAL_TSTREND_DETRENDED_TBL)
with overview;
SELECT * FROM PAL_TSTREND_TREND_TBL;
SELECT * FROM PAL_TSTREND_DETRENDED_TBL;

```

Expected Result

PAL_TSTREND_TREND_TBL:

TREND
1

PAL_TSTREND_DETRENDED_TBL:

PERIOD	DETRENDED
2	10
3	40
4	100
5	-30
6	70
7	5
8	5
9	10
10	-5
11	3
12	7

3.5.13 White Noise Test

This algorithm is used to identify whether a time series is a white noise series. If white noise exists in the raw time series, the algorithm returns the value of 1. If not, the value of 0 will be returned.

PAL uses Ljung-Box test to test for autocorrelation at different lags. The Ljung-Box test can be defined as follows:

H_0 : White noise exists in the time series ($\rho_1 = \rho_2 = \rho_3 = \dots = \rho_m = 0$).

H_1 : White noise does not exist in the time series.

The Ljung-Box test statistic is given by the following formula:

$$Q = n(n+2) \sum_{h=1}^m \frac{\hat{\rho}_h^2}{n-h}$$

Where n is the sample size, $\hat{\rho}_h$ is the sample autocorrelation at lag h , and m is the number of lags being tested. The statistic of Q follows a chi-square distribution. Based on the significance level α , the critical region for rejection of the hypothesis of randomness is

$$Q > \chi^2_{1-\alpha, m}$$

Where $\chi^2_{1-\alpha, m}$ is the chi-squared distribution with m degrees of freedom and α quantile.

Prerequisites

- No null data in the inputs. The time periods should be unique.
- The data type of time periods is integer. The data type of time series is integer or double.

WHITENOISETEST

This function identifies the white noise of a time series.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'WHITENOISETEST',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<WHITE NOISE INDICATOR OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <white
noise indicator output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer	<p>Time periods.</p> <p>Time periods do not need to be in order, but must be unique.</p>
	2nd column	Integer or double	The corresponding raw data of time series.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
LAG	Integer	Half of the sample size	Specifies the lag autocorrelation coefficient that the statistic will be based on. It corresponds to the freedom degree of chi-square distribution.
THREAD_NUMBER	Integer	1	Number of threads.
PROBABILITY	Double	0.9	The confidence level used for chi-square distribution. The value is $1 - \alpha$, where α is the significance level.

Output Table

Table	Column	Column Data Type	Description
White Noise	1st column	Varchar or nvarchar	Name of white noise
	2nd column	Integer	<ul style="list-style-type: none"> • 1: White noise exists • 0: White noise does not exist

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_WHITENOISETEST_DATA_T;
CREATE TYPE PAL_WHITENOISETEST_DATA_T AS TABLE(
    "IDCOL"      INTEGER,
    "DATACOL"    DOUBLE
);

DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME"        VARCHAR (50),
    "INTARGS"     INTEGER,
    "DOUBLEARGS"  DOUBLE,
    "STRINGARGS"  VARCHAR (100)
);
DROP TYPE PAL_WHITENOISETEST_RESULT_T;
CREATE TYPE PAL_WHITENOISETEST_RESULT_T AS TABLE(
    "NAME"        VARCHAR (100),
    "VALUE"       INT
);
DROP TABLE PAL_WHITENOISETEST_DATA_TBL;
CREATE COLUMN TABLE PAL_WHITENOISETEST_DATA_TBL("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (1,'DM_PAL','PAL_WHITENOISETEST_DATA_T','IN');
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (3,'DM_PAL','PAL_WHITENOISETEST_RESULT_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_WHITENOISETEST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'WHITENOISETEST', 'DM_PAL',
'PAL_WHITENOISETEST_PROC',PAL_WHITENOISETEST_DATA_TBL);
DROP TABLE PAL_WHITENOISETEST_DATA_TBL;
CREATE COLUMN TABLE PAL_WHITENOISETEST_DATA_TBL LIKE PAL_WHITENOISETEST_DATA_T;
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (0, 1356.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (1, 826.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (2, 1586.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (3, 1010.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (4, 1337.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (5, 1415.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (6, 1514.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (7, 1474.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (8, 1662.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (9, 1805.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (10, 2218.00);
INSERT INTO PAL_WHITENOISETEST_DATA_TBL VALUES (11, 2400.00);
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL(
    "NAME"        VARCHAR (50),
    "INTARGS"     INTEGER,
    "DOUBLEARGS"  DOUBLE,
    "STRINGARGS"  VARCHAR (100)
);
INSERT INTO PAL_CONTROL_TBL VALUES ('LAG', 3, NULL, NULL);
INSERT INTO PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 8, NULL, NULL);
INSERT INTO PAL_CONTROL_TBL VALUES ('PROBABILITY', NULL, 0.9, NULL);
DROP TABLE PAL_WHITENOISETEST_RESULT_TBL;
CREATE COLUMN TABLE PAL_WHITENOISETEST_RESULT_TBL LIKE
PAL_WHITENOISETEST_RESULT_T;
```

```
CALL DM_PAL.PAL_WHITENOISETEST_PROC(PAL_WHITENOISETEST_DATA_TBL,  
PAL_CONTROL_TBL, PAL_WHITENOISETEST_RESULT_TBL) WITH OVERVIEW;  
SELECT * FROM PAL_WHITENOISETEST_RESULT_TBL;
```

Expected Result

PAL_WHITENOISETEST_RESULT_TBL:

NAME	VALUE
WHITENOISE	1

3.6 Preprocessing Algorithms

The records in business database are usually not directly ready for predictive analysis due to the following reasons:

- Some data come in large amount, which may exceed the capacity of an algorithm.
- Some data contains noisy observations which may hurt the accuracy of an algorithm.
- Some attributes are badly scaled, which can make an algorithm unstable.

To address the above challenges, PAL provides several convenient algorithms for data preprocessing.

3.6.1 Binning

Binning data is a common requirement prior to running certain predictive algorithms. It generally reduces the complexity of the model, for example, the model in a decision tree.

Binning methods replace a value by a "bin number" defined by all elements of its neighborhood, that is, the bin it belongs to. The ordered values are distributed into a number of bins. Because binning methods consult the neighborhood of values, they perform local smoothing.

i Note

Binning can only be used on a table with only one attribute.

Binning Methods

There are four binning methods:

- Equal widths based on the number of bins

Specify an integer to determine the number of equal width bins and calculate the range values by:

$$\text{BandWidth} = (\text{MaxValue} - \text{MinValue}) / K$$

Where `MaxValue` is the biggest value of every column, `MinValue` is the smallest value of every column, and `K` is the number of bins.

For example, according to this rule:

- $\text{MinValue} + \text{BinWidth} > \text{Values in Bin 1} \geq \text{MinValue}$
- $\text{MinValue} + 2 * \text{BinWidth} > \text{Values in Bin 2} \geq \text{MinValue} + \text{BinWidth}$

- Equal bin widths defined as a parameter

Specify the bin width and calculate the start and end of bin intervals by:

Start of bin intervals = Minimum data value - 0.5 * Bin width

End of bin intervals = Maximum data value + 0.5 * Bin width

For example, assuming the data has a range from 6 to 38 and the bin width is 10:

Start of bin intervals = 6 - 0.5 * 10 = 1

End of bin intervals = 38 + 0.5 * 10 = 43

Hence, the generated bins would be the following:

Bin	Value Range
Bin 1	[1, 10)
Bin 2	[10, 20)
Bin 3	[20, 30)
Bin 4	[30, 40)
Bin 5	[40, 43]

- Equal number of records per bin

Assign an equal number of records to each bin.

For example:

- 2 bins, each containing 50% of the cases (below the median / above the median)
- 4 bins, each containing 25% of the cases (grouped by the quartiles)
- 5 bins, each containing 20% of the cases (grouped by the quintiles)
- 10 bins, each containing 10% of the cases (grouped by the deciles)
- 20 bins, each containing 5% of the cases (grouped by the vingtiles)
- 100 bins, each containing 1% of the cases (grouped by the percentiles)

A tie condition results when the values on either side of a cut point are identical. In this case we move the tied values up to the next bin.

- Mean / standard deviation bin boundaries

The mean and standard deviation can be used to create bins which are above or below the mean. The rules are as follows:

- + and -1 standard deviation, so
 - Bin 1 contains values less than -1 standard deviation from the mean
 - Bin 2 contains values between -1 and +1 standard deviation from the mean
 - Bin 3 contains values greater than +1 standard deviation from the mean
- + and -2 standard deviation, so
 - Bin 1 contains values less than -2*standard deviation from the mean
 - Bin 2 contains values less than -1 standard deviation from the mean
 - Bin 3 contains values between -1 and +1 standard deviation from the mean
 - Bin 4 contains values greater than +1 standard deviation from the mean
 - Bin 5 contains values greater than +2*standard deviation from the mean
- + and -3 standard deviation, so
 - Bin 1 contains values less than -3*standard deviation from the mean
 - Bin 2 contains values less than -2*standard deviation from the mean
 - Bin 3 contains values less than -1 standard deviation from the mean
 - Bin 4 contains values between -1 and +1 standard deviation from the mean
 - Bin 5 contains values greater than +1 standard deviation from the mean
 - Bin 6 contains values greater than +2*standard deviation from the mean

Bin 7 contains values greater than +3*standard deviation from the mean

Smoothing Methods

There are three methods for smoothing:

- Smoothing by bin means: each value within a bin is replaced by the average of all the values belonging to the same bin.
- Smoothing by bin medians: each value in a bin is replaced by the median of all the values belonging to the same bin.
- Smoothing by bin boundaries: the minimum and maximum values in a given bin are identified as the bin boundaries. Each value in the bin is then replaced by its closest boundary value.

Note

When the value is equal to both sides, it will be replaced by the front boundary value.

Prerequisites

- The input data does not contain null value.
- The data is numeric, not categorical.

BINNING

This function preprocesses the data.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'BINNING', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Binning Model OUTPUT table type>	OUT (optional)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result  
output table>, <binning model output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Raw data

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description
BINNING_METHOD	Integer	Binning methods: <ul style="list-style-type: none">• 0: equal widths based on the number of bins• 1: equal widths based on the bin width• 2: equal number of records per bin• 3: mean/ standard deviation bin boundaries
SMOOTH_METHOD	Integer	Smoothing methods: <ul style="list-style-type: none">• 0: smoothing by bin means• 1: smoothing by bin medians• 2: smoothing by bin boundaries

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
BIN_NUMBER	Integer	2	Number of needed bins.	
BIN_DISTANCE	Integer	10	Specifies the distance for binning.	Only valid when BINNING_METHOD is 1.

Name	Data Type	Default Value	Description	Dependency
SD	Integer	1	Specifies the standard deviation method. Examples: 1 S.D.; 2 S.D.; 3 S.D.	Only valid when BINNING_METHOD is 3.

Output Table

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer	Bin index assigned	The value is from 1 to BIN_NUMBER.
	3rd column	Integer or double	Smoothed value	
Binning Model (optional)	1st column	Integer	Binning model ID	This must be the first column.
	2nd column	CLOB, varchar, or nvarchar	Binning model saved as JSON string	The table must be a column table. The minimum length of each unit (row) is 5000.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_BINNING_DATA_T ;
CREATE TYPE PAL_BINNING_DATA_T AS TABLE("ID" INT, "TEMPERATURE" DOUBLE) ;
DROP TYPE PAL_CONTROL_T ;
CREATE TYPE PAL_CONTROL_T AS TABLE( "NAME" VARCHAR (50), "INTARGS"
INTEGER,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR (100));
DROP TYPE PAL_BINNING_RESULT_T ;
CREATE TYPE PAL_BINNING_RESULT_T AS TABLE("ID" INT, "BIN_NUMBER" INT,
"PRE_RESULT" DOUBLE);
DROP TYPE PAL_BINNING_MODEL_T ;
CREATE TYPE PAL_BINNING_MODEL_T AS TABLE("ID" INT, "MODEL" VARCHAR (5000));
DROP TABLE PAL_BINNING_PDATA_TBL;
CREATE COLUMN TABLE PAL_BINNING_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_BINNING_PDATA_TBL VALUES (1,'DM_PAL','PAL_BINNING_DATA_T','IN');
INSERT INTO PAL_BINNING_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T','IN');
INSERT INTO PAL_BINNING_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_BINNING_RESULT_T','OUT');

```

```

INSERT INTO PAL_BINNING_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_BINNING_MODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'BINNING_TEST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','BINNING', 'DM_PAL',
'BINNING_TEST_PROC',PAL_BINNING_PDATA_TBL);
DROP TABLE PAL_BINNING_DATA_TBL;
CREATE COLUMN TABLE PAL_BINNING_DATA_TBL LIKE PAL_BINNING_DATA_T ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (0, 6.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (1, 12.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (2, 13.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (3, 15.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (4, 10.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (5, 23.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (6, 24.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (7, 30.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (8, 32.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (9, 25.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (10, 38.0) ;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR
(50), "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('BINNING_METHOD',1, NULL , NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SMOOTH_METHOD',0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BIN_NUMBER',4, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BIN_DISTANCE',10, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SD',1, NULL, NULL);
DROP TABLE PAL_BINNING_RESULT_TBL;
CREATE COLUMN TABLE PAL_BINNING_RESULT_TBL LIKE PAL_BINNING_RESULT_T ;
DROP TABLE PAL_BINNING_MODEL_TBL;
CREATE COLUMN TABLE PAL_BINNING_MODEL_TBL LIKE PAL_BINNING_MODEL_T ;
CALL DM_PAL.BINNING_TEST_PROC(PAL_BINNING_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_BINNING_RESULT_TBL, PAL_BINNING_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_BINNING_RESULT_TBL;
SELECT * FROM PAL_BINNING_MODEL_TBL;

```

Expected Result

PAL_BINNING_RESULT_TBL:

	ID	BIN_NUMBER	PRE_RESULT
1	0	1	8
2	1	2	13.33333333333334
3	2	2	13.33333333333334
4	3	2	13.33333333333334
5	4	1	8
6	5	3	25.5
7	6	3	25.5
8	7	3	25.5
9	8	4	35
10	9	3	25.5
11	10	4	35

PAL_BINNING_MODEL_TBL:

	ID	MODEL
1	0	{"BinModel":{"BinEnd":[11.0,21.0,31.0,41.0],"BinStart":[1.0,11.0,21.0,31.0],"Mean": [8.0,13.333...]

3.6.2 Binning Assignment

Binning assignment is used to assign data to the bins previously generated by the Binning algorithm. Therefore it accepts a binning model as input.

It is assumed that the binning model generated in the previous binning stage includes bin starts (s_i) and ends (E_i) of all bins, and therefore new data (d) can be assigned to bin i directly satisfying $s_i \leq d < E_i$ (for the last bin, the less than relation should be less than or equal to).

There is some probability that new data locates too far away from all bins. Here the IQR technique is adopted to justify if a piece of data is an outlier. If new data is lower than both $Q1 - 1.5 * IQR$ and the first bin start, or higher than both $Q3 + 1.5 * IQR$ and the last bin end, it is regarded an outlier and assigned to a virtual bin of index -1 without smoothing.

The new data will not update the binning model.

Note that for the cast of Equal Number Per Bin strategy, the new data assignment may violate its original binning properties.

Prerequisites

- No missing or null data in the inputs.
- Data types must be identical to those in the binning procedure.
- The data types of the ID columns in the data input table and the result output table must be identical.

BINNINGASSIGNMENT

This function directly assigns data to bins based on the previous binning model, without running binning procedure thoroughly.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'BINNINGASSIGNMENT',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Binning Model INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN

Position	Schema Name	Table Type Name	Parameter Type
4	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <binning model input table>, <parameter table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer or double	Raw data	
Binning Model	1st column	Integer	Binning model ID	This must be the first column.
	2nd column	CLOB, varchar, or nvarchar	Binning model saved as JSON string	The table must be a column table. The minimum length of each unit (row) is 5000.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

None.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer	The assigned bin index
	3rd column	Integer or double	Smoothed data

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_BINNING_DATA_T ;
CREATE TYPE PAL_BINNING_DATA_T AS TABLE("ID" INT, "DATA" DOUBLE) ;
DROP TYPE PAL_CONTROL_T ;
CREATE TYPE PAL_CONTROL_T AS TABLE( "NAME" VARCHAR (50), "INTARGS"
INTEGER,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR (100));
DROP TYPE PAL_BINNING_RESULT_T ;
CREATE TYPE PAL_BINNING_RESULT_T AS TABLE("ID" INT, "BIN_NUMBER" INT, "SMOOTH"
DOUBLE);
DROP TYPE PAL_BINNING_MODEL_T ;
CREATE TYPE PAL_BINNING_MODEL_T AS TABLE("JID" INT, "JSMODEL" VARCHAR(5000));
DROP TABLE PAL_BINNING_PDATA_TBL;
CREATE COLUMN TABLE PAL_BINNING_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_BINNING_PDATA_TBL VALUES (1,'DM_PAL', 'PAL_BINNING_DATA_T','IN');
INSERT INTO PAL_BINNING_PDATA_TBL VALUES (2,'DM_PAL', 'PAL_CONTROL_T','IN');
INSERT INTO PAL_BINNING_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_BINNING_RESULT_T','OUT');
INSERT INTO PAL_BINNING_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_BINNING_MODEL_T','OUT');
CALL_SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'BINNINGASSIGNMENT_PROC');
CALL_SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','BINNING', 'DM_PAL',
'BINNINGASSIGNMENT_PROC',PAL_BINNING_PDATA_TBL);
DROP TABLE PAL_BINNING_DATA_TBL;
CREATE COLUMN TABLE PAL_BINNING_DATA_TBL LIKE PAL_BINNING_DATA_T ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (0, 6.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (1, 12.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (2, 13.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (3, 15.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (4, 10.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (5, 23.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (6, 24.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (7, 30.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (8, 32.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (9, 25.0) ;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (10, 38.0) ;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR
(50), "INTARGS" INTEGER,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('BINNING_METHOD',1, NULL , NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SMOOTH_METHOD',0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BIN_NUMBER',4, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('BIN_DISTANCE',10, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SD',1, NULL, NULL);
DROP TABLE PAL_BINNING_RESULT_TBL;
CREATE COLUMN TABLE PAL_BINNING_RESULT_TBL LIKE PAL_BINNING_RESULT_T ;
DROP TABLE PAL_BINNING_MODEL_TBL;
CREATE COLUMN TABLE PAL_BINNING_MODEL_TBL LIKE PAL_BINNING_MODEL_T ;
CALL_DM_PAL.BINNINGASSIGNMENT_PROC(PAL_BINNING_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_BINNING_RESULT_TBL, PAL_BINNING_MODEL_TBL)WITH OVERVIEW;
----- BINNING ASSIGN -----
DROP TYPE PAL_BINNING_ASSIGNED_T;
CREATE TYPE PAL_BINNING_ASSIGNED_T AS TABLE(
"ID" INTEGER,
"BIN_NUMER" INTEGER,
"SMOOTH" DOUBLE
);
```

```

DROP TABLE PAL_BINNINGASSIGNMENT_PDATA_TBL;
CREATE COLUMN TABLE PAL_BINNINGASSIGNMENT_PDATA_TBL("POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE"
VARCHAR(7));
INSERT INTO PAL_BINNINGASSIGNMENT_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_BINNING_DATA_T', 'IN');
INSERT INTO PAL_BINNINGASSIGNMENT_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_BINNING_MODEL_T', 'IN');
INSERT INTO PAL_BINNINGASSIGNMENT_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_BINNINGASSIGNMENT_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_BINNING_ASSIGNED_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL',
'PAL_BINNINGASSIGNMENT_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'BINNINGASSIGNMENT',
'DM_PAL', 'PAL_BINNINGASSIGNMENT_PROC', PAL_BINNINGASSIGNMENT_PDATA_TBL);
DROP TABLE PAL_BINNING_DATA_TBL;
CREATE COLUMN TABLE PAL_BINNING_DATA_TBL LIKE PAL_BINNING_DATA_T;
INSERT INTO PAL_BINNING_DATA_TBL VALUES (0, 6);
INSERT INTO PAL_BINNING_DATA_TBL VALUES (1, 67);
INSERT INTO PAL_BINNING_DATA_TBL VALUES (2, 4);
INSERT INTO PAL_BINNING_DATA_TBL VALUES (3, 12);
INSERT INTO PAL_BINNING_DATA_TBL VALUES (4, -2);
INSERT INTO PAL_BINNING_DATA_TBL VALUES (5, 40);
DROP TABLE PAL_BINNING_ASSIGNED_TBL;
CREATE COLUMN TABLE PAL_BINNING_ASSIGNED_TBL LIKE PAL_BINNING_ASSIGNED_T;
CALL "DM_PAL".PAL_BINNINGASSIGNMENT_PROC(PAL_BINNING_DATA_TBL,
PAL_BINNING_MODEL_TBL, #PAL_CONTROL_TBL, PAL_BINNING_ASSIGNED_TBL) with OVERVIEW;
SELECT * FROM PAL_BINNING_ASSIGNED_TBL;

```

Expected Result

PAL_BINNING_ASSIGNED_TBL:

	ID	BIN_NUMER	SMOOTH
1	0	1	8
2	1	-1	67
3	2	1	8
4	3	2	13.3333...
5	4	1	8
6	5	4	35

Related Information

[Binning \[page 423\]](#)

3.6.3 Convert Category Type to Binary Vector

This function converts category type to binary vector with numerical columns.

Assume that you have a Gender attribute which has two distinct values: Female and Male. You can convert it into:

Gender	Gender_1	Gender_2
Female	1	0
Male	0	1
Female	1	0

Prerequisites

- The input data must contain an ID column, and the ID column must be the first column.
- The other columns of the input table must be of the integer, varchar, or nvarchar type.
- The input data does not contain any null value.

CONV2BINARYVECTOR

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'CONV2BINARYVECTOR',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records.

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer, varchar, or nvarchar	Attribute data	

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
OUT_PUT_COLUMNS	Integer	Number of output columns.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer, varchar, or nvarchar	IDs of original tuples
	Other columns	Integer	Binary vectors

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_CONV2BINARY_DATA_T;
CREATE TYPE PAL_CONV2BINARY_DATA_T AS TABLE(
    "ID" INTEGER,
    "REGION" VARCHAR(50),
    "SALESPERIOD" VARCHAR(50),
    "REVENUE" INTEGER,
    "CLASSLABEL" VARCHAR(50)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_CONV2BINARY_RESULT_T;
CREATE TYPE PAL_CONV2BINARY_RESULT_T AS TABLE(

```

```

    "ID" INTEGER,
    "V0" INTEGER,
    "V1" INTEGER,
    "V2" INTEGER,
    "V3" INTEGER,
    "V4" INTEGER,
    "V5" INTEGER,
    "V6" INTEGER,
    "V7" INTEGER,
    "V8" INTEGER,
    "V9" INTEGER,
    "V10" INTEGER,
    "V11" INTEGER,
    "V12" INTEGER,
    "V13" INTEGER
);
DROP TABLE PAL_CONV2BINARY_PDATA_TBL;
CREATE COLUMN TABLE PAL_CONV2BINARY_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_CONV2BINARY_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_CONV2BINARY_DATA_T', 'IN');
INSERT INTO PAL_CONV2BINARY_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T',
'IN');
INSERT INTO PAL_CONV2BINARY_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_CONV2BINARY_RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_CONV2BINARY_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CONV2BINARYVECTOR',
'DM_PAL', 'PAL_CONV2BINARY_PROC', 'PAL_CONV2BINARY_PDATA_TBL');
DROP TABLE PAL_CONV2BINARY_DATA_TBL;
CREATE COLUMN TABLE PAL_CONV2BINARY_DATA_TBL LIKE PAL_CONV2BINARY_DATA_T;
INSERT INTO PAL_CONV2BINARY_DATA_TBL VALUES (1, 'South', 'Winter', 1, 'Good');
INSERT INTO PAL_CONV2BINARY_DATA_TBL VALUES (2, 'North', 'Spring', 2, 'Average');
INSERT INTO PAL_CONV2BINARY_DATA_TBL VALUES (3, 'West', 'Summer', 2, 'Poor');
INSERT INTO PAL_CONV2BINARY_DATA_TBL VALUES (4, 'East', 'Autumn', 3, 'Poor');
INSERT INTO PAL_CONV2BINARY_DATA_TBL VALUES (5, 'West', 'Spring', 3, 'Poor');
INSERT INTO PAL_CONV2BINARY_DATA_TBL VALUES (6, 'East', 'Spring', 1, 'Good');
INSERT INTO PAL_CONV2BINARY_DATA_TBL VALUES (7, 'South', 'Summer', 3, 'Poor');
INSERT INTO PAL_CONV2BINARY_DATA_TBL VALUES (8, 'South', 'Spring', 3, 'Average');
INSERT INTO PAL_CONV2BINARY_DATA_TBL VALUES (9, 'North', 'Winter', 2, 'Average');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('OUT_PUT_COLUMNS', 15, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
DROP TABLE PAL_CONV2BINARY_RESULT_TBL;
CREATE COLUMN TABLE PAL_CONV2BINARY_RESULT_TBL LIKE PAL_CONV2BINARY_RESULT_T;
CALL "DM_PAL".PAL_CONV2BINARY_PROC(PAL_CONV2BINARY_DATA_TBL, #PAL_CONTROL_TBL,
PAL_CONV2BINARY_RESULT_TBL) with OVERVIEW;
SELECT * FROM PAL_CONV2BINARY_RESULT_TBL;

```

Expected Result

ID	V0	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
1	1	0	0	0	1	0	0	0	1	0	0	1	0	0
2	0	1	0	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	0	0	0	1	0	0	1	0	0	0	1
4	0	0	0	1	0	0	0	1	0	0	1	0	0	1
5	0	0	1	0	0	1	0	0	0	0	1	0	0	1
6	0	0	0	1	0	1	0	0	1	0	0	1	0	0
7	1	0	0	0	0	0	1	0	0	0	1	0	0	1
8	1	0	0	0	0	1	0	0	0	0	1	0	1	0
9	0	1	0	0	1	0	0	0	0	1	0	0	1	0

3.6.4 Inter-Quartile Range Test

Given a series of numeric data, the inter-quartile range (IQR) is the difference between the third quartile (Q3) and the first quartile (Q1) of the data.

$$\text{IQR} = \text{Q3} - \text{Q1}$$

Q1 is equal to 25th percentile and Q3 is equal to 75th percentile.

The p-th percentile of a numeric vector is a number, which is greater than or equal to p% of all the values of this numeric vector.

IQR Test is a method to test the outliers of a series of numeric data. The algorithm performs the following tasks:

1. Calculates Q1, Q3, and IQR.
2. Set upper and lower bounds as follows:
 $\text{Upper-bound} = \text{Q3} + 1.5 \times \text{IQR}$
 $\text{Lower-bound} = \text{Q1} - 1.5 \times \text{IQR}$
3. Tests all the values of a numeric vector to determine if it is in the range. The value outside the range is marked as an outlier, meaning it does not pass the IQR test.

Prerequisites

The input data does not contain null value.

IQRTEST

This function performs the inter-quartile range test and outputs the test results.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'IQTTEST', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<IQR OUTPUT table type>	OUT
4	<schema_name>	<Test OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <IQR output table>, <test output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Data that needs to be tested

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
MULTIPLIER	Double	1.5	The multiplier used in the IQR test.

Output Tables

Table	Column	Column Data Type	Description
IQR Values	1st column	Double	Q1 value
	2nd column	Double	Q3 value

Table	Column	Column Data Type	Description
Test Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer	Test result: <ul style="list-style-type: none">• 0: a value is in the range• 1: a value is out of range

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_IQR_DATA_T;
CREATE TYPE PAL_IQR_DATA_T AS TABLE(
    "ID" VARCHAR(10),
    "VAL" DOUBLE
);
DROP TYPE PAL_IQR_CONTROL_T;
CREATE TYPE PAL_IQR_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_IQR_QQ_T;
CREATE TYPE PAL_IQR_QQ_T AS TABLE(
    "Q1" DOUBLE,
    "Q3" DOUBLE
);
DROP TYPE PAL_IQR_RESULT_T;
CREATE TYPE PAL_IQR_RESULT_T AS TABLE(
    "ID" VARCHAR(10),
    "TEST" INTEGER
);
DROP TABLE PAL_IQR_PDATA_TBL;
CREATE COLUMN TABLE PAL_IQR_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_IQR_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_IQR_DATA_T', 'IN');
INSERT INTO PAL_IQR_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_IQR_CONTROL_T', 'IN');
INSERT INTO PAL_IQR_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_IQR_QQ_T', 'OUT');
INSERT INTO PAL_IQR_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_IQR_RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_IQR_PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'IQRTEST', 'DM_PAL',
'PAL_IQR_PROC', PAL_IQR_PDATA_TBL);
DROP TABLE PAL_IQR_TESTDT_TBL;
CREATE COLUMN TABLE PAL_IQR_TESTDT_TBL LIKE PAL_IQR_DATA_T;
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P1', 10);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P2', 11);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P3', 10);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P4', 9);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P5', 10);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P6', 24);

```

```

INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P7', 11);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P8', 12);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P9', 10);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P10', 9);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P11', 1);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P12', 11);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P13', 12);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P14', 13);
INSERT INTO PAL_IQR_TESTDT_TBL VALUES ('P15', 12);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MULTIPLIER', null, 1.5, null);
DROP TABLE PAL_IQR_QQ_TBL;
CREATE COLUMN TABLE PAL_IQR_QQ_TBL LIKE PAL_IQR_QQ_T;
DROP TABLE PAL_IQR_RESULTS_TBL;
CREATE COLUMN TABLE PAL_IQR_RESULTS_TBL LIKE PAL_IQR_RESULT_T;
CALL "DM_PAL".PAL_IQR_PROC(PAL_IQR_TESTDT_TBL, "#PAL_CONTROL_TBL",
PAL_IQR_QQ_TBL, PAL_IQR_RESULTS_TBL) with overview;
SELECT * FROM PAL_IQR_QQ_TBL;
SELECT * FROM PAL_IQR_RESULTS_TBL;

```

Expected Result

PAL_IQR_TBL:

	Q1	Q3	
	10.0	12.0	

PAL_IQR_RESULTS_TBL:

ID	TEST
P1	0
P2	0
P3	0
P4	0
P5	0
P6	1
P7	0
P8	0
P9	0
P10	0
P11	1
P12	0
P13	0
P14	0
P15	0

3.6.5 Partition

The algorithm partitions an input dataset randomly into three disjoint subsets called training, testing, and validation set. The proportion of each subset is defined as a parameter. Let us remark that the union of these three subsets might not be the complete initial dataset.

