Vertica ML Python Workshop

Exercise 2: Outliers & Anomaly Detection

Ouali Badr

December 5, 2019

Executive Summary

"Science knows no country, because knowledge belongs to humanity, and is the torch which illuminates the world."

Louis Pasteur

VERTICA ML PYTHON allows the users to use Vertica advanced analytics and Machine Learning with a Python front-end Interface. In this exercise, you'll learn some basics to begin your fantastic Data Science Journey with the API. As a summary:

- Detect outliers using 4 different techniques (DBSCAN, KMeans, LOF and Normal Distribution)
- Fill outliers
- Drop outliers
- Find the KMeans K
- Realize different scatter plots

Contents

1 Presentation

Outliers are a difficult problem to handle in Data Science. They can reveal unknown patterns (system failure, frauds, errors...) and they are in some cases adding bias to the data which can lead to wrong predictions. They need to be detected in order to be deleted or adjusted. Finding all of them need the usage of some very sophisticated techniques (Algorithms like DBSCAN, Local Outlier Factor or KMeans for example). We can organize outliers into 3 categories:

- **Global Outliers:** Point Anomalies. Their values are far outside the entirety of the data set in which they are found. An example could be a student who normally withdraw no more than 100$, and suddenly he withdrew more than 1000$ in the ATM machine.
- **Contextual Outliers:** Conditional Outliers. Their values significantly deviates from the rest of the data points in the same context. An example could be that because of a very bad weather, people didn't decide to go out for the Black Friday which decreased considerably the stores taking.
- **Collective Outliers:** Anomalous Subset. Their values as a collection deviate significantly from the entire data set but they are not global or contextual outliers. An example could be that some of the purchases of different products didn't work at the same moment. Those can be merged into a single anomaly.

During this exercise, we will use different techniques (clustering and anomaly detection) to deal with outliers. We will use the heart disease dataset. This dataset contains many information of 270 patients including:

- **age:** age in years
- **thalach:** maximum heart rate achieved
- **trestbps:** resting blood pressure (in mm Hg on admission to the hospital)
- **fbs:** (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- **slope:** the slope of the peak exercise ST segment (Value 1: upsloping; Value 2: flat; Value 3: downsloping)
- **ca:** number of major vessels (0-3) colored by flourosopy
- **num:** diagnosis of heart disease (angiographic disease status) (Value 1: < 50% diameter narrowing; Value 2: > 50% diameter narrowing)
- **sex:** sex (1 = male; 0 = female)
- **cp:** hest pain type (Value 1: typical angina; Value 2: atypical angina; Value 3: non-anginal pain; Value 4: asymptomatic)
- **exang:** exercise induced angina (1 = yes; 0 = no)
- **restecg:** resting electrocardiographic results (Value 0: normal; Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV); Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria)
- **chol:** serum cholestoral in mg/dl
- **oldpeak:** ST depression induced by exercise relative to rest
- **thal:** 3 = normal; 6 = fixed defect; 7 = reversable defect

The purpose is to find people having heart complications compare to the rest.

2 Functions used during the Exercise

2.1 scatter

Library: vertica_ml_python.vDataframe

```
vDataframe.scatter(
        self,
          columns: list,
          cat_col: str = "",
          max_cardinality: int = 3,
          cat_priority: list = [],
          with_others: bool = True,
          max_nb_points: int = 20000)
```

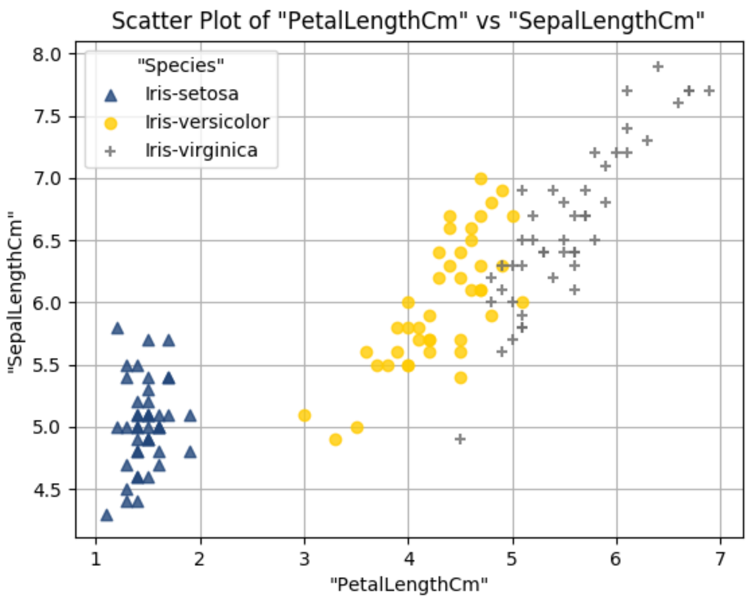
Draw the scatter plot of the input columns.

