

Teradata Vantage™ - Advanced SQL Engine Analytic Functions

Release 17.10

July 2021



Copyright and Trademarks

Copyright © 2017 - 2021 by Teradata. All Rights Reserved.

All copyrights and trademarks used in Teradata documentation are the property of their respective owners. For more information, see [Trademark Information](#).

Product Safety

Safety type	Description
	Indicates a situation which, if not avoided, could result in damage to property, such as to equipment or data, but not related to personal injury.
	Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury.
	Indicates a hazardous situation which, if not avoided, could result in death or serious personal injury.

Third-Party Materials

Non-Teradata (i.e., third-party) sites, documents or communications ("Third-party Materials") may be accessed or accessible (e.g., linked or posted) in or in connection with a Teradata site, document or communication. Such Third-party Materials are provided for your convenience only and do not imply any endorsement of any third party by Teradata or any endorsement of Teradata by such third party. Teradata is not responsible for the accuracy of any content contained within such Third-party Materials, which are provided on an "AS IS" basis by Teradata. Such third party is solely and directly responsible for its sites, documents and communications and any harm they may cause you or others.

Warranty Disclaimer

Except as may be provided in a separate written agreement with Teradata or required by applicable law, the information available from the Teradata Documentation website or contained in Teradata information products is provided on an "as-is" basis, without warranty of any kind, either express or implied, including the implied warranties of merchantability, fitness for a particular purpose, or noninfringement.

The information available from the Teradata Documentation website or contained in Teradata information products may contain references or cross-references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that Teradata Corporation intends to announce such features, functions, products, or services in your country. Please consult your local Teradata Corporation representative for those features, functions, products, or services available in your country.

The information available from the Teradata Documentation website or contained in Teradata information products may be changed or updated by Teradata at any time without notice. Teradata may also make changes in the products or services described in this information at any time without notice.

Machine-Assisted Translation

Certain materials on this website have been translated using machine-assisted translation software/tools. Machine-assisted translations of any materials into languages other than English are intended solely as a convenience to the non-English-reading users and are not legally binding. Anybody relying on such information does so at his or her own risk. No automated translation is perfect nor is it intended to replace human translators. Teradata does not make any promises, assurances, or guarantees as to the accuracy of the machine-assisted translations provided. Teradata accepts no responsibility and shall not be liable for any damage or issues that may result from using such translations. Users are reminded to use the English contents.

Feedback

To maintain the quality of our products and services, e-mail your comments on the accuracy, clarity, organization, and value of this document to: docs@teradata.com.

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed nonconfidential. Without any payment or other obligation of any kind and without any restriction of any kind, Teradata and its affiliates are hereby free to (1) reproduce, distribute, provide access to, publish, transmit, publicly display, publicly perform, and create derivative works of, the Feedback, (2) use any ideas, concepts, know-how, and techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, and marketing products and services incorporating the Feedback, and (3) authorize others to do any or all of the above.

Contents

Chapter 1: Introduction to Teradata Vantage	5
Advanced SQL Engine Analytic Functions Overview	5
Usage Notes	8
AA 7.00 Usage Notes	11
Chapter 2: Advanced SQL Engine Analytic Functions	24
Antiselect (SQL Engine)	24
Attribution (SQL Engine)	27
DecisionForestPredict (SQL Engine)	36
DecisionTreePredict (SQL Engine)	48
GLMPredict (SQL Engine)	57
MovingAverage (SQL Engine)	69
NaiveBayesPredict (SQL Engine)	85
NaiveBayesTextClassifierPredict (SQL Engine)	90
NGramSplitter (SQL Engine)	97
nPath® (SQL Engine)	103
Pack (SQL Engine)	146
Sessionize (SQL Engine)	151
StringSimilarity (SQL Engine)	155
SVMSparsePredict (SQL Engine)	161
Unpack (SQL Engine)	167
Chapter 3: Data Cleaning Functions	177
TD_ConvertTo	177
TD_GetRowsWithoutMissingValues	182
TD_OutlierFilterFit	184
TD_OutlierFilterTransform	189
TD_SimpleImputeFit	190
TD_SimpleImputeTransform	194
Chapter 4: Data Exploration Functions	197
TD_CategoricalSummary	197
TD_ColumnSummary	199
TD_GetRowsWithMissingValues	202
TD_Histogram	204
TD_QQNorm	208
TD_UnivariateStatistics	211
TD_WhichMax	215
TD_WhichMin	217

Chapter 5: Feature Engineering Transform Functions	219
TD_BinCodeFit	219
TD_BinCodeTransform	225
TD_FunctionFit	226
TD_FunctionTransform	229
TD_OneHotEncodingFit	232
TD_OneHotEncodingTransform	236
TD_PolynomialFeaturesFit	238
TD_PolynomialFeaturesTransform	241
TD_RowNormalizeFit	243
TD_RowNormalizeTransform	246
TD_ScaleFit	248
TD_ScaleTransform	254
Chapter 6: Feature Engineering Utility Functions	256
TD_FillRowID	256
TD_NumApply	258
TD_RoundColumns	261
TD_StrApply	264
Chapter 7: Hypothesis Testing Functions	270
Hypothesis Test Components	270
Hypothesis Test Types	270
TD_ChiSq	271
TD_FTest	277
TD_ZTest	284
Appendix A: How to Read Syntax	294
Appendix B: Additional Information	296

Introduction to Teradata Vantage

Teradata Vantage™ is our flagship analytic platform offering, which evolved from our industry-leading Teradata® Database. Until references in content are updated to reflect this change, the term Teradata Database is synonymous with Teradata Vantage.

Advanced SQL Engine is a core capability of Teradata Vantage, based on our best-in-class Teradata Database. Advanced SQL refers to the ability to run advanced analytic functions beyond that of standard SQL.

Advanced SQL Engine Analytic Functions Overview

Advanced SQL Engine (was NewSQL Engine) analytic functions are specifically for analyzing data. Data can include clickstreams, financial transaction data, and user interaction data.

Advanced SQL Engine Analytical Functions

Function Name	Description
Antiselect (SQL Engine)	AntiSelect returns all columns <i>except</i> those specified.
Attribution (SQL Engine)	Calculates attributions with a wide range of distribution models. Often used in web-page analysis.
DecisionForestPredict (SQL Engine)	Uses the model file output by Machine Learning Engine (ML Engine) DecisionForest function to analyze the input data and make predictions.
DecisionTreePredict (SQL Engine)	Uses the model file output by ML Engine DecisionTree function to analyze the input data and make predictions.
GLMPredict (SQL Engine)	Uses the model file output by ML Engine GLM function to analyze the input data and make predictions.
MovingAverage (SQL Engine)	Computes average values in a series.
NaiveBayesPredict (SQL Engine)	Uses the model file output by ML Engine Naive Bayes Classifier function to analyze the input data and make predictions.
NaiveBayesTextClassifierPredict (SQL Engine)	Uses the model file output by ML Engine NaiveBayesTextClassifierTrainer function to analyze the input data and make predictions.
NGramSplitter (SQL Engine)	Tokenizes (splits) an input stream and emits <i>n</i> multigrams, based on specified delimiter and reset parameters. Useful for sentiment analysis, topic identification, and document classification.
nPath® (SQL Engine)	Performs regular pattern matching over a sequence of rows from one or more inputs.

Function Name	Description
Pack (SQL Engine)	Compresses data in multiple columns into a single packed data column.
Sessionize (SQL Engine)	Maps each click in a clickstream to a unique session identifier.
StringSimilarity (SQL Engine)	Calculates the similarity between two strings, using the specified comparison method.
SVMSparsePredict (SQL Engine)	Uses the model file output by ML Engine SVMSparse function to analyze the input data and make predictions.
Unpack (SQL Engine)	Expands data from a single packed column to multiple unpacked columns.

Advanced SQL Engine analytic functions have corresponding ML Engine functions. The syntax of corresponding functions may differ, but given the same inputs and syntax element values, they produce the same results (with the minor exceptions noted in specific functions).

To execute ML Engine functions on Teradata Vantage™, contact your Teradata Support representative.

Data Cleaning Functions

Function Name	Description
TD_ConvertTo	Converts the specified input table columns to specified data types.
TD_GetRowsWithoutMissingValues	Displays the rows that have non-NULL values in the specified input table columns.
TD_OutlierFilterFit	Calculates the lower_percentile, upper_percentile, count of rows, and median for the specified input table columns.
TD_OutlierFilterTransform	Filters outliers from the input table
TD_SimpleImputeFit	Outputs a table of values to substitute for missing values in the input table.
TD_SimpleImputeTransform	Substitutes specified values for missing values in the input table.

Data Exploration Functions

Function Name	Description
TD_CategoricalSummary	Displays the distinct values and their counts for each specified input table column.
TD_ColumnSummary	Displays a summary of each specified input table column.
TD_GetRowsWithMissingValues	Displays the rows that have NULL values in the specified input table columns.
TD_Histogram	Calculates the frequency distribution of a data set.

Function Name	Description
TD_QQNorm	Checks whether the values in the specified input table columns are normally distributed.
TD_UnivariateStatistics	Displays descriptive statistics for each specified numeric input table column.
TD_WhichMax	Displays all rows that have the maximum value in a specified input table column.
TD_WhichMin	Displays all rows that have the minimum value in specified input table column.

Feature Engineering Transform Functions

Function Name	Description
TD_BinCodeFit	Converts numeric data to categorical data by binning the numeric data into multiple numeric bins (intervals).
TD_BinCodeTransform	Transforms input table columns from the BinCodeFit function output.
TD_FunctionFit	Determines whether specified numeric transformations can be applied to specified input columns.
TD_FunctionTransform	Applies numeric transformations to input columns to the FunctionFit output.
TD_OneHotEncodingFit	Outputs a table of attributes and categorical values to the TD_OneHotEncodingTransform function.
TD_OneHotEncodingTransform	Encodes specified attributes and categorical values as one-hot numeric vectors using the output from the TD_OneHotEncodingFit function.
TD_PolynomialFeaturesFit	Stores all the specified values in the argument in a tabular format.
TD_PolynomialFeaturesTransform	Extracts values of arguments from the output of the TD_PolynomialFeaturesFit function and generates a feature matrix of all polynomial combinations of the features.
TD_RowNormalizeFit	Outputs a table of parameters and specified input columns to TD_RowNormalizeTransform which normalizes the input columns row-wise.
TD_RowNormalizeTransform	Normalizes the input columns row-wise using the output of the TD_RowNormalizeFit function.
TD_ScaleFit	Outputs a table of statistics to the TD_ScaleTransform function.
TD_ScaleTransform	Scales the specified input table columns using the output of the TD_ScaleFit function.

Feature Engineering Utility Functions

Function Name	Description
TD_FillRowID	Adds a column of unique row identifiers to the input table.
TD_NumApply	Applies a specified numeric operator to the specified input table columns.
TD_RoundColumns	Rounds the values of each specified input table column to a specified number of decimal places
TD_StrApply	Applies a specified string operator to the specified input table columns.

Hypothesis Testing Functions

Function Name	Description
TD_ChiSq	Performs Pearson's chi-squared test for independence.
TD_FTest	Performs an <i>F</i> -test, for which the test statistic has an <i>F</i> -distribution under the null hypothesis.
TD_ZTest	Performs a Z-test, for which the distribution of the test statistic under the null hypothesis can be approximated by normal distribution.

More Information

Topic	Reference
ML Engine functions	<i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003
Aster Analytics functions	<i>Teradata Aster® Analytics Foundation User Guide</i> , B700-1022
Installing model files output by ML Engine functions on Advanced SQL Engine	<i>Teradata Vantage™ User Guide</i> , B700-4002

Usage Notes

These usage notes apply to every function in this document.

Function Syntax Descriptions

SELECT Statement Clauses

The function syntax descriptions in this document are SQL SELECT statements. For simplicity, the descriptions do not show every possible SELECT statement clause. However, you can use any valid

SELECT statement clauses. For information about SELECT statement options, see *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146.

Many examples in this document use ORDER BY clauses that the function syntax descriptions do not show.

Function Syntax Element Order

Function syntax elements must appear after the USING clause, but they need not appear in the order shown in the function syntax description.

Many examples in this document do not specify their syntax elements in the order shown in the function syntax description.

Related Information:

[How to Read Syntax](#)

Column Specification Syntax Elements

Some ML Engine functions have column specification syntax elements with this syntax:

```
syntax_element ( { 'column' | column_range } [, ...] )
```

The *column* is a column name. This is the syntax of *column_range*:

```
'start_column:end_column' [, '-exclude_column' ]
```

The range includes its endpoints.

The *start_column* and *end_column* can be:

- Column names (for example, '*column1:column2*')
 - Nonnegative integers that represent the indexes of columns in the table (for example, '*[0:4]*')
 - The first column has index 0; therefore, '*[0:4]*' specifies the first five columns in the table.

- Empty. For example:
 - '*[:4]*' specifies all columns up to and including the column with index 4.
 - '*[4:]*' specifies the column with index 4 and all columns after it.
 - '*[:]*' specifies all columns in the table.

The *exclude_column* is a column in the specified range, represented by either its name or its index (for example, '*[0:99]*', '*-[50]*', '*-column10*' specifies the columns with indexes 0 through 99, except the column with index 50 and *column10*).

Column ranges cannot overlap, and cannot include any specified *column*.

Functions Ignore Disallowed Syntax Elements

If you call a function with a syntax element that is disallowed, the function ignores the syntax element, without returning an error.

Input Table Schemas

Input table schemas show only the columns that a function uses. Unless otherwise noted, input tables can have additional columns, but the function ignores them.

Terminology

This document uses the following terms.

Term	Description
Path	An ordered, start-to-finish series of actions, for example, page views, for which sequences and sub-sequences can be created.
Sequence	A sequence is the path prefixed with a carat (^), which indicates the start of a path. For example, if a user visited page a, page b, and page c, in that order, the session sequence is ^,a,b,c.
Subsequence	For a given sequence of actions, a sub-sequence is one possible subset of the steps that begins with the initial action. For example, the path a,b,creates three subsequences: ^,a; ^,a,b; and ^,a,b,c.

Displaying Online Help for Advanced SQL Engine Analytic Functions

Online help is available for each Advanced SQL Engine analytic function.

- For information about a function, type:
`HELP 'function_name'`
 For example:
`HELP 'SQL NPATH'`

BC/BCE Timestamps

Advanced SQL Engine functions do not support Before the Common Era (BCE) timestamps. BCE is an alternative to Before Christ (BC). These are examples of BC/BCE timestamps:

```
4713-01-01 11:07:11-07:52:58 BC
4713-01-01 11:07:11 BC
```

Workload Management Configuration for Advanced SQL Engine Analytic Functions

Advanced SQL Engine analytic functions can be memory- and compute-intensive and impact other workloads, depending on function parameters and input table sizes. To learn to use Workload Management throttles to limit concurrency and memory, see *Teradata Vantage™ - Workload Management User Guide*, B035-1197.

AA 7.00 Usage Notes

These usage notes apply only if you use model tables created using a supported version of Aster Analytics on Aster Database as input to these functions:

- DecisionTreePredict
- DecisionForestPredict
- GLMPredict
- NaiveBayesPredict
- NaiveBayesTextClassifierPredict
- SVMSParsePredict

AA 7.00 Limitations

- The minimum supported version of Aster Analytics is AA 7.00.

Models created using an earlier version of Aster Analytics must be recreated after upgrading to a supported version of Aster Analytics.

AA 7.00 Model Tables

These notes apply only to model tables output by AA 7.00 functions.

- BLOB and CLOB data types are not supported.
- Aster data type BYTEA corresponds to Advanced SQL Engine data type VARBYTE.
- The maximum size of a VARBYTE or VARCHAR column is 64000.
- You can load a table created on Aster Database to Advanced SQL Engine using either the `load_to_teradata` command or Open Database Connectivity (ODBC).

Analytic Functions on Advanced SQL Engine and Aster Database

The following table summarizes the differences between analytic functions on Advanced SQL Engine and Aster Database.

Advanced SQL Engine Analytic Function	Aster Database Analytic Function
PARTITION BY clause lets you specify a column by its position, an integer. PARTITION BY 1 partitions rows by column 1.	PARTITION BY clause accepts only column names. PARTITION BY 1 causes the function to process all rows on a single worker node.
For table operator output, an alias is required.	For function output, an alias is optional.
To specify function syntax elements, you must use a USING clause.	Function syntax does not include USING clause.
Function syntax elements do not support column ranges.	Function syntax elements support column ranges.

Loading Aster Tables to Advanced SQL Engine Using load_to_teradata

For load_to_teradata instructions, see *Teradata Aster® Database User Guide* and the following usage notes.

load_to_teradata Usage Notes

- If a table column name includes a keyword, enclose the name in double quotation marks and alias it.
- In SELECT statements, enclose every camel-case table column name in double quotation marks.

This example shows both aliased columns and camel-case column names:

```
SELECT * FROM load_to_teradata (
  ON (
    SELECT "class" AS class_col,
           "variable" AS variable_col,
           "type" AS type_col,
           category,
           cnt,
           "sum" AS sum_col,
           "sumSq",
           "totalCnt"
    FROM aster_nb_modelSC
  )
  tdpid ('sdt12432.labs.teradata.com')
  username ('sample_user')
  password ('sample_user')
  target_table ('td_nb_modelSC')
);
```

- Cast every REAL column to DOUBLE PRECISION.

For example:

```

SELECT * FROM load_to_teradata (
  ON (
    SELECT attribute,
           predictor,
           category,
           CAST (estimate AS DOUBLE PRECISION) AS estimate,
           CAST (std_err AS DOUBLE PRECISION) AS std_err,
           CAST (z_score AS DOUBLE PRECISION) AS z_score,
           CAST (p_value AS DOUBLE PRECISION) AS p_value,
           significance,
           "family"
    FROM glm_housing_model
  )
  tdpid ('sdt12432.labs.teradata.com')
  username ('sample_user')
  password ('sample_user')
  target_table ('glm_housing_model')
);

```

- If a model table column name contains Advanced SQL Engine reserved keywords or special characters—characters other than letters, digits, or underscore (_)—enclose it in double quotation marks.

This rule applies to the following model column names:

AA 7.00 Function	Model Column Name
Single_Tree_Drive	node_gini(p) node_entropy(p) node_chisq_pv(p) split_gini(p) split_entropy(p) split_chisq_pv(p)
NaiveBayesReduce	class variable type sum sumSq totalCnt

For example:

```

CREATE SET TABLE NBUSER.td_glass_modelPD1,
  FALLBACK,
  NO BEFORE JOURNAL,

```

```

NO AFTER JOURNAL,
CHECKSUM = DEFAULT,
DEFAULT MERGEBLOCKRATIO,
MAP = TD_MAP1 (
  node_id BIGINT,
  node_size BIGINT,
  "node_gini(p)" FLOAT,
  node_entropy FLOAT,
  node_chisq_pv FLOAT,
  node_label VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
  node_majorvotes BIGINT,
  split_value FLOAT,
  "split_gini(p)" FLOAT,
  split_entropy FLOAT,
  split_chisq_pv FLOAT,
  left_id BIGINT,
  left_size BIGINT,
  left_label VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
  left_majorvotes BIGINT,
  right_id BIGINT,
  right_size BIGINT,
  right_label VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
  right_majorvotes BIGINT,
  left_bucket VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
  right_bucket VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
  left_label_problist VARCHAR(2048) CHARACTER SET UNICODE
NOT CASESPECIFIC,
  right_label_problist VARCHAR(2048) CHARACTER SET UNICODE
NOT CASESPECIFIC,
  prob_label_order VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
  attribute VARCHAR(2048) CHARACTER SET UNICODE NOT CASESPECIFIC,
)
PRIMARY INDEX (node_id);

```

Related Information:

[Loading Aster Tables to Advanced SQL Engine Using ODBC](#)

Loading Aster Tables to Advanced SQL Engine Using ODBC

The ODBC instructions follow. To follow them, you must have an account on <https://downloads.teradata.com>.

1. [Install Teradata Parallel Transporter Base](#).
2. [Set up the Aster driver on the client machine](#).

3. If the table does not exist on Aster Database, create and populate it there.
4. On Advanced SQL Engine, do the following:
 - a. If the user who is to own the table does not exist, create it.
 - b. [Write the tpt script.](#)
 - c. [Write the JobVariablesFile.](#)
 - d. [Use the tbuild command to run the tpt script.](#)

Related Information:

[Loading Aster Tables to Advanced SQL Engine Using load_to_teradata](#)
[Example: Loading Aster Table to Advanced SQL Engine Using ODBC](#)

Installing Teradata Parallel Transporter Base

1. Go to <https://support.teradata.com>.
2. Log in.
3. Download the package TTU 16.20.04.00 Windows - Base.
 Downloading the packages takes approximately 30-40 minutes.
4. On your client machine, go to the folder where the package was downloaded and unzip it.
5. Go to TeradataToolsAndUtilitiesBase\Windows and run TTU.exe.
6. Install Teradata Parallel Transporter Base.

Setting Up the Aster Driver on the Client

1. Go to <https://support.teradata.com>.
2. Log in.
3. Download AsterClients__windows_x8664.version.zip, where *version* is the version of Aster Analytics on your client machine; for example:
 AsterClients__windows_x8664.06.20.00.00.zip
 Downloading the packages takes several minutes.
4. On your client machine, go to the folder where the package was downloaded and unzip it.
5. Go to the subfolder \stage\home\beehive\clients-win*nn*, where *nn* is 32, 64, or 86, depending on your Windows machine. For example:
 \stage\home\beehive\clients-win64
6. Install nClusterODBCInstaller_*xnn*.
 If the installer requests a dependency package, install it from the web.
7. Open ODBC Data Sources (*nn*-bit).
8. On the **System DSN** tab, select **Add**.
 If Aster ODBC driver installation succeeded, the window **Aster ODBC Driver** appears.
9. In the window **Aster ODBC Driver**, select **Finish**.
10. In the DSN Setup form that appears, enter the following values and select **OK**:

Field	Value
Data Source	Name of data source to use in tpt script
Server	IP address of Aster queen
Port	2406
Database	Aster Database
Username	User name
Password	User password
MaxLenVarchar	Default length of unbounded VARCHAR data item

11. Select **OK**.

Writing the tpt Script

- Write the tpt script by substituting values for variables in the following script:

```

DEFINE JOB PRODUCT_SOURCE_LOAD

DESCRIPTION 'LOAD PRODUCT DEFINITION TABLE'
(
  DEFINE SCHEMA PRODUCT_SOURCE_SCHEMA
  DESCRIPTION 'PRODUCT INFORMATION SCHEMA'
  (
    TD_compatible_table_definition

    STEP STEP_CREATE_DDL
    (
      APPLY
      ('DROP TABLE '||@TargetTable||' ');
      ('CREATE MULTiset TABLE
' ||@TargetTable||' (TD_compatible_table_definition;)
      TO OPERATOR ($DDL() [1]);
    );
    Step Insert_Tables
    (
      APPLY
      ('Ins '||@TargetTable||' (
        :column_name_1
        ,:column_name_2
        [..., :column_name_k]
      );

```



```

)
TO OPERATOR ($LOAD()[1])

SELECT * FROM OPERATOR ($ODBC(PRODUCT_SOURCE_SCHEMA)[1]);
);
);

```

Writing the JobVariablesFile

- Write the JobVariablesFile by substituting values for variables in the following script:

```

DDLTDpId           = 'td_host_name_or_ip'
,DDLUserName       = 'td_user'
,DDLUserPassword   = 'td_user_password'
,DDLErrorList      = ['3807']
,DDLPrivateLogName = 'DDL001S1'
,TargetTable       = 'td_table_name'
,ODBCPrivateLogName = 'ODB039P1'
,ODBCDSNName       = 'Data_Source_Name_specified_in_DSN_Setup_form'
,TruncateData      = {'Y' | 'N'}
,ODBCUserName      = 'aster_user'
,ODBCUserPassword  = 'aster_user_password'
,LOADPrivateLogName = 'ODB039C1'
,LOADTDPID         = 'td_host_name_or_ip'
,LOADUserName      = 'td_user'
,LOADUserPassword  = 'td_user_password'
,SelectStmt        = 'SELECT * FROM aster_table_name;'
,LOADTargetTable   = 'td_table_name'

```

For TruncateData, 'Y' trims unused space. The default is 'N'. Specify 'Y' when the MaxLenVarchar field of the DSN Setup form (in [Setting Up the Aster Driver on the Client](#)) exceeds the maximum VARCHAR length specified in *TD_compatible_table_definition* in the tpt script; otherwise, ODBC cannot load the script.

Running the tpt Script

You are on Advanced SQL Engine, where a folder contains the tpt script and JobVariablesFile that you wrote.

1. Open the command prompt.
2. Go to the folder where the tpt script and JobVariablesFile are.
3. Run the tpt script with this command:

```
tbuild -f tptfile -v JobVariablesFile -j jobid
```

where *tptfile* and *JobVariablesFile* are the names of the tpt script and JobVariablesFile that you wrote and *jobid* is the name you are giving to this tbuild job.

Example: Loading Aster Table to Advanced SQL Engine Using ODBC

This example shows the code for the following:

- Creating and populating a table on Aster Database
- Creating a Advanced SQL Engine user to own the table
- A tpt script
- A JobVariablesFile
- Running the tpt script

Code for Creating and Populating a Table on Aster Database

This code creates and populates a training table, `aster_nb_trainerSC`, and then uses the training table and Naive Bayes Classifier function to create a model table, `aster_nb_modelSC`.

```
/* Create training table */

DROP TABLE IF EXISTS aster_nb_trainerSC;

CREATE TABLE aster_nb_trainerSC (
  id      INT,
  year    INT,
  color   VARCHAR (100),
  type    VARCHAR (100),
  origin  VARCHAR (100),
  stolen  VARCHAR (100),
  PARTITION KEY (id)
);

/* Populate training table */

INSERT INTO aster_nb_trainerSC VALUES
  (1,3,'red','sports','domestic','Yes'),
  (2,9,'red','sports','domestic','No'),
  (3,1,'red','sports','domestic','Yes'),
  (4,8,'yellow','sports','domestic','No'),
  (5,2,'yellow','sports','imported','Yes');
```

```

/* Create model table from training table */

DROP TABLE IF EXISTS aster_nb_modelSC;

CREATE TABLE aster_nb_modelSC distribute by hash(class_nb) AS (
  SELECT * FROM naiveBayesReduce (
    ON (
      SELECT * FROM naiveBayesMap (
        ON aster_nb_trainerSC
        Response ('stolen')
        NumericInputs ('year')
        CategoricalInputs ('color','origin','type')
      )
    ) PARTITION BY class_nb
  )
);

```

Code for Creating an Advanced SQL Engine User

```

CREATE USER sample_user AS
  PASSWORD = sample_user
  PERM = 10e6*(HASHAMP()+1);

GRANT ALL ON dbc TO sample_user;

```

tpt Script, astermodel.tpt

```

DEFINE JOB PRODUCT_SOURCE_LOAD
DESCRIPTION 'LOAD PRODUCT DEFINITION TABLE'
(
  DEFINE SCHEMA PRODUCT_SOURCE_SCHEMA
  DESCRIPTION 'PRODUCT INFORMATION SCHEMA'
  (
    class_nb VARCHAR(128),
    variable_nb VARCHAR(128),
    type_nb VARCHAR(128),
    category VARCHAR(32),
    cnt BIGINT,
    sum_nb FLOAT,
    sum_sq FLOAT,
    total_cnt BIGINT
  );

```

```

STEP STEP_CREATE_DDL
(
  APPLY
    ('DROP TABLE '||@TargetTable||';'),
    ('CREATE MULTISET TABLE '||@TargetTable||'(
      class_nb VARCHAR(128),
      variable_nb VARCHAR(128),
      type_nb VARCHAR(128),
      category VARCHAR(32),
      cnt BIGINT,
      sum_nb FLOAT,
      sum_sq FLOAT,
      total_cnt BIGINT) NO PRIMARY INDEX;
    ')
  TO OPERATOR ($DDL())[1]);
);
Step Insert Tables
(
  APPLY
    ('Ins '||@TargetTable||'(
      : "class_nb"
      ,: "variable_nb"
      ,: "type_nb"
      ,: "category"
      ,: "cnt"
      ,: "sum_nb"
      ,: "sum_sq"
      ,: "total_cnt");'
    )
  TO OPERATOR ($LOAD())[1])

  SELECT * FROM OPERATOR ($ODBC(PRODUCT_SOURCE_SCHEMA)[1]);
);
);

```

JobVariablesFile, attr.txt

```

DDLtdpid = 'td_host_name_or_ip'
,DDLUserName = 'alice'
,DDLUserPassword = 'alice'
,DDLErrorList = '[3807]'
,DDLPrivateLogName = 'DDL001S1'
,TargetTable = 'td_nb_modelsc'
,ODBCPrivateLogName = 'ODBC039P1'

```

```
,ODBCDSNName = 'shruti'
,TruncateData = 'Y'
,ODBCUserName = 'db_superuser'
,ODBCUserPassword = 'db_superuser'
,LOADPrivateLogName = 'ODBC039P1'
,LOADTDPID = 'td_host_name_or_ip'
,LOADUserName = 'alice'
,LOADUserPassword = 'alice'
,SelectStmt = 'SELECT * FROM aster_nb_modelsc;'
,LOADTargetTable = 'td_nb_modelsc'
```

Command for Running the tpt Script

```
tbuild -f aster_model.tpt -v attr.txt -j urr1
```

Aster Model Table Schemas

Forest_Predict Model Table Schema

```
CREATE FACT TABLE public.aster_fp_admissions_clsmodel
(
  worker_ip VARCHAR,
  task_index INTEGER,
  tree_num INTEGER,
  tree VARCHAR
)
DISTRIBUTE BY HASH (task_index)
STORAGE ROW;
```

GLMPredict Model Table Schema

```
CREATE ANALYTIC FACT TABLE public.glm_housing_model
(
  attribute INTEGER,
  predictor VARCHAR(1024),
  category VARCHAR(1024),
  estimate DOUBLE PRECISION,
  std_err DOUBLE PRECISION,
  { t_score | z_score } DOUBLE PRECISION,
  p_value DOUBLE PRECISION,
  significance VARCHAR(50),
  family VARCHAR(20)
)
```

```
DISTRIBUTE BY HASH (attribute)
STORAGE ROW;
```

The model table has the column `t_score` if created with Family ('GAUSSIAN'), otherwise it has the column `z_score`.

NaiveBayesPredict Model Table Schema

```
CREATE ANALYTIC FACT TABLE public.aster_nb_modelsc
(
  class_nb VARCHAR(128),
  variable_nb VARCHAR(128),
  type_nb VARCHAR(128),
  category VARCHAR(32),
  cnt BIGINT,
  sum_nb DOUBLE PRECISION,
  sum_sq DOUBLE PRECISION,
  total_cnt BIGINT
)
DISTRIBUTE BY HASH (class_nb)
STORAGE ROW;
```

NaiveBayesTextClassifierPredict Model Table Schema

```
CREATE DIMENSION TABLE public.nbtcp_spam_multinomialmodel
(
  token VARCHAR,
  category VARCHAR,
  prob DOUBLE PRECISION
)
DISTRIBUTE BY REPLICATION
STORAGE ROW;
```

Single_Tree_Predict Model Table Schema

```
CREATE DIMENSION TABLE public.glass_model
(
  node_id BIGINT,
  node_size BIGINT,
  node_gini_p DOUBLE PRECISION,
  node_entropy DOUBLE PRECISION,
  node_chisq_pv DOUBLE PRECISION,
  node_label VARCHAR(512),
  node_majorvotes BIGINT,
```

```

split_value DOUBLE PRECISION,
split_gini_p DOUBLE PRECISION,
split_entropy DOUBLE PRECISION,
split_chisq_pv DOUBLE PRECISION,
left_id BIGINT,
left_size BIGINT,
left_label VARCHAR(512),
left_majorvotes BIGINT,
right_id BIGINT,
right_size BIGINT,
right_label VARCHAR(512),
right_majorvotes BIGINT,
left_bucket VARCHAR(512),
right_bucket VARCHAR(512),
attribute VARCHAR(512)
)
DISTRIBUTE BY REPLICATION
STORAGE ROW;

```

SparseSVMPredict Model Table Schema

```

CREATE FACT TABLE public.aster_svm_iris_model_default
(
  classid INTEGER,
  weights BYTEA
)
DISTRIBUTE BY HASH (classid)
STORAGE ROW;

```

Advanced SQL Engine Analytic Functions

Antiselect (SQL Engine)

Antiselect returns all columns *except* those specified in the Exclude syntax element.

Note:

- This function requires the UTF8 client character set for UNICODE data.

Antiselect Syntax

```
SELECT * FROM Antiselect (
  ON { table | view | (query) }
  USING
  Exclude ({ 'exclude_column' | exclude_column_range }[,...])
) AS alias;
```

Antiselect Syntax Elements

Exclude

Specify the names of the input table columns to exclude from the output table. Column names must be valid object names, which are defined in *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

The *exclude_column* is a column name. This is the syntax of *exclude_column_range*:

```
'start_column:end_column' [, '-exclude_in-range_column' ]
```

The range includes its endpoints.

The *start_column* and *end_column* can be:

- Column names (for example, 'column1:column2')
- Column names must contain only letters in the English alphabet, digits, and special characters. If a column name includes any special characters, surround the column name with double quotation marks. For example, if the column name is a*b, specify it as "a*b". A column name cannot contain a double quotation mark.
- Nonnegative integers that represent the indexes of columns in the table (for example, '[0:4]')

The first column has index 0; therefore, '[0:4]' specifies the first five columns in the table.

- Empty. For example:
 - '[:4]' specifies all columns up to and including the column with index 4.
 - '[4:]' specifies the column with index 4 and all columns after it.
 - '[:]' specifies all columns in the table.

The *exclude_in-range_column* is a column in the specified range, represented by either its name or its index (for example, '[0:99]', '-[50]', '-column10' specifies the columns with indexes 0 through 99, except the column with index 50 and column10).

Column ranges cannot overlap, and cannot include any specified *exclude_column*.

Antiselect Input

The input table can have any schema.

Antiselect Output

The output table has all input table columns except those specified by the Exclude syntax element.

Antiselect Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

Antiselect Example: No Column Ranges

Input

The input table, antiselect_test, is a sample set of sales data containing 13 columns.

antiselect_test

sno	id	orderdate	priority	qty	sales	disct	dmode	custname	province	region	cus
1	3	2010-10-13 00:00:00	Low	6	261. 54	0.04	Regular Air	Muhammed MacIntyre	Nunavut	Nunavut	Sma Bus
49	293	2012-10-01 00:00:00	High	49	10123	0.07	Delivery Truck	Barry French	Nunavut	Nunavut	Con
50	293	2012-10-01 00:00:00	High	27	244. 57	0.01	Regular Air	Barry French	Nunavut	Nunavut	Con

sno	id	orderdate	priority	qty	sales	disct	dmode	custname	province	region	cus
80	483	2011-07-10 00:00:00	High	30	4965.76	0.08	Regular Air	Clay Rozendal	Nunavut	Nunavut	Corp
85	515	2010-08-28 00:00:00	Not specified	19	394.27	0.08	Regular Air	Carlos Soltero	Nunavut	Nunavut	Con
86	515	2010-08-28 00:00:00	Not specified	21	146.69	0.05	Regular Air	Carlos Soltero	Nunavut	Nunavut	Con
97	613	2011-06-17 00:00:00	High	12	93.54	0.03	Regular Air	Carl Jackson	Nunavut	Nunavut	Corp

SQL Call

```
SELECT * FROM Antiselect (
  ON antiselect_test
  USING
  Exclude ('id', 'orderdate', 'disct', 'province', 'custsegment')
) AS dt ORDER BY 1, 4;
```

Output

sno	priority	qty	sales	dmode	custname	region	prodcats
1	Low	6	2.615400000000000E002	Regular Air	Muhammed MacIntyre	Nunavut	Office Supplies
49	High	49	1.012300000000000E004	Delivery Truck	Barry French	Nunavut	Office Supplies
50	High	27	2.445700000000000E002	Regular Air	Barry French	Nunavut	Office Supplies
80	High	30	4.965760000000000E003	Regular Air	Clay Rozendal	Nunavut	Technology
85	Not specified	19	3.942700000000000E002	Regular Air	Carlos Soltero	Nunavut	Office Supplies
86	Not specified	21	1.466900000000000E002	Regular Air	Carlos Soltero	Nunavut	Furniture
97	High	12	9.354000000000000E001	Regular Air	Carl Jackson	Nunavut	Office Supplies

Antiselect Example: Column Range

Input

The input table is antiselect_test, as in [Antiselect Example: No Column Ranges](#).

SQL Call

```
SELECT * FROM Antiselect (
  ON antiselect_test
  USING
  Exclude ('id', '[2:3]', 'custname:prodcat')
) AS dt ORDER BY 1, 4;
```

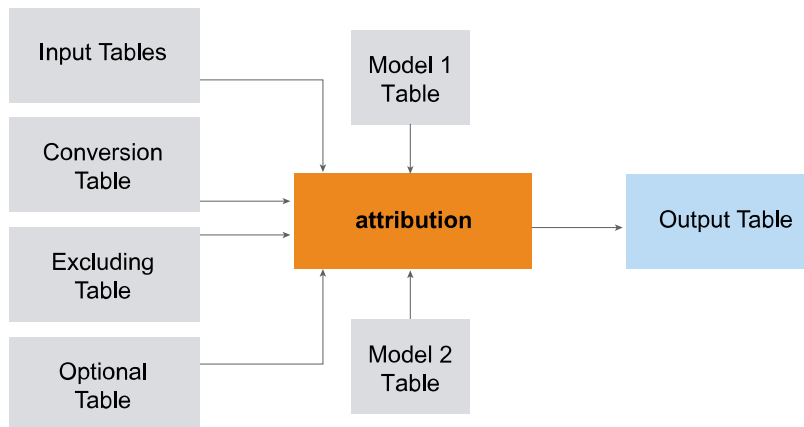
Output

sno	qty	sales	disct	dmode
1	6	2.61540000000000E 002	0.04	Regular Air
49	49	1.01230000000000E 004	0.07	Delivery Truck
50	27	2.44570000000000E 002	0.01	Regular Air
80	30	4.96576000000000E 003	0.08	Regular Air
85	19	3.94270000000000E 002	0.08	Regular Air
86	21	1.46690000000000E 002	0.05	Regular Air
97	12	9.35400000000000E 001	0.03	Regular Air

Attribution (SQL Engine)

The Attribution function is used in web page analysis, where it lets companies assign weights to pages before certain events, such as buying a product.

The function takes data and parameters from multiple tables and outputs attributions.



ML Engine function Attribution_MLE has two versions:

- Multiple-input: Accepts one or more input tables and gets many parameters from other dimension tables.
- Single-input: Accepts only one input table and gets all parameters from syntax elements.

Advanced SQL Engine Attribution function corresponds to the multiple-input version. Unlike Attribution_MLE, Attribution does not support Unicode.

A query that runs longer than 3 seconds before displaying output indicates that syntax elements supplied to the function are incorrect.

Attribution Syntax

```

SELECT * FROM Attribution (
  ON { table | view | (query) } [ AS InputTable1 ]
    PARTITION BY user_id
    ORDER BY times_column
  [ ON { table | view | (query) } [ AS InputTable2 ]
    PARTITION BY user_id
    ORDER BY time_column [,...] ]
  ON conversion_event_table AS ConversionEventTable DIMENSION
  [ ON excluding_event_table AS ExcludedEventTable DIMENSION ]
  [ ON optional_event_table AS OptionalEventTable DIMENSION ]
  ON model1_table AS FirstModelTable DIMENSION
  [ ON model2_table AS SecondModelTable DIMENSION ]
  USING
    EventColumn ('event_column')
    TimeColumn ('time_column')
    WindowSize ({'rows:K' | 'seconds:K' | 'rows:K&seconds:K2'})
) AS alias ORDER BY user_id,time_stamp;
  
```

Attribution Syntax Elements

EventColumn

Specify the name of the input column that contains the clickstream events.

TimeColumn

Specify the name of the input column that contains the timestamps of the clickstream events.

WindowSize

Specify how to determine the maximum window size for the attribution calculation:

Option	Description
rows : K	Assign attributions to at most K events before conversion event, excluding events of types specified in ExcludedEventTable.
seconds : K	Assign attributions only to rows not more than K seconds before conversion event.
rows : K &seconds : $K2$	Apply both constraints and comply with stricter one.

Attribution Input

Required

Table	Description
Input tables (maximum of five)	Contain clickstream data for computing attributions.
ConversionEventTable	Contains conversion events.
FirstModelTable	Defines type and distributions of first model.

Optional

Table	Description
ExcludedEventTable	Contains events to exclude from attribution.
OptionalEventTable	Contains optional events.
SecondModelTable	Defines type and distributions of second model.

Input Table Schema

Column	Data Type	Description
<i>userid_column</i>	INTEGER or VARCHAR	User identifier.
<i>event_column</i>	INTEGER or VARCHAR	Event from clickstream.
<i>time_column</i>	INTEGER, SMALLINT, BIGINT, TIMESTAMP, or TIME	Event timestamp.

ConversionEventTable Schema

Column	Data Type	Description
<i>conversion_event</i>	VARCHAR	Conversion event value (string or integer).

FirstModelTable and SecondModelTable Schema

Column	Data Type	Description
id	INTEGER	Row identifier. Rows are numbered 0, 1, 2, and so on.
model	VARCHAR	Row 0: Model type. Row 1, ..., <i>n</i> : Distribution model definition. SIMPLE model: Model table has single row that specifies model type and parameters. Other model types: <i>n</i> is number of rows or events included in model. For model type and specification definitions, see Model Specification .

ExcludedEventTable Schema

Column	Data Type	Description
<i>excluding_event</i>	VARCHAR	Excluded event (string or integer). Cannot be a conversion event.

OptionalEventTable Schema

Column	Data Type	Description
<i>optional_event</i>	VARCHAR	Optional event (string or integer). Cannot be a conversion or excluded event. Function attributes conversion event to optional event only if it cannot attribute it to regular event.

Model Specification

Model Types and Specification Definitions

Row 0: Model Type	Row 1, ..., n: Distribution Model Specification	Additional Information
SIMPLE	<i>MODEL: PARAMETERS</i>	Distribution model for all events. For MODEL and PARAMETER definitions, see following table.
EVENT_ REGULAR	<i>EVENT:WEIGHT: MODEL: PARAMETERS</i>	<p>Distribution model for a regular event.</p> <p>EVENT cannot be a conversion, excluded, or optional event.</p> <p>For MODEL and PARAMETER definitions, see following table.</p> <p>Sum of WEIGHT values must be 1.0.</p> <p>For example, suppose that model table has these specifications: email:0.19:LAST_CLICK:NA impression:0.81:UNIFORM:NA</p> <p>Within WindowSize of a conversion event, 19% of conversion event is attributed to last email event and 81% is attributed uniformly to all impression events.</p>
EVENT_ OPTIONAL	<i>EVENT:WEIGHT: MODEL: PARAMETERS</i>	<p>Distribution model for an optional event.</p> <p>EVENT must be in optional event table.</p> <p>For MODEL and PARAMETER definitions, see following table.</p> <p>Sum of WEIGHT values must be 1.0.</p>
SEGMENT_ ROWS	<i>K_i:WEIGHT: MODEL: PARAMETERS</i>	<p>Distribution model by row. Sum of K_i values must be value K specified by 'rows:K' in WindowSize syntax element.</p> <p>Function considers rows from most to least recent. For example, suppose that function call has these syntax elements: WindowSize ('rows:10') Model1 ('SEGMENT_ROWS', '3:0.5:UNIFORM:NA', '4:0.3:LAST_CLICK:NA', '3:0.2:FIRST_CLICK:NA')</p> <p>Attribution for a conversion event is divided among attributable events in 10 rows immediately preceding conversion event. If conversion event is in row 11, first model specification applies to rows 10, 9, and 8; second applies to rows 7, 6, 5, and 4; and third applies to rows 3, 2, and 1.</p> <p>Half attribution (5/10) is uniformly divided among rows 10, 9, and 8; 3/10 to last click in rows 7, 6, 5, and 4 (that is, in row 7), and 2/10 to first click in rows 3, 2, and 1 (that is, in row 1).</p>
SEGMENT_ SECONDS	<i>K_i:WEIGHT: MODEL: PARAMETERS</i>	<p>Distribution model by time in seconds. Sum of K_i values must be value K specified by 'seconds:K' in WindowSize syntax element.</p> <p>Function considers rows from most to least recent. For example, suppose that function call has these syntax elements: WindowSize ('seconds:20') Model1 ('SEGMENT_SECONDS', '6:0.5:UNIFORM:NA', '8:0.3:LAST_CLICK:NA', '6:0.2:FIRST_CLICK:NA')</p>

Row 0: Model Type	Row 1, ..., n: Distribution Model Specification	Additional Information
		<p>Attribution for a conversion event is divided among attributable events in 20 seconds immediately preceding conversion event. If conversion event is at second 21, first model specification applies to seconds 20-15 (counting backward); second applies to seconds 14-7; and third applies to seconds 6-1.</p> <p>Half attribution (5/10) is uniformly divided among seconds 20-15; 3/10 to last click in seconds 14-7, and 2/10 to first click in seconds 6-1.</p>

MODEL Values and Corresponding PARAMETER Values

MODEL values are case-sensitive. Attributable events are those whose types are not specified in excluding events table.

MODEL	Description	PARAMETERS
'LAST_CLICK'	Conversion event is attributed entirely to most recent attributable event.	'NA'
'FIRST_CLICK'	Conversion event is attributed entirely to first attributable event.	'NA'
'UNIFORM'	Conversion event is attributed uniformly to preceding attributable events.	'NA'
'EXPONENTIAL'	Conversion event is attributed exponentially to preceding attributable events (the more recent the event, the higher the attribution).	' <i>alpha,type</i> ' where <i>alpha</i> is a decay factor in range (0, 1) and <i>type</i> is ROW, MILLISECOND, SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR. When <i>alpha</i> is in range (0, 1), sum of series $w_i = (1 - \alpha) * \alpha^i$ is 1. Function uses w_i as exponential weights.
'WEIGHTED'	Conversion event is attributed to preceding attributable events with weights specified by PARAMETERS. SEGMENT_SECONDS (when you specify 'rows:K&seconds:K' in WindowSize syntax element)	You can specify any number of weights. If there are more attributable events than weights, extra (least recent) events are assigned zero weight. If there are more weights than attributable events, then function renormalizes weights.

Allowed FirstModelTable/SecondModelTable Combinations

FirstModelTable Type	SecondModelTable Type
SIMPLE	Not allowed
EVENT_REGULAR	
EVENT_REGULAR	EVENT_OPTIONAL (when you specify optional events table)

FirstModelTable Type	SecondModelTable Type
SEGMENT_ROWS	SEGMENT_SECONDS (when you specify 'rows:K&seconds:K' in WindowSize syntax element)
SEGMENT_ROWS	
SEGMENT_SECONDS	Not allowed

Attribution Output

Attribution Output Table Schema

Column	Data Type	Description
user_id	INTEGER or VARCHAR	User identifier from input table.
event	VARCHAR	Clickstream event from input table.
time_stamp	TIMESTAMP	Event timestamp from input table.
attribution	DOUBLE PRECISION	Fraction of attribution for conversion event that is attributed to this event.
time_to_conversion	INTEGER	Elapsed time between attributable event and conversion event.

Attribution Example: Model Assigns Weights to Events and Channels

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

Event Type Channels

This example uses models to assign attribution weights to these events and channels.

