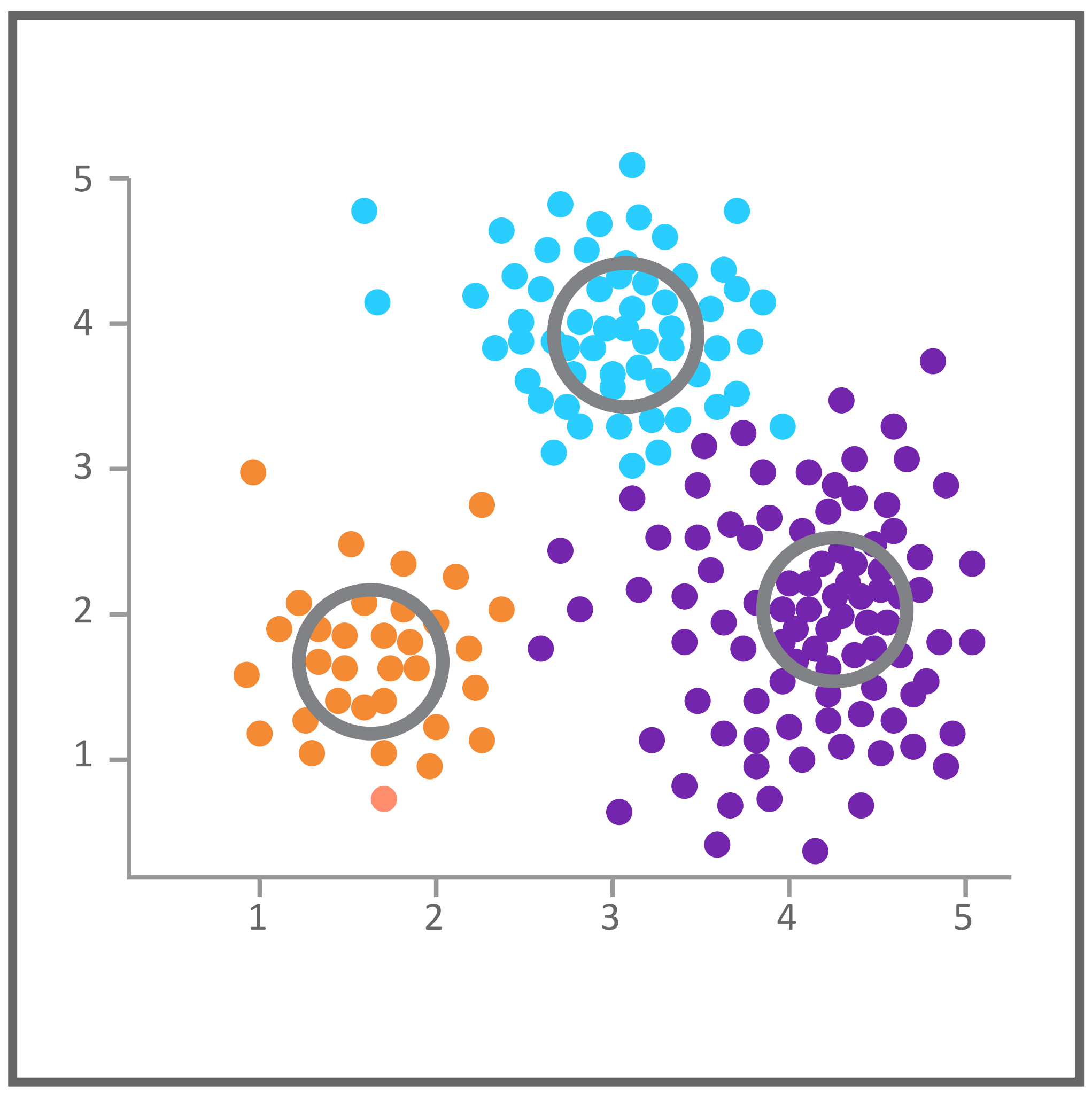
