teradata.

# Module 1: Predictive Analytics – Regression

Teradata Vantage Analytics Workshop

ADVANCED

## Objectives

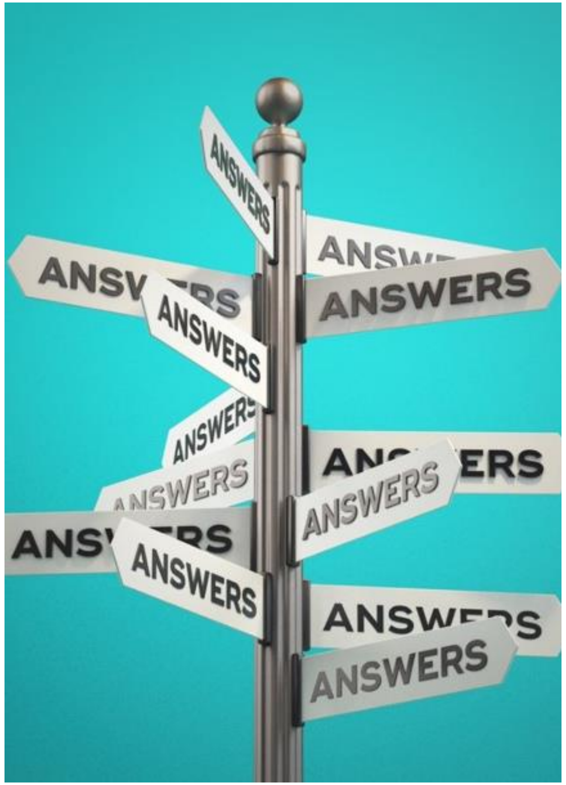
After completing this module, you will be able to:

- Write queries using these Teradata Vantage predictive and other analytic functions:
  - CRISP-DM methodology
  - GLM with Bring Your Own Model
  - Logistic Regression with Vantage Analytics Library

For more info go to docs.teradata.com

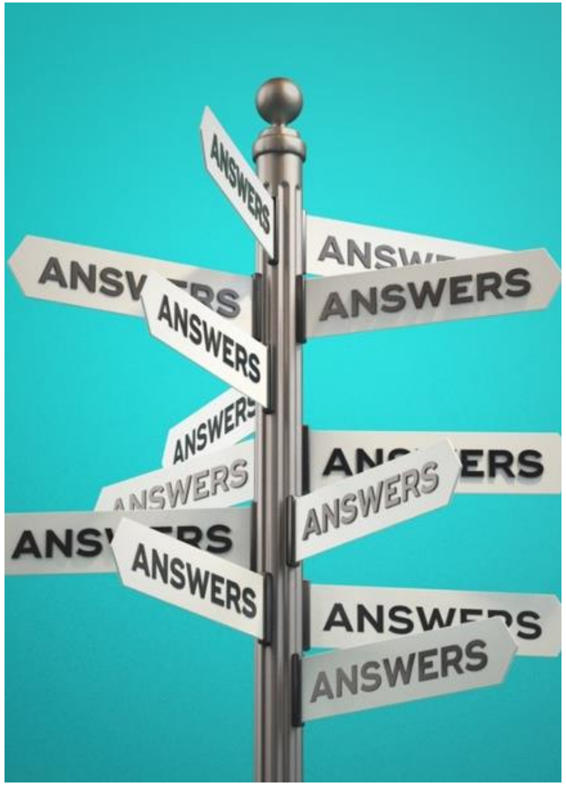
## Topics

- CRISP-DM
- GLM
- Logistic Regression
- Review & Summary



## Current Topic – CRISP-DM

- **CRISP-DM**
- GLM
- Logistic Regression
- Review & Summary



# Data Science Process (CRISP DM) – Six Steps

## CRoss Industry Standard Process for Data Mining

1. You must be clear on the business goal of the analytic request, the success criteria and Data Science goal(s)

2. You must know the underlying data. Size (#rows)? What's in the data (Columns)? Data sources? Schema type - structured or unstructured? Verify data quality. Use Vantage functions and other tools/techniques to examine the data

3. Does the data need preparing? Yes/No? Remove outliers? Missing data? Scale? Transform? Organize by geolocation? Perform data preparation using Vantage data preparation functions

4. Which predictive/analytic functions? Which arguments? Based on function, does data need further cleaning/preparing? Execute Vantage ML functions to create models/ analyses

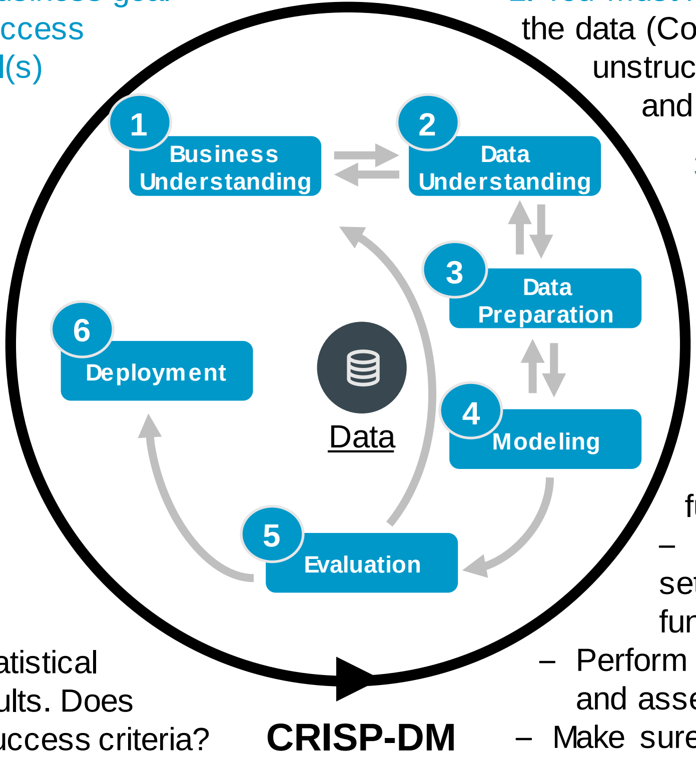
– Build supervised/predictive model(s) on training set, validate on test set, and use Vantage statistical functions to assess accuracy of results

– Perform unsupervised/descriptive analytics on full data set and assess reasonableness of results visually

– Make sure to experiment. Use visualization to assist in model/analysis assessment. Keep track of peripheral findings

5. Compare models / analysis techniques (Using Vantage statistical functions) and select best results. Does it meet your business goals/success criteria? Should it be visualized and/or operationalized?

6. Operationalize and revisit over time. Should it be operationalized? Once you have followed your companies process for operationalizing, you need to consider these monitoring and maintenance questions: Has data changed? Has new data become available? Is the model still predicting accurately? Etc.



## Data Science Process (CRISP DM) – One

1. **Business Understanding**

   The *Business Understanding* phase focuses on understanding the objectives and requirements of the project. Aside from the third task, the three other tasks in this phase are foundational project management activities that are universal to most projects:

   1. **Determine business objectives:** You should first "thoroughly understand, from a business perspective, what the customer really wants to accomplish." (CRISP-DM Guide) and then define business success criteria.
   2. **Assess situation:** Determine resources availability, project requirements, assess risks and contingencies, and conduct a cost-benefit analysis.
   3. **Determine data mining goals:** In addition to defining the business objectives, you should also define what success looks like from a technical data mining perspective.
   4. **Produce project plan:** Select technologies and tools and define detailed plans for each project phase.
      - While many teams hurry through this phase, establishing a strong business understanding is like building the foundation of a house – absolutely essential.

## Data Science Process (CRISP DM) – Two

### 2. Data Understanding

Next is the *Data Understanding* phase. Adding to the foundation of *Business Understanding*, it drives the focus to identify, collect, and analyze the data sets that can help you accomplish the project goals. This phase also has four tasks:

1. **Collect initial data:** Acquire the necessary data and (if necessary) load it into your analysis tool.
2. **Describe data:** Examine the data and document its surface properties like data format, number of records, or field identities.
3. **Explore data:** Dig deeper into the data. Query it, visualize it, and identify relationships among the data.
4. **Verify data quality:** How clean/dirty is the data? Document any quality issues.

# Data Science Process (CRISP DM) – Three

## 3. Data Preparation

This phase, which is often referred to as “data munging”, prepares the final data set(s) for modeling. It has five tasks:

1. **Select data:** Determine which data sets will be used and document reasons for inclusion/exclusion.
2. **Clean data:** Often this is the lengthiest task. Without it, you’ll likely fall victim to garbage-in, garbage-out. A common practice during this task is to correct, impute, or remove erroneous values.
3. **Construct data:** Derive new attributes that will be helpful. For example, derive someone’s body mass index from height and weight fields.
4. **Integrate data:** Create new data sets by combining data from multiple sources.
5. **Format data:** Re-format data as necessary. For example, you might convert string values that store numbers to numeric values so that you can perform mathematical operations.