Parameters

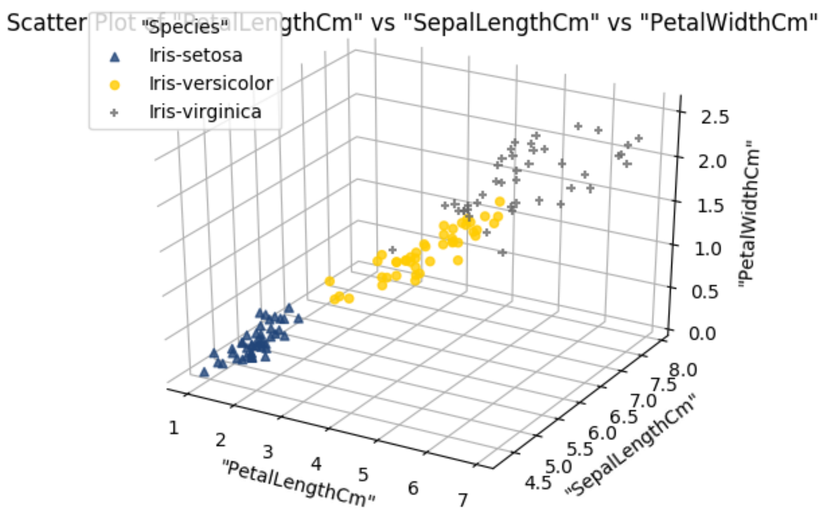
- **columns:** *<str>*
  The two or three columns used to draw the scatter plot.
- **catcol:** *<str>*, optional
  The categorical column used as label.
- **max_cardinality:** *<int>*, optional
  The maximum cardinality of the categorical column, all the other categories are merged to create the "others" category.
- **cat_priority:** *<list>*, optional
  The list of the categories took into account during the computation.
- **with_others:** *<bool>*, optional
  Include the "others" category.
- **max_nb_points:** *<int>*, optional
  The maximum number of points in the scatter plot. The points are taken randomly from the table.

Examples

```
from vertica_ml_python.learn.datasets import load_iris
iris = load_iris(cur)
iris.scatter(columns = ["PetalLengthCm", "SepalLengthCm"], catcol = "Species")
```



```
iris.scatter(columns = ["PetalLengthCm", "SepalLengthCm", "PetalWidthCm"],
    catcol = "Species")
```



2.2 outliers

Library: vertica_ml_python.vDataframe

```
vDataframe.outliers(
        self,
        columns: list = [],
        name: str = "distribution_outliers",
        threshold: float = 3.0)
```

Find the outliers of the distribution regarding specific columns and create a new Virtual Column to label the outliers.

Parameters

- **columns:** *<list>*, optional
  The columns used to compute the outliers.
- **name:** *<str>*, optional
  Name of the new feature which will represent the outliers.
- **threshold:** *<float>*, optional
  Use the Gaussian distribution to define the outliers. After normalizing the data, if the record is greater than the threshold it will be considered as an outlier.

Returns

The vDataframe itself.