Two different partitions can be obtained:

1. **Random Partition**, which randomly divides all the data.
2. **Stratified Partition**, which divides each subpopulation randomly.

In the second case, the dataset needs to have at least one categorical attribute (for example, of type varchar). The initial dataset will first be subdivided according to the different categorical values of this attribute. Each mutually exclusive subset will then be randomly split to obtain the training, testing, and validation subsets. This ensures that all "categorical values" or "strata" will be present in the sampled subset.

Prerequisites

- There is no missing or null data in the column used for stratification.
- The column used for stratification must be categorical (integer, varchar, or nvarchar).

PARTITION

This function reads the input data and generates training, testing, and validation data with the partition algorithm.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'PARTITION',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output
table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Input Data	1st column	Varchar, nvarchar, or integer	ID	Key of the table
	Other columns	Varchar, nvarchar, integer, or double	Data columns	The column used for stratification must be categorical (integer, varchar, or nvarchar).

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
THREAD_NUMBER	Integer	1	Number of threads.	
PARTITION_METHOD	Integer	0	Indicates the partition method: <ul style="list-style-type: none"> • 0: Random partition • Not 0: Stratified partition 	
RANDOM_SEED	Integer	0	Indicates the seed used to initialize the random number generator. <ul style="list-style-type: none"> • 0: Uses the system time • Not 0: Uses the specified seed 	
STRATIFIED_COLUMN	Varchar	No default value	Indicates which column is used for stratification.	Valid only when PARTITION_METHOD is set to a non-zero value (stratified partition).
TRAINING_PERCENT	Double	0.8	The percentage of training data. Value range: $0 \leq \text{value} \leq 1$	
TESTING_PERCENT	Double	0.1	The percentage of testing data. Value range: $0 \leq \text{value} \leq 1$	

Name	Data Type	Default Value	Description	Dependency
VALIDATION_PERCENT	Double	0.1	The percentage of validation data. Value range: $0 \leq \text{value} \leq 1$	
TRAINING_SIZE	Integer	No default value	Row size of training data. Value range: ≥ 0	If both TRAINING_PERCENT and TRAINING_SIZE are specified, TRAINING_PERCENT takes precedence.
TESTING_SIZE	Integer	No default value	Row size of testing data. Value range: ≥ 0	If both TESTING_PERCENT and TESTING_SIZE are specified, TESTING_PERCENT takes precedence.
VALIDATION_SIZE	Integer	No default value	Row size of validation data. Value range: ≥ 0	If both VALIDATION_PERCENT and VALIDATION_SIZE are specified, VALIDATION_PERCENT takes precedence.

Output Table

Table	Column	Column Data Type	Description	Constraint
Output Table	1st column	Varchar, nvarchar, or integer	ID	Key of the table
	2nd column	Integer	Indicates which partition the table belongs to.	<ul style="list-style-type: none"> • 1: training data • 2: testing data • 3: validation data

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_PARTITION_DATA_T;
CREATE TYPE PAL_PARTITION_DATA_T AS TABLE (
    "ID" INTEGER,
    "HomeOwner" VARCHAR (100),
    "MaritalStatus" VARCHAR (100),
    "AnnualIncome" DOUBLE,
    "DefaultedBorrower" VARCHAR (100)
);
DROP TYPE PAL_CONTROL_T;

```

```

CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_PARTITION_OUT_T;
CREATE TYPE PAL_PARTITION_OUT_T AS TABLE(
    "ID" INTEGER,
    "TYPE" INTEGER
);
DROP table PAL_PARTITION_PDATA_TBL;
CREATE COLUMN TABLE PAL_PARTITION_PDATA_TBL("POSITION" INT,"SCHEMA_NAME"
NVARCHAR(256),"TYPE_NAME" NVARCHAR (256),"PARAMETER_TYPE" VARCHAR (7));
insert into PAL_PARTITION_PDATA_TBL values
(1,'DM_PAL','PAL_PARTITION_DATA_T','IN');
insert into PAL_PARTITION_PDATA_TBL values (2,'DM_PAL','PAL_CONTROL_T','IN');
insert into PAL_PARTITION_PDATA_TBL values
(3,'DM_PAL','PAL_PARTITION_OUT_T','OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_PARTITION_PROC');
call
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','PARTITION','DM_PAL','PAL_PARTITION_
PROC',PAL_PARTITION_PDATA_TBL);
DROP TABLE PAL_PARTITION_DATA_TBL;
CREATE COLUMN TABLE PAL_PARTITION_DATA_TBL LIKE PAL_PARTITION_DATA_T;
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (0, 'YES','Single',125,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (1, 'NO','Married',100,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (2, 'NO','Single',70,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (3, 'YES','Married',120,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (4, 'NO','Divorced',95,'YES');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (5, 'NO','Married',60,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (6, 'YES','Divorced',220,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (7, 'NO','Single',85,'YES');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (8, 'NO','Married',75,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (9, 'NO','Single',90,'YES');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (10, 'YES','Single',125,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (11, 'NO','Married',100,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (12, 'NO','Single',70,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (13, 'YES','Married',120,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (14, 'NO','Divorced',95,'YES');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (15, 'NO','Married',60,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (16, 'YES','Divorced',220,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (17, 'NO','Single',85,'YES');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (18, 'NO','Married',75,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (19, 'NO','Single',90,'YES');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (20, 'YES','Single',125,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (21, 'NO','Married',100,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (22, 'NO','Single',70,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (23, 'YES','Married',120,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (24, 'NO','Divorced',95,'YES');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (25, 'NO','Married',60,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (26, 'YES','Divorced',220,'NO');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (27, 'NO','Single',85,'YES');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (28, 'NO','Married',75,'YES');
INSERT INTO PAL_PARTITION_DATA_TBL VALUES (29, 'NO','Single',90,'YES');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',1,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PARTITION_METHOD',0,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('RANDOM SEED',23,null,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TRAINING_PERCENT', null,0.6,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TESTING_PERCENT', null,0.2,null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('VALIDATION_PERCENT', null,0.2,null);
DROP TABLE PAL_PARTITION_OUT_TBL;
CREATE COLUMN TABLE PAL_PARTITION_OUT_TBL LIKE PAL_PARTITION_OUT_T;
CALL DM_PAL.PAL_PARTITION_PROC(PAL_PARTITION_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_PARTITION_OUT_TBL) with overview;
select * from PAL_PARTITION_OUT_TBL;

```

Expected Result

	ID	TYPE
1	11	1
2	3	1
3	15	1
4	14	1
5	29	1
6	27	1
7	19	1
8	9	1
9	28	1
10	2	1
11	17	1
12	21	1
13	20	1
14	6	1
15	4	1
16	12	1
17	23	1
18	8	1
19	18	2
20	13	2
21	1	2
22	0	2
23	16	2
24	10	2
25	25	3
26	7	3
27	24	3
28	5	3
29	26	3
30	22	3

3.6.6 Posterior Scaling

Posterior scaling is used to scale data based on the previous scaling model generated by the scaling range procedure.

It is assumed that new data is from similar distribution and will not update the scaling model.

Prerequisites

- No missing or null data in the inputs.
- Data types must be identical to those in the scaling range procedure.
- The data types of the ID columns in the data input table and the result output table must be identical.

POSTERIORSCALING

This function directly scales data based on the previous scaling model, without running the scaling range procedure once more.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'POSTERIORSCALING',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Scaling Model INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <scaling model input table>, <parameter table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.

Table	Column	Column Data Type	Description	Constraint
	Other columns	Integer or double	Raw data	
Scaling Model	1st column	Integer	Scaling model ID	This must be the first column.
	2nd column	CLOB, varchar, or nvarchar	Scaling model saved as JSON string	The table must be a column table. The minimum length of each unit (row) is 5000.

Parameter Table

Mandatory Parameters

None.

Optional Parameters

None.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	Other columns	Integer or double	Scaled data

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL_SYS_AFL_AFLPAL_EXECUTE or AFL_SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SCALING_DATA_T ;
CREATE TYPE PAL_SCALING_DATA_T AS TABLE("ID" INT, "X1" DOUBLE, "X2" DOUBLE) ;
DROP TYPE PAL_CONTROL_T ;
CREATE TYPE PAL_CONTROL_T AS TABLE( "NAME" VARCHAR (100), "INTARGS"
INTEGER,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR (100));
DROP TYPE PAL_SCALING_RESULT_T ;
CREATE TYPE PAL_SCALING_RESULT_T AS TABLE("ID" INT, "PRE_X1" DOUBLE, "PRE_X2"
DOUBLE);
DROP TYPE PAL_SCALING_MODEL_T ;
CREATE TYPE PAL_SCALING_MODEL_T AS TABLE("JID" INT, "JSMODEL" VARCHAR(5000));
DROP TABLE PAL_SCALING_PDATA_TBL;
CREATE COLUMN TABLE PAL_SCALING_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_SCALING_PDATA_TBL VALUES (1,'DM_PAL', 'PAL_SCALING_DATA_T','IN');
INSERT INTO PAL_SCALING_PDATA_TBL VALUES (2,'DM_PAL', 'PAL_CONTROL_T','IN');

```

```

INSERT INTO PAL_SCALING_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_SCALING_RESULT_T','OUT');
INSERT INTO PAL_SCALING_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_SCALING_MODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'SCALINGRANGE_TEST_PROC');
CALL SYS.AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'SCALINGRANGE', 'DM_PAL',
'SCALINGRANGE_TEST_PROC', PAL_SCALING_PDATA_TBL);
DROP TABLE PAL_SCALING_DATA_TBL;
CREATE COLUMN TABLE PAL_SCALING_DATA_TBL LIKE PAL_SCALING_DATA_T ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (0, 6.0, 9.0) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (1, 12.1, 8.3) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (2, 13.5, 15.3) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (3, 15.4, 18.7) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (4, 10.2, 19.8) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (5, 23.3, 20.6) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (6, 24.4, 24.3) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (7, 30.6, 25.3) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (8, 32.5, 27.6) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (9, 25.6, 28.5) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (10, 38.7, 29.4) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (11, 38.7, 29.4) ;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR
(100), "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('SCALING_METHOD',1,NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('Z-SCORE_METHOD',0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD NUMBER',2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NEW_MAX', NULL,1.0, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NEW_MIN', NULL,0.0, NULL);
DROP TABLE PAL_SCALING_RESULT_TBL;
CREATE COLUMN TABLE PAL_SCALING_RESULT_TBL LIKE PAL_SCALING_RESULT_T ;
DROP TABLE PAL_SCALING_MODEL_TBL;
CREATE COLUMN TABLE PAL_SCALING_MODEL_TBL LIKE PAL_SCALING_MODEL_T;
CALL DM_PAL.SCALINGRANGE_TEST_PROC(PAL_SCALING_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_SCALING_RESULT_TBL, PAL_SCALING_MODEL_TBL) WITH OVERVIEW;
----- POSTERIOR SCALING -----
DROP TYPE PAL_NEW_SCALING_T;
CREATE TYPE PAL_NEW_SCALING_T AS TABLE(
"ID" INTEGER,
"S_X1" DOUBLE,
"S_X2" DOUBLE
);
DROP TABLE PAL_POSTERIORSCLALING_PDATA_TBL;
CREATE COLUMN TABLE PAL_POSTERIORSCLALING_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_POSTERIORSCLALING_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_SCALING_DATA_T', 'IN');
INSERT INTO PAL_POSTERIORSCLALING_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_SCALING_MODEL_T', 'IN');
INSERT INTO PAL_POSTERIORSCLALING_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CONTROL_T',
'IN');
INSERT INTO PAL_POSTERIORSCLALING_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_NEW_SCALING_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_POSTERIORSCLALING_PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'POSTERIORSCLALING',
'DM_PAL', 'PAL_POSTERIORSCLALING_PROC', PAL_POSTERIORSCLALING_PDATA_TBL);
DROP TABLE PAL_SCALING_DATA_TBL;
CREATE COLUMN TABLE PAL_SCALING_DATA_TBL LIKE PAL_SCALING_DATA_T;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (0 ,6, 9);
INSERT INTO PAL_SCALING_DATA_TBL VALUES (1 , 6,7);
INSERT INTO PAL_SCALING_DATA_TBL VALUES (2, 4, 4);
INSERT INTO PAL_SCALING_DATA_TBL VALUES (3, 1, 2);
INSERT INTO PAL_SCALING_DATA_TBL VALUES (4, 9,-2);
INSERT INTO PAL_SCALING_DATA_TBL VALUES (5, 4, 5.0);
DROP TABLE PAL_NEW_SCALING_TBL;
CREATE COLUMN TABLE PAL_NEW_SCALING_TBL LIKE PAL_NEW_SCALING_T;
CALL "DM_PAL".PAL_POSTERIORSCLALING_PROC(PAL_SCALING_DATA_TBL,
PAL_SCALING_MODEL_TBL, #PAL_CONTROL_TBL, PAL_NEW_SCALING_TBL) with OVERVIEW;

```

```
SELECT * FROM PAL_NEW_SCALING_TBL;
```

Expected Result

PAL_NEW_SCALING_TBL:

	ID	S_X1	S_X2
1	0	-1.4873873740932577	-1.6536838199986237
2	1	-1.4873873740932577	-1.9214868677716803
3	2	-1.6667707759939523	-2.3231914394312647
4	3	-1.935845878844994	-2.5909944872043216
5	4	-1.218312271242216	-3.1266005827504344
6	5	-1.6667707759939523	-2.1892899155447365

Related Information

[Scaling Range \[page 470\]](#)

3.6.7 Principal Component Analysis (PCA)

Principal component analysis (PCA) aims at reducing the dimensionality of multivariate data while accounting for as much of the variation in the original data set as possible. This technique is especially useful when the variables within the data set are highly correlated.

Principal components seek to transform the original variables to a new set of variables that are:

- linear combinations of the variables in the data set;
- uncorrelated with each other;
- ordered according to the amount of variations of the original variables that they explain.

The signs of the columns of the loadings matrix are arbitrary, and may differ between different implementations for PCA.

Note that if there exists one variable which has constant value across data items, you cannot scale variables any more.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

PCA

This is a principal component analysis function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PCA', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Loadings OUTPUT table type>	OUT
4	<schema_name>	<Loadings information OUTPUT table type>	OUT
5	<schema_name>	<Scores OUTPUT table type>	OUT
6	<schema_name>	<Scaling information OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>,  
<loadings output table>, <loadings information output table>, <scores output  
table>, <Scaling information output table>) WITH OVERVIEW;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	Data item ID
	Other columns	Double	Dimensionality Xn

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads.
SCALING	Integer	0	Specifies whether the variables should be scaled to have unit variance before the analysis takes place. <ul style="list-style-type: none"> • 0: No • 1: Yes
SCORES	Integer	0	Specifies whether to output the scores on each principal component. <ul style="list-style-type: none"> • 0: No • 1: Yes

Output Tables

Table	Column	Column Data Type	Description
Loadings	1st column	Varchar or nvarchar	Principal component ID
	Columns	Double	Transforming weight
Loadings Information	1st column	Varchar or nvarchar	Principal component ID
	2nd column	Double	Standard deviations of the principal components
	3rd column	Double	Variance proportion to the total variance explained by each principal component
	4th column	Double	Cumulative variance proportion to the total variance explained by each principal component
Scores	1st column	Integer, bigint, varchar, or nvarchar	Data item ID
	Other columns	Double	Transformed value based on principal component
Scaling Information	1st column	Integer	Variable ID

Table	Column	Column Data Type	Description
	2nd column	Double	Mean value of each variable
	3rd column	Double	Scale value of each variable

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA "DM_PAL";
DROP TYPE PAL_PCA_DATA_T;
DROP TYPE PAL_CONTROL_T;
DROP TYPE PAL_PCA_LOADINGS_T;
DROP TYPE PAL_PCA_LOADINGS_INFO_T;
DROP TYPE PAL_PCA_SCORES_T;
DROP TYPE PAL_PCA_SCALING_INFO_T;
DROP TABLE PAL_PCA_PDATA_TBL;
DROP TABLE #PAL_CONTROL_TBL;
DROP TABLE PAL_PCA_DATA_TBL;
DROP TABLE PAL_PCA_LOADINGS_TBL;
DROP TABLE PAL_PCA_LOADINGS_INFO_TBL;
DROP TABLE PAL_PCA_SCORES_TBL;
DROP TABLE PAL_PCA_SCALING_INFO_TBL;
CREATE TYPE PAL_PCA_DATA_T AS TABLE("ID" INTEGER, "X1" DOUBLE, "X2" DOUBLE, "X3"
DOUBLE, "X4" DOUBLE, "X5" DOUBLE, "X6" DOUBLE);
CREATE TYPE PAL_CONTROL_T AS TABLE("NAME" VARCHAR(50), "INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
CREATE TYPE PAL_PCA_LOADINGS_T AS TABLE("PCID" VARCHAR(50), "X1_WEIGHT" DOUBLE,
"X2_WEIGHT" DOUBLE, "X3_WEIGHT" DOUBLE, "X4_WEIGHT" DOUBLE, "X5_WEIGHT" DOUBLE,
"X6_WEIGHT" DOUBLE);
CREATE TYPE PAL_PCA_LOADINGS_INFO_T AS TABLE("PCID" VARCHAR(50), "SDEV" DOUBLE,
"PROPORTION" DOUBLE, "CUM_PROPORTION" DOUBLE);
CREATE TYPE PAL_PCA_SCORES_T AS TABLE("ID" INTEGER, "Comp1" DOUBLE, "Comp2"
DOUBLE, "Comp3" DOUBLE, "Comp4" DOUBLE, "Comp5" DOUBLE, "Comp6" DOUBLE);
CREATE TYPE PAL_PCA_SCALING_INFO_T AS TABLE("ID" INTEGER, "MEAN_VECTOR" DOUBLE,
"SCALE_VECTOR" DOUBLE);
CREATE COLUMN TABLE PAL_PCA_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_PCA_PDATA_TBL VALUES(1, 'DM_PAL', 'PAL_PCA_DATA_T', 'IN');
INSERT INTO PAL_PCA_PDATA_TBL VALUES(2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_PCA_PDATA_TBL VALUES(3, 'DM_PAL', 'PAL_PCA_LOADINGS_T', 'OUT');
INSERT INTO PAL_PCA_PDATA_TBL VALUES(4, 'DM_PAL', 'PAL_PCA_LOADINGS_INFO_T',
'OUT');
INSERT INTO PAL_PCA_PDATA_TBL VALUES(5, 'DM_PAL', 'PAL_PCA_SCORES_T', 'OUT');
INSERT INTO PAL_PCA_PDATA_TBL VALUES(6, 'DM_PAL', 'PAL_PCA_SCALING_INFO_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_PCA_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PCA', 'DM_PAL', 'PAL_PCA_PROC', PAL_P
CA_PDATA_TBL);
CREATE COLUMN TABLE PAL_PCA_DATA_TBL LIKE PAL_PCA_DATA_T;
INSERT INTO PAL_PCA_DATA_TBL VALUES (1, 12, 52, 20, 44, 48, 16);
INSERT INTO PAL_PCA_DATA_TBL VALUES (2, 12, 57, 25, 45, 50, 16);
INSERT INTO PAL_PCA_DATA_TBL VALUES (3, 12, 54, 21, 45, 50, 16);
INSERT INTO PAL_PCA_DATA_TBL VALUES (4, 13, 52, 21, 46, 51, 17);
INSERT INTO PAL_PCA_DATA_TBL VALUES (5, 14, 54, 24, 46, 51, 17);
INSERT INTO PAL_PCA_DATA_TBL VALUES (6, 22, 52, 25, 54, 58, 26);

```

```

INSERT INTO PAL_PCA_DATA_TBL VALUES (7, 22, 56, 26, 55, 58, 27);
INSERT INTO PAL_PCA_DATA_TBL VALUES (8, 17, 52, 21, 45, 52, 17);
INSERT INTO PAL_PCA_DATA_TBL VALUES (9, 15, 53, 24, 45, 53, 18);
INSERT INTO PAL_PCA_DATA_TBL VALUES (10, 23, 54, 23, 53, 57, 24);
INSERT INTO PAL_PCA_DATA_TBL VALUES (11, 25, 54, 23, 55, 58, 25);
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('SCALING', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SCORES', 1, NULL, NULL);
CREATE COLUMN TABLE PAL_PCA_LOADINGS_TBL LIKE PAL_PCA_LOADINGS_T;
CREATE COLUMN TABLE PAL_PCA_LOADINGS_INFO_TBL LIKE PAL_PCA_LOADINGS_INFO_T;
CREATE COLUMN TABLE PAL_PCA_SCORES_TBL LIKE PAL_PCA_SCORES_T;
CREATE COLUMN TABLE PAL_PCA_SCALING_INFO_TBL LIKE PAL_PCA_SCALING_INFO_T;
CALL DM_PAL.PAL_PCA_PROC(PAL_PCA_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_PCA_LOADINGS_TBL, PAL_PCA_LOADINGS_INFO_TBL, PAL_PCA_SCORES_TBL,
PAL_PCA_SCALING_INFO_TBL) WITH OVERVIEW;
SELECT * FROM PAL_PCA_LOADINGS_TBL;
SELECT * FROM PAL_PCA_LOADINGS_INFO_TBL;
SELECT * FROM PAL_PCA_SCORES_TBL;
SELECT * FROM PAL_PCA_SCALING_INFO_TBL;

```

Expected Result

PAL_PCA_LOADINGS_TBL:

	PCID	X1_WEIGHT	X2_WEIGHT	X3_WEIGHT	X4_WEIGHT	X5_WEIGHT	X6_WEIGHT
1	Comp1	0.4529491452815981	0.13899936408534205	0.32821054855366527	0.4704875914009597	0.4720970616704374	0.47282118722674094
2	Comp2	0.23943986612392362	-0.7836762522372014	-0.5313532760455469	0.10914669994079693	0.13752463728748548	0.12392639047699265
3	Comp3	0.20976200429503897	0.5872632991924558	-0.7459306160756267	0.21932229761535177	-0.08099550837568...	0.006833181429103...
4	Comp4	-0.6301652184225306	-0.07120016929438...	-0.04735043512438...	0.495733973111604	-0.32829579116186...	0.49198574276584134
5	Comp5	-0.5353844722069163	0.09669986295964525	-0.21144789449537...	-0.17349821839060...	0.7931316257692917	0.011957581554267...
6	Comp6	-0.09879670168361...	-0.08504233228783...	0.08109729996619797	0.6654114141767854	0.1217175576242614	-0.7203073839986038

PAL_PCA_LOADINGS_INFO_TBL:

	PCID	SDEV	PROPORTION	CUM_PROPORTION
1	Comp1	2.0708452571912686	0.7147333465385953	0.7147333465385953
2	Comp2	1.1528557732691938	0.2215127389933518	0.9362460855319471
3	Comp3	0.5394023255908552	0.04849247814213...	0.9847385636740843
4	Comp4	0.2589039436293156	0.01117187533780...	0.9959104390118863
5	Comp5	0.12492794025277...	0.00260116504263...	0.9985116040545197
6	Comp6	0.09450066493353...	0.00148839594548...	0.9999999999999999

PAL_PCA_SCORES_TBL:

	ID	Comp1	Comp2	Comp3	Comp4	Comp5	Comp6
1	1	-2.5949626091856435	0.9147311780565773	0.23951091852440537	0.3250302874052749	-0.1987146574821934	-0.12613778494245287
2	2	-1.01161789154787	-2.636463041223107	0.11677373744675212	-0.07169856174942868	-0.0577490948903352	0.032425398635060776
3	3	-1.9148505995784366	-0.1822364665066...	0.5658719191081284	0.1494275522203526	0.1934433981164605	0.021234619236276453
4	4	-1.658801849476119	0.8802468877316018	-0.06054198923857...	0.23678081666380119	0.1484581615561475	0.1183892624803024
5	5	-0.9120712600039886	-0.7969536351865...	-0.44264179023409...	-0.04356505124817178	-0.16047523969849...	0.1197628611727169
6	6	2.431462002709937	0.9343813794892657	-0.9377010351991641	0.23735188588875805	-0.03249811943076...	0.04145976052255662
7	7	3.1301641330543872	-1.1358332631687...	0.12830532211380258	0.260281100358304	0.056089050564692...	-0.13552127667271385
8	8	-1.2751684705213444	1.0833112262905538	0.03734940710462245	-0.45691684602976484	-0.02885468630843...	-0.07067677490669377
9	9	-0.6506761797485403	-0.2087287686314...	-0.837171042552696	-0.29839016918884376	0.136650986348393...	-0.08700390358230314
10	10	1.9224981962744239	0.470893281997426	0.5163754694212783	-0.16140995648372136	0.008832590885013...	-0.016873883984317...
11	11	2.5340245280231963	0.6766512211510356	0.6738690835055563	-0.17689105783656067	-0.06518238966049...	0.10294172204156993

PAL_PCA_SCALING_INFO_TBL:

	ID	MEAN_VECTOR	SCALE_VECTOR
1	1	17	5.039841267341661
2	2	53.63636363636363	1.6895400127092153
3	3	23	2
4	4	48.45454545454545	4.655397649259112
5	5	53.27272727272727	3.7707004413214555
6	6	19.90909090909091	4.5266885347800425

PCAPROJECTION

This is a principal component analysis projection function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PCAPROJECTION',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Loadings INPUT table type>	IN
4	<schema_name>	<Scaling information table type>	IN
5	<schema_name>	<Projected data OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>,
<loadings input table>, <Scaling information table>, <projected data output
table>) WITH OVERVIEW;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer	Data item ID	
	Other columns	Double	Dimensionality Xn	
Loadings	1st column	Integer	Principal component ID	This is a full model, which means that it is a square matrix and the dimension is equal to the number of variables in input data.
	Other columns	Double	Transformed value based on principal component	
Scaling Information	1st column	Integer	Variable ID	Row number of this table should be equal to the number of variables in input data.
	2nd column	Double	Mean value of each variable	
	3rd column	Double	Scale value of each variable	

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads.
SCALING	Integer	0	Specifies whether the variables should be scaled to have unit variance during the projection. The value should be equal to the SCALING parameter value in the PCA process. <ul style="list-style-type: none"> • 0: No • 1: Yes
MAX_COMPONENTS	Integer	Variable number	Specifies the number of components to be retained. Value range: $0 < \text{MAX_COMPONENTS} \leq \text{variable number of original data}$

Output Table

Table	Column	Column Data Type	Description	Constraint
Projection	1st column	Integer	Data item ID	Make sure the output table has enough columns to hold the output data. The column size of output data depends on the MAX_COMPONENTS parameter.
	Columns	Double	Transformed data item	

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.
- Table PAL_PCA_LOADINGS_TBL and table PAL_PCA_SCALING_INFO_TBL are output of PCA in the previous example.