## Data Science Process (CRISP DM) – Four

4. **Modeling**

   Here you'll likely build and assess various models based on several different modeling techniques. This phase has four tasks:

   1. **Select modeling techniques:** Determine which algorithms to try (e.g., regression, neural net).
   2. **Generate test design:** Pending your modeling approach, you might need to split the data into training, test, and validation sets.
   3. **Build model:** As glamorous as this might sound, this might just be executing a few lines of code like "reg = LinearRegression().fit(X, y)".
   4. **Assess model:** Generally, multiple models are competing against each other, and the data scientist needs to interpret the model results based on domain knowledge, the pre-defined success criteria, and the test design.
      - Although the CRISP-DM guide suggests to "iterate model building and assessment until you strongly believe that you have found the best model(s)", in practice teams should continue iterating until they find a "good enough" model, proceed through the CRISP-DM lifecycle, then further improve the model in future iterations.

# Data Science Process (CRISP DM) – Five

## 5. Evaluation

Whereas the *Assess Model* task of the *Modeling* phase focuses on technical model assessment, the *Evaluation* phase looks more broadly at which model best meets the business and what to do next. This phase has three tasks:

1. **Evaluate results:** Do the models meet the business success criteria? Which one(s) should we approve for the business?
2. **Review process:** Review the work accomplished. Was anything overlooked? Were all steps properly executed? Summarize findings and correct anything if needed.
3. **Determine next steps:** Based on the previous three tasks, determine whether to proceed to deployment, iterate further, or initiate new projects.

## Data Science Process (CRISP DM) – Six

### 6. Deployment

**A model is not particularly useful unless the customer can access its results.** The complexity of this phase varies widely. This final phase has four tasks:

1. **Plan deployment:** Develop and document a plan for deploying the model.
2. **Plan monitoring and maintenance:** Develop a thorough monitoring and maintenance plan to avoid issues during the operational phase (or post-project phase) of a model.
3. **Produce final report:** The project team documents a summary of the project which might include a final presentation of data mining results.
4. **Review project:** Conduct a project retrospective about what went well, what could have been better, and how to improve in the future.
   - Your organization's work might not end there. As a project framework, CRISP-DM does not outline what to do after the project (also known as "operations"). But if the model is going to production, be sure you maintain the model in production. Constant monitoring and occasional model tuning is often required.

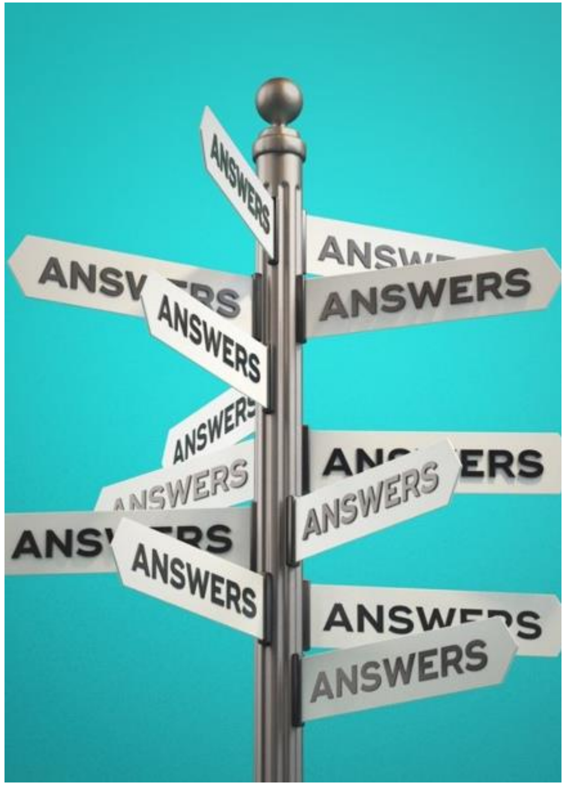
teradata.

## CRISP-DM Data Science Process Worksheet

| Step | Description | Activities | Comments |
|---|---|---|---|
| 1 | **Business Understanding** | Business goal? Success criteria? DS goal? | |
| 2 | **Data Understanding** | What does the data show? Where is it located? Is it complete/correct? | |
| 3 | **Data Preparation** | Outliers? Scale? Organize by geolocation? | |
| 4 | **Modeling/Analysis** | Which functions? Which arguments? Experiment!<br>Assess and compare models/analyses | |
| 5 | **Evaluation** | Are your business goals and criteria being met? Visualize? Pick best performers | |
| 6 | **Deployment** | Operationalize and revisit over time. | |

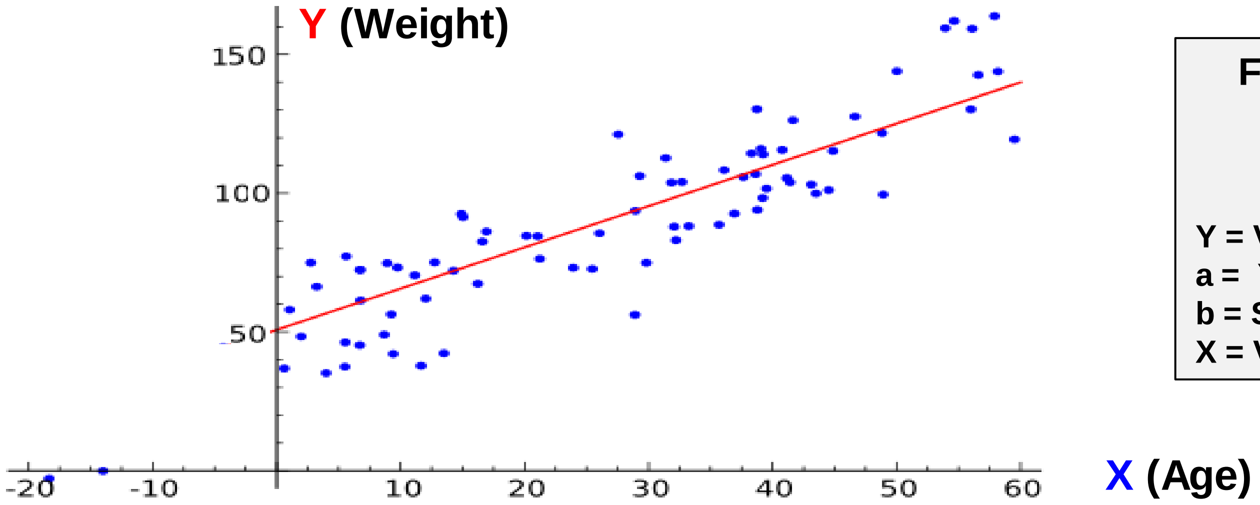
## Current Topic – GLM

- CRISP-DM
- **GLM**
- Logistic Regression
- Review & Summary



## GLM Description

The generalized linear model (GLM) is an extension of the Linear Regression model that enables the linear equation to relate to the dependent variables by a Link function. The GLM function supports several distribution families and Link functions.