Event Type	Channels
conversion	SocialNetwork, PaidSearch
excluding	Email
optional	Direct, Referral, OrganicSearch

Input**InputTable1: attribution_sample_table1**

user_id	event	time_stamp
1	impression	2001-09-27 23:00:01
1	impression	2001-09-27 23:00:05
1	Email	2001-09-27 23:00:15
2	impression	2001-09-27 23:00:31
2	impression	2001-09-27 23:00:51

InputTable2: attribution_sample_table2

user_id	event	time_stamp
1	impression	2001-09-27 23:00:19
1	SocialNetwork	2001-09-27 23:00:20
1	Direct	2001-09-27 23:00:21
1	Referral	2001-09-27 23:00:22
1	PaidSearch	2001-09-27 23:00:23
2	impression	2001-09-27 23:00:29
2	impression	2001-09-27 23:00:31
2	impression	2001-09-27 23:00:33
2	impression	2001-09-27 23:00:36
2	impression	2001-09-27 23:00:38

ConversionEventTable: conversion_event_table

conversion_events
PaidSearch
SocialNetwork

ExcludedEventTable: excluding_event_table

excluding_events
Email

OptionalEventTable: optional_event_table

optional_events
Direct
OrganicSearch
Referral

The following two model tables apply the distribution models by rows and by seconds, respectively.

FirstModelTable: model1_table

id	model
0	SEGMENT_ROWS
1	3:0.5:EXPONENTIAL:0.5,SECOND
2	4:0.3:WEIGHTED:0.4,0.3,0.2,0.1
3	3:0.2:FIRST_CLICK:NA

SecondModelTable: model2_table

id	model
0	SEGMENT_SECONDS
1	6:0.5:UNIFORM:NA
2	8:0.3:LAST_CLICK:NA
3	6:0.2:FIRST_CLICK:NA

SQL Call

```
SELECT * FROM Attribution (
  ON attribution_sample_table1 AS InputTable1
    PARTITION BY user_id ORDER BY time_stamp
  ON attribution_sample_table2 AS InputTable2
    PARTITION BY user_id ORDER BY time_stamp
  ON conversion_event_table AS ConversionEventTable DIMENSION
  ON excluding_event_table AS ExcludedEventTable DIMENSION
  ON optional_event_table AS OptionalEventTable DIMENSION
  ON model1_table AS FirstModelTable DIMENSION
  ON model2_table AS SecondModelTable DIMENSION
  USING
    EventColumn ('event')
    TimeColumn ('time_stamp')
```

```
WindowSize ('rows:10&seconds:20')
) AS dt ORDER BY user_id, time_stamp;
```

Output

user_id	event	time_stamp	attribution	time_to_conversion
1	impression	2001-09-27 23:00:01	0.285714	-19
1	impression	2001-09-27 23:00:05	0	?
1	impression	2001-09-27 23:00:19	0.714286	-1
1	SocialNetwork	2001-09-27 23:00:20	?	?
1	Direct	2001-09-27 23:00:21	0.5	-2
1	Referral	2001-09-27 23:00:22	0.5	-1
1	PaidSearch	2001-09-27 23:00:23	?	?

DecisionForestPredict (SQL Engine)

Note:

This *namePredict* function uses the model output by ML Engine *name* function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

DecisionForestPredict outputs the probability that each observation is in the *predicted* class. To use DecisionForestPredict output as input to ML Engine ROC function, you must first transform it to show the probability that each observation is in the *positive* class. One way to do this is to change the probability to (1- current probability) when the predicted class is negative.

The prediction algorithm compares floating-point numbers. Due to possible inherent data type differences between ML Engine and Advanced SQL Engine executions, predictions can differ. Before calling the function, compute the relative error, using this formula:

$$\text{relative_error} = (\text{abs}(\text{mle_prediction} - \text{td_prediction}) / \text{mle_prediction}) * 100$$

where *mle_prediction* is ML Engine prediction value and *td_prediction* is Advanced SQL Engine prediction value. Errors (*e*) follow Gaussian law; $0 < e < 3\%$ is a negligible difference, with high confidence.

DecisionForestPredict Syntax

```
SELECT * FROM DecisionForestPredict (
  ON { table | view | (query) } PARTITION BY ANY
  ON { table | view | (query) } AS Model DIMENSION
  USING
  IDColumn ('id_column')
  [ NumericInputs ({ 'numeric_input_column' | numeric_input_column_range }[,...]) ]
  [ CategoricalInputs ({ 'categorical_input_column' | categorical_input_column_range
    }[,...]) ]
  [ Detailed ({ 'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Responses ('response' [,...]) ]
  [ OutputProb ({ 'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related Information:

[Column Specification Syntax Elements](#)

DecisionForestPredict Syntax Elements

IDColumn

Specify the column that contains a unique identifier for each test point in the test set.

NumericInputs

[Optional] Specify the names of the columns that contain the numeric predictor variables.

Default behavior: The function gets these variables from the model output by DecisionForest only if you omit both NumericInputs and CategoricalInputs. If you specify this syntax element, you must specify it exactly as you specified it in the DecisionForest call that created the model.

CategoricalInputs

[Optional] Specify the names of the columns that contain the categorical predictor variables.

Default behavior: The function gets these variables from the model output by DecisionForest only if you omit both NumericInputs and CategoricalInputs. If you specify this syntax element, you must specify it exactly as you specified it in the DecisionForest call that created the model.

Detailed

[Optional] Specify whether to output detailed information about the forest trees; that is, the decision tree and the specific tree information, including task index and tree index for each tree.

Default: 'false'

Responses

[Optional] Specify the classes for which to output probabilities.

Note:

Responses works only with a classification model.

Default behavior: Output only the probability of the predicted class.

OutputProb

[Required to be 'true' with Responses, optional otherwise.] Specify whether to output the probability for each response. If you omit Responses, the function outputs only the probability of the predicted class.

Note:

OutputProb works only with a classification model.

Default: 'false'

Accumulate

[Optional] Specify the names of the input columns to copy to the output table.

DecisionForestPredict Input

Table	Description
Input	Contains test data.
Model	Has same schema as OutputTable of ML Engine DecisionForest function.

Input Table Schema

Column	Data Type	Description
<i>id_column</i>	Any	Unique test point identifier. Cannot be NULL.

Column	Data Type	Description
<i>numeric_column</i>	NUMERIC, INTEGER, BIGINT, or DOUBLE PRECISION	Numeric predictor variable. Cannot be NULL.
<i>category_column</i>	INTEGER, BIGINT, or VARCHAR	Categorical predictor variable. Cannot be NULL.
<i>accumulate_column</i>	Any	Column to copy to output table.

Model Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Column	Data Type	Description
worker_ip	VARCHAR	IP address of worker that produced decision tree.
task_index	INTEGER, BIGINT, or SMALLINT	Identifier of worker that produced decision tree.
tree_num	INTEGER, BIGINT, or SMALLINT	Decision tree identifier.
tree	VARCHAR, CLOB, or JSON	JSON representation of decision tree.

DecisionForestPredict Output

Output Table Schema

The table has a set of predictions for each test point.

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	Column copied from input table.
<i>id_column</i>	Same as in input table	Column copied from input table. Unique row identifier.
<i>prediction</i>	VARCHAR	Predicted test point value, predicted by model.
confidence_lower	DOUBLE PRECISION	[Appears with OutputProb ('false').] Lower bound of confidence interval. For classification tree, confidence_lower and confidence_upper have the same value, which is the probability of the predicted class.
confidence_upper	DOUBLE PRECISION	[Appears with OutputProb ('false').] Upper bound of confidence interval. For classification tree, confidence_lower and confidence_upper have the same value, which is the probability of the predicted class.

Column	Data Type	Description
tree_num	VARCHAR	Either the concatenation of task_index and tree_num from the model table, to show which tree created the prediction, or 'final' to show the overall prediction. This column appears only if you specify Detailed ('true').
prob	DOUBLE PRECISION	[Column appears only with OutputProb ('true') and without Responses syntax element.] Probability that observation belongs to class prediction.
prob_response	DOUBLE PRECISION	[Column appears only with OutputProb ('true') and Responses syntax element and Responses syntax element. Appears once for each specified response.] Probability that observation belongs to category response.

DecisionForestPredict Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

DecisionForestPredict Example: Specify Column Names

Input

- Input table: housing_test, which has 54 observations of 14 variables
- Model: rft_model, output by "DecisionForest Example: TreeType ('classification') and OutOfBag ('false')" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003

Input Table Column Descriptions

Column	Description
sn	Sale number (unique identifier of observation)
price	Sale price in U. S. dollars (numeric)
lotsize	Lot size in square feet (numeric)
bedrooms	Number of bedrooms (numeric)
bathrms	Number of full bathrooms (numeric)
stories	Number of stories, excluding basement (numeric)
driveway	Whether the house has a driveway—yes or no (categorical)
recroom	Whether the house has a recreation room—yes or no (categorical)
fullbase	Whether the house has a full finished basement—yes or no (categorical)

Column	Description
gashw	Whether the house uses gas to heat water—yes or no (categorical)
airco	Whether the house has central air conditioning—yes or no (categorical)
garagepl	Number of garage places (numeric)
prefarea	Whether the house is in a preferred neighborhood—yes or no (categorical)
homestyle	Style of house (response variable)

housing_test

sn	price	lotsize	bedrooms	bathrms	stories	driveway	recroom	fullbase	gashw	airco
13	27000	1700	3	1	2	yes	no	no	no	no
16	37900	3185	2	1	1	yes	no	no	no	yes
25	42000	4960	2	1	1	yes	no	no	no	no
38	67000	5170	3	1	4	yes	no	no	no	yes
53	68000	9166	2	1	1	yes	no	yes	no	yes
104	132000	3500	4	2	2	yes	no	no	yes	no
111	43000	5076	3	1	1	no	no	no	no	no
117	93000	3760	3	1	2	yes	no	no	yes	no
132	44500	3850	3	1	2	yes	no	no	no	no
140	43000	3750	3	1	2	yes	no	no	no	no
142	40000	2650	3	1	2	yes	no	yes	no	no
157	60000	2953	3	1	2	yes	no	yes	no	yes
...

rft_model

worker_ip	task_index	tree_num	CAST(tree AS VARCHAR(50))
xx.xx.xx.xx	0	0	{"responseCounts_":{"Eclectic":148,"bungalow":30,"
xx.xx.xx.xx	0	1	{"responseCounts_":{"Eclectic":158,"bungalow":26,"
xx.xx.xx.xx	0	2	{"responseCounts_":{"Eclectic":120,"bungalow":38,"
xx.xx.xx.xx	0	3	{"responseCounts_":{"Eclectic":166,"bungalow":29,"
xx.xx.xx.xx	0	4	{"responseCounts_":{"Eclectic":138,"bungalow":32,"
xx.xx.xx.xx	0	5	{"responseCounts_":{"Eclectic":158,"bungalow":34,"

worker_ip	task_index	tree_num	CAST(tree AS VARCHAR(50))
xx.xx.xx.xx	0	6	{"responseCounts_":{"Eclectic":168,"bungalow":32,"
xx.xx.xx.xx	0	7	{"responseCounts_":{"Eclectic":145,"bungalow":40,"
xx.xx.xx.xx	0	8	{"responseCounts_":{"Eclectic":150,"bungalow":34,"
xx.xx.xx.xx	0	9	{"responseCounts_":{"Eclectic":156,"bungalow":42,"
xx.xx.xx.xx	0	10	{"responseCounts_":{"Eclectic":148,"bungalow":18,"
xx.xx.xx.xx	0	11	{"responseCounts_":{"Eclectic":147,"bungalow":20,"
xx.xx.xx.xx	0	12	{"responseCounts_":{"Eclectic":150,"bungalow":31,"
xx.xx.xx.xx	0	13	{"responseCounts_":{"Eclectic":135,"bungalow":32,"
xx.xx.xx.xx	0	14	{"responseCounts_":{"Eclectic":139,"bungalow":24,"
xx.xx.xx.xx	0	15	{"responseCounts_":{"Eclectic":146,"bungalow":27,"
xx.xx.xx.xx	0	16	{"responseCounts_":{"Eclectic":152,"bungalow":23,"
xx.xx.xx.xx	0	17	{"responseCounts_":{"Eclectic":135,"bungalow":23,"
xx.xx.xx.xx	0	18	{"responseCounts_":{"Eclectic":148,"bungalow":29,"
xx.xx.xx.xx	0	19	{"responseCounts_":{"Eclectic":166,"bungalow":33,"
xx.xx.xx.xx	0	20	{"responseCounts_":{"Eclectic":142,"bungalow":28,"
xx.xx.xx.xx	0	21	{"responseCounts_":{"Eclectic":172,"bungalow":27,"
xx.xx.xx.xx	0	22	{"responseCounts_":{"Eclectic":147,"bungalow":37,"
xx.xx.xx.xx	0	23	{"responseCounts_":{"Eclectic":158,"bungalow":31,"
xx.xx.xx.xx	0	24	{"responseCounts_":{"Eclectic":158,"bungalow":33,"
xx.xx.xx.xx	1	0	{"responseCounts_":{"Eclectic":140,"bungalow":44,"
xx.xx.xx.xx	1	1	{"responseCounts_":{"Eclectic":161,"bungalow":28,"
xx.xx.xx.xx	1	2	{"responseCounts_":{"Eclectic":131,"bungalow":25,"
xx.xx.xx.xx	1	3	{"responseCounts_":{"Eclectic":167,"bungalow":28,"
xx.xx.xx.xx	1	4	{"responseCounts_":{"Eclectic":150,"bungalow":19,"
xx.xx.xx.xx	1	5	{"responseCounts_":{"Eclectic":158,"bungalow":24,"
xx.xx.xx.xx	1	6	{"responseCounts_":{"Eclectic":177,"bungalow":32,"
xx.xx.xx.xx	1	7	{"responseCounts_":{"Eclectic":156,"bungalow":24,"
xx.xx.xx.xx	1	8	{"responseCounts_":{"Eclectic":156,"bungalow":37,"
xx.xx.xx.xx	1	9	{"responseCounts_":{"Eclectic":165,"bungalow":24,"

worker_ip	task_index	tree_num	CAST(tree AS VARCHAR(50))
xx.xx.xx.xx	1	10	{"responseCounts_":{"Eclectic":135,"bungalow":29,"
xx.xx.xx.xx	1	11	{"responseCounts_":{"Eclectic":140,"bungalow":20,"
xx.xx.xx.xx	1	12	{"responseCounts_":{"Eclectic":156,"bungalow":24,"
xx.xx.xx.xx	1	13	{"responseCounts_":{"Eclectic":147,"bungalow":34,"
xx.xx.xx.xx	1	14	{"responseCounts_":{"Eclectic":151,"bungalow":22,"
xx.xx.xx.xx	1	15	{"responseCounts_":{"Eclectic":161,"bungalow":18,"
xx.xx.xx.xx	1	16	{"responseCounts_":{"Eclectic":156,"bungalow":19,"
xx.xx.xx.xx	1	17	{"responseCounts_":{"Eclectic":126,"bungalow":29,"
xx.xx.xx.xx	1	18	{"responseCounts_":{"Eclectic":148,"bungalow":26,"
xx.xx.xx.xx	1	19	{"responseCounts_":{"Eclectic":177,"bungalow":21,"
xx.xx.xx.xx	1	20	{"responseCounts_":{"Eclectic":137,"bungalow":31,"
xx.xx.xx.xx	1	21	{"responseCounts_":{"Eclectic":171,"bungalow":28,"
xx.xx.xx.xx	1	22	{"responseCounts_":{"Eclectic":146,"bungalow":30,"
xx.xx.xx.xx	1	23	{"responseCounts_":{"Eclectic":149,"bungalow":21,"
xx.xx.xx.xx	1	24	{"responseCounts_":{"Eclectic":158,"bungalow":18,"

SQL Call

Use the Accumulate syntax element to pass the homestyle variable, to easily compare the actual and predicted response for each observation.

```
CREATE MULTISET TABLE rf_housing_predict AS (
  SELECT * FROM DecisionForestPredict (
    ON housing_test PARTITION BY ANY
    ON rft_model AS Model DIMENSION
    USING
    NumericInputs ('price ','lotsize ','bedrooms ','bathrms ',
                  'stories ','garagepl')
    CategoricalInputs ('driveway ','recroom ','fullbase ','gashw ',
                     'airco ','prefarea')
    IdColumn ('sn')
    Accumulate ('homestyle')
    Detailed ('false')
  ) AS dt
) WITH DATA;
```

Output

This query returns the following table:

```
SELECT * FROM rf_housing_predict ORDER BY 2;
```

homestyle	sn	prediction
confidence_lower	confidence_upper	
-----	-----	-----
classic	13	classic
8.88888888888889E-001	8.88888888888889E-001	
classic	16	classic
8.88888888888889E-001	8.88888888888889E-001	
classic	25	classic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	38	eclectic
7.77777777777778E-001	7.77777777777778E-001	
eclectic	53	eclectic
7.77777777777778E-001	7.77777777777778E-001	
bungalow	104	eclectic
7.77777777777778E-001	7.77777777777778E-001	
classic	111	classic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	117	eclectic
1.00000000000000E 000	1.00000000000000E 000	
classic	132	classic
8.88888888888889E-001	8.88888888888889E-001	
classic	140	classic
8.88888888888889E-001	8.88888888888889E-001	
classic	142	classic
8.88888888888889E-001	8.88888888888889E-001	
eclectic	157	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	161	eclectic
1.00000000000000E 000	1.00000000000000E 000	
bungalow	162	bungalow
5.55555555555556E-001	5.55555555555556E-001	
eclectic	176	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	177	eclectic
1.00000000000000E 000	1.00000000000000E 000	
classic	195	classic
1.00000000000000E 000	1.00000000000000E 000	

classic	198	classic
8.88888888888889E-001	8.88888888888889E-001	
eclectic	224	eclectic
1.00000000000000E 000	1.00000000000000E 000	
classic	234	classic
1.00000000000000E 000	1.00000000000000E 000	
classic	237	classic
8.88888888888889E-001	8.88888888888889E-001	
classic	239	classic
1.00000000000000E 000	1.00000000000000E 000	
classic	249	classic
1.00000000000000E 000	1.00000000000000E 000	
classic	251	classic
8.88888888888889E-001	8.88888888888889E-001	
eclectic	254	eclectic
8.88888888888889E-001	8.88888888888889E-001	
eclectic	255	eclectic
1.00000000000000E 000	1.00000000000000E 000	
classic	260	classic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	274	eclectic
8.88888888888889E-001	8.88888888888889E-001	
classic	294	classic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	301	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	306	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	317	eclectic
7.77777777777778E-001	7.77777777777778E-001	
bungalow	329	bungalow
8.88888888888889E-001	8.88888888888889E-001	
bungalow	339	bungalow
5.55555555555556E-001	5.55555555555556E-001	
eclectic	340	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	353	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	355	eclectic
8.88888888888889E-001	8.88888888888889E-001	
eclectic	364	eclectic
1.00000000000000E 000	1.00000000000000E 000	
bungalow	367	bungalow
7.77777777777778E-001	7.77777777777778E-001	

bungalow	377	bungalow
7.77777777777778E-001	7.77777777777778E-001	
eclectic	401	eclectic
8.88888888888889E-001	8.88888888888889E-001	
eclectic	403	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	408	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	411	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	440	eclectic
8.88888888888889E-001	8.88888888888889E-001	
eclectic	441	eclectic
1.00000000000000E 000	1.00000000000000E 000	
eclectic	443	eclectic
1.00000000000000E 000	1.00000000000000E 000	
classic	459	classic
8.88888888888889E-001	8.88888888888889E-001	
classic	463	classic
7.77777777777778E-001	7.77777777777778E-001	
eclectic	469	eclectic
7.77777777777778E-001	7.77777777777778E-001	
eclectic	472	eclectic
1.00000000000000E 000	1.00000000000000E 000	
bungalow	527	bungalow
7.77777777777778E-001	7.77777777777778E-001	
bungalow	530	eclectic
6.66666666666667E-001	6.66666666666667E-001	
eclectic	540	eclectic
8.88888888888889E-001	8.88888888888889E-001	

Prediction Accuracy

This query returns the prediction accuracy:

```
SELECT (SELECT count(sn) FROM rf_housing_predict
WHERE homestyle = prediction) / (SELECT count(sn)
FROM rf_housing_predict) AS PA;
```

pa

0.7777777777777778

DecisionForestPredict Example: Specify Column Range, OutputProb, Responses

Input

- Input table: housing_test_sample

sn	price	lotsize	bedrooms	bathrms				
stories	driveway	recroom	fullbase	gashw	airco	garagepl		
prefarea	homestyle							
---	-----	-----	-----	-----				
-----	-----	-----	-----	-----				
-----	-----							
329	1.154420000000000E 005	7.00000000000000E 003		3		2		
4	yes	no	no	no	yes	2	no	bungalow
224	7.85000000000000E 004	2.81700000000000E 003		4		2		
2	no	yes	yes	no	no	1	no	Eclectic
198	4.05000000000000E 004	4.35000000000000E 003		3		1		
2	no	no	no	yes	no	1	no	Classic
162	1.30000000000000E 005	6.00000000000000E 003		4		1		
2	yes	no	yes	no	no	2	no	bungalow
339	1.41000000000000E 005	8.10000000000000E 003		4		1		
2	yes	yes	yes	no	yes	2	yes	bungalow

- Model: rft_model_classification

worker_ip	task_index	tree_num	tree
-----	-----	-----	----
172.24.57.209	1	0	
{"responseCounts_":{"classic":48,"bungalow":21,"eclectic":97},			
172.24.96.213	0	0	
{"responseCounts_":{"classic":48,"bungalow":17,"eclectic":99},			
172.24.106.80	2	0	
{"responseCounts_":{"classic":40,"bungalow":16,"eclectic":108},			
172.24.57.209	1	1	
{"responseCounts_":{"classic":73,"bungalow":30,"eclectic":107},			
172.24.96.213	0	1	
{"responseCounts_":{"classic":66,"bungalow":23,"eclectic":114},			
172.24.106.80	2	1	
{"responseCounts_":{"classic":65,"bungalow":24,"eclectic":115},			
172.24.57.209	1	2	
{"responseCounts_":{"classic":61,"bungalow":19,"eclectic":80},			
172.24.96.213	0	2	
{"responseCounts_":{"classic":47,"bungalow":18,"eclectic":93},			

```
172.24.106.80      2      2
{"responseCounts_":{"classic":53,"bungalow":8,"eclectic":97},
```

(In the preceding screen, tree column is truncated on the right.)

SQL Call

```
SELECT * FROM DecisionForestPredict (
  ON housing_test_sample PARTITION BY ANY
  ON rft_model_classification AS Model DIMENSION
  USING
  NumericInputs ('[1:5]','garagepl')
  CategoricalInputs ('[6:10]','prefarea')
  IdColumn ('sn')
  Accumulate ('homestyle')
  Detailed ('false')
  OutputProb ('t')
  Responses ('classic','eclectic','bungalow')
)AS dt;
```

Output

homestyle	sn	prediction	prob_classic	prob_eclectic	prob_bungalow
bungalow	329	bungalow	0.00	0.12	0.88
Eclectic	224	eclectic	0.00	1.00	0.00
Classic	198	classic	0.89	0.11	0.00
bungalow	162	bungalow	0.00	0.44	0.56
bungalow	339	bungalow	0.00	0.44	0.56

DecisionTreePredict (SQL Engine)

Note:

This *namePredict* function uses the model output by ML Engine *name* function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

DecisionTreePredict Syntax

```
SELECT * FROM DecisionTreePredict (
  ON { table | view | (query) } AS AttributeTable
  PARTITION BY pid_col [,...]
  ON { table | view | (query) } AS Model DIMENSION
  USING
  AttrTableGroupbyColumns ({ 'gcol' | gcol_range }[,...])
  AttrTablePIDColumns ({ 'pid_col' | pid_col_range }[,...])
  AttrTableValColumn ('value_column')
  [ OutputProb ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})
    [ Responses ('response'[,...]) ]
  ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related Information:

[Column Specification Syntax Elements](#)

DecisionTreePredict Syntax Elements

AttrTableGroupByColumns

Specify the names of the columns on which AttributeTable is partitioned. Each partition contains one attribute of the input data.

AttrTablePIDColumns

Specify the names of the columns that define the data point identifiers.

AttrTableValColumn

Specify the name of the AttributeTable column that contains the input values.

OutputProb

[Required to be 'true' with Responses, optional otherwise.] Specify whether to output probabilities.

Default: 'false'

Responses

[Optional with OutputProb, disallowed otherwise.] Specify the labels for which to output probabilities.

Default behavior: Output only the probability of the predicted class.

Accumulate

[Optional] Specify the names of the input columns to copy to the output table.

If you are using this function to create input for ML Engine ROC function, this syntax element must specify *actual_label*.

DecisionTreePredict Input

Table	Description
AttributeTable	Contains test data. Has same schema as ML Engine DecisionTree InputTable.
Model	Model output by ML Engine DecisionTree function.

AttributeTable Schema

See *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

Model Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Double quotation marks around some column names are required because the names contain special characters.

Column	Data Type	Description
node_id	INTEGER, SMALLINT, or BIGINT	Node identifier.
node_size	INTEGER, SMALLINT, or BIGINT	Number of objects in node.
"node_gini(p)" or node_gini	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	GINI impurity value for information in node. For ImpurityMeasurement ('gini'), column name is node_gini(p); otherwise, it is node_gini.
"node_entropy(p)" or node_entropy	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Entropy impurity value for the information in the node. For ImpurityMeasurement ('entropy'), column name is node_entropy(p); otherwise, it is node_entropy.
"node_chisq_pv(p)" or node_chisq_pv	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Chi-square impurity value for the information in the node. For ImpurityMeasurement ('chisquare'), column name is node_chisq_pv(p); otherwise, it is node_chisq_pv.
node_label	CHARACTER or VARCHAR	Output category for node.

Column	Data Type	Description
node_majorvotes	INTEGER, SMALLINT, or BIGINT	Number of objects that belong to category identified by node_label.
split_value	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Numeric split value.
"split_gini(p)" or split_gini	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	GINI impurity measurement for information in node after splitting. For ImpurityMeasurement ('gini'), column name is split_gini(p); otherwise, it is split_gini.
"split_entropy(p)" or split_entropy	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Entropy impurity measurement for the information in node after splitting. For ImpurityMeasurement ('entropy'), column name is split_entropy(p); otherwise, it is split_entropy.
"split_chisq_pv(p)" or split_chisq_pv	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	Chi-square impurity measurement for information in node after splitting. For ImpurityMeasurement ('chisquare'), column name is split_chisq_pv(p); otherwise, it is split_chisq_pv.
left_id	INTEGER, SMALLINT, or BIGINT	Identifier of left child of node.
left_size	INTEGER, SMALLINT, or BIGINT	Number of objects in left child of node.
left_label	CHARACTER or VARCHAR	Output category for left child of node.
left_majorvotes	INTEGER, SMALLINT, or BIGINT	Number of objects that belong to category identified by left_label.
right_id	INTEGER, SMALLINT, or BIGINT	Identifier of right child of node.
right_size	INTEGER, SMALLINT, or BIGINT	Number of objects in right child of node.
right_label	CHARACTER or VARCHAR	Output category for right child of node.
right_majorvotes	INTEGER, SMALLINT, or BIGINT	Number of objects that belong to category identified by right_label.
left_bucket	CHARACTER or VARCHAR	When split value is categorical attribute, value in left child of node.
right_bucket	CHARACTER or VARCHAR	When split value is categorical attribute, value in right child of node.
left_label_prob_list	CHARACTER or VARCHAR	[Column appears only with OutputResponseProbList ('true').] Probability of each label for left child of node.
right_label_prob_list	CHARACTER or VARCHAR	[Column appears only with OutputResponseProbList ('true').] Probability of each label for right child of node.

Column	Data Type	Description
prob_label_order	CHARACTER or VARCHAR	[Column appears only with OutputResponseProbList ('true').] Label order probability for left and right children of node.
attribute	CHARACTER or VARCHAR	Split attribute.
node_majorfreq	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	[Column appears only with Weighted ('true').] Weighted objects that belong to category identified by node_label.
left_majorfreq	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	[Column appears only with Weighted ('true').] Weighted objects that belong to category identified by left_label.
right_majorfreq	INTEGER, SMALLINT, BIGINT, NUMBER, or DOUBLE PRECISION	[Column appears only with Weighted ('true').] Weighted objects that belong to category identified by right_label.

DecisionTreePredict Output

Output Table Schema

Column	Data Type	Description																		
id_column	Any	Data point identifier from attribute_table (DecisionTree InputTable).																		
pred_label	VARCHAR	Predicted response value for data point.																		
prob	DOUBLE PRECISION	<div><p>[Column appears only with OutputProb ('true') and without Responses syntax element.] Probability that observation belongs to class pred_label, which depends on value of DecisionTree syntax element ResponseProbDistType used to create model:</p><table><tr><th>ResponseProbDistType</th><th>Probability Formula</th></tr><tr><td>'laplace'</td><td>$(TP + 1) / (TP + FP + C)$</td></tr><tr><td>'frequency' or 'rawcount'</td><td>L_c / L</td></tr></table><p>Where:</p><table><tr><th>Operand</th><th>Description</th></tr><tr><td>TP</td><td>Number of true positives at leaf.</td></tr><tr><td>FP</td><td>Number of false positives at leaf.</td></tr><tr><td>C</td><td>Number of trees.</td></tr><tr><td>L_c</td><td>Number of leaf nodes for class.</td></tr><tr><td>L</td><td>Number of leaf nodes in tree.</td></tr></table></div>	ResponseProbDistType	Probability Formula	'laplace'	$(TP + 1) / (TP + FP + C)$	'frequency' or 'rawcount'	L_c / L	Operand	Description	TP	Number of true positives at leaf.	FP	Number of false positives at leaf.	C	Number of trees.	L_c	Number of leaf nodes for class.	L	Number of leaf nodes in tree.
ResponseProbDistType	Probability Formula																			
'laplace'	$(TP + 1) / (TP + FP + C)$																			
'frequency' or 'rawcount'	L_c / L																			
Operand	Description																			
TP	Number of true positives at leaf.																			
FP	Number of false positives at leaf.																			
C	Number of trees.																			
L_c	Number of leaf nodes for class.																			
L	Number of leaf nodes in tree.																			

Column	Data Type	Description
<i>prob_for_label_response</i>	DOUBLE PRECISION	[Column appears only with Responses syntax element.] Probability that observation belongs to category <i>response</i> , calculated as in the description of column <i>prob</i> .
<i>accumulate_column</i>	Same as in AttributeTable	Column copied from AttributeTable.

DecisionTreePredict Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment ① in the left sidebar.

DecisionTreePredict Examples Input

- AttributeTable: *iris_attribute_test*
- Model: *iris_attribute_output*

Both tables are created in "DecisionTree Example: Create Model" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

For input table column descriptions, see [NaiveBayesPredict Example](#).

iris_attribute_test

pid	attribute	attrvalue
5	petal_length	1.4
5	petal_width	0.2
5	sepal_length	5
5	sepal_width	3.6
10	petal_length	1.5
10	petal_width	0.1
10	sepal_length	4.9
10	sepal_width	3.1
15	petal_length	1.2
15	petal_width	0.2
15	sepal_length	5.8
15	sepal_width	4

pid	attribute	attrvalue
...

iris_attribute_output

node_id	node_size	node_gini(p)	node_entropy	node_chisq_pv	node_label	node_majorvotes	split_value
0	120	0.666666666666667	1.58496250072116	1	1	40	3
2	80	0.5	1	1	2	40	1.70000004768372
5	39	0.0499671268902038	0.172036949353113	1	2	38	4.90000009536743
6	41	0.0928019036287924	0.281193796432043	1	3	39	4.90000009536743
14	37	0.0525931336742148	0.179256066928321	1	3	36	2.90000009536743
30	24	0.0798611111111112	0.249882292833186	1	3	23	3.20000004768372
61	14	0.13265306122449	0.371232326640875	1	3	13	6.30000019073486

DecisionTreePredict Example: Apply Model to Test Data**Input**

See [DecisionTreePredict Examples Input](#).

SQL Call

```
CREATE MULTISET TABLE decisiontree_predict AS (
  SELECT * FROM DecisionTreePredict (
    ON iris_attribute_test AS AttributeTable PARTITION BY pid
    ON iris_attribute_output as Model DIMENSION
    USING
    AttrTableGroupbyColumns ('attribute')
    AttrTablePIDColumns ('pid')
    AttrTableValColumn ('attrvalue')
  ) AS dt
) WITH DATA;
```

Output

This query returns the following table:

```
SELECT * FROM decisiontree_predict ORDER BY 1;
```

The predict labels 1, 2, and 3 correspond to species setosa, versicolor, and virginica.

pid	pred_label
5	1
10	1
15	1
20	1
25	1
30	1
35	1
40	1
45	1
50	1
55	2
60	2
65	2
70	2
75	2
80	2
85	2
90	2
95	2
100	2
105	3
110	3
115	3
120	2

pid	pred_label
125	3
130	2
135	2
140	3
145	3
150	3

DecisionTreePredict Example: OutputProb, Responses

Input

See [DecisionTreePredict Examples Input](#).

SQL Call

```
SELECT * FROM DecisionTreePredict (
  ON iris_attribute_test AS AttributeTable PARTITION BY pid
  ON iris_attribute_output AS Model DIMENSION
  USING
    AttrTableGroupByColumns ('attribute')
    AttrTablePIDColumns ('pid')
    AttrTableValColumn ('attrvalue')
    OutputProb ('true')
    Responses ('1','2','3')
) AS dt ORDER BY pid;
```

Output

pid	pred_label	prob_for_label_1	prob_for_label_2	prob_for_label_3
5	1	0.95348	0.02326	0.02326
10	1	0.95348	0.02326	0.02326
15	1	0.95348	0.02326	0.02326
20	1	0.95348	0.02326	0.02326
25	1	0.95348	0.02326	0.02326
30	1	0.95348	0.02326	0.02326
35	1	0.95348	0.02326	0.02326

pid	pred_label	prob_for_label_1	prob_for_label_2	prob_for_label_3
40	1	0.95348	0.02326	0.02326
45	1	0.95348	0.02326	0.02326
50	1	0.95348	0.02326	0.02326
55	2	0.02632	0.94736	0.02632
60	2	0.02632	0.94736	0.02632
65	2	0.02632	0.94736	0.02632
70	2	0.02632	0.94736	0.02632
75	2	0.02632	0.94736	0.02632
80	2	0.02632	0.94736	0.02632
85	2	0.02632	0.94736	0.02632
90	2	0.02632	0.94736	0.02632
95	2	0.02632	0.94736	0.02632
100	2	0.02632	0.94736	0.02632
105	3	0.06250	0.12500	0.81250
110	3	0.07692	0.07692	0.84616
115	3	0.06250	0.06250	0.87500
120	3	0.14286	0.57143	0.28571
125	3	0.07692	0.07692	0.84616
130	3	0.14286	0.57143	0.28571
135	3	0.14286	0.57143	0.28571
140	3	0.06250	0.12500	0.81250
145	3	0.07692	0.07692	0.84616
150	3	0.25000	0.25000	0.50000

GLMPredict (SQL Engine)

Note:

This *namePredict* function uses the model output by ML Engine *name* function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

GLMPredict Syntax

```
SELECT * FROM GLMPredict (
  ON { table | ( query ) } [ PARTION BY ANY ]
  ON { table | view | (query) } AS Model DIMENSION
  [ USING
    [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
    [ Family ('family') ]
    [ LinkFunction ('link') ]
  ]
) AS alias;
```

You must specify the keyword USING to use any function syntax element.

Related Information:

[Column Specification Syntax Elements](#)

GLMPredict Syntax Elements

Accumulate

[Optional] Specify the names of input table columns to copy to the output table.

Family

[Optional] Specify the distribution exponential family.

If you specify this syntax element, you must give it the same value that you used for the Family syntax element of ML Engine GLM function when you created the model table.

Default: Read from the model table

LinkFunction

[Optional] Specify the link function. For the canonical link functions (default link functions) and the link functions allowed for each exponential family, see the GLM function description in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

If you specify this syntax element, you must give it the same value that you used for the LinkFunction syntax element of ML Engine GLM function when you created the model table.

Default: 'CANONICAL'

GLMPredict Input

Table	Description
Input	Contains new data.
Model	Model output by ML Engine GLM function. For schema, see <i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003. If the GLM call that created the model table specified the Step syntax element, include the optional ORDER BY clause in the GLMPredict call; otherwise, the GLMPredict result is nondeterministic.

Input Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Any	Column to copy to output table.
<i>dependent_variable_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, DOUBLE PRECISION, VARCHAR(<i>n</i>), CHAR(<i>n</i>)	Dependent/response variables. Cannot be NULL.
<i>predictor_variable_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, DOUBLE PRECISION	Independent/predictor variable. Cannot be NULL.

Any numeric *dependent_variable_column* or *predictor_variable_column* that is expected to be categorical must be cast to VARCHAR.

Model Table Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Column	Data Type	Description
attribute	INTEGER	Numeric index of predictor.
predictor	CHARACTER or VARCHAR	Predictor name.
category	CHARACTER or VARCHAR	For categorical predictor, its level. For numeric predictor, NULL.
estimate	DOUBLE PRECISION	Estimated coefficient.
std_error	DOUBLE PRECISION	Standard error of coefficient.
t_score	DOUBLE PRECISION	[Column appears only with Family ('GAUSSIAN').] The t_score follows a t(N-p-1) distribution.
z_score	DOUBLE PRECISION	[Column appears only without Family ('GAUSSIAN').] The z-score follows the N(0,1) distribution.

Column	Data Type	Description
p_value	DOUBLE PRECISION	p-value for z_score. (p-value represents significance of each coefficient.)
significance	CHARACTER or VARCHAR	Significance code for p_value.
family	CHARACTER or VARCHAR	Distribution exponential family, specified by Family syntax element.

GLMPredict Output

Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in <i>input_table</i>	Column copied from input table.
fitted_value	DOUBLE PRECISION	Score of the input data, given by equation $g^{-1}(X\beta)$, where g^{-1} is the inverse link function, X the predictors, and β is the vector of coefficients estimated by the GLM function. For other values of Family, the scores are the expected values of dependent/response variable, conditional on the predictors.

GLMPredict Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

GLMPredict Example: Logistic Distribution Prediction

This example predicts the admission status of students.

Input

- Input table: admissions_test, which has admissions information for 20 students
- Model: glm_admissions_model, output by "GLM Example: Logistic Regression Analysis with Intercept" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003

Input Table Column Descriptions

Column	Description
id	Student identifier (unique)

Column	Description
masters	Whether student has a masters degree—yes or no (categorical)
gpa	Grade point average on a 4.0 scale (numerical)
stats	Statistical skills—Novice, Beginner, or Advanced (categorical)
programming	Programming skills—Novice, Beginner, or Advanced (categorical)
admitted	Whether student was admitted—1 (yes) or 0 (no)

admissions_test

id	masters	gpa	stats	programming	admitted
50	yes	3.95	Beginner	Beginner	0
51	yes	3.76	Beginner	Beginner	0
52	no	3.7	Novice	Beginner	1
53	yes	3.5	Beginner	Novice	1
54	yes	3.5	Beginner	Advanced	1
55	no	3.6	Beginner	Advanced	1
56	no	3.82	Advanced	Advanced	1
57	no	3.71	Advanced	Advanced	1
58	no	3.13	Advanced	Advanced	1
59	no	3.65	Novice	Novice	1
60	no	4	Advanced	Novice	1
61	yes	4	Advanced	Advanced	1
62	no	3.7	Advanced	Advanced	1
63	no	3.83	Advanced	Advanced	1
64	yes	3.81	Advanced	Advanced	1
65	yes	3.9	Advanced	Advanced	1
66	no	3.87	Novice	Beginner	1
67	yes	3.46	Novice	Beginner	0
68	no	1.87	Advanced	Novice	1
69	no	3.96	Advanced	Advanced	1

SQL Call

```
CREATE MULTISET TABLE glmpredict_admissions AS (
  SELECT * FROM GLMPredict (
    ON admissions_test PARTITION BY ANY
    ON glm_admissions_model AS Model DIMENSION
    USING
    Accumulate ('id','masters','gpa','stats','programming','admitted')
    Family ('LOGISTIC')
    LinkFunction ('LOGIT')
  ) AS dt
) WITH DATA;
```

Output

This query returns the following table:

```
SELECT * FROM glmpredict_admissions ORDER BY 1;
```

Fitted values can vary in precision, because they depend on the model table output by ML Engine GLM function and fetched to Advanced SQL Engine.

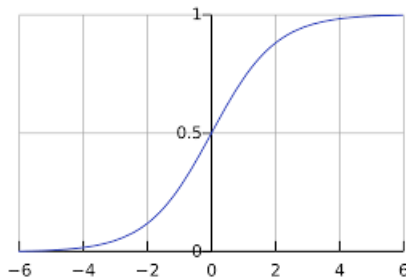
glmpredict_admissions

id	masters	gpa	stats	programming	admitted	fitted value
50	yes	3.950000000000000E 000	Beginner	Beginner	0	3.50763408888030E-001
51	yes	3.760000000000000E 000	Beginner	Beginner	0	3.55708978581653E-001
52	no	3.700000000000000E 000	Novice	Beginner	1	7.58306140231079E-001
53	yes	3.500000000000000E 000	Beginner	Novice	1	5.56012779663342E-001
54	yes	3.500000000000000E 000	Beginner	Advanced	1	7.69474352959112E-001
55	no	3.600000000000000E 000	Beginner	Advanced	1	9.68031141480050E-001
56	no	3.820000000000000E 000	Advanced	Advanced	1	9.45772725968165E-001
57	no	3.710000000000000E 000	Advanced	Advanced	1	9.46411914806798E-001
58	no	3.130000000000000E 000	Advanced	Advanced	1	9.49666186386367E-001
59	no	3.650000000000000E 000	Novice	Novice	1	8.74189685344822E-001
60	no	4.000000000000000E 000	Advanced	Novice	1	8.65058992199339E-001
61	yes	4.000000000000000E 000	Advanced	Advanced	1	6.50618727191735E-001
62	no	3.700000000000000E 000	Advanced	Advanced	1	9.46469669158547E-001
63	no	3.830000000000000E 000	Advanced	Advanced	1	9.45714262806374E-001
64	yes	3.810000000000000E 000	Advanced	Advanced	1	6.55523357798207E-001
65	yes	3.900000000000000E 000	Advanced	Advanced	1	6.53204164468356E-001
66	no	3.870000000000000E 000	Novice	Beginner	1	7.54738501228429E-001
67	yes	3.460000000000000E 000	Novice	Beginner	0	2.60034297764359E-001

id	masters	gpa	stats	programming	admitted	fitted value
68	no	1.87000000000000E 000	Advanced	Novice	1	8.90965518431337E-001
69	no	3.96000000000000E 000	Advanced	Advanced	1	9.44948816395031E-001

Categorizing fitted_value Column

The fitted_value column gives the probability that a student belongs to one of the output classes. The following figure shows a typical logistic regression graph, mapping the input x-axis against a y probability value between [0,1].



A fitted_value probability greater than or equal to 0.5 implies class 1 (student admitted); a probability less than 0.5 implies class 0 (student rejected).

The following code adds a fitted_category column to glmpredict_admissions and populates it:

```
ALTER table glmpredict_admissions
  ADD fitted_category int;
UPDATE glmpredict_admissions SET fitted_category = 1
  WHERE fitted_value > 0.4999;
UPDATE glmpredict_admissions SET fitted_category = 0
  WHERE fitted_value < 0.4999;
```

This query returns the following table:

```
SELECT * FROM glmpredict_admissions ORDER BY 1;
```

id	masters	gpa	stats	programming	admitted	fitted_value	fitted_category
50	yes	3.95000000000000E 000	Beginner	Beginner	0	3.50763408888030E-001	0
51	yes	3.76000000000000E 000	Beginner	Beginner	0	3.55708978581653E-001	0

2: Advanced SQL Engine Analytic Functions

id	masters	gpa	stats	programming	admitted	fitted_value	fitted_category
52	no	3.70000000000000E000	Novice	Beginner	1	7.58306140231079E-001	1
53	yes	3.50000000000000E000	Beginner	Novice	1	5.56012779663342E-001	1
54	yes	3.50000000000000E000	Beginner	Advanced	1	7.69474352959112E-001	1
55	no	3.60000000000000E000	Beginner	Advanced	1	9.68031141480050E-001	1
56	no	3.82000000000000E000	Advanced	Advanced	1	9.45772725968165E-001	1
57	no	3.71000000000000E000	Advanced	Advanced	1	9.46411914806798E-001	1
58	no	3.13000000000000E000	Advanced	Advanced	1	9.49666186386367E-001	1
59	no	3.65000000000000E000	Novice	Novice	1	8.74189685344822E-001	1
60	no	4.00000000000000E000	Advanced	Novice	1	8.65058992199339E-001	1
61	yes	4.00000000000000E000	Advanced	Advanced	1	6.50618727191735E-001	1
62	no	3.70000000000000E000	Advanced	Advanced	1	9.46469669158547E-001	1
63	no	3.83000000000000E000	Advanced	Advanced	1	9.45714262806374E-001	1
64	yes	3.81000000000000E000	Advanced	Advanced	1	6.55523357798207E-001	1
65	yes	3.90000000000000E000	Advanced	Advanced	1	6.53204164468356E-001	1

id	masters	gpa	stats	programming	admitted	fitted_value	fitted_category
66	no	3.870000000000000E000	Novice	Beginner	1	7.54738501228429E-001	1
67	yes	3.460000000000000E000	Novice	Beginner	0	2.60034297764359E-001	0
68	no	1.870000000000000E000	Advanced	Novice	1	8.90965518431337E-001	1
69	no	3.960000000000000E000	Advanced	Advanced	1	9.44948816395031E-001	1

Prediction Accuracy

This query returns the prediction accuracy:

```
SELECT (SELECT COUNT(id) FROM glmpredict_admissions
WHERE admitted = fitted_category)/(SELECT count(id)
FROM glmpredict_admissions) AS prediction_accuracy;
```

prediction_accuracy
1.00000000000000000000

GLMPredict Example: Gaussian Distribution Prediction

This example evaluates the predictions for new houses, comparing the original price information with root mean square error evaluation (RMSE).

Input

- Input table: housing_test, as in [DecisionForestPredict Example: Specify Column Names](#)
- Model: glm_housing_model, output by "GLM Example: Gaussian Distribution Analysis" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003

SQL Call

The canonical link specifies the default family link, which is "identity" for the Gaussian distribution.

```

DROP TABLE glmpredict_housing;

CREATE MULTISET TABLE glmpredict_housing AS (
  SELECT * FROM GLMPredict (
    ON housing_test PARTITION BY ANY
    ON glm_housing_model AS Model DIMENSION
    USING
    Accumulate ('sn', 'price')
    Family ('GAUSSIAN')
    LinkFunction ('CANONICAL')
  ) AS dt
) WITH DATA;

```

Output

This query returns the following table:

```
SELECT * FROM glmpredict_housing ORDER BY 1;
```

sn	price	fitted_value
13	27000	3.734584400000000E 004
16	37900	4.368713175000000E 004
25	42000	4.090202800000000E 004
38	67000	7.248767050000000E 004
53	68000	7.923869370000000E 004
104	132000	1.115280070000000E 005
111	43000	3.910288120000000E 004
117	93000	6.693695100000000E 004
132	44500	4.181988650000000E 004
140	43000	4.161179150000000E 004
142	40000	4.439414650000000E 004
157	60000	6.657126435000000E 004
161	63900	6.490098290000000E 004

The fitted_value column gives the predicted house price.