```

DROP TYPE PAL_PCAPROJ_DATA_T;
DROP TYPE PAL_PCAPROJ_CONTROL_T;
DROP TYPE PAL_PCAPROJ_SCORES_T;
CREATE TYPE PAL_PCAPROJ_DATA_T AS TABLE("ID" INTEGER, "X1" DOUBLE, "X2" DOUBLE,
"X3" DOUBLE, "X4" DOUBLE, "X5" DOUBLE, "X6" DOUBLE);
CREATE TYPE PAL_PCAPROJ_CONTROL_T AS TABLE("NAME" VARCHAR(50), "INTARGS"
INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
CREATE TYPE PAL_PCAPROJ_SCORES_T AS TABLE("ID" INTEGER, "Comp1" DOUBLE, "Comp2"
DOUBLE, "Comp3" DOUBLE, "Comp4" DOUBLE, "Comp5" DOUBLE, "Comp6" DOUBLE);
DROP TABLE PAL_PCAPROJ_DATA_TBL;
DROP TABLE PAL_PCAPROJ_CONTROL_TBL;
DROP TABLE PAL_PCAPROJ_SCORES_TBL;
DROP TABLE PAL_PCAPROJ_PDATA_TBL;
CREATE COLUMN TABLE PAL_PCAPROJ_DATA_TBL LIKE PAL_PCAPROJ_DATA_T;
CREATE COLUMN TABLE PAL_PCAPROJ_CONTROL_TBL LIKE PAL_PCAPROJ_CONTROL_T;
CREATE COLUMN TABLE PAL_PCAPROJ_SCORES_TBL LIKE PAL_PCAPROJ_SCORES_T;
CREATE COLUMN TABLE PAL_PCAPROJ_PDATA_TBL ("POSITION" INTEGER, "SCHEMA_NAME"
VARCHAR(100), "TYPE_NAME" VARCHAR(100), "PARAMETER_TYPE" VARCHAR(100));
INSERT INTO PAL_PCAPROJ_PDATA_TBL VALUES(1, 'DM_PAL', 'PAL_PCAPROJ_DATA_T',
'IN');
INSERT INTO PAL_PCAPROJ_PDATA_TBL VALUES(2, 'DM_PAL', 'PAL_PCAPROJ_CONTROL_T',
'IN');
INSERT INTO PAL_PCAPROJ_PDATA_TBL VALUES(3, 'DM_PAL', 'PAL_PCA_LOADINGS_T',
'IN');
INSERT INTO PAL_PCAPROJ_PDATA_TBL VALUES(4, 'DM_PAL', 'PAL_PCA_SCALING_INFO_T',
'IN');
INSERT INTO PAL_PCAPROJ_PDATA_TBL VALUES(5, 'DM_PAL', 'PAL_PCAPROJ_SCORES_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_PCAPROJ_PROC');
CALL
SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PCAPROJECTION', 'DM_PAL', 'PAL_PCAPR
OJ_PROC', 'PAL_PCAPROJ_PDATA_TBL');
INSERT INTO PAL_PCAPROJ_DATA_TBL VALUES(1,2,32,10,54,38,20);
INSERT INTO PAL_PCAPROJ_DATA_TBL VALUES(2,9,57,20,25,48,19);
INSERT INTO PAL_PCAPROJ_DATA_TBL VALUES(3,12,24,28,35,30,20);
INSERT INTO PAL_PCAPROJ_DATA_TBL VALUES(4,15,42,27,36,61,27);

```

```

INSERT INTO PAL_PCAPROJ_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, NULL, NULL);
INSERT INTO PAL_PCAPROJ_CONTROL_TBL VALUES ('SCALING', 1, NULL, NULL);
INSERT INTO PAL_PCAPROJ_CONTROL_TBL VALUES ('MAX_COMPONENTS', 4, NULL, NULL);
CALL DM_PAL.PAL_PCAPROJ_PROC(PAL_PCAPROJ_DATA_TBL, PAL_PCAPROJ_CONTROL_TBL,
PAL_PCA_LOADINGS_TBL, PAL_PCA_SCALING_INFO_TBL, PAL_PCAPROJ_SCORES_TBL) WITH
OVERVIEW;
SELECT * FROM PAL_PCAPROJ_SCORES_TBL;

```

Expected Result

PAL_PCAPROJ_SCORES_TBL:

	ID	Comp1	Comp2	Comp3	Comp4	Comp5	Comp6
1	1	-6.603741527545116	12.352443837611009	-2.7068428225172037	5.025234762683863	?	?
2	2	-4.060068807988197	-1.9103252645918178	0.9619990768389393	-1.207742710356588	?	?
3	3	-6.331078830027881	11.018844535223138	-12.507987030133343	2.35913598150486...	?	?
4	4	-0.0312231746867...	4.423641401385762	-6.361790121224823	-0.5825785566358...	?	?

3.6.8 Random Distribution Sampling

Random distribution sampling is a random generation function with a given distribution.

In PAL, this function supports four different distributions. The probability density functions of different distributions are defined as follows:

- Uniform

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$

- Normal

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Weibull

$$f(x; \lambda, k) = \begin{cases} \frac{k}{\lambda} \left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

- Gamma

$$f(x; k, \theta) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)} \quad \text{for } x > 0 \text{ and } k, \theta > 0$$

DISTRRANDOM

This is a random generation function with a given distribution.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'DISTRANDOM',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Distribution parameter INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<distribution parameter input table>,
<parameter table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Distribution Parameter	1st column	Varchar or nvarchar	<p>Name of distribution parameters. The supported distributions and their parameters are as follows:</p> <ul style="list-style-type: none"> • Uniform <ul style="list-style-type: none"> ◦ Min (default: 0) in $(-\infty, +\infty)$ ◦ Max (default: 1) in $(-\infty, +\infty)$ (Min < Max) • Normal <ul style="list-style-type: none"> ◦ Mean (default: 0) in $(-\infty, +\infty)$ ◦ Variance (default: 1) in $(0, +\infty)$ ◦ SD (default: 1) in $(0, +\infty)$ <p>Variance and SD cannot be used together. Choose one from them.</p> • Weibull <ul style="list-style-type: none"> ◦ Shape (default: 1) in $(0, +\infty)$ ◦ Scale (default: 1) in $(0, +\infty)$ • Gamma <ul style="list-style-type: none"> ◦ Shape (default: 1) in $(0, +\infty)$ ◦ Scale (default: 1) in $(0, +\infty)$
	2nd column	Varchar or nvarchar	Value of distribution parameters

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
NUM_RANDOM	Integer	100	Specifies the number of random data to be generated.

Name	Data Type	Default Value	Description
SEED	Integer	0	<p>Indicates the seed used to initialize the random number generator.</p> <ul style="list-style-type: none"> • 0: Uses the system time • Not 0: Uses the specified seed
THREAD_NUMBER	Integer	1	Specifies the number of threads.

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	2nd column	Double	Defined number of random data to be generated

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DISTRRANDOM_DISTRPARAM_T;
CREATE TYPE PAL_DISTRRANDOM_DISTRPARAM_T AS TABLE(
"NAME" VARCHAR(100),
"VALUE" VARCHAR(100)
);
DROP TYPE PAL_DISTRRANDOM_RESULT_T;
CREATE TYPE PAL_DISTRRANDOM_RESULT_T AS TABLE(
"ID" INTEGER,
"RANDOM" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
"NAME" VARCHAR(100),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_DISTRRANDOM_PDATA_TBL;
CREATE COLUMN TABLE PAL_DISTRRANDOM_PDATA_TBL(
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_DISTRRANDOM_PDATA_TBL VALUES
(1,'DM_PAL','PAL_DISTRRANDOM_DISTRPARAM_T', 'IN');
INSERT INTO PAL_DISTRRANDOM_PDATA_TBL VALUES (2,'DM_PAL','PAL_CONTROL_T', 'IN');

```

```

INSERT INTO PAL_DISTRRANDOM_PDATA_TBL VALUES
(3,'DM_PAL','PAL_DISTRRANDOM_RESULT_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DISTRRANDOM_PROC');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL','DISTRRANDOM',
'DM_PAL','PAL_DISTRRANDOM_PROC', 'PAL_DISTRRANDOM_PDATA_TBL');
DROP TABLE PAL_DISTRRANDOM_DISTRPARAM_TBL;
CREATE COLUMN TABLE PAL_DISTRRANDOM_DISTRPARAM_TBL LIKE
PAL_DISTRRANDOM_DISTRPARAM_T;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TBL VALUES ('DistributionName',
'UNIFORM');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TBL VALUES ('MIN', '2');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TBL VALUES ('MAX', '2.1');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY TABLE #PAL_CONTROL_TBL LIKE PAL_CONTROL_T;
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NUM_RANDOM', 10000, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SEED', 0, null, null);
DROP TABLE PAL_DISTRRANDOM_RESULT_TBL;
CREATE COLUMN TABLE PAL_DISTRRANDOM_RESULT_TBL LIKE PAL_DISTRRANDOM_RESULT_T;
CALL DM_PAL.PAL_DISTRRANDOM_PROC(PAL_DISTRRANDOM_DISTRPARAM_TBL,
#PAL_CONTROL_TBL, PAL_DISTRRANDOM_RESULT_TBL) with overview;
SELECT * FROM PAL_DISTRRANDOM_RESULT_TBL;

```

Expected Result

	ID	RANDOM
1	0	0.44016...
2	1	0.53195...
3	2	0.52984...
4	3	0.57767...
5	4	0.63764...

3.6.9 Sampling

In business scenarios the number of records in the database is usually quite large, and it is common to use a small portion of the records as representatives, so that a rough impression of the dataset can be given by analyzing sampling.

This release of PAL provides eight sampling methods, including:

- First_N
- Middle_N
- Last_N
- Every_Nth
- SimpleRandom_WithReplacement
- SimpleRandom_WithoutReplacement
- Systematic
- Stratified

Prerequisites

The input data does not contain null value.

SAMPLING

This function takes samples from a population.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'SAMPLING',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output
table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	Columns	Integer, double, varchar, or nvarchar	Any data users need

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description	Dependency
SAMPLING_METHOD	Integer	<p>Sampling method:</p> <ul style="list-style-type: none"> • 0: First_N • 1: Middle_N • 2: Last_N • 3: Every_Nth • 4: SimpleRandom_WithReplacement • 5: SimpleRandom_WithoutReplacement • 6: Systematic • 7: Stratified_WithReplacement • 8: Stratified_WithoutReplacement <p>Note: For the random methods (method 4, 5, 6 in the above list), the system time is used for the seed.</p>	
INTERVAL	Integer	The interval between two samples.	Only required when SAMPLING_METHOD is 3. If this parameter is not specified, the SAMPLING_SIZE parameter will be used.
COLUMN_CHOOSE	Integer	The column that is used to do the stratified sampling.	Only required when SAMPLING_METHOD is 7 or 8.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
SAMPLING_SIZE	Integer	1	Number of the samples.	Use this parameter when PERCENTAGE is not set.
PERCENTAGE	Double	0.1	Percentage of the samples. Use this parameter when SAMPLING_SIZE is not set.	If both SAMPLING_SIZE and PERCENTAGE are specified, PERCENTAGE takes precedence.

Output Table

Table	Column	Column Data Type	Description
Result	Columns	Integer, double, varchar, or nvarchar	The Output Table has the same structure as defined in the Input Table.

Examples

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL_SYS_AFL_AFLPAL_EXECUTE or AFL_SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_SAMPLING_DATA_T;
CREATE TYPE PAL_SAMPLING_DATA_T
    AS TABLE
    (
        "EMPNO"           INT,
        "GENDER"          VARCHAR (50),
        "INCOME"          DOUBLE
    );
DROP TYPE PAL_CONTROL_T ;
CREATE TYPE PAL_CONTROL_T
    AS TABLE
    (
        "NAME"            VARCHAR (50),
        "INTARGS"         INTEGER,
        "DOUBLEARGS"      DOUBLE,
        "STRINGARGS"      VARCHAR (100)
    );
DROP TYPE PAL_SAMPLING_RESULT_T;
CREATE TYPE PAL_SAMPLING_RESULT_T
    AS TABLE
    (
        "RESULT_EMPNO"    INT,
        "RESULT_GENDER"   VARCHAR (50),
        "RESULT_INCOME"   DOUBLE
    );
DROP TABLE PAL_SAMPLING_PDATA_TBL;
CREATE COLUMN TABLE
    PAL_SAMPLING_PDATA_TBL
    (
        "POSITION"        INTEGER,
        "SCHEMA_NAME"    NVARCHAR(256),
        "TYPE_NAME"       NVARCHAR(256),
        "PARAMETER_TYPE" VARCHAR(7)
    );
INSERT INTO  PAL_SAMPLING_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_SAMPLING_DATA_T',
'IN');
INSERT INTO  PAL_SAMPLING_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO  PAL_SAMPLING_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_SAMPLING_RESULT_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'SAMPLING_TEST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'SAMPLING', 'DM_PAL',
'SAMPLING_TEST_PROC', PAL_SAMPLING_PDATA_TBL);
DROP TABLE PAL_SAMPLING_DATA_TBL;
CREATE COLUMN TABLE PAL_SAMPLING_DATA_TBL LIKE PAL_SAMPLING_DATA_T;
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (1, 'male', 4000.5);

```

```

INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (2, 'male', 5000.7);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (3, 'female', 5100.8);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (4, 'male', 5400.9);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (5, 'female', 5500.2);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (6, 'male', 5540.4);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (7, 'male', 4500.9);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (8, 'female', 6000.8);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (9, 'male', 7120.8);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (10, 'female', 8120.9);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (11, 'female', 7453.9);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (12, 'male', 7643.8);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (13, 'male', 6754.3);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (14, 'male', 6759.9);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (15, 'male', 9876.5);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (16, 'female', 9873.2);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (17, 'male', 9889.9);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (18, 'male', 9910.4);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (19, 'male', 7809.3);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (20, 'female', 8705.7);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (21, 'male', 8756.0);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (22, 'female', 7843.2);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (23, 'male', 8576.9);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (24, 'male', 9560.9);
INSERT INTO PAL_SAMPLING_DATA_TBL VALUES (25, 'female', 8794.9);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL
(
    "NAME"          VARCHAR(50),
    "INTARGS"       INTEGER,
    "DOUBLEARGS"    DOUBLE,
    "STRINGARGS"    VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SAMPLING_METHOD', 0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SAMPLING_SIZE', 8, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INTERVAL', 5, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('COLUMN_CHOOSE', 2, NULL, NULL);
DROP TABLE PAL_SAMPLING_RESULT_TBL;
CREATE COLUMN TABLE PAL_SAMPLING_RESULT_TBL LIKE PAL_SAMPLING_RESULT_T;
CALL DM_PAL.SAMPLING_TEST_PROC(PAL_SAMPLING_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_SAMPLING_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_SAMPLING_RESULT_TBL;

```

Expected Result

If method is 0 and SAMPLING_SIZE is 8:

	RESULT_EMPNO	RESULT_GENDER	RESULT_INCOME
1	1	male	4000.5
2	2	male	5000.7
3	3	female	5100.8
4	4	male	5400.9
5	5	female	5500.2
6	6	male	5540.4
7	7	male	4500.9
8	8	female	6000.8

If method is 1 and SAMPLING_SIZE is 8:

	RESULT_EMPNO	RESULT_GENDER	RESULT_INCO...
1	10	female	8120.9
2	11	female	7453.9
3	12	male	7643.8
4	13	male	6754.3
5	14	male	6759.9
6	15	male	9876.5
7	16	female	9873.2
8	17	male	9889.9

If method is 2 and SAMPLING_SIZE is 8:

RESULT_EMPNO	RESULT_GENDER	RESULT_INCOME
25	female	8794.9
24	male	9560.9
23	male	8576.9
22	female	7843.2
21	male	8756.0
20	female	8705.7
19	male	7809.3
18	male	9910.4

If method is 3 and INTERVAL is 5:

RESULT_EMPNO	RESULT_GENDER	RESULT_INCOME
5	female	5500.2
10	female	8120.9
15	male	9876.5
20	female	8705.7
25	female	8794.9

If method is 4 and SAMPLING_SIZE is 8:

RESULT_EMPNO	RESULT_GENDER	RESULT_INCOME
6	male	5540.4
14	male	6759.9
6	male	5540.4
25	female	8794.9
11	female	7453.9
15	male	9876.5
7	male	4500.9
11	female	7453.9

If method is 5 and SAMPLING_SIZE is 8:

RESULT_EMPNO	RESULT_GENDER	RESULT_INCOME
11	female	7453.9
15	male	9876.5
7	male	4500.9
14	male	6759.9
23	male	8576.9
20	female	8705.7
6	male	5540.4
17	male	9889.9

If method is 6 and SAMPLING_SIZE is 8:

RESULT_EMPNO	RESULT_GENDER	RESULT_INCOME
3	female	5,100.8
6	male	5,540.4
9	male	7,120.8
12	male	7,643.8
15	male	9,876.5
18	male	9,910.4
21	male	8,756
24	male	9,560.9

If method is 7 and SAMPLING_SIZE is 8:

RESULT_EMPNO	RESULT_GENDER	RESULT_INCOME
9	male	7,120.8
9	male	7,120.8
13	male	6,754.3
7	male	4,500.9
15	male	9,876.5
5	female	5,500.2
16	female	9,873.2
3	female	5,100.8

If method is 0 and PERCENTAGE is 0.1:

RESULT_EMPNO	RESULT_GENDER	RESULT_INCOME
1	male	4000.5
2	male	5000.7
3	female	5100.8

Example 2

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_SAMPLING_DATA_T;
CREATE TYPE PAL_SAMPLING_DATA_T
AS TABLE
(
    "EMPNO"          INT,
    "GENDER"         VARCHAR(50),
    "DEP"            VARCHAR(50),
    "LOC"            INT,
    "INCOME"         DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T
AS TABLE
(
    "NAME"           VARCHAR(50),
    "INTARGS"        INTEGER,
    "DOUBLEARGS"     DOUBLE,
    "STRINGARGS"     VARCHAR(100)
);
DROP TYPE PAL_SAMPLING_RESULT_T;
CREATE TYPE PAL_SAMPLING_RESULT_T
AS TABLE
(
    "RESULT_EMPNO"   INT,
    "RESULT_GENDER"  VARCHAR(50),
    "RESULT_DEP"     VARCHAR(50),
    "RESULT_LOC"     INT,
    "RESULT_INCOME"  DOUBLE
);
```

```

DROP TABLE PAL_SAMPLING_PDATA_TBL;
CREATE COLUMN TABLE
  PAL_SAMPLING_PDATA_TBL
  (
    "POSITION"          INTEGER,
    "SCHEMA_NAME"      NVARCHAR(256),
    "TYPE_NAME"        NVARCHAR(256),
    "PARAMETER_TYPE"   VARCHAR(7)
  );
  
INSERT INTO  PAL_SAMPLING_PDATA_TBL
VALUES
(
  1,
  'DM_PAL',
  'PAL_SAMPLING_DATA_T',
  'IN'
);
INSERT INTO  PAL_SAMPLING_PDATA_TBL
VALUES
(
  2,
  'DM_PAL',
  'PAL_CONTROL_T',
  'IN'
);
INSERT INTO  PAL_SAMPLING_PDATA_TBL
VALUES
(
  3,
  'DM_PAL',
  'PAL_SAMPLING_RESULT_T',
  'OUT'
);
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP ('DM_PAL', 'SAMPLING_TEST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'SAMPLING', 'DM_PAL',
'SAMPLING_TEST_PROC', PAL_SAMPLING_PDATA_TBL);
DROP TABLE PAL_SAMPLING_DATA_TBL;
CREATE COLUMN TABLE PAL_SAMPLING_DATA_TBL
  LIKE PAL_SAMPLING_DATA_T;
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (1, 'F', 'DEV', 1, 4000.5);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (2, 'M', 'DEV', 1, 5000.7);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (3, 'F', 'SALES', 1, 5100.8);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (4, 'M', 'SALES', 1, 5400.9);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (5, 'F', 'DEV', 0, 5500.2);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (6, 'M', 'DEV', 0, 5540.4);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (7, 'F', 'SALES', 0, 4500.9);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (8, 'M', 'SALES', 0, 6000.8);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (9, 'F', 'DEV', 1, 7120.8);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (10, 'M', 'DEV', 1, 8120.9);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (11, 'F', 'SALES', 1, 7453.9);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (12, 'M', 'SALES', 1, 7643.8);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (13, 'F', 'DEV', 0, 6754.3);
INSERT INTO PAL_SAMPLING_DATA_TBL
  VALUES (14, 'M', 'DEV', 0, 6759.9);

```

```

INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (15, 'F', 'SALES', 0, 9876.5);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (16, 'M', 'SALES', 0, 9873.2);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (17, 'F', 'DEV', 1, 9889.9);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (18, 'M', 'DEV', 1, 9910.4);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (19, 'F', 'SALES', 1, 7809.3);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (20, 'M', 'SALES', 1, 8705.7);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (21, 'F', 'DEV', 0, 8756.0);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (22, 'M', 'DEV', 0, 7843.2);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (23, 'F', 'SALES', 0, 8576.9);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (24, 'M', 'SALES', 0, 9560.9);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (25, 'F', 'DEV', 1, 8794.9);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (26, 'M', 'DEV', 1, 9873.2);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (27, 'F', 'SALES', 1, 9889.9);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (28, 'M', 'SALES', 1, 9910.4);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (29, 'F', 'DEV', 0, 7809.3);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (30, 'M', 'DEV', 0, 8705.7);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (31, 'F', 'SALES', 0, 8756.0);
INSERT INTO PAL_SAMPLING_DATA_TBL
    VALUES (32, 'M', 'SALES', 0, 7843.2);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL
(
    "NAME"                VARCHAR (50),
    "INTARGS"              INTEGER,
    "DOUBLEARGS"            DOUBLE,
    "STRINGARGS"            VARCHAR (100)
);

INSERT INTO "#PAL_CONTROL_TBL"
    VALUES ('SAMPLING_METHOD', 7, NULL, NULL);
INSERT INTO "#PAL_CONTROL_TBL"
    VALUES ('SAMPLING_SIZE', 8, NULL, NULL);
INSERT INTO "#PAL_CONTROL_TBL"
    VALUES ('INTERVAL', 5, NULL, NULL);
INSERT INTO "#PAL_CONTROL_TBL"
    values ('PERCENTAGE', null, 0.5, null);
INSERT INTO "#PAL_CONTROL_TBL"
    VALUES ('COLUMN_CHOOSE', 2, NULL, NULL);
INSERT INTO "#PAL_CONTROL_TBL"
    VALUES ('COLUMN_CHOOSE', 3, NULL, NULL);
INSERT INTO "#PAL_CONTROL_TBL"
    VALUES ('COLUMN_CHOOSE', 4, NULL, NULL);

DROP TABLE PAL_SAMPLING_RESULT_TBL;
CREATE COLUMN TABLE PAL_SAMPLING_RESULT_TBL
    LIKE PAL_SAMPLING_RESULT_T;
CALL DM_PAL.SAMPLING_TEST_PROC (PAL_SAMPLING_DATA_TBL, #PAL_CONTROL_TBL,
PAL_SAMPLING_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM "#PAL_CONTROL_TBL";
SELECT * FROM PAL_SAMPLING_DATA_TBL;
SELECT * FROM PAL_SAMPLING_RESULT_TBL;

```

Expected Result

	RESULT_EMPNO	RESULT_GENDER	RESULT_HEADCOUNT	RESULT_INCOME
1	8	female	1	6,000.8
2	8	female	1	6,000.8
3	10	female	0	8,120.9
4	10	female	0	8,120.9
5	7	male	1	4,500.9
6	6	male	1	5,540.4
7	6	male	1	5,540.4
8	15	male	0	9,876.5
9	19	male	0	7,809.3
10	14	male	0	6,759.9

3.6.10 Scaling Range

In real world scenarios the collected continuous attributes are usually distributed within different ranges. It is a common practice to have the data well scaled so that data mining algorithms like neural networks, nearest neighbor classification and clustering can give more reliable results.

This release of PAL provides three scaling range methods described below. In the following, x_{ip} and y_{ip} are the original value and transformed value of the i -th record and p -th attribute, respectively.

1. Min-Max Normalization

Each transformed value is within the range $[new_minA, new_maxA]$, where new_minA and new_maxA are user-specified parameters. Supposing that $minA$ and $maxA$ are the minimum and maximum values of attribute A, we get the following calculation formula:

$$y_{ip} = (x_{ip} - minA) \times (new_maxA - new_minA) / (maxA - minA) + new_minA$$

2. Z-Score Normalization (or zero-mean normalization).

PAL uses three z-score methods.

- **Mean-Standard Deviation**

The transformed values have mean 0 and standard deviation 1. The transformation is made as follows:

$$y_{ip} = \frac{x_{ip} - \mu_p}{\sigma_p}$$

Where μ_p and σ_p are mean and standard deviations of the original values of the p -th attributes.

- **Mean-Mean Absolute Deviation**

The transformation is made as follows:

$$Y_{ip} = \frac{X_{ip} - \mu_p}{s_p}$$

$$\mu_p = \frac{1}{N} \sum_{i=1}^N X_{ip}$$

$$s_p = \sqrt{\frac{1}{N} \sum_{i=1}^N |X_{ip} - \mu_p|^2}$$

- **Median-Median Absolute Deviation**

The transformation is made as follows:

$$Y_{ip} = \frac{X_{ip} - m_p}{s_p}$$

$$m_p = \text{median}[X_{1p}, X_{2p}, \dots, X_{np}]$$

$$s_p = \text{median}[|X_{1p} - m_p|, |X_{2p} - m_p|, \dots, |X_{np} - m_p|]$$

3. Normalization by Decimal Scaling

This method transforms the data by moving the decimal point of the values, so that the maximum absolute value for each attribute is less than or equal to 1. Mathematically, $Y_{ip} = X_{ip} \times 10^{K_p}$ for each attribute p , where K_p is selected so that

$$\max(|Y_{1p}|, |Y_{2p}|, \dots, |Y_{np}|) \leq 1$$

Prerequisites

- The input data does not contain null value.
- The data is numeric, not categorical.

SCALINGRANGE

This function normalizes the data.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'SCALINGRANGE',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Scaling model OUTPUT table type>	OUT (optional)

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result output table>, <scaling model output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	Other columns	Integer or double	Variable X _n

Parameter Table

Mandatory Parameters

The following parameters are mandatory and must be given a value.

Name	Data Type	Description	Dependency
SCALING_METHOD	Integer	Scaling method: <ul style="list-style-type: none"> • 0: Min-max normalization • 1: Z-Score normalization • 2: Decimal scaling normalization 	

Name	Data Type	Description	Dependency
Z-SCORE_METHOD	Integer	<ul style="list-style-type: none"> • 0: Mean-Standard deviation • 1: Mean-Mean absolute deviation • 2: Median-Median absolute deviation 	Only valid when SCALING_METHOD is 1.
NEW_MAX	Double or integer	The new maximum value of the min-max normalization method	Only valid when SCALING_METHOD is 0.
NEW_MIN	Double or integer	The new minimum value of min-max normalization method	Only valid when SCALING_METHOD is 0.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads

Output Table

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer or double	Variable X_n	
Scaling Model (optional)	1st column	Integer	Scaling model ID	This must be the first column.
	2nd column	CLOB, varchar, or nvarchar	Binning model saved as JSON string	The table must be a column table. The minimum length of each unit (row) is 5000.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
```

```

DROP TYPE PAL_SCALING_DATA_T ;
CREATE TYPE PAL_SCALING_DATA_T AS TABLE("ID" INT, "X1" DOUBLE, "X2" DOUBLE) ;
DROP TYPE PAL_CONTROL_T ;
CREATE TYPE PAL_CONTROL_T AS TABLE( "NAME" VARCHAR (100), "INTARGS"
INTEGER,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR (100));
DROP TYPE PAL_SCALING_RESULT_T ;
CREATE TYPE PAL_SCALING_RESULT_T AS TABLE("ID" INT, "PRE_X1" DOUBLE, "PRE_X2"
DOUBLE);
DROP TYPE PAL_SCALING_MODEL_T ;
CREATE TYPE PAL_SCALING_MODEL_T AS TABLE("ID" INT, "MODEL" VARCHAR (5000));
DROP TABLE PAL_SCALING_PDATA_TBL;
CREATE COLUMN TABLE PAL_SCALING_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_SCALING_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_SCALING_DATA_T','IN');
INSERT INTO PAL_SCALING_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T','IN');
INSERT INTO PAL_SCALING_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_SCALING_RESULT_T','OUT');
INSERT INTO PAL_SCALING_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_SCALING_MODEL_T','OUT');
CALL SYS.AFLLANG_WRAPPER PROCEDURE_DROP('DM_PAL', 'SCALINGRANGE_TEST_PROC');
CALL SYS.AFLLANG_WRAPPER PROCEDURE_CREATE('AFLPAL','SCALINGRANGE', 'DM_PAL',
'SCALINGRANGE_TEST_PROC', PAL_SCALING_PDATA_TBL);
DROP TABLE PAL_SCALING_DATA_TBL;
CREATE COLUMN TABLE PAL_SCALING_DATA_TBL LIKE PAL_SCALING_DATA_T ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (0, 6.0, 9.0) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (1, 12.1, 8.3) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (2, 13.5, 15.3) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (3, 15.4, 18.7) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (4, 10.2, 19.8) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (5, 23.3, 20.6) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (6, 24.4, 24.3) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (7, 30.6, 25.3) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (8, 32.5, 27.6) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (9, 25.6, 28.5) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (10, 38.7, 29.4) ;
INSERT INTO PAL_SCALING_DATA_TBL VALUES (11, 38.7, 29.4) ;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL ( "NAME" VARCHAR
(100), "INTARGS" INTEGER,"DOUBLEARGS" DOUBLE,"STRINGARGS" VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('SCALING_METHOD',0,NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('Z-SCORE METHOD',0, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD NUMBER',2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NEW_MAX', NULL,1.0, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NEW_MIN', NULL,0.0, NULL);
DROP TABLE PAL_SCALING_RESULT_TBL;
CREATE COLUMN TABLE PAL_SCALING_RESULT_TBL LIKE PAL_SCALING_RESULT_T ;
DROP TABLE PAL_SCALING_MODEL_TBL;
CREATE COLUMN TABLE PAL_SCALING_MODEL_TBL LIKE PAL_SCALING_MODEL_T ;
CALL DM_PAL.SCALINGRANGE_TEST_PROC(PAL_SCALING_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_SCALING_RESULT_TBL, PAL_SCALING_MODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_SCALING_RESULT_TBL;
SELECT * FROM PAL_SCALING_MODEL_TBL;

```

Expected Result

PAL_SCALING_RESULT_TBL:

	ID	PRE_X1	PRE_X2
1	0	0	0.033175355450236935
2	1	0.18654434250764523	0
3	2	0.22935779816513757	0.3317535545023697
4	3	0.2874617737003058	0.4928909952606635
5	4	0.12844036697247702	0.5450236966824645
6	5	0.529051987767584	0.5829383886255924
7	6	0.5626911314984708	0.7582938388625593
8	7	0.7522935779816513	0.8056872037914693
9	8	0.8103975535168194	0.9146919431279622
10	9	0.599388379204893	0.957345971563981
11	10	1	1
12	11	1	1

PAL_SCALING_MODEL_TBL:

	ID	MODEL
1	0	{"Control":{"NEW_MAX":1.0,"NEW_MIN":0.0,"SCALING_METHOD":0}, "ScalingModel":{"Ma...}}

3.6.11 Substitute Missing Values

This function is used to replace the missing values with some statistical values. Currently, three methods are provided. The missing values of a specific attribute are replaced by one of the following values (of the attribute/column):

- **Mode:** the value that appears most in the given column.
- **Mean:** the arithmetic mean of not null values in the given column. The mean value is calculated by the following, where N is the number of not null values.

$$\bar{x} = \frac{1}{N} \sum x_i$$

- **Median:** the numerical value separating the higher half of the values from the lower half. The median value is calculated by:

$$\text{median} = a[\frac{N}{2}]$$

If N is an odd number:

$$\text{median} = \frac{1}{2} (a[\frac{N}{2} - 1] + a[\frac{N}{2}])$$

i Note

The Mode can only be used for categorical attributes, whereas the Mean and Median can only be used for continuous attributes.

Prerequisite

Each column must contain at least one valid value.

SUBSTITUTE_MISSING_VALUES

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL',
'SUBSTITUTE_MISSING_VALUES', '<schema_name>', '<procedure_name>',
<signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Statistical table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>, <statistical table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	Columns	Varchar, nvarchar, double, or integer	Attribute data

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
<column name>	Integer	<p>Defines the method for substituting missing values.</p> <ul style="list-style-type: none"> • 100: Mode • 200: Mean • 201: Median • 101: Use specific string value to replace missing values • 202: Use specific integer value to replace missing values • 203: Use specific double value to replace missing values <p>For methods 101, 202, and 203, use the below syntax to specify a new value to replace missing values:</p> <pre>INSERT INTO #PAL_CONTROL_TBL VALUES ('V0', <METHOD>, NULL, '<NEW VALUE>');</pre> <p>Where V0 is the column name, <METHOD> is 101, 202, or 203, and <NEW VALUE> is the value that you want to use to replace missing values.</p>

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Specifies the number of threads.

Output Tables

Table	Column	Column Data Type	Description
Result	Columns	Varchar, nvarchar, integer, or double	Each column type should match the type of the original input data.
Statistical	1st column	Varchar or nvarchar	Column name
	2nd column	Varchar or nvarchar	Substitute value
	3rd column	Integer	Hit times

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and

- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_MISSING_VALUES_DATA_T;
CREATE TYPE PAL_MISSING_VALUES_DATA_T AS TABLE(
    "V0" INTEGER,
    "V1" DOUBLE,
    "V2" VARCHAR(5)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_MISSING_VALUES_RESULT_T;
CREATE TYPE PAL_MISSING_VALUES_RESULT_T AS TABLE(
    "V0" INTEGER,
    "V1" DOUBLE,
    "V2" VARCHAR(5)
);

DROP TYPE PAL_MISSING_VALUES_NULL_MAP_T;
CREATE TYPE PAL_MISSING_VALUES_NULL_MAP_T AS TABLE(
    "ATTRIBUTE_NAME" VARCHAR(20),
    "ATTRIBUTE_VALUE" VARCHAR(20),
    "HIT_TIMES" INTEGER
);
DROP TABLE PAL_MISSING_VALUES_PDATA_TBL;
CREATE COLUMN TABLE PAL_MISSING_VALUES_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_MISSING_VALUES_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_MISSING_VALUES_DATA_T', 'IN');
INSERT INTO PAL_MISSING_VALUES_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T',
'IN');
INSERT INTO PAL_MISSING_VALUES_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_MISSING_VALUES_RESULT_T', 'OUT');
INSERT INTO PAL_MISSING_VALUES_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_MISSING_VALUES_NULL_MAP_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL',
'AFL_PAL_MISSING_VALUES_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL',
'SUBSTITUTE_MISSING_VALUES', 'DM_PAL', 'AFL_PAL_MISSING_VALUES_PROC',
PAL_MISSING_VALUES_PDATA_TBL);
DROP TABLE PAL_MISSING_VALUES_DATA_TBL;
CREATE COLUMN TABLE PAL_MISSING_VALUES_DATA_TBL LIKE PAL_MISSING_VALUES_DATA_T;
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (1, 0.5, 'A');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (1, 1.5, 'A');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (2, 1.5, 'A');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (3, 0.5, 'A');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (4, 1.1, 'A');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (5, 0.5, 'B');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (6, 1.5, 'B');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (7, 1.5, 'B');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (8, null, 'B');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (null, 1.2, null);
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (10, 15.5, 'C');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (11, 16.5, 'C');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (12, 16.5, 'C');

```

```

INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (13, 15.5, 'C');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (null, 15.6, 'D');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (15, 15.5, null);
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (16, 16.5, 'D');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (17, 16.5, 'D');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (18, 15.5, 'D');
INSERT INTO PAL_MISSING_VALUES_DATA_TBL VALUES (19, 15.7, 'A');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('V0', 100, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('V1', 201, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('V2', 100, null, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, null, null);
DROP TABLE PAL_MISSING_VALUES_RESULT_TBL;
CREATE COLUMN TABLE PAL_MISSING_VALUES_RESULT_TBL LIKE
PAL_MISSING_VALUES_RESULT_T;

DROP TABLE PAL MISSING_VALUES_NULL_MAP_TBL;
CREATE COLUMN TABLE PAL_MISSING_VALUES_NULL_MAP_TBL LIKE
PAL_MISSING_VALUES_NULL_MAP_T;
CALL "DM_PAL".AFL_PAL_MISSING_VALUES_PROC(PAL_MISSING_VALUES_DATA_TBL,
#PAL_CONTROL_TBL, PAL_MISSING_VALUES_RESULT_TBL,
PAL_MISSING_VALUES_NULL_MAP_TBL) with OVERVIEW;
SELECT * FROM PAL_MISSING_VALUES_RESULT_TBL;
SELECT * FROM PAL_MISSING_VALUES_NULL_MAP_TBL;

```

Expected Result

PAL_MISSING_VALUES_RESULT_TBL:

V0	V1	V2
1	0.5	A
1	1.5	A
2	1.5	A
3	0.5	A
4	1.1	A
5	0.5	B
6	1.5	B
7	1.5	B
8	15.5	B
1	1.2	A
10	15.5	C
11	16.5	C
12	16.5	C
13	15.5	C
1	15.6	D
15	15.5	A
16	16.5	D
17	16.5	D
18	15.5	D
19	15.7	A

PAL_MISSING_VALUES_NULL_MAP_TBL:

ATTRIBUTE_NAME	ATTRIBUTE_VALUE	HIT_TIMES
V0	1	2
V1	15.5	1
V2	A	2

3.6.12 Variance Test

Variance Test is a method to identify the outliers of n number of numeric data $\{x_i\}$ where $0 < i < n+1$, using the mean $\{\mu\}$ and the standard deviation of $\{\sigma\}$ of n number of numeric data $\{x_i\}$.