**Formula for Predicting Y:**

**$Y = a + bX$**

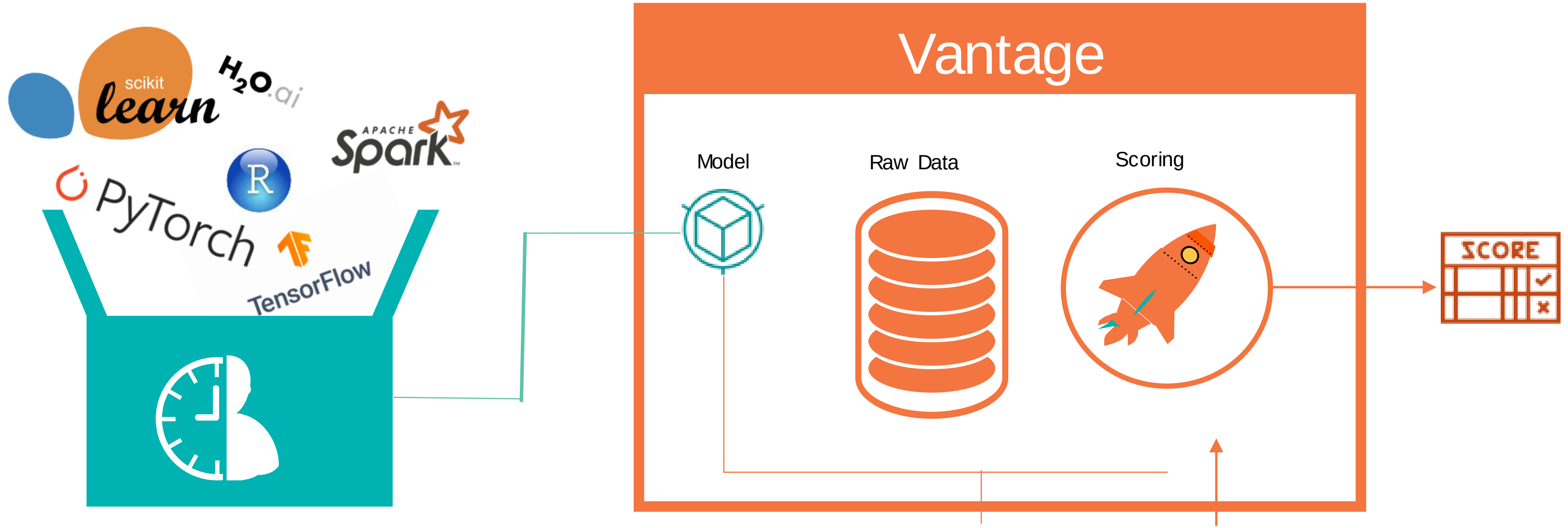
**Y = Value being predicted (Weight)**
**a = Y-intercept (50)**
**b = Slope of line (.3)**
**X = Value you know (Age)**

Cause & Effect: the Independent variable (X) is the Cause, the Dependent variable (Y) is the Effect

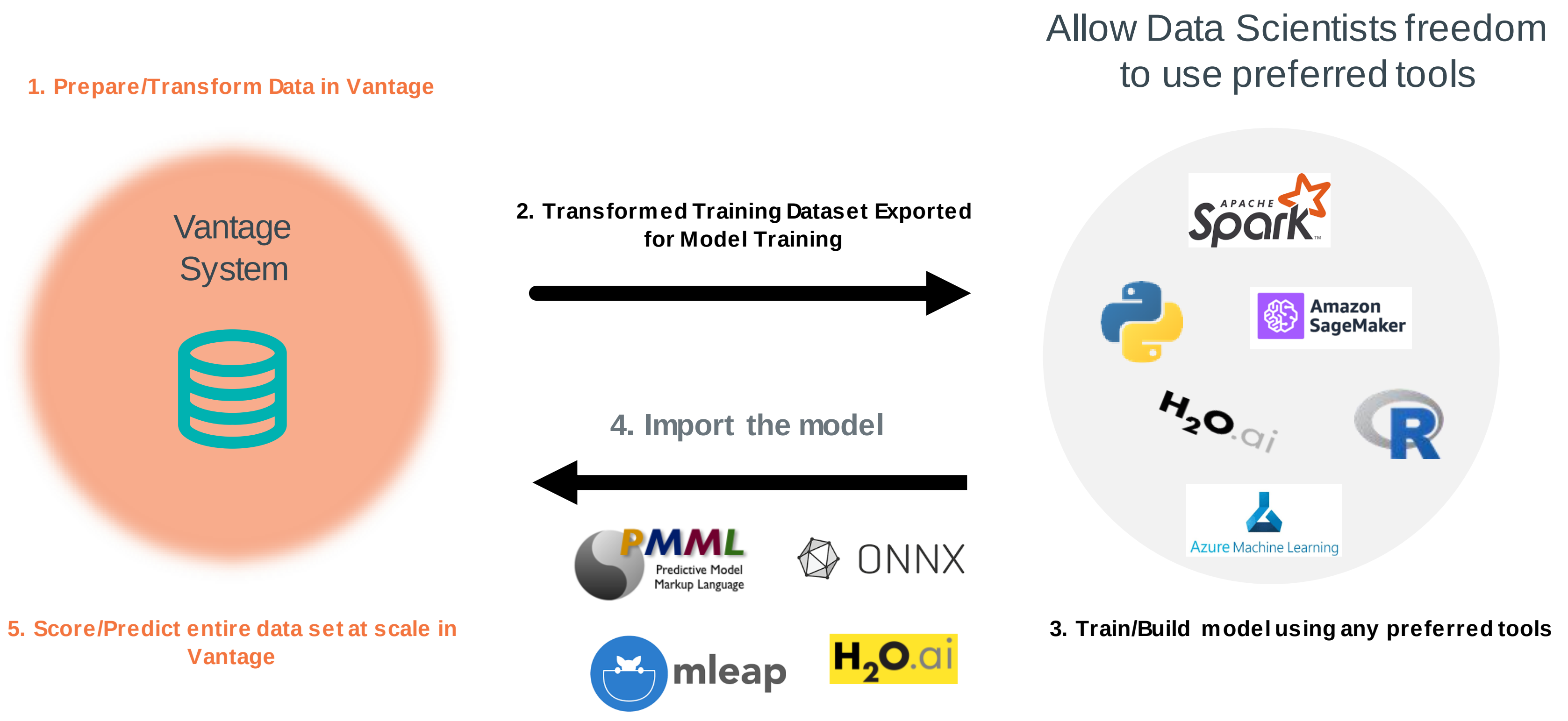
## Bring Your Own Model Vision

Customers can build models in any language using most-popular tools or platforms and score them at scale in Vantage with minimal data movement.

- Import existing Machine Learning models to Vantage



# Bring Your Own Model (BYOM) Workflow


Allow Data Scientists freedom to use preferred tools
1. Prepare/Transform Data in Vantage
Vantage System
2. Transformed Training Dataset Exported for Model Training
4. Import the model
PMML Predictive Model Markup Language
ONNX
mleap
H2O.ai
Apache Spark
Amazon SageMaker
H2O.ai
R
Azure Machine Learning
5. Score/Predict entire data set at scale in Vantage
3. Train/Build model using any preferred tools

teradata.

# Data Science Workflow for Vantage BYOM Feature

MODEL OPERATIONALIZATION

## Data Preparation

- Explore & profile data
- Clean, normalize & transform data
- Derive new variables
- Create Analytic Dataset or Feature Store

## Model Creation

- Export sample data to modelling system
- Identify Machine Learning algorithm
- Fit Model
- Create predictive model using Analytic dataset for training

## Model Scoring

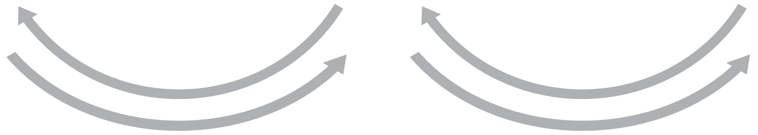
- Export model to standard model format (PMML, MLeap, ONNX, etc.)
- Import model to Vantage
- Score model using BYOM Predict Function
- Evaluate Quality of Predictions

## Environment Re-creation

- Transform production dataset
- Create Analytic Dataset or Feature Store
- Upload Model to production system

## Model Scoring

- Schedule batch and/or on-demand Scoring job
- Setup downstream Business reports and program actions derived from Predicted values
- Periodically review and refine model as needed



# Lab 1a: Jupyter Load Libraries and Java

Install these packages from the Launcher Page. Choose File menu, New Launcher
From Launcher choose Terminal, and enter these commands one at a time:

```
pip install sklearn2pmml
pip install setuptools-git
pip install cmdstanpy==0.4
pip install install-jdk
pip install xgboost
# Run code to the left to find Java location
cd /opt/conda/bin
ln -s /home/jovyan/.jdk/jdk-11.0.12+7/bin/java  java
```

To find Java location:

```
python3
>>>import jdk
>>> jdk.install('11')
# Wait for location of jdk to display
/home/jovyan/.jdk/jdk-11.0.12+7
>>>exit()
```