Root Mean Square Error Evaluation

This query returns the root mean square error evaluation (RMSE):

```
SELECT SQRT(AVG(POWER(glmpredict_housing.price -
  glmpredict_housing.fitted_value, 2))) AS RMSE FROM glmpredict_housing;
```

rmse
1.06854695738768E 004

GLMPredict Example: Casting Input Column to VARCHAR

Like [GLMPredict Example: Logistic Distribution Prediction](#), this example predicts the admission status of students. In both examples, the input column masters is categorical—the value can be yes or no. In the other example, the value is 'yes' or 'no'. In this example, the value is numerical—1 for yes or 0 for no—therefore, it must be cast to VARCHAR.

Input

- Input table: admissions_test_2, which has admissions information for 20 students
- Model: glm_admissions_model, output by "GLM Example: Logistic Regression Analysis with Intercept" in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003, with the category column modified as follows:

attribute	predictor	category
1	masters	'1'
2	masters	'0'

admissions_test_2

id	masters	gpa	stats	programming	admitted
50	1	3.95000000000000E 000	Beginner	Beginner	0
51	1	3.76000000000000E 000	Beginner	Beginner	0
52	0	3.70000000000000E 000	Novice	Beginner	1
53	1	3.50000000000000E 000	Beginner	Novice	1
54	1	3.50000000000000E 000	Beginner	Advanced	1
55	0	3.60000000000000E 000	Beginner	Advanced	1
56	0	3.82000000000000E 000	Advanced	Advanced	1

id	masters	gpa	stats	programming	admitted
57	0	3.71000000000000E 000	Advanced	Advanced	1
58	0	3.13000000000000E 000	Advanced	Advanced	1
59	0	3.65000000000000E 000	Novice	Novice	1
60	0	4.00000000000000E 000	Advanced	Novice	1
61	1	4.00000000000000E 000	Advanced	Advanced	1
62	0	3.70000000000000E 000	Advanced	Advanced	1
63	0	3.83000000000000E 000	Advanced	Advanced	1
64	1	3.81000000000000E 000	Advanced	Advanced	1
65	1	3.90000000000000E 000	Advanced	Advanced	1
66	0	3.87000000000000E 000	Novice	Beginner	1
67	1	3.46000000000000E 000	Novice	Beginner	0
68	0	1.87000000000000E 000	Advanced	Novice	1
69	0	3.96000000000000E 000	Advanced	Advanced	1

SQL Call

```
CREATE MULTiset TABLE glmpredict_admissions_2 AS (
  SELECT * FROM GLMPredict (
    ON (
      SELECT id, CAST(masters AS varchar(10)) AS masters,
        gpa, stats, programming, admitted
      FROM admissions_test
    ) PARTITION BY ANY
    ON glm_admissions_model AS Model DIMENSION
    USING
    Accumulate ('id','masters','gpa','stats','programming','admitted')
    Family ('LOGISTIC')
    LinkFunction ('LOGIT')
  ) AS dt
) WITH DATA;
```

Output

This query returns the following table:

```
SELECT * FROM glmpredict_admissions_2 ORDER BY 1;
```

Fitted values can vary in precision, because they depend on the model table output by ML Engine GLM function and fetched to Advanced SQL Engine.

glmpredict_admissions_2

id	masters	gpa	stats	programming	admitted	fitted value
50	1	3.95000000000000E 000	Beginner	Beginner	0	3.50763408888030E-001
51	1	3.76000000000000E 000	Beginner	Beginner	0	3.55708978581653E-001
52	0	3.70000000000000E 000	Novice	Beginner	1	7.58306140231079E-001
53	1	3.50000000000000E 000	Beginner	Novice	1	5.56012779663342E-001
54	1	3.50000000000000E 000	Beginner	Advanced	1	7.69474352959112E-001
55	0	3.60000000000000E 000	Beginner	Advanced	1	9.68031141480050E-001
56	0	3.82000000000000E 000	Advanced	Advanced	1	9.45772725968165E-001
57	0	3.71000000000000E 000	Advanced	Advanced	1	9.46411914806798E-001
58	0	3.13000000000000E 000	Advanced	Advanced	1	9.49666186386367E-001
59	0	3.65000000000000E 000	Novice	Novice	1	8.74189685344822E-001
60	0	4.00000000000000E 000	Advanced	Novice	1	8.65058992199339E-001
61	1	4.00000000000000E 000	Advanced	Advanced	1	6.50618727191735E-001
62	0	3.70000000000000E 000	Advanced	Advanced	1	9.46469669158547E-001
63	0	3.83000000000000E 000	Advanced	Advanced	1	9.45714262806374E-001
64	1	3.81000000000000E 000	Advanced	Advanced	1	6.55523357798207E-001
65	1	3.90000000000000E 000	Advanced	Advanced	1	6.53204164468356E-001
66	0	3.87000000000000E 000	Novice	Beginner	1	7.54738501228429E-001
67	1	3.46000000000000E 000	Novice	Beginner	0	2.60034297764359E-001
68	0	1.87000000000000E 000	Advanced	Novice	1	8.90965518431337E-001
69	0	3.96000000000000E 000	Advanced	Advanced	1	9.44948816395031E-001

Categorizing fitted_value Column

See [GLMPredict Example: Logistic Distribution Prediction](#).

Prediction Accuracy

See [GLMPredict Example: Logistic Distribution Prediction](#).

MovingAverage (SQL Engine)

The MovingAverage function computes average values in a series, using the specified moving average type.

Moving Average Type	Description
Cumulative Moving Average	Computes cumulative moving average of value from beginning of series.

Moving Average Type	Description
Exponential Moving Average	Computes average of points in series, applying damping factor that exponentially decreases weights of older values.
Modified Moving Average	Computes first value as simple moving average. Computes subsequent values by adding new value and subtracting last average from resulting sum.
Simple Moving Average	Computes unweighted mean of previous n data points.
Triangular Moving Average	Computes double-smoothed average of points in series.
Weighted Moving Average	Computes average of points in series, applying weights to older values. Weights for older values decrease arithmetically.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- The ORDER BY clause supports only ASCII collation.
- The PARTITION BY clause assumes column names are in Normalization Form C (NFC).

Weighted Moving Average

A weighted average has multiplying factors that give different weights to different data points. Mathematically, the moving average is the convolution of the data points with a moving average function. In technical analysis, a weighted moving average (WMA) has weights that decrease arithmetically. In an n -point weighted moving average, the most recent data point has weight n , the second most recent data point has weight $(n - 1)$, and so on, until the weight is zero.

With MAvgType ('W'), the MovingAverage function uses this formula:

$$WMA_M = (nV_M + (n-1)V_{M-1} + \dots + 2V_{(M-n+2)} + V_{(M-n+1)}) / (n + (n-1) + \dots + 2 + 1)$$

Where $Total_M = V_M + \dots + V_{(M-n+1)}$, the following equations are true:

- $Total_{M+1} = Total_M + V_{M+1} - V_{(M-n+1)}$
- $Numerator_{M+1} = Numerator_M + n*V_{M+1} - Total_M$
- $WMA_{M+1} = Numerator_{M+1} / (n*(n+1)/2)$

V_M is the target column value at index M in the window under consideration.

The value n —the number of old values to use when calculating the new weighted moving average—is specified by the WindowSize syntax element.

Triangular Moving Average

The triangular moving average (TMA) differs from the simple moving average in that it is double-smoothed; that is, averaged twice. Double-smoothing keeps the triangular moving average from responding to new data points as fast as the simple moving average. For an average that responds quickly to new data points, use the simple moving average or exponential moving average.

With MAvgType ('T'), the MovingAverage function uses this procedure:

1. Compute the window size, N :

$$N = \text{ceil}(\text{window_size} + 1) / 2$$

The value *window_size* is specified by the WindowSize syntax element.

2. Compute the simple moving average of each target column, using this formula:

$$\text{SMA}_i = (V_1 + V_2 + \dots + V_N) / N$$

The function calculates SMA_i on the i th window of the target column from the start of the row.

V_i is the value of the target column at index i in the window.

3. Compute the triangular moving average by computing the simple moving average with window size N on the values obtained in step 2, using this formula:

$$\text{TMA} = (\text{SMA}_1 + \text{SMA}_2 + \dots + \text{SMA}_N) / N$$

The function writes the cumulative moving average values computed for the first n rows, where n is less than N , to the output table.

Simple Moving Average

The simple moving average (SMA) is the unweighted mean of the previous n data points. For example, a 10-day simple moving average of closing price is the mean of closing prices for the previous 10 days.

With MAvgType ('S'), the MovingAverage function uses this procedure:

1. Compute the arithmetic average of the first *window_size* rows.

The value *window_size* is specified by the WindowSize syntax element. In the next step, it is called N .

2. For each subsequent row, compute the new simple moving average value with this formula:

$$\text{SMA} = (V_1 + V_2 + \dots + V_N) / N$$

V_i is the value of the target column at index i in the window.

Modified Moving Average

The first point of the modified moving average (MMA) is calculated like the first point of the simple moving average. Each subsequent point is calculated by adding the new value to the most recently calculated

modified moving average value and then, from that sum, subtracting the last average value. The difference is the new modified moving average value.

With MAvgType ('M'), the MovingAverage function uses this procedure:

1. Compute the arithmetic average of the first *window_size* rows.

The value *window_size* is specified by the WindowSize syntax element.

2. For each subsequent row, compute the new modified moving average value with this formula:

$$\text{MMA}_M = \text{MMA}_{M-1} + (1/\text{window_size}) (V_M - \text{MMA}_{M-1})$$

V_M is the current value.

Exponential Moving Average

Exponential moving average (EMA), or exponentially weighted moving average (EWMA), applies a damping factor, *alpha*, that exponentially decreases the weights of older values. This technique gives much more weight to recent observations, while retaining older observations.

With MAvgType ('E'), the MovingAverage function uses this procedure:

1. Compute the arithmetic average of the first *n* rows.

The value *n* is specified by the StartRows syntax element.

2. For each subsequent row, compute the new exponential moving average value with this formula:

$$\text{EMA}_M = \text{alpha} * V + (1 - \text{alpha}) * \text{EMA}_{M-1}$$

The value *alpha* is specified by the Alpha syntax element. *V* is the new value.

Cumulative Moving Average

In a cumulative moving average (CMA), the data are added to the data set in an ordered data stream over time. The objective is to compute the average of all the data at each point in time when new data arrived. A typical use case is an investor who wants to find the average price of all transactions of a specific stock over time, up to the current time.

With MAvgType ('C'), the MovingAverage function computes the arithmetic average of all the rows from the beginning of the series with this formula:

$$\text{CMA} = (V_1 + V_2 + \dots + V_N)/N$$

V_i is a value. *N* is the number of rows from the beginning of the data set.

MovingAverage Syntax

```
SELECT * FROM MovingAverage (
  ON { table | view | (query) }
  [ PARTITION BY partition_column [, ...] ]
```



```

    [ ORDER BY order_column [,...] ]
  [ USING
    [ MAvgType ({ 'C' | 'E' | 'M' | 'S' | 'T' | 'W' }) ]
    [ TargetColumns ({ 'target_column' | 'target_column_range' }[,...])]
    [ Alpha (alpha) ]
    [ StartRows (n) ]
    [ IncludeFirst ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
    [ WindowSize (window_size) ]
  ]
) AS alias;

```

Note:

If the ON clause does not include the PARTITION BY and ORDER BY clauses, results are nondeterministic.

MovingAverage Syntax Elements

MAvgType

[Optional] Specify one of the following moving average types:

Type	Description
'C' (Default)	Cumulative moving average.
'E'	Exponential moving average.
'M'	Modified moving average.
'S'	Simple moving average.
'T'	Triangular moving average.
'W'	Weighted moving average.

TargetColumns

[Optional] Specify the input column names for which to compute the moving average.

Default behavior: The function copies every input column to the output table but does not compute any moving averages.

Alpha

[Optional with MAvgType E, otherwise ignored.] Specify the damping factor, a value in the range [0, 1], which represents a percentage in the range [0, 100]. For example, if *alpha* is 0.2, the damping factor is 20%. A higher *alpha* discounts older observations faster.

Default: 0.1

StartRows

[Optional with MAvgType E, otherwise ignored.] Specify the number of rows to skip before calculating the exponential moving average. The function uses the arithmetic average of these rows as the initial value of the exponential moving average. The value *n* must be an integer.

Default: 2

IncludeFirst

[Ignored with MAvgType C, otherwise optional.] Specify whether to include the starting rows in the output table. If you specify 'true', the output columns for the starting rows contain NULL, because their moving average is undefined.

Default: 'false'

WindowSize

[Optional with MAvgType M, S, T, and W; otherwise ignored.] Specify the number of previous values to consider when computing the new moving average. The data type of *window_size* must be BYTEINT, SMALLINT, or INTEGER.

Minimum value: 3

Default: '10'

MovingAverage Input

Input Table Schema

Column	Data Type	Description
<i>partition_column</i>	Any	Column by which input data is partitioned. This column must contain all rows of an entity. For example, if function is to compute moving average of a particular stock share price, all transactions of that stock must be in one partition. PARTITION BY clause assumes column names are in Normalization Form C (NFC).
<i>order_column</i>	Any	Column by which input table is ordered. ORDER BY clause supports only ASCII collation.
<i>target_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, NUMBER, or	Values to average.

Column	Data Type	Description
	DOUBLE PRECISION	

MovingAverage Output

Output Table Schema

Column	Data Type	Description
<i>partition_column</i>	Same as in input table	Column by which input data is partitioned.
<i>order_column</i>	Same as in input table	Column by which input table is ordered.
<i>target_column</i>	Same as in input table	Column copied from input table.
<i>target_column_typemavg</i>	DOUBLE PRECISION	Moving average of <i>target_column</i> values when row was added to data set, where <i>type</i> is value of MAvgType syntax element.

MovingAverage Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

MovingAverage Examples Input

The input table, `company1_stock`, contains 25 observations of common stock closing prices.

`company1_stock`

id	name	period	stockprice
1	Company1	1961-05-17 00:00:00.000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000
4	Company1	1961-05-22 00:00:00.000000	459.000000000000
5	Company1	1961-05-23 00:00:00.000000	462.000000000000
6	Company1	1961-05-24 00:00:00.000000	459.000000000000
7	Company1	1961-05-25 00:00:00.000000	463.000000000000
8	Company1	1961-05-26 00:00:00.000000	479.000000000000

id	name	period	stockprice
9	Company1	1961-05-29 00:00:00.000000	493.000000000000
10	Company1	1961-05-31 00:00:00.000000	490.000000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000
14	Company1	1961-06-06 00:00:00.000000	497.000000000000
15	Company1	1961-06-07 00:00:00.000000	496.000000000000
16	Company1	1961-06-08 00:00:00.000000	490.000000000000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000
20	Company1	1961-06-14 00:00:00.000000	491.000000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000
22	Company1	1961-06-16 00:00:00.000000	482.000000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000
24	Company1	1961-06-20 00:00:00.000000	478.000000000000
25	Company1	1961-06-21 00:00:00.000000	479.000000000000

MovingAverage Example: Cumulative Moving Average

This example computes a cumulative moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
    MAvgType ('C')
    TargetColumns ('stockprice')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_cmavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	458.500000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	456.333333333333
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	457.000000000000
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	458.000000000000
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	458.166666666667
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	458.857142857143
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	461.375000000000
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	464.888888888889
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.400000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	469.636363636364
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	472.000000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	474.076923076923
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	475.714285714286
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	477.066666666667
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	477.875000000000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	478.529411764706
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	478.500000000000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	478.947368421053
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	479.550000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	479.904761904762
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	480.000000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	479.956521739130
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	479.875000000000
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	479.840000000000

MovingAverage Example: Exponential Moving Average

This example computes an exponential moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
  MAvgType ('E')
  TargetColumns ('stockprice')
  StartRows (10)
  Alpha (0.1)
  IncludeFirst ('true')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_emavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	?
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	?
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	?
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	?
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	?
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	?
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	?
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	?
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	?
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.400000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	469.860000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	472.674000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	475.306600000000
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	477.475940000000

id	name	period	stockprice	stockprice_emavg
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	479.328346000000
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	480.395511400000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	481.255960260000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	480.930364234000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	481.537327810600
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	482.483595029540
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	482.935235526586
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	482.841711973927
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	482.457540776535
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	482.011786698881
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	481.710608028993

MovingAverage Example: Modified Moving Average

This example computes the triangular moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
    MAvgType ('M')
    TargetColumns ('stockprice')
    WindowSize (10)
    IncludeFirst ('true')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_mmavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	459.700000000000

id	name	period	stockprice	stockprice_mmavg
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	458.930000000000
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	458.937000000000
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	459.243300000000
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	459.218970000000
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	459.597073000000
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	461.537365700000
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	464.683629130000
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.215266217000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	469.693739595300
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	472.524365635770
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	475.171929072193
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	477.354736164974
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	479.219262548476
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	480.297336293629
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	481.167602664266
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	480.850842397839
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	481.465758158055
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	482.419182342250
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	482.877264108025
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	482.789537697222
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	482.410583927500
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	481.969525534750
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	481.672572981275

MovingAverage Example: Simple Moving Average

This example computes a simple moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```

SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
  MAvgType ('S')
  TargetColumns ('stockprice')
  WindowSize (10)
  IncludeFirst ('true')
) AS dt ORDER BY id;

```

Output

id	name	period	stockprice	stockprice_smaavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	?
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	?
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	?
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	?
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	?
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	?
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	?
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	?
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	?
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.400000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	470.600000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	474.700000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	479.400000000000
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	483.200000000000
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	486.600000000000
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	489.700000000000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	492.300000000000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	492.200000000000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	491.600000000000
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	491.700000000000

id	name	period	stockprice	stockprice_savg
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	491.200000000000
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	489.600000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	487.600000000000
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	485.700000000000
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	484.000000000000

MovingAverage Example: Triangular Moving Average

This example computes the triangular moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
  MAvgType ('T')
  TargetColumns ('stockprice')
  WindowSize (10)
  IncludeFirst ('true')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_tmavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	459.250000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	458.277777777778
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	457.958333333333
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	457.966666666667
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	458.000000000000
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	457.777777777778
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	458.416666666667

id	name	period	stockprice	stockprice_tmavg
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	460.555555555556
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	463.444444444444
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	467.000000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	471.611111111111
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	477.138888888889
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	482.555555555556
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	486.916666666667
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	490.416666666667
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	493.000000000000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	493.944444444444
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	493.555555555556
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	492.500000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	491.111111111111
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	489.500000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	487.694444444444
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	486.444444444444
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	485.305555555556

MovingAverage Example: Weighted Moving Average

This example computes the weighted moving average for the price of stock.

Input

See [MovingAverage Examples Input](#).

SQL Call

```
SELECT * FROM MovingAverage (
  ON company1_stock PARTITION BY name ORDER BY period
  USING
  MAvgType ('W')
  TargetColumns ('stockprice')
  WindowSize (10)
```

```
IncludeFirst ('true')
) AS dt ORDER BY id;
```

Output

id	name	period	stockprice	stockprice_wmavg
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	?
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	?
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	?
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	?
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	?
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	?
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	?
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	?
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	?
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	473.454545454545
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	477.927272727273
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	482.909090909091
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	487.327272727273
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	490.527272727273
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	492.854545454545
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	493.472727272727
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	493.345454545455
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	490.745454545455
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	489.800000000000
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	489.690909090909
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	488.836363636364
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	487.163636363636
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	485.236363636364
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	483.490909090909
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	482.272727272727

NaiveBayesPredict (SQL Engine)

Note:

This *name*Predict function uses the model output by ML Engine *name* function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

NaiveBayesPredict Syntax

```
SELECT * FROM NaiveBayesPredict (
  ON { table | view | (query) } PARTITION BY ANY
  ON { table | view | (query) } AS Model DIMENSION
  USING
  IDColumn ('test_point_id_col')
  NumericInputs ('numeric_input_column'[,...] )
  CategoricalInputs ('categorical_input_column'[,...] )
  Responses ('response'[,...])
) AS alias;
```

NaiveBayesPredict Syntax Elements

IDColumn

Specify the name of the input table column that contains the ID that uniquely identifies the test input data.

NumericInputs

[Required if CategoricalInputs is omitted.] Specify the same *numeric_input_columns* that you specified when you used the NaiveBayesMap and NaiveBayesReduce functions to create the model table from the training data.

CategoricalInputs

[Required if NumericInputs is omitted.] Specify the same *categorical_input_columns* that you specified when you used the NaiveBayesMap and NaiveBayesReduce functions to create the model table from the training data.

Responses

Specify the responses to output.

NaiveBayesPredict Input

Table	Description
Input	Contains test data. Has same schema as ML Engine Naive Bayes Classifier input table.
Model	Model output by ML Engine Naive Bayes Classifier function.

Input Table Schema

See *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

Model Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Double quotation marks around some column names are required because the names are either Advanced SQL Engine reserved keywords or are camel-case.

Column	Data Type	Description
"class" or class_nb	VARCHAR	Response.
"variable" or variable_nb	VARCHAR	Input variable (name of input column).
"type" or type_nb	VARCHAR	Input variable types ('NUMERIC' or 'CATEGORICAL').
category	VARCHAR	For categorical predictor, its level. For numeric predictor, NULL.
cnt	INTEGER or BIGINT	Count of observations with this class, variable, and category.
"sum" or sum_nb	INTEGER or DOUBLE_PRECISION	For numerical predictor, sum of variable values for observations with this class, variable, and category. For categorical predictor, NULL.
"sumSq" or sum_sq	INTEGER or DOUBLE_PRECISION	For numerical predictor, sum of square of variable values for observations with this class, variable, and category. For categorical predictor, NULL.
"totalCnt" or total_cnt	INTEGER or BIGINT	Total count of observations.

NaiveBayesPredict Output

Output Table Schema

Each row of the table represents one observation.

Column	Data Type	Description
id	INTEGER	Row (observation) identifier.
prediction	VARCHAR	Prediction for observation.
loglik_response_i	DOUBLE PRECISION	Loglikelihood (natural logarithm of probability) that observation has <i>response</i> .

NaiveBayesPredict Example

Input

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

- Input table: nb_iris_input_test
- Model: nb_iris_model

The model is created in the Naive Bayes example in *Teradata Vantage™ Machine Learning Engine Analytic Function Reference*, B700-4003.

Input Table Column Descriptions

Column	Description
id	Unique identifier of observation
sepal_length	Numeric
sepal_width	Numeric
petal_length	Numeric
petal_width	Numeric
species	Setosa, versicolor, or virginica

nb_iris_input_test

id	sepal_length	sepal_width	petal_length	petal_width	species
5	5	3.6	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
15	5.8	4	1.2	0.2	setosa
20	5.1	3.8	1.5	0.3	setosa
25	4.8	3.4	1.9	0.2	setosa
30	4.7	3.2	1.6	0.2	setosa

id	sepal_length	sepal_width	petal_length	petal_width	species
35	4.9	3.1	1.5	0.2	setosa
40	5.1	3.4	1.5	0.2	setosa
45	5.1	3.8	1.9	0.4	setosa
50	5	3.3	1.4	0.2	setosa
55	6.5	2.8	4.6	1.5	versicolor
60	5.2	2.7	3.9	1.4	versicolor
65	5.6	2.9	3.6	1.3	versicolor
70	5.6	2.5	3.9	1.1	versicolor
75	6.4	2.9	4.3	1.3	versicolor
80	5.7	2.6	3.5	1	versicolor
85	5.4	3	4.5	1.5	versicolor
90	5.5	2.5	4	1.3	versicolor
95	5.6	2.7	4.2	1.3	versicolor
100	5.7	2.8	4.1	1.3	versicolor
105	6.5	3	5.8	2.2	virginica
110	7.2	3.6	6.1	2.5	virginica
115	5.8	2.8	5.1	2.4	virginica
120	6	2.2	5	1.5	virginica
125	6.7	3.3	5.7	2.1	virginica
130	7.2	3	5.8	1.6	virginica
135	6.1	2.6	5.6	1.4	virginica
140	6.9	3.1	5.4	2.1	virginica
145	6.7	3.3	5.7	2.5	virginica
150	5.9	3	5.1	1.8	virginica

nb_iris_model

class	variable	type	category	cnt	sum	sumSq	totalcnt
setosa	sepal_width	NUMERIC	?	40	136.700000524521	473.290003499985	40
setosa	petal_width	NUMERIC	?	40	10.1000002026558	3.03000012755394	40
setosa	sepal_length	NUMERIC	?	40	199.900000095367	1004.27000005722	40

class	variable	type	category	cnt	sum	sumSq	totalcnt
setosa	petal_length	NUMERIC	?	40	57.6999998092651	84.2099996709824	40
versicolor	sepal_width	NUMERIC	?	40	111.10000038147	313.130002088547	40
versicolor	petal_width	NUMERIC	?	40	53.299999833107	72.7099995040894	40
versicolor	sepal_length	NUMERIC	?	40	239.599999427795	1446.13999296188	40
versicolor	petal_length	NUMERIC	?	40	172.399999141693	752.219992570878	40
virginica	sepal_width	NUMERIC	?	40	118.799999952316	356.539999780655	40
virginica	petal_width	NUMERIC	?	40	81.1999989748001	166.999995970726	40
virginica	sepal_length	NUMERIC	?	40	264.400000572205	1764.92000530243	40
virginica	petal_length	NUMERIC	?	40	222.299999713898	1249.1499958992	40

SQL Call

```

DROP TABLE nb_iris_predict;

CREATE MULTISET TABLE nb_iris_predict AS (
  SELECT * FROM NaiveBayesPredict (
    ON nb_iris_input_test PARTITION BY ANY
    ON nb_iris_model AS Model DIMENSION
    USING
    IDColumn ('id')
    NumericInputs ('sepal_length','sepal_width','petal_length','petal_width')
    Responses ('virginica','setosa','versicolor')
  ) AS dt
) WITH DATA;

```

Output

This query returns the following table:

```
SELECT * FROM nb_iris_predict ORDER BY 1;
```

The output provides a prediction for each row in the test data set and specifies the log likelihood values that were used to make the predictions for each category.

id	prediction	loglik_virginica	loglik_setosa	loglik_versicolor
5	setosa	-60.9907330174083	0.940424559067427	-38.2319825308929
10	setosa	-61.5861966261907	-0.173043897170957	-37.6660830556247
15	setosa	-64.7169548001753	-3.55476375390931	-42.613272284101

id	prediction	loglik_virginica	loglik_setosa	loglik_versicolor
20	setosa	-57.7992844148636	0.531796840642284	-35.7613053354934
25	setosa	-55.0939143017897	-3.23703029869347	-32.1179858509341
30	setosa	-58.0673073752287	0.109611164911179	-34.9285997859276
35	setosa	-58.1980267787658	0.660202577013632	-34.9335988704833
40	setosa	-58.3538858459019	0.976840811041703	-35.4425587940391
45	setosa	-50.3847602463201	-4.36921429673761	-29.0537478266948
50	setosa	-59.4745348026195	1.00257959230347	-36.5026022674224
55	versicolor	-5.22108005914589	-270.465431908161	-1.7396367893394
60	versicolor	-11.3356467465064	-174.565470791378	-2.31925264962004
65	versicolor	-12.6496488706934	-138.435722453706	-2.1898005756116
70	versicolor	-15.236843619572	-152.47255627778	-2.3538459106499
75	versicolor	-8.34632493685681	-214.383653794905	-1.14727508911532
80	versicolor	-18.455946984498	-109.900955754698	-3.72743011721095
85	versicolor	-7.00283150694931	-249.656488976769	-2.00455589365379
90	versicolor	-12.0279925543069	-177.470336291088	-1.74539749109463
95	versicolor	-10.1802450220293	-198.037109900803	-1.10567314638237
100	versicolor	-10.1315405651018	-187.294956922171	-1.02885306444447
105	virginica	-1.58321671192447	-540.56351949849	-14.859643718252
110	virginica	-6.11301966870239	-654.801984259278	-28.8385135092999
115	virginica	-3.64635253153959	-456.647579953406	-15.3298808321577
120	versicolor	-7.73615017754911	-322.909009762056	-3.53629430321742
125	virginica	-1.87627054598219	-509.817023097936	-13.7515396871732
130	virginica	-3.36908052149115	-469.802937074554	-9.13832860900173
135	versicolor	-5.81482980902253	-403.678170868448	-4.51644862072851
140	virginica	-1.48430911768034	-463.610989255182	-12.0238603485835
145	virginica	-3.82266629516761	-576.395460020916	-22.6942168473031
150	virginica	-2.57004648415525	-366.506113945482	-4.84887216455807

NaiveBayesTextClassifierPredict (SQL Engine)

Note:

This *namePredict* function uses the model output by ML Engine *name* function to analyze the input data and make predictions.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

NaiveBayesTextClassifierPredict Syntax

```
SELECT * FROM NaiveBayesTextClassifierPredict (
  ON { table | view | (query) } AS PredictorValues PARTITION BY doc_id_column [,...]
  ON { table | view | (query) } AS Model DIMENSION
  USING
    InputTokenColumn ('input_token_column')
    [ ModelType ({ 'Multinomial' | 'Bernoulli' }) ]
    DocIDColumns ({ 'doc_id_column' | 'doc_id_column_range' }[,...])
    [ ModelTokenColumn ('model_token_column')
      ModelCategoryColumn ('model_category_column')
      ModelProbColumn ('model_probability_column') ]
    [ TopK ({ num_of_top_k_predictions | 'num_of_top_k_predictions' }) |
      Responses ('response' [,...]) ]
    [ OutputProb {'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'} ]
    [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related Information:

[Column Specification Syntax Elements](#)

NaiveBayesTextClassifierPredict Syntax Elements

InputTokenColumn

Specify the name of the PredictorValues column that contains the tokens.

ModelType

[Optional] Specify the model type of the text classifier.

Default: 'Multinomial'

DocIDColumns

Specify the names of the PredictorValues columns that contain the document identifier.

ModelTokenColumn

[Optional] Specify the name of the Model table column that contains the tokens.

Default: First column of Model table

ModelCategoryColumn

[Optional] Specify the name of the Model table column that contains the prediction categories.

Default: Second column of Model table

ModelProbColumn

[Optional] Specify the name of the Model table column that contains the probability values.

Default: Third column of Model table

TopK

[Disallowed with Responses, otherwise optional.] Specify the number of most likely prediction categories to output with their loglikelihood values (for example, the top 10 most likely prediction categories). To see the probability of each class, use OutputProb ('true').

Default: All prediction categories

Responses

[Disallowed with TopK, otherwise optional.] Specify the labels for which to output loglikelihood values and probabilities (with OutputProb ('true')).

OutputProb

Specify whether to output the calculated probability for each observation.

Default: 'false'

Accumulate

Specify the names of the PredictorValues table columns to copy to the output table.

Note:

- Specify either all or none of the syntax elements ModelTokenColumn, ModelCategoryColumn, and ModelProbColumn.
- Specifying neither TopK nor Responses is equivalent to specifying TopK.

NaiveBayesTextClassifierPredict Input

Table	Description
PredictorValues	Contains test data, for which to predict outcomes, in document-token pairs. To transform input document into this form, input it to ML Engine function TextTokenizer or TextParser. TextTokenizer and TextParser have language-processing limitations that might limit support for Unicode input data (see <i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003).
Model	Model output by ML Engine NaiveBayesTextClassifierTrainer2 function. For schema, see <i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003.

PredictorValues Schema

Column	Data Type	Description
<i>doc_id_column</i>	CHARACTER, VARCHAR, INTEGER, or SMALLINT	Identifier of document that contains classified testing tokens.
<i>token_column</i>	CHARACTER or VARCHAR	Testing token.
<i>accumulate_column</i>	Any	Column to copy to output table.

Model Schema

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Column	Data Type	Description
token	CHARACTER or VARCHAR	Classified training token.
category	CHARACTER or VARCHAR	Prediction category for token.
prob	DOUBLE PRECISION	Probability that token is in category.

NaiveBayesTextClassifierPredict Output

Output Table Schema

Column	Data Type	Description
doc_id	CHARACTER, VARCHAR, INTEGER, or SMALLINT	Single- or multiple-column document identifier.
prediction	VARCHAR	Prediction category.
loglik	DOUBLE PRECISION	Loglikelihood that document belongs to category.
loglik_response	DOUBLE PRECISION	[Column appears only with Responses syntax element. Loglikelihood that document belongs to class label <i>response</i> .
prob	DOUBLE PRECISION	[Column appears only when you both specify OutputProb ('true') and omit Responses.] Probability that document belongs to class label in prediction column, which is $\max(\text{softmax}(\text{loglik}))$.
prob_response	DOUBLE PRECISION	[Column appears only when you specify both OutputProb ('true') and Responses. Column appears once for each specified <i>response</i> .] Probability that document belongs to class label <i>response</i> , which is $\text{softmax}(\text{loglik_response})$.
accumulate_column	Same as in PredictorValues table	Column copied from PredictorValues table.

NaiveBayesTextClassifierPredict Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

NaiveBayesTextClassifierPredict Example: TopK Specified

Input

- PredictorValues: complaints_test_tokenized, created by applying ML Engine TextTokenizer function to the table complaints_test, a log of vehicle complaints, as follows:

```
CREATE MULTISET TABLE complaints_test_tokenized AS (
  SELECT doc_id, doc_name, lower(cast(token AS VARCHAR(20))) AS token
  FROM TextTokenizer (
    ON complaints_test PARTITION BY ANY
    USING
```

```

    TextColumn ('text_data')
    OutputByWord ('true')
    Accumulate ('doc_id', doc_name)
  ) AS dt
) WITH DATA;

```

- Model: Use the following query to create the complaints_tokens_model:

```

CREATE TABLE complaints_tokens_model(
  token VARCHAR(100),
  category VARCHAR(100),
  prob DOUBLE PRECISION
);

```

complaints_test

doc_id	doc_name	text_data
1	A	ELECTRICAL CONTROL MODULE IS SHORTENING OUT, CAUSING THE VEHICLE TO STALL. ENGINE WILL BECOME TOTALLY INOPERATIVE. CONSUMER HAD TO CHANGE ALTERNATOR/ BATTERY AND STARTER, AND MODULE REPLACED 4 TIMES, BUT DEFECT STILL OCCURRING CANNOT DETERMINE WHAT IS CAUSING THE PROBLEM.
2	B	ABS BRAKES FAIL TO OPERATE PROPERLY, AND AIR BAGS FAILED TO DEPLOY DURING A CRASH AT APPROX. 28 MPH IMPACT. MANUFACTURER NOTIFIED.
3	C	WHILE DRIVING AT 60 MPH GAS PEDAL GOT STUCK DUE TO THE RUBBER THAT IS AROUND THE GAS PEDAL.
4	D	THERE IS A KNOCKING NOISE COMING FROM THE CATALYTIC CONVERTER , AND THE VEHICLE IS STALLING. ALSO, HAS PROBLEM WITH THE STEERING.
5	E	CONSUMER WAS MAKING A TURN ,DRIVING AT APPROX 5- 10 MPH WHEN CONSUMER HIT ANOTHER VEHICLE. UPON IMPACT, DUAL AIRBAGS DID NOT DEPLOY . ALL DAMAGE WAS DONE FROM ENGINE TO TRANSMISSION,TO THE FRONT OF VEHICLE, AND THE VEHICLE CONSIDERED A TOTAL LOSS.
6	F	WHEEL BEARING AND HUBS CRACKED, CAUSING THE METAL TO GRIND WHEN MAKING A RIGHT TURN. ALSO WHEN APPLYING THE BRAKES, PEDAL GOES TO THE FLOOR, CAUSE UNKNOWN. WAS ADVISED BY MIDAS NOT TO DRIVE VEHICLE- WHEEL COULD COME OFF.
7	G	DRIVING ABOUT 5-10 MPH, THE VEHICLE HAD A LOW FRONTAL IMPACT IN WHICH THE OTHER VEHICLE HAD NO DAMAGES. UPON IMPACT, DRIVER'S AND THE PASSENGER'S AIR BAGS DID NOT DEPLOY, RESULTING IN INJURIES. PLEASE PROVIDE FURTHER INFORMATION AND VIN#.
8	H	THE AIR BAG WARNING LIGHT HAS COME ON. INDICATING AIRBAGS ARE INOPERATIVE.THEY WERE FIXED ONE AT THE TIME, BUT PROBLEM HAS REOCCURRED.

doc_id	doc_name	text_data
9	I	CONSUMER WAS DRIVING WEST WHEN THE OTHER CAR WAS GOING EAST. THE OTHER CAR TURNED IN FRONT OF CONSUMER'S VEHICLE, CONSUMER HIT OTHER VEHICLE AND STARTED TO SPIN AROUND , COULDN'T STOP, RESULTING IN A CRASH. UPON IMPACT, AIRBAGS DIDN'T DEPLOY.
10	J	WHILE DRIVING ABOUT 65 MPH AND THE TRANSMISISON MADE A STRANGE NOISE, AND THE LEFT FRONT AXLE LOCKED UP. THE DEALER HAS REPAIRED THE VEHICLE.

SQL Call

```

SELECT * FROM NaiveBayesTextClassifierPredict (
  ON complaints_test_tokenized AS PredictorValues PARTITION BY doc_id
  ON complaints_tokens_model AS Model DIMENSION
  USING
  ModelType ('Bernoulli')
  InputTokenColumn ('token')
  DocIDColumns ('doc_id')
  OutputProb ('true')
  Accumulate ('doc_name')
  TopK ('2')
) AS dt ORDER BY doc_id;

```

Output

doc_id	prediction	loglik	prob	doc_name
1	crash	-1.38044220625651E 002	1.41243173571687E-009	A
1	no_crash	-1.17666267644292E 002	9.99999998587568E-001	A
2	crash	-1.04652470718918E 002	1.70704288519507E-003	B
2	no_crash	-9.82811865081127E 001	9.98292957114805E-001	B
3	crash	-1.03026451289745E 002	2.26862573862878E-012	C
3	no_crash	-7.62146044204976E 001	9.9999999997731E-001	C
4	crash	-1.10830711173169E 002	1.42026355157382E-011	D
4	no_crash	-8.58531176043404E 001	9.9999999985797E-001	D
5	no_crash	-1.23936921216052E 002	3.43620138383542E-002	E
5	crash	-1.20601083912966E 002	9.65637986161646E-001	E
6	crash	-1.30310015371040E 002	2.61074198636704E-006	F
6	no_crash	-1.17454141890718E 002	9.99997389258014E-001	F
7	no_crash	-1.23123774759574E 002	4.23603936872661E-002	G
7	crash	-1.20005517060745E 002	9.57639606312734E-001	G
8	crash	-1.08617321658980E 002	8.92398441816595E-009	H

8	no_crash	-9.00827983614664E 001	9.99999991076016E-001	H
9	crash	-1.19919230739025E 002	2.24857954852037E-002	I
9	no_crash	-1.16147101713878E 002	9.77514204514796E-001	I
10	crash	-1.06104244132225E 002	7.57068462691010E-008	J
10	no_crash	-8.97078469668254E 001	9.99999924293154E-001	J

NaiveBayesTextClassifierPredict Example: Responses Specified

Input

As in [NaiveBayesTextClassifierPredict Example: TopK Specified](#)

- PredictorValues: complaints_test_tokenized
- Model: complaints_tokens_model

SQL Call

```
SELECT * FROM NaiveBayesTextClassifierPredict (
  ON complaints_test_tokenized AS PredictorValues PARTITION BY doc_id
  ON complaints_tokens_model AS Model DIMENSION
  USING
  ModelType ('Bernoulli')
  InputTokenColumn ('token')
  DocIDColumns ('doc_id')
  OutputProb ('true')
  Accumulate ('doc_name')
  Responses ('crash', 'no crash')
) AS dt ORDER BY doc_id;
```

Output

doc_id	prediction	loglik_crash	loglik_no_crash	prob_crash	prob_no_crash	doc_name
1	no_crash	-1.38044220625651E 002	-1.17666267644292E 002	1.41243173571687E-009	9.99999998587568E-001	A
2	no_crash	-1.04652470718918E 002	-9.82811865081127E 001	1.70704288519507E-003	9.98292957114805E-001	B
3	no_crash	-1.03026451289745E 002	-7.62146044204976E 001	2.26862573862878E-012	9.9999999997731E-001	C
4	no_crash	-1.10830711173169E 002	-8.58531176043404E 001	1.42026355157382E-011	9.9999999985797E-001	D
5	crash	-1.20601083912966E 002	-1.23936921216052E 002	9.65637986161646E-001	3.43620138383542E-002	E
6	no_crash	-1.30310015371040E 002	-1.17454141890718E 002	2.61074198636704E-006	9.99997389258014E-001	F
7	crash	-1.20005517060745E 002	-1.23123774759574E 002	9.57639606312734E-001	4.23603936872661E-002	G
8	no_crash	-1.08617321658980E 002	-9.00827983614664E 001	8.92398441816595E-009	9.99999991076016E-001	H
9	no_crash	-1.19919230739025E 002	-1.16147101713878E 002	2.24857954852037E-002	9.77514204514796E-001	I
10	no_crash	-1.06104244132225E 002	-8.97078469668254E 001	7.57068462691010E-008	9.99999924293154E-001	J

NGramSplitter (SQL Engine)

The NGramSplitter function *tokenizes* (splits) an input stream of text and outputs *n* multigrams (called *n-grams*) based on the specified Reset, Punctuation, and Delimiter syntax elements. NGramSplitter first splits sentences, next removes punctuation characters from them, and finally splits the words into *n*-grams.

NGramSplitter provides more flexibility than standard tokenization when performing text analysis. Many two-word phrases carry important meaning (for example, "machine learning") that single-word tokens do

not capture. This, combined with additional analytical techniques, can be useful for performing sentiment analysis, topic identification, and document classification.

NGramSplitter considers each input row to be one document, and returns a row for each unique n -gram in each document. NGramSplitter also returns, for each document, the counts of each n -gram and the total number of n -grams.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

NGramSplitter Syntax

```
SELECT * FROM NGramSplitter (
  ON { table | view | (query) }
  USING
  TextColumn ('text_column')
  Grams ({ gram_number | 'value_range' }[,...])
  [ OverLapping({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ ConvertToLowerCase ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ Reset ('reset_character...') ]
  [ Punctuation ('punctuation_character...') ]
  [ Delimiter ('delimiter_character...') ]
  [ OutputTotalGramCount ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  [ TotalCountColName ('total_count_column') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  [ NGramColName ('ngram_column') ]
  [ GramLengthColName ('gram_length_column') ]
  [ FrequencyColName ('frequency_column') ]
) AS alias;
```

Related Information:

[Column Specification Syntax Elements](#)

NGramSplitter Syntax Elements

TextColumn

Specify the name of the column that contains the input text. This column must have a SQL string data type.

Grams

Specify the length, in words, of each n -gram (that is, the value of n). A *value_range* has the syntax *integer1-integer2*, where $integer1 \leq integer2$. The values of n , *integer1*, and *integer2* must be positive.

OverLapping

[Optional] Specify whether the function allows overlapping n -grams.

Default: 'true' (Each word in each sentence starts an n -gram, if enough words follow it in the same sentence to form a whole n -gram of the specified size. For information on sentences, see the Reset syntax element description.)

ConvertToLowerCase

[Optional] Specify whether the function converts all letters in the input text to lowercase.

Default: 'true'

Reset

[Optional] Specify, in a string, the characters that can end a sentence. At the end of a sentence, the function discards any partial n -grams and searches for the next n -gram at the beginning of the next sentence. An n -gram cannot span sentences.

Default: '.,?!'

Punctuation

[Optional] Specify, in a string, the punctuation characters for the function to remove before evaluating the input text.

Punctuation characters can be from both Unicode and Latin character sets.

Default: '`~#^&*()-'

Delimiter

[Optional] Specify the character or string that separates words in the input text.

Default: ' ' (space)

OutputTotalGramCount

[Optional] Specify whether the function returns the total number of n -grams in the document (that is, in the row) for each length n specified in the Grams syntax element. If you specify 'true', the TotalCountColName syntax element determines the name of the output table column that contains these totals.

The total number of n -grams is not necessarily the number of unique n -grams.

Default: 'false'

TotalCountColName

[Optional] Specify the name of the output table column that appears if the value of the OutputTotalGramCount syntax element is 'true'.

Default: 'totalcnt'

Accumulate

[Optional] Specify the names of the input table columns to copy to the output table for each n -gram. These columns cannot have the same names as those specified by the syntax elements NGramColName, GramLengthColName, and TotalCountColName.

Default: All input columns for each n -gram

NGramColName

[Optional] Specify the name of the output table column that is to contain the created n -grams.

Default: 'ngram'

GramLengthColName

[Optional] Specify the name of the output table column that is to contain the length of n -gram (in words).

Default: 'n'

FrequencyColName

[Optional] Specify the name of the output table column that is to contain the count of each unique n -gram (that is, the number of times that each unique n -gram appears in the document).

Default: 'frequency'

NGramSplitter Input

Input Table Schema

Each row of the table has a document to tokenize.

Column	Data Type	Description
<i>text_column</i>	VARCHAR or CLOB	Document to tokenize.
<i>accumulate_column</i>	Any	Column to copy to output table.

NGramSplitter Output

Output Table Schema

The table has a row for each unique n -gram in each input document.

Column	Data Type	Description
<i>total_count_column</i>	INTEGER	[Column appears only with TotalCountColName ('true').] Total number of n -grams in document for each length n specified in Grams syntax element.
<i>accumulate_column</i>	Any	Column copied from input table.
<i>ngram_column</i>	VARCHAR	Created n -gram.
<i>gram_length_column</i>	INTEGER	Length of n -gram in words (value n).
<i>frequency_column</i>	INTEGER	Count of each unique n -gram in document.

NGramSplitter Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

NGramSplitter Example: Omit Accumulate

Input

The input table, paragraphs_input, contains sentences about commonly used machine learning techniques.

paragraphs_input

paraid	paratopic	paratext
1	Decision Trees	Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the items target value. It is one of the predictive modeling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.
2	Simple Regression	In statistics, simple linear regression is the least squares estimator of a linear regression model with a single explanatory variable. In other words, simple linear regression fits a straight line through the set of n points in such a way that makes the sum of squared residuals of the model (that is, vertical

paraid	paratopic	paratext
		distances between the points of the data set and the fitted line) as small as possible
...

SQL Call

```
SELECT * FROM NGramSplitter (
  ON paragraphs_input
  USING
  TextColumn ('paratext')
  Grams ('4-6')
  OutputTotalGramCount ('true')
) AS dt;
```

Output

paraid	paratopic	ngram	n	frequency	totalcnt
1	Decision Trees	decision tree learning uses	4	1	73
1	Decision Trees	decision tree learning uses a	5	1	66
1	Decision Trees	decision tree learning uses a decision	6	1	60
1	Decision Trees	tree learning uses a	4	1	73
1	Decision Trees	tree learning uses a decision	5	1	66
1	Decision Trees	tree learning uses a decision tree	6	1	60
1	Decision Trees	learning uses a decision	4	1	73
1	Decision Trees	learning uses a decision tree	5	1	66
1	Decision Trees	learning uses a decision tree as	6	1	60
...