Example

```
from vertica_ml_python.learn.datasets import load_titanic
titanic = load_titanic(cur)
titanic.outliers()

#Output
The new vColumn "distribution_outliers" was added to the vDataframe.
        age    body    survived      ticket                            home.dest
      \\
0     2.000    None           0      113781    Montreal, PQ / Chesterville, ON
      \\
1    30.000     135           0      113781    Montreal, PQ / Chesterville, ON
      \\
2    25.000    None           0      113781    Montreal, PQ / Chesterville, ON
      \\
3    39.000    None           0      112050                        Belfast, NI
      \\
4    71.000      22           0    PC 17609                Montevideo, Uruguay
      \\
...     ...     ...         ...         ...                                ...
      \\
      cabin        sex    pclass    embarked    parch    \\
0    C22 C26    female         1           S        2    \\
1    C22 C26      male         1           S        2    \\
2    C22 C26    female         1           S        2    \\
3        A36      male          1           S        0    \\
4        None     male          1           C        0    \\
...      ...      ...         ...         ...      ...    \\
          fare                                    name      boat    \\
0    151.55000               Allison, Miss. Helen Loraine    None    \\
1    151.55000        Allison, Mr. Hudson Joshua Creighton    None    \\
2    151.55000  Allison, Mrs. Hudson J C (Bessie Wald...    None    \\
3      0.00000                    Andrews, Mr. Thomas Jr    None    \\
4     49.50420                   Artagaveytia, Mr. Ramon    None    \\
...        ...                                       ...     ...    \\
     sibsp    distribution_outliers
0        1                        0
1        1                        0
2        1                        0
3        0                        0
4        0                        0
...    ...                      ...
Name: titanic, Number of rows: 1234, Number of columns: 15
```

2.3 drop_outliers

Library: vertica_ml_python.vDataframe

```
vDataframe[].drop_outliers(
        self,
        alpha: float = 0.05,
        use_threshold: bool = True,
        threshold: float = 4.0)
```

Drop the columns outliers.

Parameters

- **alpha:** *<float>*, optional
  Number representing the outliers threshold. Values lesser than quantile(alpha) or greater than quantile(1-alpha) will be dropped.
- **use_threshold:** *<bool>*, optional
  Use the threshold instead of the 'alpha' parameter
- **threshold:** *<float>*, optional
  Use the Gaussian distribution to define the outliers. After normalizing the data, if the record is greater than the threshold it will be considered as an outlier.

Returns

The parent vDataframe.

Example

```
from vertica_ml_python.vdataframe import vdf_from_relation
relation = "((SELECT 5 AS x) UNION ALL (SELECT -1 AS x) UNION ALL (SELECT 2 AS
     x) UNION ALL (SELECT -9 AS x) UNION ALL (SELECT -600 AS x) UNION ALL (
    SELECT 100000 AS x)) z"
vdf = vdf_from_relation(relation, dsn = "VerticaDSN")
vdf["x"]

#Output
          x
0         5
1        -1
2         2
3        -9
4      -600
5    100000
Name: x, Number of rows: 6, dtype: int

vdf["x"].div(x = 5)

#Output
      x
0     5
1    -1
2     2
3    -9
Name: x, Number of rows: 4, dtype: int
```

2.4 fill_outliers

Library: vertica_ml_python.vDataframe

```
vDataframe[].fill_outliers(
      self,
      method: str = "winsorize",
      alpha: float = 0.05,
      use_threshold: bool = True,
      threshold: float = 4.0)
```

Fill the outliers using the input method.

Parameters

- **method:** *<dict>*, optional
  Method used to fill the column outliers.

winsorize | null | mean
winsorize (default): clip the data using as lower bound quantile(alpha) and as upper bound quantile(1-alpha).
null: Replace the outliers by the NULL value.
mean: Replace the upper and lower outliers by their respective average.

- **alpha:** *<float>*, optional
  Number representing the outliers threshold. Values lesser than quantile(alpha) or greater than quantile(1-alpha) will be filled.
- **use_threshold:** *<bool>*, optional
  Use the threshold instead of the 'alpha' parameter
- **threshold:** *<float>*, optional
  Use the Gaussian distribution to define the outliers. After normalizing the data, if the record is greater than the threshold it will be considered as an outlier.

Returns

The parent vDataframe.