Below is the algorithm for Variance Test:

1. Calculate the mean (μ) and the standard deviation (σ):

$$\mu = \frac{1}{n} \times \sum_{i=1}^n x_i$$
$$\sigma = \sqrt{\frac{1}{n-1} \times \sum_{i=1}^n (x_i - \mu)^2}$$

2. Set the upper and lower bounds as follows:

$$\text{Upper-bound} = \mu + \text{multiplier} * \sigma$$

$$\text{Lower-bound} = \mu - \text{multiplier} * \sigma$$

Where the multiplier is a double type coefficient provided by the user to test whether all the values of a numeric vector are in the range.

If a value is outside the range, it means it doesn't pass the Variance Test. The value is marked as an outlier.

Prerequisites

- No missing or null data in the inputs.
- The data is numeric, not categorical.

VARIANCETEST

This is a variance test function.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'VARIANCETEST',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Test OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>, <test output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Integer or double	Raw data

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
SIGMA_NUM	Double	Multiplier for sigma

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads

Output Tables

Table	Column	Column Data Type	Description	Constraint
Result	1st column	Double	Mean value	
	2nd column	Double	Standard deviation	
Test	1st column	Integer, bigint, varchar, or nvarchar	ID	
	2nd column	Integer	Result output	<ul style="list-style-type: none"> • 0: in bounds • 1: out of bounds

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_VT_DATA_T;
CREATE TYPE PAL_VT_DATA_T AS TABLE (
    "ID" INTEGER,
    "X" DOUBLE
);
DROP TYPE PAL_VT_RESULT_T;
CREATE TYPE PAL_VT_RESULT_T AS TABLE(
    "MEAN" DOUBLE,
    "SD" DOUBLE
);
DROP TYPE PAL_VT_TEST_T;
CREATE TYPE PAL_VT_TEST_T AS TABLE(
    "ID" INTEGER,
    "Test" INTEGER
);
DROP TYPE PAL_VT_CONTROL_T;
CREATE TYPE PAL_VT_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP table PAL_VT_PDATA_TBL;
CREATE COLUMN TABLE PAL_VT_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_VT_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_VT_DATA_T', 'IN');
INSERT INTO PAL_VT_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_VT_CONTROL_T', 'IN');
INSERT INTO PAL_VT_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_VT_RESULT_T', 'OUT');
INSERT INTO PAL_VT_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_VT_TEST_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_VARIANCE_TEST_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'VARIANCETEST', 'DM_PAL',
'PAL_VARIANCE_TEST_PROC', PAL_VT_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" NVARCHAR(100)
);

```

```

    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SIGMA_NUM', null, 3.0, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 8, null, null);
DROP TABLE PAL_VT_DATA_TBL;
CREATE COLUMN TABLE PAL_VT_DATA_TBL LIKE PAL_VT_DATA_T;
INSERT INTO PAL_VT_DATA_TBL VALUES (0,25);
INSERT INTO PAL_VT_DATA_TBL VALUES (1,20);
INSERT INTO PAL_VT_DATA_TBL VALUES (2,23);
INSERT INTO PAL_VT_DATA_TBL VALUES (3,29);
INSERT INTO PAL_VT_DATA_TBL VALUES (4,26);
INSERT INTO PAL_VT_DATA_TBL VALUES (5,23);
INSERT INTO PAL_VT_DATA_TBL VALUES (6,22);
INSERT INTO PAL_VT_DATA_TBL VALUES (7,21);
INSERT INTO PAL_VT_DATA_TBL VALUES (8,22);
INSERT INTO PAL_VT_DATA_TBL VALUES (9,25);
INSERT INTO PAL_VT_DATA_TBL VALUES (10,26);
INSERT INTO PAL_VT_DATA_TBL VALUES (11,28);
INSERT INTO PAL_VT_DATA_TBL VALUES (12,29);
INSERT INTO PAL_VT_DATA_TBL VALUES (13,27);
INSERT INTO PAL_VT_DATA_TBL VALUES (14,26);
INSERT INTO PAL_VT_DATA_TBL VALUES (15,23);
INSERT INTO PAL_VT_DATA_TBL VALUES (16,22);
INSERT INTO PAL_VT_DATA_TBL VALUES (17,23);
INSERT INTO PAL_VT_DATA_TBL VALUES (18,25);
INSERT INTO PAL_VT_DATA_TBL VALUES (19,103);
DROP TABLE PAL_VT_RESULT_TBL;
CREATE COLUMN TABLE PAL_VT_RESULT_TBL LIKE PAL_VT_RESULT_T;
DROP TABLE PAL_VT_TEST_TBL;
CREATE COLUMN TABLE PAL_VT_TEST_TBL LIKE PAL_VT_TEST_T;
CALL "DM_PAL".PAL_VARIANCE_TEST_PROC(PAL_VT_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_VT_RESULT_TBL, PAL_VT_TEST_TBL) with overview;
SELECT * FROM PAL_VT_RESULT_TBL;
SELECT * FROM PAL_VT_TEST_TBL;

```

Expected Result

PAL_VT_RESULT_TBL:

MEAN	SD
28.4	17.747942801468863

PAL_VT_TEST_TBL:

ID	Test
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	1

3.7 Statistics Algorithms

This section describes the statistics functions that are provided by the Predictive Analysis Library.

3.7.1 Chi-Squared Test for Goodness of Fit

The chi-squared test for goodness of fit tells whether or not an observed distribution differs from an expected chi-squared distribution.

The chi-squared value is defined as:

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i}$$

where O_i is an observed frequency and E_i is an expected frequency.

The chi-squared value χ^2 will be used to calculate a p-value by comparing the value of chi-squared to a chi-squared distribution. The degree of freedom is set to $n-p$ where p is the reduction in degrees of freedom.

Prerequisites

- The input data has three columns. The first column is ID column with integer, varchar, or nvarchar type; the second column is the observed data with integer or double type; the third column is the expected frequency.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

CHISQTESTFIT

This function does the Pearson's chi-squared test for goodness of fit according to the user's input.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'CHISQTESTFIT',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<Result OUTPUT table type>	OUT
3	<schema_name>	<StatValue OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name> (<input table>, <result output table>,
<statvalue output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, result, and statvalue tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer or double	Observed data	
	3rd column	Double	Expected frequency P	The sum of expected frequency must be 1.

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	2nd column	Double	Observed data
	3rd column	Double	Expected data
	4th column	Double	The residual
Stat Value	1st column	Varchar or nvarchar	Name of statistics
	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_CHISQTESTFIT_DATA_T;
CREATE TYPE PAL_CHISQTESTFIT_DATA_T AS TABLE(
    "ID" INTEGER,
    "OBSERVED" DOUBLE,
    "P" DOUBLE
) ;

DROP TYPE PAL_CHISQTESTFIT_RESULT_T;
CREATE TYPE PAL_CHISQTESTFIT_RESULT_T AS TABLE(
    "ID" INTEGER,
    "OBSERVED" DOUBLE,
    "EXPECTED" DOUBLE,
    "RESIDUAL" DOUBLE
) ;
DROP TYPE PAL_CHISQTESTFIT_STATVALUE_T;
CREATE TYPE PAL_CHISQTESTFIT_STATVALUE_T AS TABLE(
    "NAME" VARCHAR(100),
    "VALUE" DOUBLE
) ;
DROP TABLE PAL_CHISQTESTFIT_PDATA_TBL;
CREATE COLUMN TABLE PAL_CHISQTESTFIT_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_CHISQTESTFIT_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_CHISQTESTFIT_DATA_T','IN');
INSERT INTO PAL_CHISQTESTFIT_PDATA_TBL VALUES (2,'DM_PAL',
'PAL_CHISQTESTFIT_RESULT_T','OUT');
INSERT INTO PAL_CHISQTESTFIT_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_CHISQTESTFIT_STATVALUE_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_CHISQTESTFIT_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CHISQTESTFIT', 'DM_PAL',
'PAL_CHISQTESTFIT_PROC', 'PAL_CHISQTESTFIT_PDATA_TBL');
DROP TABLE PAL_CHISQTESTFIT_DATA_TBL;
CREATE COLUMN TABLE PAL_CHISQTESTFIT_DATA_TBL LIKE PAL_CHISQTESTFIT_DATA_T;
INSERT INTO PAL_CHISQTESTFIT_DATA_TBL VALUES (0,519,0.3);
INSERT INTO PAL_CHISQTESTFIT_DATA_TBL VALUES (1,364,0.2);
INSERT INTO PAL_CHISQTESTFIT_DATA_TBL VALUES (2,363,0.2);
INSERT INTO PAL_CHISQTESTFIT_DATA_TBL VALUES (3,200,0.1);
INSERT INTO PAL_CHISQTESTFIT_DATA_TBL VALUES (4,212,0.1);
INSERT INTO PAL_CHISQTESTFIT_DATA_TBL VALUES (5,193,0.1);
DROP TABLE PAL_CHISQTESTFIT_RESULT_TBL;

```

```

CREATE COLUMN TABLE PAL_CHISQTESTFIT_RESULT_TBL LIKE PAL_CHISQTESTFIT_RESULT_T;
DROP TABLE PAL_CHISQTESTFIT_STATVALUE_TBL;
CREATE COLUMN TABLE PAL_CHISQTESTFIT_STATVALUE_TBL LIKE
PAL_CHISQTESTFIT_STATVALUE_T;
CALL DM_PAL.PAL_CHISQTESTFIT_PROC(PAL_CHISQTESTFIT_DATA_TBL,
PAL_CHISQTESTFIT_RESULT_TBL, PAL_CHISQTESTFIT_STATVALUE_TBL) WITH OVERVIEW;
SELECT * FROM PAL_CHISQTESTFIT_RESULT_TBL;
SELECT * FROM PAL_CHISQTESTFIT_STATVALUE_TBL;

```

Expected Result

PAL_CHISQTESTFIT_RESULT_TBL:

	ID	OBSERVED	EXPECTED	RESIDUAL
1	0	519	555.3	-36.29999999999955
2	1	364	370.20000000000005	-6.200000000000455
3	2	363	370.20000000000005	-7.200000000000455
4	3	200	185.10000000000002	14.89999999999977
5	4	212	185.10000000000002	26.89999999999977
6	5	193	185.10000000000002	7.89999999999977

PAL_CHISQTESTFIT_STATVALUE_TBL:

	NAME	VALUE
1	Chi-squared Value	8.062668827660708
2	degree of freedom	5
3	p-value	0.15281547755083613

3.7.2 Chi-Squared Test for Independent

The chi-squared test for independent tells whether observations of two variables are independent from each other.

The chi-squared value is defined as:

$$\chi^2 = \sum_{i=1}^r \sum_{j=1}^c \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}}$$

where $O_{i,j}$ is an observed frequency and $E_{i,j}$ is an expected frequency. Then χ^2 will be used to calculate a p-value by comparing the value of chi-squared to a chi-squared distribution. The degree of freedom is set to $(r-1) * (c-1)$.

Prerequisites

- The input data contains an ID column in the first column with the type of integer, varchar, or nvarchar and the other columns are of integer or double data type.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

CHISQTESTIND

This function does the Pearson's chi-squared test for independent according to the user's input.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'CHISQTESTIND',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<ExpectedResult OUTPUT table type>	OUT
4	<schema_name>	<StatValue OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name> (<input table>, <parameter table>,
<expectedresult output table>, <statvalue output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, result, and statvalue tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	Other columns	Integer or double	Observed data	

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
CORRECTION_TYPE	Integer	0	<p>Controls whether to perform the Yates's correction for continuity.</p> <ul style="list-style-type: none"> • 0: Does not perform the Yates's correction • 1: Performs the Yates's correction

Output Tables

Table	Column	Column Data Type	Description
Expected Result	1st column	Integer, bigint, varchar, or nvarchar	ID
	Other columns	Double	Expected data
Stat Value	1st column	Varchar or nvarchar	Name of statistics
	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_CHISQTESTIND_DATA_T;
CREATE TYPE PAL_CHISQTESTIND_DATA_T AS TABLE(
    "ID" VARCHAR(100),
    "X1" INTEGER,
    "X2" DOUBLE,
    "X3" INTEGER,
    "X4" DOUBLE
);

DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_CHISQTESTIND_EXPECTEDRESULT_T;
CREATE TYPE PAL_CHISQTESTIND_EXPECTEDRESULT_T AS TABLE(
    "ID" VARCHAR(100),
    "X1" DOUBLE,
    "X2" DOUBLE,
    "X3" DOUBLE,
    "X4" DOUBLE
);
DROP TYPE PAL_CHISQTESTIND_STATVALUE_T;
CREATE TYPE PAL_CHISQTESTIND_STATVALUE_T AS TABLE(
    "NAME" VARCHAR(100),
    "VAL" DOUBLE
);

```

```

    "VALUE" DOUBLE
);
DROP TABLE PAL_CHISQTESTIND_PDATA_TBL;
CREATE COLUMN TABLE PAL_CHISQTESTIND_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_CHISQTESTIND_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_CHISQTESTIND_DATA_T','IN');
INSERT INTO PAL_CHISQTESTIND_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_CONTROL_T','IN');
INSERT INTO PAL_CHISQTESTIND_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_CHISQTESTIND_EXPECTEDRESULT_T','OUT');
INSERT INTO PAL_CHISQTESTIND_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_CHISQTESTIND_STATVALUE_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_CHISQTESTIND_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CHISQTESTIND', 'DM_PAL',
'PAL_CHISQTESTIND_PROC', 'PAL_CHISQTESTIND_PDATA_TBL');
DROP TABLE PAL_CHISQTESTIND_DATA_TBL;
CREATE COLUMN TABLE PAL_CHISQTESTIND_DATA_TBL LIKE PAL_CHISQTESTIND_DATA_T;
INSERT INTO PAL_CHISQTESTIND_DATA_TBL VALUES ('male',25,23,11,14);
INSERT INTO PAL_CHISQTESTIND_DATA_TBL VALUES ('female',41,20,18,6);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('CORRECTION_TYPE',0,null,null); //default
value is 0, it can be {0,1}
DROP TABLE PAL_CHISQTESTIND_EXPECTEDRESULT_TBL;
CREATE COLUMN TABLE PAL_CHISQTESTIND_EXPECTEDRESULT_TBL LIKE
PAL_CHISQTESTIND_EXPECTEDRESULT_T;
DROP TABLE PAL_CHISQTESTIND_STATVALUE_TBL;
CREATE COLUMN TABLE PAL_CHISQTESTIND_STATVALUE_TBL LIKE
PAL_CHISQTESTIND_STATVALUE_T;
CALL DM_PAL.PAL_CHISQTESTIND_PROC(PAL_CHISQTESTIND_DATA_TBL, #PAL_CONTROL_TBL,
PAL_CHISQTESTIND_EXPECTEDRESULT_TBL, PAL_CHISQTESTIND_STATVALUE_TBL) WITH
OVERVIEW;
SELECT * FROM PAL_CHISQTESTIND_EXPECTEDRESULT_TBL;
SELECT * FROM PAL_CHISQTESTIND_STATVALUE_TBL;

```

Expected Result

PAL_CHISQTESTIND_EXPECTEDRESULT_TBL:

	ID	X1	X2	X3	X4
1	male	30.49367088607595	19.867088607594937	13.39873417721519	9.240506329113924
2	female	35.50632911392405	23.132911392405063	15.60126582278481	10.759493670886076

PAL_CHISQTESTIND_STATVALUE_TBL:

	NAME	VALUE
1	Chi-squared Value	8.113152118695105
2	degree of freedom	3
3	p-value	0.043730169309533995

3.7.3 Cumulative Distribution Function

This algorithm evaluates the probability of a variable x from the cumulative distribution function (CDF) or complementary cumulative distribution function (CCDF) for a given probability distribution.

CDF

CDF describes the lower tail probability of a probability distribution. It is the probability that a random variable x with a given probability distribution takes a value less than or equal to x . The CDF $F(x)$ of a real-valued random variable x is given by:

$$F(x) = P[X \leq x]$$

CCDF

CCDF describes the upper tail probability of a probability distribution. It is the probability that a random variable x with a given probability distribution takes a value greater than x . The CCDF $\bar{F}(x)$ of a real-valued random variable x is given by:

$$\bar{F}(x) = P[X > x] = 1 - F(x)$$

Prerequisites

- The input data is numeric, not categorical.
- The input data and distribution parameters do not contain null data. The algorithm will issue errors when encountering null values.

DISTRPROB

This function calculates the value of CDF or CCDF (depending on the parameter given by user) for a given distribution.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'DISTRPROB',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Distribution parameter INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN

Position	Schema Name	Table Type Name	Parameter Type
4	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <distribution parameter input table>, <parameter table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Data	1st column	Double	Input data
Distribution Parameter	1st column	Varchar or nvarchar	Names of the distribution parameters. See Distribution Parameters Definition table for details.
	2nd column	Varchar or nvarchar	Values of distribution parameters. See Distribution Parameters Definition table for details.

Distribution Parameters Definition Table

The following parameters are mandatory and must be given a value.

Distribution	Parameter Name	Parameter Value	Constraint
Uniform	"DistributionName"	"Uniform"	
	"Min"	"0.0"	Min < Max
	"Max"	"1.0"	
Normal	"DistributionName"	"Normal"	
	"Mean"	"0.0"	
	"Variance"	"1.0"	Variance > 0
Weibull	"DistributionName"	"Weibull"	
	"Shape"	"1.0"	Shape > 0
	"Scale"	"1.0"	Scale > 0
Gamma	"DistributionName"	"Gamma"	
	"Shape"	"1.0"	Shape > 0
	"Scale"	"1.0"	Scale > 0

i Note

The names and values of the distribution parameters are not case sensitive.

Parameter Table

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
LOWER_UPPER	Integer	0	<ul style="list-style-type: none">• 0: Calculates the value of CDF• 1: Calculates the value of CCDF

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Double	Input data
	2nd column	Double	Probabilities

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_DISTRPROB_DATA_T;
CREATE TYPE PAL_DISTRPROB_DATA_T AS TABLE ("DATACOL" DOUBLE);
DROP TYPE PAL_DISTRPROB_DISTRPARAM_T;
CREATE TYPE PAL_DISTRPROB_DISTRPARAM_T AS TABLE (
"NAME" VARCHAR(50),
"VALUE" VARCHAR(50)
);
DROP TYPE PAL_DISTRPROB_RESULT_T;
CREATE TYPE PAL_DISTRPROB_RESULT_T AS TABLE (
"INPUTDATA" DOUBLE,
"PROBABILITY" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE (
"NAME" VARCHAR(50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_DISTRPROB_PDATA_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_PDATA_TBL(
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
```

```

INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_DISTRPROB_DATA_T', 'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_DISTRPROB_DISTRPARAM_T', 'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_DISTRPROB_RESULT_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_DISTRPROB_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL',
'DISTRPROB', 'DM_PAL', 'PAL_DISTRPROB_PROC', 'PAL_DISTRPROB_PDATA_TBL');
DROP TABLE PAL_DISTRPROB_DATA_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_DATA_TBL LIKE PAL_DISTRPROB_DATA_T;
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (37.4);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (277.9);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (463.2);
DROP TABLE PAL_DISTRPROB_DISTRPARAM_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_DISTRPARAM_TBL LIKE PAL_DISTRPROB_DISTRPARAM_T;
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('DistributionName', 'Weibull');
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('Shape', '2.11995');
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('Scale', '277.698');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LOWER_UPPER', 0, null, null);
DROP TABLE PAL_DISTRPROB_RESULT_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_RESULT_TBL LIKE PAL_DISTRPROB_RESULT_T;
CALL DM_PAL.PAL_DISTRPROB_PROC(PAL_DISTRPROB_DATA_TBL,
PAL_DISTRPROB_DISTRPARAM_TBL, #PAL_CONTROL_TBL, PAL_DISTRPROB_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_DISTRPROB_RESULT_TBL;

```

Expected Result

	INPUTDATA	PROBABILITY
1	37.4	0.014160024030305474
2	277.9	0.6326876484179044
3	463.2	0.9480935416104985

3.7.4 Distribution Fitting

This algorithm aims to fit a probability distribution for a variable according to a series of measurements to the variable. There are many probability distributions of which some can be fitted more closely to the observed variable than others.

In PAL, you can choose one probability distribution type from a supporting list (Normal, Gamma, Weibull, and Uniform) and then PAL will calculate the optimized parameters of this distribution which fit the observed variable best.

PAL supports two distribution fitting interfaces: DISTRFIT and DISTRFITCENSORED. DISTRFIT fits un-censored data while DISTRFITCENSORED fits censored data.

Maximum likelihood and median rank are two estimation methods for finding the optimized parameters. In PAL, the maximum likelihood method supports all distribution types in the supporting list for un-censored data and supports Weibull distribution fitting for a mixture of left, right, and interval censored data. The median rank

method supports the Weibull distribution fitting for both right-censored and un-censored data but it does not support other distribution types.

Prerequisites

- The input data is numeric, not categorical.
- For DISTRFIT, the input data does not contain null data. The algorithm will issue errors when encountering null values.

DISTRFIT

This is a distribution fitting function with un-censored data.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'DISTRFIT',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT
4	<schema_name>	<Statistics OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <parameter table>,
<result output table>, <statistics output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Double	Input data

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
DISTRIBUTIONNAME	Varchar	<p>Distribution name:</p> <ul style="list-style-type: none"> • Normal • Gamma • Weibull • Uniform

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
OPTIMAL_METHOD	Integer	1 (for Weibull distribution) 0 (for other distributions)	<p>Estimation method:</p> <ul style="list-style-type: none"> • 0: Maximum likelihood • 1: Median rank (only for Weibull distribution)

i Note

For Gamma or Weibull distribution, you can specify the start values of parameters for optimization. If the start values are not specified, the algorithm will calculate them automatically.

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Name of distribution parameters
	2nd column	Varchar or nvarchar	Value of distribution parameters
Statistics	1st column	Varchar or nvarchar	Name of statistics
	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_DISTRFIT_DATA_T;
CREATE TYPE PAL_DISTRFIT_DATA_T AS TABLE ("DATACOL" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
  "NAME" VARCHAR (50),
  "INTARGS" INTEGER,
  "DOUBLEARGS" DOUBLE,
  "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_DISTRFIT_ESTIMATION_T;
CREATE TYPE PAL_DISTRFIT_ESTIMATION_T AS TABLE(
  "NAME" VARCHAR(50),
  "VALUE" VARCHAR(50)
);
DROP TYPE PAL_DISTRFIT_STATISTICS_T;
CREATE TYPE PAL_DISTRFIT_STATISTICS_T AS TABLE(
  "NAME" VARCHAR(50),
  "VALUE" DOUBLE
);
DROP TABLE PDATA_TBL;
CREATE COLUMN TABLE PDATA_TBL(
  "POSITION" INT,
  "SCHEMA_NAME" NVARCHAR(256),
  "TYPE_NAME" NVARCHAR(256),
  "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_DISTRFIT_DATA_T', 'IN');
INSERT INTO PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_DISTRFIT_ESTIMATION_T', 'OUT');
INSERT INTO PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_DISTRFIT_STATISTICS_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DISTRFIT_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'DISTRFIT',
  'DM_PAL','PAL_DISTRFIT_PROC', PDATA_TBL);
DROP TABLE PAL_DISTRFIT_DATA_TBL;
CREATE COLUMN TABLE PAL_DISTRFIT_DATA_TBL LIKE PAL_DISTRFIT_DATA_T;
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (71);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (83);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (92);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (104);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (120);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (134);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (138);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (146);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (181);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (191);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (206);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (226);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (276);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (283);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (291);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (332);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (351);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (401);
INSERT INTO PAL_DISTRFIT_DATA_TBL VALUES (466);
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL(
  "NAME" VARCHAR (50),
  "INTARGS" INTEGER,
```

```

"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
INSERT INTO PAL_CONTROL_TBL VALUES ('DISTRIBUTIONNAME', NULL, NULL, 'WEIBULL');
INSERT INTO PAL_CONTROL_TBL VALUES ('OPTIMAL_METHOD', 0, NULL, NULL);
DROP TABLE PAL_DISTRFIT_ESTIMATION_TBL;
CREATE COLUMN TABLE PAL_DISTRFIT_ESTIMATION_TBL LIKE PAL_DISTRFIT_ESTIMATION_T;
DROP TABLE PAL_DISTRFIT_STATISTICS_TBL;
CREATE COLUMN TABLE PAL_DISTRFIT_STATISTICS_TBL LIKE PAL_DISTRFIT_STATISTICS_T;
CALL DM_PAL.PAL_DISTRFIT_PROC(PAL_DISTRFIT_DATA_TBL, PAL_CONTROL_TBL,
PAL_DISTRFIT_ESTIMATION_TBL, PAL_DISTRFIT_STATISTICS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_DISTRFIT_ESTIMATION_TBL;
SELECT * FROM PAL_DISTRFIT_STATISTICS_TBL;

```

Expected Result

PAL_DISTRFIT_ESTIMATION_TBL:

NAME	VALUE
DISTRIBUTIONNAME	WEIBULL
SCALE	244.401
SHAPE	2.06698

PAL_DISTRFIT_STATISTICS_TBL:

NAME	VALUE
LOGLIKELIHOOD	-115.11379382624872

DISTRFITCENSORED

This is a Weibull distribution fitting function with censored data. This release of PAL only supports the maximum likelihood estimation method on a mixture of left, right, and interval censored data and the median rank estimation method on right censored data.

Procedure Generation

```

CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'DISTRFITCENSORED',
'<schema_name>', '<procedure_name>', <signature_table>);

```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Position	Schema Name	Table Type Name	Parameter Type
4	<schema_name>	<Statistics OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name> (<data input table>, <parameter table>,
<result output table>, <statistics output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Double	<ul style="list-style-type: none"> For un-censored data, both 1st column and 2nd column should be observed data. Example: (17, 17) For right censored data, 1st column should be lower bound and 2nd column should be NULL. Example: (19, NULL) For left censored data, 1st column should be NULL and 2nd column should be upper bound. Example: (NULL, 21) For interval censored data, 1st column should be lower bound and 2nd column should be upper bound. Example: (23, 50) <p>Note: The current release of PAL supports the following:</p> <ul style="list-style-type: none"> Using the maximum likelihood method on un-censored data to estimate all distribution types in the supporting list;

Table	Column	Column Data Type	Description
	2nd column	Double	<ul style="list-style-type: none"> Using the maximum likelihood method on a mixture of left, right, and interval censored data to estimate weibull distribution; Using the media rank estimation method to estimate Weibull distribution on uncensored and right censored data.

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
DISTRIBUTIONNAME	Varchar	Distribution name: <ul style="list-style-type: none"> Weibull

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
OPTIMAL_METHOD	Integer	0	Estimation method: <ul style="list-style-type: none"> 0: Maximum likelihood 1: Median rank (only for weibull distribution)

Output Tables

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Name of distribution parameters
	2nd column	Varchar or nvarchar	Value of distribution parameters
Statistics	1st column	Varchar or nvarchar	Name of statistics

Table	Column	Column Data Type	Description
	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Example 1: Median Rank Estimation Method

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DISTRFITCENSORED_DATA_T;
CREATE TYPE PAL_DISTRFITCENSORED_DATA_T AS TABLE(
"LEFT" DOUBLE,
"RIGHT" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_DISTRFITCENSORED_ESTIMATION_T;
CREATE TYPE PAL_DISTRFITCENSORED_ESTIMATION_T AS TABLE(
"NAME" VARCHAR(50),
"VALUE" VARCHAR(50)
);
DROP TYPE PAL_DISTRFITCENSORED_STATISTICS_T;
CREATE TYPE PAL_DISTRFITCENSORED_STATISTICS_T AS TABLE(
"NAME" VARCHAR(50),
"VALUE" DOUBLE
);
DROP TABLE PDATA_TBL;
CREATE COLUMN TABLE PDATA_TBL(
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PDATA_TBL VALUES (1, 'DM_PAL','PAL_DISTRFITCENSORED_DATA_T', 'IN');
INSERT INTO PDATA_TBL VALUES (2, 'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PDATA_TBL VALUES (3, 'DM_PAL','PAL_DISTRFITCENSORED_ESTIMATION_T',
'OUT');
INSERT INTO PDATA_TBL VALUES (4, 'DM_PAL','PAL_DISTRFITCENSORED_STATISTICS_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DISTRFITCENSORED_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL',
'DISTRFITCENSORED','DM_PAL','PAL_DISTRFITCENSORED_PROC',PDATA_TBL);
DROP TABLE PAL_DISTRFITCENSORED_DATA_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_DATA_TBL LIKE
PAL_DISTRFITCENSORED_DATA_T;
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (17,NULL);--right censored data.
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (55, NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (55, NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (71,71);--uncensored data
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (83,83);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (85, NULL);

```

```

INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (92,92);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (104,104);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (115, NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (120,120);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (125, NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (134,134);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (138,138);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (146,146);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (163, NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (171, NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (181,181);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (191,191);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (195, NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (206,206);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (226,226);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (260, NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (276,276);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (283,283);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (291,291);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (332,332);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (351,351);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (384, NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (401,401);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (466,466);
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
INSERT INTO PAL_CONTROL_TBL VALUES ('DISTRIBUTIONNAME', NULL, NULL,'WEIBULL');
INSERT INTO PAL_CONTROL_TBL VALUES ('OPTIMAL_METHOD',1, NULL, NULL);
DROP TABLE PAL_DISTRFITCENSORED_ESTIMATION_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_ESTIMATION_TBL LIKE
PAL_DISTRFITCENSORED_ESTIMATION_T;
DROP TABLE PAL_DISTRFITCENSORED_STATISTICS_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_STATISTICS_TBL LIKE
PAL_DISTRFITCENSORED_STATISTICS_T;
CALL DM_PAL.PAL_DISTRFITCENSORED_PROC(PAL_DISTRFITCENSORED_DATA_TBL,
PAL_CONTROL_TBL, PAL_DISTRFITCENSORED_ESTIMATION_TBL,
PAL_DISTRFITCENSORED_STATISTICS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_DISTRFITCENSORED_ESTIMATION_TBL;
SELECT * FROM PAL_DISTRFITCENSORED_STATISTICS_TBL;

```

Expected Result

PAL_DISTRFITCENSORED_ESTIMATION_TBL:

NAME	VALUE
DISTRIBUTIONNAME	WEIBULL
SCALE	277.697
SHAPE	2.11995

PAL_DISTRFITCENSORED_STATISTICS_TBL:

NAME	VALUE

Note: The median rank estimation method only supports Weibull distribution, and this example does not have statistics output.