NGramSplitter Example: Specify Accumulate

Input

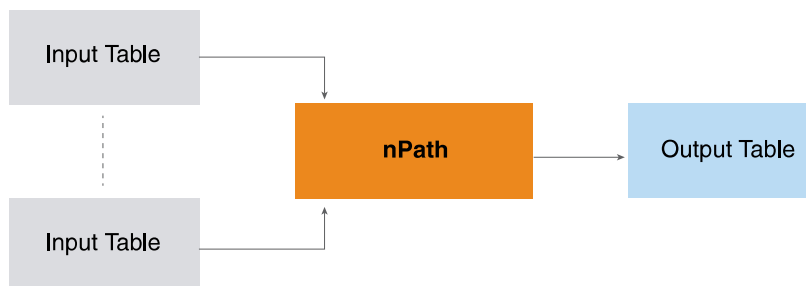
The input table is paragraphs_input, as in [NGramSplitter Example: Omit Accumulate](#).

SQL Call

```
SELECT * FROM NGramSplitter (
  ON paragraphs_input
  USING
  TextColumn ('paratext')
  Grams ('4-6')
  OutputTotalGramCount ('true')
  Accumulate ('[0:1]')
) AS dt;
```

Output**nPath® (SQL Engine)**

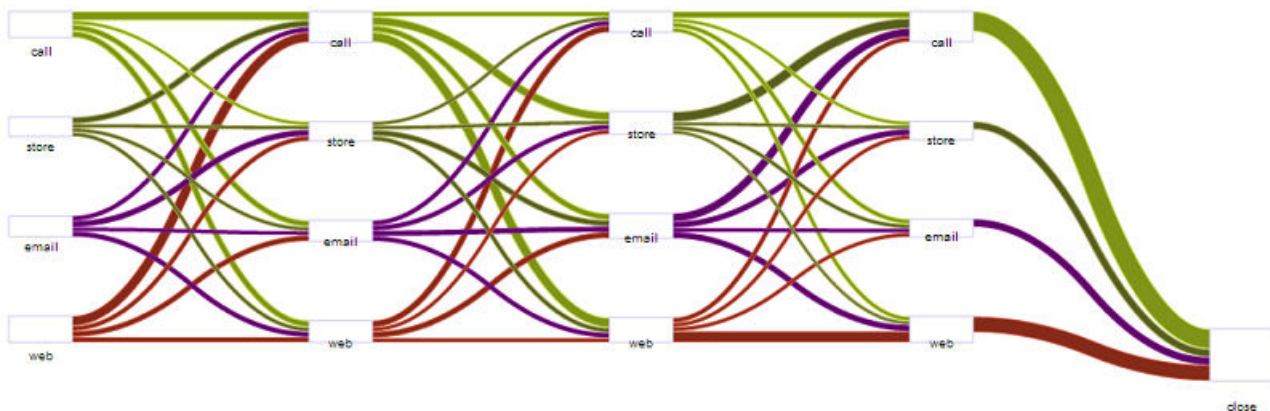
The nPath function scans a set of rows, looking for patterns that you specify. For each set of input rows that matches the pattern, nPath produces a single output row. The function provides a flexible pattern-matching capability that lets you specify complex patterns in the input data and define the values that are output for each matched input set.



nPath is useful when your goal is to identify the paths that lead to an outcome. For example, you can use nPath to analyze:

- Web site click data, to identify paths that lead to sales over a specified amount
- Sensor data from industrial processes, to identify paths to poor product quality
- Healthcare records of individual patients, to identify paths that indicate that patients are at risk of developing conditions such as heart disease or diabetes
- Financial data for individuals, to identify paths that provide information about credit or fraud risks

The output from the nPath function can be input to other ML Engine functions or to a visualization tool such as Teradata® AppCenter.

Sankey Diagram of Advanced SQL Engine nPath Output

An nPath call specifies:

- [Mode \(overlapping or nonoverlapping\)](#)
- [Pattern to match](#)
- [Symbols to use](#)
- [Optional] [Filters to apply](#)
- [Results to output](#)

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.
- When used with this function, the ORDER BY clause supports only ASCII collation.
- When used with this function, the PARTITION BY clause assumes column names are in Normalization Form C (NFC).

nPath Syntax

```
SELECT * FROM nPath (
  ON { table | view | (query) }
  PARTITION BY partition_column
  ORDER BY order_column [ ASC | DESC ][...]
  [ ON { table | view | (query) }
  [ PARTITION BY partition_column | DIMENSION ]
  ORDER BY order_column [ ASC | DESC ]
```



```

][...]
USING
Mode ({ OVERLAPPING | NONOVERLAPPING })
Pattern ('pattern')
Symbols ({ col_expr = symbol_predicate AS symbol }[,...])
[ Filter (filter_expression[,...]) ]
Result ({ aggregate_function (expression OF [ANY] symbol [,...]) AS alias_1 }[,...])
) AS alias_2;

```

nPath Syntax Elements

Mode

Specify the pattern-matching mode:

Option	Description
OVERLAPPING	Find every occurrence of pattern in partition, regardless of whether it is part of a previously found match. One row can match multiple symbols in a given matched pattern.
NONOVERLAPPING	Start next pattern search at row that follows last pattern match.

Pattern

Specify the pattern for which the function searches. You compose *pattern* with the symbols (which you define in the Symbols syntax element), operators, and parentheses.

When patterns have multiple operators, the function applies them in order of precedence, and applies operators of equal precedence from left to right. To force the function to evaluate a subpattern first, enclose it in parentheses. For more information, see [nPath Patterns](#).

Symbols

Defines the symbols that appear in the values of the Pattern and Result syntax elements. The *col_expr* is an expression whose value is a column name, *symbol* is any valid identifier, and *symbol_predicate* is a SQL predicate (often a column name).

Each *col_expr* = *symbol_predicate* must satisfy the SQL syntax of the Advanced SQL Engine when nPath is invoked. Otherwise, it is a syntax error.

For example, this Symbols syntax element is for analyzing website visits:

```

Symbols (
  pagetype = 'homepage' AS H,
  pagetype <> 'homepage' AND pagetype <> 'checkout' AS PP,
  pagetype = 'checkout' AS CO
)

```

The *symbol* is case-insensitive; however, a *symbol* of one or two uppercase letters is easy to identify in patterns.

If *col_expr* represents a column that appears in multiple input tables, you must qualify the ambiguous column name with its table name. For example:

```
Symbols (
  weblog.pagetype = 'homepage' AS H,
  weblog.pagetype = 'thankyou' AS T,
  ads.adname = 'xmaspromo' AS X,
  ads.adname = 'realtorpromo' AS R
)
```

For more information about symbols that appear in the Pattern syntax element value, see [nPath Symbols](#). For more information about symbols that appear in the Result syntax element value, see [nPath Results](#).

Filter

[Optional] Specify filters to impose on the matched rows. The function combines the filter expressions using the AND operator.

This is the *filter_expression* syntax:

```
symbol_expression comparison_operator symbol_expression
```

The two symbol expressions must be type-compatible. This is the *symbol_expression* syntax:

```
{ FIRST | LAST }(column_with_expression OF [ANY](symbol[, ...]))
```

The *column_with_expression* cannot contain the operator AND or OR, and all its columns must come from the same input. If the function has multiple inputs, *column_with_expression* and *symbol* must come from the same input.

The *comparison_operator* is either <, >, <=, >=, =, or <>.

Result

Defines the output columns. The *col_expr* is an expression whose value is a column name; it specifies the values to retrieve from the matched rows. The function applies *aggregate_function* to these values. For details, see [nPath Results](#).

The function evaluates this syntax element once for every matched pattern in the partition (that is, it outputs one row for each pattern match).

nPath Input

The function requires at least one partitioned input table, and can have additional input tables that are either partitioned or DIMENSION tables.

Note:

If the input to nPath is nondeterministic, the results are nondeterministic.

Input Table Schema

Column	Data Type	Description
<i>partition_column</i>	INTEGER or VARCHAR	Column by which every partitioned input table is partitioned.
<i>order_column</i>	INTEGER or VARCHAR	Column by which every input table is ordered.
<i>input_column</i>	INTEGER or VARCHAR	Contains data to search for patterns.

nPath Output

The Result syntax element determines the output—see [nPath Results](#).

nPath Symbols

A *symbol* identifies a row in the Pattern and Result syntax elements. A symbol can be any valid identifier (that is, a sequence of characters and digits that begins with a character) but is typically one or two uppercase letters. Symbols are case-insensitive; that is, 'SU' is identical to 'su', and the system reports an error if you use both.

For example, suppose that you have this input table:

record	city	temp	rh	cloudcover	windspeed	winddirection	rained_next_day
1	?	81	30	0.0	5	NW	1
2	Tempe	76	40	0.2	15	NE	0
3	?	70	70	0.4	10	N	0
4	Tusayan	75	50	0.4	5	NW	0

This table has examples of symbol definitions and the rows of the table that they match in NONOVERLAPPING mode:

Symbol Definition	Rows Matched
temp >= 80 AS H	1

Symbol Definition	Rows Matched						
<code>winddirection = 'NW' AS NW</code>	1, 4						
<code>winddirection = 'NW' OR windspeed > 12 AS W</code>	1, 2, 4						
<code>cloudcover <> 0.0 AND rh > 35 AS C</code>	2, 3, 4						
<code>TRUE AS A</code>	1, 2, 3, 4 This symbol definition matches all rows, for any input table.						
<code>city like 'tu%' AS TU</code>	The like operator depends on Teradata Session mode: <table border="1"> <thead> <tr> <th>Mode</th><th>Match</th></tr> </thead> <tbody> <tr> <td>BTET</td><td>1, 3, 4</td></tr> <tr> <td>ANSI</td><td>None</td></tr> </tbody> </table>	Mode	Match	BTET	1, 3, 4	ANSI	None
Mode	Match						
BTET	1, 3, 4						
ANSI	None						
<code>city not like 'tu%' AS TU</code>	2						
<code>city not like 'Tu%' AS N</code>	2						
<code>city like 'Tu%n' as T</code>	1, 3, 4 The % operator matches any number of characters.						
<code>city like 'Tu__n' as T</code>	1, 3 The underscore (_) operator matches any single character. The pattern 'Tu__n' has three underscores, so it matches 'Tucson' but not 'Tusayan'.						

Rows with NULL values do not match any symbol. That is, the function ignores rows with missing values.

LAG and LEAD Expressions in Symbol Predicates

You can create symbol predicates that compare a row to a previous or subsequent row, using a LAG or LEAD operator.

LAG Expression Syntax

```
{ current_expr operator LAG (previous_expr, lag_rows [, default]) |
  LAG (previous_expr, lag_rows [, default]) operator current_expr }
```

where:

- `current_expr` is the name of a column from the current row (or an expression operating on this column).

- *operator* is either >, >=, <, <=, =, or <>
- *previous_expr* is the name of a column from a previous row (or an expression operating on this column).
- *lag_rows* is the number of rows to count backward from the current row to reach the previous row. For example, if *lag_rows* is 1, the previous row is the immediately preceding row.
- *default* is the value to use for *previous_expr* when there is no previous row (that is, when the current row is the first row or there is no row that is *lag_rows* before the current row).

LAG and LEAD Expression Rules

- A symbol definition can have multiple LAG and LEAD expressions.
- A symbol definition that has a LAG or LEAD expression cannot have an OR operator.
- If a symbol definition has a LAG or LEAD expression and the input is not a table, you must create an alias of the input query, as in [LAG and LEAD Expressions Example: Alias for Input Query](#).

LAG and LEAD Expressions Example: Alias for Input Query

Input

bank_web_clicks

customer_id	session_id	page	timestamp
529	0	ACCOUNT SUMMARY	2004-03-17 16:35:00
529	0	FAQ	2004-03-17 16:38:00
529	0	ACCOUNT HISTORY	2004-03-17 16:42:00
529	0	FUNDS TRANSFER	2004-03-17 16:45:00
529	0	ONLINE STATEMENT ENROLLMENT	2004-03-17 16:49:00
529	0	PROFILE UPDATE	2004-03-17 16:50:00
529	0	ACCOUNT SUMMARY	2004-03-17 16:51:00
529	0	CUSTOMER SUPPORT	2004-03-17 16:53:00
529	0	VIEW DEPOSIT DETAILS	2004-03-17 16:57:00
529	1	ACCOUNT SUMMARY	2004-03-18 01:16:00
529	1	ACCOUNT SUMMARY	2004-03-18 01:18:00
529	1	FAQ	2004-03-18 01:20:00
...

SQL Call

```

SELECT * FROM nPath (
  ON (SELECT customer_id, session_id, datestamp, page FROM bank_web_clicks)
  AS dt1
  PARTITION BY customer_id, session_id
  ORDER BY datestamp
  USING
  Mode (NONOVERLAPPING)
  Pattern ('(DUP|A)*')
  Symbols (
    TRUE AS A,
    page = LAG (page,1) AS DUP
  )
  Result (
    FIRST (customer_id OF any (A)) AS customer_id,
    FIRST (session_id OF A) AS session_id,
    FIRST (datestamp OF A) AS first_date,
    LAST (datestamp OF ANY(A,DUP)) AS last_date,
    ACCUMULATE (page OF A) AS page_path,
    ACCUMULATE (page OF DUP) AS dup_path
  )
) AS dt2;

```

Output**Columns 1-4**

customer_id	session_id	first_date	last_date
529	0	2004-03-17 16:35:00	2004-03-17 16:57:00
529	1	2004-03-18 01:16:00	2004-03-18 01:28:00
529	2	2004-03-18 09:22:00	2004-03-18 09:36:00
529	3	2004-03-18 22:41:00	2004-03-18 22:55:00
529	4	2004-03-19 08:33:00	2004-03-19 08:41:00
529	5	2004-03-19 10:06:00	2004-03-19 10:14:00
...

Columns 5-6

page_path	dup_path
[ACCOUNT SUMMARY, FAQ, ACCOUNT HISTORY, FUNDS TRANSFER, ONLINE STATEMENT ENROLLMENT, PROFILE UPDATE, ACCOUNT SUMMARY, CUSTOMER SUPPORT, VIEW DEPOSIT DETAILS]	[]
[ACCOUNT SUMMARY, FAQ, ACCOUNT SUMMARY, FUNDS TRANSFER, ACCOUNT HISTORY, VIEW DEPOSIT DETAILS, ACCOUNT SUMMARY, ACCOUNT HISTORY]	[ACCOUNT SUMMARY]
[ACCOUNT SUMMARY, ACCOUNT HISTORY, FUNDS TRANSFER, ACCOUNT SUMMARY, FAQ]	[ACCOUNT SUMMARY, ACCOUNT SUMMARY, FAQ]
[ACCOUNT SUMMARY, ACCOUNT HISTORY, ACCOUNT SUMMARY, ACCOUNT HISTORY, FAQ, ACCOUNT SUMMARY]	[ACCOUNT SUMMARY]
[ACCOUNT SUMMARY, FAQ, VIEW DEPOSIT DETAILS, FAQ]	[]
[ACCOUNT SUMMARY, FUNDS TRANSFER, VIEW DEPOSIT DETAILS, ACCOUNT HISTORY]	[VIEW DEPOSIT DETAILS]
...	...

LAG and LEAD Expressions Example: No Alias for Input Query

Input

aggregate_clicks

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	sneakers	home	2009-07-29 20:17:59	Company1	100
1039	2	books	home	2009-04-21 13:17:59	Company4	300
1039	3	television	home	2009-05-23 13:17:59	Company2	500
1039	4	envelopes	home	2009-07-16 11:17:59	Company3	10
1039	4	envelopes	home1	2009-07-16 11:18:16	Company3	10
1039	4	envelopes	page1	2009-07-16 11:18:18	Company3	10
1039	5	bookcases	home	2009-08-19 22:17:59	Company5	150
1039	5	bookcases	home1	2009-08-19 22:18:02	Company5	150
1039	5	bookcases	page1	2009-08-19 22:18:05	Company5	150
1039	5	bookcases	page2	2009-08-22 04:20:05	Company5	150
1039	5	bookcases	checkout	2009-08-24 14:30:05	Company5	150
1039	5	bookcases	page2	2009-08-27 23:03:05	Company5	150
1040	1	tables	home	2009-07-29 20:17:59	Company5	250

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1040	2	Appliances	home	2009-04-21 13:17:59	Company6	1500
1040	3	laptops	home	2009-05-23 13:17:59	Company7	800
1040	4	chairs	home	2009-07-16 11:17:59	Company3	400
1040	4	chairs	home1	2009-07-16 11:18:16	Company3	400
1040	4	chairs	page1	2009-07-16 11:18:18	Company3	400
1040	5	cellphones	home	2009-08-19 22:17:59	Company8	600
1040	5	cellphones	home1	2009-08-19 22:18:02	Company8	600
1040	5	cellphones	page1	2009-08-19 22:18:05	Company8	600
1040	5	cellphones	page2	2009-08-22 04:20:05	Company8	600
1040	5	cellphones	checkout	2009-08-24 14:30:05	Company8	600
1040	5	cellphones	page2	2009-08-27 23:03:05	Company8	600
...

SQL Call

```

SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime ASC
  USING
  Mode (NONOVERLAPPING)
  Pattern ('H+.D*.X*.P1.P2+')
  Symbols (
    TRUE AS X,
    pagetype = 'home' AS H,
    pagetype <> 'home' AND pagetype <> 'checkout' AS D,
    pagetype = 'checkout' AS P1,
    pagetype = 'checkout' AND
    productprice > 100 AND
    productprice > LAG (productprice, 1, 100) AS P2
  )
  Result (
    FIRST (productname OF P1) AS first_product,
    MAX_CHOOSE (productprice, productname OF P2) AS max_product,
    FIRST (sessionid OF P2) AS sessionid
  )
) AS dt ORDER BY sessionid;

```


Output

first_product	max_product	sessionid
bookcases	cellphones	5

nPath Patterns

The value of the Pattern syntax element specifies the sequence of rows for which the function searches. You compose the pattern definition, *pattern*, with symbols (which you define in the Symbols syntax element), operators, and parentheses. In the pattern definition, symbols represent rows. You can combine symbols with pattern operators to define simple or complex patterns of rows for which to search.

The following table lists and describes the basic pattern operators, in decreasing order of precedence. In the table, A and B are symbols that have been defined in the Symbols syntax element.

Basic Pattern Operators

Operator	Description	Precedence
A	Matches one row that meets the definition of A .	1 (highest)
A.	Matches one row that meets the definition of A .	1
A?	Matches 0 or 1 rows that satisfy the definition of A .	1
A*	Matches 0 or more rows that satisfy the definition of A (greedy operator).	1
A+	Matches 1 of more rows that satisfy the definition of A (greedy operator).	1
A.B	Matches two rows, where the first row meets the definition of A and the second row meets the definition of B .	2
A B	Matches one row that meets the definition of either A or B .	3

The nPath function uses greedy pattern matching. That is, it finds the longest available match when matching patterns specified by nongreedy operators. For more information, see [nPath Greedy Pattern Matching](#).

These examples show the pattern operator precedence rules:

- A.B+ is the same as A.(B+)
- A|B* is the same as A|(B*)
- A.B|C is the same as (A.B)|C

Example:

```
A. (B|C)+.D?.X*.A
```

The preceding pattern definition matches any set of rows whose first row meets the definition of symbol A, followed by a nonempty sequence of rows, each of which meets the definition of either symbol B or C, optionally followed by one row that meets the definition of symbol D, followed by any number of rows that meet the definition of symbol X, and ending with a row that meets the definition of symbol A.

You can use parentheses to define precedence rules. Parentheses are recommended for clarity, even where not strictly required.

To indicate that a sequence of rows must start or end with a row that matches a certain symbol, use the start anchor (^) or end anchor (\$) operator.

Start Anchor and End Anchor Pattern Operators

Operator	Description
[^] A	Appears only at the beginning of a pattern. Indicates that a set of rows must start with a row that meets the definition of A.
A ^{\$}	Appears only at the end of a pattern. Indicates that a set of rows must end with a row that meets the definition of A.

Subpattern operators let you specify how often a subpattern must appear in a match. You can specify a minimum number, exact number, or range. In the following table, X represents any pattern definition composed of symbols and any of the previously described pattern operators.

Subpattern Operators

Operator	Description
(X){a}	Matches exactly a occurrences of the pattern X.
(X){a,}	Matches at least a occurrences of the pattern X.
(X){a,b}	Matches at least a and no more than b occurrences of the pattern X.

nPath Greedy Pattern Matching

The nPath function uses greedy pattern matching, finding the longest available match despite any nongreedy operators in the pattern.

For example, consider the input table link2:

nPath Greedy Pattern Matching Examples Input Table link2

userid	job_title	startdate	enddate
21	Chief Exec Officer	1994-10-01	2005-02-28
21	Software Engineer	1996-10-01	2001-06-30
21	Software Engineer	1998-10-01	2001-06-30
21	Chief Exec Officer	2005-03-01	2007-03-31
21	Chief Exec Officer	2007-06-01	?

The following query returns the following table:

```
SELECT job_transition_path, count(*) AS path_count FROM nPath (
  ON link2 PARTITION BY userid ORDER BY startdate
  USING
  Mode (NONOVERLAPPING)
  Pattern ('CEO.ENGR.OTHER*')
  Symbols (
    job_title like '%Software Eng%' AS ENGR,
    TRUE AS OTHER,
    job_title like 'Chief Exec Officer' AS CEO
  )
  Result (accumulate(job_title OF ANY(ENGR,OTHER,CEO)) AS job_transition_path)
) AS dt GROUP BY 1 ORDER BY 2 DESC;
```

job_transition_path	path_count
[Chief Exec Officer, Software Engineer, Software Engineer, Chief Exec Officer, Chief Exec Officer]	1

In the pattern, CEO matches the first row, ENGR matches the second row, and OTHER* matches the remaining rows:

title	pattern('CEO.ENGR.OTHER*')
Chief Exec Officer	
Software Engineer	
Software Engineer	
Chief Exec Officer	
Chief Exec Officer	

The following query returns the following table:

```
SELECT job_transition_path , count(*) AS path_count FROM nPath (
  ON link2 PARTITION BY userid ORDER BY startdate
```

```

USING
Mode (NONOVERLAPPING)
Pattern ('CEO.ENGR.OTHER*.CEO')
Symbols (
  job_title like '%Software Eng%' AS ENGR,
  TRUE AS OTHER,
  job_title like 'Chief Exec Officer' AS CEO
)
Result (accumulate(job_title OF ANY(ENGR,OTHER,CEO)) AS job_transition_path)
) AS dt GROUP BY 1 ORDER BY 2 DESC;

```

job_transition_path	path_count
[Chief Exec Officer, Software Engineer, Software Engineer, Chief Exec Officer, Chief Exec Officer]	1

In the pattern, CEO matches the first row, ENGR matches the second row, OTHER* matches the next two rows, and CEO matches the last row:

title	pattern('CEO.ENGR.OTHER*.CEO')
Chief Exec Officer	
Software Engineer	
Software Engineer	
Chief Exec Officer	
Chief Exec Officer	

nPath Filters

The Filter syntax element specifies filters to impose on the matched rows.

nPath Filters Example

Using clickstream data from an online store, this example finds the sessions where the user visited the checkout page within 10 minutes of visiting the home page. Because there is no way to know in advance how many rows might appear between the home page and the checkout page, the example cannot use a LAG or LEAD expression. Therefore, it uses the Filter syntax element.

Input

clickstream			
userid	sessionid	clicktime	pagetype
1	1	10-10-2012 10:15	home
1	1	10-10-2012 10:16	view

userid	sessionid	clicktime	pagetype
1	1	10-10-2012 10:17	view
1	1	10-10-2012 10:20	checkout
1	1	10-10-2012 10:30	checkout
1	1	10-10-2012 10:35	view
1	1	10-10-2012 10:45	view
2	2	10-10-2012 13:15	home
2	2	10-10-2012 13:16	view
2	2	10-10-2012 13:43	checkout
2	2	10-10-2012 13:35	view
2	2	10-10-2012 13:45	view

SQL Call

```

SELECT * FROM Npath (
  ON clickstream PARTITION BY userid ORDER BY clicktime
  USING
  Symbols (
    pagetype='home' AS home,
    pagetype <> 'home' AND pagetype <> 'checkout' AS clickview,
    pagetype='checkout' AS checkout
  )
  Pattern ('home.clickview*.checkout')
  Result (
    FIRST(userid of ANY(home, checkout, clickview)) AS userid,
    FIRST (sessionid of ANY(home, checkout, clickview)) AS sessionid,
    COUNT (*) of any(home, checkout, clickview) AS cnt,
    FIRST (clicktime of ANY(home)) AS firsthome,
    LAST (clicktime of ANY(checkout)) AS lastcheckout
  )
  Filter (
    FIRST (clicktime + interval '10' minute OF ANY (home)) >
    FIRST (clicktime of any(checkout))
  )
  Mode (NONOVERLAPPING)
);

```

Output

userid	sessionid	cnt	firsthome	lastcheckout
1	1	4	2012-10-10 10:15:00	2012-10-10 10:20:00

nPath Results

The Result syntax element defines the output columns, specifying the values to retrieve from the matched rows and the aggregate function to apply to these values.

For each pattern, the nPath function can apply one or more aggregate functions to the matched rows and output the aggregated results. These are the supported aggregate functions:

- SQL aggregate functions AVG, COUNT, MAX, MIN, and SUM, described in *Teradata Vantage™ - SQL Functions, Expressions, and Predicates*, B035-1145
- ML Engine nPath sequence aggregate functions described in the following table

In the following table, *col_expr* is an expression whose value is a column name, *symbol* is defined by the Symbols syntax element, and *symbol_list* has this syntax:

```
{ symbol | ANY (symbol[,...]) }
```

Function	Description
COUNT ({ * [DISTINCT] col_ expr } OF symbol_list)	Returns either the number of total number of matched rows (*) or the number (or distinct number) of col_expr values in the matched rows.
FIRST (col_expr OF symbol_list)	Returns the col_expr value of the first matched row.
LAST (col_expr OF symbol_list)	Returns the col_expr value of the last matched row.
NTH (col_expr, n OF symbol_ list)	Returns the col_expr value of the nth matched row, where n is a nonzero value of the data type SMALLINT, INTEGER, or BIGINT. The sign of n determines whether the nth matched row is nth from the first or last matched row. For example, if n is 1, the nth matched row is the first matched row, and if n is -1, the nth matched row is the last matched row. If n is greater than the number of matched rows, the nth function returns NULL.

Function	Description
FIRST_NOTNULL (<i>col_expr</i> OF <i>symbol_list</i>)	Returns the first non-null <i>col_expr</i> value in the matched rows.
LAST_NOTNULL (<i>col_expr</i> OF <i>symbol_list</i>)	Returns the last non-null <i>col_expr</i> value in the matched rows.
MAX_CHOOSE (<i>quantifying_col_expr</i> , <i>descriptive_col_expr</i> OF <i>symbol_list</i>)	<p>Returns the <i>descriptive_col_expr</i> value of the matched row with the highest-sorted <i>quantifying_col_expr</i> value. For example, MAX_CHOOSE (product_price, product_name OF B) returns the product_name of the most expensive product in the rows that map to B.</p> <p>The <i>descriptive_col_expr</i> can have any data type. The <i>qualifying_col_expr</i> must have a sortable datatype (SMALLINT, INTEGER, BIGINT, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, VARCHAR, or CHARACTER).</p>
MIN_CHOOSE (<i>quantifying_col_expr</i> , <i>descriptive_col_expr</i> OF <i>symbol_list</i>)	<p>Returns the <i>descriptive_col_expr</i> value of the matched row with the lowest-sorted <i>qualifying_col_expr</i> value. For example, MIN_CHOOSE (product_price, product_name OF B) returns the product_name of the least expensive product in the rows that map to B.</p> <p>The <i>descriptive_col_expr</i> can have any data type. The <i>qualifying_col_expr</i> must have a sortable datatype (SMALLINT, INTEGER, BIGINT, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, VARCHAR, or CHARACTER).</p>
DUPCOUNT (<i>col_expr</i> OF <i>symbol_list</i>)	<p>Returns the duplicate count for <i>col_expr</i> in the matched rows. That is, for each matched row, the function returns the number of occurrences of the current value of <i>col_expr</i> in the immediately preceding matched row.</p> <p>When <i>col_expr</i> is also the ORDER BY <i>col_expr</i>, this function returns the equivalent of ROW_NUMBER() - RANK().</p>
DUPCOUNTCUM (<i>col_expr</i> OF <i>symbol_list</i>)	<p>Returns the cumulative duplicate count for <i>col_expr</i> in the matched rows. That is, for each matched row, the function returns the number of occurrences of the current value of <i>col_expr</i> in all preceding matched rows.</p> <p>When <i>col_expr</i> is also the ORDER BY <i>col_expr</i>, this function returns the equivalent of ROW_NUMBER() - DENSE_RANK().</p>
ACCUMULATE ([DISTINCT CDISTINCT] <i>col_expr</i> OF <i>symbol_list</i> [DELIMITER ' <i>delimiter</i> '])	<p>Returns, for each matched row, the concatenated values in <i>col_expr</i>, separated by <i>delimiter</i>.</p> <p><i>delimiter</i> is a string of LATIN characters. Its default value is ', ' (a comma followed by a space).</p> <p>DISTINCT limits the concatenated values to distinct values. CDISTINCT limits the concatenated values to consecutive distinct values.</p> <p>Accumulated string can have at most 32000 UNICODE characters or 64000 LATIN characters. If longer, function truncates string to maximum number of characters allowed.</p>

You can compute an aggregate over more than one symbol. For example, `SUM (val OF ANY (A,B))` computes the sum of the values of the attribute *val* across all rows in the matched segment that map to A or B.

nPath Results Examples

nPath Results Example: FIRST, LAST_NOTNULL, MAX_CHOOSE, and MIN_CHOOSE

Input

trans1				
userid	gender	ts	productname	productamt
1	M	2012-01-01 00:00:00	shoes	100
1	M	2012-02-01 00:00:00	books	300
1	M	2012-03-01 00:00:00	television	500
1	M	2012-04-01 00:00:00	envelopes	10
2		2012-01-01 00:00:00	bookcases	150
2		2012-02-01 00:00:00	tables	250
2	F	2012-03-01 00:00:00	appliances	1500
3	F	2012-01-01 00:00:00	chairs	400
3	F	2012-02-01 00:00:00	cellphones	600
3	F	2012-03-01 00:00:00	dvds	50

SQL Call

```
SELECT * FROM nPath (
  ON trans1 PARTITION BY userid ORDER BY ts
  USING
  Mode (NONOVERLAPPING)
  Pattern ('A+')
  Symbols (TRUE AS A)
  Result (
    FIRST (userid OF A) AS Userid,
    LAST_NOTNULL (gender OF A) AS Gender,
    MAX_CHOOSE (productamt, productname OF A) AS Max_prod,
    MIN_CHOOSE (productamt, productname OF A) AS Min_prod
```



```
)
) ORDER BY 1;
```

Output

userid	gender	max_prod	min_prod
1	M	television	envelopes
2	F	appliances	bookcases
3	F	cellphones	dvds

nPath Results Example: FIRST and Three Forms of ACCUMULATE

Input

clicks

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	?	home	06:59:13	Company1	100
1039	1	?	home	07:00:10	Company2	300
1039	1	television	checkout	07:00:12	Company2	500
1039	1	television	checkout	07:00:18	Company2	10
1039	1	envelopes	checkout	07:01:00	Company3	10
1039	1	?	checkout	07:01:10	Company3	10

SQL Call

```
SELECT * FROM nPath (
  ON clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Symbols (
    pagetype='home' AS H,
    pagetype='checkout' AS C,
    pagetype <> 'home' AND pagetype <>'checkout' AS A
  )
  Pattern ('^H+.A*.C+$')
  Result (
    FIRST (sessionid OF ANY (H, A, C)) AS sessionid,
    FIRST (clicktime OF H) AS firsthome,
```

```
    FIRST (clicktime OF C) AS firstcheckout,  
    ACCUMULATE (productname OF ANY (H,A,C) DELIMITER '*')  
AS products_accumulate,  
    ACCUMULATE (CDISTINCT productname OF ANY (H,A,C) DELIMITER '$$')  
AS cde_dup_products,  
    ACCUMULATE (DISTINCT productname OF ANY (H,A,C)) AS de_dup_products  
)  
) ORDER BY sessionid;
```

Output

sessionid	firsthome	firstcheckout	products_accumulate	cde_dup_products	de_dup_products
1	06:59:13	07:00:12	[null*null*television*television*envelopes*null]	[null\$\$television\$ \$envelopes\$\$null]	[null, television, envelopes]

nPath Results Example: FIRST, Three Forms of ACCUMULATE, COUNT, and NTH**Input**

The input table for this example is clicks, as in [nPath Results Example: FIRST and Three Forms of ACCUMULATE](#).

SQL Call

```
SELECT * FROM nPath (
  ON clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Symbols (
    pagetype='home' AS H,
    pagetype='checkout' AS C,
    pagetype <> 'home' AND pagetype <> 'checkout' AS A
  )
  Pattern ('^H+.A*.C+$')
  Result (
    FIRST (sessionid OF ANY (H, A, C)) AS sessionid,
    FIRST (clicktime OF H) AS firsthome,
    FIRST (clicktime OF C) AS firstcheckout,
    ACCUMULATE (productname OF ANY (H,A,C)) AS products_accumulate,
    COUNT (DISTINCT productname OF ANY(H,A,C)) AS count_distinct_products,
    ACCUMULATE (CDISTINCT productname OF ANY
(H,A,C)) AS consecutive_distinct_products,
    ACCUMULATE (DISTINCT productname OF ANY (H,A,C)) AS distinct_products,
    NTH (productname, -1 OF ANY(H,A,C)) AS nth
  )
) ORDER BY sessionid;
```

Output

sessionid	firsthome	firstcheckout	products_accumulate	count_distinct_products	consecutive_distinct_products	distinct_products	nth
1	06:59:13	07:00:12	[null, null, television, television, envelopes, null]	2	[null, television, envelopes, null]	[null, television, envelopes]	?

nPath Results Example: Combine Values from One Row with Values from the Next Row

Input

The input table is clickstream, as in [nPath Filters Example](#).

SQL Call

```
SELECT * FROM nPath (
  ON clickstream PARTITION BY userid ORDER BY userid, sessionid, clicktime
  USING
  Mode (OVERLAPPING)
  Pattern ('A.B')
  Symbols (TRUE AS A, TRUE AS B)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    FIRST (pagetype OF A) AS pageid,
    FIRST (pagetype OF B) AS next_pageid
  )
) ORDER BY sessionid;
```

Output

sessionid	pageid	next_pageid
1	home	view
1	view	view
1	checkout	view
1	checkout	checkout
1	view	checkout
1	view	view
2	checkout	view
2	home	view
2	view	view
2	view	checkout

nPath Results Example: Hindi Input, ACCUMULATE FIRST_NOTNULL**Input**

The example has two input tables that include Hindi characters.

हिंदी टेबल

सत्रआईडी	क्लिककरें	token	उत्पादकानाम	पेजकाप्रकार	रेफरर
1	06:59:13.000000	1		घर	गूगल डॉट कॉम
13	15:35:08.000000	15		घर	गूगल डॉट कॉम
400	10:00:00.000000	300		घर	
9000	05:30:15.000000	?		घर	
1	07:00:10.000000	11		घर	गूगल डॉट कॉम
9001	05:30:15.000000	?		लॉग इन	
400	10:05:04.000000	12	आकाशगंगा s4	चेक आउट	
9000	05:30:20.000000	?	लेनोवो g580	चेक आउट	
1	07:00:12.000000	111	ipod	चेक आउट	गूगल डॉट कॉम
9001	05:30:15.000000	?		घर	
400	10:05:03.000000	12		कागज एक	
9000	05:50:44.000000	?	लैपटॉप case	चेक आउट	
1	07:01:00.000000	1111	बोस	चेक आउट	
9001	05:30:20.000000	?	prod4	चेक आउट	
400	09:59:55.000000	12		लॉग इन	
9000	12:50:55.000000	?		लॉग आउट	
1	18:00:00.000000	10		लॉग इन	
9001	12:50:55.000000	?		लॉग आउट	
400	10:05:55.000000	18		लॉग आउट	
14	13:18:30.000000	2		घर	गूगल डॉट कॉम
1	18:00:10.000000	10		घर	
8000	16:00:00.000000	8001		लॉग इन	
400	10:05:02.000000	18		घर	
14	13:18:31.000000	8		कागज एक	

सत्रआईडी	क्लिककरें	token	उत्पादकानाम	पेजकाप्रकार	रेफरर
1	18:00:15.000000	10	आई - फ़ोन USB cable	चेक आउट	
8000	16:00:10.000000	8002		घर	
400	18:00:10.000000	18		घर	
14	13:18:32.000000	8		page2	
666	12:50:15.000000	40		घर	
8000	16:00:20.000000	8003	nexus7	चेक आउट	
400	18:05:10.000000	18	prod1	चेक आउट	
14	13:18:40.000000	20	आई - फ़ोन	चेक आउट	
666	12:50:20.000000	41	लेनोवो g580	चेक आउट	
8000	16:00:40.000000	8004		लॉग आउट	
400	18:08:10.000000	100	prod2	चेक आउट	
14	13:19:00.000000	20	बोस	चेक आउट	
666	12:50:44.000000	42	लैपटॉप case	चेक आउट	
400	18:10:10.000000	150		लॉग आउट	
14	13:20:00.000000	20	सैमसंग	चेक आउट	
666	12:50:55.000000	50		लॉग आउट	
400	09:59:45.000000	150		लॉग इन	
500	08:15:12.000000	12		लॉग इन	
10000	16:00:00.000000	1		लॉग इन	
400	09:59:40.000000	210		लॉग इन	
500	08:15:15.000000	31		घर	
10000	16:00:10.000000	1		घर	
400	09:59:55.000000	220		लॉग इन	
500	08:15:20.000000	123		कागज एक	
10000	16:00:20.000000	2	nexus7	चेक आउट	
400	09:59:55.000000	220		लॉग इन	
500	08:16:00.000000	1231	आकाशगंगा चार्जर	चेक आउट	
10000	16:00:40.000000	4		लॉग आउट	
500	08:16:30.000000	1232	हेडफोन	चेक आउट	

सत्रआईडी	क्लिक करें	token	उत्पादकानाम	पेजकाप्रकार	रेफरर
2	15:34:25.000000	333		घर	गूगल डॉट कॉम
500	08:12:12.000000	1233		लॉग इन	
2	15:34:25.000000	333		लॉग आउट	
250	20:00:01.000000	8		घर	गो डैडी डॉट कॉम
250	20:02:00.000000	80	बोस	चेक आउट	
250	20:02:50.000000	81	itrip	चेक आउट	
250	20:03:00.000000	82	आई - फोन	चेक आउट	

विज्ञापन

रिलेसमय	channel	विज्ञापन	duration
20:02:01.000000	सीएनबीसी	13	1000
15:35:06.000000	सीएनबीसी	14	1000
15:34:26.000000	food network	11	1000
13:18:42.000000	espn	12	1000
07:00:20.000000	सीएनबीसी	10	1000

SQL Call

```

SELECT * FROM nPath (
  ON "हिंदी टेबल" PARTITION BY "सत्र आईडी" ORDER BY "क्लिक करें"
  ON "विज्ञापन" DIMENSION ORDER BY "रिले समय"
  USING
  Mode (NONOVERLAPPING)
  Pattern ('^(X|A)+.C')
  Symbols ( "पेज का प्रकार" IS NULL OR "पेज का प्रकार" IS NOT NULL AS X,
    "पेज का प्रकार" = 'चेक आउट' AS C,
    "विज्ञापन" IS NULL OR "विज्ञापन" IS NOT NULL AS A
  )
  Result (
    ACCUMULATE ("रेफरर" of ANY(X)) AS "रेफरल पथ" ,
    FIRST_NOTNULL ("सत्र आईडी" of ANY(X)) AS "सत्र आईडी"
  )
) AS dt;
```

Output

रेफरल पथ	सत्रआईडी
[गूगल डॉट कॉम, याहू डॉट कॉम, याहू डॉट कॉम, , ,]	1
[,]	8000
[गूगल डॉट कॉम, , , ,]	14
[गो डैडी डॉट कॉम, ,]	250
[, , , , , , , , ,]	400
[, , ,]	500
[,]	666
[,]	9001
[,]	9000
[,]	10000

nPath Results Example: Hindi Input, ACCUMULATE DISTINCT and CDISTINCT**Input**

unicode_path

id	price	event
2	2.200000000000000E 000	ఈవెంట్ 1
2	5.133000000000000E 001	ఈవెంట్ 2
2	9.881230000000000E 002	ఈవెంట్ 3
2	-1.200000000000000E-001	ఈవెంట్ 4
2	1.100000000000000E 000	ఈవెంట్ 5
2	1.120000000000000E 001	ఈవెంట్ 5
2	1.220000000000000E 001	ఈవెంట్ 5
2	1.200000000000000E 000	ఈవెంట్ 1

SQL Call

```
SELECT * FROM nPath (
  ON unicode_path PARTITION BY id
  USING
```

```

Mode (NONOVERLAPPING)
Pattern ('A*')
Symbols (true AS A)
Result (
  ACCUMULATE (DISTINCT event OF A DELIMITER ', ' ) AS acc_result_distinct,
  ACCUMULATE (CDISTINCT event OF A DELIMITER ', ' ) AS acc_result_cdistinct
)
) AS dt;

```

Output

acc_result_distinct	acc_result_cdistinct
[ఈవటు3, ఈవటు5, ఈవటు1, ఈవటు4, ఈవటు2]	[ఈవటు3, ఈవటు5, ఈవటు1, ఈవటు4, ఈవటు2, ఈవటు1]

nPath Differences on Aster Database and Advanced SQL Engine

Aster Database nPath	Advanced SQL Engine nPath
In a symbol, the Boolean expression TRUE, NOT TRUE, or <i>integer</i> can be enclosed in parentheses or quotation marks.	In a symbol, the Boolean expression TRUE, NOT TRUE, or <i>integer</i> cannot be enclosed in parentheses or quotation marks.
Aggregate functions compare strings using Unicode value of each character (lexicographic order), ignoring CHARACTER SET.	Aggregate functions compare strings using sort order, based on CHARACTER SET, CASESPECIFIC, and COLLATION.

AVG

Database	Syntax Element Data Type	Return Data Type
Aster	SMALLINT, INTEGER, or BIGINT	NUMERIC
	DOUBLE PRECISION, NUMERIC, or INTERVAL	Same as syntax element data type
Teradata	NUMERIC	DOUBLE PRECISION
	INTERVAL, or DATE without TIME or TIMESTAMP	Same as syntax element data type

COUNT

Database	Syntax Element Data Type	Return Data Type
Aster	Any	BIGINT
Teradata	Any	TD mode: INTEGER

Database	Syntax Element Data Type	Return Data Type
		ANSI mode: Depends on MaxDecimal value in DBSControl—see following table.