Example

```
from vertica_ml_python.vdataframe import vdf_from_relation
relation = "((SELECT 5 AS x) UNION ALL (SELECT 600 AS x) UNION ALL (SELECT 2
    AS x) UNION ALL (SELECT -100 AS x) UNION ALL (SELECT -2 AS x) UNION ALL (
    SELECT 12 AS x)) z"
vdf = vdf_from_relation(relation, dsn = "VerticaDSN")

#Output
         x
0        5
1      600
2        2
3     -100
4       -2
5       12
Name: x, Number of rows: 6, dtype: int

vdf["x"].fill_outliers(method = "null")

#Output
         x
0        5
1     None
2        2
3     None
4       -2
5       12
Name: x, Number of rows: 6, dtype: int
```

2.5 eval

Library: vertica_ml_python.vDataframe

```
vDataframe.eval(self, name: str, expr: str)
```

Evaluate an expression and create a new column if the expression is correct.

Parameters

- **name:** *<str>*
  Name of the new column
- **expr:** *<str>*
  The expression used to create the new column

Returns

The vDataframe itself.

Example

```
from vertica_ml_python.vdataframe import vdf_from_relation
relation = "((SELECT 2 AS x, 4 AS y) UNION ALL (SELECT 3 AS x, 4 AS y) UNION
    ALL (SELECT 10 AS x, 5 AS y)) z"
vdf = vdf_from_relation(relation, dsn = "VerticaDSN")


#Output
      x    y
0     2    4
1     3    4
2    10    5
Name: VDF, Number of rows: 3, Number of columns: 2

vdf.eval("z", "x * y")

#Output
      x    y     z
0     2    4     8
1     3    4    12
2    10    5    50
Name: VDF, Number of rows: 3, Number of columns: 3
```

2.6 LocalOutlierFactor

Create a LocalOutlierFactor object by using the Local Outlier Factor algorithm as defined by Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng and Jörg Sander. This object is using pure SQL to compute all the distances and final score. It is using CROSS JOIN and may be really expensive in some cases. It will index all the elements of the table in order to be optimal (the CROSS JOIN will happen only with IDs which are integers). As LocalOutlierFactor is using the p-distance, it is highly sensible to un-normalized data.

initialization

Library: vertica_ml_python.learn.neighbors

```
class LocalOutlierFactor(
  name: str,
  cursor,
  n_neighbors: int = 20,
  p: int = 2)
```

Parameters

- **name:** *<str>*
  Name of the relation created after fitting the model.
- **cursor:** *<object>*
  DB cursor.
- **n_neighbors:** *<int>*, optional
  Number of neighbors to consider when computing the score.
- **p:** *<int>*, optional
  The p corresponding to the one of the p-distance (distance metric used during the model computation).

Methods

The LocalOutlierFactor object has five main methods:

```
# Fit the model with the input columns
def fit(self, input_relation: str, X: list, key_columns: list = [], index = ""
    )

# Print the model information
def info(self)

# Plot the model (only possible if the number of columns <= 3)
def plot(self)

# Create a vDataframe using the final relation
def to_vdf(self)
```

The index parameter (name of the primary key column in the relation) of the 'fit' method is very important as without it an indexed copy of the main relation will be created (it could be really expensive). It is highly recommended to have a primary key column in the main table to avoid unnecessary computations.

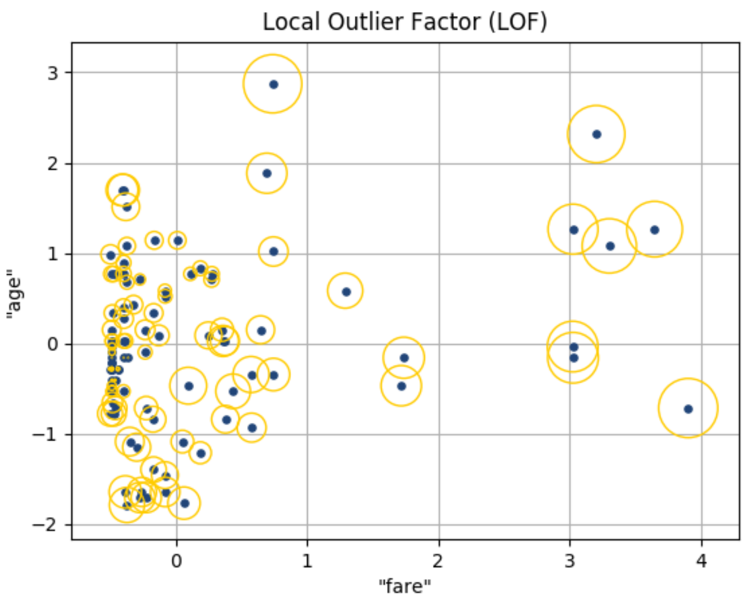
Attributes

The LocalOutlierFactor object has one main attribute:

```
self.n_errors # Number of errors
```