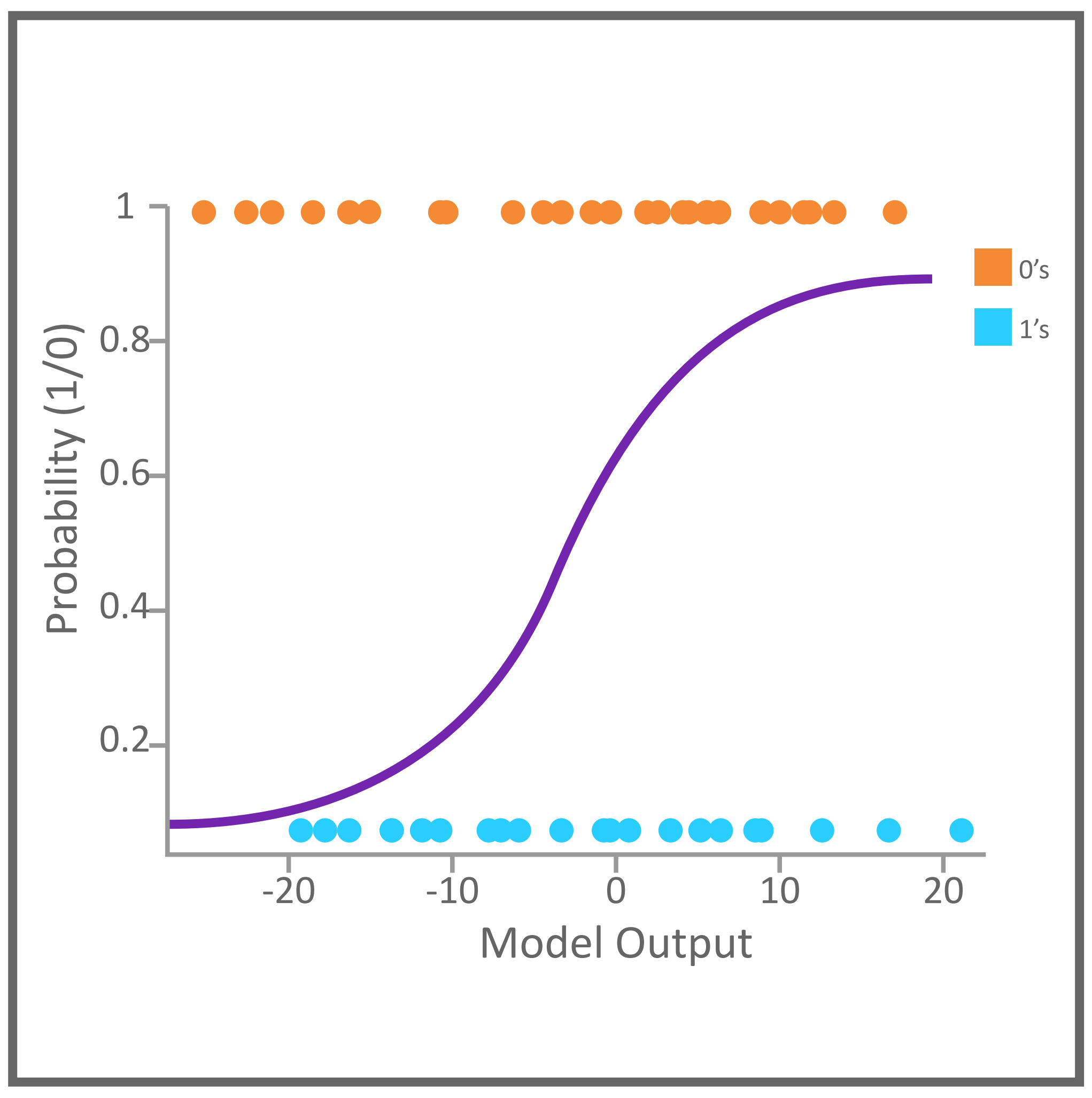
Vertica Machine Learning Overview