Example 2: Maximum Likelihood Estimation Method

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_DISTRFITCENSORED_DATA_T;
CREATE TYPE PAL_DISTRFITCENSORED_DATA_T AS TABLE(
"LEFT" DOUBLE,
"RIGHT" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_DISTRFITCENSORED_ESTIMATION_T;
CREATE TYPE PAL_DISTRFITCENSORED_ESTIMATION_T AS TABLE(
"NAME" VARCHAR(50),
"VALUE" VARCHAR(50)
);
DROP TYPE PAL_DISTRFITCENSORED_STATISTICS_T;
CREATE TYPE PAL_DISTRFITCENSORED_STATISTICS_T AS TABLE(
"NAME" VARCHAR(50),
"VALUE" DOUBLE
);
DROP TABLE PDATA_TBL;
CREATE COLUMN TABLE PDATA_TBL(
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_DISTRFITCENSORED_DATA_T', 'IN');
INSERT INTO PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_DISTRFITCENSORED_ESTIMATION_T',
'OUT');
INSERT INTO PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_DISTRFITCENSORED_STATISTICS_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DISTRFITCENSORED_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL',
'DISTRFITCENSORED','DM_PAL','PAL_DISTRFITCENSORED_PROC',PDATA_TBL);
DROP TABLE PAL_DISTRFITCENSORED_DATA_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_DATA_TBL LIKE
PAL_DISTRFITCENSORED_DATA_T;
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (NULL,0.04);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (100,100);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (0.04,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (15,30);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (0.04,0.04);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (100,100);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (1,1);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (1,1);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (0.04,0.04);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (1,NULL);
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
INSERT INTO PAL_CONTROL_TBL VALUES ('DISTRIBUTIONNAME', NULL, NULL,'WEIBULL');
INSERT INTO PAL_CONTROL_TBL VALUES ('OPTIMAL_METHOD',0, NULL, NULL);
DROP TABLE PAL_DISTRFITCENSORED_ESTIMATION_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_ESTIMATION_TBL LIKE
PAL_DISTRFITCENSORED_ESTIMATION_T;
DROP TABLE PAL_DISTRFITCENSORED_STATISTICS_TBL;
```

```

CREATE COLUMN TABLE PAL_DISTRFITCENSORED_STATISTICS_TBL LIKE
PAL_DISTRFITCENSORED_STATISTICS_T;
CALL DM_PAL.PAL_DISTRFITCENSORED_PROC(PAL_DISTRFITCENSORED_DATA_TBL,
PAL_CONTROL_TBL, PAL_DISTRFITCENSORED_ESTIMATION_TBL,
PAL_DISTRFITCENSORED_STATISTICS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_DISTRFITCENSORED_ESTIMATION_TBL;
SELECT * FROM PAL_DISTRFITCENSORED_STATISTICS_TBL;

```

Expected Result

PAL_DISTRFITCENSORED_ESTIMATION_TBL:

	NAME	VALUE
1	DISTRIBUTIONNAME	WEIBULL
2	SCALE	8.42129
3	SHAPE	0.335718

PAL_DISTRFITCENSORED_STATISTICS_TBL:

	NAME	VALUE
1	LOGLIKELIHOOD	-23.629601580055684

3.7.5 Grubbs' Test

Grubbs' test is used to detect outliers from a given univariate data set $Y = \{Y_1, Y_2, \dots, Y_n\}$. The algorithm assumes that Y comes from Gaussian distribution.

The basic steps of the algorithm are as follows:

1. Define the hypothesis.
 H_0 : There are no outliers in the data set Y .
 H_1 : There is at least one outlier in data set Y .
2. Calculate Grubbs' test statistic.

$$G = \frac{\max_{i=1, \dots, n} |Y_i - \bar{Y}|}{s}$$

$$\bar{Y} = \frac{1}{n} \sum_{i=1}^n Y_i, s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (Y_i - \bar{Y})^2}$$

Here

3. Given the significance level α , if

$$G > T = \frac{n-1}{\sqrt{n}} \sqrt{\frac{\left(\frac{t_{\alpha/2, n-2}}{2n}\right)^2}{n-2 + \left(\frac{t_{\alpha/2, n-2}}{2n}\right)^2}}$$

The algorithm will reject the hypothesis at the significance level α , which means that the data set contains

outlier. Here $\frac{t_{\alpha/2}}{2n}, n-2$ denotes the quantile value of t-distribution with $n-2$ degrees and a significance level $\frac{\alpha}{2n}$.

The above is called two-sided test. There is another version called one-sided test for minimum value or maximum value.

$$G = \frac{\bar{Y} - Y_{\min}}{s}$$

- For minimum value:

$$G = \frac{Y_{\max} - \bar{Y}}{s}$$

- For maximum value:

Note that you must replace $\frac{\alpha}{2n}$ with $\frac{\alpha}{n}$ for one-sided test.

Suppose Y_{\max} is an outlier from Grubbs' test, you can calculate the statistic value U as shown below:

1. Remove Y_{\max} from original data and get $Z = \{Z_1, Z_2, \dots, Z_{n-1}\}$.

$$U = \frac{\sum_{i=1}^{n-1} (Z_i - \bar{Z})^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}$$

2. Calculate

PAL also supports the repeat version of two-sided test. The steps are as follows:

1. Perform two-sided test for data Y .
2. If there is outlier in Y , remove it from Y and go to Step 1; Otherwise go to Step 3.
3. Return all the outliers.

Prerequisites

- The length of the input data must not be less than 5.
- The input data does not contain null value.

GRUBBTEST

This function performs Grubbs' test for identifying outliers from input data.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'GRUBBTEST',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Outlier OUTPUT table type>	OUT
4	<schema_name>	<Statistics OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <output table>, <outlier output table>, <statistics output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description	Constraint
Data	1st column	Integer, bigint, varchar, or nvarchar	ID	This must be the first column.
	2nd column	Integer or double	Value data	

Parameter Table

Mandatory Parameter

None.

Optional Parameter

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
METHOD	Integer	1	<ul style="list-style-type: none"> • 1: Two-sided test • 2: One-sided test for minimum value • 3: One-sided test for maximum value • 4: Repeat two-sided test
ALPHA	Double	0.05	Significance level

Output Table

Table	Column	Column Data Type	Description
Outlier	1st column	Integer	ID
	2nd column	Integer, bigint, varchar, or nvarchar	The ID of the original input data. This must be of the same data type as the input data.
	3rd column	Integer or double	The value of the original input data. This must be of the same data type as the input data.
Statistics	1st column	Integer	ID
	2nd column	Varchar or nvarchar	Statistic name
	3rd column	Double	Statistic value

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_GRUBBS_DATA_T;
CREATE TYPE PAL_GRUBBS_DATA_T AS TABLE(
    "ID" INTEGER,
    "VAL" DOUBLE
);
DROP TYPE PAL_GRUBBS_CONTROL_T;
CREATE TYPE PAL_GRUBBS_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_GRUBBS_OUTLIERS_T;
CREATE TYPE PAL_GRUBBS_OUTLIERS_T AS TABLE(
    "ID" INTEGER,
    "SOURCE_ID" INTEGER,
    "VALUE" DOUBLE
);
DROP TYPE PAL_GRUBBS_STATISTICS_T;
CREATE TYPE PAL_GRUBBS_STATISTICS_T AS TABLE(
    "ID" INTEGER,
    "NAME" VARCHAR(50),
    "VALUE" DOUBLE
);
DROP TABLE PAL_GRUBBS_PDATA_TBL;
CREATE COLUMN TABLE PAL_GRUBBS_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
)

```

```

);
INSERT INTO PAL_GRUBBS_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_GRUBBS_DATA_T', 'IN');
INSERT INTO PAL_GRUBBS_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_GRUBBS_CONTROL_T',
'IN');
INSERT INTO PAL_GRUBBS_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_GRUBBS_OUTLIERS_T',
'OUT');
INSERT INTO PAL_GRUBBS_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_GRUBBS_STATISTICS_T',
'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_GRUBBS_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'GRUBBSTEST', 'DM_PAL',
'PAL_GRUBBS_PROC', 'PAL_GRUBBS_PDATA_TBL');
DROP TABLE PAL_GRUBBS_DATA_TBL;
CREATE COLUMN TABLE PAL_GRUBBS_DATA_TBL LIKE PAL_GRUBBS_DATA_T;
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (100, 4.254843);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (200, 0.135000);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (300, 11.072257);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (400, 14.797838);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (500, 12.125133);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (600, 14.265839);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (700, 7.731352);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (800, 6.856739);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (900, 15.094403);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (101, 8.149382);
INSERT INTO PAL_GRUBBS_DATA_TBL VALUES (201, 9.160144);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('ALPHA', null, 0.2, null);
INSERT INTO #PAL_CONTROL_TBL VALUES ('METHOD', 2, null, null);
DROP TABLE PAL_GRUBBS_OUTLIERS_TBL;
CREATE COLUMN TABLE PAL_GRUBBS_OUTLIERS_TBL LIKE PAL_GRUBBS_OUTLIERS_T;
DROP TABLE PAL_GRUBBS_STATISTICS_TBL;
CREATE COLUMN TABLE PAL_GRUBBS_STATISTICS_TBL LIKE PAL_GRUBBS_STATISTICS_T;
CALL "DM_PAL".PAL_GRUBBS_PROC(PAL_GRUBBS_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_GRUBBS_OUTLIERS_TBL, PAL_GRUBBS_STATISTICS_TBL) with overview;
SELECT * FROM PAL_GRUBBS_OUTLIERS_TBL;
SELECT * FROM PAL_GRUBBS_STATISTICS_TBL;

```

Expected Result

PAL_GRUBBS_OUTLIERS_TBL:

ID	SOURCE_ID	VALUE
0	200	0.135

PAL_GRUBBS_STATISTICS_TBL:

ID	NAME	VALUE
0	MEAN	9.4220845454545
0	STANDARD_SAMPLE_VARIANCE	4.675935228561435
0	T	1.9102192033966785
0	G	1.9861448226928808
0	U	0.5660751617619288

3.7.6 Kaplan-Meier Survival Analysis

The Kaplan-Meier estimator is a non-parametric statistic used to estimate the survival function from lifetime data. It is often used to measure the time-to-death of patients after treatment or time-to-failure of machine parts.

Sometimes subjects under study are refused to remain in the study or some of the subjects may not experience the event before the end of the study, or you lose touch with them midway in the study. These situations are labeled as censored observations.

The Kaplan-Meier estimator of survival function is calculated as:

$$\hat{S}(t) = \prod_{t_i \leq t} \frac{n_i - d_i}{n_i}$$

Where n_i is the number of subjects at risk and d_i is the number of subjects who fail, both at time t_i .

The Kaplan-Meier estimator $\hat{S}(t)$ can be regarded as a point estimate of the survival function $s(t)$ at any time t . We can construct 95% confidence intervals around each of these estimates. To compute the confidence intervals, Greenwood's Formula gives an asymptotic estimate of the variance of $\hat{S}(t)$ for large groups:

$$\widehat{\text{Var}}[\hat{S}(t)] = \hat{S}(t)^2 \sum_{t_i \leq t} \frac{d_i}{n_i(n_i - d_i)}$$

So the Greenwood's confidence interval is:

$$\hat{S}(t) \pm z_{\alpha/2} \sqrt{\widehat{\text{Var}}[\hat{S}(t)]}$$

Where $z_{\alpha/2}$ is the $\alpha/2$ -th quantile of the normal distribution.

However the endpoints of Greenwood's confidence interval can be negative or greater than one. Here we use another confidence interval based on the large sample normal distribution of $\log(-\log(\hat{S}(t)))$ with:

$$\widehat{\text{Var}}[\log(-\log(\hat{S}(t)))] = \frac{1}{(\log(\hat{S}(t)))^2} \sum_{t_i \leq t} \frac{d_i}{n_i(n_i - d_i)}$$

So we get:

$$ci_lower = \log(-\log(\hat{S}(t))) + z_{\alpha/2} \sqrt{\widehat{\text{Var}}[\log(-\log(\hat{S}(t)))]}$$

$$ci_upper = \log(-\log(\hat{S}(t))) - z_{\alpha/2} \sqrt{\widehat{\text{Var}}[\log(-\log(\hat{S}(t)))]}$$

Transform endpoints (ci_lower, ci_upper) back to obtain confidence interval:

`(exp(-exp(ci_lower)), exp(-exp(ci_upper)))`

Equality comparison of two or more Kaplan-Meier survival functions can be done using a statistical hypothesis test called the log rank test by weighting all time points the same. It is used to test the null hypothesis where there is no difference between the population survival functions.

For $i=1, 2, \dots, g$ and $j=1, 2, \dots, k$, where g is number of groups and k is number of distinct failure times.

n_{ij} = number at risk in i th group at j th ordered failure time

o_{ij} = observed number of failures in i th group at j th ordered failure time

e_{ij} = expected number of failures in i th group at j th ordered failure time = $\frac{n_{ij}}{\sum_{i=1}^g n_{ij}} (\sum_{i=1}^g o_{ij})$

$$n_j = \sum_{i=1}^g n_{ij}$$

$$o_j = \sum_{i=1}^g o_{ij}$$

$$O_i - E_i = \sum_{j=1}^k o_{ij} - e_{ij}$$

$$\widehat{Var}(O_i - E_i) = \sum_{j=1}^k \frac{n_{i1}(n_j - n_{i1})o_{ij}(n_j - o_{ij})}{n_j^2(n_j - 1)}$$

$$\widehat{Cov}(O_i - E_i, O_p - E_p) = \sum_{j=1}^k \frac{-n_{i1}n_{p1}o_{ij}(n_j - o_{ij})}{n_j^2(n_j - 1)}$$

$$d = (O_1 - E_1, O_2 - E_2, \dots, O_{g-1} - E_{g-1})$$

$$V = ((v_{ip}))' \text{ where } v_{ii} = \widehat{Var}(O_i - E_i) \text{ and } v_{ip} = \widehat{Cov}(O_i - E_i, O_p - E_p) \text{ for } i=1, 2, \dots, g-1; p = 1, 2, \dots, g-1.$$

The, the log rank statistics is given by the matrix product formula:

$$d' V^{-1} d$$

Which has approximately a chi-squared distribution with $g-1$ degrees of freedom under the null hypothesis that all g groups have a common survival function.

Comparison of two Kaplan-Meier survival functions can be simplified as:

$$\frac{(O_1 - E_1)^2}{\widehat{Var}(O_1 - E_1)}$$

$$\text{Where } \widehat{Var}(O_i - E_i) = \sum_{j=1}^k \frac{n_{i1}n_{2j}(o_{1j} + o_{2j})(n_{1j} + n_{2j} - o_{1j} - o_{2j})}{(n_{1j} + n_{2j})^2(n_{1j} + n_{2j} - 1)}$$

Prerequisite

No missing or null data in the inputs.

KMSURV

This function estimates the probability of surviving time t (t is an event time) using Kaplan-Meier estimator and compares several groups of survival functions using log rank test. This function does log rank test if the lifetime data comes from multiple groups, otherwise it skips doing it.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'KMSURV', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Input data table type>	IN
2	<schema_name>	<Parameter table type>	IN
3	<schema_name>	<Survival estimates OUTPUT table type>	OUT
4	<schema_name>	<Log rank test statistics_1 OUTPUT table type>	OUT
5	<schema_name>	<Log rank test statistics_2 OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input data tables>, <parameter table>,  
<survival estimates output table>, <log rank test statistics details output  
table>, <log rank test statistics output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer or double	Follow-up time
	2nd column	Integer	Status indicator

Table	Column	Column Data Type	Description
	3rd column	Integer	Occurrence number of events or censoring at the follow-up time. It also allows for multiple rows for one follow-up time.
	4th column	Integer or varchar	Group

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
EVENT_INDICATOR	Integer	1	Specifies one value to indicate an event has occurred.
CONF_LEVEL	Double	0.95	Specifies confidence level for a two-sided confidence interval on the survival estimate.

Output Tables

Table	Column	Column Data Type	Description	Constraint
Survival Estimates	1st column	Integer or varchar	Group	
	2nd column	Integer or double	Event occurrence time	Survival estimates at all event times are output.
	3rd column	Integer	Number at risk (total number of survivors at the beginning of each period)	
	4th column	Integer	Number of event occurrences	

Table	Column	Column Data Type	Description	Constraint
	5th column	Double	Probability of surviving beyond event occurrence time	
	6th column	Double	Standard error for the survivor estimate	
	7th column	Double	Lower of confidence interval	
	8th column	Double	Upper of confidence interval	
Log Rank Test Statistics_1	1st column	Integer or varchar	Group	
	2nd column	Integer	All individuals in the lifetime study	
	3rd column	Integer	Observed event number	
	4th column	Double	Expected event number	
	5th column	Double	Log rank test statistics $\frac{(O_1 - E_1)^2}{\sqrt{Var(O_1 - E_1)}}$	
Log Rank Test Statistics_2	1st column	Varchar	Name Examples: Chisq, df, p-value	
	2nd column	Double	Value	

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_KMSURV_DATA_T;
CREATE TYPE PAL_KMSURV_DATA_T AS TABLE ("TIME" INTEGER,
  "STATUS" INTEGER,
  "OCCURENCES" INTEGER,
  "GROUP" INTEGER);

DROP TYPE PAL_KMSURV_ESTIMATES_T;
```

```

CREATE TYPE PAL_KMSURV_ESTIMATES_T AS TABLE("GROUP" INTEGER, "TIME" INTEGER,
"RISKNUM" INTEGER, "EVENTSNUM" INTEGER,
"PROB" DOUBLE, "STDERR" DOUBLE,
"LOWERLIMIT" DOUBLE, "UPPERLIMIT" DOUBLE);
DROP TYPE PAL_KMSURV_LOGRANK_STAT1_T;
CREATE TYPE PAL_KMSURV_LOGRANK_STAT1_T AS TABLE("GROUP" INTEGER, "TOTALRISK"
INTEGER, "OBSERVED" INTEGER, "EXPECTED" DOUBLE, "LOGRANKSTAT" DOUBLE);
DROP TYPE PAL_KMSURV_LOGRANK_STAT2_T;
CREATE TYPE PAL_KMSURV_LOGRANK_STAT2_T AS TABLE("STATISTICS" VARCHAR(20),
"VALUE" DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE( "NAME" VARCHAR(100), "INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TABLE PAL_KMSURV_PDATA_TBL;
CREATE COLUMN TABLE PAL_KMSURV_PDATA_TBL( "POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_KMSURV_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_KMSURV_DATA_T',
'in');
INSERT INTO PAL_KMSURV_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'in');
INSERT INTO PAL_KMSURV_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_KMSURV_ESTIMATES_T',
'out');
INSERT INTO PAL_KMSURV_PDATA_TBL VALUES (4, 'DM_PAL',
'PAL_KMSURV_LOGRANK_STAT1_T', 'out');
INSERT INTO PAL_KMSURV_PDATA_TBL VALUES (5, 'DM_PAL',
'PAL_KMSURV_LOGRANK_STAT2_T', 'out');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_KMSURV_PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'KMSURV',
'DM_PAL', 'PAL_KMSURV_PROC', PAL_KMSURV_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL( "NAME" VARCHAR (100),
"INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
DROP TABLE PAL_KMSURV_DATA_TBL;
CREATE COLUMN TABLE PAL_KMSURV_DATA_TBL LIKE PAL_KMSURV_DATA_T;
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(9, 1, 1, 2);
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(10, 1, 1, 1 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(1, 1, 2, 0 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(31, 0, 1, 1 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(2, 1, 1, 0 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(25, 1, 3, 1 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(255, 0, 1, 0 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(90, 1, 1, 0 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(22, 1, 1, 1 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(100, 0, 1, 1 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(28, 0, 1, 0 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(5, 1, 1, 1 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(7, 1, 1, 1 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(11, 0, 1, 0 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(20, 0, 1, 0 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(30, 1, 2, 2 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(101, 0, 1, 2 );
INSERT INTO PAL_KMSURV_DATA_TBL VALUES(8, 0, 1, 1 );
DROP TABLE PAL_KMSURV_ESTIMATES_TBL;
CREATE COLUMN TABLE PAL_KMSURV_ESTIMATES_TBL LIKE PAL_KMSURV_ESTIMATES_T;
DROP TABLE PAL_KMSURV_LOGRANK_STAT1_TBL;
CREATE COLUMN TABLE PAL_KMSURV_LOGRANK_STAT1_TBL LIKE PAL_KMSURV_LOGRANK_STAT1_T;
DROP TABLE PAL_KMSURV_LOGRANK_STAT2_TBL;
CREATE COLUMN TABLE PAL_KMSURV_LOGRANK_STAT2_TBL LIKE PAL_KMSURV_LOGRANK_STAT2_T;
CALL "DM_PAL".PAL_KMSURV_PROC(PAL_KMSURV_DATA_TBL, #PAL_CONTROL_TBL,
PAL_KMSURV_ESTIMATES_TBL, PAL_KMSURV_LOGRANK_STAT1_TBL,
PAL_KMSURV_LOGRANK_STAT2_TBL) with OVERVIEW;
SELECT * FROM PAL_KMSURV_ESTIMATES_TBL;
SELECT * FROM PAL_KMSURV_LOGRANK_STAT1_TBL;
SELECT * FROM PAL_KMSURV_LOGRANK_STAT2_TBL;

```

Expected Result

PAL_KMSURV_ESTIMATES_TBL:

GROUP	TIME	RISKNUM	EVENTSNUM	PROB	STDERR	LOWERLIMIT	UPPERLIMIT
0	1	8	2	0.75	0.15309...	0.314807113...	0.930898317...
0	2	6	1	0.625	0.17116...	0.229333322...	0.860698276...
0	90	2	1	0.3...	0.23696...	0.015390059...	0.723157898...

PAL_KMSURV_LOGRANK_STAT1_TBL:

GROUP	TOTALRISK	OBSERVED	EXPECTED	LOGRANKSTAT
0	8	4	4.353652...	0.04571216146...
1	10	7	6.024638...	0.30795141540...
2	4	3	3.621709...	0.16178636178...

PAL_KMSURV_LOGRANK_STAT2_TBL:

STATISTICS	VALUE
chiSqr	0.32797068374749033
df	2
p-value	0.8487544629391057

3.7.7 Multivariate Statistics

This function calculates several basic multivariate statistics including covariance matrix and Pearson's correlation coefficient matrix. The function treats each column as a data sample.

Covariance Matrix

The covariance between two data samples (random variables) x and y is:

$$\text{cov}(x, y) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x}) * (y_i - \bar{y})$$

Suppose that each column represents a data sample (random variable), the covariance matrix Σ is defined as the covariance between any two random variables:

$$\Sigma_{ij} = \text{cov}(X_i, X_j)$$

where $X = [x_1, x_2, \dots, x_n]$.

Pearson's Correlation Coefficient Matrix

The Pearson's correlation coefficient between two data samples (random variables) x and y is the covariance of x and y divided by the product of standard deviation of x and the standard deviation y :

$$\text{cor}(x, y) = \frac{\text{cov}(X, Y)}{s_x * s_y}$$

Similar to the covariance matrix, the Pearson's correlation coefficient matrix Σ is:

$$\Sigma_{ij} = \text{cor}(X_i, X_j)$$

where $X = [X_1, X_2, \dots, X_n]$.

Prerequisites

- The input data columns are of integer or double data type.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

MULTIVARSTAT

This function reads input data and calculates the basic multivariate statistics values for each column, including covariance matrix and Pearson's correlation coefficient matrix.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'MULTIVARSTAT',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name> (<input table>, <parameter table>, <result
output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	All columns	Integer or double	Attribute data

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
RESULT_TYPE	Integer	0	<p>Definition of result matrix.</p> <ul style="list-style-type: none">• 0: Covariance matrix• 1: Pearson's correlation coefficient matrix

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	Row name of output matrix to keep the row order correctly
	Other columns	Double	Value of covariance or Pearson's correlation coefficient matrix

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_MULTIVARSTAT_DATA_T;
CREATE TYPE PAL_MULTIVARSTAT_DATA_T AS TABLE(
    "X" INTEGER,
    "Y" DOUBLE
);

DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_MULTIVARSTAT_RESULT_T;
CREATE TYPE PAL_MULTIVARSTAT_RESULT_T AS TABLE(
    "ID" VARCHAR(100),
    "X" DOUBLE,
    "Y" DOUBLE
);
DROP TABLE PAL_MULTIVARSTAT_PDATA_TBL;
CREATE COLUMN TABLE PAL_MULTIVARSTAT_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_MULTIVARSTAT_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_MULTIVARSTAT_DATA_T','IN');
INSERT INTO PAL_MULTIVARSTAT_PDATA_TBL VALUES (2,'DM_PAL', 'PAL_CONTROL_T','IN');
```

```

INSERT INTO PAL_MULTIVARSTAT_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_MULTIVARSTAT_RESULT_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_MULTIVARIATESTAT_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'MULTIVARSTAT', 'DM_PAL',
'PAL_MULTIVARIATESTAT_PROC', 'PAL_MULTIVARSTAT_PDATA_TBL');
DROP TABLE PAL_MULTIVARSTAT_DATA_TBL;
CREATE COLUMN TABLE PAL_MULTIVARSTAT DATA_TBL LIKE PAL_MULTIVARSTAT_DATA_T;
INSERT INTO PAL_MULTIVARSTAT_DATA_TBL VALUES (1,2.4);
INSERT INTO PAL_MULTIVARSTAT_DATA_TBL VALUES (5,3.5);
INSERT INTO PAL_MULTIVARSTAT_DATA_TBL VALUES (3,8.9);
INSERT INTO PAL_MULTIVARSTAT_DATA_TBL VALUES (10,-1.4);
INSERT INTO PAL_MULTIVARSTAT_DATA_TBL VALUES (-4,-3.5);
INSERT INTO PAL_MULTIVARSTAT_DATA_TBL VALUES (11,32.8);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('RESULT_TYPE',0,null,null); //default value
is 0, it can be {0,1}
DROP TABLE PAL_MULTIVARSTAT_RESULT_TBL;
CREATE COLUMN TABLE PAL_MULTIVARSTAT_RESULT_TBL LIKE PAL_MULTIVARSTAT_RESULT_T;
CALL DM_PAL.PAL_MULTIVARIATESTAT_PROC(PAL_MULTIVARSTAT_DATA_TBL,
#PAL_CONTROL_TBL, PAL_MULTIVARSTAT_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_MULTIVARSTAT_RESULT_TBL;

```

Expected Result

ID		X	Y
ID			
1	X	31.866666666666667	44.47333333333333
2	Y	44.47333333333333	176.67766666666662

3.7.8 Quantile Function

This algorithm evaluates the inverse $F^{-1}(x)$ of the cumulative distribution function (CDF) or the inverse $\bar{F}^{-1}(x)$ of the complementary cumulative distribution function (CCDF) for a given probability p and probability distribution.