After done with above commands restart kernel

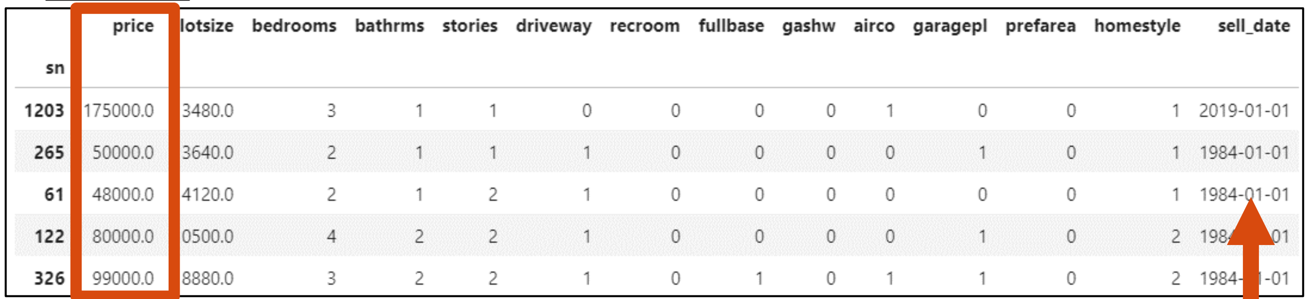
# Lab 1b: View the Data

- Here's the data we'll be using. We will be predicting Y-variable ‘price'. This is a classification model
- There are 12 X-variables, all numeric
- homestyle, lotsize, bedrooms, bathrms, stories, garagepl
- 0=no, 1=yes: driveway, recroom, fullbase, gashw, airco, prefarea

```
train_df = DataFrame.from_query("select *
FROM TRNG_TDU_TD01.housing_train_int")
traid_pd = train_df.to_pandas()
traid_pd
```

Home Style
0 = bungalow
1 = classic
2 = eclectic

Y-var x x x x x x x x x x x x



| | price | lotsize | bedrooms | bathrms | stories | driveway | recroom | fullbase | gashw | airco | garagepl | prefarea | homestyle | sell_date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **sn** | | | | | | | | | | | | | | |
| **1203** | 175000.0 | 3480.0 | 3 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 2019-01-01 |
| **265** | 50000.0 | 3640.0 | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1984-01-01 |
| **61** | 48000.0 | 4120.0 | 2 | 1 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1984-01-01 |
| **122** | 80000.0 | 0500.0 | 4 | 2 | 2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 198[illegible]01 |
| **326** | 99000.0 | 8880.0 | 3 | 2 | 2 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 2 | 1984-[illegible]-01 |

Not used

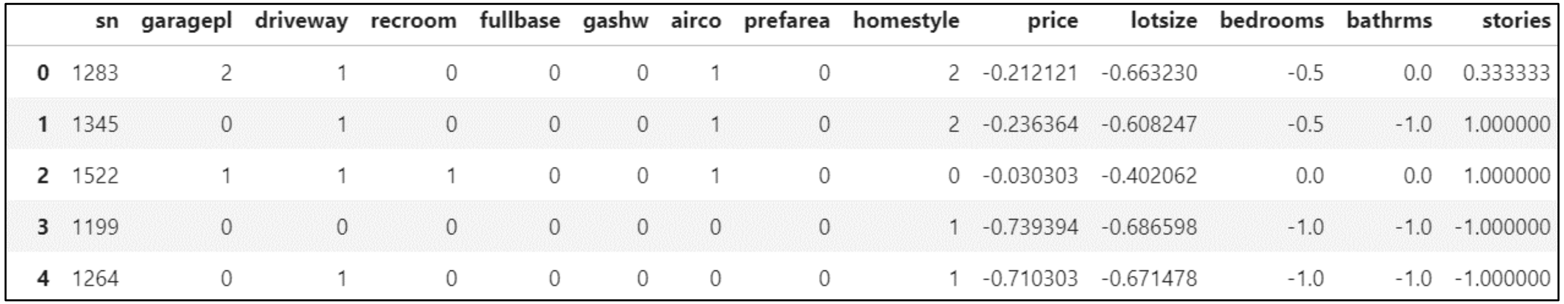
# Lab 1c: Scale the Train Data

- Price and lot size are very large compared to other X variables
- Let's scale all but the yes/no variables and homestyle
- scaleFit and scaleTransform will do this
- scaleFit calculates the statistics and scaleTransform scales the rows

```
con.execute("SELECT * FROM TD_ScaleFit (
ON (SELECT * FROM TRNG_TDU_TD01.housing_train_int WHERE
sell_date EQ '2019-01-01') AS InputTable
OUT PERMANENT TABLE OutputTable (scaleFitOut)
USING
TargetColumns
('price','lotsize','bedrooms','bathrms','stories')
MissValue ('keep')
ScaleMethod ('midrange')
GlobalScale ('f')
) AS dt;")
```

# Lab 1c: Scale the Train Data

```
train_df2 = DataFrame.from_query("SELECT * FROM TD_scaleTransform (
ON (SELECT * FROM TRNG_TDU_TD01.housing_train_int WHERE sell_date
EQ '2019-01-01') AS InputTable
ON scaleFitOut AS FitTable DIMENSION
USING
Accumulate
('sn','garagepl','driveway','recroom','fullbase','gashw','airco','p
refarea','homestyle')
) AS dt;")
traid_pd2 = train_df2.to_pandas()
traid_pd2
```



| | sn | garagepl | driveway | recroom | fullbase | gashw | airco | prefarea | homestyle | price | lotsize | bedrooms | bathrms | stories |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1283 | 2 | 1 | 0 | 0 | 0 | 1 | 0 | 2 | -0.212121 | -0.663230 | -0.5 | 0.0 | 0.333333 |
| **1** | 1345 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 2 | -0.236364 | -0.608247 | -0.5 | -1.0 | 1.000000 |
| **2** | 1522 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | -0.030303 | -0.402062 | 0.0 | 0.0 | 1.000000 |
| **3** | 1199 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | -0.739394 | -0.686598 | -1.0 | -1.0 | -1.000000 |
| **4** | 1264 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | -0.710303 | -0.671478 | -1.0 | -1.0 | -1.000000 |

## Lab 2a: GLM – Create 'Regression' Model

teradata.

WIP

### Define X and Y variables

```
X = traid_pd[['homestyle ','lotsize','bedrooms', 'bathrms', 'stories',
'garagepl', 'driveway', 'recroom', 'fullbase', 'gashw', 'airco', 'prefarea']]

y=traid_pd[[price']]
```

### Pipeline and Export to PMML

```
pipeline = PMMLPipeline([
("classifier", linear_model.Ridge(alpha = 300))
])
pipeline.fit(X, y.values.ravel())
sklearn2pmml(pipeline, "housing_db_glm_model.pmml", with_repr = True)
```

## Lab 2b: GLM – Upload 'Regression' Model

teradata.

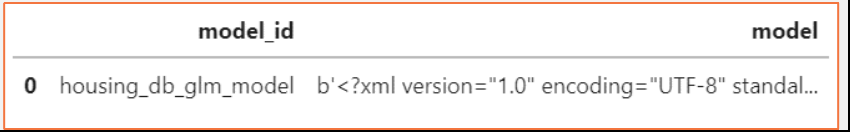
WIP

### Create & Clear model table

```
con.execute("CREATE SET TABLE pmml_models
     (model_id VARCHAR(40) CHARACTER SET LATIN NOT CASESPECIFIC,
      model BLOB(2097088000)) PRIMARY INDEX ( model_id );")
con.execute("delete from pmml_models where model_id = 'housing_db_glm_model'")
```

### Uploading the PMML model

```
model_bytes = open("housing_db_glm_model.pmml", "rb").read()
con.execute("insert into pmml_models  (model_id, model) values(?,?)",
'housing_db_glm_model', model_bytes)
model_list = pd.read_sql("select * from
      pmml_models", con)
model_list
```



|   | model_id | model |
|---|---|---|
| 0 | housing_db_glm_model | b'<?xml version="1.0" encoding="UTF-8" standal... |

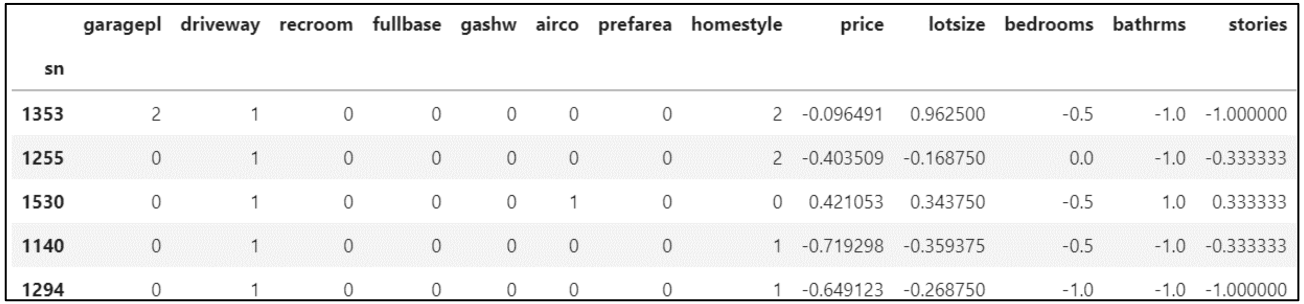
# Lab 2c: Scale the Test Data

- Price and lot size are very large compared to other X variables
- Let's scale all but the yes/no variables and homestyle
- scaleFit and scaleTransform will do this
- scaleFit calculates the statistics and scaleTransform scales the rows

```
con.execute("SELECT * FROM TD_ScaleFit (
ON (SELECT * FROM TRNG_TDU_TD01.housing_test_int WHERE
sell_date EQ '2019-01-01') AS InputTable
OUT PERMANENT TABLE OutputTable (scaleFitOutTest)
USING
TargetColumns
('price','lotsize','bedrooms','bathrms','stories')
MissValue ('keep')
ScaleMethod ('midrange')
GlobalScale ('f')
) AS dt;")
```