MaxDecimal Value	Result Data Type	Result Data Type Format
0 or 15	NUMERIC(15,0)	-(15)9
18	NUMERIC(18,0)	-(18)9
38	NUMERIC(38,0)	-(38)9

MAX and MIN

Database	Syntax Element Data Type	Return Data Type
Aster	Any numeric, string, or DateTime type	Same as syntax element data type
Teradata	Any numeric, character, DateTime or Interval data type, or BYTE	If not UDT: Same as syntax element data type UDT: Data type to which UDT is implicitly cast

SUM

Database	Syntax Element Data Type	Return Data Type
Aster	SMALLINT or INTEGER	BIGINT
	BIGINT	NUMERIC
	DOUBLE PRECISION	DOUBLE PRECISION
	NUMERIC or INTERVAL	Same as syntax element data type
Teradata	NUMERIC, INTERVAL, or DATE without TIME or TIMESTAMP	Same as syntax element data type, except for NUMERIC(<i>n,m</i>), which returns NUMERIC(<i>p,m</i>), where <i>p</i> depends on MaxDecimal value in DBSControl—see following table.
	CHARACTER or VARCHAR	DOUBLE PRECISION

MaxDecimal Value	<i>n</i>	<i>p</i>
0 or 15	$n \leq 15$	15
	$15 < n \leq 18$	18
	$n > 18$	38
18	$n \leq 18$	18

MaxDecimal Value	<i>n</i>	<i>p</i>
	$n > 18$	38
38	Any value	38

nPath Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

Symbols and Symbol Predicates That Examples Use

Symbol	Symbol Predicate
A	pageid IN (10, 25)
B	category = 10 OR (category = 20 AND pageid <> 33)
C	category IN (SELECT pageid FROM clicks1 GROUP BY userid HAVING COUNT(*) > 10)
D	referrer LIKE '%Amazon%'
X	TRUE

nPath ClickStream Data Examples

Input

This statement creates the input table of clickstream data that the examples use:

```
CREATE MULTISET TABLE clicks1 (
  userid INTEGER,
  sessionid INTEGER,
  pageid INTEGER,
  category INTEGER,
  ts TIMESTAMP FORMAT 'YYYY-MM-DDbHH:MI:SS',
  referrer VARCHAR (256),
  val FLOAT
) PRIMARY INDEX ( userid );
```

This statement gets the pageid for each row and the pageid for the next row in sequence:

```

SELECT dt.sessionid, dt.pageid, dt.next_pageid FROM nPath (
  ON clicks1 PARTITION BY sessionid ORDER BY ts
  USING
  Mode (OVERLAPPING)
  Pattern ('A.B')
  Symbols (TRUE AS A, TRUE AS B)
  Result (
    FIRST(sessionid OF A) AS sessionid,
    FIRST (pageid OF A) AS pageid,
    FIRST (pageid OF B) AS next_pageid
  )
) AS dt;

```

Example: Counting Preceding Rows in a Sequence

For each row, this invocation counts the number of preceding rows in a given sequence (including the current row). The ORDER BY clause specifies DESC because the pattern must be matched over the rows preceding the start row, while the semantics dictate that the pattern be matched over the rows following the start row.

```

SELECT dt.sessionid, dt.pageid, dt.countrank FROM nPath (
  ON clicks1 PARTITION BY sessionid ORDER BY ts DESC
  USING
  Mode (OVERLAPPING)
  Pattern ('A*')
  Symbols (TRUE AS A)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    FIRST (pageid OF A) AS pageid,
    COUNT (* OF A) AS countrank
  )
) AS dt;

```

Example: Complex Path Query

This query finds the user click-paths that start at pageid 50 and proceed either to pageid 80 or to pages in category 9 or category 10, finds the pageid of the last page in the path, counts the visits to page 80, and returns the maximum count for each last page, by which it sorts the output. The query ignores paths of fewer than five pages and pages for which category is less than zero.

```

SELECT dt.last_pageid, MAX(dt.count_page80) FROM nPath (
  ON (SELECT * FROM clicks1 WHERE category >= 0)
  PARTITION BY sessionid ORDER BY ts
  USING
  Pattern ('A.(B|C)*')

```

```

Mode (OVERLAPPING)
Symbols (
  pageid = 50 AS A,
  pageid = 80 AS B,
  pageid <> 80 AND category IN (9,10) AS C
)
Result (
  LAST(pageid OF ANY (A,B,C)) AS last_pageid,
  COUNT (* OF B) AS count_page80,
  COUNT (* OF ANY (A,B,C)) AS count_any
)
) AS dt WHERE dt.count_any >= 5
GROUP BY dt.last_pageid
ORDER BY MAX(dt.count_page80);

```

nPath Range-Matching Examples

Whenever a user visits the home page and then visits checkout pages and buys increasingly expensive products, the nPath query returns the first purchase and the most expensive purchase.

nPath Example Input Table: aggregate_clicks

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	sneakers	home	2009-07-29 20:17:59	Company1	100
1039	2	books	home	2009-04-21 13:17:59	Company4	300
1039	3	television	home	2009-05-23 13:17:59	Company2	500
1039	4	envelopes	home	2009-07-16 11:17:59	Company3	10
1039	4	envelopes	home1	2009-07-16 11:18:16	Company3	10
1039	4	envelopes	page1	2009-07-16 11:18:18	Company3	10
1039	5	bookcases	home	2009-08-19 22:17:59	Company5	150
1039	5	bookcases	home1	2009-08-19 22:18:02	Company5	150

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	5	bookcases	page1	2009-08-19 22:18:05	Company5	150
1039	5	bookcases	page2	2009-08-22 04:20:05	Company5	150
1039	5	bookcases	checkout	2009-08-24 14:30:05	Company5	150
1039	5	bookcases	page2	2009-08-27 23:03:05	Company5	150
1040	1	tables	home	2009-07-29 20:17:59	Company5	250
1040	2	Appliance	home	2009-04-21 13:17:59	Company6	1500
1040	3	laptops	home	2009-05-23 13:17:59	Company7	800
1040	4	chairs	home	2009-07-16 11:17:59	Company3	400
1040	4	chairs	home1	2009-07-16 11:18:16	Company3	400
1040	4	chairs	page1	2009-07-16 11:18:18	Company3	400
1040	5	cellphones	home	2009-08-19 22:17:59	Company8	600
1040	5	cellphones	home1	2009-08-19 22:18:02	Company8	600
1040	5	cellphones	page1	2009-08-19 22:18:05	Company8	600
1040	5	cellphones	page2	2009-08-22 04:20:05	Company8	600
1040	5	cellphones	checkout	2009-08-24 14:30:05	Company8	600
1040	5	cellphones	page2	2009-08-27 23:03:05	Company8	600
...

nPath Range-Matching Example: Accumulate Pages Visited in Each Session

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('A*')
  Symbols (TRUE AS A)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    ACCUMULATE (pagetype OF A) AS path
  )
) AS dt ORDER BY dt.sessionid;
```

Output

sessionid	path
1	[home, home1, page1, home, home1, page1, home, home, home, home1, page1, checkout, home, home, home, home, home, home, home, home]
2	[home, home, home, home, home, home, home, home, home, home, home1, page1, checkout, checkout, home, home]
3	[home, home, home, home, home, home, home, home, home, home1, page1, home, home1, page1, home]
4	[home, home, home, home, home, home, home1, home1, home1, page1, page1, page1]
5	[home, home, home, home, home1, home1, home1, page1, page1, page1, page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]

nPath Range-Matching Example: Find Sessions That Start at Home Page and Visit Page1

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```

SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('^H.A*.P1.A*')
  Symbols (pagetype='home' AS H, pagetype='page1' AS P1, TRUE AS A)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    ACCUMULATE (pagetype OF ANY(H,P1,A)) AS path
  )
) AS dt ORDER BY dt.sessionid;

```

Output

sessionid	path
1	[home, home1, page1, home, home1, page1, home, home, home, home1, page1, checkout, home, home, home, home, home, home, home, home]
2	[home, home, home, home, home, home, home, home, home, home, home1, page1, checkout, checkout, home, home]
3	[home, home, home, home, home, home, home, home, home, home1, page1, home, home1, page1, home]
4	[home, home, home, home, home, home, home1, home1, home1, page1, page1, page1]
5	[home, home, home, home, home1, home1, home1, page1, page1, page1, page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]

nPath Range-Matching Example: Find Paths to Checkout Page for Purchases Over \$200**Input**

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```

SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('A*.C+.A*')
  Symbols (

```

```

    productprice > 200 AND
    pagetype='checkout' AS C, TRUE AS A
  )
  Result (
    FIRST(sessionid OF A) AS sessionid,
    ACCUMULATE (pagetype OF ANY(A,C)) AS path,
    AVG (productprice OF ANY(A,C)) AS totalsum
  )
) AS dt ORDER BY dt.sessionid;

```

Output

sessionid	path	totalsum
1	[home, home1, page1, home, home1, page1, home, home, home, home1, page1, checkout, home, home, home, home, home, home]	602.857142857143
5	[home, home, home, home, home1, home1, home1, page1, page1, page1, page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]	363.157894736842

nPath Range-Matching Example: Use OVERLAPPING Mode

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```

SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (OVERLAPPING)
  Pattern ('A.A')
  Symbols (TRUE AS A)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    ACCUMULATE (pagetype OF A) AS path
  )
) AS dt ORDER BY dt.sessionid;

```

nPath Range-Matching Example Output

sessionid	path
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[home, home]
1	[checkout, home]
1	[page1, checkout]
1	[home1, page1]
1	[home, home1]
1	[home, home]
1	[home, home]
1	[page1, home]
1	[home1, page1]
1	[home, home1]
1	[page1, home]
1	[home1, page1]
1	[home, home1]
2	[home, home]
2	[checkout, home]
2	[checkout, checkout]
...	...

nPath Range-Matching Example: Find First Product with Multiple Referrers in Any Session

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('REFERRER{2,}')
  Symbols (referrer IS NOT NULL AS REFERRER)
  Result (
    FIRST (sessionid OF REFERRER) AS sessionid,
    FIRST (productname OF REFERRER) AS product
  )
) AS dt ORDER BY dt.sessionid;
```

Output

sessionid	product
1	envelopes
2	tables
3	bookcases
4	tables
5	Appliances

nPath Range-Matching Example: Find Data for Sessions That Checked Out 3-6 Products

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
```

```

USING
Mode (NONOVERLAPPING)
Pattern ('H+.D*.C{3,6}.D')
Symbols (
  pagetype = 'home' AS H,
  pagetype='checkout' AS C,
  pagetype<>'home' AND pagetype<>'checkout' AS D
)
Result (
  FIRST (sessionid OF C) AS sessionid,
  max_choose (productprice, productname OF C) AS
  most_expensive_product,
  MAX (productprice OF C) AS max_price,
  min_choose (productprice, productname of C) AS
  least_expensive_product,
  MIN (productprice OF C) AS min_price)
) AS dt ORDER BY dt.sessionid;

```

Output

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
5	cellphones	600	bookcases	150

nPath Range-Matching Example: Find Data for Sessions That Checked Out at Least 3 Products

Input

The input table is aggregate_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

Modify the previous query call in [nPath Range-Matching Example: Find Data for Sessions That Checked Out 3-6 Products](#) to find sessions where the user checked out at least three products by changing the Pattern syntax element to:

```
Pattern ('H+.D*.C{3,}.D')
```

SQL-MapReduce Call

```

SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
  Mode (NONOVERLAPPING)
  Pattern ('H+.D*.C{3,}.D')
  Symbols (

```

```

pagetype = 'home' AS H,
pagetype='checkout' AS C,
pagetype<>'home' AND pagetype<>'checkout' AS D
)
Result (
  FIRST(sessionid OF C) AS sessionid,
  max_choose(productprice, productname OF C) AS
  most_expensive_product,
  MAX (productprice OF C) AS max_price,
  min_choose (productprice, productname OF C) AS
  least_expensive_product,
  MIN (productprice OF C) AS min_price
)
) AS dt ORDER BY dt.sessionid;

```

Output

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
5	cellphones	600	bookcases	150

nPath Range-Matching Example: Multiple Partitioned Input Tables and Dimension Input Table

An e-commerce store wants to count the advertising impressions that lead to a user clicking an online advertisement. The example counts the online advertisements that the user viewed and the television advertisements that the user might have viewed.

Input

impressions

userid	ts	imp
1	2012-01-01	ad1
1	2012-01-02	ad1
1	2012-01-03	ad1
1	2012-01-04	ad1
1	2012-01-05	ad1
1	2012-01-06	ad1
1	2012-01-07	ad1
2	2012-01-08	ad2

userid	ts	imp
2	2012-01-09	ad2
2	2012-01-10	ad2
2	2012-01-11	ad2
...

clicks2

userid	ts	click
1	2012-01-01	ad1
2	2012-01-08	ad2
3	2012-01-16	ad3
4	2012-01-23	ad4
5	2012-02-01	ad5
6	2012-02-08	ad6
7	2012-02-14	ad7
8	2012-02-24	ad8
9	2012-03-02	ad9
10	2012-03-10	ad10
11	2012-03-18	ad11
12	2012-03-25	ad12
13	2012-03-30	ad13
14	2012-04-02	ad14
15	2012-04-06	ad15

tv_spots

ts	tv_imp
2012-01-01	ad2
2012-01-02	ad2
2012-01-03	ad3
2012-01-04	ad4
2012-01-05	ad5

ts	tv_imp
2012-01-06	ad6
2012-01-07	ad7
2012-01-08	ad8
2012-01-09	ad9
2012-01-10	ad10
2012-01-11	ad11
2012-01-12	ad12
2012-01-13	ad13
2012-01-14	ad14
2012-01-15	ad15

SQL-MapReduce Call

The tables impressions and clicks have a user_id column, but the table tv_spots is only a record of television advertisements shown, which any user might have seen. Therefore, tv_spots must be a dimension table.

```
SELECT * FROM nPath (
  ON impressions PARTITION BY userid ORDER BY ts
  ON clicks2 PARTITION BY userid ORDER BY ts
  ON tv_spots DIMENSION ORDER BY ts
  USING
  Mode (NONOVERLAPPING)
  Symbols (TRUE AS imp, TRUE AS click, TRUE AS tv_imp)
  Pattern ('(imp|tv_imp)*.click')
  Result (
    COUNT(* of imp) AS imp_cnt,
    COUNT (* of tv_imp) AS tv_imp_cnt
  )
) AS dt ORDER BY dt.imp_cnt;
```

Output

dt.imp_cnt	tv_imp_cnt
18	0
19	0

dt.imp_cnt	tv_imp_cnt
19	0
20	0
21	0
22	0
22	0
22	0
22	0
22	0
23	0
23	0
23	0
24	0
25	0

Pack (SQL Engine)

The Pack function packs data from multiple input columns into a single column. The packed column has a virtual column for each input column. By default, virtual columns are separated by commas and each virtual column value is labeled with its column name.

Pack complements the function [Unpack \(SQL Engine\)](#), but you can use it on any columns that meet the input requirements.

Note:

To use Pack and Unpack together, you must run both on Advanced SQL Engine. Pack and Unpack are incompatible with ML Engine functions Pack_MLE and Unpack_MLE.

Before packing columns, note their data types—you need them if you want to unpack the packed column.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support locale-based formatting with the SDF file.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

Pack Syntax

```
SELECT * FROM Pack (
  ON { table | view | (query) }
  USING
  [ TargetColumns ({ 'target_column' | target_column_range }[,...]) ]
  [ Delimiter ('delimiter') ]
  [ IncludeColumnName ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
  OutputColumn ('output_column')
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  [ ColCast ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
) AS alias;
```

Related Information:

[Column Specification Syntax Elements](#)

Pack Syntax Elements

TargetColumns

[Optional] Specify the names of the input table columns to pack into a single output column. Column names must be valid object names, which are defined in *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

These names become the column names of the virtual columns. If you specify this syntax element, but do not specify all input table columns, the function copies the unspecified input table columns to the output table.

Default behavior: All input table columns are packed into a single output column.

Delimiter

[Optional] Specify the delimiter—a single Unicode character in Normalization Form C (NFC)—that separates the virtual columns in the packed data. The *delimiter* is case-sensitive.

Default: ',' (comma)

IncludeColumnName

[Optional] Specify whether to label each virtual column value with its column name (making the virtual column *target_column:value*).

Default: 'true'

OutputColumn

Specify the name to give to the packed output column. The name must be a valid object name, as defined in *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

Accumulate

[Optional] Specify the input columns to copy to the output table.

ColCast

[Optional] Specify whether to cast each numeric *target_column* to VARCHAR.

Specifying 'true' decreases run time for queries with numeric target columns.

Default: false

Pack Input

Input Table Schema

Column	Data Type	Description
<i>target_column</i>	Any	Column to pack, with other input columns, into single output column.
<i>accumulate_column</i> or <i>other_input_column</i>	Any	Column to copy to output table. Typically, one such column contains row identifiers.

Pack Output

Output Table Schema

Column	Data Type	Description
<i>output_column</i>	VARCHAR If a column of type DATE is packed into the output, its format is 'YYYY-MM-DD'.	Packed column.
<i>accumulate_column</i>	Nonnumeric column or numeric column with ColCast ('false'): Same as in input table Numeric column with ColCast ('true') : VARCHAR	Column copied from input table.

Column	Data Type	Description
<i>other_input_column</i>	Same as in input table	[Column appears only without Accumulate, once for each specified <i>other_input_column</i> .] Column copied from input table.

Pack Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

Pack Examples Input

The input table, `ville_temperature` contains temperature readings for the cities Nashville and Knoxville, in the state of Tennessee.

ville_temperature

sn	city	state	period	temp_f
1	Nashville	Tennessee	2010-01-01 00:00:00	35.1
2	Nashville	Tennessee	2010-01-01 01:00:00	36.2
3	Nashville	Tennessee	2010-01-01 02:00:00	34.5
4	Nashville	Tennessee	2010-01-01 03:00:00	33.6
5	Nashville	Tennessee	2010-01-01 04:00:00	33.1
6	Knoxville	Tennessee	2010-01-01 03:00:00	33.2
7	Knoxville	Tennessee	2010-01-01 04:00:00	32.8
8	Knoxville	Tennessee	2010-01-01 05:00:00	32.4
9	Knoxville	Tennessee	2010-01-01 06:00:00	32.2
10	Knoxville	Tennessee	2010-01-01 07:00:00	32.4

Pack Example: Default Options

This example specifies the default options for `Delimiter` and `IncludeColumnName`.

Input

See [Pack Examples Input](#).

SQL Call

```
SELECT * FROM Pack (
  ON ville_temperature
  USING
  Delimiter (',')
  OutputColumn ('packed_data')
  IncludeColumnName ('true')
  TargetColumns ('[1:4]')
  Accumulate ('sn')
) AS dt ORDER BY 2;
```

Output

The columns specified by TargetColumns are packed in the column packed_data. Virtual columns are separated by commas, and each virtual column value is labeled with its column name. The input column sn, which was not specified by TargetColumns, is unchanged in the output table.

packed_data	sn
city:Nashville,state:Tennessee,period:2010-01-01 00:00:00,temp_f:35.1	1
city:Nashville,state:Tennessee,period:2010-01-01 01:00:00,temp_f:36.2	2
city:Nashville,state:Tennessee,period:2010-01-01 02:00:00,temp_f:34.5	3
city:Nashville,state:Tennessee,period:2010-01-01 03:00:00,temp_f:33.6	4
city:Nashville,state:Tennessee,period:2010-01-01 04:00:00,temp_f:33.1	5
city:Knoxville,state:Tennessee,period:2010-01-01 03:00:00,temp_f:33.2	6
city:Knoxville,state:Tennessee,period:2010-01-01 04:00:00,temp_f:32.8	7
city:Knoxville,state:Tennessee,period:2010-01-01 05:00:00,temp_f:32.4	8
city:Knoxville,state:Tennessee,period:2010-01-01 06:00:00,temp_f:32.2	9
city:Knoxville,state:Tennessee,period:2010-01-01 07:00:00,temp_f:32.4	10

Pack Example: Nondefault Options

This example specifies the pipe character (|) for Delimiter and 'false' for IncludeColumnName.

Input

See [Pack Examples Input](#).

SQL Call

```
SELECT * FROM Pack (
  ON ville_temperature
  USING
  Delimiter ('|')
  OutputColumn ('packed_data')
  IncludeColumnName ('false')
  TargetColumns ('city', 'state', 'period', 'temp_f')
) AS dt ORDER BY 2;
```

Output

Virtual columns are separated by pipe characters and not labeled with their column names.

packed_data	sn
Nashville Tennessee 2010-01-01 00:00:00 35.1	1
Nashville Tennessee 2010-01-01 01:00:00 36.2	2
Nashville Tennessee 2010-01-01 02:00:00 34.5	3
Nashville Tennessee 2010-01-01 03:00:00 33.6	4
Nashville Tennessee 2010-01-01 04:00:00 33.1	5
Knoxville Tennessee 2010-01-01 03:00:00 33.2	6
Knoxville Tennessee 2010-01-01 04:00:00 32.8	7
Knoxville Tennessee 2010-01-01 05:00:00 32.4	8
Knoxville Tennessee 2010-01-01 06:00:00 32.2	9
Knoxville Tennessee 2010-01-01 07:00:00 32.4	10

Sessionize (SQL Engine)

The Sessionize function maps each click in a session to a unique session identifier. A *session* is a sequence of clicks by one user that are separated by at most *n* seconds.

The function is useful for both sessionization and detecting web crawler ("bot") activity. A typical use is to understand user browsing behavior on a web site.

Sessionize Syntax

```
SELECT * FROM Sessionize (
  ON { table | view | (query) }
```

```

PARTITION BY expression [,...]
ORDER BY order_column [,...]
USING
TimeColumn ('time_column')
TimeOut (session_timeout)
[ ClickLag (min_human_click_lag) ]
[ EmitNull ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})]
) AS alias;

```

Sessionize Syntax Elements

TimeColumn

Specify the name of the input column that contains the click times.

The *time_column* must also be an *order_column*.

TimeOut

Specify the number of seconds at which the session times out. If *session_timeout* seconds elapse after a click, the next click starts a new session. The data type of *session_timeout* is DOUBLE PRECISION.

ClickLag

[Optional] Specify the minimum number of seconds between clicks for the session user to be considered human. If clicks are more frequent, indicating that the user is a bot, the function ignores the session. The *min_human_click_lag* must be less than *session_timeout*. The data type of *min_human_click_lag* is DOUBLE PRECISION.

Default behavior: The function ignores no session, regardless of click frequency.

EmitNull

[Optional] Specify whether to output rows that have NULL values in their session id and rapid fire columns, even if their *time_column* has a NULL value.

Default: 'false'

Sessionize Input

Input Table Schema

Column	Data Type	Description
<i>time_column</i>	TIME, TIMESTAMP, INTEGER, BIGINT, SMALLINT, or DATE	Click times (in milliseconds if data type is INTEGER, BIGINT, or SMALLINT).

Column	Data Type	Description
<i>partition_column</i>	Any	Column by which input data is partitioned. Input data must be partitioned such that each partition contains all rows of an entity.
<i>order_column</i>	Any	Column by which input data is ordered.

No input table column can have the name 'sessionid' or 'clicklag', because these are output table column names.

Tip:

To create a single timestamp column from separate date and time columns:

```
SELECT (datecolumn || ' ' || timecolumn)::timestamp AS
mytimestamp FROM table;
```

Sessionize Output

Output Table Schema

Column	Data Type	Description
<i>input_column</i>	Same as in input table	Column copied from input table. Function copies every <i>input_column</i> to output table.
sessionid	INTEGER or BIGINT	Identifier that function assigned to session.
clicklag	BYTEINT	'1' if the session exceeded <i>min_human_click_lag</i> , '0' otherwise.

Sessionize Example

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

Input

sessionize_table

partition_id	clicktime	userid	productname	pagetype	referrer	productprice
1	1110000	333		Home	www.yahoo.com	
1	1112000	333	Ipod	Checkout	www.yahoo.com	200.2
1	1160000	333	Bose	Checkout		340

partition_id	clicktime	userid	productname	pagetype	referrer	productprice
1	1200000	333		Home	www.google.com	
1	1203000	67403		Home	www.google.com	
1	1300000	67403		Home	www.google.com	
1	1301000	67403		Home		
1	1302000	67403		Home		
1	1340000	67403	Iphone	Checkout		650
1	1450000	67403	Bose	Checkout		750
1	1450200	80000		Home	godaddy.com	
1	1450600	80000	Bose	Checkout		340
1	1450800	80000	ltrip	Checkout		450
1	1452000	880000	Iphone	Checkout		650

SQL Call

```

SELECT * FROM Sessionize (
  ON sessionize_table PARTITION BY partition_id ORDER BY clicktime
  USING
    TimeColumn ('clicktime')
    TimeOut (60)
    ClickLag (0.2)
) ORDER BY partition_id, clicktime;

```

Output

partition_id	clicktime	userid	productname	pagetype	referrer	productprice	SESSIONID	CLICKLAG
1	1110000	333	?	Home	www.yahoo.com	?	0	f
1	1112000	333	Ipod	Checkout	www.yahoo.com	200.2	0	f
1	1160000	333	Bose	Checkout	?	340	0	f
1	1200000	333	?	Home	www.google.com	?	0	f
1	1203000	67403	?	Home	www.google.com	?	0	f
1	1300000	67403	?	Home	www.google.com	?	1	f
1	1301000	67403	?	Home	?	?	1	f
1	1302000	67403	?	Home	?	?	1	f
1	1340000	67403	Iphone	Checkout	?	650	1	f

partition_id	clicktime	userid	productname	pagetype	referrer	productprice	SESSIONID	CLICKLAG
1	1450000	67403	Bose	Checkout	?	750	2	f
1	1450200	80000	?	Home	godaddy.com	?	2	t
1	1450600	80000	Bose	Checkout	?	340	2	f
1	1450800	80000	ltrip	Checkout	?	450	2	t
1	1452000	880000	Iphone	Checkout	?	650	2	f

StringSimilarity (SQL Engine)

The StringSimilarity function calculates the similarity between two strings, using the specified comparison method. The similarity is a value in the range [0, 1].

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- When comparing strings, the function assumes that they are in the same Unicode script in Normalization Form C (NFC).
- When used with this function, the ORDER BY clause supports only ASCII collation.

StringSimilarity Syntax

```
SELECT * FROM StringSimilarity (
  ON { table | view | (query) } [ PARTITION BY ANY ]
  USING
  ComparisonColumnPairs ('comparison_type (column1,column2[,constant])[ AS
output_column]' [,...])
  [ CaseSensitive ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}[,...]) ]
```

```
[ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related Information:

[Column Specification Syntax Elements](#)

StringSimilarity Syntax Elements

ComparisonColumnPairs

Specify the names of the input table columns that contain strings to compare (*column1* and *column2*), how to compare them (*comparison_type*), and (optionally) a constant and the name of the output column for their similarity (*output_column*). The similarity is a value in the range [0, 1].

For *column1* and *column2*:

- If *column1* or *column2* includes any special characters (that is, characters other than letters, digits, or underscore (_)), surround the column name with double quotation marks. For example, if *column1* and *column2* are c(col1) and c(col2), respectively, specify them as "c(col1)" and "c(col2)".

If *column1* or *column2* includes double quotation marks, replace each double quotation mark with a pair of double quotation marks. For example, if *column1* and *column2* are c1"c and c2"c, respectively, specify them as "c1""c" and "c2""c".

Note:

These rules do not apply to *output_column*. For example, this is valid syntax:
ComparisonColumnPairs ('jaro ("c1""c", "c2""c") AS out"col')

- If *column1* or *column2* supports more than 200 characters, you can cast it to VARCHAR(200), as in the following example; however, the string may be truncated. For information about the CAST operation, see *Teradata Vantage™ - SQL Functions, Expressions, and Predicates*, B035-1145.

```
SELECT * FROM StringSimilarity (
  ON (
    SELECT id, CAST(a AS VARCHAR(200)) AS a, CAST(b AS
  VARCHAR(200)) AS b
    FROM max_varchar_strlen
  ) PARTITION BY ANY
  USING
  ComparisonColumnPairs ('ld(a,b) AS sim_fn')
  Accumulate ('id')
) AS dt ORDER BY 1;
```

For *comparison_type*, use one of these values:

<i>comparison_type</i>	Description
'jaro'	Jaro distance.
'jaro_winkler'	Jaro-Winkler distance: 1 for an exact match, 0 otherwise. If you specify this comparison type, you can specify the value of factor <i>p</i> with <i>constant</i> . $0 \leq p \leq 0.25$. Default: $p = 0.1$
'n_gram'	<i>N</i> -gram similarity. If you specify this comparison type, you can specify the value of <i>N</i> with <i>constant</i> . Default: $N = 2$
'LD'	Levenshtein distance: Number of edits needed to transform one string into the other. Edits are insertions, deletions, or substitutions of individual characters.
'LDWS'	Levenshtein distance without substitution: Number of edits needed to transform one string into the other using only insertions or deletions of individual characters.
'OSA'	Optimal string alignment distance: Number of edits needed to transform one string into the other. Edits are insertions, deletions, substitutions, or transpositions of characters. A substring can be edited only once.
'DL'	Damerau-Levenshtein distance: Like 'OSA' except that a substring can be edited any number of times.
'hamming'	Hamming distance: For strings of equal length, number of positions where corresponding characters differ (that is, minimum number of substitutions needed to transform one string into the other). For strings of unequal length, -1.
'LCS'	Longest common substring: Length of longest substring common to both strings.
'jaccard'	Jaccard indexed-based comparison.
'cosine'	Cosine similarity.
'soundexcode'	Only for English strings: -1 if either string has a non-English character; otherwise, 1 if their soundex codes are the same and 0 otherwise.

The function ignores *constant* for every *comparison_type* except 'jaro_winkler' and 'n_gram'.

You can specify a different *comparison_type* for every pair of columns.

Default: *output_column* is 'sim_*i*', where *i* is the sequence number of the column pair.

CaseSensitive

[Optional] Specify whether string comparison is case-sensitive. You can specify either one value for all pairs or one value for each pair. If you specify one value for each pair, the *i*th value applies to the *i*th pair.

Default: 'false'

Accumulate

[Optional] Specify the names of input table columns to copy to the output table.

StringSimilarity Input

Column	Data Type	Description
<i>column1</i>	CHARACTER or VARCHAR	String to compare to string in <i>column2</i> .
<i>column2</i>	CHARACTER or VARCHAR	String to compare to string in <i>column1</i> .
<i>accumulate_column</i>	Any	Column to copy to output table.

If any *column1* or *column2* in the input table schema supports more than 200 characters, you must cast it to VARCHAR(200). See example in [StringSimilarity Syntax Elements](#).

StringSimilarity Output

Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Any	Column copied from input table.
<i>output_column</i>	DOUBLE PRECISION	Similarity between strings in column pair.

StringSimilarity Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

StringSimilarity Example: Specify Column Names

Input

strsimilarity_input

id	src_text1	src_text2	tar_text
1	astre	astter	aster
2	hone	fone	phone
3	acqiese	acquire	acquiesce

id	src_text1	src_text2	tar_text
4	AAAACCCCCGGGGA	CCCGGGAACCAACC	CCAGGGAAACCCAC
5	alice	allen	allies
6	angela	angle	angels
7	senter	center	centre
8	chef	cheap	chief
9	circus	circle	circuit
10	debt	debut	debris
11	deal	dell	lead
12	bare	bear	bear

SQL Call

```

SELECT * FROM StringSimilarity (
  ON strsimilarity_input PARTITION BY ANY
  USING
  ComparisonColumnPairs ('jaro (src_text1, tar_text) AS jaro1_sim',
                        'LD (src_text1, tar_text) AS ld1_sim',
                        'n_gram (src_text1, tar_text, 2) AS ngram1_sim',
                        'jaro_winkler (src_text1, tar_text, 0.1) AS jw1_sim'
  )
  CaseSensitive ('true')
  Accumulate ('id', 'src_text1', 'tar_text')
) AS dt ORDER BY id;

```

Output

Columns 1-3

id	src_text1	tar_text
1	astre	aster
2	hone	phone
3	acqiese	acquiesce
4	AAAACCCCCGGGGA	CCAGGGAAACCCAC
5	alice	allies
6	angela	angels
7	senter	centre

id	src_text1	tar_text
8	chef	chief
9	circus	circuit
10	debt	debris
11	deal	lead
12	bare	bear

Columns 4-7

jaro1_sim	ld1_sim	ngram1_sim	jw1_sim
0.9333333333333333	0.6	0.5	0.9533333333333333
0.9333333333333333	0.8	0.75	0.9333333333333333
0.925925925925926	0.777777777777778	0.5	0.948148148148148
0.824175824175824	0.214285714285714	0.384615384615385	0.824175824175824
0.822222222222222	0.5	0.4	0.857777777777778
0.888888888888889	0.833333333333333	0.8	0.933333333333333
0.822222222222222	0.5	0.4	0.822222222222222
0.933333333333333	0.8	0.5	0.946666666666667
0.849206349206349	0.714285714285714	0.666666666666667	0.90952380952381
0.75	0.5	0.4	0.825
0.666666666666667	0.5	0.333333333333333	0.666666666666667
0.833333333333333	0.5	0.333333333333333	0.85

StringSimilarity Example: Specify Column Ranges**Input**

The input table is strsimilarity_input, as in [StringSimilarity Example: Specify Column Names](#).

SQL Call

```
SELECT * FROM StringSimilarity (
  ON strsimilarity_input PARTITION BY ANY
  USING
  ComparisonColumnPairs ('jaro (src_text1, tar_text) AS jaro1_sim',
```



```

        'LD (src_text1, tar_text) AS ld1_sim',
        'n_gram (src_text1, tar_text, 2) AS ngram1_sim',
        'jaro_winkler (src_text1, tar_text, 0.1) AS jw1_sim'
    )
    CaseSensitive ('true')
    Accumulate ('[0:1]', 'tar_text')
) AS dt ORDER BY id;

```

Output

SVMSparsePredict (SQL Engine)

Note:

This *namePredict* function uses the model output by ML Engine *name* function to analyze the input data and make predictions.

If the SVMSparse call that created the model specified HashProjection ('true'), SVMSparsePredict does not support UNICODE data.

If your model table was created using a supported version of Aster Analytics on Aster Database, see [AA 7.00 Usage Notes](#).

SVMSparsePredict Syntax

```

SELECT * FROM SVMSparsePredict (
  ON { table | view | (query) } AS InputTable PARTITION BY id_column
  ON { table | view | (query) } AS Model DIMENSION
  USING
  IDColumn ('id_column')
  AttributeNameColumn ('attribute_name_column')
  [ AttributeValueColumn ('attribute_value_column') ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  [ TopK ({ output_class_number | 'output_class_number' }) ]
  [ OutputProb ({'true'|'t'|"yes"|"y"|"1"|"false"|"f"|"no"|"n"|"0"}) ]

```

```
[ Responses ('response' [,...]) ]
) AS alias;
```

Related Information:

[Column Specification Syntax Elements](#)

SVMSparsePredict Syntax Elements

IDColumn

Specify the name of the InputTable column that contains the identifiers of the test samples. The InputTable must be partitioned by this column.

AttributeNameColumn

Specify the name of the InputTable column that contains the attributes of the test samples.

AttributeValueColumn

[Optional] Specify the name of the InputTable column that contains the attribute values.

Default behavior: Each attribute has the value 1.

Accumulate

[Optional] Specify the names of the InputTable columns to copy to the output table.

TopK

[Disallowed with Responses, otherwise optional] Specify the number of class labels to appear in the output table. For each observation, the output table has n rows, corresponding to the n most likely classes. To see the probability of each class, use OutputProb ('true').

OutputProb

[Required to be 'true' with Responses, optional otherwise.] Specify whether to output the probability for each response. If you omit Responses, the function outputs only the probability of the predicted class.

Default: 'true'

Responses

[Optional] Specify the classes for which to output probabilities.

Default behavior: Output only the probability of the predicted class.

SVMSParsePredict Input

Table	Description
InputTable	Contains test data.
Model	Output by ML Engine SVMSParse function. Model is in binary format. To display its readable content, use ML Engine SVMSParseSummary function.

InputTable Schema

Column	Data Type	Description
<i>id_column</i>	BYTEINT, INTEGER, SMALLINT, BIGINT, NUMERIC, NUMERIC(p), NUMERIC(p,a), VARCHAR, or VARCHAR(n)	Test sample identifier.
<i>attribute_name_column</i>	BYTEINT, INTEGER, SMALLINT, BIGINT, VARCHAR, or VARCHAR(n)	Test sample attribute.
<i>attribute_value_column</i>	BYTEINT, INTEGER, SMALLINT, BIGINT, NUMERIC, NUMERIC(p), or NUMERIC(p,a)	Attribute value.
<i>accumulate_column</i>	Any	Column to copy to output table.

Model Table Schema

Column	Data Type	Description
classid	BYTEINT, INTEGER, SMALLINT, or BIGINT	Identifier of class of model attribute.
weights	BYTE, VARBYTE, or BLOB	Weight of model attribute.

SVMSParsePredict Output

Output Table Schema

The table has the predicted class of each test sample.

If you specify TopK (n), the output table has n rows for each observation.

Column	Data Type	Description
<i>id_column</i>	BYTEINT, INTEGER, SMALLINT, BIGINT, NUMERIC, NUMERIC(p), NUMERIC(p, a), VARCHAR, or VARCHAR(n)	Test sample identifier.

Column	Data Type	Description
predict_value	VARCHAR	Predicted class of test sample.
predict_confidence	DOUBLE PRECISION	[Column appears only with OutputProb ('true') and without Responses syntax element] Probability that observation belongs to class in predict_value column.
prob_response	DOUBLE PRECISION	[Column appears only without Responses syntax element, and appears once for each specified response] Probability that observation belongs to category response.
accumulate_column	Any	Column copied from InputTable.

Calculation of prob_response and predict_confidence

The function calculates the values of prob_response and predict_confidence with the following formulas.

This is the formula for the value of class r :

$$value_r = W_r \cdot X$$

where:

- X is the vector of predictor values corresponding to an observation.
- W_r is the vector of predictor weights calculated by the model for class r , where r is a class specified by the Responses syntax element.

prob_response

For binary classification, the formula for the probability that a response belongs to class r is:

$$prob_response = \text{sigmoid}(value_r)$$

For multiple-class classification, the formula for the probability that a response belongs to class r is:

$$prob_response = \text{softmax}_r(values)$$

predict_confidence

The column predict_confidence, which appears only if you omit the Responses syntax element, displays the probability that the observation belongs to the class in the column predict_value. This value is the maximum value of prob_response over all responses r .

SVMsparsePredict Example

Input

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment ① in the left sidebar.

- InputTable: svm_iris_input_test
- Model: svm_iris_model, output by ML Engine SVMSparse function

The model is in binary format. To display its readable content, use ML Engine `SVMSparseSummary` function.

svm_iris_input_test

id	species	attribute	value
5	setosa	sepal_length	5.0
5	setosa	sepal_width	3.6
5	setosa	petal_length	1.4
5	setosa	petal_width	0.2
10	setosa	sepal_length	4.9
10	setosa	sepal_width	3.1
10	setosa	petal_length	1.5
10	setosa	petal_width	0.1
15	setosa	sepal_length	5.8
15	setosa	sepal_width	4.0
15	setosa	petal_length	1.2
15	setosa	petal_width	0.2
...

svm_iris_model[illegible]

classid	weights
1	3FE4F1C5871DE4A0C000B12D7E8C18FE3FE558515B291C5DBFF6F2558D9050370000000000000000
2	3FF9424250696DEA4003BA4B98AB24FCBFF5FBB2667D07A7BFF1D36766E2FE0A0000000000000000

SQL Call

```
SELECT * FROM SVMsparsePredict (
  ON svm_iris_input_test AS InputTable PARTITION BY id
  ON svm_iris_model AS Model DIMENSION
  USING
  IDColumn ('id')
  AttributeNameColumn ('attribute')
  AttributeValueColumn ('value')
  Accumulate ('species')
) AS dt;
```

Output

This query returns the following table:

```
SELECT * FROM svm_iris_predict_out ORDER BY id;
```

id	predict_value	predict_confidence	species
5	setosa	9.47345262648877E-001	setosa
10	setosa	8.46460012681134E-001	setosa
15	setosa	9.76489773801754E-001	setosa
...

Prediction Accuracy

This query returns the prediction accuracy:

```
SELECT (SELECT count(id)
FROM svm_iris_predict_out
WHERE predict_value = species)/(1.00*(
  SELECT count(id) FROM svm_iris_predict_out)) AS prediction_accuracy;
```

prediction_accuracy
0.83

Unpack (SQL Engine)

The Unpack function unpacks data from a single packed column into multiple columns. The packed column is composed of multiple virtual columns, which become the output columns. To determine the virtual columns, the function must have either the delimiter that separates them in the packed column or their lengths.

Unpack complements the function [Pack \(SQL Engine\)](#), but you can use it on any packed column that meets the input requirements.

Note:

- To use Pack and Unpack together, you must run both on Advanced SQL Engine. Pack and Unpack are incompatible with ML Engine functions Pack_MLE and Unpack_MLE.
- This function requires the UTF8 client character set for UNICODE data.
- This function does not support the following:
 - Locale-based parsing with the SDF file
 - This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.
-

Unpack Syntax

```
SELECT * FROM Unpack (
  ON { table | view | (query) }
  USING
  TargetColumn ('target_column')
  OutputColumns ('output_column' [,...])
  OutputDataTypes ('datatype' [,...])
  [ Delimiter ('delimiter') ]
  [ ColumnLength ('column_length' [,...] ) ]
  [ Regex ('regular_expression') ]
  [ RegexSet ('group_number') ]
  [ IgnoreInvalid ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
```

```
[ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

Related Information:

[Column Specification Syntax Elements](#)

Unpack Syntax Elements

TargetColumn

Specify the name of the input column that contains the packed data.

OutputColumns

Specify the names to give to the output columns, in the order in which the corresponding virtual columns appear in *target_column*. The names must be valid object names, as defined in *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

If you specify fewer output column names than there are virtual input columns, the function ignores the extra virtual input columns. That is, if the packed data contains $x+y$ virtual columns and the OutputColumns syntax element specifies x output column names, the function assigns the names to the first x virtual columns and ignores the remaining y virtual columns.

OutputDataTypes

Specify the datatypes of the unpacked output columns. Supported data types are VARCHAR, INTEGER, DOUBLE PRECISION, TIME, DATE, and TIMESTAMP.

If OutputDataTypes specifies only one value and OutputColumns specifies multiple columns, the specified value applies to every *output_column*.

If OutputDataTypes specifies multiple values, it must specify a value for each *output_column*. The *n*th *datatype* corresponds to the *n*th *output_column*.

The function can output only 16 VARCHAR columns.

Delimiter

[Optional] Specify the delimiter—a single Unicode character in Normalization Form C (NFC)—that separates the virtual columns in the packed data. The *delimiter* is case-sensitive.

Do not specify both this syntax element and the ColumnLength syntax element. If the virtual columns are separated by a delimiter, specify the delimiter with this syntax element; otherwise, specify the ColumnLength syntax element.

Default: ',' (comma)

ColumnLength

[Optional] Specify the lengths of the virtual columns; therefore, to use this syntax element, you must know the length of each virtual column.

If ColumnLength specifies only one value and OutputColumns specifies multiple columns, the specified value applies to every *output_column*.

If ColumnLength specifies multiple values, it must specify a value for each *output_column*. The *n*th *datatype* corresponds to the *n*th *output_column*. However, the last *output_column* can be an asterisk (*), which represents a single virtual column that contains the remaining data. For example, if the first three virtual columns have the lengths 2, 1, and 3, and all remaining data belongs to the fourth virtual column, you can specify ColumnLength ('2', '1', '3', *).

Do not specify both this syntax element and the Delimiter syntax element.

Regex

[Optional] Specify a regular expression that describes a row of packed data, enabling the function to find the data values.

A row of packed data contains a data value for each virtual column, but the row might also contain other information (such as the virtual column name). In the *regular_expression*, each data value is enclosed in parentheses.

For example, suppose that the packed data has two virtual columns, age and sex, and that one row of packed data is age:34, sex:ma1e. The *regular_expression* that describes the row is `'.*:(.*)'`. The `'.*:'` matches the virtual column names, age and sex, and the `'(.)'` matches the values, 34 and male.

To represent multiple data groups in *regular_expression*, use multiple pairs of parentheses. Without parentheses, the last data group in *regular_expression* represents the data value (other data groups are assumed to be virtual column names or unwanted data). If a different data group represents the data value, specify its group number with the RegexSet syntax element.

Default: `'(.)'`, which matches the whole string (between delimiters, if any). When applied to the preceding sample row, the default *regular_expression* causes the function to return 'age:34' and 'sex:ma1e' as data values.

RegexSet

[Optional] Specify the ordinal number of the data group in *regular_expression* that represents the data value in a virtual column.

Default behavior: The last data group in *regular_expression* represents the data value. For example, suppose that *regular_expression* is `'([a-zA-Z]*):(.*)'`. If *group_number* is '1', `'([a-zA-Z]*)'` represents the data value. If *group_number* is '2', `'(.)'` represents the data value.

Maximum: 30

IgnoreInvalid

[Optional] Specify whether the function ignores rows that contain invalid data.

IgnoreInvalid may not behave as you expect if an item in a virtual column has trailing special characters. See [Unpack Example: IgnoreInvalid \('true'\) with Trailing Special Characters](#).

Default: 'false' (The function fails if it encounters a row with invalid data.)

Accumulate

[Optional] Specify the input columns to copy to the output table.

Unpack Input

Input Table Schema

Column	Data Type	Description
<i>target_column</i>	CHARACTER, VARCHAR, or CLOB	Packed data.
<i>accumulate_column</i> or <i>other_input_column</i>	Any	[Column appears zero or more times.] Column to copy to output table. Typically, one such column contains row identifiers.

Unpack Output

Output Table Schema

Column	Data Type	Description
<i>output_column</i>	Specified by OutputDataTypes syntax element.	Unpacked column.
<i>accumulate_column</i>	Same as in input table	Column copied from input table.
<i>other_input_column</i>	Same as in input table	[Column appears only without Accumulate, once for each input table column not specified by TargetColumn syntax element.] Column copied from input table.

Unpack Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment ① in the left sidebar.

Unpack Example: Delimiter Separates Virtual Columns

Input

The input table, ville_tempdata, is a collection of temperature readings for two cities, Nashville and Knoxville, in the state of Tennessee. In the column of packed data, the delimiter comma (,) separates the virtual columns. The last row contains invalid data.

ville_tempdata

sn	packed_temp_data
10	Nashville,Tennessee,35.1
11	Nashville,Tennessee,36.2
12	Nashville,Tennessee,34.5
13	Nashville,Tennessee,33.6
14	Nashville,Tennessee,33.1
15	Nashville,Tennessee,33.2
16	Nashville,Tennessee,32.8
17	Nashville,Tennessee,32.4
18	Nashville,Tennessee,32.2
19	Nashville,Tennessee,32.4
20	Thisisbaddata

SQL Call

Because comma is the default delimiter, the Delimiter syntax element is optional.

```
SELECT * FROM Unpack (
  ON ville_tempdata
  USING
  TargetColumn ('packed_temp_data')
  OutputColumns ('city', 'state', 'temp_f')
  OutputDataTypes ('varchar', 'varchar', 'real')
```

```

Delimiter (',')
Regex ('(.*?)')
RegexSet (1)
IgnoreInvalid ('true')
Accumulate ('sn')
) AS dt ORDER BY sn;

```

Output

Because of IgnoreInvalid ('true'), the function did not fail when it encountered the row with invalid data, but it did not output that row.

city	state	temp_f	sn
Nashville	Tennessee	3.510000000000000E 001	10
Nashville	Tennessee	3.620000000000000E 001	11
Nashville	Tennessee	3.450000000000000E 001	12
Nashville	Tennessee	3.360000000000000E 001	13
Nashville	Tennessee	3.310000000000000E 001	14
Nashville	Tennessee	3.320000000000000E 001	15
Nashville	Tennessee	3.280000000000000E 001	16
Nashville	Tennessee	3.240000000000000E 001	17
Nashville	Tennessee	3.220000000000000E 001	18
Nashville	Tennessee	3.240000000000000E 001	19

Unpack Example: No Delimiter Separates Virtual Columns

Input

The input table, ville_tempdata1, is like the input table for the previous example, except that no delimiter separates the virtual columns in the packed data. To enable the function to determine the virtual columns, the function call specifies the column lengths.

ville_tempdata1

sn	packed_temp_data
10	NashvilleTennessee35.1
11	NashvilleTennessee36.2
12	NashvilleTennessee34.5

sn	packed_temp_data
13	NashvilleTennessee33.6
14	NashvilleTennessee33.1
15	NashvilleTennessee33.2
16	NashvilleTennessee32.8
17	NashvilleTennessee32.4
18	NashvilleTennessee32.2
19	NashvilleTennessee32.4
20	Thisisbaddata

SQL Call

```
SELECT * FROM Unpack (
  ON ville_tempdata1
  USING
    TargetColumn ('packed_temp_data')
    OutputColumns ('city', 'state', 'temp_f')
    OutputDataTypes ('varchar', 'varchar', 'real')
    ColumnLength ('9', '9', '4')
    Regex ('(.*)')
    RegexSet (1)
    IgnoreInvalid ('true')
) AS dt ORDER BY sn;
```

Output

city	state	temp_f	sn
Nashville	Tennessee	3.510000000000000E 001	10
Nashville	Tennessee	3.620000000000000E 001	11
Nashville	Tennessee	3.450000000000000E 001	12
Nashville	Tennessee	3.360000000000000E 001	13
Nashville	Tennessee	3.310000000000000E 001	14
Nashville	Tennessee	3.320000000000000E 001	15
Nashville	Tennessee	3.280000000000000E 001	16
Nashville	Tennessee	3.240000000000000E 001	17

city	state	temp_f	sn
Nashville	Tennessee	3.22000000000000E 001	18
Nashville	Tennessee	3.24000000000000E 001	19

Unpack Example: More Input Columns than Output Columns

Input

The input table is ville_tempdata1, as in [Unpack Example: No Delimiter Separates Virtual Columns](#). Its packed_temp_data column has three virtual columns.

SQL Call

The OutputColumns syntax element specifies only two output column names.

```
SELECT * FROM Unpack (
  ON ville_tempdata1
  USING
    TargetColumn ('packed_temp_data')
    OutputColumns ('city', 'state')
    OutputDataTypes ('varchar', 'varchar')
    ColumnLength ('9', '9')
    Regex ('(.*)')
    RegexSet (1)
    IgnoreInvalid ('true')
) AS dt ORDER BY sn;
```

Output

The output table has columns for the first two virtual input columns, but not for the third.

city	state	sn
Nashville	Tennessee	10
Nashville	Tennessee	11
Nashville	Tennessee	12
Nashville	Tennessee	13
Nashville	Tennessee	14
Nashville	Tennessee	15
Nashville	Tennessee	16

city	state	sn
Nashville	Tennessee	17
Nashville	Tennessee	18
Nashville	Tennessee	19

Unpack Example: IgnoreInvalid ('true') with Trailing Special Characters

In this example, the items in the first virtual input column have trailing special characters. No delimiter separates the virtual columns. ColumnLength is 2. The call to Unpack includes IgnoreInvalid ('true'), but the output is unexpected.

Input

t2
c1
1,1919-04-05
1.1919-04-05
5,.1919-04-05
2,2019/04/05
4,.1919-04-05
32019/04/05

SQL Call

```

SEL * FROM Unpack (
  ON t2
  USING
  TargetColumn ('c1')
  OutputColumns ('a','b')
  OutputDataTypes ('int','date')
  ColumnLength ('2','*')
  IgnoreInvalid ('True')
) AS dt;

```

Output

a	b
1	19/04/05
1	19/04/05
2	19/04/05

The reason for the unexpected output is the behavior of an internal library that Unpack uses, which is as follows:

Input Row	Behavior
1,1919-04-05	Library prunes trailing comma, converts "1" to integer and "1919-04-05" to date. (Output row 1.)
1.1919-04-05	Library prunes trailing period, converts "1" to integer and "1919-04-05" to date. (Output row 2.)
5,.1919-04-05	Library prunes trailing comma, converts 5 to integer, but cannot convert ".1919-04-05" to date. (No output row.) With ColumnLength ('3','*'), library prunes trailing comma and period, converts "5" to integer and "1919-04-05" to date, and outputs a row for this input row.
2,2019/04/05	Library prunes trailing comma, converts "2" to integer and "1919/04/05" to date. (Output row 3.)
4,.,1919-04-05	Library converts "4." to integer, but cannot convert ",.1919-04-05" to date. (No output row.)
32019/04/05	Library converts "32" to integer, but cannot convert "019/04/05" to date. (No output row.)