The CDF $F(x)$ of a real-valued random variable x is given by

$$F(x) = P[X \leq x]$$

The CCDF $\bar{F}(x)$ of a real-valued random variable x is given by

$$\bar{F}(x) = P[X > x] = 1 - F(x)$$

Prerequisites

- The input data is numeric, not categorical.

- The input data and distribution parameters do not contain null data. The algorithm will issue errors when encountering null values.

DISTRQUANTILE

This function calculates quantiles for a given distribution.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'DISTRQUANTILE',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Distribution parameter INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <distribution parameter input table>, <parameter table>, <result output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Data	1st column	Double	Probabilities. Must be in the open interval (0.0, 1.0).
Distribution Parameter	1st column	Varchar or nvarchar	Names of the distribution parameters. See Distribution Parameters Definition table for details.
	2nd column	Varchar or nvarchar	Values of the distribution parameters. See Distribution Parameters Definition table for details.

Distribution Parameters Definition

The following parameters are mandatory and must be given a value.

Distribution	Parameter Name	Parameter Value	Constraint
Uniform	"DistributionName"	"Uniform"	
	"Min"	"0.0"	Min < Max
	"Max"	"1.0"	
Normal	"DistributionName"	"Normal"	
	"Mean"	"0.0"	
	"Variance"	"1.0"	Variance > 0
Weibull	"DistributionName"	"Weibull"	
	"Shape"	"1.0"	Shape > 0
	"Scale"	"1.0"	Scale > 0
Gamma	"DistributionName"	"Gamma"	
	"Shape"	"1.0"	Shape > 0
	"Scale"	"1.0"	Scale > 0

i Note

The names and values of the distribution parameters are not case sensitive.

Parameter Table

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
LOWER_UPPER	Integer	0	<ul style="list-style-type: none"> • 0: Calculates the inverse of CDF • 1: Calculates the inverse of CCDF

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Double	Input probabilities
	2nd column	Double	Quantiles

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
```

```

DROP TYPE PAL_DISTRQUANTILE_DATA_T;
CREATE TYPE PAL_DISTRQUANTILE_DATA_T AS TABLE("DATACOL" DOUBLE);
DROP TYPE PAL_DISTRQUANTILE_DISTRPARAM_T;
CREATE TYPE PAL_DISTRQUANTILE_DISTRPARAM_T AS TABLE("NAME" VARCHAR(50), "VALUE"
VARCHAR(50));
DROP TYPE PAL_DISTRQUANTILE_RESULT_T;
CREATE TYPE PAL_DISTRQUANTILE_RESULT_T AS TABLE("INPUTDATA" DOUBLE, "QUANTILE"
DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
"NAME" VARCHAR(50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_DISTRQUANTILE_PDATA_TBL;
CREATE COLUMN TABLE PAL_DISTRQUANTILE_PDATA_TBL(
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_DISTRQUANTILE_PDATA_TBL VALUES (1,
'DM_PAL','PAL_DISTRQUANTILE_DATA_T', 'IN');
INSERT INTO PAL_DISTRQUANTILE_PDATA_TBL VALUES (2,
'DM_PAL','PAL_DISTRQUANTILE_DISTRPARAM_T', 'IN');
INSERT INTO PAL_DISTRQUANTILE_PDATA_TBL VALUES (3, 'DM_PAL','PAL_CONTROL_T',
'IN');
INSERT INTO PAL_DISTRQUANTILE_PDATA_TBL VALUES (4,
'DM_PAL','PAL_DISTRQUANTILE_RESULT_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DISTRQUANTILE_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'DISTRQUANTILE',
'DM_PAL','PAL_DISTRQUANTILE_PROC',PAL_DISTRQUANTILE_PDATA_TBL);
DROP TABLE PAL_DISTRQUANTILE_DATA_TBL;
CREATE COLUMN TABLE PAL_DISTRQUANTILE_DATA_TBL LIKE PAL_DISTRQUANTILE_DATA_T;
INSERT INTO PAL_DISTRQUANTILE_DATA_TBL VALUES (0.3);
INSERT INTO PAL_DISTRQUANTILE_DATA_TBL VALUES (0.5);
INSERT INTO PAL_DISTRQUANTILE_DATA_TBL VALUES (0.632);
INSERT INTO PAL_DISTRQUANTILE_DATA_TBL VALUES (0.8);
DROP TABLE PAL_DISTRQUANTILE_DISTRPARAM_TBL;
CREATE COLUMN TABLE PAL_DISTRQUANTILE_DISTRPARAM_TBL LIKE
PAL_DISTRQUANTILE_DISTRPARAM_T;
INSERT INTO PAL_DISTRQUANTILE_DISTRPARAM_TBL VALUES ('DistributionName',
'Weibull');
INSERT INTO PAL_DISTRQUANTILE_DISTRPARAM_TBL VALUES ('Shape', '2.11995');
INSERT INTO PAL_DISTRQUANTILE_DISTRPARAM_TBL VALUES ('Scale', '277.698');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
"NAME" VARCHAR(50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LOWER_UPPER',0,null,null);
DROP TABLE PAL_DISTRQUANTILE_RESULT_TBL;
CREATE COLUMN TABLE PAL_DISTRQUANTILE_RESULT_TBL LIKE PAL_DISTRQUANTILE_RESULT_T;
CALL DM_PAL.PAL_DISTRQUANTILE_PROC(PAL_DISTRQUANTILE_DATA_TBL,
PAL_DISTRQUANTILE_DISTRPARAM_TBL, #PAL_CONTROL_TBL,
PAL_DISTRQUANTILE_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_DISTRQUANTILE_RESULT_TBL;

```

Expected Result

	INPUTDATA	QUANTILE
1	0.3	170.7558541487521
2	0.5	233.60850568865035
3	0.632	277.6550753190647
4	0.8	347.5864951691035

3.7.9 Univariate Statistics

This function calculates several basic univariate statistics including mean, median, variance, standard deviation, skewness and kurtosis. The function treats each column as one dataset and calculates the statistics respectively.

Mean

$$\text{mean } (\mathbf{x}) = \frac{\sum_{i=1}^n x_i}{n}$$

where x_i is the i-th element of the dataset and n is the size of the dataset.

Median

The median is defined as the numerical value separating the higher half of a dataset from the lower half. If there is an even number of observations, the median is defined to be the mean of the two middle elements.

Lower Quartile and Upper Quartile

Use the median to divide the elements of the dataset into two halves. Do not include the median in either half. The lower quartile value is the median of the lower half of the data. The upper quartile value is the median of the upper half of the data.

Variance (population)

$$\text{var } (\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

where n is the size of the dataset and \bar{x} is the mean of \mathbf{x} .

Variance (sample)

$$\text{var } (\mathbf{x}) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

where n is the size of the dataset and \bar{x} is the mean of \mathbf{x} .

Standard Deviation (population)

$$\sigma_x = \sqrt{\text{var } (\mathbf{x}) \text{ (population)}}$$

Standard Deviation (sample)

$$s_x = \sqrt{\text{var}(x) \text{ (sample)}}$$

Skewness

Skewness is a measure of the degree of asymmetry. Suppose that

$$y = \frac{\sqrt{n} * \sum x'^3}{(\sum x'^2)^{3/2}}$$

where $x' = x - \bar{x}$ and n is the size of the dataset. There are three definitions of skewness:

Definition 1	y
Definition 2	$y * \frac{\sqrt{n} * (n-1)}{n-2}$
Definition 3	$y * (1 - \frac{1}{n})^{3/2}$

Kurtosis

Kurtosis is a measure of the peakedness or flatness compared to a normal distribution. Suppose that

$$r = \frac{n * \sum x'^4}{(\sum x'^2)^2}$$

where $x' = x - \bar{x}$ and n is the number of elements. There are three definitions of kurtosis:

Definition 1	$r-3$
Definition 2	$((n+1)*(r-3)+6) * \frac{n-1}{(n-2)*(n-3)}$
Definition 3	$r * \left(1 - \frac{1}{n}\right)^2 - 3$

Prerequisites

- The input data columns are of integer or double data type.

- The input data does not contain null value. The algorithm will issue errors when encountering null values.

UNIVARSTAT

This function reads input data and calculates the basic univariate statistics values for each column, including mean, median, variance, standard deviation, skewness, and kurtosis.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'UNIVARSTAT',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name> (<input table>, <parameter table>, <result
output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and result tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	All columns	Integer or double	Attribute data

Parameter Table

Mandatory Parameters

None.

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
SKEWNESS_TYPE	Integer	1	<p>Calculation type for skewness.</p> <ul style="list-style-type: none"> • 0: Definition 1 • 1: Definition 2 • 2: Definition 3
KURTOSIS_TYPE	Integer	1	<p>Calculation type for kurtosis.</p> <ul style="list-style-type: none"> • 0: Definition 1 • 1: Definition 2 • 2: Definition 3
DATASET_TYPE	Integer	0	<p>Type of the input dataset.</p> <ul style="list-style-type: none"> • 0: Sample dataset • 1: Population dataset

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	Name of univariate statistics
	Other columns	Double	Value for different statistics. The column number and order must be the same as those of the input table.

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_UNIVARSTAT_DATA_T;
CREATE TYPE PAL_UNIVARSTAT_DATA_T AS TABLE("X" INTEGER, "Y" DOUBLE);

DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_UNIVARSTAT_RESULT_T;
CREATE TYPE PAL_UNIVARSTAT_RESULT_T AS TABLE(
    "STATISTICSNAME" VARCHAR(100),
    "VALUEX" DOUBLE,
    "VALUEY" DOUBLE
);
DROP TABLE PAL_UNIVARSTAT_PDATA_TBL;
CREATE COLUMN TABLE PAL_UNIVARSTAT_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_UNIVARSTAT_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_UNIVARSTAT_DATA_T','IN');

```

```

INSERT INTO PAL_UNIVARSTAT_PDATA_TBL VALUES (2,'DM_PAL', 'PAL_CONTROL_T','IN');
INSERT INTO PAL_UNIVARSTAT_PDATA_TBL VALUES (3,'DM_PAL',
'PAL_UNIVARSTAT_RESULT_T','OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_UNIVARIATESTAT_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'UNIVARSTAT', 'DM_PAL',
'PAL_UNIVARIATESTAT_PROC', PAL_UNIVARSTAT_PDATA_TBL);
DROP TABLE PAL_UNIVARSTAT_DATA_TBL;
CREATE COLUMN TABLE PAL_UNIVARSTAT_DATA_TBL LIKE PAL_UNIVARSTAT_DATA_T;
INSERT INTO PAL_UNIVARSTAT_DATA_TBL VALUES (1,2.4);
INSERT INTO PAL_UNIVARSTAT_DATA_TBL VALUES (5,3.5);
INSERT INTO PAL_UNIVARSTAT_DATA_TBL VALUES (3,8.9);
INSERT INTO PAL_UNIVARSTAT_DATA_TBL VALUES (10,-1.4);
INSERT INTO PAL_UNIVARSTAT_DATA_TBL VALUES (-4,-3.5);
INSERT INTO PAL_UNIVARSTAT_DATA_TBL VALUES (11,32.8);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
  "NAME" VARCHAR (50),
  "INTARGS" INTEGER,
  "DOUBLEARGS" DOUBLE,
  "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('SKEWNESS_TYPE',2,null,null); --default
value is 1, it can be {0,1,2}
INSERT INTO #PAL_CONTROL_TBL VALUES ('KURTOSIS_TYPE',2,null,null); --default
value is 1, it can be {0,1,2}
INSERT INTO #PAL_CONTROL_TBL VALUES ('DATASET_TYPE',0,null,null); --default
value is 0, it can be {0,1}
DROP TABLE PAL_UNIVARSTAT_RESULT_TBL;
CREATE COLUMN TABLE PAL_UNIVARSTAT_RESULT_TBL LIKE PAL_UNIVARSTAT_RESULT_T;
CALL DM_PAL.PAL_UNIVARIATESTAT_PROC(PAL_UNIVARSTAT_DATA_TBL, #PAL_CONTROL_TBL,
PAL_UNIVARSTAT_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_UNIVARSTAT_RESULT_TBL;

```

Expected Result

	STATISTICSNAME	VALUEX	VALUEY
1	mean	4.33333333333334	7.116666666666666
2	median	4	2.95
3	lower quartile	1	-1.4
4	upper quartile	10	8.9
5	variance	31.866666666666667	176.67766666666662
6	standard deviation	5.645056834671079	13.29201514694693
7	skewness	-0.12929773682816026	1.0631792841005492
8	kurtosis	-1.6942557611619775	-0.5772430525222081

3.7.10 Variance Equal Test

This function is used to test the equality of two random variances using F-test. The null hypothesis is that two independent normal variances are equal. The observed sums of some selected squares are then examined to see whether their ratio is significantly incompatible with this null hypothesis.

Let x_1, x_2, \dots, x_n and y_1, y_2, \dots, y_n be independent and identically distributed samples from two populations.

Let the sample mean of x and y be:

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n} \quad \text{and} \quad \bar{y} = \frac{\sum_{i=1}^m y_i}{m}$$

Therefore, the sample variance of x and y are:

$$s_x^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \quad \text{and} \quad s_y^2 = \frac{1}{m-1} \sum_{i=1}^m (y_i - \bar{y})^2$$

Then the test statistics is:

$$F = \frac{s_x^2}{s_y^2}$$

The F value will be used to calculate a p-value by comparing the value of F to an F-distribution. The degree of freedom is set to $(n-1)$ and $(m-1)$.

Prerequisites

- The input data has two tables and each table has only one column with the type of integer or double.
- The input data does not contain null value. The algorithm will issue errors when encountering null values.

VAREQUALTEST

This function tests the equality of variances between two input data.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'VAREQUALTEST',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<input1 INPUT table type>	IN
2	<schema_name>	<input2 INPUT table type>	IN
3	<schema_name>	<PARAMETER table type>	IN
4	<schema_name>	<StatValue OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name> (<input1 table>, <input2 table>, <parameter table>, <statvalue output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input1, input2, parameter, and statvalue tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description
Input1	1st column	Integer or double	Attribute data
Input2	1st column	Integer or double	Attribute data

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
TEST_TYPE	Integer	0	<p>The alternative hypothesis type.</p> <ul style="list-style-type: none">• 0: Two sides• 1: Less• 2: Greater

Output Table

Table	Column	Column Data Type	Description
Stat Value	1st column	Varchar or nvarchar	Name of statistics
	2nd column	Double	Value of statistics

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_VAREQUALTEST_DATA1_T;
CREATE TYPE PAL_VAREQUALTEST_DATA1_T AS TABLE ("X" INTEGER);
DROP TYPE PAL_VAREQUALTEST_DATA2_T;
CREATE TYPE PAL_VAREQUALTEST_DATA2_T AS TABLE ("Y" DOUBLE);
```

```

DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_VAREQUALTEST_STATVALUE_T;
CREATE TYPE PAL_VAREQUALTEST_STATVALUE_T AS TABLE ("NAME" VARCHAR(100), "VALUE"
DOUBLE);
DROP TABLE PAL_VAREQUALTEST_PDATA_TBL;
CREATE COLUMN TABLE PAL_VAREQUALTEST_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_VAREQUALTEST_PDATA_TBL VALUES (1,'DM_PAL',
'PAL_VAREQUALTEST_DATA1_T', 'IN');
INSERT INTO PAL_VAREQUALTEST_PDATA_TBL VALUES (2,'DM_PAL',
'PAL_VAREQUALTEST_DATA2_T', 'IN');
INSERT INTO PAL_VAREQUALTEST_PDATA_TBL VALUES (3,'DM_PAL', 'PAL_CONTROL_T',
'IN');
INSERT INTO PAL_VAREQUALTEST_PDATA_TBL VALUES (4,'DM_PAL',
'PAL_VAREQUALTEST_STATVALUE_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_VAREQUALTEST_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'VAREQUALTEST', 'DM_PAL',
'PAL_VAREQUALTEST_PROC', 'PAL_VAREQUALTEST_PDATA_TBL');
DROP TABLE PAL_VAREQUALTEST_DATA1_TBL;
CREATE COLUMN TABLE PAL_VAREQUALTEST_DATA1_TBL LIKE PAL_VAREQUALTEST_DATA1_T;
INSERT INTO PAL_VAREQUALTEST_DATA1_TBL VALUES (1);
INSERT INTO PAL_VAREQUALTEST_DATA1_TBL VALUES (2);
INSERT INTO PAL_VAREQUALTEST_DATA1_TBL VALUES (4);
INSERT INTO PAL_VAREQUALTEST_DATA1_TBL VALUES (7);
INSERT INTO PAL_VAREQUALTEST_DATA1_TBL VALUES (3);
DROP TABLE PAL_VAREQUALTEST_DATA2_TBL;
CREATE COLUMN TABLE PAL_VAREQUALTEST_DATA2_TBL LIKE PAL_VAREQUALTEST_DATA2_T;
INSERT INTO PAL_VAREQUALTEST_DATA2_TBL VALUES (10);
INSERT INTO PAL_VAREQUALTEST_DATA2_TBL VALUES (15);
INSERT INTO PAL_VAREQUALTEST_DATA2_TBL VALUES (12);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR (50),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('TEST_TYPE',0,null,null); //default value
is 0, it can be {0,1,2}
DROP TABLE PAL_VAREQUALTEST_STATVALUE_TBL;
CREATE COLUMN TABLE PAL_VAREQUALTEST_STATVALUE_TBL LIKE
PAL_VAREQUALTEST_STATVALUE_T;
CALL DM_PAL.PAL_VAREQUALTEST_PROC(PAL_VAREQUALTEST_DATA1_TBL,
PAL_VAREQUALTEST_DATA2_TBL, #PAL_CONTROL_TBL, PAL_VAREQUALTEST_STATVALUE_TBL)
WITH OVERVIEW;
SELECT * FROM PAL_VAREQUALTEST_STATVALUE_TBL;

```

Expected Result

	NAME	VALUE
1	F Value	0.8368421052631579
2	numerator degree of freedom	4
3	denominator degree of freedom	2
4	p-value	0.7837125674251348

3.8 Social Network Analysis Algorithms

This section describes the algorithms provided by the PAL that are mainly used for social network analysis.

3.8.1 Link Prediction

Predicting missing links is a common task in social network analysis. The Link Prediction algorithm in PAL provides four methods to compute the distance of any two nodes using existing links in a social network, and make prediction on the missing links based on these distances.

Let x and y be two nodes in a social network, and $\Gamma(x)$ be the set containing the neighbor nodes of x , the four methods to compute the distance of x and y are briefly described as follows.

Common Neighbors

The quantity is computed as the number of common neighbors of x and y :

$$|\Gamma(x) \cap \Gamma(y)|$$

Then, it is normalized by the total number of nodes.

Jaccard's Coefficient

The quantity is just a slight modification of the common neighbors:

$$\frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|}$$

Adamic/Adar

The quantity is computed as the sum of inverse log degree over all the common neighbors:

$$\sum_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log |\Gamma(z)|}$$

Katz $_\beta$

The quantity is computed as a weighted sum of the number of paths of length l connecting x and y :

$$\sum_{l=1}^{\infty} \beta^l |\text{path}_{x,y}^l|$$

Where $\beta \in [0,1]$ is the user-specified parameter, and $|\text{path}_{x,y}^l|$ is the number of paths with length l which starts from node x and ends at node y .

Prerequisites

The input data does not contain any null value.

LINKPREDICTION

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'LINKPREDICTION',
'<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<Result OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <result
output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, varchar, or nvarchar	Node 1 in existing edge (Node1 – Node2)
	2nd column	Integer, varchar, or nvarchar	Node 2 in existing edge (Node1 – Node2)

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
METHOD	Integer	Prediction method: • 1: Common Neighbors • 2: Jaccard's Coefficient • 3: Adamic/Adar • 4: Katz

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description	Dependency
THREAD_NUMBER	Integer	1	Number of threads	
BETA	Double	0.005	Parameter for the Katz method. BETA should be between 0 and 1. A smaller BETA is preferred.	Only valid when METHOD is 4.
MIN_SCORE	Double	0	The links whose scores are lower than this threshold will be filtered out from the result table.	

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	Node 1 in missing edge (Node1 – Node2)
	2nd column	Integer	Node 2 in missing edge (Node1 – Node2)
	3rd column	Double	Prediction score

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_LINK_PREDICTION_DATA_T;
CREATE TYPE PAL_LINK_PREDICTION_DATA_T AS TABLE(
  "NODE1" INTEGER,
  "NODE2" INTEGER
);
DROP TYPE PAL_CONTROL_T;
  
```

```

CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_LINK_PREDICTION_RESULT_T;
CREATE TYPE PAL_LINK_PREDICTION_RESULT_T AS TABLE(
    "NODE1" INTEGER,
    "NODE2" INTEGER,
    "SCORE" DOUBLE
);
DROP TABLE PAL_LINK_PREDICTION_PDATA_TBL;
CREATE COLUMN TABLE PAL_LINK_PREDICTION_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_LINK_PREDICTION_PDATA_TBL VALUES (1, 'DM_PAL',
'PAL_LINK_PREDICTION_DATA_T', 'in');
INSERT INTO PAL_LINK_PREDICTION_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T',
'in');
INSERT INTO PAL_LINK_PREDICTION_PDATA_TBL VALUES (3, 'DM_PAL',
'PAL_LINK_PREDICTION_RESULT_T', 'out');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_LINK_PREDICTION_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'LINKPREDICTION', 'DM_PAL',
'PAL_LINK_PREDICTION_PROC', PAL_LINK_PREDICTION_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('METHOD', 1, NULL, NULL);
DROP TABLE PAL_LINK_PREDICTION_DATA_TBL;
CREATE COLUMN TABLE PAL_LINK_PREDICTION_DATA_TBL LIKE PAL_LINK_PREDICTION_DATA_T;
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('1','2');
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('1','4');
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('2','3');
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('3','4');
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('5','1');
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('6','2');
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('7','4');
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('7','5');
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('6','7');
INSERT INTO PAL_LINK_PREDICTION_DATA_TBL VALUES ('5','4');
DROP TABLE PAL_LINK_PREDICTION_RESULT_TBL;
CREATE COLUMN TABLE PAL_LINK_PREDICTION_RESULT_TBL LIKE
PAL_LINK_PREDICTION_RESULT_T;
CALL DM_PAL.PAL_LINK_PREDICTION_PROC(PAL_LINK_PREDICTION_DATA_TBL,
#PAL_CONTROL_TBL, PAL_LINK_PREDICTION_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_LINK_PREDICTION_RESULT_TBL ORDER BY NODE1, NODE2;

```

Expected Result

12	NODE1	12	NODE2	12	SCORE
1		3		0.2857142857142857	
1		6		0.14285714285714285	
1		7		0.2857142857142857	
2		4		0.2857142857142857	
2		5		0.14285714285714285	
2		7		0.14285714285714285	
4		6		0.14285714285714285	
3		5		0.14285714285714285	
3		6		0.14285714285714285	
3		7		0.14285714285714285	
5		6		0.14285714285714285	

3.9 Miscellaneous

This section describes the ABC Analysis and Weighted Score Table algorithms that are provided by the Predictive Analysis Library.

3.9.1 ABC Analysis

This algorithm is used to classify objects (such as customers, employees, or products) based on a particular measure (such as revenue or profit). It suggests that inventories of an organization are not of equal value, thus can be grouped into three categories (A, B, and C) by their estimated importance. “A” items are very important for an organization. “B” items are of medium importance, that is, less important than “A” items and more important than “C” items. “C” items are of the least importance.

An example of ABC classification is as follows:

- “A” items – 20% of the items (customers) accounts for 70% of the revenue.
- “B” items – 30% of the items (customers) accounts for 20% of the revenue.
- “C” items – 50% of the items (customers) accounts for 10% of the revenue.

Prerequisites

- Input data cannot contain null value.
- The item names in the input table must be of varchar or nvarchar data type and be unique.

ABC

This function performs the ABC analysis algorithm.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'ABC', '<schema_name>',  
'<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<INPUT table type>	IN
2	<schema_name>	<PARAMETER table type>	IN
3	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<input table>, <parameter table>, <output  
table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Table

Table	Column	Column Data Type	Description
Data	1st column	Integer, bigint, varchar, or nvarchar	Item name
	2nd column	Double	Value

Parameter Table

Mandatory Parameter

The following parameter is mandatory and must be given a value.

Name	Data Type	Description
PERCENT_A	Double	Interval for A class
PERCENT_B	Double	Interval for B class
PERCENT_C	Double	Interval for C class

Optional Parameters

The following parameters are optional. If a parameter is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Varchar or nvarchar	ABC class
	2nd column	Integer, bigint, varchar, or nvarchar	Items

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_ABC_DATA_T;
CREATE TYPE PAL_ABC_DATA_T AS TABLE(
    "ITEM" VARCHAR(100),
    "VALUE" DOUBLE
);
DROP TYPE PAL_ABC_CONTROL_T;
CREATE TYPE PAL_ABC_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
DROP TYPE PAL_ABC_RESULT_T;
CREATE TYPE PAL_ABC_RESULT_T AS TABLE(
    "ABC" VARCHAR(10),
    "ITEM" VARCHAR(100)
);
DROP TABLE PAL_ABC_PDATA_TBL;
CREATE COLUMN TABLE PAL_ABC_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_ABC_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_ABC_DATA_T', 'IN');
INSERT INTO PAL_ABC_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_ABC_CONTROL_T', 'IN');
INSERT INTO PAL_ABC_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_ABC_RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_ABC_PROC');
CALL "SYS".AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'ABC', 'DM_PAL',
'PAL_ABC_PROC', PAL_ABC_PDATA_TBL);
DROP TABLE #PAL_ABC_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_ABC_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_ABC_CONTROL_TBL VALUES ('THREAD NUMBER',1,null,null);
INSERT INTO #PAL_ABC_CONTROL_TBL VALUES ('PERCENT_A',null,0.7,null);

```

```

INSERT INTO #PAL_ABC_CONTROL_TBL VALUES ('PERCENT_B',null,0.2,null);
INSERT INTO #PAL_ABC_CONTROL_TBL VALUES ('PERCENT_C',null,0.1,null);
DROP TABLE PAL_ABC_DATA_TBL;
CREATE COLUMN TABLE PAL_ABC_DATA_TBL LIKE PAL_ABC_DATA_T;
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item1', 15.4);
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item2', 200.4);
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item3', 280.4);
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item4', 100.9);
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item5', 40.4);
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item6', 25.6);
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item7', 18.4);
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item8', 10.5);
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item9', 96.15);
INSERT INTO PAL_ABC_DATA_TBL VALUES ('item10', 9.4);
DROP TABLE PAL_ABC_RESULT_TBL;
CREATE COLUMN TABLE PAL_ABC_RESULT_TBL LIKE PAL_ABC_RESULT_T;
CALL "DM_PAL".PAL_ABC_PROC(PAL_ABC_DATA_TBL, "#PAL_ABC_CONTROL_TBL",
PAL_ABC_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_ABC_RESULT_TBL;

```

Expected Result

PAL_ABC_RESULT_TBL:

ABC	ITEM
A	item3
A	item2
A	item4
B	item9
B	item5
B	item6
C	item7
C	item1
C	item8
C	item10

3.9.2 Weighted Score Table

A weighted score table is a method of evaluating alternatives when the importance of each criterion differs. In a weighted score table, each alternative is given a score for each criterion. These scores are then weighted by the importance of each criterion. All of an alternative's weighted scores are then added together to calculate its total weighted score. The alternative with the highest total score should be the best alternative.

You can use weighted score tables to make predictions about future customer behavior. You first create a model based on historical data in the data mining application, and then apply the model to new data to make the prediction. The prediction, that is, the output of the model, is called a score. You can create a single score for your customers by taking into account different dimensions.

A function defined by weighted score tables is a linear combination of functions of a variable.

$$f(x_1, \dots, x_n) = w_1 \times f_1(x_1) + \dots + w_n \times f_n(x_n)$$

Prerequisites

- The input data does not contain null value.
- The column of the Map Function table is sorted by the attribute order of the Input Data table.

WEIGHTEDTABLE

This function performs weighted table calculation. It is similar to the Volume Driver function in the Business Function Library (BFL). Volume Driver calculates only one column, but weightedTable calculates multiple columns at the same time.

Procedure Generation

```
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE ('AFLPAL', 'WEIGHTEDTABLE',
  '<schema_name>', '<procedure_name>', <signature_table>);
```

The signature table should contain the following records:

Position	Schema Name	Table Type Name	Parameter Type
1	<schema_name>	<Data INPUT table type>	IN
2	<schema_name>	<Map INPUT table type>	IN
3	<schema_name>	<Weights INPUT table type>	IN
4	<schema_name>	<PARAMETER table type>	IN
5	<schema_name>	<OUTPUT table type>	OUT

Procedure Calling

```
CALL <schema_name>.<procedure_name>(<data input table>, <map input table>,
  <weights input table>, <parameter table>, <output table>) with overview;
```

The procedure name is the same as specified in the procedure generation.

The input, parameter, and output tables must be of the types specified in the signature table.

Signature

Input Tables

Table	Column	Column Data Type	Description	Constraint
Target/ Input Data	Columns	Varchar, nvarchar, integer, or double	Specifies which will be used to calculate the scores	Discrete value: integer, varchar, nvarchar, double Continuous value: integer, double An ID column is mandatory. Its data type should be integer.
Map Function	Columns	Varchar, nvarchar, integer, or double	Creates the map function	Every attribute (except ID) in the Input Data table maps to two columns in the Map Function table: Key column and Value column. The Value column must be of double type.
Weights	Columns	Integer or double		This table has three columns. When the Input Data table has n attributes (except ID), the Weight Table will have n rows.

Parameter Table

Mandatory Parameters

None.

Optional Parameter

The following parameter is optional. If it is not specified, PAL will use its default value.

Name	Data Type	Default Value	Description
THREAD_NUMBER	Integer	1	Number of threads

Output Table

Table	Column	Column Data Type	Description
Result	1st column	Integer	ID
	2nd column	Double	Result value

Example

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DATA_T;
CREATE TYPE PAL_DATA_T AS TABLE(
    "ID" INTEGER,
    "GENDER" VARCHAR(10),
    "INCOME" INTEGER,
    "HEIGHT" DOUBLE
);
DROP TYPE PAL_MAP_FUN_T;
CREATE TYPE PAL_MAP_FUN_T AS TABLE(
    "GENDER" VARCHAR(10),
    "VAL1" DOUBLE,
    "INCOME" INTEGER,
    "VAL2" DOUBLE,
    "HEIGHT" DOUBLE,
    "VAL3" DOUBLE
);
DROP TYPE PAL PARA_T;
CREATE TYPE PAL PARA_T AS TABLE(
    "WEIGHT" DOUBLE,
    "ISDIS" INTEGER,
    "ROWNUM" INTEGER
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_RESULT_T;
CREATE TYPE PAL_RESULT_T AS TABLE(
    "ID" INTEGER,
    "RESULT" DOUBLE
);
-- create procedure
DROP TABLE PAL_WS_PDATA_TBL;
CREATE COLUMN TABLE PAL_WS_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_WS_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL DATA T', 'IN');
INSERT INTO PAL_WS_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_MAP_FUN_T', 'IN');
INSERT INTO PAL_WS_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL PARA_T', 'IN');
INSERT INTO PAL_WS_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_WS_PDATA_TBL VALUES (5, 'DM_PAL', 'PAL RESULT_T', 'OUT');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL WEIGHTEDTABLE PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'WEIGHTEDTABLE', 'DM_PAL',
'PAL WEIGHTEDTABLE PROC', PAL_WS_PDATA_TBL);
DROP TABLE PAL_DATA_TBL;
CREATE COLUMN TABLE PAL DATA_TBL LIKE PAL DATA_T;
INSERT INTO PAL_DATA_TBL VALUES (0,'male',5000,1.73);
INSERT INTO PAL_DATA_TBL VALUES (1,'male',9000,1.80);
INSERT INTO PAL_DATA_TBL VALUES (2,'female',6000,1.55);
INSERT INTO PAL_DATA_TBL VALUES (3,'male',15000,1.65);
INSERT INTO PAL_DATA_TBL VALUES (4,'female',2000,1.70);
INSERT INTO PAL_DATA_TBL VALUES (5,'female',12000,1.65);
INSERT INTO PAL_DATA_TBL VALUES (6,'male',1000,1.65);
INSERT INTO PAL_DATA_TBL VALUES (7,'male',8000,1.60);

```

```

INSERT INTO PAL_DATA_TBL VALUES (8,'female',5500,1.85);
INSERT INTO PAL_DATA_TBL VALUES (9,'female',9500,1.85);
DROP TABLE PAL_MAP_FUN_TBL;
CREATE COLUMN TABLE PAL_MAP_FUN_TBL LIKE PAL_MAP_FUN_T;
INSERT INTO PAL_MAP_FUN_TBL VALUES ('male',2.0, 0,0.0, 1.5,0.0);
INSERT INTO PAL_MAP_FUN_TBL VALUES ('female',1.5, 5500,1.0, 1.6,1.0);
INSERT INTO PAL_MAP_FUN_TBL VALUES (null,0.0, 9000,2.0, 1.71,2.0);
INSERT INTO PAL_MAP_FUN_TBL VALUES (null,0.0, 12000,3.0, 1.80,3.0);
DROP TABLE PAL PARA_TBL;
CREATE COLUMN TABLE PAL PARA_TBL LIKE PAL PARA_T;
INSERT INTO PAL PARA_TBL VALUES (0.5,1,2);
INSERT INTO PAL PARA_TBL VALUES (2.0,-1,4);
INSERT INTO PAL PARA_TBL VALUES (1.0,-1,4);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL (
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER',2,null,null);
DROP TABLE PAL RESULT_TBL;
CREATE COLUMN TABLE PAL RESULT_TBL LIKE PAL RESULT_T;
CALL "DM_PAL".PAL WEIGHTEDTABLE PROC(PAL DATA_TBL, PAL MAP FUN_TBL,
PAL PARA_TBL, #PAL CONTROL_TBL, PAL RESULT_TBL) with overview;
SELECT * FROM PAL RESULT_TBL;

```

Expected Result

PAL_WT_RESULT_TBL:

ID	result
0	3.0
1	8.0
2	2.75
3	8.0
4	1.75
5	7.75
6	2.0
7	4.0
8	5.75
9	7.75

4 End-to-End Scenarios

This section provides end-to-end scenarios of predictive analysis with PAL algorithms.