# Lab 2c: Scale the Test Data

```
con.execute("CREATE TABLE housing_test_scale AS (
            SELECT * FROM TD_scaleTransform (
ON (SELECT * FROM TRNG_TDU_TD01.housing_test_int WHERE sell_date EQ '2019-01-01') AS
InputTable
ON scaleFitOutTest AS FitTable DIMENSION
USING
Accumulate
('sn','garagepl','driveway','recroom','fullbase','gashw','airco','prefarea','homestyle')
) AS dt
) WITH DATA;")traid_pd2 = train_df2.to_pandas()
test_df2 = DataFrame.from_query("SELECT * FROM housing_test_scale;")
testd_pd2 = test_df2.to_pandas()
testd_pd2
```



| | garagepl | driveway | recroom | fullbase | gashw | airco | prefarea | homestyle | price | lotsize | bedrooms | bathrms | stories |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **sn** | | | | | | | | | | | | | |
| **1353** | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | -0.096491 | 0.962500 | -0.5 | -1.0 | -1.000000 |
| **1255** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | -0.403509 | -0.168750 | 0.0 | -1.0 | -0.333333 |
| **1530** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0.421053 | 0.343750 | -0.5 | 1.0 | 0.333333 |
| **1140** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | -0.719298 | -0.359375 | -0.5 | -1.0 | -0.333333 |
| **1294** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | -0.649123 | -0.268750 | -1.0 | -1.0 | -1.000000 |

## Lab 2c: GLM – Score 'Regression' Model

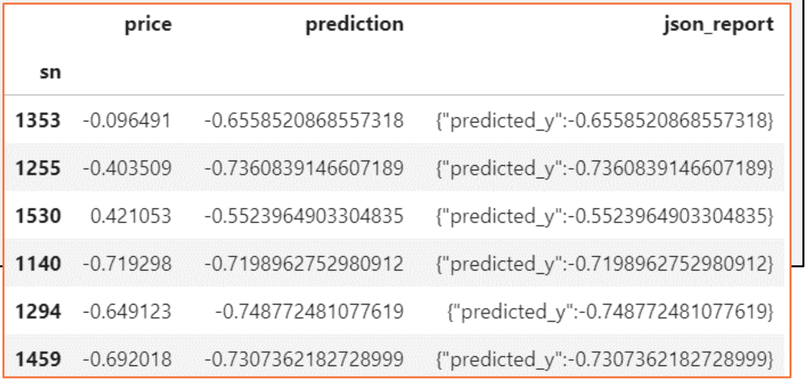
WIP

### Clear scoring table

```
con.execute("DROP TABLE housing_glm_out;")
```

### Call scoring function—PMMLPredict

```
con.execute("CREATE TABLE housing_glm_out AS (
SELECT * FROM TRNG_BYOM.PMMLPredict(
    ON TRNG_TDU_TD01.housing_test_scale
    ON (select * from pmml_models where model_id='housing_db_glm_model')
DIMENSION
    USING
        Accumulate('sn','price')
) AS dt
) WITH DATA;")
```



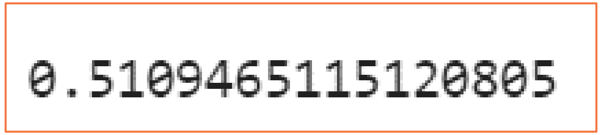
| sn | price | prediction | json_report |
|---|---|---|---|
| 1353 | -0.096491 | -0.6558520868557318 | {"predicted_y":-0.6558520868557318} |
| 1255 | -0.403509 | -0.7360839146607189 | {"predicted_y":-0.7360839146607189} |
| 1530 | 0.421053 | -0.5523964903304835 | {"predicted_y":-0.5523964903304835} |
| 1140 | -0.719298 | -0.7198962752980912 | {"predicted_y":-0.7198962752980912} |
| 1294 | -0.649123 | -0.748772481077619 | {"predicted_y":-0.748772481077619} |
| 1459 | -0.692018 | -0.7307362182728999 | {"predicted_y":-0.7307362182728999} |

WIP

## Lab 2e: GLM – Check Accuracy

### Check Accuracy using mean squared error

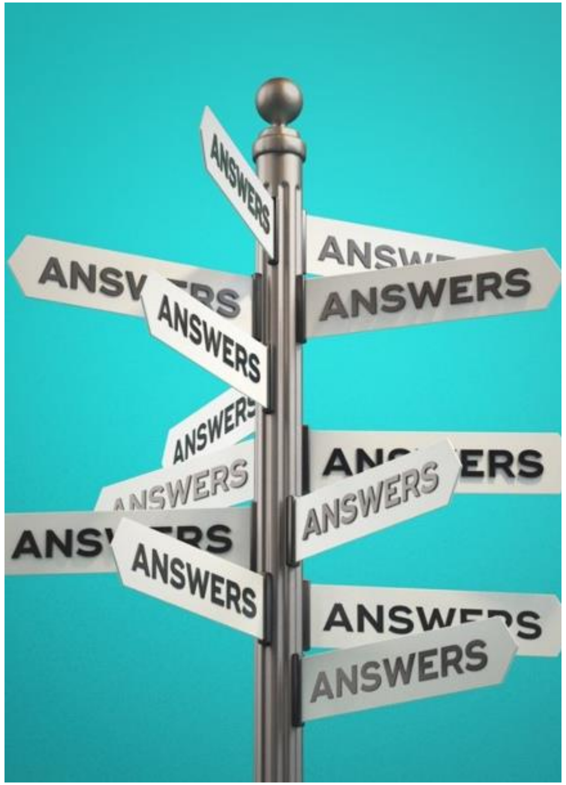
```
from sklearn.metrics import mean_squared_error
housing_sq = mean_squared_error(housing_pd.price,
housing_pd.prediction)
housing_sq
```



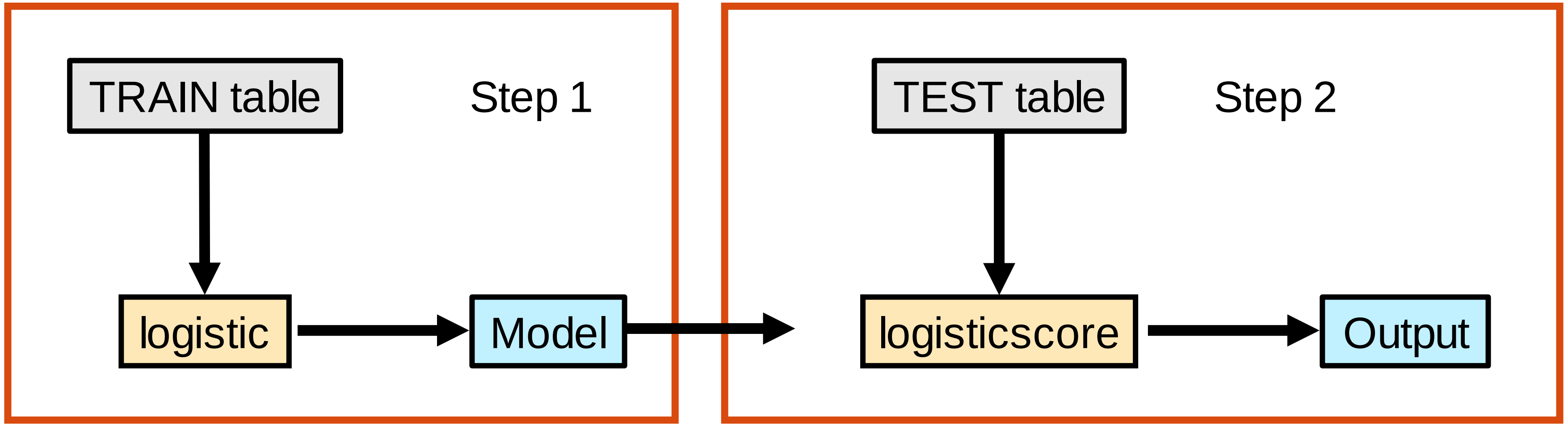
```
0.5109465115120805
```

## Current Topic – Logistic Regression

- CRISP-DM
- GLM
- **Logistic Regression**
- Review & Summary



## Logistic Regression (VAL) – Workflow



## Logistic Regression – Overview

Logistic Regression is one of the most widely used types of statistical analysis. In Logistic Regression, a set of independent variables (in this case columns) is processed to predict the value of a dependent variable (column) that assumes two values referred to as response (1) and non-response (0). Actually, the user specifies what value of the dependent variable to treat as the response, and all other values assumed by the dependent variable are treated as non-response. The result is not however a continuous numeric variable as seen in Linear Regression, but rather a probability between 0 and 1 that the response value is assumed by the dependent variable. There are many types of analysis that lend themselves to the use of Logistic Regression, and when scoring a model, benefit from the estimation of a probability rather than a fixed value. For example, when predicting who should be targeted for a marketing campaign, the scored customers can be ordered by the predicted probability from most to least likely, and the top n values taken from the customer list.