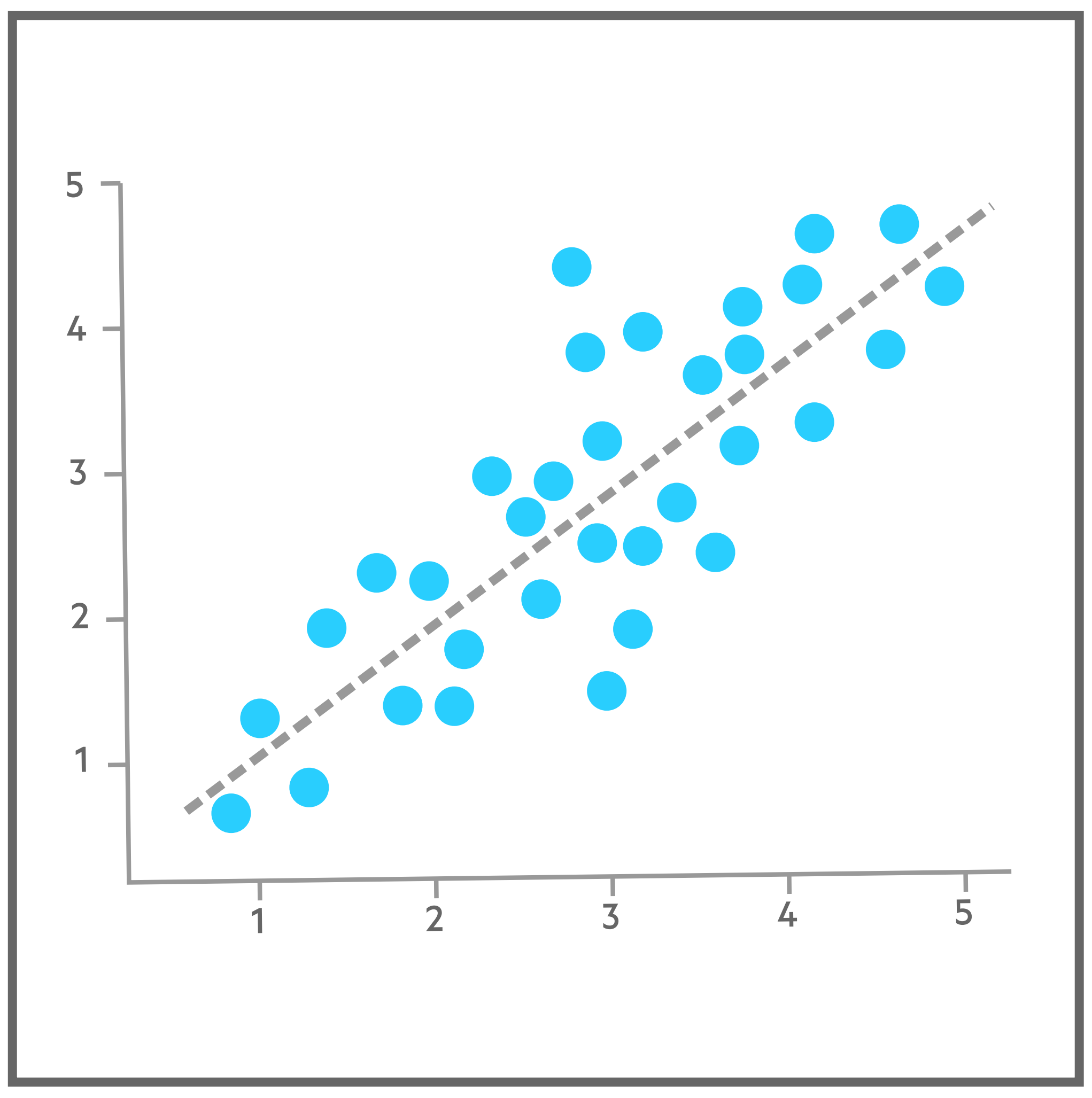
K-Means Clustering



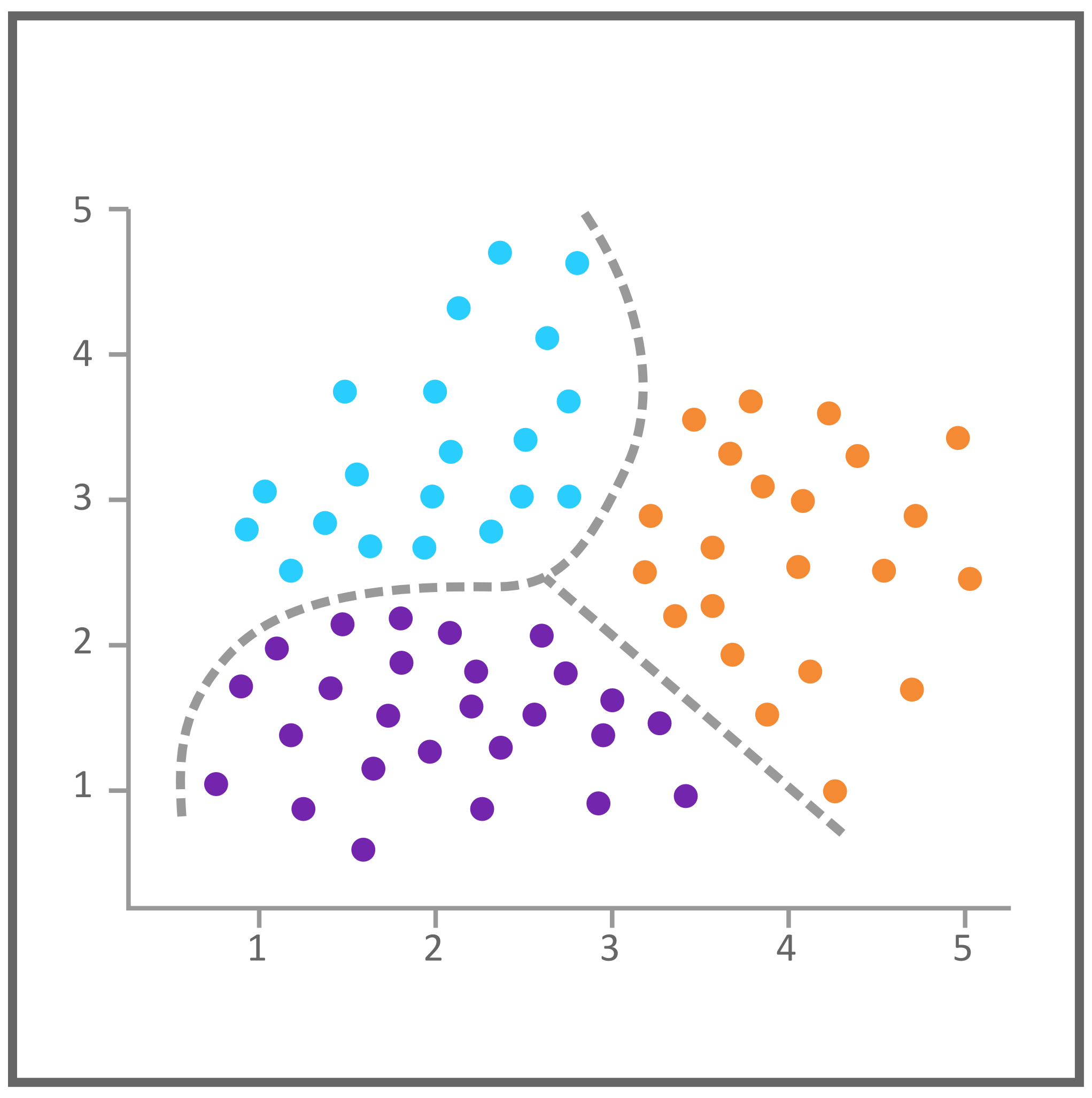
Logistic Regression



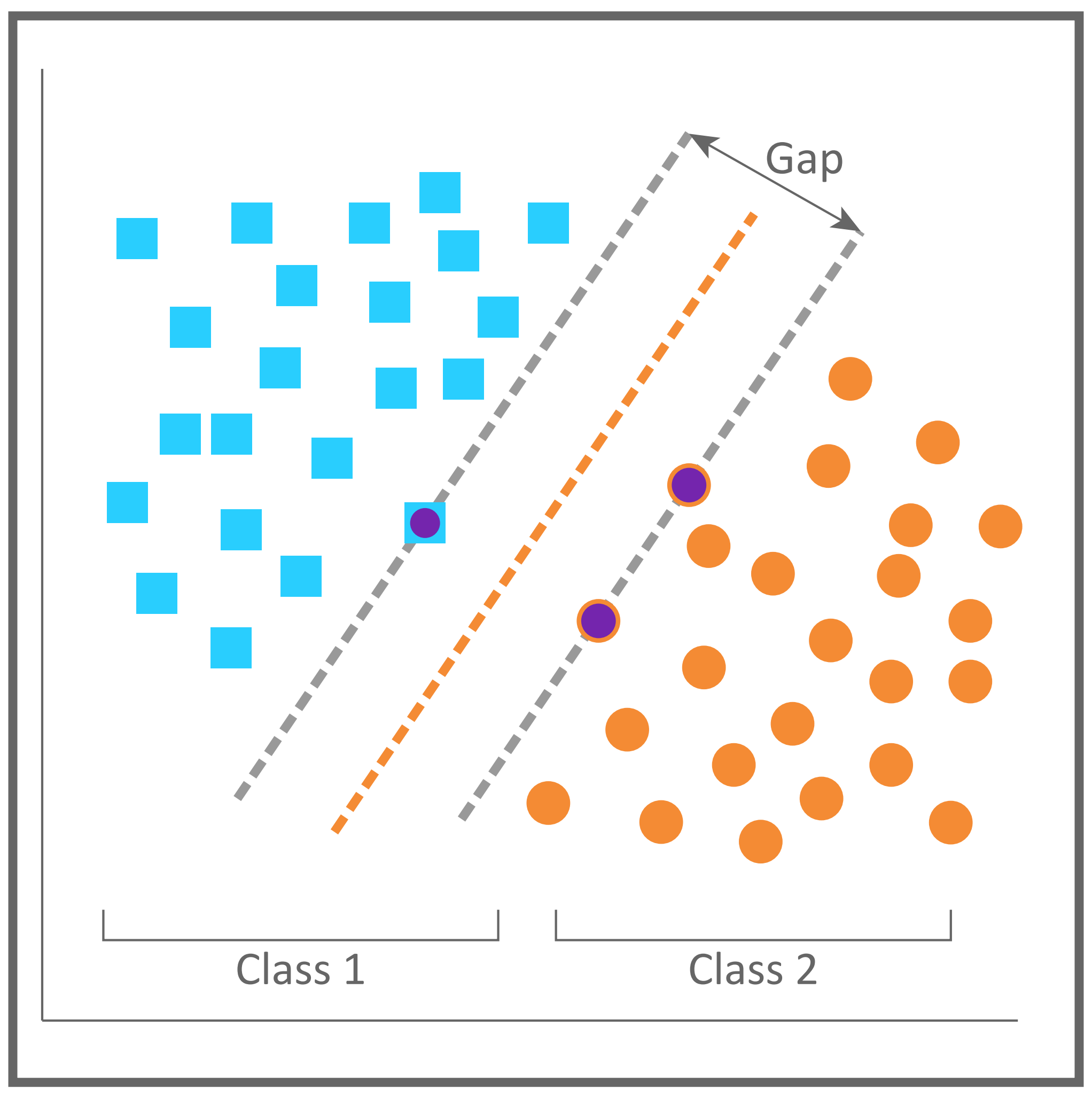
Linear Regression



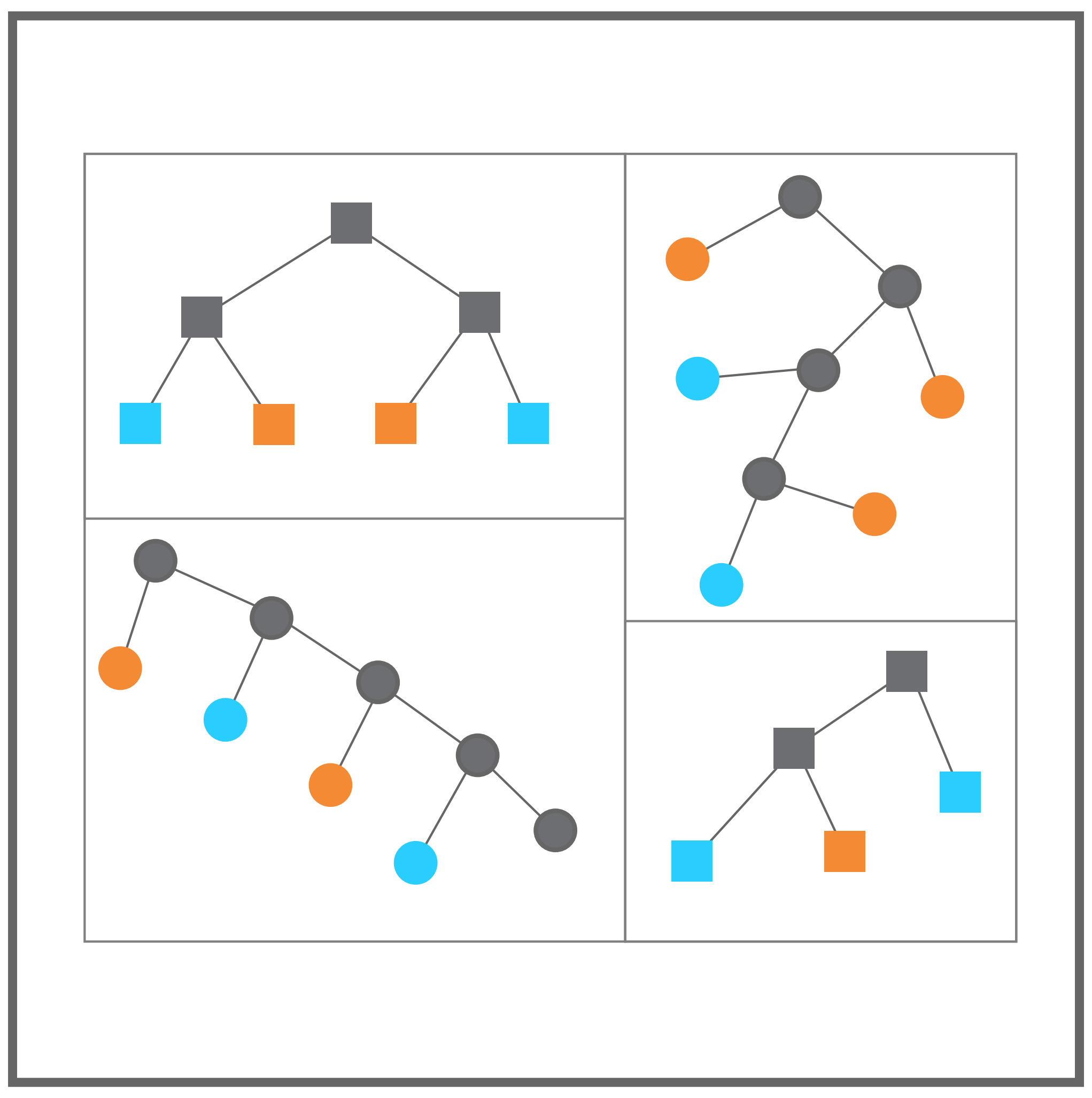
Naive Bayes



Support Vector Machines



Random Forests



Managing models
List
```
SELECT * FROM models;
```

Delete
```
DROP MODEL myLinearRegModel;
```

Rename, change owner and change schema
```
ALTER MODEL myKmeansModel OWNER TO user1;

ALTER MODEL myKmeansModel SET SCHEMA public;

ALTER MODEL myKmeansModel RENAME to myKmeans;
```

Summarize model
```
SELECT summarize_model('myLinearRegModel');
```

Read model attributes
```
SELECT get_model_attribute(USING PARAMETERS
model_name='myLinearRegModel'); --list all attributes in
the model

SELECT get_model_attribute(USING PARAMETERS
model_name='myLinearRegModel', attr_name='data');
--return the value for attribute 'data'
```

Unsupervised Learning Functions
You can use the following unsupervised learning functions to run analytics on a data set:

KMEANS Use this function to cluster data points into k different groups.