Data Cleaning Functions

Data cleaning functions prepare the input data set for the next set of transformations.

TD_ConvertTo

TD_ConvertTo converts the specified input table columns to specified data types.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_ConvertTo Syntax

```
SELECT * FROM TD_ConvertTo (
  ON { table | view | (query) } AS InputTable
  USING
  TargetColumns ({ 'target_column' | target_column_range }[,...])
  TargetDataType ('target_datatype' [,...])
) AS alias;
```

TD_ConvertTo Syntax Elements

TargetColumns

Specify the names of the InputTable columns to convert to another data type.

TargetDataType

Specify either a single target data type for all target columns or a target data type for each target column. If you specify multiple target data types, the function assigns the *n*th target data type to the *n*th target column.

Allowed Values for the TargetDataType Element	Output Data Type
BYTEINT	BYTEINT

Allowed Values for the TargetDataType Element	Output Data Type										
SMALLINT	SMALLINT										
INTEGER	INTEGER										
BIGINT	BIGINT										
REAL	REAL										
DECIMAL	DECIMAL (38,19)										
VARCHAR	<p>Depends on input data type:</p> <table> <tr> <th>Input Data Type</th><th>Output Data Type</th></tr> <tr> <td>VARCHAR</td><td>VARCHAR with same CHARLEN, CHARACTER SET, and CASESPECIFIC values as input data type.</td></tr> <tr> <td>CHAR</td><td>VARCHAR(32000) with same CHARACTER SET and CASESPECIFIC values as input data type.</td></tr> <tr> <td>CLOB</td><td>VARCHAR(32000) with same CHARACTER SET as input data type, NOT CASESPECIFIC</td></tr> <tr> <td>Other</td><td>VARCHAR(32000), CHARACTER SET UNICODE, NOT CASESPECIFIC.</td></tr> </table>	Input Data Type	Output Data Type	VARCHAR	VARCHAR with same CHARLEN, CHARACTER SET, and CASESPECIFIC values as input data type.	CHAR	VARCHAR(32000) with same CHARACTER SET and CASESPECIFIC values as input data type.	CLOB	VARCHAR(32000) with same CHARACTER SET as input data type, NOT CASESPECIFIC	Other	VARCHAR(32000), CHARACTER SET UNICODE, NOT CASESPECIFIC.
Input Data Type	Output Data Type										
VARCHAR	VARCHAR with same CHARLEN, CHARACTER SET, and CASESPECIFIC values as input data type.										
CHAR	VARCHAR(32000) with same CHARACTER SET and CASESPECIFIC values as input data type.										
CLOB	VARCHAR(32000) with same CHARACTER SET as input data type, NOT CASESPECIFIC										
Other	VARCHAR(32000), CHARACTER SET UNICODE, NOT CASESPECIFIC.										
VARCHAR(charlen= <i>len</i> , charset={LATIN UNICODE}, casespecific={YES NO})	VARCHAR(<i>len</i>) with CHARACTER SET charset value, CASESPECIFIC casespecific value.										
CHAR	<p>Depends on input data type:</p> <table> <tr> <th>Input Data Type</th><th>Output Data Type</th></tr> <tr> <td>CHAR</td><td>CHAR with same CHARLEN, CHARACTER SET, and CASESPECIFIC values as input data type.</td></tr> <tr> <td>VARCHAR</td><td>CHAR(32000) with same CHARACTER SET and CASESPECIFIC values as input data type.</td></tr> <tr> <td>CLOB</td><td>CHAR(32000) with same CHARACTER SET as input data type, NOT CASESPECIFIC</td></tr> </table>	Input Data Type	Output Data Type	CHAR	CHAR with same CHARLEN, CHARACTER SET, and CASESPECIFIC values as input data type.	VARCHAR	CHAR(32000) with same CHARACTER SET and CASESPECIFIC values as input data type.	CLOB	CHAR(32000) with same CHARACTER SET as input data type, NOT CASESPECIFIC		
Input Data Type	Output Data Type										
CHAR	CHAR with same CHARLEN, CHARACTER SET, and CASESPECIFIC values as input data type.										
VARCHAR	CHAR(32000) with same CHARACTER SET and CASESPECIFIC values as input data type.										
CLOB	CHAR(32000) with same CHARACTER SET as input data type, NOT CASESPECIFIC										

Allowed Values for the TargetDataType Element	Output Data Type				
	<table> <tr> <th>Input Data Type</th><th>Output Data Type</th></tr> <tr> <td>Other</td><td>CHAR(32000), CHARACTER SET UNICODE, NOT CASESPECIFIC.</td></tr> </table>	Input Data Type	Output Data Type	Other	CHAR(32000), CHARACTER SET UNICODE, NOT CASESPECIFIC.
Input Data Type	Output Data Type				
Other	CHAR(32000), CHARACTER SET UNICODE, NOT CASESPECIFIC.				
CHAR(charlen= <i>len</i> , charset={LATIN UNICODE}, casespecific={YES NO})	CHAR(<i>len</i>) with CHARACTER SET charset value, CASESPECIFIC casespecific value.				
DATE	DATE FORMAT 'YY/MM/DD'				
TIME	TIME(6)				
TIMESTAMP	TIMESTAMP(6)				
TIME WITH ZONE	TIME(6) WITH TIMEZONE				
TIMESTAMP WITH ZONE	TIMESTAMP(6) WITH TIMEZONE				
INTERVAL YEAR	INTERVAL YEAR(4)				
INTERVAL MONTH	INTERVAL MONTH(4)				
INTERVAL DAY	INTERVAL DAY(4)				
INTERVAL HOUR	INTERVAL HOUR(4)				
INTERVAL MINUTE	INTERVAL MINUTE(4)				
INTERVAL SECOND	INTERVAL SECOND(4,6)				
INTERVAL YEAR TO MONTH	INTERVAL YEAR(4) TO MONTH				
INTERVAL DAY TO HOUR	INTERVAL DAY(4) TO HOUR				
INTERVAL DAY TO MINUTE	INTERVAL DAY(4) TO MINUTE				
INTERVAL DAY TO SECOND	INTERVAL DAY(4) TO SECOND(6)				
INTERVAL HOUR TO MINUTE	INTERVAL HOUR(4) TO MINUTE				
INTERVAL HOUR TO SECOND	INTERVAL HOUR(4) TO SECOND(6)				
INTERVAL MINUTE TO SECOND	INTERVAL MINUTE(4) TO SECOND(6)				
CLOB	<p>Depends on input data type:</p> <table> <tr> <th>Input Data Type</th><th>Output Data Type</th></tr> <tr> <td>CLOB</td><td>CLOB with same CHARLEN and CHARACTER SET value as input data type.</td></tr> </table>	Input Data Type	Output Data Type	CLOB	CLOB with same CHARLEN and CHARACTER SET value as input data type.
Input Data Type	Output Data Type				
CLOB	CLOB with same CHARLEN and CHARACTER SET value as input data type.				

Allowed Values for the TargetDataType Element	Output Data Type	
	Input Data Type	Output Data Type
	VARCHAR or CHAR	CLOB(1048544000) with same CHARACTER SET as input data type.
	Other	CLOB(1048544000), CHARACTER SET UNICODE.
CLOB(charlen= <i>len</i> , charset={LATIN UNICODE})	CLOB(<i>len</i>) with CHARACTER SET charset value.	
BYTE	BYTE(32000)	
BYTE(charlen= <i>len</i>)	BYTE(<i>len</i>)	
VARBYTE	VARBYTE(32000)	
VARBYTE(charlen= <i>len</i>)	VARBYTE(<i>len</i>)	
BLOB	BLOB(2097088000)	
BLOB(charlen= <i>len</i>)	BLOB(<i>len</i>)	
JSON	JSON(32000), CHARACTER SET UNICODE.	
XML	XML(2097088000) INLINE LENGTH 4046	

TD_ConvertTo Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	Any	Column to convert to <i>target_datatype</i> .

TD_ConvertTo Output

Output Table Schema

Column	Data Type	Description
<i>target_column</i>	<i>target_datatype</i>	Column converted to <i>target_datatype</i> .
<i>input_column</i>	Same as in InputTable	Column copied from InputTable.

TD_ConvertTo Example

InputTable: input_table

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

passenger	survived	pclass	name	sex	age		
sibsp	parch	ticket	fare	cabin	embarked		
0	0	97	0	1	Goldschmidt; Mr. George B	male	71
0	0	488	0	1	Kent; Mr. Edward Austin	male	58
0	0	505	1	1	Maioni; Miss. Roberta	female	16
0	0	631	1	1	Barkworth; Mr. Algernon Henry Wilson	male	80
0	0	873	0	1	Carlsson; Mr. Frans Olof	male	33
0	0	695	5	B51 B53 B55	S		

SQL Call

```
SELECT * FROM TD_ConvertTo (
  ON input_table AS InputTable
  USING
    TargetColumns ('fare')
    TargetDataType ('integer')
) AS dt ORDER BY 1;
```

Output

passenger	survived	pclass	name	sex	age	sibsp	
parch	ticket	fare	cabin	embarked			
0	0	97	0	1	Goldschmidt; Mr. George B	male	71
0	0	488	0	1	Kent; Mr. Edward Austin	male	58
0	0	11771	29	B37	C		

	505	1	1 Maioni; Miss. Roberta	female	16
0	0 110152	86 B79	S		
	631	1	1 Barkworth; Mr. Algernon Henry Wilson	male	80
0	0 27042	30 A23	S		
	873	0	1 Carlsson; Mr. Frans Olof	male	33
0	0 695	5 B51 B53 B55 S			

TD_GetRowsWithoutMissingValues

TD_GetRowsWithoutMissingValues displays the rows that have non-NULL values in the specified input table columns.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

Related Information:

[TD_GetRowsWithMissingValues](#)

TD_GetRowsWithoutMissingValues Syntax

```
SELECT * FROM TD_GetRowsWithoutMissingValues (
  ON { table | view | (query) } AS InputTable
  [ PARTITION BY ANY [ORDER BY order_column ] ]
  [ USING
    TargetColumns ({ 'target_column' | target_column_range }[,...])
  ]
) AS alias;
```

TD_GetRowsWithoutMissingValues Syntax Elements

TargetColumns

[Optional] Specify the target column names to check for non-Null values.

Default: If omitted, the function considers all columns of the Input table.

TD_GetRowsWithoutMissingValues Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	Any	Columns for which non-NULL values are checked.

TD_GetRowsWithoutMissingValues Output

Output Table Schema

Same as InputTable schema

TD_GetRowsWithoutMissingValues Example

InputTable: input_table

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 0 in the left sidebar.

```

passenger survived pclass name                sex  age  sibsp
parch ticket   fare      cabin      embarked
-----
1      1      0      3 Braund; Mr. Owen Harris      male  22
0      0 A/5 21171 7.25      null      S
      30      0      3 Todoroff; Mr. Lalio      male  null
0      0 349216 7.8958      null      S
      505      1      1 Maioni; Miss. Roberta      female 16
0      0 110152 86.5      B79      S
      631      1      1 Barkworth; Mr. Algernon Henry Wilson male  80
0      0 27042 30      A23      S
      873      0      1 Carlsson; Mr. Frans Olof      male  33
0      0 695 5      B51 B53 B55      S

```

SQL Call

```

SELECT * FROM TD_getRowsWithoutMissingValues (
  ON input_table AS InputTable
  USING

```

```
TargetColumns ('[name:cabin]')
) AS dt;
```

Output

passenger	survived	pclass	name	sex	age	sibsp
parch	ticket	fare	cabin	embarked		
	505	1	1 Maioni; Miss. Roberta	female	16	
0	0 110152	86.5	B79	S		
	631	1	1 Barkworth; Mr. Algernon Henry Wilson	male	80	
0	0 27042	30	A23	S		
	873	0	1 Carlsson; Mr. Frans Olof	male	33	
0	0 695	5	B51 B53 B55 S			

TD_OutlierFilterFit

TD_OutlierFilterFit function calculates the lower_percentile, upper_percentile, count of rows, and median for the specified input table columns. The calculated values for each column help the TD_OutlierFilterTransform function detect outliers in the input table.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_OutlierFilterFit Syntax

```
CREATE TABLE fit_table AS (
  SELECT * FROM TD_OutlierFilterFit (
    ON { table | view | (query) } AS InputTable
    USING
    TargetColumns ({ 'target_column' | target_column_range }[,...])
    [ GroupColumns ('group_column') ]
    OutlierMethod ({ 'percentile' | 'tukey' | 'carling' })
```



```

LowerPercentile (min_value)
UpperPercentile (max_value)
[ IQRMultiplier (k) ]
ReplacementValue ({ 'delete' | 'null' | 'median' | replacement_value})
[ RemoveTail ({ 'both' | 'upper' | 'lower' }) ]
PercentileMethod ({ 'PercentileCont' | 'PercentileDISC' })
) AS alias
) WITH DATA;

```

TD_OutlierFilterFit Syntax Elements

TargetColumns

Specify the names of the numeric InputTable columns for which to compute metrics.

GroupColumns

[Optional] Specify the name of the InputTable column by which to group the input data.

Default behavior: Function does not group input data.

OutlierMethod

Specify one of these methods for filtering outliers:

Method	Values Outside This Range Are Outliers
percentile	$[min_value, max_value]$.
tukey	$[Q1 - k*(Q3-Q1), Q1 + k*(Q3-Q1)]$ where: Q1 = 25th quartile of data Q3 = 75th quartile of data k = interquartile range multiplier (see IQRMultiplier)
carling	$Q2 \pm c*(Q3-Q1)$ where: Q2 = median of data Q1 = 25th quartile of data Q3 = 75th quartile of data $c = (17.63*r - 23.64) / (7.74*r - 3.71)$ r = count of rows in <i>group_column</i> if you specify GroupColumns, otherwise count of rows in InputTable

LowerPercentile

Specify a lower range of percentile to use to detect whether the value is an outlier.

Value 0 to 1 is supported. For Tukey and Carling, use 0.25 as the lower percentile.

UpperPercentile

Specify a upper range of percentile to use to detect whether the value is an outlier.

Value 0 to 1 is supported. For Tukey and Carling, use 0.75 as the upper percentile.

IQRMultiplier

Specify interquartile range multiplier (IQR), k , for Tukey filtering.

The IQR is an estimate of the spread (dispersion) of the data in the target columns ($IQR = |Q3 - Q1|$).

Use $k = 1.5$ for moderate outliers and $k = 3.0$ for serious outliers.

Default: 1.5

ReplacementValue

Specify how to handle outliers:

Option	Description
delete	Do not copy row to output table.
null	Copy row to output table, replacing each outlier with NULL.
median	Copy row to output table, replacing each outlier with median value for its group.
<i>replacement_value</i> (Must be numeric.)	Copy row to output table, replacing each outlier with <i>replacement_value</i> .

RemoveTail

[Optional] Specify whether to remove the upper tail, the lower tail, or both.

Default: both

PercentileMethod

Specify either the PercentileCont or the PercentileDISC method for calculating the upper and lower percentiles of the input data values.

TD_OutlierFilterFit Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	NUMERIC	The input table column names for computing metrics and filtering outliers using the TD_OUTLIERFILTERTRANSFORM function.
<i>group_column</i>	Any	[Optional] Column by which to group input data.

TD_OutlierFilterFit Output

FitTable (*fit_table*) Schema

Column	Data Type	Description
TD_OutlierMethod_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	Value of OutlierMethod ('percentile', 'tukey', or 'carling').
<i>group_column</i>	Same as in Input table	[Column appears only if you specify GroupColumns.] Column by which input data is grouped.
TD_IQRMultiplier_OFTFIT	NUMERIC	Value of IQRMultiplier (<i>k</i>).
TD_RemoveTail_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	Value of RemoveTail ('both', 'upper', or 'lower').
TD_ReplacementValue_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	Value of ReplacementValue ('delete', 'null', 'median', or <i>replacement_value</i>).
TD_MinThreshold_OFTFIT	NUMERIC	Value of LowerPercentile (<i>min_value</i>).
TD_MaxThreshold_OFTFIT	NUMERIC	Value of UpperPercentile (<i>max_value</i>).
TD_AttributeValue_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	[Column appears once for each specified <i>target_column</i> .] <i>target_column</i>
TD_CountValue_OFTFIT	NUMERIC	Count of rows in <i>group_column</i> if you specify GroupColumns, otherwise count of rows in input table.
TD_MedianValue_OFTFIT	NUMERIC	Median values for target columns.
TD_LowerPercentile_OFTFIT	NUMERIC	Lower percentile of input data values, calculated by method specified by PercentileMethod (PercentileCont or PercentileDISC).

Column	Data Type	Description
TD_UpperPercentile_OFTFIT	NUMERIC	Upper percentile of input data values, calculated by method specified by PercentileMethod.

TD_OutlierFilterFit Example

InputTable: titanic

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

```

passenger pclass fare      survived
-----
1         3   7.250000000      0
2         1  71.283300000      1
3         3   7.925000000      1
4         1  53.100000000      1
5         3   8.050000000      0

```

SQL Call

```

CREATE TABLE outlier_fit AS (
  SELECT * FROM TD_OutlierFilterFit (
    ON titanic AS InputTable
    USING
    TargetColumns ('Fare')
    LowerPercentile (0.1)
    UpperPercentile (0.9)
    OutlierMethod ('Percentile')
    ReplacementValue ('median')
    PercentileMethod ('PercentileCont')
  ) AS dt
) WITH DATA;

```

Output

```

TD_OUTLIERMETHOD_OFTFIT TD_IQRMULTIPLIER_OFTFIT TD_REMOVETAIL_OFTFIT
TD_REPLACEMENTVALUE_OFTFIT TD_MINTHRESHOLD_OFTFIT TD_MAXTHRESHOLD_OFTFIT
TD_ATTRIBUTEVALUE_OFTFIT TD_COUNTVALUE_OFTFIT TD_MEDIANVALUE_OFTFIT
TD_LOWERPERCENTILE_OFTFIT TD_UPPERPERCENTILE_OFTFIT
-----
-----

```

```

-----
PERCENTILE                                1.500000000 BOTH
MEDIAN                                    0.100000000
0.900000000 fare                          5
8.050000000                               7.520000000 64.009980000

```

TD_OutlierFilterTransform

TD_OutlierFilterTransform filters outliers from the input table. The metrics for determining outliers come from [TD_OutlierFilterFit](#) output.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_OutlierFilterTransform Syntax

TD_OutlierFilterFit Call without GroupColumns

```

SELECT * FROM TD_OutlierFilterTransform (
  ON { table | view | (query) } AS InputTable PARTITION BY ANY
  ON { table | view | (query) } AS FitTable DIMENSION
) AS alias;

```

TD_OutlierFilterFit Call with GroupColumns

```

SELECT * FROM TD_OutlierFilterTransform (
  ON { table | view | (query) } AS InputTable PARTITION BY group_column
  ON { table | view | (query) } AS FitTable PARTITION BY group_column
) AS alias;

```

TD_OutlierFilterTransform Input

InputTable Schema

See [TD_OutlierFilterFit Input](#).

FitTable Schema

See [TD_OutlierFilterFit Output](#).

TD_OutlierFilterTransform Output

Column	Data Type	Description
<i>target_column</i>	NUMERIC	Column for which metrics have been computed.
OtherColumns	Any	The columns from the input table excluding the target columns are displayed.

TD_OutlierFilterTransform Example

Input

- InputTable: titanic, as in [TD_OutlierFilterFit Example](#)
- FitTable: outlier_fit, created by [TD_OutlierFilterFit Example](#)

SQL Call

```
SELECT * FROM TD_OutlierFilterTransform (
  ON titanic AS InputTable PARTITION BY ANY
  ON outlier_fit AS FitTable DIMENSION
) AS dt;
```

Output

```
passenger pclass fare      survived
-----
      1      3  8.050000000      0
      2      1  8.050000000      1
      3      3  7.925000000      1
      4      1 53.100000000      1
      5      3  8.050000000      0
```

TD_SimpleImputeFit

Imputation is the process of replacing missing values with substitute values.

TD_SimpleImputeFit outputs a table of values to substitute for missing values in the input table. The output table is input to [TD_SimpleImputeTransform](#), which makes the substitutions.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_SimpleImputeFit Syntax

```
SELECT * FROM TD_SimpleImputeFit (
  ON { table | view | (query) } AS InputTable
  [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table) ]
  USING
  { { literal_specification | stats_specification } |

    literal_specification
    stats_specification
  }
) AS alias;
```

literal_specification

```
ColsForLiterals ({ 'literal_column' | literal_column_range } [,...])
Literals ('literal' [,...])
```

stats_specification

```
ColsForStats ({ 'stats_column' | stats_column_range } [,...])
Stats ('statistic' [,...])
[ PartitionColumn ('partition_column') ]
```

TD_SimpleImputeFit Syntax Elements

OutputTable

[Optional] Specify a name for the output table.

If you omit OutputTable, you must create the output table for TD_SimpleImputeTransform with a CREATE TABLE AS statement:

```
CREATE TABLE output_table AS (
  SELECT * FROM TD_SimpleImputeFit ( ... ) AS alias
) WITH DATA;
```

ColsForLiterals

[Optional] Specify the names of the InputTable columns in which to find missing values to replace with specified literal values.

Literals

[Optional] Specify the literal values to substitute for missing values in the columns specified by ColsForLiterals. A *literal* must not exceed 128 characters.

The function maps each *literal* to the column in the same position in ColsForLiterals. For example, ColsForLiterals ('[1:5]', '-[2:4]', '[3]') specifies the column with index 3 last, so the function maps the last specified *literal* to it.

ColsForStats

[Optional] Specify the names of the InputTable columns in which find missing values to replace with specified statistics.

Stats

[Optional] Specify the statistics to substitute for missing values in the columns specified by ColsForStats.

For numeric columns, the value of the *Stats* argument must be one of the following values:

- MIN
- MAX
- MEAN
- MEDIAN

For columns with the following data types, the value of the *Stats* argument can be MODE:

- CHARACTER
- VARCHAR
- BYTEINT
- SMALLINT
- INTEGER

CHARACTER and VARCHAR values must not exceed 128 characters.

The output of MODE is the value that appeared in the last according to the alphabetical order, in case of a tie.

The function maps the value of each *Stats* argument to the column in the same position in *ColsForStats*. For example, *ColsForStats* ('[1:5]', '-[2:4]', '[3]') specifies the column with index 3 last, so the function maps the last specified *Stats* argument to it.

PartitionColumn

[Optional] Specify the name of the InputTable column on which to partition the input.

Default behavior: The function treats all rows as a single partition.

TD_SimpleImputeFit Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	CHAR, VARCHAR, or (with CHARACTER SET LATIN or UNICODE) or numeric	Column in which to find missing values.

TD_SimpleImputeFit Output

FitTable Schema

Column	Data Type	Description
TD_INDEX_SIMFIT	INTEGER	Unique row identifier.
TD_TARGETCOLUMN_SIMFIT	VARCHAR (CHARACTER SET UNICODE)	Target column name (<i>literal_column</i> or <i>stats_column</i>).
TD_NUM_COLVAL_SIMFIT	NUMERIC	If column is numeric, value substituted for missing value; otherwise NULL.
TD_STR_COLVAL_SIMFIT	VARCHAR (CHARACTER SET UNICODE)	If column is nonnumeric, value substituted for missing value or MODE value of column; otherwise NULL.
TD_ISNUMERIC_SIMFIT	BYTEINT	1 if column is numeric, otherwise 0.
<i>partition_column</i>	Same as in InputTable	Column on which input is partitioned.

TD_SimpleImputeFit Example

InputTable: simpleimpute_fit_input

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

passenger	pclass	sex	fare	survived
1	3	male	725.320000000	0
2	1	female	712.250000000	1
3	null	female	null	1
4	1	null	531.780000000	1
5	3	male	805.210000000	0

SQL Call

```
CREATE TABLE fit_table AS (
  SELECT * FROM TD_SimpleImputeFit (
    ON simpleimpute_fit_input AS InputTable
    USING
    ColsForLiterals ('Pclass')
    Literals ('2')
    ColsForStats ('Sex','Fare')
    Stats ('mode','median')
  ) AS dt
) WITH DATA;
```

Output

TD_INDEX_SIMFIT	TD_TARGETCOLUMN_SIMFIT	TD_NUM_COLVAL_SIMFIT	TD_STR_COLVAL_SIMFIT	TD_ISNUMERIC_SIMFIT
1	pclass	2.000000000		
2	sex		null	
3	fare	718.785000000		

TD_SimpleImputeTransform

SimpleImputeTransform substitutes specified values for missing values in the input table. The specified values come from [TD_SimpleImputeFit](#) output.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.
-

TD_SimpleImputeTransform Syntax

TD_SimpleImputeTransform syntax depends on whether the TD_SimpleImputeFit call that output FitTable omitted or specified PartitionColumn.

TD_SimpleImputeFit Call Omitted PartitionColumn

```
SELECT * FROM TD_SimpleImputeTransform (
  ON { table | view | (query) } AS InputTable PARTITION BY ANY
  ON { table | view | (query) } AS FitTable DIMENSION
) AS alias;
```

TD_SimpleImputeFit Call Specified PartitionColumn

```
SELECT * FROM TD_SimpleImputeTransform (
  ON { table | view | (query) } AS InputTable PARTITION BY partition_column
  ON { table | view | (query) } AS FitTable DIMENSION PARTITION BY partition_column
) AS alias;
```

TD_SimpleImputeTransform Input

InputTable Schema

See [TD_SimpleImputeFit Input](#).

FitTable Schema

See [TD_SimpleImputeFit Output](#).

TD_SimpleImputeTransform Output

Output Table Schema

Column	Data Type	Description
<i>target_column</i>	Same as in InputTable	Column in which missing values have been replaced..

TD_SimpleImputeTransform Example

Input

FitTable: fit_table created by [TD_SimpleImputeFit Example](#)

SQL Call

```
SELECT * FROM TD_SimpleImputeTransform (
  ON simpleimputetransform_test AS InputTable
  ON fit_table AS FitTable DIMENSION
) AS dt;
```

Output

```
passenger pclass sex    fare          survived
-----
          5      3 male  805.210000000    0
          4      1 male  531.780000000    1
          3      2 female 718.785000000    1
          1      3 male  725.320000000    0
          2      1 female 712.250000000    1
```

Data Exploration Functions

Data exploration functions help you learn about the variables (columns) of the input data set.

TD_CategoricalSummary

TD_CategoricalSummary displays the distinct values and their counts for each specified input table column.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_CategoricalSummary Syntax

```
SELECT * FROM TD_CategoricalSummary (
  ON { table | view | (query) } AS InputTable
  USING
  TargetColumns ({ 'target_column' | target_column_range }[,...])
) AS alias;
```

TD_CategoricalSummary Syntax Elements

TargetColumns

Specify the names of the InputTable columns for which to display the distinct values and their counts.

TD_CategoricalSummary Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE)	Column for which to display distinct values and their counts.

TD_CategoricalSummary Output

Output Table Schema

Column	Data Type	Description
ColumnName	VARCHAR (CHARACTER SET UNICODE)	Name of <i>target_column</i> .
DistinctValue	VARCHAR (CHARACTER SET UNICODE)	Name of distinct value in <i>target_column</i> . Table has one row for each distinct value.
DistinctValueCount	BIGINT	Count of distinct value in <i>target_column</i> . Table has one row for each distinct value.

TD_CategoricalSummary Example

InputTable: cat_titanic_train

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

```

passenger survived pclass name                                sex   age sibsp
parch ticket  fare      cabin      embarked
-----
0      97      0      1 Goldschmidt; Mr. George B            male   71
0      0 PC 17754 34.6542      A5      C
0      488      0      1 Kent; Mr. Edward Austin            male   58
0      0 11771  29.7      B37      C
0      505      1      1 Maioni; Miss. Roberta            female 16
0      0 110152 86.5      B79      S
0      631      1      1 Barkworth; Mr. Algernon Henry Wilson male   80
0      0 27042  30      A23      S
0      873      0      1 Carlsson; Mr. Frans Olof            male   33
0      0 695      5      B51 B53 B55 S

```

SQL Call

```

SELECT * FROM TD_CategoricalSummary (
  ON cat_titanic_train AS InputTable
  USING

```

```
TargetColumns ('sex')
) AS dt;
```

Output

ColumnName	DistinctValue	DistinctValueCount
sex	female	1
sex	male	4

TD_ColumnSummary

TD_ColumnSummary displays the following for each specified input table column:

- Column name
- Column data type
- Count of these values:
 - Non-NULL
 - NULL
 - Blank (all space characters) (NULL for numeric data type)
 - Zero (NULL for nonnumeric data type)
 - Positive (NULL for nonnumeric data type)
 - Negative (NULL for nonnumeric data type)
- Percentage of NULL values
- Percentage of non-NULL values

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.
-

TD_ColumnSummary Syntax

```
SELECT * FROM TD_ColumnSummary (
  ON { table | view | (query) } AS InputTable
  USING
```

```
TargetColumns ({ 'target_column' | target_column_range }[,...])
) AS alias;
```

TD_ColumnSummary Syntax Elements

TargetColumns

Specify the names of the InputTable columns to summarize.

TD_ColumnSummary Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	Any	Column for which to display summary.

TD_ColumnSummary Output

Output Table Schema

Column	Data Type	Description
ColumnName	VARCHAR (CHARACTER SET UNICODE)	Name of <i>target_column</i> .
DataType	VARCHAR (CHARACTER SET LATIN)	Data type of <i>target_column</i> .
NonNullCount	BIGINT	Count of non-NULL values in <i>target_column</i> .
NullCount	BIGINT	Count of NULL values in <i>target_column</i> .
BlankCount	BIGINT	If data type is char or varchar, count of blank values (values with all space characters) in <i>target_column</i> ; otherwise NULL.
ZeroCount	BIGINT	If DataType is numeric, count of zero values in <i>target_column</i> ; otherwise NULL.
PositiveCount	BIGINT	If DataType is numeric, count of positive values in <i>target_column</i> ; otherwise NULL.
NegativeCount	BIGINT	If DataType is numeric, count of negative values in <i>target_column</i> ; otherwise NULL.
NullPercentage	DOUBLE PRECISION	Percentage of NULL values in <i>target_column</i> .
NonNullPercentage	DOUBLE PRECISION	Percentage of non-NULL values in <i>target_column</i> .

TD_ColumnSummary Example

InputTable: col_titanic_train

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 0 in the left sidebar.

passenger	survived	pclass	name	sex	age	sibsp
parch	ticket	fare	cabin	embarked		
	49	0	3 Samaan; Mr. Youssef	male	null	
2	0 2662	21.679	null C			
	78	0	3 Moutal; Mr. Rahamin Haim	male	null	
0	0 374746	8.05	null S			
	505	1	1 Maioni; Miss. Roberta	female	16	
0	0 110152	8.65	B79 S			
	631	1	1 Barkworth; Mr. Algernon Henry Wilson	male	80	
0	0 27042	30	A23 S			
	873	0	1 Carlsson; Mr. Frans Olof	male	33	
0	0 695	5	B51 B53 B55 S			

SQL Call

```
SELECT * FROM TD_ColumnSummary (
  ON col_titanic_train AS InputTable
  USING
    TargetColumns ('age','pclass','embarked','cabin')
) AS dt;
```

Output

ColumnName	Datatype	NonNullCount	NullCount	BlankCount	
ZeroCount	PositiveCount	NegativeCount	NullPercentage	NonNullPercentage	
age	INTEGER	3	2		
null	0	3	0	4.00E+001	6.00E+001
cabin	VARCHAR(20) CHARACTER SET LATIN	3	2		
0 null	null	null	4.00E+001	6.00E+001	
embarked	VARCHAR(20) CHARACTER SET LATIN	5	0		

0	null	null	null	0.00E000	1.00E+002
pclass	INTEGER			5	0
null	0	5	0	0.00E000	1.00E+002

TD_GetRowsWithMissingValues

TD_GetRowsWithMissingValues displays the rows that have NULL values in the specified input table columns.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

Related Information:

[TD_GetRowsWithoutMissingValues](#)

TD_GetRowsWithMissingValues Syntax

```
SELECT * FROM TD_GetRowsWithMissingValues (
  ON { table | view | (query) } AS InputTable
  [ PARTITION BY ANY [ORDER BY order_column ] ]
  [ USING
    TargetColumns ({ 'target_column' | target_column_range }[,...])
  ]
) AS alias;
```

TD_GetRowsWithMissingValues Syntax Elements

TargetColumns

[Optional] Specify the target column names to check for Null values.

Default: If omitted, the function considers all columns of the Input table.

TD_GetRowsWithMissingValues Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	Any	Columns for which NULL values are checked.

TD_GetRowsWithMissingValues Output

Output Table Schema

Same as InputTable schema

TD_GetRowsWithMissingValues Example

InputTable: input_table

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment ① in the left sidebar.

```

passenger survived pclass name                sex  age
sibsp parch ticket   fare      cabin      embarked
-----
-----
1      1      0      3 Braund; Mr. Owen Harris      male  22
0      0 A/5 21171  7.25      null      S
      30      0      3 Todoroff; Mr. Lalio      male  null
0      0 349216  7.8958      null      S
      505      1      1 Maioni; Miss. Roberta      female 16
0      0 110152  86.5      B79      S
      631      1      1 Barkworth; Mr. Algernon Henry Wilson male  80
0      0 27042  30      A23      S
      873      0      1 Carlsson; Mr. Frans Olof      male  33
0      0 695      5      B51 B53 B55 S

```

SQL Call

```

SELECT * FROM TD_getRowsWithMissingValues (
  ON input_table AS InputTable
  USING

```

```
TargetColumns ('[name:cabin]')
) AS dt;
```

Output

```
passenger survived pclass name                sex age  sibsp parch
ticket    fare    cabin embarked              -----
-----
          1         0      3 Braund; Mr. Owen Harris male  22    1    0 A/5
21171  7.25    null  S
          30         0      3 Todoroff; Mr. Lalio   male null    0    0
349216  7.8958 null  S
```

TD_Histogram

TD_Histogram calculates the frequency distribution of a data set using your choice of these methods:

- Sturges
- Scott
- Variable-width
- Equal-width

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_Histogram Syntax

```
SELECT * TD_Histogram (
  ON { table | view | (query) } AS InputTable
  [ ON { table | view | (query) } AS MinMax DIMENSION ]
  USING
  MethodType ({ 'Sturges' | 'Scott' | 'Variable-Width' | 'Equal-Width' })
  TargetColumn ('target_column')
  [ NBins ('number_of_bins') ]
```

```
[ Inclusion ({ 'left' | 'right' }) ]
) AS alias;
```

TD_Histogram Syntax Elements

MethodType

Specify the method for calculating the frequency distribution of the data set:

Available Methods	Description
Sturges	<p>Algorithm for calculating bin width, w:</p> $w = r / (1 + \log_2 n)$ <p>where:</p> <ul style="list-style-type: none"> w = bin width r = data value range n = number of elements in data set <p>Sturges algorithm performs best if data is normally distributed and n is at least 30.</p>
Scott	<p>Algorithm for calculating bin width, w:</p> $w = 3.49s / (n^{1/3})$ <p>where:</p> <ul style="list-style-type: none"> w = bin width s = standard deviation of data values n = number of elements in data set r = data value range <p>Number of bins: r/w</p> <p>Scott algorithm performs best on normally distributed data.</p>
Variable-Width	<p>Requires MinMax table, which specifies the minimum value and the maximum value of the bin in column1 and column2 respectively, and the label of the bin in column3.</p> <p>Maximum number of bins cannot exceed 3500.</p>
Equal-Width	<p>Requires MinMax table, which specifies the minimum value of the bins in column1 and the maximum value of the bins in column2.</p> <p>Algorithm for calculating bin width, w:</p> $w = (max - min) / k$ <p>where:</p> <ul style="list-style-type: none"> min = minimum value of the bins max = maximum value of the bins k = number of intervals into which algorithm divides data set <p>Interval boundaries: $min+w, min+2w, \dots, min+(k-1)w$</p>

TargetColumn

Specify the name of the InputTable column that contains the data set.

NBins

[Required with methods Variable-Width and Equal-Width, otherwise ignored.] Specify the number of bins (number of data value ranges).

Inclusion

[Optional] Specify where to put data points that are on bin boundaries—in the bin to the left of the boundary or the bin to the right of boundary.

Default: left

TD_Histogram Input**InputTable Schema**

Column	Data Type	Description
<i>target_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	Data set

MinMax Table Schema

Column	Data Type	Description
MinValue	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	Minimum value of the bins.
MaxValue	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	Maximum value of the bins.
Label	CHAR, VARCHAR, BYTEINT, SMALLINT, INTEGER, or BIGINT Character types: CHARACTER SET can be either LATIN or UNICODE.	[Required only with MethodType ('Variable-Width').] Labels for bins or data value ranges. Maximum label length: 128 UNICODE characters.

TD_Histogram Output

Output Table Schema

Column	Data Type	Description
Label	VARCHAR (with CHARACTER SET UNICODE) or BIGINT	Labels for bins or data value ranges.
MinValue	DOUBLE PRECISION	Minimum values for bins or data value ranges.
MaxValue	DOUBLE PRECISION	Maximum values for bins or data value ranges.
CountOfValues	BIGINT	Counts of values in bins or data value ranges.
Bin_percent	DOUBLE PRECISION	Percentage of InputTable rows in bins or data value ranges.

TD_Histogram Example

Input

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

- InputTable: hist_titanic_train

```

passenger survived pclass name                sex  age sibsp
parch ticket  fare      cabin      embarked
-----
0      0  97      0      1 Goldschmidt; Mr. George B      male  71
0      0  488      0      1 Kent; Mr. Edward Austin      male  58
0      0 11771     29.7     B37      C
0      0  505      1      1 Maioni; Miss. Roberta      female 16
0      0 110152    86.5     B79      S
0      0  631      1      1 Barkworth; Mr. Algernon Henry Wilson male  80
0      0 27042     30      A23      S
0      0  873      0      1 Carlsson; Mr. Frans Olof      male  33
0      0  695      5      B51 B53 B55 S

```

- MinMax: hist_titanic_train_dim

```

minVal  maxVal  label
-----

```

0	20	Young Age
21	45	Middle Age
46	91	Old Age

SQL Call

```
SELECT * FROM TD_Histogram (
  ON hist_titanic_train AS InputTable
  ON hist_titanic_train_dim AS MinMax DIMENSION
  USING
  TargetColumn ('age')
  MethodType ('variable-width')
  nbins (3)
) AS dt;
```

Output

Label	MinValue	MaxValue	CountOfValues	bin_Percent
Middle Age	21	45	1	20
Old Age	46	90	3	60
Young Age	0	20	1	20

TD_QQNorm

TD_QQNorm checks whether the values in the specified input table columns are normally distributed. The function returns the quantiles of the column values and corresponding theoretical quantile values from a normal distribution. If the column values are normally distributed, then the quantiles of column values and normal quantile values appear in a straight line when plotted on a 2D graph.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_QQNorm Syntax

```
SELECT * FROM TD_QQNorm (
  ON { table | view | (query) } AS InputTable
```



```

    [ PARTITION BY ANY [ ORDER BY order_column ] ]
  USING
  TargetColumns ({ 'target_column' | target_column_range }[,...])
  RankColumns ({ 'rank_column' | rank_column_range }[,...])
  [ OutputColumns ('output_column' [,...]) ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;

```

TD_QQNorm Syntax Elements

TargetColumns

Specify the names of the numeric InputTable columns to check for normal distribution.

RankColumns

Specify the names of the InputTable columns that contain the ranks for the target columns.

OutputColumns

[Optional] Specify names for the output table columns that contain the theoretical quantiles of the target columns.

Default: *target_column_theoretical_quantiles*

Accumulate

[Optional] Specify the names of the InputTable columns to copy to the output table.

TD_QQNorm Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	Column to check for normal distribution.
<i>rank_column</i>	BYTEINT, SMALLINT, INTEGER, or BIGINT	Ranks for <i>target_column</i> .
<i>accumulate_column</i>	Any	Column to copy to output table.

TD_QQNorm Output

Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in InputTable	Column copied from InputTable.
<i>target_column</i>	DOUBLE PRECISION	Column checked for normal distribution.
<i>output_column</i> if specified, otherwise <i>target_column_theoretical_quantiles</i>	DOUBLE PRECISION	Theoretical quantile values for <i>target_column</i> .

TD_QQNorm Example

Input

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

```

passenger survived pclass  name                                sex  age
sibsp  parch  ticket      fare      cabin      embarked
-----
-----
0      0      97      0      1 Goldschmidt; Mr. George B      male  71
0      0      PC 17754  34.6542  A5      C
0      0      488      0      1 Kent; Mr. Edward Austin      male  58
0      0      11771  29.7      B37      C
0      0      505      1      1 Maioni; Miss. Roberta      female  16
0      0      110152  86.5      B79      S
0      0      631      1      1 Barkworth; Mr. Algernon Henry Wilson  male  80
0      0      27042  30 A23      S
0      0      873      0      1 Carlsson; Mr. Frans Olof      male  33
0      0      695      5      B51 B53 B55  S

```

From input_table, create RankTable with this statement:

```

CREATE TABLE RankTable AS (
SELECT age,
fare,
CAST (ROW_NUMBER() OVER (ORDER BY age ASC NULLS LAST) AS BIGINT)
AS rank_age,
CAST (ROW_NUMBER() OVER (ORDER BY fare ASC NULLS LAST) AS BIGINT)

```

```
AS rank_fare
FROM input_table AS dt
) WITH DATA;
```

age	fare	rank_age	rank_fare
16	86.5	1	5
33	5	2	1
58	29.7	3	2
71	34.6542	4	4
80	30	5	3

SQL Call

```
SELECT * FROM TD_QQNorm (
  ON RankTable AS InputTable
  USING
  TargetColumns ('[0:1]')
  RankColumns ('[2:3]')
) AS dt;
```

Output

age	age_theoretical_quantiles	fare	fare_theoretical_quantiles
16	-1.17986882170049	86.5	1.17986882170049
33	-0.496788749686441	5	-1.17986882170049
58	-0.000000101006675468085	29.7	-0.496788749686441
71	0.496788749686441	34.6542	0.496788749686441
80	1.17986882170049	30	-0.000000101006675468085

TD_UnivariateStatistics

UnivariateStatistics displays descriptive statistics for each specified numeric input table column.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_UnivariateStatistics Syntax

```
SELECT * FROM UnivariateStatistics (
  ON { table | view | (query) }
  AS InputTable [ PARTITION BY ANY ]
  USING
  TargetColumns ({ 'target_column' | target_column_range }[,...])
  [ PartitionColumns ('partition_column' [,...]) ]
  [ Stats ('statistic' [,...]) ]
  [ Centiles ('percentile' [,...]) ]
  [ TrimPercentile ('trimmed_percentile') ]
) AS alias;
```

TD_UnivariateStatistics Syntax Elements

TargetColumns

Specify the names of the numeric InputTable columns for which to compute statistics.

PartitionColumns

[Optional] Specify the names of the InputTable columns on which to partition the input. The function copies these columns to the output table.

Default behavior: The function treats all rows as a single partition.

Stats

[Optional] Specify the statistics to calculate. *statistic* is one of these:

- SUM
- COUNT or CNT
- MAXIMUM or MAX
- MINIMUM or MIN
- MEAN

- UNCORRECTED SUM OF SQUARES or USS
- NULL COUNT or NLC
- POSITIVE VALUES COUNT or PVC
- NEGATIVE VALUES COUNT or NVC
- ZERO VALUES COUNT or ZVC
- TOP5 or TOP
- BOTTOM5 or BTM
- RANGE or RNG
- GEOMETRIC MEAN or GM
- HARMONIC MEAN or HM
- VARIANCE or VAR
- STANDARD DEVIATION or STD
- STANDARD ERROR or SE
- SKEWNESS or SKW
- KURTOSIS or KUR
- COEFFICIENT OF VARIATION or CV
- CORRECTED SUM OF SQUARES or CSS
- MODE
- MEDIAN or MED
- UNIQUE ENTITY COUNT or UEC
- INTERQUARTILE RANGE or IQR
- TRIMMED MEAN or TM
- PERCENTILES or PRC
- ALL

Default: ALL

Centiles

[Optional] Specify the centile to calculate. *percentile* is an INTEGER in the range [1, 100].

The function ignores Centiles unless Stats specifies PERCENTILES, PRC, or ALL.

Default: 1, 5, 10, 25, 50, 75, 90, 95, 99

TrimPercentile

[Optional] Specify the trimmed lower percentile, an integer value in the range [1, 50].

The function calculates the mean of the values between the trimmed lower percentile (*trimmed_percentile*) and trimmed upper percentile (*1-trimmed_percentile*).

The function ignores TrimPercentile unless Stats specifies TRIMMED MEAN, TM, or ALL.

Default: 20

TD_UnivariateStatistics Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	NUMERIC	Column for which to calculate statistics.
<i>partition_column</i>	Any	Defines a partition for statistics calculation.

TD_UnivariateStatistics Output

Output Table Schema

Column	Data Type	Description
<i>partition_column</i>	Same as in InputTable	Column copied from InputTable. Defines a partition for statistics calculation.
Attribute	VARCHAR	<i>target_column</i> for which function calculated statistics.
StatsName	VARCHAR	[Column appears once for each specified <i>statistic</i> .] Statistic.
StatsValue	DOUBLE PRECISION	[Column appears once for each specified <i>statistic</i> .] Statistic value.

TD_UnivariateStatistics Example

InputTable: titanic_train

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

passenger	survived	name	sex	age	fare
97	0	Goldschmidt; Mr. George B	male		
71	34.6542				
488	0	Kent; Mr. Edward Austin	male		
58	29.7				
505	1	Maioni; Miss. Roberta	female		
16	86.5				
631	1	Barkworth; Mr. Algernon Henry Wilson	male	80	30
873	0	Carlsson; Mr. Frans Olof	male	33	5

SQL Call

```
SELECT * FROM TD_UnivariateStatistics (
  ON titanic_train
  USING
  TargetColumns ('age','fare')
  Stats ('MEAN', 'MEDIAN', 'MODE')
) AS dt;
```

Output

ATTRIBUTE	StatName	StatValue
-----	-----	-----
age	MEAN	5.16000000000000E 001
age	MEDIAN	5.80000000000000E 001
age	MODE	1.60000000000000E 001
fare	MEAN	3.71708400000000E 001
fare	MEDIAN	3.00000000000000E 001
fare	MODE	5.00000000000000E 000

TD_WhichMax

TD_WhichMax displays all rows that have the maximum value in a specified input table column.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_WhichMax Syntax

```
SELECT * FROM TD_WhichMax (
  ON { table | view | (query) } AS InputTable
  [ PARTITION BY ANY [ORDER BY order_by_column ] ]
  USING
  TargetColumn ('target_column')
) AS alias;
```

TD_WhichMax Syntax Elements

TargetColumn

Specify the target column names to check for maximum values.

TD_WhichMax Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	Any except BLOB, CLOB, and UDT.	Columns for which maximum values are checked.