## Logistic Regression Score – Syntax

The type of logistic regression model that Analytics Library supports is one with a two-valued dependent variable, referred to as a binary logit model. However, Analytics Library is capable of coding values for the dependent variable so that the user is not required to code their dependent variable to two distinct values. The user can choose which values to represent as the response value (i.e., 1 or TRUE) and all other will be treated as non-response values (i.e., 0 or FALSE). Even though values other than 1 and 0 are supported in the dependent variable, throughout this section, the dependent variable response value is represented as 1 and the non-response value as 0 for ease of reading. The primary sources of information and formulae in this section are [Hosmer] and [Neter].

## Logistic Regression – Syntax

```
call ${XSPDB}.td_analyze(logistic',
                          'database=database;
                           tablename=table;
                           columns=column1,column2,…;
                           dependent=response_column;
                           outputdatabase=database;
                           outputtablename=LogisticOut;);
```

# Logistic Regression – Required Arguments

- **Database:** Specify the database that has the input file
- **TableName:** Specify the table to build the model from
- **Dependent:** Specify the name of the column that contains the response variable (that is, what you want to predict). (**Y**-var)
- **Columns:** (**X**-var). Specify the names of columns that contain numeric predictor variables (which must be numeric values) and specify the names of the columns that contain the categorical predictor variables (which can be either numeric or VARCHAR values).

# Logistic Regression – Optional Arguments

- **backward:** Whether to start with all independent variables in the model and do the following until no more independent variables can be removed from the model:
  - Take one backward step, removing the independent variable that worst explains the variance of the dependent variable
  - Take one forward step, adding the independent variable that best explains the variance of the dependent variable
- **backwardonly:** Like backward without the forward step
- **constant:** Whether the linear model includes a constant term. Default: True
- **convergence:** The convergence criterion. The algorithm stops iterating when the change in the log likelihood function falls below this value. Default: .001

# Logistic Regression – Optional Arguments (cont.)

- **forward:** Whether to start with no independent variables in the model and do the following until no more independent variables can be added to the model:
  - Take one forward step, adding the independent variable that best explains the variance of the dependent variable
  - Take one backward step, removing the independent variable that wors explains the variance of the dependent variable
- **forwardonly:** Like forward without the backward step
- **groupby:** The input table columns for which to separately analyze each value or combination of values. Default behavior: Input is not grouped.
- **Overwrite:** When overwrite is set to true (default), the output tables are dropped before creating new ones.
- **maxiterations:** The maximum number of attempts to converge on a solution. Default: 100

## Logistic Regression – Optional Arguments (cont.)

- **response:** The value assumed by the dependent column, to treat as the response value
- **stepwise:** Whether to perform the stepwise procedure (**forward**, **forwardonly**, **backward**, or **backwardonly**). Default: false
- **ColumnsToExclude:** If a column specifier such as all is used in the columns' parameter, the columnstoexclude parameter may be used to exclude specific columns from the analysis. For convenience, when the columnstoexclude parameter is used, dependent variable and group by columns, if any, are automatically excluded as input columns and do not need to be included as columnstoexclude.

# Lab 3a: View the Data

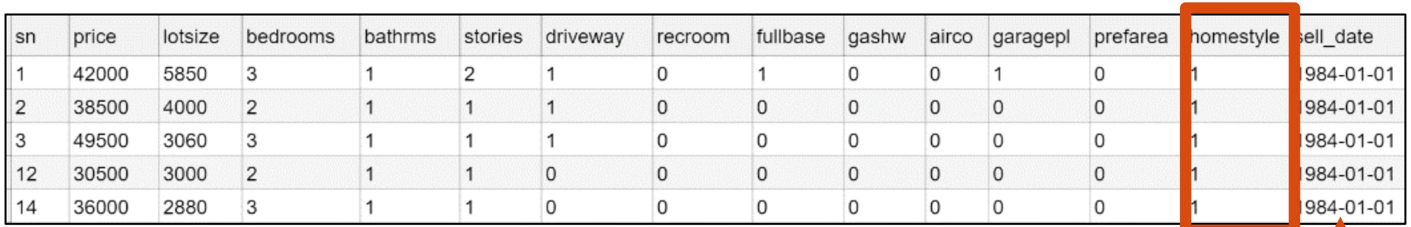
- Here's the data we'll be using. We will be predicting Y-variable 'homestyle'. This is a classification model
- There are 12 X-variables, all numeric
- price, lotsize, bedrooms, bathrms, stories, garagepl
- 0=no, 1=yes: driveway, recroom, fullbase, gashw, airco, prefarea

```
SELECT *
FROM
housing_train_int_binary;
```

Home Style
0 = bungalow
1 = classic

x x x x x x x x x x x x

Y-var



| sn | price | lotsize | bedrooms | bathrms | stories | driveway | recroom | fullbase | gashw | airco | garagepl | prefarea | homestyle | sell_date |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 42000 | 5850 | 3 | 1 | 2 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1984-01-01 |
| 2 | 38500 | 4000 | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1984-01-01 |
| 3 | 49500 | 3060 | 3 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1984-01-01 |
| 12 | 30500 | 3000 | 2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1984-01-01 |
| 14 | 36000 | 2880 | 3 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1984-01-01 |

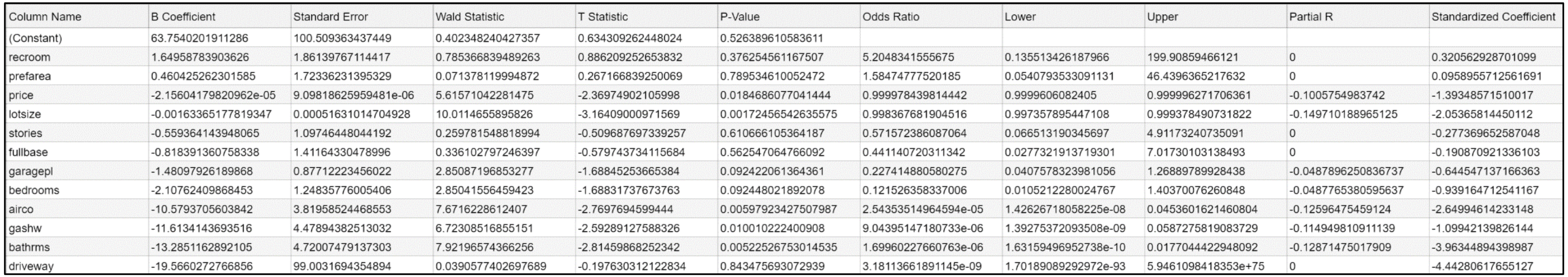
Not used

## Lab 3b: Logistic Regression – Create 'Classification' Model

```
call TRNG_XSP.td_analyze('logistic',
                                      'database=TRNG_TDU_TD01;
                                       tablename=housing_train_int_binary;
columns=price,lotsize,bedrooms,bathrms,stories,garagepl,driveway,
                                        recroom,fullbase,gashw,airco,prefarea;
                                       dependent=homestyle;
                                       statstable=true;
                                       successtable=true;
                                       thresholdtable=true;
                                       lifttable=true;
                                       outputdatabase=${QLID};
                                       outputtablename=LogisticOut1;
                                        overwrite=true;');
```