Clustering K-means
```
SELECT kmeans('myKmeansModel', 'iris', '*', 5  USING
PARAMETERS max_iterations=20, key_columns='id',
exclude_columns='species, id');

SELECT id, apply_kmeans(sepal_length, 2.2, 1.3,
petal_width USING PARAMETERS model_name='myKmeansModel',
match_by_pos='true') FROM iris;
```

Supervised Learning Functions
You can use the following supervised learning functions to run predictive analytics on a data set:

LINEAR_REG Use this function to model the linear relationship between independent variables and some dependent variable.
LOGISTIC_REG Use this function to model the relationship between independent variables and some dependent variable.
NAIVE_BAYES Use this function to classify your data when features can be assumed independent.
RF_CLASSIFIER Use this function to create an ensemble model of decision trees.
SVM_CLASSIFIER Use this function to assign data to one category or the other.
SVM_REGRESSOR Use this function to predict continuous ordered variables.

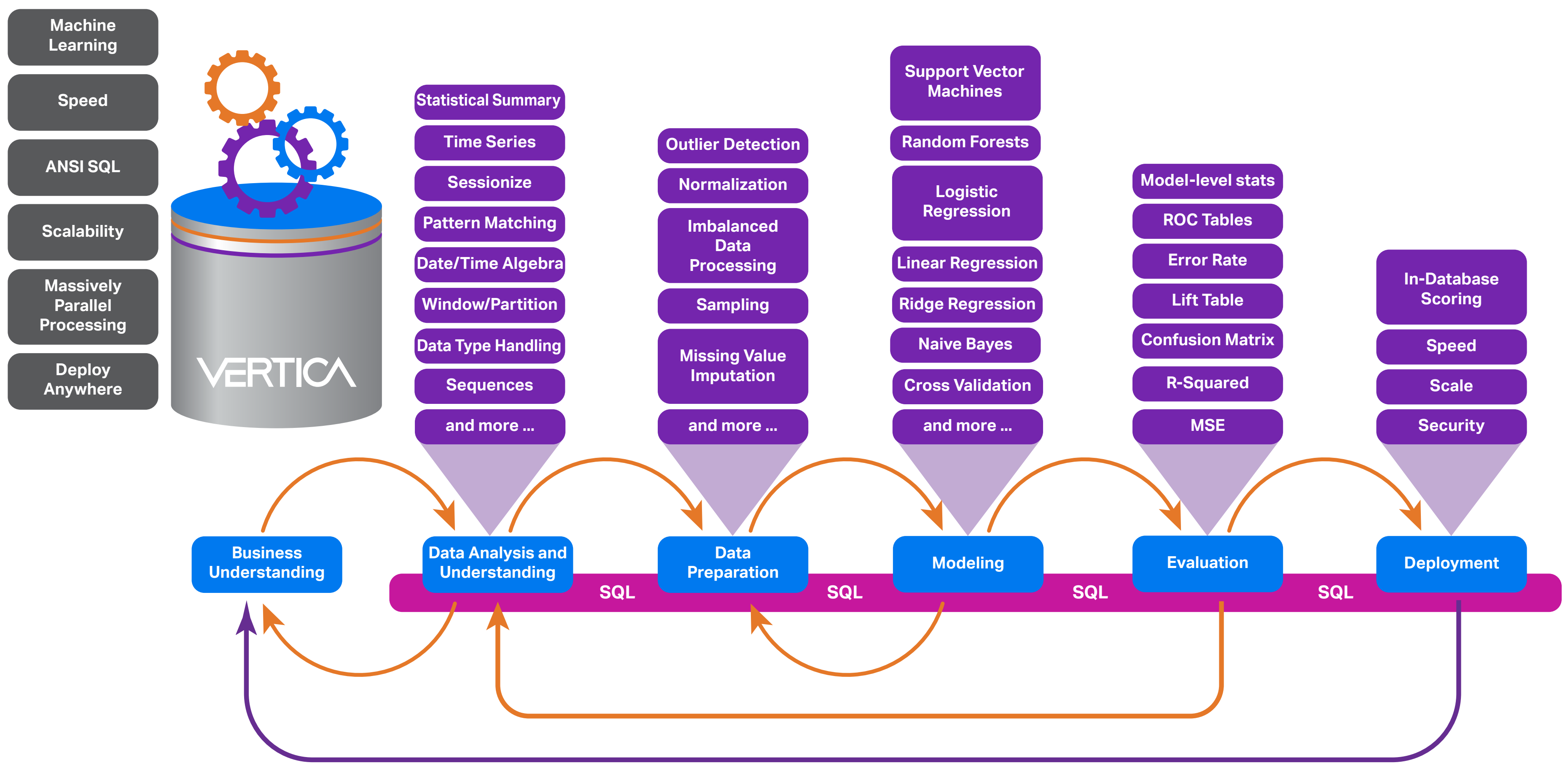
Training and predicting Regression
Linear Regression
```
SELECT linear_reg('myLinearRegModel', 'faithful_train',
'eruptions', 'waiting' USING PARAMETERS
optimizer='BFGS', regularization='L2');

SELECT id, predict_linear_reg(waiting USING PARAMETERS
model_name='myLinearRegModel') FROM faithful_test;
```