4.1 Scenario: Predict Segmentation of New Customers for a Supermarket

We wish to predict segmentation/clustering of new customers for a supermarket. First use the K-means function in PAL to perform segmentation/clustering for existing customers in the supermarket. The output can then be used as the training data for the C4.5 Decision Tree function to predict new customers' segmentation/clustering.

Technology Background

- K-means clustering is a method of cluster analysis whereby the algorithm partitions N observations or records into K clusters, in which each observation belongs to the cluster with the nearest center. It is one of the most commonly used algorithms in clustering method.
- Decision trees are powerful and popular tools for classification and prediction. Decision tree learning, used in statistics, data mining, and machine learning uses a decision tree as a predictive model which maps the observations about an item to the conclusions about the item's target value.

Implementation Steps

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Step 1

Input customer data and use the K-means function to partition the data set into K clusters. In this example, nine rows of data will be input. K equals 3, which means the customers will be partitioned into three levels.

```
SET SCHEMA DM_PAL;
DROP TYPE PAL_KMEANS_RESASSIGN_T;
CREATE TYPE PAL_KMEANS_RESASSIGN_T AS TABLE(
    "ID" INT,
    "CENTER_ASSIGN" INT,
```

```

        "DISTANCE" DOUBLE
    );
DROP TYPE PAL_KMEANS_DATA_T;
CREATE TYPE PAL_KMEANS_DATA_T AS TABLE(
    "ID" INT,
    "AGE" DOUBLE,
    "INCOME" DOUBLE,
    PRIMARY KEY("ID")
);
DROP TYPE PAL_KMEANS_CENTERS_T;
CREATE TYPE PAL_KMEANS_CENTERS_T AS TABLE(
    "CENTER_ID" INT,
    "V000" DOUBLE,
    "V001" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
-- create kmeans procedure
DROP TABLE PAL_KMEANS_PDATA_TBL;
CREATE COLUMN TABLE PAL_KMEANS_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_KMEANS_DATA_T', 'IN');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_KMEANS_RESASSIGN_T',
'OUT');
INSERT INTO PAL_KMEANS_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_KMEANS_CENTERS_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_KMEANS_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'KMEANS', 'DM_PAL',
'PAL_KMEANS_PROC', PAL_KMEANS_PDATA_TBL);
DROP TABLE PAL_KMEANS_DATA_TBL;
CREATE COLUMN TABLE PAL_KMEANS_DATA_TBL LIKE PAL_KMEANS_DATA_T;
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (0 , 20, 100000);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (1 , 21, 101000);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (2 , 22, 102000);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (3 , 30, 200000);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (4 , 31, 201000);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (5 , 32, 202000);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (6 , 40, 400000);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (7 , 41, 401000);
INSERT INTO PAL_KMEANS_DATA_TBL VALUES (8 , 42, 402000);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('GROUP_NUMBER', 3, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('INIT_TYPE', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('DISTANCE_LEVEL', 2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('MAX_ITERATION', 100, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('EXIT_THRESHOLD', NULL, 0.000001, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('NORMALIZATION', 0, NULL, NULL);
--clean kmeans result
DROP TABLE PAL_KMEANS_RESASSIGN_TBL;
CREATE COLUMN TABLE PAL_KMEANS_RESASSIGN_TBL LIKE PAL_KMEANS_RESASSIGN_T;
DROP TABLE PAL_KMEANS_CENTERS_TBL;

```

```

CREATE COLUMN TABLE PAL_KMEANS_CENTERS_TBL LIKE PAL_KMEANS_CENTERS_T;
CALL DM_PAL.PAL_KMEANS_PROC(PAL_KMEANS_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_KMEANS_RESASSIGN_TBL, PAL_KMEANS_CENTERS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_KMEANS_CENTERS_TBL;
SELECT * FROM PAL_KMEANS_RESASSIGN_TBL;
DROP TABLE PAL_KMEANS_RESULT_TBL;
CREATE COLUMN TABLE PAL_KMEANS_RESULT_TBL(
    "AGE" DOUBLE,
    "INCOME" DOUBLE,
    "LEVEL" INT
);
TRUNCATE TABLE PAL_KMEANS_RESULT_TBL;
INSERT INTO PAL_KMEANS_RESULT_TBL(
    SELECT PAL_KMEANS_DATA_TBL.AGE, PAL_KMEANS_DATA_TBL.INCOME,
PAL_KMEANS_RESASSIGN_TBL.CENTER_ASSIGN
    FROM PAL_KMEANS_RESASSIGN_TBL
    INNER JOIN PAL_KMEANS_DATA_TBL
        ON PAL_KMEANS_RESASSIGN_TBL.ID = PAL_KMEANS_DATA_TBL.ID
);
SELECT * FROM PAL_KMEANS_RESULT_TBL;

```

The result should show the following in PAL_KMEANS_RESULT_TBL:

	AGE	INCOME	LEVEL
1	20	100,000	0
2	21	101,000	0
3	22	102,000	0
4	30	200,000	1
5	31	201,000	1
6	32	202,000	1
7	40	400,000	2
8	41	401,000	2
9	42	402,000	2

Step 2

Use the above output as the training data of C4.5 Decision Tree. The C4.5 Decision Tree function will generate a tree model which maps the observations about an item to the conclusions about the item's target value.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_CDT_DATA_T;
CREATE TYPE PAL_CDT_DATA_T AS TABLE(
    "AGE" DOUBLE,
    "INCOME" DOUBLE,
    "LEVEL" INT
);
DROP TYPE PAL_CDT_JSONMODEL_T;
CREATE TYPE PAL_CDT_JSONMODEL_T AS TABLE(
    "ID" INT,
    "JSONMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CDT_PMMODEL_T;
CREATE TYPE PAL_CDT_PMMODEL_T AS TABLE(
    "ID" INT,
    "PMMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,

```

```

    "STRINGARGS" VARCHAR(100)
);
--create procedure
DROP TABLE PAL_CDT_PDATA_TBL;
CREATE COLUMN TABLE PAL_CDT_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_CDT_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_CDT_DATA_T', 'IN');
INSERT INTO PAL_CDT_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_CDT_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CDT_JSONMODEL_T', 'OUT');
INSERT INTO PAL_CDT_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_CDT_PMMODEL_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_CREATEDT_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'CREATEDT', 'DM_PAL',
'PAL_CREATEDT_PROC', PAL_CDT_PDATA_TBL);
DROP TABLE PAL_CDT_TRAINING_TBL;
CREATE COLUMN TABLE PAL_CDT_TRAINING_TBL(
    "REGION" VARCHAR(50),
    "SALESPERIOD" VARCHAR(50),
    "REVENUE" Double,
    "CLASSLABEL" VARCHAR(50)
);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PERCENTAGE', NULL, 1.0, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('IS_SPLIT_MODEL', 1, NULL, NULL);
INSERT INTO #PAL_CONTROL_TBL VALUES ('PMML_EXPORT', 2, NULL, NULL);
DROP TABLE PAL_CDT_JSONMODEL_TBL;
CREATE COLUMN TABLE PAL_CDT_JSONMODEL_TBL LIKE PAL_CDT_JSONMODEL_T;
DROP TABLE PAL_CDT_PMMODEL_TBL;
CREATE COLUMN TABLE PAL_CDT_PMMODEL_TBL LIKE PAL_CDT_PMMODEL_T;
CALL DM_PAL.PAL_CREATEDT_PROC(PAL_KMEANS_RESULT_TBL, "#PAL_CONTROL_TBL",
PAL_CDT_JSONMODEL_TBL, PAL_CDT_PMMODEL_TBL) WITH OVERVIEW;
SELECT * FROM PAL_CDT_JSONMODEL_TBL;
SELECT * FROM PAL_CDT_PMMODEL_TBL;

```

Step 3

Use the above tree model to map each new customer to the corresponding level he or she belongs to.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_PCDT_DATA_T;
CREATE TYPE PAL_PCDT_DATA_T AS TABLE(
    "ID" INT,
    "AGE" DOUBLE,
    "INCOME" DOUBLE
);
DROP TYPE PAL_PCDT_JSONMODEL_T;
CREATE TYPE PAL_PCDT_JSONMODEL_T AS TABLE(
    "ID" INT,
    "JSONMODEL" VARCHAR(5000)
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
    "NAME" VARCHAR(100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR(100)
);

```

```

DROP TYPE PAL_PCDT_RESULT_T;
CREATE TYPE PAL_PCDT_RESULT_T AS TABLE(
    "ID" INT,
    "CLASSLABEL" VARCHAR(50)
);
-- create procedure
DROP TABLE PAL_PCDT_PDATA_TBL;
CREATE COLUMN TABLE PAL_PCDT_PDATA_TBL(
    "POSITION" INT,
    "SCHEMA_NAME" NVARCHAR(256),
    "TYPE_NAME" NVARCHAR(256),
    "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_PCDT_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_PCDT_DATA_T', 'IN');
INSERT INTO PAL_PCDT_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_PCDT_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_PCDT_JSONMODEL_T',
'IN');
INSERT INTO PAL_PCDT_PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_PCDT_RESULT_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_PREDICTWITHDT_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'PREDICTWITHDT', 'DM_PAL',
'PAL_PREDICTWITHDT_PROC', 'PAL_PCDT_PDATA_TBL');
DROP TABLE PAL_PCDT_DATA_TBL;
CREATE COLUMN TABLE PAL_PCDT_DATA_TBL LIKE PAL_PCDT_DATA_T;
INSERT INTO PAL_PCDT_DATA_TBL VALUES (10, 20, 100003);
INSERT INTO PAL_PCDT_DATA_TBL VALUES (11, 30, 200003);
INSERT INTO PAL_PCDT_DATA_TBL VALUES (12, 40, 400003);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
    "NAME" VARCHAR (100),
    "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE,
    "STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('THREAD_NUMBER', 2, NULL, NULL);
DROP TABLE PAL_PCDT_RESULT_TBL;
CREATE COLUMN TABLE PAL_PCDT_RESULT_TBL LIKE PAL_PCDT_RESULT_T;
CALL DM_PAL.PAL_PREDICTWITHDT_PROC(PAL_PCDT_DATA_TBL, "#PAL_CONTROL_TBL",
PAL_PCDT_JSONMODEL_TBL, PAL_PCDT_RESULT_TBL) WITH OVERVIEW;
SELECT * FROM PAL_PCDT_RESULT_TBL;

```

The expected prediction result is as follows:

	ID	CLASSLABEL
1	10	1
2	11	2
3	12	0

4.2 Scenario: Analyze the Cash Flow of an Investment on a New Product

We wish to do an analysis of the cash flow of an investment required to create a new product. Projected estimates are given for the product revenue, product costs, overheads, and capital investment for each year of the analysis, from which the cash flow can be calculated. For capital investment appraisal the cash flows are summed for each year and discounted for future values, in other words the net present value of the cash flow is derived as a single value measuring the benefit of the investment.

The projected estimates are single point estimates of each data point and the analysis provides a single point value of project net present value (NPV). This is referred to as deterministic modeling, which is in contrast to probabilistic modeling whereby we examine the probability of outcomes, for example, what is the probability of a NPV greater than zero. Probabilistic modeling is also called Monte Carlo Simulation.

Monte Carlo Simulation is used in our example to estimate the net present value (NPV) of the investment. The equations used in the simulation are:

For each year $i=0, 1, \dots, k$

```
Product margin(i) = product revenue(i) - product cost(i)
```

```
Total profit(i) = product margin(i) - overhead(i)
```

```
Cash flow(i) = total profit(i) - capital investment(i)
```

Suppose the simulation covers k years' time periods and the discount rate is r , the net present value of the investment is defined as:

$$\text{NPV of investment} = \sum_{i=0}^k \frac{\text{Cash flow}_i}{(1+r)^i} = \sum_{i=0}^k \frac{\text{product revenue}_i - \text{product cost}_i - \text{overhead}_i - \text{capital flow}_i}{(1+r)^i}$$

Technology Background

Monte Carlo Simulation is a computational algorithm that repeatedly generates random samples to compute numerical results based on a formula or model in order to obtain the unknown probability distribution of an event or outcome.

In PAL, the Random Distribution Sampling, Distribution Fitting, and Cumulative Distribution algorithms may be used for Monte Carlo Simulation.

Implementation Steps

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Step 1

Input the given estimates (single point deterministic values) for product revenue, product costs, overheads, and capital investment. In this example, the time periods are 5 (from year 1 to year 5).

The probability distribution for each variable is assumed as follows:

- Product Revenue:
Normal distribution and the mean and standard deviation are listed in the following table.
- Product Costs:
Normal distribution and the mean and standard deviation are listed in the following table.

- Overheads:
Uniform distribution and min and max values are listed in the following table.
- Capital Investment (for year 1 and year 2)
Gamma distribution and shape and scale values are listed in the following table.

	Year 1	Year 2	Year 3	Year 4	Year 5
Product Revenue					
Mean	£0	£3,000	£8,000	£18,000	£30,000
Standard Deviation	0	300	800	1800	3000
Product Costs					
Mean	£1,000	£1,000	£2,500	£7,000	£10,000
Standard Deviation	75	75	187.5	525	750
Overheads					
Min	£1,400	£1,800	£2,200	£2,600	£3,000
Max	£1,500	£2,200	£2,800	£3,400	£4,000
Capital Investment					
Mean	£10,000	£2,000			
Standard Deviation	500	100			

Run the Random Distribution Sampling algorithm for each variable and generate 1,000 sample sets. The number of sample sets is a choice for the analysis. The larger the value then the more smooth the output distribution and the closer it will be to a normal distribution.

```

SET SCHEMA DM_PAL;
-----
---Random sampling process---
-----
DROP TYPE PAL_DISTRRANDOM_DISTRPARAM_T;
CREATE TYPE PAL_DISTRRANDOM_DISTRPARAM_T AS TABLE( NAME VARCHAR(50), VAL
VARCHAR(50));
DROP TYPE PAL_DISTRRANDOM_RESULT_T;
CREATE TYPE PAL_DISTRRANDOM_RESULT_T AS TABLE(ID INTEGER, RANDOM DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(NAME VARCHAR (50), INTARGS INTEGER,
DOUBLEARGS DOUBLE, STRINGARGS VARCHAR (100));
DROP TYPE PAL_CASHFLOW_T;
CREATE TYPE PAL_CASHFLOW_T AS TABLE(ID INTEGER, CASH DOUBLE);
DROP TABLE PDATA;
CREATE COLUMN TABLE PDATA("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PDATA VALUES (1, 'DM_PAL','PAL_DISTRRANDOM_DISTRPARAM_T', 'IN');
INSERT INTO PDATA VALUES (2, 'DM_PAL','PAL_CONTROL_T', 'IN');

```

```

INSERT INTO PDATA VALUES (3, 'DM_PAL', 'PAL_DISTRRANDOM_RESULT_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_DISTRRANDOM');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'DISTRRANDOM',
'DM_PAL', 'PAL_DISTRRANDOM', PDATA);
-----
-----YEAR 1 -----
----Product revenue-----
DROP TABLE PAL_DISTRRANDOM_DISTRPARAM_TAB;
CREATE COLUMN TABLE PAL_DISTRRANDOM_DISTRPARAM_TAB LIKE
PAL_DISTRRANDOM_DISTRPARAM_T;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '0');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '0.0001');
DROP TABLE PAL_CONTROL_TAB;
CREATE COLUMN TABLE PAL_CONTROL_TAB LIKE PAL_CONTROL_T;
INSERT INTO PAL_CONTROL_TAB VALUES ('NUM_RANDOM', 5000, null, null);
INSERT INTO PAL_CONTROL_TAB VALUES ('SEED', 0, null, null);
INSERT INTO PAL_CONTROL_TAB VALUES ('THREAD NUMBER', 8, null, null);
DROP TABLE PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
CREATE COLUMN TABLE PAL_DISTRRANDOM_RESULT_TAB_REVENUE LIKE
PAL_DISTRRANDOM_RESULT_T;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
-----Product Costs-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '1000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '5625');
DROP TABLE PAL_DISTRRANDOM_RESULT_TAB_COSTS;
CREATE COLUMN TABLE PAL_DISTRRANDOM_RESULT_TAB_COSTS LIKE
PAL_DISTRRANDOM_RESULT_T;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_COSTS) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_COSTS;
-----OVERHEADS-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME',
'UNIFORM');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MIN', '1400');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MAX', '1500');
DROP TABLE PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;
CREATE COLUMN TABLE PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS LIKE
PAL_DISTRRANDOM_RESULT_T;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS) with overview;
-- SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;
-----INVESTMENT-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '10000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '250000');
DROP TABLE PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT;
CREATE COLUMN TABLE PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT LIKE
PAL_DISTRRANDOM_RESULT_T;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT;
-----calculate cash flow -----
DROP TABLE PAL_CASHFLOW_YEAR1;
CREATE COLUMN TABLE PAL_CASHFLOW_YEAR1(ID INTEGER, CASH DOUBLE);
INSERT INTO PAL_CASHFLOW_YEAR1 SELECT PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.RANDOM -
PAL_DISTRRANDOM_RESULT_TAB_COSTS.RANDOM -
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.RANDOM -
PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT.RANDOM
FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE left
join PAL_DISTRRANDOM_RESULT_TAB_COSTS on

```

```

PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_COSTS.ID left
join
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.ID
left join
PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT.ID;
--SELECT * from PAL_CASHFLOW_YEAR1;
-----
-----YEAR 2 -----
--product revenue---
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '3000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '90000');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
----Product Costs-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '1000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '5625');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_COSTS;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_COSTS) with overview;
SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_COSTS;
----OVERHEADS-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME',
'UNIFORM');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MIN', '1800');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MAX', '2200');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS) with overview;
SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;
----INVESTMENT-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '2000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '10000');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT;
-- calculate cash flow ---
DROP TABLE PAL_CASHFLOW_YEAR2;
CREATE COLUMN TABLE PAL_CASHFLOW_YEAR2(ID INTEGER,CASH DOUBLE);
INSERT INTO PAL_CASHFLOW_YEAR2 SELECT PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.RANDOM
- PAL_DISTRRANDOM_RESULT_TAB_COSTS.RANDOM -
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.RANDOM -
PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT.RANDOM
FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE left
join PAL_DISTRRANDOM_RESULT_TAB_COSTS on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_COSTS.ID left
join
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.ID
left join
PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_INVESTMENT.ID;
--SELECT * from PAL_CASHFLOW_YEAR2;
-----
-----YEAR 3 -----
--product revenue---

```

```

DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '8000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '640000');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
----Product Costs-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '2500');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '35156.25');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_COSTS;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_COSTS) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_COSTS;
----OVERHEADS-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME',
'UNIFORM');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MIN', '2200');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MAX', '2800');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;
-- calculate cash flow ---
DROP TABLE PAL_CASHFLOW_YEAR3;
CREATE COLUMN TABLE PAL_CASHFLOW_YEAR3(ID INTEGER,CASH DOUBLE);
INSERT INTO PAL_CASHFLOW_YEAR3 SELECT PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.RANDOM
- PAL_DISTRRANDOM_RESULT_TAB_COSTS.RANDOM -
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.RANDOM
FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE left
join PAL_DISTRRANDOM_RESULT_TAB_COSTS on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_COSTS.ID left
join
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.ID;
--SELECT * from PAL_CASHFLOW_YEAR3;
-----
-----YEAR 4 -----
--Product revenue-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '18000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '3240000');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
----Product Costs-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '7000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '275625');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_COSTS;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_COSTS) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_COSTS;
----OVERHEADS-----
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME',
'UNIFORM');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MIN', '2600');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MAX', '3400');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;

```

```

CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;
-- calculate cash flow ---
DROP TABLE PAL_CASHFLOW_YEAR4;
CREATE COLUMN TABLE PAL_CASHFLOW_YEAR4(ID INTEGER,CASH DOUBLE);
INSERT INTO PAL_CASHFLOW_YEAR4 SELECT PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.RANDOM
- PAL_DISTRRANDOM_RESULT_TAB_COSTS.RANDOM -
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.RANDOM
FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE left
join PAL_DISTRRANDOM_RESULT_TAB_COSTS on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_COSTS.ID left
join
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.ID;
--SELECT * from PAL_CASHFLOW_YEAR4;
-----
-----YEAR 5 -----
--Product revenue---
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '30000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '9000000');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE;
---Product Costs---
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME', 'NORMAL');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MEAN', '10000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('VARIANCE', '562500');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_COSTS;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_COSTS) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_COSTS;
---OVERHEADS---
DELETE FROM PAL_DISTRRANDOM_DISTRPARAM_TAB;
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('DISTRIBUTIONNAME',
'UNIFORM');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MIN', '3000');
INSERT INTO PAL_DISTRRANDOM_DISTRPARAM_TAB VALUES ('MAX', '4000');
DELETE FROM PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;
CALL DM_PAL.PAL_DISTRRANDOM(PAL_DISTRRANDOM_DISTRPARAM_TAB, PAL_CONTROL_TAB,
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS) with overview;
--SELECT * FROM PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS;
-- calculate cash flow ---
DROP TABLE PAL_CASHFLOW_YEAR5;
CREATE COLUMN TABLE PAL_CASHFLOW_YEAR5(ID INTEGER,CASH DOUBLE);
INSERT INTO PAL_CASHFLOW_YEAR5 SELECT PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID,
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.RANDOM
- PAL_DISTRRANDOM_RESULT_TAB_COSTS.RANDOM -
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.RANDOM
FROM PAL_DISTRRANDOM_RESULT_TAB_REVENUE left
join PAL_DISTRRANDOM_RESULT_TAB_COSTS on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_COSTS.ID left
join
PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS on
PAL_DISTRRANDOM_RESULT_TAB_REVENUE.ID=PAL_DISTRRANDOM_RESULT_TAB_OVERHEADS.ID;
--SELECT * from PAL_CASHFLOW_YEAR5;

```

Step 2

Calculate the net present value of the investment by the following equation for each sampling.

$$\text{NPV of investment} = \sum_{i=0}^k \frac{\text{Cash flow}_i}{(1+r)^i} = \sum_{i=0}^k \frac{\text{product revenue}_i - \text{product cost}_i - \text{overhead}_i - \text{capital flow}_i}{(1+r)^i}$$

```
---- calculate net present value of investment ----
DROP TABLE NPV;
CREATE COLUMN TABLE NPV(NPVALUE DOUBLE);
INSERT INTO NPV SELECT PAL_CASHFLOW_YEAR1.CASH +
PAL_CASHFLOW_YEAR2.CASH/1.05 +
PAL_CASHFLOW_YEAR3.CASH/POWER(1.05,2) +
PAL_CASHFLOW_YEAR4.CASH/POWER(1.05,3) +
PAL_CASHFLOW_YEAR5.CASH/POWER(1.05,4)
FROM PAL_CASHFLOW_YEAR1 left join
PAL_CASHFLOW_YEAR2 on PAL_CASHFLOW_YEAR1.ID =
PAL_CASHFLOW_YEAR2.ID
left join PAL_CASHFLOW_YEAR3 on PAL_CASHFLOW_YEAR1.ID
= PAL_CASHFLOW_YEAR3.ID
left join PAL_CASHFLOW_YEAR4 on PAL_CASHFLOW_YEAR1.ID
= PAL_CASHFLOW_YEAR4.ID
left join PAL_CASHFLOW_YEAR5 on PAL_CASHFLOW_YEAR1.ID
= PAL_CASHFLOW_YEAR5.ID;
SELECT * FROM NPV;
```

The expected result is as follows:

	NPVALUE
1	25,850,733484770433
2	15,592,640682833186
3	17,432,805376344313
4	17,084,511392504068
5	11,315,65449460255

Step 3

Plot the distribution of the net present value of the investment and run Distribution Fitting to fit a normal distribution to the NPV of the investment as. (The Central Limit theorem states that the output distribution will be a normal distribution.)

```
----- distribution fit process -----
-----
DROP TYPE PAL_DISTRFIT_DATA_T;
CREATE TYPE PAL_DISTRFIT_DATA_T AS TABLE(NPVALUE DOUBLE);
DROP TYPE PAL_DISTRFIT_ESTIMATION_T;
CREATE TYPE PAL_DISTRFIT_ESTIMATION_T AS TABLE(NAME VARCHAR(50),VAL VARCHAR(50));
DROP TYPE PAL_DISTRFIT_STATISTICS_T;
CREATE TYPE PAL_DISTRFIT_STATISTICS_T AS TABLE(NAME VARCHAR(50),VAL DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(NAME VARCHAR (50),INTARGS INTEGER,DOUBLEARGS
DOUBLE,STRINGARGS VARCHAR (100));
DROP TABLE PDATA_TBL;
CREATE COLUMN TABLE PDATA_TBL("POSITION" INT, "SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PDATA_TBL VALUES (1, 'DM_PAL','PAL_DISTRFIT_DATA_T', 'IN');
INSERT INTO PDATA_TBL VALUES (2, 'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PDATA_TBL VALUES (3, 'DM_PAL','PAL_DISTRFIT_ESTIMATION_T', 'OUT');
INSERT INTO PDATA_TBL VALUES (4, 'DM_PAL','PAL_DISTRFIT_STATISTICS_T', 'OUT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DISTRFIT');
call SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'DISTRFIT',
'DM_PAL','PAL_DISTRFIT', PDATA_TBL);
DROP TABLE PAL_CONTROL_TBL;
```

```

CREATE COLUMN TABLE PAL_CONTROL_TBL(NAME VARCHAR (50),INTARGS INTEGER,DOUBLEARGS
DOUBLE,STRINGARGS VARCHAR (100));
INSERT INTO PAL_CONTROL_TBL VALUES ('DISTRIBUTIONNAME', null, null, 'NORMAL');
INSERT INTO PAL_CONTROL_TBL VALUES ('OPTIMAL_METHOD', 0, null, null);
DROP TABLE PAL_DISTRFIT_ESTIMATION_TBL;
CREATE COLUMN TABLE PAL_DISTRFIT_ESTIMATION_TBL(NAME VARCHAR(50),VAL
VARCHAR(50));
DROP TABLE PAL_DISTRFIT_STATISTICS_TBL;
CREATE COLUMN TABLE PAL_DISTRFIT_STATISTICS_TBL(NAME VARCHAR(50),VAL DOUBLE);
CALL DM_PAL.PAL_DISTRFIT(NPV, PAL_CONTROL_TBL, PAL_DISTRFIT_ESTIMATION_TBL,
PAL_DISTRFIT_STATISTICS_TBL) with overview;
SELECT * FROM PAL_DISTRFIT_ESTIMATION_TBL;

```

The expected result is as follows:

	NAME	VAL
1	DISTRIBUTIONNAME	NORMAL
2	MEAN	14472
3	VARIANCE	4696.97

Step 4

According to the fitted model, run the Cumulative Distribution function to obtain the probability of having an NPV of investment smaller than or equal to a given NPV of the investment.

```

-----
----distribution probability process --
-----
DROP TYPE PAL_DISTRPROB_DATA_T;
CREATE TYPE PAL_DISTRPROB_DATA_T AS TABLE(DATACOL DOUBLE);
DROP TYPE PAL_DISTRPROB_DISTRPARAM_T;
CREATE TYPE PAL_DISTRPROB_DISTRPARAM_T AS TABLE(NAME VARCHAR(50),VALUEEE
VARCHAR(50));
DROP TYPE PAL_DISTRPROB_RESULT_T;
CREATE TYPE PAL_DISTRPROB_RESULT_T AS TABLE(INPUTDATA DOUBLE,PROBABILITY DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(NAME VARCHAR (50),INTARGS INTEGER,DOUBLEARGS
DOUBLE,STRINGARGS VARCHAR (100));
DROP TABLE PAL_DISTRPROB_PDATA_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_PDATA_TBL("POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (1, 'DM_PAL','PAL_DISTRPROB_DATA_T',
'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_DISTRPROB_DISTRPARAM_T', 'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (3, 'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (4,
'DM_PAL','PAL_DISTRPROB_RESULT_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DISTRPROB_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL', 'DISTRPROB',
'DM_PAL','PAL_DISTRPROB_PROC', PAL_DISTRPROB_PDATA_TBL);
DROP TABLE PAL_DISTRPROB_DATA_TBL;
CREATE TABLE PAL_DISTRPROB_DATA_TBL LIKE PAL_DISTRPROB_DATA_T;
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (7000);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (8000);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (9000);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (10000);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (11000);
DROP TABLE PAL_DISTRPROB_DISTRPARAM_TBL;
CREATE TABLE PAL_DISTRPROB_DISTRPARAM_TBL LIKE PAL_DISTRPROB_DISTRPARAM_T;
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('DISTRIBUTIONNAME', 'Normal');
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('MEAN', '100');
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('VARIANCE', '1');

```

```

UPDATE PAL_DISTRPROB_DISTRPARAM_TBL SET VALUEE = (SELECT VAL FROM
PAL_DISTRFIT_ESTIMATION_TBL WHERE NAME = 'MEAN') WHERE NAME = 'MEAN';
UPDATE PAL_DISTRPROB_DISTRPARAM_TBL SET VALUEE = (SELECT VAL*VAL FROM
PAL_DISTRFIT_ESTIMATION_TBL WHERE NAME = 'SD') WHERE NAME = 'VARIANCE';
SELECT * FROM PAL_DISTRPROB_DISTRPARAM_TBL;
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(NAME VARCHAR (50),INTARGS
INTEGER,DOUBLEARGS DOUBLE,STRINGARGS VARCHAR (100));
INSERT INTO #PAL_CONTROL_TBL VALUES ('LOWER_UPPER',0,null,null);
DROP TABLE PAL_DISTRPROB_RESULT_TBL;
CREATE TABLE PAL_DISTRPROB_RESULT_TBL LIKE PAL_DISTRPROB_RESULT_T;
CALL DM_PAL.PAL_DISTRPROB PROC(PAL_DISTRPROB_DATA_TBL,
PAL_DISTRPROB_DISTRPARAM_TBL, #PAL_CONTROL_TBL, PAL_DISTRPROB_RESULT_TBL) WITH
OVERVIEW;
SELECT * FROM PAL_DISTRPROB_RESULT_TBL;

```

The expected result is as follows:

	INPUTDATA	PROBABILITY
1	14,500	0.6585663437995875
2	14,600	0.9690963012741993
3	14,400	0.14672886476982...