Example

```
from vertica_ml_python import vDataframe
from vertica_ml_python.learn.neighbors import LocalOutlierFactor

# Building a normalized relation of the features, we will use
titanic = vDataframe("titanic", cur)
# We will use a sample of the data in order to have a beautiful plot
titanic.main_relation += " TABLESAMPLE(10)"
titanic = titanic.select(["fare", "age"])
titanic.normalize()
titanic.to_db("titanic_normalize")

# We can build the model
model = LocalOutlierFactor("lof_titanic", cur)
model.fit("titanic_normalize", ["fare", "age"])

# We can see the different information relative to the model
model.info()

# Output
All the LOF scores were computed.

# And Plot the model
model.plot()
```



2.7 DBSCAN

Create a DBSCAN object by using the DBSCAN algorithm as defined by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu. This object is using pure SQL to compute all the distances and neighbors. It is also using Python to compute the cluster propagation (non scalable phase). This model is using CROSS JOIN and may be really expensive in some cases. It will index all the elements of the table in order to be optimal (the CROSS JOIN will happen only with IDs which are integers). As DBSCAN is using the p-distance, it is highly sensible to un-normalized data. However, DBSCAN is really robust to outliers and can find non-linear clusters. It is a very powerful algorithm for outliers detection and clustering.

initialization

Library: vertica_ml_python.learn.cluster

```
class DBSCAN(
  name: str,
  cursor,
  eps: float = 0.5,
  min_samples: int = 5,
  p: int = 2)
```

Parameters

- **name:** *<str>*
  Name of the relation created after fitting the model.
- **cursor:** *<object>*
  DB cursor.
- **eps:** *<float>*, optional
  The radius of a neighborhood with respect to some point.
- **min_samples:** *<int>*, optional
  Minimum number of points required to form a dense region.
- **p:** *<int>*, optional
  The p corresponding to the one of the p-distance (distance metric used during the model computation).

Methods

The DBSCAN object has four main methods:

```
# Fit the model with the input columns
def fit(self, input_relation: str, X: list, key_columns: list = [], index = ""
    )

# Print the model information
def info(self)

# Plot the model (only possible if the number of columns <= 3)
def plot(self)

# Create a vDataframe using the final relation
```

```
def to_vdf(self)
```

The index parameter (name of the primary key column in the relation) of the 'fit' method is very important as without it an indexed copy of the main relation will be created (it could be really expensive). It is highly recommended to have a primary key column in the main table to avoid unnecessary computations.

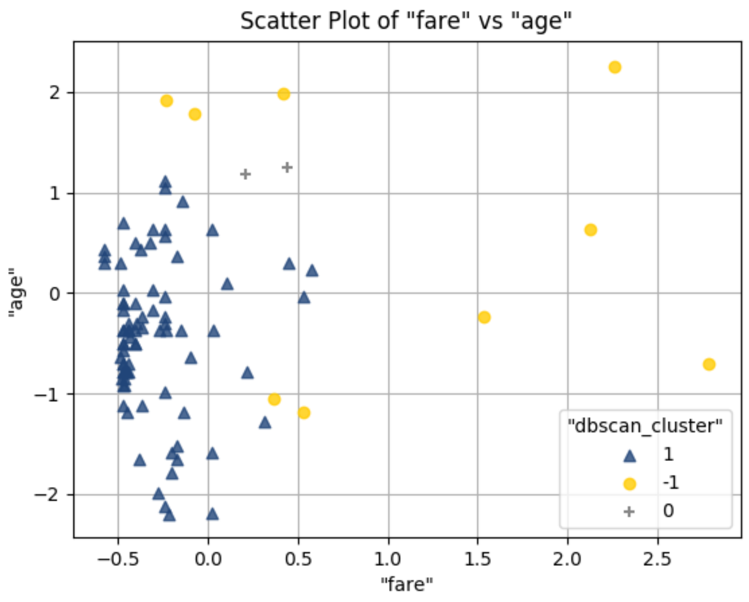
Attributes

The DBSCAN object has two main attributes:

```
self.n_cluster # Number of clusters
self.n_noise # Number of elements considered as noise by the algorithm
```