TD_WhichMax Output

Output Table Schema

Same as InputTable schema

TD_WhichMax Example

InputTable: titanic_dataset

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

passenger	survived	pclass	sex	age	sibsp	parch	fare	cabin	embarked
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
1	0	3	male	22	1	0	7.25	null	S
2	1	1	female	38	1	0	71.28	C85	C
3	1	3	female	26	0	0	7.93	null	S
4	1	1	female	35	1	0	53.10	C123	S
5	0	3	male	35	0	0	8.05	null	S

SQL Call

```
SELECT * FROM TD_WhichMax (
  ON titanic_dataset AS InputTable
  USING
    TargetColumn ('fare')
) AS dt;
```


Output

```

passenger survived pclass sex    age sibsp parch    fare    cabin embarked
-----
                2         1         1 female  38      1      0    71.28      C85      C

```

TD_WhichMin

TD_WhichMin displays all rows that have the minimum value in specified input table column.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_WhichMin Syntax

```

SELECT * FROM TD_WhichMin (
  ON { table | view | (query) } AS InputTable
    [ PARTITION BY ANY [ORDER BY order_by_column ] ]
  USING
    TargetColumn ('target_column')
) AS alias;

```

TD_WhichMin Syntax Elements**TargetColumn**

Specify the target column names to check for minimum values.

TD_WhichMin Input**InputTable Schema**

Column	Data Type	Description
<i>target_column</i>	Any except BLOB, CLOB, and UDT.	Columns for which minimum values are checked.

TD_WhichMin Output

Output Table Schema

Same as InputTable schema

TD_WhichMin Example

InputTable: titanic_dataset

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

passenger	survived	pclass	sex	age	sibsp	parch	fare	cabin	embarked
1	0	3	male	22	1	0	7.25	null	S
2	1	1	female	38	1	0	71.28	C85	C
3	1	3	female	26	0	0	7.93	null	S
4	1	1	female	35	1	0	53.10	C123	S
5	0	3	male	35	0	0	8.05	null	S

SQL Call

```
SELECT * FROM TD_WhichMin (  
  ON titanic_dataset AS InputTable  
  USING  
    TargetColumn ('fare')  
) AS dt;
```

Output

passenger	survived	pclass	sex	age	sibsp	parch	fare	cabin	embarked
1	0	3	male	22	1	0	7.25	null	S

Feature Engineering Transform Functions

Feature engineering transform functions encapsulate variable transformations during the training phase so you can chain them to create a pipeline for operationalization.

Each `TD_nameFit` function outputs a table to input to the `TD_nameTransform` function as `FitTable`. For example, `TD_BinCodeFit` outputs a `FitTable` for `TD_BinCodeTransform`.

TD_BinCodeFit

`TD_BinCodeFit` outputs a table of information to input to [TD_BinCodeTransform](#), which bin-codes the specified input table columns.

Bin-coding is typically used to convert numeric data to categorical data by binning the numeric data into multiple numeric bins (intervals). The bins can have a fixed-width with auto-generated labels or can have specified variable widths and labels.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.
-

TD_BinCodeFit Syntax

For Equal-Width Bins with Generated Labels

```
CREATE TABLE primary_output_table AS (
  SELECT * FROM TD_BincodeFit (
    ON { table | view | (query) } AS InputTable
    [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table) ]
    USING
    TargetColumns ({ 'target_column' | target_column_range }[,...])
    MethodType ('equal-width')
    NBins ('single bin value' | 'separate bin values')
    [ LabelPrefix ('single prefix' | 'separate prefix values') ]
  ) AS alias
) WITH DATA;
```

For Variable-Width Bins with Specified Labels

```
CREATE TABLE primary_output_table AS (
  SELECT * FROM TD_BincodeFit (
    ON { table | view | (query) } AS InputTable
    ON { table | view | (query) } AS FitInput DIMENSION
    USING
    TargetColumns ({ 'target_column' | target_column_range }[,...])
    MethodType ('variable-width')
    [ MinValueColumn ('minvalue_column') ]
    [ MaxValueColumn ('maxvalue_column') ]
    [ LabelColumn ('label_column') ]
    [ TargetColNames ('target_names_column') ]
  ) AS alias
) WITH DATA;
```

TD_BinCodeFit Syntax Elements

OutputTable

[MethodType ('equal-width') only.] [Optional] Specify a name for the secondary output table which contains the actual bins used by the BincodeTransform function.

TargetColumns

Specify the names of the InputTable columns to bin-code.

The maximum number of target columns is 2018.

MethodType

Specify the bin-coding method:

Method Type	Description
equal-width	<p>Bins have fixed width with auto-generated labels.</p> <p>Function finds minimum and maximum values of target columns (<i>min</i> and <i>max</i>) and computes bin width, <i>w</i>, with this formula:</p> $w = (max - min) / k$ <p><i>k</i>, the number of bins, is determined by NBins.</p> <p>For bin boundaries and names of generated labels, see LabelPrefix.</p>
variable-width	<p>Bins have specified variable widths and specified labels, provided by FitInput table.</p> <p>Maximum number of bins is 3000.</p>

NBins

[MethodType ('equal-width') only.] Specify either a single bin value for all the target columns or separate bin values for each of the target columns.

LabelPrefix

[MethodType ('equal-width') only.][Optional] Specify either a prefix for all the target columns or a separate prefix for each of the target columns.

Lower Bin Boundary	Upper Bin Boundary	prefix-Generated Bin Label	prefix_count-Generated Bin Label
min	$min + w$	$prefix_1$	$target_column_1$
$min + w$	$min + 2w$	$prefix_2$	$target_column_2$
...
$min + kw$	$min + (k - 1)w$	$prefix_k$	$target_column_k$

Default: Target column name is used as the prefix

MinValueColumn

[MethodType ('variable-width') only.][Optional] Specify the name of the FitInput column that has the minimum value of the bin (lower bin boundaries).

Default: MinValue

MaxValueColumn

[MethodType ('variable-width') only.][Optional] Specify the name of the FitInput column that has the maximum value of the bin (upper bin boundaries).

Default: MaxValue

LabelColumn

[MethodType ('variable-width') only.][Optional] Specify the name of the FitInput column that has the bin labels.

Default: Label

TargetColNames

[MethodType ('variable-width') only.][Optional] Specify the name of the FitInput column that has the target column names.

Note:

Column range is not supported

TD_BinCodeFit Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	Column to bin-code.

FitTable Schema

Required with specify MethodType ('variable-width'), ignored otherwise.

Column	Data Type	Description
<i>target_names_column</i>	CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE)	Bin column name
<i>minvalue_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	Minimum value of the bin.
<i>maxvalue_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	Maximum value of the bin.
<i>label_column</i>	CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE)	Bin labels.

TD_BinCodeFit Output

Output Table Schema

Column	Data Type	Description
TD_ColumnName_BINFIT	VARCHAR (CHARACTER SET UNICODE)	<i>target_names_column</i> . Bin column name.
TD_MinValue_BINFIT	DOUBLE PRECISION	<i>minvalue_column</i> . Minimum value of the bin.
TD_MaxValue_BINFIT	DOUBLE PRECISION	<i>maxvalue_column</i> . Maximum value of the bin.
TD_LabelPrefix_BINFIT	VARCHAR (CHARACTER SET UNICODE)	[Column appears only with MethodType ('equal-width').] Label prefix.

Column	Data Type	Description
TD_Label_BINFIT	VARCHAR (CHARACTER SET UNICODE)	[Column appears only with MethodType ('variable-width').] Bin label.
TD_Bins_BINFIT	INTEGER	<i>target_names_column</i> . Bin count.
TD_IndexValue_BINFIT	SMALLINT	Index value.
TD_MaxLenLabel_BINFIT	SMALLINT	Maximum bin label length.
<i>target_column</i>	Same as in Input table	Target column for TD_BinCodeTransform.

OutputTable Schema

The function outputs this secondary output table only if you specify MethodType ('equal-width').

Column	Data Type	Description
ColumnName	VARCHAR (CHARACTER SET UNICODE)	The column name of the bin.
MinValue	DOUBLE PRECISION	The minimum value of the bin.
MaxValue	DOUBLE PRECISION	The maximum value of the bin.
label	VARCHAR (CHARACTER SET UNICODE)	The label of the bin.

TD_BinCodeFit Example

InputTable: bin_titanic_train

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

```

passenger survived pclass name                sex  age sibsp
parch ticket  fare      cabin    embarked
-----
0      97      0      1 Goldschmidt; Mr. George B   male  71
0      0 PC 17754 34.654200000 A5      C
      488      0      1 Kent; Mr. Edward Austin   male  58
0      0 11771  29.700000000 B37      C
      505      1      1 Maioni; Miss. Roberta     female 16

```

```

0      0 110152    86.5000000000 B79          S
      631         1         1 Barkworth; Mr. Algernon Henry Wilson male    80
0      0 27042    30.0000000000 A23          S
      873         0         1 Carlsson; Mr. Frans Olof                male    33
0      0 695      5.0000000000 B51 B53 B55 S

```

Input: FitInput table

ColumnName	MinValue	MaxValue	Label
age	0.00	20.00	Young Age
age	21.00	45.00	Middle Age
age	46.00	90.00	Old Age

SQL Call

```

CREATE TABLE FitOutputTable AS (
  SELECT * FROM TD_BincodeFit (
    ON bin_titanic_train AS InputTable
    ON FitInputTable AS FitInput DIMENSION
    USING
      TargetColumns ('age')
      MethodType ('Variable-Width')
      MinValueColumn ('MinValue')
      MaxValueColumn ('MaxValue')
      LabelColumn ('Label')
      TargetColNames ('ColumnName')
  ) AS dt
) WITH DATA;

```

Output

TD_ColumnName_BINFIT	TD_MinValue_BINFIT	TD_MaxValue_BINFIT	TD_Label_BINFIT
TD_Bins_BINFIT	TD_IndexValue_BINFIT	TD_MaxLenLabel_BINFIT	age
age	46.0000000000	90.0000000000	Old
Age	3	0	10 null
age	21.0000000000	45.0000000000	Middle
Age	3	0	10 null
age	0.0000000000	20.0000000000	Young
Age	3	0	10 null

TD_BinCodeTransform

TD_BinCodeTransform bin-codes input table columns, using [TD_BinCodeFit](#) output.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_BinCodeTransform Syntax

```
SELECT * FROM TD_BincodeTransform (
  ON { table | view | (query) } AS InputTable
  ON { table | view | (query) } AS FitTable DIMENSION
  USING
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

TD_BinCodeTransform Syntax Elements

Accumulate

[Optional] Specify names of InputTable columns to copy to the output table.

TD_BinCodeTransform Input

InputTable Schema

See [TD_BinCodeFit Input](#).

FitTable Schema

See [TD_BinCodeFit Output](#).

TD_BinCodeTransform Output

Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in InputTable.	Column copied from InputTable.
<i>target_column</i>	VARCHAR (CHARACTER SET UNICODE)	[Column appears once for each specified <i>target_column</i> .] Bin labels.

TD_BinCodeTransform Example

- InputTable: bin_titanic_train, as in [TD_BinCodeFit Example](#)
- FitTable: FitOutputTable, created by [TD_BinCodeFit Example](#)

SQL Call

```
SELECT * FROM TD_BincodeTransform (
  ON bin_titanic_train AS InputTable
  ON FitOutputTable AS FitTable DIMENSION
  USING
  Accumulate ('passenger')
) AS dt;
```

Output

```
passenger age
-----
      873 Middle Age
      631 Old Age
      505 Young Age
      488 Old Age
       97 Old Age
```

TD_FunctionFit

TD_FunctionFit determines whether specified numeric transformations can be applied to specified input columns and outputs a table to use as input to [TD_FunctionTransform](#), which does the transformations.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

Related Information:

[TD_NumApply](#)

TD_FunctionFit Syntax

```
CREATE TABLE output_table AS (
  SELECT * FROM TD_FunctionFit (
    ON { table | view | (query) } AS InputTable
    ON { table | view | (query) } AS TransformationTable DIMENSION
  ) AS alias
) WITH DATA;
```

TD_FunctionFit Input

InputTable Schema

Column	Data Type	Description
<i>input_column</i>	VARCHAR (CHARACTER SET LATIN or UNICODE) or NUMERIC	Column whose name can appear as TargetColumn in TransformationTable.

TransformationTable Schema

Column	Data Type	Description
TargetColumn	VARCHAR (CHARACTER SET LATIN or UNICODE)	Name of InputTable column to transform.
Transformation	VARCHAR (CHARACTER SET LATIN or UNICODE)	Transformation to apply to TargetColumn—for allowed transformations, see following table.
Parameters	VARCHAR (CHARACTER SET LATIN or UNICODE)	[Optional] Transformation parameters in JSON format.

Column	Data Type	Description
		If this column is absent and transformation has parameter, function uses default value in Parameters column of following table.
DefaultValue	NUMERIC	[Optional] Default value for transformed value if TargetColumn is nonnumeric or NULL. If this column is absent, function uses default value 0.

Transformations

Transformation	Parameter	Operation on TargetColumn Value x
ABS	None	$ x $
CEIL	None	CEIL (x) (Least integer $\geq x$)
EXP	None	e^x ($e = 2.718$)
FLOOR	None	FLOOR (x) (Greatest integer $\leq x$)
LOG	[Optional] {"base": <i>base</i> } Default: e	$\text{LOG}_{base}(x)$
POW	[Optional] {"exponent": <i>exponent</i> } Default: 1	$x^{exponent}$
SIGMOID	None	$1 / (1 + e^{-x})$
TANH	None	$(e^x - e^{-x}) / (e^x + e^{-x})$

TD_FunctionFit Output

Output Table Schema

Same as TransformationTable schema (see [TD_FunctionFit Input](#)).

TD_FunctionFit Example

Input

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

InputTable: function_input_table

```

passenger survived pclass name
sex    age sibsp parch ticket          fare          cabin embarked
-----
-----
      1      0      3 Braund; Mr. Owen Harris
male   22      1      0 A/5 21171          7.250000000      S
      2      1      1 Cumings; Mrs. John Bradley (Florence Briggs Thayer)
female 38      1      0 PC 17599          71.283300000 C85   C
      3      1      3 Heikkinen; Miss. Laina
female 26      0      0 STON/O2. 3101282  7.925000000      S
      4      1      1 Futrelle; Mrs. Jacques Heath (Lily May Peel)
female 35      1      0 113803          53.100000000 C123  S
      5      0      3 Allen; Mr. William Henry
male   35      0      0 373450          8.050000000      S

```

SQL Call

```

CREATE TABLE fit_out AS (
  SELECT * FROM TD_FunctionFit (
    ON function_input_table AS InputTable
    ON transformations AS TransformationTable DIMENSION
  ) AS dt
) WITH DATA;

```

Output

TargetColumn	Transformation	Parameters	Defaultvalue
age	LOG	{"base":2}	0.000000000
fare	POW	{"exponent": 2}	10.000000000

TD_FunctionTransform

TD_FunctionTransform applies numeric transformations to input columns, using [TD_FunctionFit](#) output.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_FunctionTransform Syntax

```
SELECT * FROM TD_FunctionTransform (
  ON { table | view | (query) } AS InputTable
  ON { table | view | (query) } AS FitTable DIMENSION
  [ IDColumns ({ 'id_column' | id_column_range }[,...])]
) AS alias;
```

TD_FunctionTransform Syntax Elements

IDColumns

[Optional] Specify the names of the InputTable columns with NUMERIC datatypes to exclude from transformations. The columns with VARCHAR datatypes are automatically excluded.

No *id_column* can be a *target_column* in the TransformationTable for the TD_FunctionFit call that output FitTable.

TD_FunctionTransform Input

InputTable Schema

See [TD_FunctionFit Input](#).

FitTable Schema

See [TD_FunctionFit Output](#).

TD_FunctionTransform Output

Output Table Schema

Column	Data Type	Description
<i>input_column</i>	If NUMERIC in TD_FunctionFit InputTable: DOUBLE PRECISION	Transformed values.

Column	Data Type	Description
	Otherwise: Same as in TD_FunctionFit InputTable	

TD_FunctionTransform Example

Input

- InputTable: titanic_data, as in [TD_FunctionFit Example](#)
- FitTable: fit_out, created by [TD_FunctionFit Example](#)

SQL Call

```
SELECT * FROM TD_FunctionTransform (
  ON function_input_table AS InputTable
  ON fit_out AS FitTable DIMENSION
  USING
    IDColumns ('[0:2]','[6:7]')
) AS dt ORDER BY Passenger;
```

Output

passenger name	survived	pclass	age	sibsp	parch	embarked
ticket			fare	cabin		
1	0	3	Braund; Mr. Owen			
Harris						
male	4.45943161863730E 000		1		0	A/5 21171
5.25625000000000E 001			S			
2	1	1	Cumings; Mrs. John Bradley (Florence Briggs			
Thayer)			female		5.24792751344359E	
000	1	0	PC 17599		5.08130885889000E 003	
C85		C				
3	1	3	Heikkinen; Miss.			
Laina						
female	4.70043971814109E 000		0		0	STON/O2. 3101282
6.28056250000000E 001			S			
4	1	1	Futrelle; Mrs. Jacques Heath (Lily May			
Peel)			female		5.12928301694497E	

```

000          1          0 113803          2.81961000000000E 003
C123          S
          5          0          3 Allen; Mr. William
Henry                                          male
5.12928301694497E 000          0          0 373450
6.48025000000000E 001          S

```

TD_OneHotEncodingFit

TD_OneHotEncodingFit outputs a table of attributes and categorical values to input to [TD_OneHotEncodingTransform](#), which encodes them as one-hot numeric vectors.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_OneHotEncodingFit Syntax

```

CREATE TABLE fit_table AS (
  SELECT * FROM TD_OneHotEncodingFit (
    ON { table | view | (query) } AS InputTable
    [ PARTITION BY { ANY [ ORDER BY order_column ] | attribute_column } ]
    USING
    IsInputDense ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' })
    { for_dense_input | for_sparse_input }
  ) AS alias
) WITH DATA;

```

for_dense_input

```

TargetColumn ('target_column')
CategoricalValues ('categorical_value' [,...])
[ OtherColumnName ('other_column') ]

```


for_sparse_input

```

AttributeColumn ('attribute_column')
ValueColumn ('value_column')
TargetAttributes ('target_attribute' [,...])
[ OtherAttributeNames ('other_attribute' [,...]) ]

```

TD_OneHotEncodingFit Syntax Elements**IsInputDense**

Specify whether the input is in dense format.

TargetColumn

[Required with IsInputDense ('true'), disallowed otherwise.] Specify the name of the InputTable column of categorical values.

CategoricalValues

[Required with IsInputDense ('true'), disallowed otherwise.] Specify one or more categorical values in *target_column* to encode in one-hot form.

OtherColumnName

[Optional with IsInputDense ('true'), disallowed otherwise.] Specify a category name for values that CategoricalValues does not specify (categorical values not to encode in one-hot form).

Default: 'other'

AttributeColumn

[Required with IsInputDense ('false'), disallowed otherwise.] Specify the name of the InputTable column of attributes.

ValueColumn

[Required with IsInputDense ('false'), disallowed otherwise.] Specify the name of the InputTable column of attribute values.

TargetAttributes

[Required with IsInputDense ('false'), disallowed otherwise.] Specify one or more attributes to encode in one-hot form. Every *target_attribute* must be in *attribute_column*.

OtherAttributeNames

[Optional with IsInputDense ('false'), disallowed otherwise.] For each *target_attribute*, specify a category name (*other_attribute*) for attributes that TargetAttributes does not specify. The *n*th *other_attribute* corresponds to the *n*th *target_attribute*.

TD_OneHotEncodingFit Input**InputTable Schema for Dense Input**

Column	Data Type	Description
<i>target_column</i>	CHAR or VARCHAR (CHARACTER SET LATIN or UNICODE)	Categorical values.

InputTable Schema for Sparse Input

Column	Data Type	Description
<i>attribute_column</i>	CHAR or VARCHAR (CHARACTER SET LATIN or UNICODE)	Attribute names.
<i>value_column</i>	CHAR or VARCHAR (CHARACTER SET LATIN or UNICODE)	Attribute values.

TD_OneHotEncodingFit Output**Output Table Schema for Dense Input**

Column	Data Type	Description
<i>target_column</i>	INTEGER	Preserves target column name for TD_OneHotEncodingTransform. Contains only NULL values.
<i>target_column_categorical_value</i>	VARCHAR (CHARACTER SET UNICODE)	Preserves target column definition for TD_OneHotEncodingTransform.
<i>target_column_other_column</i>	VARCHAR (CHARACTER SET UNICODE)	Categorical values not to encode in one-hot form.

Output Table Schema for Sparse Input

Column	Data Type	Description
TD_attribute_column_type_OHEFIT	INTEGER	1 if row has <i>attribute_column-target_attribute</i> pair. 0 if row has <i>attribute_column-other_attribute</i> pair.
<i>attribute_column</i>	VARCHAR (CHARACTER SET UNICODE)	Preserves attribute column name for TD_OneHotEncodingTransform. Contains only NULL values.
<i>value_column</i>	VARCHAR (CHARACTER SET UNICODE)	Preserves value column name for TD_OneHotEncodingTransform. Contains only NULL values.

TD_OneHotEncodingFit Example**InputTable: input_table**

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

Passenger	Survived	Pclass	Name	Age	Sex
1	0	3	Mr. Owen Harris	22	Male
2	1	1	Mrs. John Bradley	38	Female
3	1	3	Mrs. Laina	26	Female
4	0	3	Mrs. Jacques Heath	35	Female

SQL Call

```
CREATE TABLE onehotencodingfit_output AS (
  SELECT * FROM TD_OneHotEncodingFit (
    ON onehotencodingfit_input AS InputTable
    USING
      TargetColumn ('Sex')
      OtherColumnName ('other')
      CategoricalValues ('male', 'female')
      IsInputDense ('true')
  ) AS dt
) WITH DATA;
```

Output

```
Sex  Sex_male Sex_female Sex_other
-----
null  male    female    null
```

TD_OneHotEncodingTransform

TD_OneHotEncodingTransform encodes specified attributes and categorical values as one-hot numeric vectors, using [TD_OneHotEncodingFit](#) output.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_OneHotEncodingTransform Syntax

```
SELECT * FROM TD_OneHotEncodingTransform (
  ON { table | view | (query) } AS InputTable
    [ PARTITION BY { ANY [ ORDER BY order_column ] | attribute_column } ]
  ON { table | view | (query) } AS FitTable
    { DIMENSION | PARTITION BY attribute_column }
  USING
  IsInputDense ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' })
) AS alias;
```

TD_OneHotEncodingTransform Syntax Elements**IsInputDense**

Specify whether the input is in dense format.

TD_OneHotEncodingTransform Input**InputTable Schema**

See [TD_OneHotEncodingFit Input](#).

FitTable Schema

See [TD_OneHotEncodingFit Output](#).

TD_OneHotEncodingTransform Output**Output Table Schema for Dense Input**

Column	Data Type	Description
<i>input_column</i>	Same as in InputTable	Column copied from InputTable.
<i>target_column_categorical_value</i>	INTEGER	One-hot-encoded categorical values.

Output Table Schema for Sparse Input

Column	Data Type	Description
<i>input_column</i>	Same as in InputTable	Column copied from InputTable.
<i>attribute_column</i>	CHAR or VARCHAR (CHARACTER SET UNICODE)	Attribute names.
<i>value_column</i>	CHAR or VARCHAR (CHARACTER SET UNICODE)	One-hot-encoded attribute values.

TD_OneHotEncodingTransform Example

- InputTable: onehotencoding_input, as in [TD_OneHotEncodingFit Example](#)
- FitTable: onehotencodingfit_output, created by [TD_OneHotEncodingFit Example](#)

SQL Call

```
SELECT * FROM TD_OneHotEncodingTransform (
  ON onehotencoding_input AS InputTable PARTITION BY ANY
  ON onehotencodingfit_output AS FitTable DIMENSION
  USING
  IsInputDense ('true')
) AS dt;
```

Output

Passenger	Survived	Pclass	Name	Age	Sex_male	Sex_female	Sex_other
1	0	3	Mr. Owen Harris	22	1	0	0
4	0	3	Mrs. Jacques Heath	35	0	1	0

3	1	3 Mrs. Laina	26	0	1	0
2	1	1 Mrs. John Bradley	38	0	1	0

TD_PolynomialFeaturesFit

TD_PolynomialFeaturesFit function stores all the specified values in the argument in a tabular format.

All polynomial combinations of the features with degrees less than or equal to the specified degree are generated. For example, for a 2-D input sample [x, y], the degree-2 polynomial features are [x, y, x-squared, xy, y-squared, 1].

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_PolynomialFeaturesFit Syntax

```
SELECT * FROM TD_PolynomialFeaturesFit (
  ON { table | view | (query) } AS InputTable
  [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table) ]
  USING
  TargetColumns ({ 'target_column' | target_column_range }[,...])
  [ IncludeBias ({'true'|'t'|"yes"|"y"|"1"|"false"|"f"|"no"|"n"|"0"}) ]
  [ InteractionOnly ({'true'|'t'|"yes"|"y"|"1"|"false"|"f"|"no"|"n"|"0"}) ]
  [ Degree (degree) ]
) AS alias;
```

TD_PolynomialFeaturesFit Syntax Elements

OutputTable

[Optional] Specify a name for the output table.

If you omit OutputTable, you must create the output table for TD_PolynomialFeaturesTransform with a CREATE TABLE AS statement:

```
CREATE TABLE output_table AS (
  SELECT * FROM TD_PolynomialFeaturesFit ( ... ) AS alias
) WITH DATA;
```

TargetColumns

Specify the names of the InputTable columns for which to output polynomial combinations for features (no more than five).

IncludeBias

[Optional] Specify whether the output table is to include a bias column for the feature in which all polynomial powers are zero (that is, a column of ones). A bias column acts as an intercept term in a linear model.

Default: true

InteractionOnly

[Optional] Specify whether to output polynomial combinations only for interaction features (features that are products of at most *degree distinct* input features).

Default: false

Degree

[Optional] Specify the maximum degree of the input features for which to output polynomial combinations, an integer in the range [1,2,3].

Default: 2

TD_PolynomialFeaturesFit Input**InputTable Schema**

Column	Data Type	Description
<i>target_column</i>	NUMERIC	Column for which to output polynomial combinations for features.

TD_PolynomialFeaturesFit Output**OutputTable Schema**

Column	Data Type	Description
TD_IncludeBias_POLFIT: <i>boolean</i>	INTEGER	1 if <i>boolean</i> is 'True', 0 if it is 'False'.

Column	Data Type	Description
<i>boolean</i> is IncludeBias value, 'True' or 'False'.		
TD_InteractionOnly_ POLFIT: <i>boolean</i> <i>boolean</i> is InteractionOnly value, 'True' or 'False'.	INTEGER	1 if <i>boolean</i> is 'True', 0 if it is 'False'.
TD_Degree_POLFIT: <i>degree</i>	INTEGER	<i>degree</i>
<i>target_column</i>	NUMERIC	Preserves target column name for TD_PolynomialFeaturesTransform. Contains only NULL values.

TD_PolynomialFeaturesFit Example

InputTable: polynomialFeatures

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

```
id col1 col2 col3
-- ----
1    2    3    4
2    5    6    7
3    1    2    4
4    5    3    5
5    3    2    6
```

SQL Call

```
CREATE TABLE polynomialFit AS (
  SELECT * FROM TD_PolynomialFeaturesFit (
    ON polynomialFeatures AS InputTable
    USING
    TargetColumns ('[1:2]')
    Degree (2)
  ) AS dt
) WITH DATA;
```


Output

```

TD_INCLUDEBIAS_POLFIT:TRUE TD_INTERACTIONONLY_POLFIT:FALSE TD_DEGREE_POLFIT:2
col1 col2
-----
----
null null          1          0          2

```

TD_PolynomialFeaturesTransform

TD_PolynomialFeaturesTransform function extracts values of arguments [TargetColumns, Degree, IncludeBias, and InteractionOnlygenerates] from the output of the TD_PolynomialFeaturesFit function and generates a feature matrix of all polynomial combinations of the features.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_PolynomialFeaturesTransform Syntax

```

SELECT * FROM TD_PolynomialFeaturesTransform (
  ON { table | view | (query) } AS InputTable PARTITION BY ANY
  ON { table | view | (query) } AS FitTable DIMENSION
  USING
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;

```

TD_PolynomialFeaturesTransform Syntax Elements**Accumulate**

[Optional] Specify the names of the InputTable columns to copy to the output table.

Note:

If two or more column names are concatenated and the column name exceeds 128 characters, then the function replaces the actual column names with names such as col1, col2, col3, col4, col5 in the output.

TD_PolynomialFeaturesTransform Input

InputTable Schema

See [TD_PolynomialFeaturesFit Input](#).

FitTable Schema

See [TD_PolynomialFeaturesFit Output](#).

TD_PolynomialFeaturesTransform Output

Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in InputTable.	Column copied from InputTable.
One	DOUBLE PRECISION	[Column appears only with IncludeBias ('true').] Column for feature in which all polynomial powers are zero (column of ones).
<i>output_column</i>	DOUBLE PRECISION	[Column appears once for each polynomial combination of <i>degree</i> or less.] Polynomial combination. For example, for two-dimensional input feature [x, y], output columns are x, y, x_square, y_square, x_y, where x and y are InputTable target columns.

TD_PolynomialFeaturesTransform Example

- InputTable: polynomialFeatures, as in [TD_PolynomialFeaturesFit Example](#)
- FitTable: polynomialFit, created by [TD_PolynomialFeaturesFit Example](#)

SQL Call

```
SELECT * FROM TD_PolynomialFeaturesTransform (
  ON polynomialFeatures AS InputTable PARTITION BY ANY
  ON polynomialFit AS FitTable DIMENSION
  USING
```

```
Accumulate ('[0:0]')
) AS dt;
```

Output

id	ONE	col1	col1_col2	col1_SQUARE	col2	col2_SQUARE
1	1.000000000	2.000000000	6.000000000	4.000000000	3.000000000	9.000000000
2	1.000000000	5.000000000	30.000000000	25.000000000	6.000000000	36.000000000
3	1.000000000	1.000000000	2.000000000	1.000000000	2.000000000	4.000000000
4	1.000000000	5.000000000	15.000000000	25.000000000	3.000000000	9.000000000
5	1.000000000	3.000000000	6.000000000	9.000000000	2.000000000	4.000000000

TD_RowNormalizeFit

TD_RowNormalizeFit outputs a table of parameters and specified input columns to input to [TD_RowNormalizeTransform](#), which normalizes the input columns row-wise.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_RowNormalizeFit Syntax

```
SELECT * FROM TD_RowNormalizeFit (
  ON { table | view | (query) } AS InputTable
  [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table_name) ]
  USING
  TargetColumns ({ 'target_column' | target_column_range }[,...])
  [ Approach ({ 'UNITVECTOR' | 'FRACTION' | 'PERCENTAGE' | 'INDEX' }) ]
  [ BaseColumn ('base_column')
    BaseValue (base_value) ]
) AS alias;
```

TD_RowNormalizeFit Syntax Elements

OutputTable

[Optional] Specify a name for the output table.

If you omit OutputTable, you must create the output table for TD_RowNormalizeTransform with a CREATE TABLE AS statement:

```
CREATE TABLE output_table AS (
  SELECT * FROM TD_RowNormalizeFit ( ... ) AS alias
) WITH DATA;
```

TargetColumns

Specify the names of the InputTable columns to normalize row-wise.

Approach

[Optional] Specify the normalization method:

Option	Normalizing Formula
UNITVECTOR (Default)	$X' = X / (\text{sqrt}(\sum_{i \in [1, n]} X_i^2))$
FRACTION	$X' = X / (\sum_{i \in [1, n]} X_i)$
PERCENTAGE	$X' = X * 100 / (\sum_{i \in [1, n]} X_i)$
INDEX	$X' = V + ((X - B) / B) * 100$

In the normalizing formulas:

X' is the normalized value.

X is the original value.

B is the value in the base column.

V is the base value.

BaseColumn

[Required with Approach ('INDEX'), ignored otherwise.] Specify the name of the InputTable column that has the B values to use in the normalizing formula.

BaseValue

[Required with Approach ('INDEX'), ignored otherwise.] Specify the V value to use in the normalizing formula.

TD_RowNormalizeFit Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	Column to normalize row-wise.
<i>base_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	[Column appears only with Approach ('INDEX').] <i>B</i> value to use in normalizing formula.

TD_RowNormalizeFit Output

OutputTable Schema

Column	Data Type	Description
TD_KEY_ROWFIT	VARCHAR (CHARACTER SET LATIN)	Parameter key.
TD_VALUE_ROWFIT	VARCHAR (CHARACTER SET UNICODE)	Parameter value.
<i>target_column</i>	Same as in InputTable	[Column appears once for each specified <i>target_column</i> .] NULL value.

TD_RowNormalizeFit Example

InputTable: input_table

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

```
id x y
-- - --
1 0 1
2 3 4
3 5 12
4 7 24
```

SQL Call

```
CREATE TABLE fit_output AS (
  SELECT * FROM TD_RowNormalizeFit (
    ON input_table AS InputTable
    USING
    TargetColumns ('[1:2]')
    Approach ('INDEX')
    BaseColumn ('y')
    BaseValue (100)
  ) AS dt
) WITH DATA;
```

Output

```
SELECT * FROM fit_output;
```

TD_KEY_ROWFIT	TD_VALUE_ROWFIT	x	y
Approach	INDEX	null	null
BaseColumn	y	null	null
BaseValue	100	null	null

TD_RowNormalizeTransform

TD_RowNormalizeTransform normalizes input columns row-wise, using [TD_RowNormalizeFit](#) output.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_RowNormalizeTransform Syntax

```
SELECT * FROM TD_RowNormalizeTransform (
  ON { table | view | (query) } AS InputTable [ PARTITION BY ANY [ ORDER BY
order_column ] ]
  ON { table | view | (query) } AS FitTable DIMENSION
```

```

USING
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;

```

TD_RowNormalizeTransform Syntax Elements

Accumulate

[Optional] Specify the names of the InputTable columns to copy to the output table.

TD_RowNormalizeTransform Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	Column to normalize row-wise.
<i>base_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal/Numeric, Float, Real, Double precision	[Column appears only with Approach ('INDEX').] <i>B</i> value to use in normalizing formula.
<i>accumulate_column</i>	Any	The input table column names copied to the output table.

FitTable Schema

See [TD_RowNormalizeFit Output](#).

TD_RowNormalizeTransform Output

Column	Data Type	Description
<i>accumulate_column</i>	Same as in InputTable	Column copied from InputTable.
<i>target_column</i>	DOUBLE PRECISION	Row-normalized values.

TD_RowNormalizeTransform Example

- InputTable: input_table, as in [TD_RowNormalizeFit Example](#)
- FitTable: fit_output, output by [TD_RowNormalizeFit Example](#)

SQL Call

```
SELECT * FROM TD_RowNormalizeTransform (
  ON input_table AS InputTable
  ON fit_output AS FitTable DIMENSION
  USING
  Accumulate ('id')
) AS dt;
```

Output

id	x	y
1	0.00	100.00
2	75.00	100.00
3	41.66	100.00
4	29.16	100.00

TD_ScaleFit

TD_ScaleFit outputs a table of statistics to input to [TD_ScaleTransform](#), which scales specified input table columns.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_ScaleFit Syntax

```
SELECT * FROM TD_ScaleFit (
  ON { table | view | (query) } AS InputTable
  [ PARTITION BY ANY [ ORDER BY order_column ] ]
  [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table) ]
  USING
  TargetColumns ( { 'target_column' | target_column_range }[,...] )
  ScaleMethod ('scale_method' [,...])
  [ Multiplier ('multiplier' [,...]) ]
```



```
[ Intercept ('intercept' [,...]) ]
[ GlobalScale ({'true'|'t' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
[ MissValue ({ 'KEEP' | 'ZERO' | 'LOCATION' }) ]
) AS alias;
```

TD_ScaleFit Syntax Elements

OutputTable

[Optional] Specify a name for the output table.

TargetColumns

Specify the names of the InputTable columns for which to output statistics. The columns must contain numeric data in the range $(-1e^{308}, 1e^{308})$.

ScaleMethod

Specify either one *scale_method* for all target columns or one *scale_method* for each *target_column*. The *nth scale_method* applies to the *nth target_column*.

The following table lists each possible *scale_method* and its *location* and *scale* values.

The TD_ScaleTransform function uses the *location* and *scale* values in the following formula to scale target column value *X* to scaled value *X'*:

$$X' = \text{intercept} + \text{multiplier} * ((X - \text{location})/\text{scale})$$

Intercept and Multiplier determine *intercept* and *multiplier*.

In the table, Xmin, Xmax, and XMean are the minimum, maximum, and mean values of *target_column*.

<i>scale_method</i>	Description	<i>location</i>	<i>scale</i>
MAXABS	Maximum absolute value.	0	Maximum X
MEAN	Mean.	XMean	1
MIDRANGE	Midrange.	(Xmax+Xmin)/2	(Xmax-Xmin)/2
RANGE	Range.	Xmin	Xmax-Xmin
RESCALE	Rescale using specified lower bound, upper bound, or both. See syntax after this table.	See table after RESCALE syntax.	See table after RESCALE syntax.
STD	Standard deviation.	XMean	$\sqrt{(\sum((X_i - X_{\text{mean}})^2)/N)}$ where <i>N</i> is count of valid values.

<i>scale_method</i>	Description	<i>location</i>	<i>scale</i>
SUM	Sum.	0	ΣX
USTD	Unbiased standard deviation.	XMean	$\sqrt{(\sum((X_i - X_{\text{mean}})^2) / (N - 1))}$ where N is count of valid values.

```
RESCALE ({ lb=lower_bound | ub=upper_bound | lb=lower_bound,
ub=upper_bound })
```

RESCALE *location* and *scale*

	<i>location</i>	<i>scale</i>
Lower bound only	$X_{\text{min}} - \text{lower_bound}$	1
Upper bound only	$X_{\text{max}} - \text{upper_bound}$	1
Lower and upper bounds	$X_{\text{min}} - (\text{lower_bound} / (\text{upper_bound} - \text{lower_bound}))$	$(X_{\text{max}} - X_{\text{min}}) / (\text{upper_bound} - \text{lower_bound})$

Intercept

[Optional] Specify either one *intercept* for all target columns or one *intercept* for each *target_column*. The function uses the *nth intercept* for the *nth target_column*.

Default: '0'

Multiplier

[Optional] Specify either one *multiplier* for all target columns or one *multiplier* for each *target_column*. The function uses the *nth multiplier* for the *nth target_column*.

Default: '1'

GlobalScale

[Optional] Specify whether to scale all target columns to the same location and scale.

Default: 'false' (scale each target column separately)

MissValue

[Optional] Specify how to handle NULL values:

Option	Description
KEEP (Default)	Keep NULL values.

Option	Description
ZERO	Replace each NULL value with 0.
LOCATION	Replace each NULL value with its location value.

TD_ScaleFit Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	BYTEINT, SMALLINT, INTEGER, BIGINT, Decimal /Numeric, Float, Real, Double precision	Column for which to output statistics.

TD_ScaleFit Output

Output Table Schema

Column	Data Type	Description
TD_STATTYPE_SCLFIT	VARCHAR (CHARACTER SET LATIN)	Statistic names and parameters—see following table..
<i>target_column</i>	DOUBLE PRECISION	Statistics values for <i>target_column</i> .

TD_STATTYPE_SCLFIT Statistic Names

Statistic Name	Description
min	Minimum value in <i>target_column</i> .
max	Maximum value in <i>target_column</i> .
sum	Sum of non-NULL values in <i>target_column</i> .
count	Count of non-NULL values in <i>target_column</i> .
null	Count of NULL values in <i>target_column</i> .
avg	Average of non-NULL values in <i>target_column</i> .
variance	Variance of non-NULL values in <i>target_column</i> , calculated according to N-1 degrees of freedom. [N is count of non-NULL values in <i>target_column</i>].
ustd	Unbiased Standard deviation of non-NULL values in <i>target_column</i> .
std	Standard deviation of non-NULL values in <i>target_column</i> .
multiplier	<i>multiplier</i> for <i>target_column</i> .

Statistic Name	Description																		
intercept	<i>intercept</i> for <i>target_column</i> .																		
methodnumbermapping	Identifier of <i>scale_method</i> for <i>target_column</i> : <table> <tr> <th>methodnumbermapping</th><th><i>scale_method</i></th></tr> <tr><td>0</td><td>MEAN</td></tr> <tr><td>1</td><td>SUM</td></tr> <tr><td>2</td><td>USTD</td></tr> <tr><td>3</td><td>STD</td></tr> <tr><td>4</td><td>RANGE</td></tr> <tr><td>5</td><td>MIDRANGE</td></tr> <tr><td>6</td><td>MAXABS</td></tr> <tr><td>7</td><td>RESCALE</td></tr> </table>	methodnumbermapping	<i>scale_method</i>	0	MEAN	1	SUM	2	USTD	3	STD	4	RANGE	5	MIDRANGE	6	MAXABS	7	RESCALE
methodnumbermapping	<i>scale_method</i>																		
0	MEAN																		
1	SUM																		
2	USTD																		
3	STD																		
4	RANGE																		
5	MIDRANGE																		
6	MAXABS																		
7	RESCALE																		
global_true or global_false	GlobalScale value.																		
missvalue_keep, missvalue_zero, or missvalue_location	MissValue value.																		

TD_ScaleFit Example

InputTable: scale_input_table

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

```

passenger survived pclass name                sex  age sibsp
parch ticket  fare      cabin      embarked
-----
0      97      0      1 Goldschmidt; Mr. George B      male  71
0      0 PC 17754 34.6542      A5      C
0      488      0      1 Kent; Mr. Edward Austin      male  58
0      0 11771  29.7      B37      C
0      505      1      1 Maioni; Miss. Roberta      female 16
0      0 110152 86.5      B79      S
0      631      1      1 Barkworth; Mr. Algernon Henry Wilson male  80
0      0 27042  30      A23      S

```

0	873	0	1 Carlsson; Mr. Frans Olof	male	33
0	0 695	5	B51 B53 B55 S		

SQL Call

```
SELECT * FROM TD_ScaleFit (  
  ON scale_input_table AS InputTable  
  OUT PERMANENT TABLE OutputTable (scaleFitOut)  
  USING  
  TargetColumns ('fare')  
  MissValue ('keep')  
  ScaleMethod ('range')  
  GlobalScale ('f')  
) AS dt;
```

Output

TD_STATTYPE_SCLFIT
fare

min
5.000000000
max
86.500000000
sum
185.854200000
count
5.000000000
null
0.000000000
avg
37.170840000
multiplier
1.000000000
intercept
0.000000000
location
5.000000000
scale
81.500000000
globalscale_false

```

        null
ScaleMethodNumberMapping:
[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]    4.0000000000

missvalue_KEEP
        null

```

TD_ScaleTransform

TD_ScaleTransform scales specified input table columns, using [TD_ScaleFit](#) output.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_ScaleTransform Syntax

```

SELECT * FROM TD_ScaleTransform (
  ON { table | view | (query) } AS InputTable
    [ PARTITION BY ANY [ ORDER BY order_column ] ]
  ON { table | view | (query) } AS FitTable DIMENSION
  [ USING
    Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...])
  ]
) AS alias;

```

TD_ScaleTransform Syntax Elements

Accumulate

[Optional] Specify InputTable columns to copy to the output table.

TD_ScaleTransform Input

InputTable Schema

See [TD_ScaleFit Input](#).

FitTable Schema

See [TD_ScaleFit Output](#).

TD_ScaleTransform Output**Output Table Schema**

Column	Data Type	Description
<i>accumulate_column</i>	Same as in InputTable	Column copied from InputTable.
<i>target_column</i>	DOUBLE PRECISION	Scaled values.

TD_ScaleTransform Example

- InputTable: input_table, as in [TD_ScaleFit Example](#)
- FitTable: scaleFitOut, output by [TD_ScaleFit Example](#)

SQL Call

```
SELECT * FROM TD_scaleTransform (
  ON scale_input_table AS InputTable
  ON scaleFitOut AS FitTable DIMENSION
  USING
    Accumulate ('passenger')
) AS dt ORDER BY 1;
```

Output

passenger fare

```
-----
    97 0.363855214723926
   488 0.303067484662577
   505 1
   631 0.306748466257669
   873 0
```

Feature Engineering Utility Functions

Feature engineering utility functions for analyzing and extracting features of the input data set.

TD_FillRowID

TD_FillRowID adds a column of unique row identifiers to the input table.

Note:

This function may not return the same RowIDs if the function is run multiple times.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.
-

TD_FillRowID Syntax

```
SELECT * FROM TD_FillRowID (
  ON { table | view | (query) } AS InputTable [ PARTITION BY ANY [ ORDER BY
order_column ] ]
  USING
  [ RowIDColumnName ('row_id_column') ]
) AS alias;
```

TD_FillRowID Syntax Elements

RowIDColumnName

[Optional] Specify a name for the output column of row identifiers.

Default: row_id

TD_FillRowID Input

InputTable Schema

InputTable can have any schema.

TD_FillRowID Output

Output Table Schema

Column	Data Type	Description
<i>row_id_column</i>	BIGINT	Column of row identifiers.
<i>input_column</i>	Same as in InputTable	Column copied from InputTable.

TD_FillRowID Example

InputTable: titanic

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

```
Survived Pclass Name          Age
-----
0        3 Mrs. Jacques Heath 35
0        3 Mr. Owen Harris    22
1        3 Mrs. Laina         26
1        1 Mrs. John Bradley   38
```

SQL Call

```
SELECT * FROM TD_FillRowID (
  ON fillrowid_input AS InputTable
  USING
  RowIDColumnName ('PassengerId')
) AS dt;
```

Output

```
Survived Pclass Name          Age PassengerId
-----
1        1 Mrs. John Bradley   38          0
```

0	3	Mr. Owen Harris	22	7
1	3	Mrs. Laina	26	8
0	3	Mrs. Jacques Heath	35	15

TD_NumApply

TD_NumApply applies a specified numeric operator to the specified input table columns. For the list of numeric operators, see [TD_NumApply Syntax Elements](#).

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

Related Information:

[TD_StrApply](#)

[TD_FunctionFit](#)

[TD_FunctionTransform](#)

TD_NumApply Syntax

```
SELECT * FROM TD_NumApply (
  ON { table | view | (query) } AS InputTable [ PARTITION BY ANY [ ORDER BY
order_column ] ]
  USING
  TargetColumns ({ 'target_column' | target_column_range }[,...])
  [ OutputColumns ('output_column' [,...]) ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  ApplyMethod ('num_operator')
  [ SigmoidStyle ({ 'logit' | 'modifiedlogit' | 'tanh' }) ]
  [ InPlace ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
) AS alias;
```

TD_NumApply Syntax Elements

TargetColumns

Specify the names of the InputTable columns to which to apply the numeric operator.

OutputColumns

[Ignored with InPlace ('true'), otherwise optional.] Specify names for the output columns. An *output_column* cannot exceed 128 characters.

Default: With InPlace ('false'), *target_column_operator*; otherwise *target_column*

Note:

If any *target_column_operator* exceeds 128 characters, specify an *output_column* for each *target_column*.

Accumulate

[Optional] Specify the names of the InputTable columns to copy to the output table.

With InPlace ('true'), no *target_column* can be an *accumulate_column*.

ApplyMethod

Specify one of these numeric operators:

<i>num_operator</i>	Description
EXP	Raises e (base of natural logarithms) to power of value, where e = 2.71828182845905.
LOG	Computes base 10 logarithm of value.
SIGMOID	Applies sigmoid function to value. See SigmoidStyle.
SININV	Computes inverse hyperbolic sine of value.
TANH	Computes hyperbolic tangent of value.

SigmoidStyle

[Required with ApplyMethod ('sigmoid'), otherwise ignored.] Specify the sigmoid style.

Default: logit

InPlace

[Optional] Specify whether the output columns have the same names as the target columns.

InPlace ('true') effectively replaces each value in each target column with the result of applying *num_operator* to it.

InPlace ('false') copies the target columns to the output table and adds output columns whose values are the result of applying *num_operator* to each value.

With InPlace ('true'), no *target_column* can be an *accumulate_column*.

Default: true

TD_NumApply Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	NUMERIC	Column to which to apply <i>num_operator</i> .
<i>accumulate_column</i>	Any	Column to copy to output table.

TD_NumApply Output

Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in InputTable	Column copied from InputTable.
<i>output_column</i>	Same as in InputTable	Column to which <i>num_operator</i> was applied. With InPlace ('true'), <i>output_column</i> is <i>target_column</i> . With InPlace ('false'), OutputColumns determines <i>output_column</i> .

TD_NumApply Example

InputTable: input_table

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