Support Vector Machines (SVM)
```
SELECT svm_regressor('mySvmRegModel', 'faithful_train',
'eruptions', 'waiting' USING PARAMETERS
error_tolerance=0.1, max_iterations=100);

SELECT id, predict_svm_regressor(waiting USING
PARAMETERS model_name='mySvmRegModel') FROM
faithful_test;
```

Classification
Logistic Regression
```
SELECT logistic_reg('myLogisticRegModel',
'mtcars_train', 'am','mpg, cyl, disp, hp, drat, wt,
qsec, vs, gear, carb' USING PARAMETERS
exclude_columns='hp', optimizer='BFGS',
regularization='L2');

SELECT car_model, predict_logistic_reg(mpg, cyl, disp,
drat, wt, qsec, vs, gear, carb USING PARAMETERS
model_name='myLogisticRegModel') FROM mtcars_test;
```

Vertica Machine Learning supports the whole workflow of machine learning via a SQL interface. To learn the full capability of Vertica ML, go to my.vertica.com/documentation. Example data sets used in the cheat sheet are available on github.com/vertica/Machine-Learning-Examples.

A basic example
```
CREATE TABLE iris (id int, sepal_length float, sepal_width float, petal_length float, petal_width float, species varchar(10));

COPY iris FROM LOCAL 'iris.csv' DELIMITER ',' ENCLOSED BY '"' SKIP 1;

CREATE TABLE iris_test AS SELECT * FROM iris TABLESAMPLE(25);

CREATE TABLE iris_train AS (SELECT * FROM iris EXCEPT SELECT * FROM iris_test);

SELECT rf_classifier('myRFModel', 'iris_train', 'species', 'sepal_length, sepal_width, petal_length, petal_width' USING
PARAMETERS ntree=100, sampling_size=0.3);

CREATE TABLE iris_prediction AS SELECT species, predict_rf_classifier(sepal_length, sepal_width, petal_length, petal_width USING
PARAMETERS model_name='myRFModel') AS predicted FROM iris_test;

SELECT confusion_matrix(CASE species WHEN 'setosa' THEN 2 WHEN 'versicolor' THEN 1 ELSE 0 END, CASE predicted WHEN 'setosa' THEN
2 WHEN 'versicolor' THEN 1 ELSE 0 END USING PARAMETERS num_classes=3) OVER() FROM iris_prediction;
```



Support Vector Machines (SVM)
```
SELECT svm_classifier('mySvmClassModel', 'mtcars_train', 'am',
'mpg,cyl,disp,hp,drat,wt,qsec,vs,gear,carb' USING PARAMETERS
exclude_columns='hp,drat');

SELECT car_model,
predict_svm_classifier(mpg,cyl,disp,wt,qsec,vs,gear,carb USING
PARAMETERS model_name='mySvmClassModel') FROM mtcars_test;
```

Data Preparation Functions
You can use the following functions to pre-process your data:

APPLY_NORMALIZE Use this function to apply normalization parameters saved in a model to specific columns.

BALANCE Use this function to balance your data.
DETECT_OUTLIERS Use this function to remove the outliers from your data.
IMPUTE Imputes missing values in a data set.
NORMALIZE Use this function before running one of the machine learning algorithms on your data.
NORMALIZE_FIT Use this function to compute normalization parameters for specific columns in an input table. The normalization parameters are saved.

REVERSE_NORMALIZE Use this function to reverse the normalization transformation.