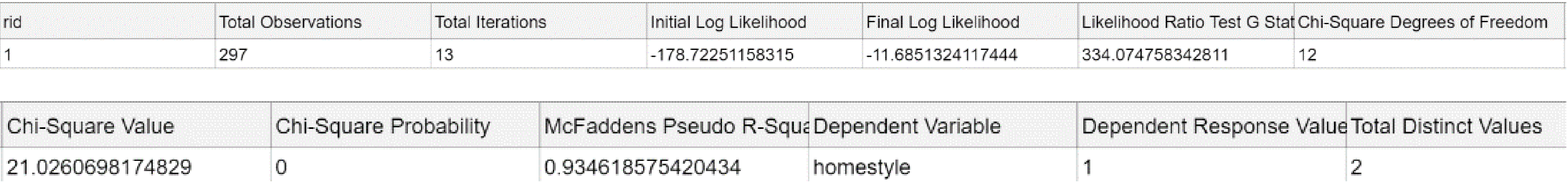
# Lab 3b: Logistic Regression – Create 'Classification' Model

```
SELECT * FROM LogisticOut1;
```



| Column Name | B Coefficient | Standard Error | Wald Statistic | T Statistic | P-Value | Odds Ratio | Lower | Upper | Partial R | Standardized Coefficient |
|---|---|---|---|---|---|---|---|---|---|---|
| (Constant) | 63.7540201911286 | 100.509363437449 | 0.402348240427357 | 0.634309262448024 | 0.526389610583611 | | | | | |
| recroom | 1.64958783903626 | 1.86139767114417 | 0.785366839489263 | 0.886209252653832 | 0.376254561167507 | 5.2048341555675 | 0.135513426187966 | 199.90859466121 | 0 | 0.320562928701099 |
| prefarea | 0.460425262301585 | 1.72336231395329 | 0.071378119994872 | 0.267166839250069 | 0.789534610052472 | 1.58474777520185 | 0.0540793533091131 | 46.4396365217632 | 0 | 0.0958955712561691 |
| price | -2.15604179820962e-05 | 9.09818625959481e-06 | 5.61571042281475 | -2.36974902105998 | 0.0184686077041444 | 0.999978439814442 | 0.9999606082405 | 0.999996271706361 | -0.1005754983742 | -1.39348571510017 |
| lotsize | -0.00163365177819347 | 0.00051631014704928 | 10.0114655895826 | -3.16409000971569 | 0.00172456542635575 | 0.998367681904516 | 0.997357895447108 | 0.999378490731822 | -0.149710188965125 | -2.05365814450112 |
| stories | -0.559364143948065 | 1.09746448044192 | 0.259781548818994 | -0.509687697339257 | 0.610666105364187 | 0.571572386087064 | 0.066513190345697 | 4.91173240735091 | 0 | -0.277369652587048 |
| fullbase | -0.818391360758338 | 1.41164330478996 | 0.336102797246397 | -0.579743734115684 | 0.562547064766092 | 0.441140720311342 | 0.0277321913719301 | 7.01730103138493 | 0 | -0.190870921336103 |
| garagepl | -1.48097926189868 | 0.87712223456022 | 2.85087196853277 | -1.68845253665384 | 0.092422061364361 | 0.227414880580275 | 0.0407578323981056 | 1.26889789928438 | -0.0487896250836737 | -0.644547137166363 |
| bedrooms | -2.10762409868453 | 1.24835776005406 | 2.85041556459423 | -1.68831737673763 | 0.092448021892078 | 0.121526358337006 | 0.0105212280024767 | 1.40370076260848 | -0.0487765380595637 | -0.939164712541167 |
| airco | -10.5793705603842 | 3.81958524468553 | 7.6716228612407 | -2.7697694599444 | 0.00597923427507987 | 2.54353514964594e-05 | 1.42626718058225e-08 | 0.0453601621460804 | -0.12596475459124 | -2.64994614233148 |
| gashw | -11.6134143693516 | 4.47894382513032 | 6.72308516855151 | -2.59289127588326 | 0.010010222400908 | 9.04395147180733e-06 | 1.39275372093508e-09 | 0.0587275819083729 | -0.114949810911139 | -1.09942139826144 |
| bathrms | -13.2851162892105 | 4.72007479137303 | 7.92196574366256 | -2.81459868252342 | 0.00522526753014535 | 1.69960227660763e-06 | 1.63159496952738e-10 | 0.0177044422948092 | -0.12871475017909 | -3.96344894398987 |
| driveway | -19.5660272766856 | 99.0031694354894 | 0.0390577402697689 | -0.197630312122834 | 0.843475693072939 | 3.18113661891145e-09 | 1.70189089292972e-93 | 5.9461098418353e+75 | 0 | -4.44280617655127 |

```
SELECT * FROM LogisticOut1_rpt;
```

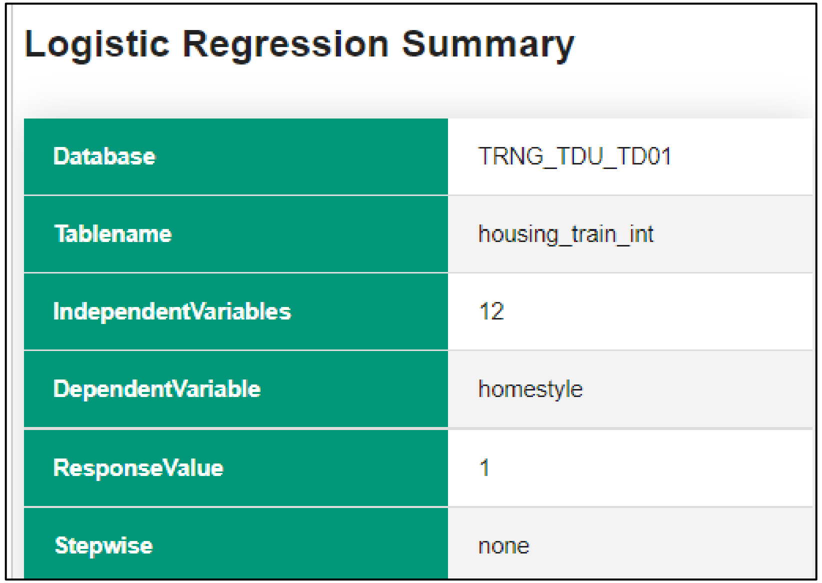


| rid | Total Observations | Total Iterations | Initial Log Likelihood | Final Log Likelihood | Likelihood Ratio Test G Stat | Chi-Square Degrees of Freedom |
|---|---|---|---|---|---|---|
| 1 | 297 | 13 | -178.72251158315 | -11.6851324117444 | 334.074758342811 | 12 |

| Chi-Square Value | Chi-Square Probability | McFaddens Pseudo R-Squa | Dependent Variable | Dependent Response Value | Total Distinct Values |
|---|---|---|---|---|---|
| 21.0260698174829 | 0 | 0.934618575420434 | homestyle | 1 | 2 |

# Logistic Regression Report Summary

```
call TRNG_XSP.td_analyze ('report',
                          'database=${QLID};
                           tablename=LogisticOut1;
                           analysistype=logistic');
```



Logistic Regression Summary

| | |
|---|---|
| Database | TRNG_TDU_TD01 |
| Tablename | housing_train_int |
| IndependentVariables | 12 |
| DependentVariable | homestyle |
| ResponseValue | 1 |
| Stepwise | none |

## Logistic Regression Score – Syntax

```
call TRNG_XSP.td_analyze('logisticscore',
                         'database=TRNG_TDU_TD01;
                          tablename=housing_test_int_binary;
                          modeldatabase=${QLID};
                          modeltablename=LogisticOut1;
                          outputdatabase=${QLID};
                          outputtablename=LogisticScore1;
                          estimate=Estimate;
                          probability=Probability;
                          retain=homestyle;
                          samplescoresize=25;
                          lifttable=true;
                          successtable=true;
                          scoringmethod=scoreandevaluate');
```

## Logistic Regression Score – Arguments

- **Database:** Specify the database that has the input file.
- **ModelDatabase:** The database containing the table representing the decision tree model input to the analysis.
- **ModelTableName:** The table containing the decision tree model in PMML format that is used to score the data. It must reside in the database indicated by the modeldatabase parameter.
- **OutputDatabase:** The database containing the output table.
- **OutputTableName:** The output table containing the predicted values of the dependent variable. It must reside in the database indicated by the outputdatabase parameter.
- **TableName:** The table containing the columns to analyze, representing the dependent and independent variables in the analysis. It must reside in the database indicated by the database parameter.

## Logistic Regression Score – Optional Arguments

- **Confusionmatrix:** A table delivered in the function’s XML output string, displaying counts of predicted versus actual values of the dependent variable of the decision tree model.
- **Gensqlonly:** When true, the SQL for the requested function is returned as a result set but not run. When not specified or set to false, the SQL is run but not returned.
- **Includeconfidence:** If selected, the output table will contain a column indicating how likely it is, for a particular leaf node on the tree, that the prediction is correct.
- **Index:** By default, the primary index columns of the score output table are the primary index columns of the input table. This parameter allows the user to specify one or more columns for the primary index of the score output table.