4.3 Scenario: Survival Analysis

In clinical trials or community trials, the effect of an intervention is assessed by measuring the number of subjects who have survived or are saved after that intervention over a period of time. We wish to measure the survival probability of Dukes'C colorectal cancer patients after treatment and evaluate statistically whether the patients who accept treatment can survive longer than those who are only controlled conservatively.

Option 1: Kaplan-Meier Estimate

Technology Background

Kaplan-Meier estimate is one of the simplest way to measure the fraction of subjects living for a certain amount of time after treatment. The time starting from a defined point to the occurrence of a given event, for example death, is called as survival time.

This scenarios describes a clinical trial of 49 patients for the treatment of Dukes'C colorectal cancer. The following data shows the survival time in 49 patients with Dukes'C colorectal cancer who are randomly assigned to either linoleic acid or control treatment.

Treatment	Survival Time (months)
Linoleic acid (n = 25)	1+, 5+, 6, 6, 9+, 10, 10, 10+, 12, 12, 12, 12+, 13+, 15+, 16+, 20+, 24, 24+, 27+, 32, 34+, 36+, 36+, 44+

Treatment	Survival Time (months)
Control (n = 24)	3+, 6, 6, 6, 6, 8, 8, 12, 12, 12+, 15+, 16+, 18+, 18+, 20, 22+, 24, 28+, 28+, 28+, 30, 30+, 33+, 42

The + sign indicates censored data. Until 6 months after treatment, there are no deaths. The effect of the censoring is to remove from the alive group those that are censored. At time 6 months two subjects have been censored so the number alive just before 6 months is 23. There are two deaths at 6 months. Thus,

$$\hat{S}(6) = 1 * (23 - 2) / 23 = 0.9130$$

We now reduce the number alive ("at risk") by two. The censored event at 9 months reduces the "at risk" set to 20. At 10 months there are two deaths. So the proportion surviving is $18/20 = 0.9$, and the cumulative proportion surviving is $0.913 * 0.90 = 0.8217$.

The data can then be loaded into table as follows:

TIME	STATUS	OCCURENCES	GROUP
1	0	1	linoleic acid
5	0	1	linoleic acid
6	1	1	linoleic acid
6	1	1	linoleic acid
9	0	1	linoleic acid
10	1	1	linoleic acid
10	1	1	linoleic acid
10	0	1	linoleic acid
12	1	4	linoleic acid
12	0	1	linoleic acid
13	0	1	linoleic acid
15	0	1	linoleic acid
16	0	1	linoleic acid
20	0	1	linoleic acid
24	1	1	linoleic acid
24	0	1	linoleic acid
27	0	1	linoleic acid
32	1	1	linoleic acid
34	0	1	linoleic acid
36	0	2	linoleic acid
44	0	1	linoleic acid
3	0	1	control
6	1	4	control
8	1	2	control
12	1	2	control
12	0	1	control

Then we get the survival estimates as follows:

GROUP	TIME	RISKNUM	EVENTSNUM	PROB	STDERR
control	6	23	4	0.8260869565217391	0.0790341964475118
control	8	19	2	0.7391304347826088	0.09156053715170008
control	12	17	2	0.6521739130434783	0.09931134621220186
control	20	10	1	0.5869565217391305	0.10870510355884236
control	24	8	1	0.5135869565217391	0.11729212936202411
control	30	4	1	0.3851902173913043	0.14178453369559724
control	42	1	1	0	?
linoleic acid	6	23	2	0.9130434782608696	0.058753384755841...
linoleic acid	10	20	2	0.8217391304347826	0.08091665803648351
linoleic acid	12	17	4	0.628388746803069	0.10476559644019677
linoleic acid	24	8	1	0.5498401534526853	0.11748198297014116
linoleic acid	32	5	1	0.43987212276214...	0.13604287804080634

To compare survival estimates produced from two groups, we use log-rank test. It is a hypothesis test to compare the survival distribution of two groups (some of the observations may be censored) and is used to test the null hypothesis that there is no difference between the populations (treatment group and control group) in the probability of an event (here a death) at any time point. The methods are nonparametric in that they do not make assumptions about the distributions of survival estimates. The analysis is based on the times of events (here deaths). For each such time we calculate the observed number of deaths in each group and the number expected if there were in reality no difference between the groups. It is widely used in clinical trials to establish the efficacy of a new treatment in comparison with a control treatment when the measurement is the time to event (such as the time from initial treatment to death).

Because the log-rank test is purely a test of significance, it cannot provide an estimate of the size of the difference between the groups.

GROUP	TOTALRISK	OBSERVED	EXPECTED	LOGRANKSTAT
control	24	12	10.714679152837048	0.328949386383633
linoleic acid	25	10	11.285320847162952	0.328949386383633

STATISTICS	VALUE
chiSqr	0.3289493863836331
df	1
p-value	0.5662783832720855

Implementation Step

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Input customer data and use the Kaplan-Meier function to get the survival estimates and log-rank test statistics.

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_TRIAL_DATA_T;
CREATE TYPE PAL_TRIAL_DATA_T AS TABLE ("TIME" INTEGER,
    "STATUS" INTEGER,
    "OCCURENCES" INTEGER,
    "GROUP" VARCHAR(50));

DROP TYPE PAL_TRIAL_ESTIMATES_T;
CREATE TYPE PAL_TRIAL_ESTIMATES_T AS TABLE("GROUP" VARCHAR(50), "TIME" INTEGER,
    "RISKNUM" INTEGER, "EVENTSNUM" INTEGER,
    "PROB" DOUBLE, "STDERR" DOUBLE,
    "LOWERLIMIT" DOUBLE, "UPPERLIMIT" DOUBLE);
DROP TYPE PAL_TRIAL_LOGRANK_STAT1_T;
CREATE TYPE PAL_TRIAL_LOGRANK_STAT1_T AS TABLE("GROUP" VARCHAR(50), "TOTALRISK"
    INTEGER, "OBSERVED" INTEGER, "EXPECTED" DOUBLE, "LOGRANKSTAT" DOUBLE);
DROP TYPE PAL_TRIAL_LOGRANK_STAT2_T;
CREATE TYPE PAL_TRIAL_LOGRANK_STAT2_T AS TABLE("STATISTICS" VARCHAR(20), "VALUE"
    DOUBLE);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE( "NAME" VARCHAR(100), "INTARGS" INTEGER,
    "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR(100));
DROP TABLE PAL_TRIAL_PDATA_TBL;
CREATE COLUMN TABLE PAL_TRIAL_PDATA_TBL( "POSITION" INT, "SCHEMA_NAME"
NVARCHAR(256), "TYPE_NAME" NVARCHAR(256), "PARAMETER_TYPE" VARCHAR(7));
INSERT INTO PAL_TRIAL_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_TRIAL_DATA_T', 'in');
INSERT INTO PAL_TRIAL_PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'in');
INSERT INTO PAL_TRIAL_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_TRIAL_ESTIMATES_T',
    'out');
INSERT INTO PAL_TRIAL_PDATA_TBL VALUES (4, 'DM_PAL',
    'PAL_TRIAL_LOGRANK_STAT1_T', 'out');
INSERT INTO PAL_TRIAL_PDATA_TBL VALUES (5, 'DM_PAL',
    'PAL_TRIAL_LOGRANK_STAT2_T', 'out');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE DROP('DM_PAL', 'PAL_KMSURV_PROC');
CALL "SYS".AFLLANG_WRAPPER PROCEDURE CREATE('AFLPAL', 'KMSURV',
    'DM_PAL', 'PAL_KMSURV_PROC', PAL_TRIAL_PDATA_TBL);
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL( "NAME" VARCHAR (100),
    "INTARGS" INTEGER, "DOUBLEARGS" DOUBLE, "STRINGARGS" VARCHAR (100));
DROP TABLE PAL_TRIAL_DATA_TBL;
CREATE COLUMN TABLE PAL_TRIAL_DATA_TBL LIKE PAL_TRIAL_DATA_T;
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(1, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(5, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(6, 1, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(6, 1, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(9, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(10, 1, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(10, 1, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(10, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(12, 1, 4, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(12, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(13, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(15, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(16, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(20, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(24, 1, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(24, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(27, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(32, 1, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(34, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(36, 0, 2, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(44, 0, 1, 'linoleic acid');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(3, 0, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(6, 1, 4, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(8, 1, 2, 'control');

```

```

INSERT INTO PAL_TRIAL_DATA_TBL VALUES(12, 1, 2, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(12, 0, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(15, 0, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(16, 0, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(18, 0, 2, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(20, 1, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(22, 0, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(24, 1, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(28, 0, 3, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(30, 0, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(30, 1, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(33, 0, 1, 'control');
INSERT INTO PAL_TRIAL_DATA_TBL VALUES(42, 1, 1, 'control');

DROP TABLE PAL_TRIAL_ESTIMATES_TBL;
CREATE COLUMN TABLE PAL_TRIAL_ESTIMATES_TBL LIKE PAL_TRIAL_ESTIMATES_T;
DROP TABLE PAL_TRIAL_LOGRANK_STAT1_TBL;
CREATE COLUMN TABLE PAL_TRIAL_LOGRANK_STAT1_TBL LIKE PAL_TRIAL_LOGRANK_STAT1_T;
DROP TABLE PAL_TRIAL_LOGRANK_STAT2_TBL;
CREATE COLUMN TABLE PAL_TRIAL_LOGRANK_STAT2_TBL LIKE PAL_TRIAL_LOGRANK_STAT2_T;
CALL "DM_PAL".PAL_KMSURV_PROC(PAL_TRIAL_DATA_TBL, #PAL_CONTROL_TBL,
PAL_TRIAL_ESTIMATES_TBL, PAL_TRIAL_LOGRANK_STAT1_TBL,
PAL_TRIAL_LOGRANK_STAT2_TBL) with OVERVIEW;
SELECT * FROM PAL_TRIAL_ESTIMATES_TBL;
SELECT * FROM PAL_TRIAL_LOGRANK_STAT1_TBL;
SELECT * FROM PAL_TRIAL_LOGRANK_STAT2_TBL;

```

Option 2: Weibull Distribution

Technology Background

Weibull distribution is often used for reliability and survival analysis. It is defined by 3 parameters: shape, scale, and location. Scale works as key to magnify or shrink the curve. Shape is the crucial factor to define how the curve looks like, as described below:

- Shape = 1: The failure rate is constant over time, indicating random failure.
- Shape < 1: The failure rate decreases over time.
- Shape > 1: The failure rate increases over time.

For the same raw data as in the above Kaplan-Meier option, also shown below:

Treatment	Survival Time (months)
Linoleic acid (n = 25)	1+, 5+, 6, 6, 9+, 10, 10, 10+, 12, 12, 12, 12, 12+, 13+, 15+, 16+, 20+, 24, 24+, 27+, 32, 34+, 36+, 36+, 44+
Control (n = 24)	3+, 6, 6, 6, 6, 8, 8, 12, 12, 12+, 15+, 16+, 18+, 18+, 20, 22+, 24, 28+, 28+, 30, 30+, 33+, 42

The DISTRFITCENSORED function is used to fit the Weibull distribution on the censored data. For the two types of treatment, linoleic acid and control, two separate calls of DISTRFITCENSORED are performed to get two Weibull distributions.

Implementation Steps

Assume that:

- DM_PAL is a schema belonging to USER1; and
- USER1 has been assigned the AFLPM_CREATOR_ERASER_EXECUTE role; and
- USER1 has been assigned the AFL__SYS_AFL_AFLPAL_EXECUTE or AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION role.

Step 1

Get Weibull distribution and statistics from the linoleic acid treatment data:

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DISTRFITCENSORED_DATA_T;
CREATE TYPE PAL_DISTRFITCENSORED_DATA_T AS TABLE(
"LEFT" DOUBLE,
"RIGHT" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_DISTRFITCENSORED_ESTIMATION_T;
CREATE TYPE PAL_DISTRFITCENSORED_ESTIMATION_T AS TABLE(
"NAME" VARCHAR(50),
"VALUE" VARCHAR(50)
);
DROP TYPE PAL_DISTRFITCENSORED_STATISTICS_T;
CREATE TYPE PAL_DISTRFITCENSORED_STATISTICS_T AS TABLE(
"NAME" VARCHAR(50),
"VALUE" DOUBLE
);
DROP TABLE PDATA_TBL;
CREATE COLUMN TABLE PDATA_TBL(
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_DISTRFITCENSORED_DATA_T', 'IN');
INSERT INTO PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_DISTRFITCENSORED_ESTIMATION_T',
'OUT');
INSERT INTO PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_DISTRFITCENSORED_STATISTICS_T',
'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DISTRFITCENSORED_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL',
'DISTRFITCENSORED','DM_PAL','PAL_DISTRFITCENSORED_PROC',PDATA_TBL);
DROP TABLE PAL_DISTRFITCENSORED_DATA_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_DATA_TBL LIKE
PAL_DISTRFITCENSORED_DATA_T;
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (1,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (5,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (6,6);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (6,6);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (9,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (10,10);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (10,10);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (10,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (12,12);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (12,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (13,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (15,NULL);

```

```

INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (16,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (20,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (24,24);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (24,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (27,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (32,32);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (34,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (36,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (36,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (44,NULL);
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
INSERT INTO PAL_CONTROL_TBL VALUES ('DISTRIBUTIONNAME', NULL, NULL,'WEIBULL');
INSERT INTO PAL_CONTROL_TBL VALUES ('OPTIMAL_METHOD',0, NULL, NULL);
DROP TABLE PAL_DISTRFITCENSORED_ESTIMATION_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_ESTIMATION_TBL LIKE
PAL_DISTRFITCENSORED_ESTIMATION_T;
DROP TABLE PAL_DISTRFITCENSORED_STATISTICS_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_STATISTICS_TBL LIKE
PAL_DISTRFITCENSORED_STATISTICS_T;
CALL DM_PAL.PAL_DISTRFITCENSORED_PROC(PAL_DISTRFITCENSORED_DATA_TBL,
PAL_CONTROL_TBL, PAL_DISTRFITCENSORED_ESTIMATION_TBL,
PAL_DISTRFITCENSORED_STATISTICS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_DISTRFITCENSORED_ESTIMATION_TBL;
SELECT * FROM PAL_DISTRFITCENSORED_STATISTICS_TBL;

```

The expected results are as follows:

	NAME	VALUE
1	DISTRIBUTIONNAME	WEIBULL
2	SCALE	36.3069
3	SHAPE	1.40528

	NAME	VALUE
1	LOGLIKELIHOOD	-47.035827236974946

Step 2

Get Weibull distribution and statistics from the control treatment data:

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DISTRFITCENSORED_DATA_T;
CREATE TYPE PAL_DISTRFITCENSORED_DATA_T AS TABLE(
"LEFT" DOUBLE,
"RIGHT" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
DROP TYPE PAL_DISTRFITCENSORED_ESTIMATION_T;
CREATE TYPE PAL_DISTRFITCENSORED_ESTIMATION_T AS TABLE(
"NAME" VARCHAR(50),
"VALUE" VARCHAR(50)
);

```

```

DROP TYPE PAL_DISTRFITCENSORED_STATISTICS_T;
CREATE TYPE PAL_DISTRFITCENSORED_STATISTICS_T AS TABLE(
  "NAME" VARCHAR(50),
  "VALUE" DOUBLE
);
DROP TABLE PDATA_TBL;
CREATE COLUMN TABLE PDATA_TBL(
  "POSITION" INT,
  "SCHEMA_NAME" NVARCHAR(256),
  "TYPE_NAME" NVARCHAR(256),
  "PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_DISTRFITCENSORED_DATA_T', 'IN');
INSERT INTO PDATA_TBL VALUES (2, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_DISTRFITCENSORED_ESTIMATION_T',
  'OUT');
INSERT INTO PDATA_TBL VALUES (4, 'DM_PAL', 'PAL_DISTRFITCENSORED_STATISTICS_T',
  'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_DISTRFITCENSORED_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL',
  'DISTRFITCENSORED', 'DM_PAL', 'PAL_DISTRFITCENSORED_PROC', PDATA_TBL);
DROP TABLE PAL_DISTRFITCENSORED_DATA_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_DATA_TBL LIKE
PAL_DISTRFITCENSORED_DATA_T;
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (3,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (6,6);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (8,8);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (8,8);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (12,12);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (12,12);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (12,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (15,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (16,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (18,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (18,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (20,20);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (22,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (24,24);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (28,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (28,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (30,30);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (30,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (33,NULL);
INSERT INTO PAL_DISTRFITCENSORED_DATA_TBL VALUES (42,42);
DROP TABLE PAL_CONTROL_TBL;
CREATE COLUMN TABLE PAL_CONTROL_TBL(
  "NAME" VARCHAR (50),
  "INTARGS" INTEGER,
  "DOUBLEARGS" DOUBLE,
  "STRINGARGS" VARCHAR (100)
);
INSERT INTO PAL_CONTROL_TBL VALUES ('DISTRIBUTIONNAME', NULL, NULL, 'WEIBULL');
INSERT INTO PAL_CONTROL_TBL VALUES ('OPTIMAL_METHOD', 0, NULL, NULL);
DROP TABLE PAL_DISTRFITCENSORED_ESTIMATION_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_ESTIMATION_TBL LIKE
PAL_DISTRFITCENSORED_ESTIMATION_T;
DROP TABLE PAL_DISTRFITCENSORED_STATISTICS_TBL;
CREATE COLUMN TABLE PAL_DISTRFITCENSORED_STATISTICS_TBL LIKE
PAL_DISTRFITCENSORED_STATISTICS_T;
CALL DM_PAL.PAL_DISTRFITCENSORED_PROC(PAL_DISTRFITCENSORED_DATA_TBL,
  PAL_CONTROL_TBL, PAL_DISTRFITCENSORED_ESTIMATION_TBL,
  PAL_DISTRFITCENSORED_STATISTICS_TBL) WITH OVERVIEW;
SELECT * FROM PAL_DISTRFITCENSORED_ESTIMATION_TBL;
SELECT * FROM PAL_DISTRFITCENSORED_STATISTICS_TBL;

```

The expected results are as follows:

	NAME	VALUE
1	DISTRIBUTIONNAME	WEIBULL
2	SCALE	20.444
3	SHAPE	1.71902

	NAME	VALUE
1	LOGLIKELIHOOD	-57.590127685481136

The results show that the shape values for both treatments are greater than 1, indicating the failure rate increases over time.

Step 3

Get the CDF (cumulative distribution function) of Weibull distribution for the linoleic acid treatment data:

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DISTRPROB_DATA_T;
CREATE TYPE PAL_DISTRPROB_DATA_T AS TABLE ("DATACOL" DOUBLE);
DROP TYPE PAL_DISTRPROB_DISTRPARAM_T;
CREATE TYPE PAL_DISTRPROB_DISTRPARAM_T AS TABLE (
"NAME" VARCHAR(50),
"VALUE" VARCHAR(50)
);
DROP TYPE PAL_DISTRPROB_RESULT_T;
CREATE TYPE PAL_DISTRPROB_RESULT_T AS TABLE (
"INPUTDATA" DOUBLE,
"PROBABILITY" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE (
"NAME" VARCHAR(50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR(100)
);
DROP TABLE PAL_DISTRPROB_PDATA_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_PDATA_TBL(
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)
);
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (1, 'DM_PAL','PAL_DISTRPROB_DATA_T',
'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (2,'DM_PAL',
'PAL_DISTRPROB_DISTRPARAM_T', 'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (3, 'DM_PAL','PAL_CONTROL_T', 'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (4,
'DM_PAL','PAL_DISTRPROB_RESULT_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL','PAL_DISTRPROB_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL',
'DISTRPROB','DM_PAL','PAL_DISTRPROB_PROC', PAL_DISTRPROB_PDATA_TBL);
DROP TABLE PAL_DISTRPROB_DATA_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_DATA_TBL LIKE PAL_DISTRPROB_DATA_T;
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (6);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (8);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (12);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (20);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (24);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (30);

```

```

INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (42);
DROP TABLE PAL_DISTRPROB_DISTRPARAM_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_DISTRPARAM_TBL LIKE PAL_DISTRPROB_DISTRPARAM_T;
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('DistributionName', 'Weibull');
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('Shape', '1.40528');
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('Scale', '36.3069');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LOWER_UPPER',0,null,null);
DROP TABLE PAL_DISTRPROB_RESULT_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_RESULT_TBL LIKE PAL_DISTRPROB_RESULT_T;
CALL DM_PAL.PAL_DISTRPROB_PROC(PAL_DISTRPROB_DATA_TBL,
PAL_DISTRPROB_DISTRPARAM_TBL, #PAL_CONTROL_TBL, PAL_DISTRPROB_RESULT_TBL) WITH
OVERVIEW;
SELECT * FROM PAL_DISTRPROB_RESULT_TBL;

```

The expected result is as follows:

	INPUTDATA	PROBABILITY
1	6	0.07657965127936783
2	8	0.1125151447665862
3	12	0.19024473441441703
4	20	0.35118218925555456
5	24	0.42818254577905457
6	30	0.5345725815649545
7	42	0.7068736406026395

Step 4

Get the CDF (cumulative distribution function) of Weibull distribution for the control treatment data:

```

SET SCHEMA DM_PAL;
DROP TYPE PAL_DISTRPROB_DATA_T;
CREATE TYPE PAL_DISTRPROB_DATA_T AS TABLE ("DATACOL" DOUBLE);
DROP TYPE PAL_DISTRPROB_DISTRPARAM_T;
CREATE TYPE PAL_DISTRPROB_DISTRPARAM_T AS TABLE(
"NAME" VARCHAR(50),
"VALUE" VARCHAR(50)
);
DROP TYPE PAL_DISTRPROB_RESULT_T;
CREATE TYPE PAL_DISTRPROB_RESULT_T AS TABLE(
"INPUTDATA" DOUBLE,
"PROBABILITY" DOUBLE
);
DROP TYPE PAL_CONTROL_T;
CREATE TYPE PAL_CONTROL_T AS TABLE(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
DROP TABLE PAL_DISTRPROB_PDATA_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_PDATA_TBL(
"POSITION" INT,
"SCHEMA_NAME" NVARCHAR(256),
"TYPE_NAME" NVARCHAR(256),
"PARAMETER_TYPE" VARCHAR(7)

```

```

);
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (1, 'DM_PAL', 'PAL_DISTRPROB_DATA_T',
'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (2, 'DM_PAL',
'PAL_DISTRPROB_DISTRPARAM_T', 'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (3, 'DM_PAL', 'PAL_CONTROL_T', 'IN');
INSERT INTO PAL_DISTRPROB_PDATA_TBL VALUES (4,
'DM_PAL', 'PAL_DISTRPROB_RESULT_T', 'OUT');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_DROP('DM_PAL', 'PAL_DISTRPROB_PROC');
CALL SYS.AFLLANG_WRAPPER_PROCEDURE_CREATE('AFLPAL',
'DISTRPROB', 'DM_PAL', 'PAL_DISTRPROB_PROC', 'PAL_DISTRPROB_PDATA_TBL');
DROP TABLE PAL_DISTRPROB_DATA_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_DATA_TBL LIKE PAL_DISTRPROB_DATA_T;
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (6);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (10);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (12);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (24);
INSERT INTO PAL_DISTRPROB_DATA_TBL VALUES (32);
DROP TABLE PAL_DISTRPROB_DISTRPARAM_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_DISTRPARAM_TBL LIKE PAL_DISTRPROB_DISTRPARAM_T;
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('DistributionName', 'Weibull');
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('Shape', '1.71902');
INSERT INTO PAL_DISTRPROB_DISTRPARAM_TBL VALUES ('Scale', '20.444');
DROP TABLE #PAL_CONTROL_TBL;
CREATE LOCAL TEMPORARY COLUMN TABLE #PAL_CONTROL_TBL(
"NAME" VARCHAR (50),
"INTARGS" INTEGER,
"DOUBLEARGS" DOUBLE,
"STRINGARGS" VARCHAR (100)
);
INSERT INTO #PAL_CONTROL_TBL VALUES ('LOWER_UPPER',0,null,null);
DROP TABLE PAL_DISTRPROB_RESULT_TBL;
CREATE COLUMN TABLE PAL_DISTRPROB_RESULT_TBL LIKE PAL_DISTRPROB_RESULT_T;
CALL DM_PAL.PAL_DISTRPROB_PROC(PAL_DISTRPROB_DATA_TBL,
PAL_DISTRPROB_DISTRPARAM_TBL, #PAL_CONTROL_TBL, PAL_DISTRPROB_RESULT_TBL) WITH
OVERVIEW;
SELECT * FROM PAL_DISTRPROB_RESULT_TBL;

```

The expected result is as follows:

	INPUTDATA	PROBABILITY
1	6	0.11445660910656985
2	10	0.25360736088540314
3	12	0.3297943140085646
4	24	0.7321725929430624
5	32	0.8846979463928679

Related Information

[Kaplan-Meier Survival Analysis \[page 509\]](#)

[Distribution Fitting \[page 494\]](#)

5 Best Practices

- Create an SQL view for the input table if the table structure does not meet what is specified in this guide.
- Avoid null values in the input data. You can replace the null values with the default values via an SQL statement (SQL view or SQL update) because PAL functions cannot infer the default values.
- Create the parameter table as a local temporary table to avoid table name conflicts.
- If you do not use PMML export, you do not need to create a PMML output table to store the result. Just set the `PMML_EXPORT` parameter to 0 and pass `? or null` to the function.
- When using the `KMEANS` function, different `INIT_TYPE` and `NORMALIZATION` settings may produce different results. You may need to try a few combinations of these two parameters to get the best result.
- When using the `APRIORIRULE` function, in some circumstances the rules set can be huge. To avoid an extra long runtime, you can set the `MAXITEMLENGTH` parameter to a smaller number, such as 2 or 3.

6 Important Disclaimer for Features in SAP HANA Platform, Options and Capabilities

SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support Package Stack (SPS). Although various features included in SAP HANA options and capabilities are cited in the SAP HANA platform documentation, each SAP HANA edition governs the options and capabilities available. Based on this, customers do not necessarily have the right to use features included in SAP HANA options and capabilities. For customers to whom these license restrictions apply, the use of features included in SAP HANA options and capabilities in a production system requires purchasing the corresponding software license(s) from SAP. The documentation for the SAP HANA options is available in SAP Help Portal. If you have additional questions about what your particular license provides, or wish to discuss licensing features available in SAP HANA options, please contact your SAP account team representative.

Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP's gross negligence.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP's gross negligence or willful misconduct. All links are categorized for transparency (see: <https://help.sap.com/viewer/disclaimer>).



[go.sap.com/registration/
contact.html](http://go.sap.com/registration/contact.html)



© 2018 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/corporate/en/legal/copyright.html> for additional trademark information and notices.