Preprocessing the data
Detect outliers
```
SELECT detect_outliers('baseball_outliers', 'baseball_roster',
'*', 'robust_zscore' USING PARAMETERS outlier_threshold=3.0,
exclude_columns='id, last_name');
```

Normalize
```
SELECT normalize('mtcars_normz', 'mtcars', 'wt, hp', 'zscore');
--output a view 'mtcars_normz'

SELECT normalize_fit('mtcars_normfitrz', 'mtcars', 'wt,hp',
'robust_zscore'); --store normalization parameters in a model
'mtcars_normfitrz'

SELECT apply_normalize(wt, hp USING PARAMETERS model_name =
'mtcars_normfitrz') FROM mtcars; --apply the normalization
parameters to 'mtcars'

SELECT reverse_normalize(wt, hp USING PARAMETERS model_name =
'mtcars_normfitrz') FROM mtcars; --reverse the normalization in
'mtcars'
```

Impute missing values
```
SELECT impute ('myImputedView', 'small_input_impute', 'x1,x2,
x3', 'mean' USING PARAMETERS partition_columns='pclass,gender');
--impute the missing value for each cluster independently
```

Process imbalance data
```
SELECT balance ('myOutputView', 'small_input_impute', 'gender',
'over_sampling' USING PARAMETERS sampling_ratio=1.0); --make the
sample size even between 'male' and 'female' samples
```

Sample
```
CREATE TABLE baseball_sample AS SELECT * FROM baseball
TABLESAMPLE(25); --generate a 25% sample set randomly
```

Naive Bayes
```
SELECT naive_bayes('naive_house84_model', 'house84_train',
'party', '*' USING PARAMETERS exclude_columns='party, id');

SELECT party, predict_naive_bayes(vote1, vote2, vote3 USING
PARAMETERS model_name='naive_house84_model', type='response')
AS predicted_party FROM house84_test;

SELECT predict_naive_bayes_classes(id, vote1, vote2, vote3
USING PARAMETERS model_name='naive_house84_model',
key_columns='id', exclude_columns='id', classes='democrat,
republican', match_by_pos='false') OVER() FROM house84_test;
--return the probability of the predicted class and the
specified class 'democrat' and 'republican'
```

Random Forest
```
SELECT rf_classifier('myRFModel', 'iris_train', 'species',
'sepal_length, sepal_width, petal_length, petal_width' USING
PARAMETERS ntree=100, sampling_size=0.3);

SELECT id, predict_rf_classifier(sepal_length, sepal_width,
petal_length, petal_width USING PARAMETERS
model_name='myRFModel') FROM iris_test;

SELECT predict_rf_classifier_classes(id, sepal_length,
sepal_width, petal_length, petal_width USING PARAMETERS
model_name='myRFModel', key_columns ='id',
exclude_columns='id') OVER () FROM iris_test; --return the
probability of the predicted class
```

Evaluating model performance
Regression metrics
Mean Squared Error
```
SELECT mse(obs, pred) OVER() FROM (SELECT eruptions AS obs,
PREDICT_LINEAR_REG (waiting USING PARAMETERS
model_name='myLinearRegModel') AS pred FROM faithful_testing)
AS prediction_output;
```

R Squared
```
SELECT rsquared(obs, pred) OVER() FROM (SELECT eruptions AS
obs, PREDICT_LINEAR_REG (waiting USING PARAMETERS
model_name='myLinearRegModel') AS pred FROM faithful_testing)
AS prediction_output;
```

Classification metrics
Confusion Matrix
```
SELECT confusion_matrix(obs::int, pred::int USING PARAMETERS
num_classes=2) OVER() FROM (SELECT am AS obs,
predict_logistic_reg(mpg, cyl, disp, drat, wt, qsec, vs,
gear, carb USING PARAMETERS
model_name='myLogisticRegModel')::INT AS pred FROM mtcars) AS
prediction_output;
```

Error Rate
```
SELECT error_rate(obs::int, pred::int USING PARAMETERS
num_classes=2) OVER() FROM (SELECT am AS obs,
predict_logistic_reg(mpg, cyl, disp, drat, wt, qsec, vs,
gear, carb USING PARAMETERS model_name='myLogisticRegModel',
type='response') AS pred FROM mtcars) AS prediction_output;
```

Lift Table
```
SELECT lift_table(obs::int, prob USING PARAMETERS num_bins=2)
OVER() FROM (SELECT am AS obs, predict_logistic_reg(mpg, cyl,
disp, drat, wt, qsec, vs, gear, carb USING PARAMETERS
model_name='myLogisticRegModel', type='probability') AS prob
FROM mtcars) AS prediction_output;
```

ROC
```
SELECT roc(obs::int, prob USING PARAMETERS num_bins=2) OVER()
FROM (SELECT am AS obs, predict_logistic_reg(mpg, cyl, disp,
drat, wt, qsec, vs, gear, carb USING PARAMETERS
model_name='myLogisticRegModel', type='probability') AS prob
FROM mtcars) AS prediction_output;
```