Example

```
from vertica_ml_python import vDataframe
from vertica_ml_python.learn.cluster import DBSCAN

# Building a normalized relation of the features, we will use
titanic = vDataframe("titanic", cur)
# We will use a sample of the data in order to have a beautiful plot
titanic.main_relation += " TABLESAMPLE(10)"
titanic = titanic.select(["fare", "age"])
titanic.normalize()
titanic.to_db("titanic_normalize")

# We can build the model
model = DBSCAN("dbscan_titanic", cur)
model.fit("titanic_normalize", ["fare", "age"])

# We can see the different information relative to the model
model.info()

# Output
DBSCAN was successfully achieved by building 3 cluster(s) and by identifying
    21 elements as noise.
If you are not happy with the result, do not forget to normalize the data
    before applying DBSCAN. As this algorithm is using the p-distance, it is
    really sensible to the data distribution.

# And Plot the model
model.plot()
```



2.8 KMeans

Create a KMEANS object by using the Vertica Highly Distributed and Scalable KMEANS directly on the data.

initialization

Library: vertica_ml_python.learn.cluster

```
class KMEANS(
  name: str,
  cursor,
  n_cluster: int = 8,
  init = "kmeanspp",
  max_iter: int = 300,
  tol: float = 1e-4)
```

Parameters

- **name:** *<str>*
  Name of the the model. The model is stored in the DB.
- **cursor:** *<object>*
  DB cursor.
- **n_cluster:** *<int>*, optional
  Number of clusters.
- **init:** *<str or list>*, optional
  The method used to find the initial cluster centers. The method can be in {random | kmeanspp}. It can be also a list with the initial cluster centers to use.
- **max_iter:** *<int>*, optional
  The maximum number of iterations the algorithm performs.

- **tol:** *<float>*, optional
  Determines whether the algorithm has converged. The algorithm is considered converged after no center has moved more than a distance of 'tol' from the previous iteration.

Methods

The KMeans object has four main methods:

```python
# Add the cluster prediction in a vDataframe
def add_to_vdf(self, vdf, name: str = "")

# Drop the model
def drop(self)

# Fit the model with the input columns
def fit(self, input_relation: str, X: list)

# Plot the model (only possible if the number of columns <= 3)
def plot(self)
```

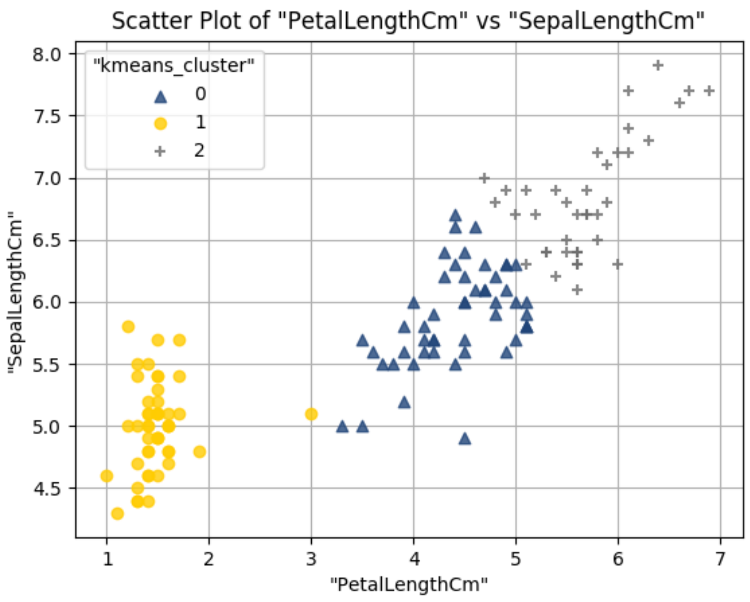
Attributes

The KMEANS object has two main attributes:

```python
self.cluster_centers # Position of the cluster centers
self.metrics # Different metrics to evaluate the model
```