```

passenger survived pclass sex    age sibsp parch fare
cabin embarked
-----
-----
1          0          3 male    22     1      0  7.250000000
null  S
2          1          1 female  38     1      0 71.280000000
C85    C
3          1          3 female  26     0      0  7.930000000
null  S
4          1          1 female  35     1      0 53.100000000

```

```
C123 S
      5      0      3 male    35      0      0 8.050000000
null S
```

SQL Call

```
SELECT * FROM TD_NumApply (
  ON numApply_input AS InputTable PARTITION BY ANY
  USING
  TargetColumns ('Age','Fare')
  ApplyMethod ('log')
  Accumulate ('Passenger')
  InPlace ('true')
) AS dt;
```

Output

passenger	age	fare
5	3.555348061	2.085672091
4	3.555348061	3.972176928
3	3.258096538	2.070653036
1	3.091042453	1.981001469
2	3.637586160	4.266615783

TD_RoundColumns

TD_RoundColumns rounds the values of each specified input table column to a specified number of decimal places.

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

TD_RoundColumns Syntax

```
SELECT * FROM TD_RoundColumns (
  ON { table | view | (query) } AS InputTable
  USING
  TargetColumns ({ 'target_column' | target_column_range }[,...])
  [ PrecisionDigit (precision) ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
) AS alias;
```

TD_RoundColumns Syntax Elements

TargetColumns

Specify the names of the InputTable columns in which to round every value to *precision* digits.

PrecisionDigit

[Optional] Specify the number of decimal places to which to round values.

If *precision* is positive, the function rounds values to the right of the decimal point.

If *precision* is negative, the function rounds values to the left of the decimal point.

Default: If the PrecisionDigit value is not provided, the function rounds the column values to 0 places.

Note:

If the column values have the DECIMAL/NUMERIC data type with a precision less than 38, then the function increases the precision by 1. For example, when a DECIMAL (4,2) value of 99.99 is rounded to 0 places, the function returns a DECIMAL (5,2) value, 100.00. However, if the precision is 38, then the function only reduces the scale value by 1 unless the scale is 0. For example, the function returns a DECIMAL (38, 36) value of 99.999999999 as a DECIMAL (38, 35) value, 100.00.

Accumulate

[Optional] Specify the names of the InputTable columns to copy to the output table.

TD_RoundColumns Input

InputTable Schema

Column	Data Type	Description
<i>target_column</i>	NUMERIC	Column in which to round every value to <i>precision</i> digits.
<i>accumulate_column</i>	Any	Column to copy to output table.

TD_RoundColumns Output

Output Table Schema

Column	Data Type	Description
<i>target_column</i>	NUMERIC	Column in which every value is rounded to <i>precision</i> digits.
<i>accumulate_column</i>	Same as in InputTable	Column copied from InputTable.

TD_RoundColumns Example

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

InputTable: titanic

```

passenger pclass  fare      survived
-----
      1      3   7.25         0
      2      1  71.2833        1
      3      3   7.925        1
      4      1  53.100        1
      5      3   8.050         0

```

SQL Call

```

SELECT * FROM TD_RoundColumns (
  ON titanic AS InputTable
  USING
    TargetColumns ('Fare')
    PrecisionDigit (1)

```

```
Accumulate ('[0:1]','Survived')
) AS dt;
```

Output

passenger	pclass	survived	fare
1	3	0	7.30
2	1	1	71.30
3	3	1	7.90
4	1	1	53.10
5	3	0	8.10

TD_StrApply

TD_StrApply applies a specified string operator to the specified input table columns. For the list of string operators, see [TD_StrApply Syntax Elements](#).

Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).

For information about PTCs, see *Teradata Vantage™ - Advanced SQL Engine International Character Set Support*, B035-1125.

- This function does not support KanjiSJIS or Graphic data types.

Related Information:

[TD_NumApply](#)

TD_StrApply Syntax

```
SELECT * FROM TD_strApply (
  ON { table | view | (query) } AS InputTable PARTITION BY ANY
  USING
  TargetColumns ({ 'target_column' | target_column_range }[,...])
  [ OutputColumns ('output_column' [,...]) ]
  [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
  StringOperation (str_operator)
  [ String ('string')]
  [ StringLength ('length') ]
  [ OperatingSide ({ 'Left' | 'Right' })]
  [ IsCaseSpecific ({ 'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0' }) ]
```



```
[ EscapeString ('escape_string')]
[ IgnoreTrailingBlank ({'true'|'t' | 'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ StartIndex ('start_index')]
[ InPlace ({'true'|'t' | 'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})]
) AS alias;
```

TD_StrApply Syntax Elements

TargetColumns

Specify the names of the input table columns to which to apply the string operator.

OutputColumns

[Ignored with Inplace ('true'), otherwise optional.] Specify names for the output columns. An *output_column* cannot exceed 128 characters.

Default: With InPlace ('false'), *target_column_operator*; otherwise *target_column*

Note:

If any *target_column_operator* exceeds 128 characters, specify an *output_column* for each *target_column*.

Accumulate

[Optional] Specify the names of the input table columns to copy to the output table.

With InPlace ('true'), no *target_column* can be an *accumulate_column*.

StringOperation

Specify a *str_operator* from the following table. If *str_operator* requires *string*, *length*, or *start_index*, specify that value with String, StringLength, or StartIndex.

<i>str_operator</i>	Description
CHARTOHEXINT	Converts value to its hexadecimal representation.
GETNCHARS	Returns <i>length</i> characters from value. Option: See OperatingSide.
INITCAP	Capitalizes first letter of value.
STRINGCON	Concatenates <i>string</i> to value.
STRINGINDEX	Returns index of first character of <i>string</i> in value. Options: See IsCaseSpecific.
STRINGLIKE	Returns first string that matches specified pattern if one exists in value. Options: See EscapeString, IsCaseSpecific, IgnoreTrailingBlank.

<i>str_operator</i>	Description
STRINGPAD	Pads value with <i>string</i> to <i>length</i> . Option: See OperatingSide.
STRINGREVERSE	Reverses order of characters in value.
STRINGTRIM	If value contains <i>string</i> , trim <i>string</i> from value. Option: See OperatingSide.
SUBSTRING	Returns substring starting at <i>start_index</i> with <i>length</i> from value.
TOLOWER	Replaces uppercase letters in value with lowercase equivalents.
TOUPPER	Replaces lowercase letters in value with uppercase equivalents.
TRIMSPACES	Trims leading and trailing space characters from value.
UNICODESTRING	Converts LATIN value to UNICODE.

String

[Required when *str_operator* needs *string* argument, ignored otherwise.] Specify *string* argument for *str_operator*:

<i>str_operator</i>	<i>string</i>
STRINGCON	String to concatenate to value.
STRINGINDEX	String for which to return index of its first character in value.
STRINGLIKE	Pattern that describes string to find in value.
STRINGPAD	String with which to pad value.
STRINGTRIM	String to trim from value.

StringLength

[Optional] [Required when *str_operator* needs *length* argument, ignored otherwise.] Specify *length* argument for *str_operator*:

<i>str_operator</i>	<i>length</i>
GETNCHARS	Number of characters to return from value.
STRINGPAD	Length to which to pad value.
SUBSTRING	Substring length.

OperatingSide

[Optional] Applies only when *str_operator* is GETNCHARS, STRINGPAD, or STRINGTRIM. Specifies side of value on which to apply *str_operator*.

Default: left

IsCaseSpecific

[Optional] Applies only when *str_operator* is STRINGINDEX or STRINGLIKE. Specify whether search for *string* is case-specific.

Default: true (search is case-specific)

EscapeString

[Optional] Applies only when *str_operator* is STRINGLIKE. Specify the escape characters.

IgnoreTrailingBlank

[Optional] [Optional] Applies only when *str_operator* is STRINGLIKE. Specify whether to ignore trailing space characters.

StartIndex

[Optional] Applies only when *str_operator* is SUBSTRING. Specify the index of the character at which the substring starts.

InPlace

[Optional] Specify whether the output columns have the same names as the target columns.

InPlace ('true') effectively replaces each value in each target column with the result of applying *str_operator* to it.

InPlace ('false') copies the target columns to the output table and adds output columns whose values are the result of applying *str_operator* to each value.

With InPlace ('true'), no *target_column* can be an *accumulate_column*.

Default: true

TD_StrApply Input**Input Table Schema**

Column	Data Type	Description
<i>target_column</i>	CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE.)	Column to which to apply <i>str_operator</i> .

Column	Data Type	Description
<i>accumulate_column</i>	Any	Column to copy to output table.

TD_StrApply Output

Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in input table	Column copied from input table.
<i>output_column</i>	Same as in input table	Column to which <i>str_operator</i> was applied. With InPlace ('true'), <i>output_column</i> is <i>target_column</i> . With InPlace ('false'), OutputColumns determines <i>output_column</i> .

TD_StrApply Example

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

Input Table: input_table

```

passenger survived pclass  name
sex    age sibsp parch ticket          fare          cabin embarked
-----
-----
      5      0      3 Allen; Mr. William Henry
male   35      0      0 373450          8.050000000 null   S
      4      1      1  1 Futrelle; Mrs. Jacques Heath (Lily May Peel)
female 35      1      0 113803          53.100000000 C123   S
      3      1      3 Heikkinen; Miss. Laina
female 26      0      0 STON/O2. 3101282  7.925000000 null   S
      1      0      3 Braund; Mr. Owen Harris
male   22      1      0 A/5 21171          7.250000000 null   S
      2      1      1  1 Cumings; Mrs. John Bradley (Florence Briggs Thayer)
female 38      1      0 PC 17599          71.283300000 C85    C

```

SQL Call

```
SELECT * FROM TD_strApply (  
ON strApply_input_table as InputTable PARTITION BY ANY  
USING  
TargetColumns ('Sex')  
stringOperation ('toUpper')  
Accumulate('Passenger')  
InPlace('True')  
) as dt order by 1;
```

Output

```
passenger sex  
-----  
1 MALE  
2 FEMALE  
3 FEMALE  
4 FEMALE  
5 MALE
```

Hypothesis Testing Functions

Hypothesis testing functions find the relative likelihood of hypotheses. You can accept the most likely hypotheses and reject the least likely.

Hypothesis Test Components

All hypothesis tests have the following components:

Component	Description
Null hypothesis (H_0)	The null hypothesis is known as a hypothesis of no difference. Example: Experimental drug is no better than placebo. The null hypothesis is accepted or rejected based on a statistical test of the hypothesis.
Alternate hypothesis (H_1)	Hypothesis accepted if null hypothesis is rejected. Example: Experimental drug is more effective than placebo.
Alpha (α) (Also called <i>significance level</i> or <i>Type I error</i> .)	Probability of rejecting null hypothesis when true (value below which null hypothesis is rejected). Most common α values are 0.01, 0.05, and 0.10, corresponding to 99%, 95%, and 90% confidence, respectively. Results are "statistically significant at α ."
Test statistic	Value to which data set is reduced, used in hypothesis test. Its sampling distribution under null hypothesis must be calculable (exactly or approximately), making p_values calculable.
Degrees of freedom	Number of independent pieces of information needed to estimate a population parameter (for example, μ or σ^2) for sample of specified size.
Critical value	Quantile of distribution of test statistic under null hypothesis. Used to determine rejection region.
p_value	Probability of test results at least as extreme as test statistic results observed under assumption that null hypothesis is true. The smaller the p_value, the stronger the evidence against the null hypothesis.
Hypothesis test conclusion	Acceptance or rejection of null hypothesis.

Hypothesis Test Types

A hypothesis test is either:

- *One-tailed or two-tailed*

A one-tailed test can be either *lower-tailed* or *upper-tailed*.

- *One-sample* or *two-sample*
- *Paired* or *unpaired*

Hypothesis Test Term Definitions

Term	Description
One-sample test	Uses one test sample.
One-tailed test	Rejection region is the lower tail or the upper tail of the sampling distribution under the null hypothesis H_0 .
Lower-tailed test	Alternate hypothesis (H_1): $\mu < \mu_0$
Upper-tailed test	Alternate hypothesis (H_1): $\mu > \mu_0$
Two-tailed test	The null hypothesis assumes that $\mu = \mu_0$ where μ_0 is a specified value. Two-tailed test considers both lower and upper tails of distribution of test statistic. Alternate hypothesis (H_1): $\mu \neq \mu_0$
Two-sample test	Uses two test samples.
Paired test	Compares study subjects at two different times. The null and alternative hypotheses are the same as one sample test. The paired test becomes a one-sample test because the test considers the differences between sample values before and after the subjects are exposed to treatment.
Unpaired test	Compares different subjects drawn from two independent populations. H_0 : $\mu_1 = \mu_2$ The alternate hypotheses are as follows: <ul style="list-style-type: none"> • Alternate hypothesis for upper-tailed test (H_1): $\mu_1 > \mu_2$ • Alternate hypothesis for lower-tailed test (H_1): $\mu_1 < \mu_2$ • Two-tailed test $\mu_1 \neq \mu_2$

TD_ChiSq

TD_ChiSq performs Pearson's chi-squared (χ^2) test for independence, which determines if there is a statistically significant difference between the expected and observed frequencies in one or more categories of a contingency table (also called a *cross tabulation*).

Test Type

- One-tailed, upper-tailed
- One-sample
- Unpaired

Computational Method

The Chi-Square test finds statistically significant associations between categorical variables. The test determines if the categorical variables are statistically independent or not.

The data for analysis is organized in a table known as contingency tables. A two-way contingency table consists of r rows and c columns wherein:

- The rows correspond to variable 1 that consists of r categories
- The columns correspond to variable 2 that consists of c categories

Each cell of the contingency table is the count of the joint occurrence of particular levels of variable 1 and variable 2.

For example, the following two-way contingency table shows the categorical variable Gender with two levels (Male, Female) and the categorical variable Affiliation with two levels (Smokers, Non-smokers).

Gender Affiliation table

Gender	Affiliation	
	Smokers	Non-Smokers
Male	n_{11}	n_{12}
Female	n_{21}	n_{22}

The cell counts n_{ij} , $i = 1, 2$; $j = 1, 2$ are number of joint occurrences of Gender and Affiliation at their i^{th} and the j^{th} levels respectively. The Null and alternative hypotheses H_0 and H_1 corresponding to a χ^2 test of independence is as follows:

H_0 : The two categorical variables are independent

vs

H_1 : The two categorical variables are not independent

Using the above table, the expected cell counts are calculated:

$$e_{11} = n_{11} + n_{21}$$

$$e_{12} = n_{11} + n_{12}$$

$$e_{21} = n_{21} + n_{22}$$

$$e_{22} = n_{12} + n_{22}$$

The χ^2 test statistic is calculated as:

$$\chi^2_{stat} = \sum_{i=1}^r \sum_{j=1}^c \frac{(n_{ij} - e_{ij})^2}{e_{ij}}$$

The χ^2 statistic follows a Chi-Square distribution with $r - 1$ and $c - 1$ degrees of freedom. In the Gender Affiliation table, $r=2$ and $c=2$. The Null hypothesis H_0 is rejected if $\chi^2_{stat} > \chi^2_{r-1, c-1, \alpha}$ where $\alpha \in \{0.10, 0.05, 0.01\}$.

The Cramer's V statistic is calculated using the following formula:

$$V = \sqrt{\frac{\phi^2}{\min(c-1, r-1)}} = \sqrt{\frac{\chi^2/n}{\min(c-1, r-1)}}$$

where:

- ϕ is the phi coefficient
- χ^2 is derived from the Pearson's chi-squared test
- n is the grand total of observations
- c is the number of columns
- r is the number of rows

The following rules are used to compute the hypothesis conclusion:

- If the chi-square statistic is greater than the critical value, then the function rejects the Null hypothesis.
- If the chi-square statistic is lesser than or equal to the critical value, then the function fails to reject the Null hypothesis.

TD_ChiSq Syntax

```
SELECT * from TD_CHISQ (
  ON { table | view | (query) } AS CONTINGENCY
  [ OUT [ PERMANENT | VOLATILE ] TABLE EXPCOUNTS (expected_values_table) ]
  USING
  [ Alpha (alpha) ]
) AS alias;
```

TD_ChiSq Syntax Elements

expected_values_table

[Optional] Specify a name for the table of expected values. *expected_values_table* cannot be the name of an existing table.

Default behavior: Function does not output this table.

alpha

[Optional] Specify the probability of rejecting the null hypothesis when it is true (value below which null hypothesis is rejected). *alpha* must be a numeric value in the range [0, 1].

Default: 0.05

TD_ChiSq Input

A contingency table also known as a two-way frequency table is a tabular mechanism with at least two rows and two columns used in statistics to present categorical data in terms of frequency counts.

A contingency table shows the observed frequency of two variables arranged into rows and columns. The intersection of a row and a column of a contingency table is called a cell.

For example, a cell count n_{ij} represents a joint occurrence of row i and column j where i is a value between 1 to r (total number of rows) and j is a value between 2 to c (total number of columns).

You can interpret the contingency table in the example as follows:

- The First column represents the first category, gender, and has two labels, female and male which are represented by two rows.
- The second category, habits has two labels, smokers and non-smokers which are represented by the second and third columns.

The second category can have at most 2046 unique labels. The function ignores NULL values in the table.

Maximum label length is 64000 for *category_1*, 128 for all other columns.

For a valid test output, the value of each observed frequency in the CONTINGENCY table must be at least 5.

CONTINGENCY Table Schema

Column	Data Type	Description
<i>Name of categorical column 1</i>	Any	Columns can have one or multiple labels. Can either be an integer, LATIN, or UTF8 code.
<i>category_2_label_1</i>	INTEGER, SMALLINT, BYTEINT, or BIGINT	Joint frequency of category 1 label i and category 2 label 1, where i has a value between 1 to r .
<i>category_2_label_2</i>	INTEGER, SMALLINT, BYTEINT, or BIGINT	Joint frequency of category 1 label i and category 2 label 2, where i has a value between 1 to r .
.		

Column	Data Type	Description
.		
.		
.		
<i>category_2_label_c</i>	INTEGER, SMALLINT, BYTEINT, or BIGINT	[Column appears zero or more times.] Joint frequency of category 1 label i and category 2 label c, where i has a value between 1 to r.

TD_ChiSq Output

Output Table Schema

Column	Data Type	Description
chi_square	DOUBLE PRECISION	Chi-squared statistic.
cramers_v	DOUBLE PRECISION	Cramer's V statistic.
df	INTEGER	Degrees of freedom.
alpha	DOUBLE PRECISION	<i>alpha</i> (see TD_ChiSq Syntax Elements).
p_value	DOUBLE PRECISION	Probability associated with chi-squared statistic.
criticalvalue	DOUBLE PRECISION	Critical value calculated using Alpha for test.
conclusion	VARCHAR	Chi-squared test result, either 'reject null hypothesis' or 'fail to reject null hypothesis'.

Table of Expected Values

The function outputs this table only if you include the OUT clause in the function call. The OUT clause specifies its name.

This table contains the expected frequencies calculated under the assumption that the null hypothesis is true.

This table has the same schema as the CONTINGENCY table that contains the observed frequencies (see [TD_ChiSq Input](#)), except that all columns but the first have the data type DOUBLE PRECISION.

TD_ChiSq Example

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment ① in the left sidebar.

This example tests whether gender influences smoking habits affiliation. The null hypothesis is that gender and smoking habits affiliation are independent. TD_ChiSq compares the null hypothesis (expected frequencies) to the contingency table (observed frequencies).

Input: contingency1

The contingency table contains the frequencies of men and women affiliated with each smoking habit. *category_1*, gender, has labels "female" and "male". *category_2*, habits, has labels "smokers" and "non-smokers".

This example illustrates a two-way contingency table with two categories, *category_1* and *category_2* respectively.

Each row has a label, *i* which has a value between 1 to *r*, and each column has a label, *j* which has a value between 2 to *c*. The values of *c* and *r* are 3 and 2 respectively.

Here, *category_1_label1* corresponds to females and *category_1_label2* corresponds to males. Similarly, *category2_label1* corresponds to smokers and *category2_label2* corresponds to non-smokers.

Query to create the contingency table is as follows:

```
--Query to construct table contingency1
CREATE MULTISET TABLE contingency1 as
(
  select gender as gender
    , sum((case when smokinghabit = 'smoker' then 1 else 0 end)) as smoker_cnt
    , sum((case when smokinghabit= 'nonsmoker' then 1 else 0 end)) as nonsmoker_cnt
  from mytesttable
  group by gender
) with data;

--Alternate Query to construct table contingency1 using PIVOT
CREATE MULTISET TABLE contingency1 as
(
  SELECT *
  FROM (select gender as gender, smokinghabit, count(smokinghabit)
  as smokinghabit_count
  from mytesttable group by gender, smokinghabit) as mytesttable
  PIVOT ( SUM(smokinghabit_count) as smokinghabit_cnt
  FOR smokinghabit
          IN ('smoker' AS smoker, 'nonsmoker' AS nonsmoker))Tmp
) with data;
```

gender	Habits	
	smokers	non-smokers
-----	-----	-----

female	6	9
male	8	5

SQL Call

```
SELECT * from TD_CHISQ (
  ON contingency1 AS CONTINGENCY
  OUT TABLE EXPCOUNTS (exptable1)
  USING
  Alpha (0.05)
) AS dt;
```

Output Table

chi_square p_value	cramers_v criticalvalue	df conclusion	alpha
1.29230769230769E 000	2.14834462211830E-001	1	5.00000000000000E-002
2.55623107546413E-001	3.84145882069412E 000	Fail to reject Null hypothesis	

exptable1

gender	smokers	non-smokers
female	7.50000000000000E 000	7.50000000000000E 000
male	6.50000000000000E 000	6.50000000000000E 000

TD_FTest

TD_FTest performs an *F*-test, for which the test statistic follows an *F*-distribution under the Null hypothesis.

TD_FTest compares the variances of two independent populations. If the variances are significantly different, TD_FTest rejects the Null hypothesis, indicating that the variances may not come from the same underlying population.

Use TD_FTest to compare statistical models that have been fitted to a data set, to identify the model that best fits the population from which the data were sampled.

Assumptions

- Populations from which samples are drawn are normally distributed.
- Populations are independent of each other.
- Data is numeric.

Test Type

- One-tailed (lower and upper-tailed) or two-tailed (your choice)
- Two-sample
- Unpaired

Computational Method

The F-test is used to test the Null hypothesis $\sigma^2 = \sigma_0^2$ in various applications. For example, you might need to test the variability in the measurement of the thickness of a manufactured part in a factory. If the thickness is not equal to a certain thickness (σ_0^2) then you can conclude that the manufacturing process is uncontrolled. The types of hypothesis are as follows:

$$H_0: \sigma^2 = \sigma_0^2$$

versus

$$H_1: \sigma^2 > \sigma_0^2 \text{ (upper-tailed)}$$

or

$$H_1: \sigma^2 < \sigma_0^2 \text{ (lower-tailed)}$$

or

$$H_1: \sigma^2 \neq \sigma_0^2 \text{ (two-tailed)}$$

Let x_1, x_2, \dots, x_n be a random sample. To test the above hypotheses, the test statistic is calculated as:

$$\chi^2 = \frac{(n-1)s^2}{\sigma_0^2}$$

$$\text{where } s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

The statistic χ^2 follows an F distribution with n-1 degrees of freedom.

For the one-sided upper-tailed test $\sigma^2 > \sigma_0^2$ the Null hypothesis H_0 is rejected if $\frac{(n-1)s^2}{\sigma_0^2} > \chi_{n-1, \alpha}^2$.

For the one-sided lower-tailed test $\sigma^2 < \sigma_0^2$, the Null hypothesis H_0 is rejected

$$\text{if } \frac{(n-1)s^2}{\sigma_0^2} < \chi_{n-1, 1-\alpha}^2.$$

For the two-sided alternative $\sigma^2 \neq \sigma_0^2$, the Null hypothesis H_0 is rejected if

$$\frac{(n-1)s^2}{\sigma_0^2} \geq \chi_{n-1, \alpha/2}^2$$

or

$$\frac{(n-1)s^2}{\sigma_0^2} \leq \chi_{n-1, 1-\alpha/2}^2$$

Also, the F-test is used to test if the variances of two populations are equal. The F-test can have the following tests:

- One-tailed test: The test is used to determine if the variance of one population is either greater than (upper-tailed) or less than (lower-tailed) the variance of another population.
- Two-tailed test: The test is used to determine significant differences in variances of the two populations and tests the Null hypothesis (H_0) against the alternative hypothesis (H_1) to find out if the variances are not equal.

Let $x_1, x_2, \dots, x_{n1} \sim N(\mu_1, \sigma^2)$ and $y_1, y_2, \dots, y_{n2} \sim N(\mu_2, \sigma^2)$ be random samples from two independent populations. The corresponding sample means and variances are as follows:

- Sample Means Formula: $\bar{x} = \frac{1}{n_1} \sum_{i=1}^{n_1} x_i$
- Sample Variance Formula: $\bar{y} = \frac{1}{n_2} \sum_{i=1}^{n_2} y_i$
- Sample Variance Formula for S_1^2 and S_2^2 : $s_1^2 = \frac{1}{n_1-1} \sum_{i=1}^{n_1} (x_i - \bar{x})^2$
and $s_2^2 = \frac{1}{n_2-1} \sum_{i=1}^{n_2} (y_i - \bar{y})^2$

In the following calculation, assume that sample 1 has a larger variance than sample 2. If sample 2 has a larger variance than sample 1, switch the samples and apply the same formula.

$$H_0: \sigma_1^2 = \sigma_2^2$$

versus

$$H_1: \sigma_1^2 > \sigma_2^2$$

or

$$\sigma_1^2 < \sigma_2^2$$

The test statistic for the one-sided upper tailed test ($\sigma_1^2 > \sigma_2^2$) is calculated as:

$$\frac{s_1^2}{s_2^2} \sim F_{n_1-1, n_2-1}$$

where: n_1-1 and n_2-1 are degrees of freedom corresponding to sample 1 and sample 2.

The Null hypothesis H_0 is rejected if $\frac{s_1^2}{s_2^2} \geq F_{n_1-1, n_2-1, \alpha}$.

The test statistic for the one-sided lower-tailed test ($\sigma_1^2 < \sigma_2^2$) is calculated as:

$$\frac{s_2^2}{s_1^2} \sim F_{n_2-1, n_1-1}$$

The Null hypothesis H_0 is rejected if $\frac{s_2^2}{s_1^2} \geq F_{n_2-1, n_1-1, \alpha}$.

For the two-sided hypothesis test:

$$H_0: \sigma_1^2 = \sigma_2^2$$

versus

$$H_1: \sigma_1^2 \neq \sigma_2^2$$

The Null hypothesis H_0 is rejected if:

$$\frac{s_1^2}{s_2^2} \geq F_{n_1-1, n_2-1, \alpha/2}$$

and

$$\frac{s_2^2}{s_1^2} \geq F_{n_2-1, n_1-1, \alpha/2}$$

The two-tailed test is based on the upper tail of the F-distribution.

TD_FTest Syntax

```
SELECT * from TD_FTEST (
  [ ON { table | view | (query) } AS InputTable ]
```



```

USING
  first_sample_specifier
  second_sample_specifier
[ AlternativeHypothesis ({ 'lower-tailed' | 'upper-tailed' | 'two-tailed' })
[ Alpha (alpha) ]
) AS alias;

```

first_sample_specifier

```

{ FirstSampleColumn ('sample_column_1') |
  FirstSampleVariance (variance_1)
  DF1 (degrees_of_freedom_first_sample)
}

```

second_sample_specifier

```

{ SecondSampleColumn ('sample_column_2') |
  SecondSampleVariance (variance_2)
  DF2 (degrees_of_freedom_second_sample)
}

```

TD_FTest Syntax Elements

FirstSampleColumn

[Required if you omit FirstSampleVariance, disallowed otherwise.] Specify the name of the input column that contains the data for the first sample population.

FirstSampleVariance

[Required if you omit FirstSampleColumn, disallowed otherwise.] Specify the variance of the first sample population.

DF1

[Required if you omit FirstSampleColumn, disallowed otherwise.] Specify the degrees of freedom of the first sample.

SecondSampleColumn

[Required if you omit SecondSampleVariance, disallowed otherwise.] Specify the name of the input column that contains the data for the second sample population.

SecondSampleVariance

[Required if you omit SecondSampleColumn, disallowed otherwise.] Specify the variance of the second sample population.

DF2

[Required if you omit SecondSampleColumn, disallowed otherwise.] Specify the degrees of freedom of the second sample.

AlternativeHypothesis

[Optional] Specify the alternative hypothesis:

Option	Description
'lower-tailed'	Alternate hypothesis (H_1): $\mu < \mu_0$
'upper-tailed'	Alternate hypothesis (H_1): $\mu > \mu_0$
'two-tailed' (Default)	Rejection region is on two sides of sampling distribution of test statistic. Two-tailed test considers both lower and upper tails of distribution of test statistic. Alternate hypothesis (H_1): $\mu \neq \mu_0$

Alpha

[Optional] Specify the probability of rejecting the null hypothesis when it is true (value below which null hypothesis is rejected). *alpha* must be a numeric value in the range [0, 1].

Default: 0.05

TD_FTest Input

InputTable is required only if you specify either FirstSampleColumn or SecondSampleColumn. If you specify FirstSampleVariance, SecondSampleVariance, DF1, and DF2, the function ignores InputTable.

InputTable Schema

Column	Data Type	Description
<i>sample_column_1</i>	Numeric	Data for first sample population.
<i>sample_column_2</i>	Numeric	Data for second sample population.

TD_FTest Output

Output Table Schema

Column	Data Type	Description
FirstSampleVariance	DOUBLE PRECISION	Variance of first sample population.
SecondSampleVariance	DOUBLE PRECISION	Variance of second sample population.
VarianceRatio	DOUBLE PRECISION	FirstSampleVariance/SecondSampleVariance
DF1	INTEGER	Degrees of freedom of first sample.
DF2	INTEGER	Degrees of freedom of second sample.
CriticalValue	DOUBLE PRECISION	Critical value calculated using Alpha for test.
Alpha	DOUBLE PRECISION	<i>alpha</i> (see TD_FTest Syntax Elements).
p_value	DOUBLE PRECISION	Probability associated with <i>F</i> -test statistic.
Conclusion	VARCHAR	<i>F</i> -test result, either 'reject null hypothesis' or 'fail to reject null hypothesis'.

TD_FTest Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

TD_FTest Example: Specify Two Sample Columns

Input

SQL Call

```
SELECT * FROM td_ftest (
  ON example_table AS InputTable
  USING
    FirstSampleColumn ('A')
    SecondSampleColumn ('B')
) AS dt;
```

Output

```

    firstsamplevariance    secondsamplevariance    varianceratio
df1      df2      CriticalValue      Alpha
p_value Conclusion
-----
-----
-----
1.3856111111111111E 003    5.21221052631579E 002    2.65839436859915E 000
9          19    2.88005204672380E 000    5.00000000000000E-002
6.96767848508086E-002    Fail to reject Null hypothesis

```

TD_FTest Example: Specify Two Sample Variances**Input**

With two sample variances instead of two sample columns, you do not need InputTable.

SQL Call

```

SELECT * FROM td_ftest (
  USING
  FirstSampleVariance (1385.61)
  SecondSampleVariance (521.22)
  DF1 (9)
  DF2 (19)
) AS dt;

```

Output

```

    firstsamplevariance    secondsamplevariance
varianceratio      df1      df2
CriticalValue      Alpha      p_value Conclusion
-----
-----
-----
1.385610000000000E 003    5.21220000000000E 002    2.65839760561759E
000          9          19    2.88005204672380E 000
5.00000000000000E-002    6.96764431913391E-002    Fail to reject Null hypothesis

```

TD_ZTest

TD_ZTest performs a Z-test, for which the distribution of the test statistic under the Null hypothesis can be approximated by normal distribution.

TD_ZTest tests the equality of two means under the assumption that the population variances are known (rarely true). For large samples, sample variances approximate population variances, so TD_ZTest uses sample variances instead of population variances in the test statistic.

Assumptions

- Sample distribution is normal.
- Data is numeric, not categorical.

Test Type

- One-tailed or two-tailed (your choice)
- One-sample or two-sample (your choice)

Use one-sample to test whether the mean of a population is greater than, less than, or not equal to a specific value. TD_ZTest finds the answer by comparing the critical values of the normal distribution at levels of significance ($\alpha = 0.01, 0.05, 0.10$) to the Z-test statistic.

- Unpaired

Computational Method

A test of the hypothesis (ToH) involves the following framework:

- A Null hypothesis H_0 and an alternative hypothesis H_1
- A random sample x_1, x_2, \dots, x_n in the case of a one sample test
- Two random samples x_1, x_2, \dots, x_n and y_1, y_2, \dots, y_n in the case of a two sample test
- A test statistic Z_{stat}
- A level of significance $\alpha \in \{0.10, 0.05, 0.01\}$
- Compare the sample based Z_{stat} with the percentage point of the normal distribution $|z|$ or $|z_{\alpha/2}|$
- Compute the p-value
- Conclusion

One Sample Z-Tests

Let x_1, x_2, \dots, x_n be a random sample drawn from a population with mean μ and variance σ^2 . Also, assume that the data follows a normal distribution $N(\mu, \sigma^2)$.

$H_0; \mu \leq \mu_0$

versus

$H_1; \mu > \mu_0$

or

$$H_0: \mu \geq \mu_0$$

versus

$$H_1: \mu < \mu_0$$

$$H_0: \mu = \mu_0$$

versus

$$H_1: \mu \neq \mu_0$$

The test statistic for testing the above hypotheses is the Z-stat. The validity of the Z-stat is predicated on the assumption that the population variance σ^2 is known.

The assumption of known variance is not practical because if the variance is known, then the mean μ is known. So, if the mean μ is known, the test is not required.

However, for large sample sizes (which is common in Big data applications), the sample variance s^2 is approximately equal to the unknown variance σ^2 . Therefore, a scenario that involves a large sample size validates the application of the Z-statistic.

The z-statistic is calculated as:

$$Z_{stat} = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$$

where the unknown standard deviation σ is replaced by the sample standard deviation

$$s = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$$

as $n \rightarrow \infty$ (sample size is very large). Therefore, the z-statistic is rewritten as:

$$Z_{stat} = \frac{\bar{x} - \mu_0}{s / \sqrt{n}}$$

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

where

In case I of the upper tailed hypothesis test, the Null hypothesis is rejected if $Z_{stat} > z_{\alpha}$ where $\alpha \in \{0.10, 0.05, 0.01\}$. In case II of the lower tailed hypothesis test, the Null hypothesis is rejected if $Z_{stat} < z_{\alpha}$ where $\alpha \in \{0.10, 0.05, 0.01\}$. In case III of the two-tailed test, the Null hypothesis is rejected if $Z_{stat} > z_{\alpha/2}$ and $Z_{stat} < z_{\alpha/2}$, $\alpha \in \{0.10, 0.05, 0.01\}$.

Two Sample Z tests

The two sample z-test is used for testing equality of means of two populations. Let $x_1, x_2, \dots, x_{n1} \sim N(\mu_1, \sigma_1^2)$ and $y_1, y_2, \dots, y_{n2} \sim N(\mu_2, \sigma_y^2)$ be random samples from two independent populations. The Null hypothesis H_0 and the alternative hypothesis H_1 respectively for a one-sided lower-tailed test is given as:

$$H_0: \mu_1 \geq \mu_2$$

versus

$$H_1: \mu_1 < \mu_2$$

$$Z_{stat} = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_1^2}{n_1}}}$$

The Null hypothesis is rejected if $Z_{stat} < -z_\alpha$ where $\alpha \in \{0.10, 0.05, 0.01\}$. Also, note that $-z_\alpha$ is a percentile of the normal distribution with area to its left.

A one-sided upper-tailed test is calculated as:

$$H_0: \mu_1 \leq \mu_2$$

versus

$$H_1: \mu_1 > \mu_2$$

$$Z_{stat} = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_1^2}{n_1}}}$$

The Null hypothesis is rejected if $Z_{stat} > z_\alpha$ with $\alpha \in \{0.10, 0.05, 0.01\}$. Also, note that z_α is a percentile of the normal distribution with $(1 - \alpha) \times 100$ area to its left. So, $-z_\alpha$ puts 100α area to its left.

$$H_0: \mu_1 = \mu_2$$

versus

$$H_1: \mu_1 \neq \mu_2$$

$$Z_{stat} = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

The Null hypothesis is rejected if $Z_{stat} > z_{1-\alpha/2}$ or $Z_{stat} < -z_{\alpha/2}$ with $\alpha \in \{0.10, 0.05, 0.01\}$. Also, note that $z_{1-\alpha/2}$ is a percentile of the normal distribution with $(1 - \alpha/2) \times 100$ area to its left. So, $-z_{\alpha}$ puts 100α area to its left. Note $Z_{stat} \sim N(0,1)$.

TD_ZTest Syntax

```
SELECT * FROM TD_ZTest (
  ON { table | view | (query) }
  USING
  FirstSampleColumn (sample_column_1)
  [ FirstSampleVariance (variance_1) ]
  [ SecondSampleColumn (sample_column_2) ]
  [ SecondSampleVariance (variance_2) ]
  [ AlternativeHypothesis ({ 'upper-tailed' | 'lower-tailed' | 'two-tailed' }) ]
  [ MeanUnderH0 (mean_under_H0) ]
  [ Alpha (alpha) ]
) AS dt;
```

TD_ZTest Syntax Elements

FirstSampleColumn

Specify the name of the input column that contains the data for the first sample population.

FirstSampleVariance

[Required if first sample size is less than 30, optional otherwise.] Specify the variance of the first sample population. *variance_1* is a numeric value in the range (0,1.79769e+308).

Default behavior: If sample size is greater than 30, the function approximates the variance.

SecondSampleColumn

[Optional] Specify the name of the input column that contains the data for the second sample population.

SecondSampleVariance

[Required if you specify SecondSampleColumn and second sample size is less than 30, optional otherwise.] Specify the variance of the second sample population. *variance_2* is a numeric value in the range (0, 1.79769e+308).

Default behavior: If sample size is greater than 30, the function approximates the variance.

AlternativeHypothesis

[Optional] Specify the alternative hypothesis:

Option	Description
'lower-tailed'	Alternate hypothesis (H_1): $\mu < \mu_0$
'upper-tailed'	Alternate hypothesis (H_1): $\mu > \mu_0$
'two-tailed'	Rejection region is on two sides of sampling distribution of test statistic. Two-tailed test considers both lower and upper tails of distribution of test statistic. Alternate hypothesis (H_1): $\mu \neq \mu_0$

Default: 'two-tailed'

MeanUnderH0

[Optional] Specify the mean under the null hypothesis (H_0). *mean_under_H0* is a numeric value in the range (-1.79769e+308, 1.79769e+308).

Default: 0

Alpha

[Optional] Specify the probability of rejecting the null hypothesis when it is true (value below which null hypothesis is rejected). *alpha* must be a numeric value in the range [0, 1].

The null hypothesis is rejected if $p_value < alpha$. (For a description of *p_value*, see [TD_ZTest Output](#).) If the null hypothesis is rejected, the rejection *confidence level* is $1-alpha$.

Default: 0.05

TD_ZTest Input

Input Table Schema

Column	Data Type	Description
<i>sample_column_1</i>	INTEGER, BYTEINT, SMALLINT, BIGINT, DOUBLE PRECISION, NUMERIC, NUMBER	Data for first sample population.
<i>sample_column_2</i>	INTEGER, BYTEINT, SMALLINT, BIGINT, DOUBLE PRECISION, NUMERIC, NUMBER	[Optional] Data for second sample population.

TD_ZTest Output

Output Table Schema

Column	Data Type	Description
<i>sample_column_1</i>	VARCHAR	Data for first sample population.
<i>sample_column_2</i>	VARCHAR	[Column appears only if you specify SecondSampleColumn.] Data for second sample population.
N1	INTEGER	Size of first sample.
N2	INTEGER	[Column appears only if you specify SecondSampleColumn.] Size of second sample.
mean1	DOUBLE PRECISION	Mean of first sample.
mean2	DOUBLE PRECISION	[Column appears only if you specify SecondSampleColumn.] Mean of second sample.
AlternativeHypothesis	VARCHAR	Hypothesis accepted if null hypothesis is rejected (H_1).
z_score	DOUBLE PRECISION	Test statistic z score.
Alpha	DOUBLE PRECISION	<i>alpha</i> (see TD_ZTest Syntax Elements).
CriticalValue	DOUBLE PRECISION	Critical value calculated using Alpha for test (z_α).
p_value	DOUBLE PRECISION	Probability associated with Z-test statistic: <ul style="list-style-type: none"> One-tailed, lower-tailed test: $P(z < z_score)$ This is the lower-tail probability under normal distribution. One-tailed, upper-tailed test: $P(z > z_score)$

Column	Data Type	Description
		This is the upper-tail probability under normal distribution. • Two-tailed test: $P(z > z_score) + P(z < z_score)$ If $p_value < \alpha$, reject null hypothesis.
Conclusion	VARCHAR	Z-test result, either 'reject null hypothesis' or 'fail to reject null hypothesis'. If Conclusion is 'reject null hypothesis', rejection confidence level is $1-\alpha$.

TD_ZTest Example

Input: example_table

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 📎 in the left sidebar.

col1	col2
-----	-----
93	12
?	12
22	4
?	87
1	10
?	43
92	31
?	23
2	3
?	52
21	65
?	49
?	17
?	17
?	14
?	24
53	20
85	9
50	11
86	1


```

firstsamplecolumn col1
secondsamplecolumn col2
      N1          10
      N2          20
      mean1      5.05000000000000E 001
      mean2      2.52000000000000E 001
AlternativeHypothesis TWO-TAILED
      z_score    1.55377718139113E 002
      Alpha      5.00000000000000E-002
      CriticalValue 1.95996398454005E 000
      p_value    0.00000000000000E 000
      Conclusion  Reject Null hypothesis

```

How to Read Syntax

This document uses the following syntax conventions.

Syntax Convention	Meaning
KEYWORD	Keyword. Spell exactly as shown. Many environments are case-insensitive. Syntax shows keywords in uppercase unless operating system restrictions require them to be lowercase or mixed-case.
<i>variable</i>	Variable. Replace with actual value.
<i>number</i>	String of one or more digits. Do not use commas in numbers with more than three digits. Example: 10045
[x]	x is optional.
[x y]	You can specify x, y, or nothing.
{ x y }	You must specify either x or y.
x [...]	You can repeat x, separating occurrences with spaces. Example: x x x See note after table.
x [, ...]	You can repeat x, separating occurrences with commas. Example: x, x, x See note after table.
x [delimiter...]	You can repeat x, separating occurrences with specified delimiter. Examples: <ul style="list-style-type: none"> If <i>delimiter</i> is semicolon: x; x; x If <i>delimiter</i> is { , OR }, you can do either of the following: <ul style="list-style-type: none"> x, x, x x OR x OR x See note after table.

Note:

You can repeat only the immediately preceding item. For example, if the syntax is:

```
KEYWORD x [...]
```

You can repeat x. Do not repeat KEYWORD.

If there is no white space between x and the delimiter, the repeatable item is x and the delimiter. For example, if the syntax is:

```
[ x, [...] ] y
```

- You can omit x: y
 - You can specify x once: x, y
 - You can repeat x and the delimiter: x, x, x, y
-

Additional Information

Changes and Additions

Date	Release	Description
July 2021	17.10	<p>New Features:</p> <ul style="list-style-type: none"> • TD_ConvertTo • TD_GetRowsWithoutMissingValues • TD_OutlierFilterFit • TD_OutlierFilterTransform • TD_SimpleImputeFit • TD_SimpleImputeTransform • TD_CategoricalSummary • TD_ColumnSummary • TD_GetRowsWithMissingValues • TD_Histogram • TD_QQNorm • TD_UnivariateStatistics • TD_WhichMax • TD_WhichMin • TD_BinCodeFit • TD_BinCodeTransform • TD_FunctionFit • TD_FunctionTransform • TD_OneHotEncodingFit • TD_OneHotEncodingTransform • TD_PolynomialFeaturesFit • TD_PolynomialFeaturesTransform • TD_RowNormalizeFit • TD_RowNormalizeTransform • TD_ScaleFit • TD_ScaleTransform • TD_FillRowID • TD_NumApply • TD_RoundColumns • TD_StrApply • TD_ChiSq • TD_FTest • TD_ZTest
		Enhancements:

Date	Release	Description
		<ul style="list-style-type: none"> • DecisionForestPredict (SQL Engine) function: Changed syntax. • DecisionTreePredict (SQL Engine) function: Changed syntax and output table schema. • GLMPredict (SQL Engine) function: Added column range support for syntax element Accumulate. • NaiveBayesTextClassifierPredict (SQL Engine) function: Changed syntax. • nPath® (SQL Engine) function: Added UNICODE support. • Pack (SQL Engine) function: <ul style="list-style-type: none"> ◦ Added column range support for syntax element TargetColumns. ◦ Added syntax elements Accumulate and ColCast. • StringSimilarity (SQL Engine) function: Added column range support for syntax element Accumulate. • SVMSparsePredict (SQL Engine) function: Changed syntax, input data types, and output table schema. • Unpack (SQL Engine) function: <ul style="list-style-type: none"> ◦ Added column range support for syntax element TargetColumns. ◦ Added syntax element Accumulate.
June 2020	17.00	Initial release.

Teradata Links

Link	Description
https://docs.teradata.com/	Search Teradata Documentation, customize content to your needs, and download PDFs. Customers: Log in to access Orange Books.
https://support.teradata.com	One-stop source for Teradata community support, software downloads, and product information. Log in for customer access to: <ul style="list-style-type: none"> • Community support • Software updates • Knowledge articles
https://www.teradata.com/University/Overview	Teradata education network
https://support.teradata.com/community	Link to Teradata community

Related Documentation

Title	Publication ID
<i>Teradata Vantage™ - Advanced SQL Engine Release Summary</i>	B035-1098

Title	Publication ID
<i>Teradata Vantage™ - SQL Fundamentals</i>	B035-1141
<i>Teradata Vantage™ - SQL Functions, Expressions, and Predicates</i>	B035-1145
<i>Teradata Vantage™ - SQL Data Manipulation Language</i>	B035-1146
<i>Teradata Vantage™ - Advanced SQL Engine International Character Set Support</i>	B035-1125
<i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i>	B700-4003
<i>Teradata Aster® Database User Guide</i>	