# Logistic Regression Score – Optional Arguments (cont.)

- **Overwrite:** When overwrite is set to true or not set, the output table is dropped before creating a new one.
- **Predicted:** If the ‘scoringmethod’ parameter is set to ‘score’ or ‘scoreandevaluate’, the name of the predicted value column is entered here. If not entered here, the name of the dependent column in the input table is used.
- **Retain:** One or more columns from the input table can optionally be specified here to be passed along to the score output table.
- **Samplescoresize:** When a scoring function produces a score table, the user has the option to view a sample of the rows using the "samplescoresize=n" parameter, where n is an integer number of rows to view in a result set. Cases where a sample is not returned include when you only generating SQL and when you are only evaluating (i.e., not scoring). By default, a sample of output score rows is not returned.

# Logistic Regression Score – Optional Arguments (cont.)

- **Scoringmethod:** Three scoring methods are available as outlined below. By default, the model is scored but not evaluated, as requested in this manner: scoringmethod=score.
  - score
  - evaluate
  - scoreandevaluate
- **Targetedvalue:** If selected, the output table will contain a column indicating how likely it is, for a particular leaf node and targeted value of a predicted result with only two values, that the prediction is correct.

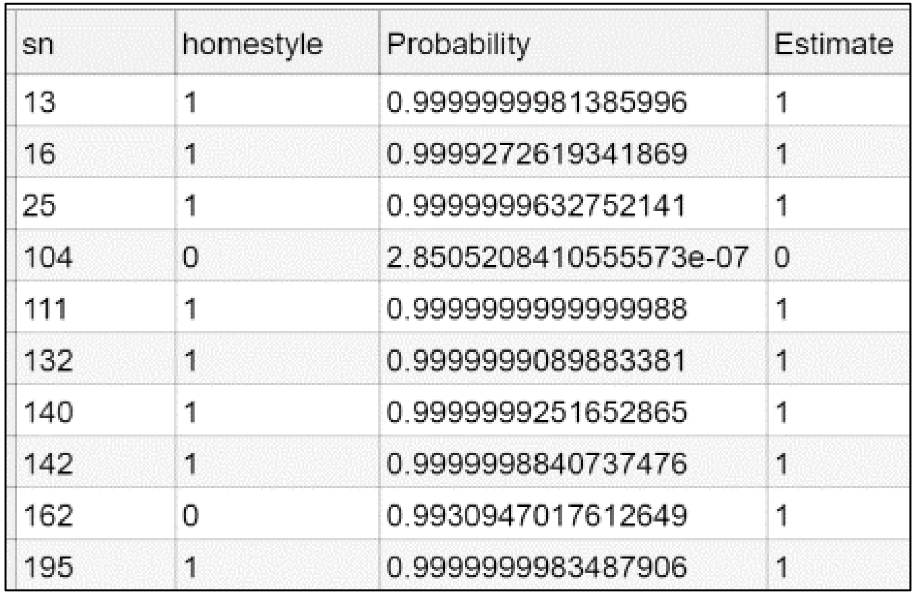
# Lab 4a: Using VAL Logistic Regression Score

```
call TRNG_XSP.td_analyze('logisticscore',
                         'database=${QLID};
                          tablename=housing_test_int_binary;
                          modeldatabase=${QLID};
                          modeltablename=LogisticOut1;
                          outputdatabase=${QLID};
                          outputtablename=LogisticScore1;
                          estimate=Estimate;
                          probability=Probability;
                          retain=homestyle;
                          samplescoresize=25;
                          lifttable=true;
                          successtable=true;
                          scoringmethod=scoreandevaluate');
```

## Lab 4b: View Prediction Results and Accuracy

```
SELECT * FROM LogisticScore1;
```

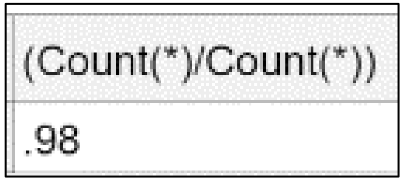
Actual Predict



| sn | homestyle | Probability | Estimate |
|---|---|---|---|
| 13 | 1 | 0.99999999981385996 | 1 |
| 16 | 1 | 0.9999272619341869 | 1 |
| 25 | 1 | 0.9999999632752141 | 1 |
| 104 | 0 | 2.850520841055573e-07 | 0 |
| 111 | 1 | 0.9999999999999988 | 1 |
| 132 | 1 | 0.9999999089883381 | 1 |
| 140 | 1 | 0.9999999251652865 | 1 |
| 142 | 1 | 0.9999998840737476 | 1 |
| 162 | 0 | 0.9930947017612649 | 1 |
| 195 | 1 | 0.9999999983487906 | 1 |

98% accurate

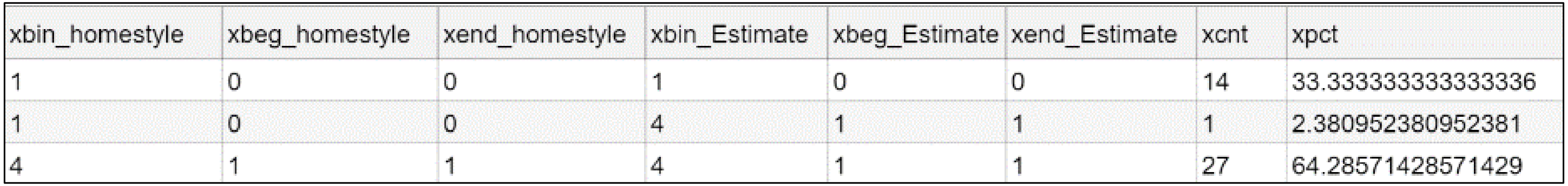
```
SELECT
(SELECT cast(count(*) as dec(4,2))
 FROM LogisticScore1
WHERE homestyle = Estimate)/
(SELECT cast(count(*) as dec(4,2))
 FROM LogisticScore1);
```



| (Count(*)/Count(*)) |
|---|
| .98 |

# Lab 4b: View Prediction Results and Accuracy (cont.)

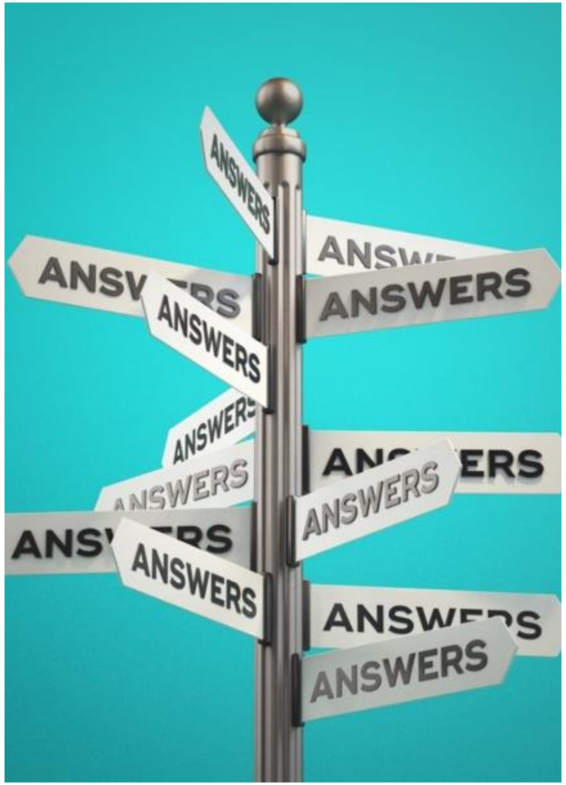
```
call TRNG_XSP.td_analyze('histogram',
                         'database=${QLID};
                          tablename=LogisticScore1;
                          columns=homestyle,Estimate;
                           quantiles=10;
                           style=crosstab');
```



| xbin_homestyle | xbeg_homestyle | xend_homestyle | xbin_Estimate | xbeg_Estimate | xend_Estimate | xcnt | xpct |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 14 | 33.333333333333336 |
| 1 | 0 | 0 | 4 | 1 | 1 | 1 | 2.380952380952381 |
| 4 | 1 | 1 | 4 | 1 | 1 | 27 | 64.28571428571429 |

## Current Topic – Review & Summary

- CRISP-DM
- GLM
- Logistic Regression
- **Review & Summary**



## Summary

In this module, you learned to:

- Write queries using these Teradata Vantage predictive and other analytic functions:
  - CRISP-DM methodology
  - GLM with Bring Your Own Model
  - Logistic Regression with Vantage Analytics Library

# Thank you.

teradata.