Example

```python
from vertica_ml_python.learn.cluster import KMeans

# We can build the model
model = KMeans("kmeans_iris", cur, n_cluster = 3)
model.fit("iris", ["PetalLengthCm", "SepalLengthCm"])

# We can see the different metrics relative to the model
model.metrics

# Output
                                                    value
Between-Cluster Sum of Squares                  512.23072
Total Sum of Squares                            566.03207
Total Within-Cluster Sum of Squares             53.801351
Between-Cluster SS / Total SS          0.9049499969144859
converged                                            True

# And Plot the model
model.plot()
```



2.9 elbow

Library: vertica_ml_python.learn.plot

```
def elbow(
        X: list,
        input_relation: str,
        cursor,
        n_cluster = (1, 15),
        init = "kmeanspp",
        max_iter: int = 50,
        tol: float = 1e-4)
```

Draw the Elbow Curve. This curve is helpful to find the number of clusters of a KMeans model.

Parameters

- **X:** *<list>*
  List of the predictor columns.
- **input_relation:** *<str>*
  Relation used to train the model.
- **cursor:** *<object>*
  DB cursor.
- **n_cluster:** *<tuple or list>*, optional
  Tuple representing the number of cluster to start with and to end with. You can also use a list of different cluster values.

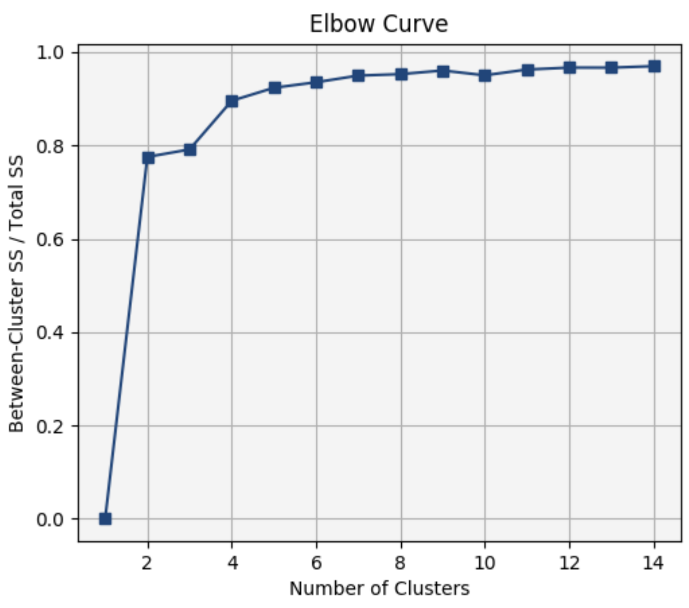
- **init:** *<str or list>*, optional
  The method used to find the initial cluster centers. The method can be in {random | kmeanspp}.
  It can be also a list with the initial cluster centers to use.

- **max_iter:** *<int>*, optional
  The maximum number of iterations the algorithm performs.

- **tol:** *<float>*, optional
  Determines whether the algorithm has converged. The algorithm is considered converged after no center has moved more than a distance of 'tol' from the previous iteration.

Example

```python
from vertica_ml_python.learn.plot import elbow

# We can build the model
elbow(["PetalLengthCm", "SepalWidthCm", "SepalLengthCm"], "iris", cur)
```



3 Questions

Turn on Jupyter with the 'jupyter notebook' command. Start the notebook exercise2.ipynb and answer to the following questions.

- **Question 1:** Draw the Scatter Plot of the two variables. What do you notice ?

- **Question 2:** Add the Global Outliers of the distribution using the 'outliers' method. Draw again the Scatter Plot with this time marking the outliers. Find visually the number of outliers and explain the result.

- **Question 3:** Use the DBSCAN with epsilon = 20 and min_samples = 10. The outliers will be labeled with the class -1. What do you notice ?

- **Question 4:** Now, use the Local Outlier Factor Algorithm and draw the LOF graph. Build a Virtual Dataframe using the created table and create a new label based on the LOF (>1.5 = outliers)
- **Question 5:** The last technique to find outliers is using the KMeans algorithm. However, we never know the number of clusters in advance. Use the elbow curve to find it. Create a KMeans model with this number. Draw the model prediction and identify the outliers.
- **Question 6:** Use your favourite technique to identify the outliers and draw histograms to see what part of the population is affected by huge complications.