ASSIGNMENT TWO

Semester 2 - 2021

PAPER NAME: Data Mining and Machine Learning

PAPER CODE: COMP809

DUE DATE: Sunday 24 Oct 2021 at midnight

TOTAL MARKS: 100

Student Name: Bernard O'Leary

Student ID: 19075193

Note: This assignment must be complemented individually

Submission: A soft copy needs to be submitted through Turnitin (a link for this purpose will be set up in Blackboard) Include your actual code (no screenshot) in Appendix with appropriate comments for each task.

INSTRUCTIONS:

1. ACADEMIC INTEGRITY GUIDELINES

The following actions may be deemed to constitute a breach of the General Academic Regulations Part 7: Academic Discipline, Section 2 Dishonesty During Assessment or Course of Study

- 2.1.1 copies from, or inappropriately communicates with another person
- 2.1.3 plagiarises the work of another person without indicating that the work is not the student's own – using the full work or partial work of another person without giving due credit to the original creator of that work
- 2.1.4 Unauthorised collaboration in Assessment - collaborates with others in the preparation of material, except where this has been approved as an assessment requirement. This includes contract cheating where a student obtains services to produce or assist with an assessment
- 2.1.5 resubmits previously submitted work without prior approval of the exam board
- 2.1.6 Using any other unfair means

Please email DCT.EXAM@AUT.AC.NZ if you have any technical issues with your online submission on Blackboard immediately

Contents

Part A

The pre-processing performed was to bring all of the data in to Pandas dataframes, then apply categorisation to all the data so that it is all numeric and then go through the feature selection process. Following feature selection for each dataset, the selected features were nomalised prior to going through the algorithm.

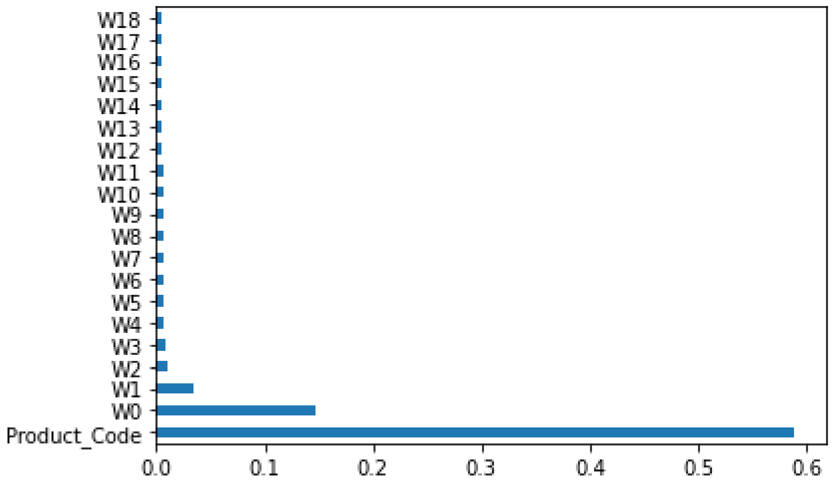
For consistency, the PCA algorithm was used for feature selection for each of the datasets.

Datasets and feature selection

Sales Transaction Data

The Sales Transaction dataset contains weekly volumes of 800 products sold over a period of 52 week. Weeks are represented as columns and products are represented as rows. The dataset resembled a matrix, with a product column on the left. A normalised figure is provided for each record, however I have elected to normailise the entire dataset anyway.

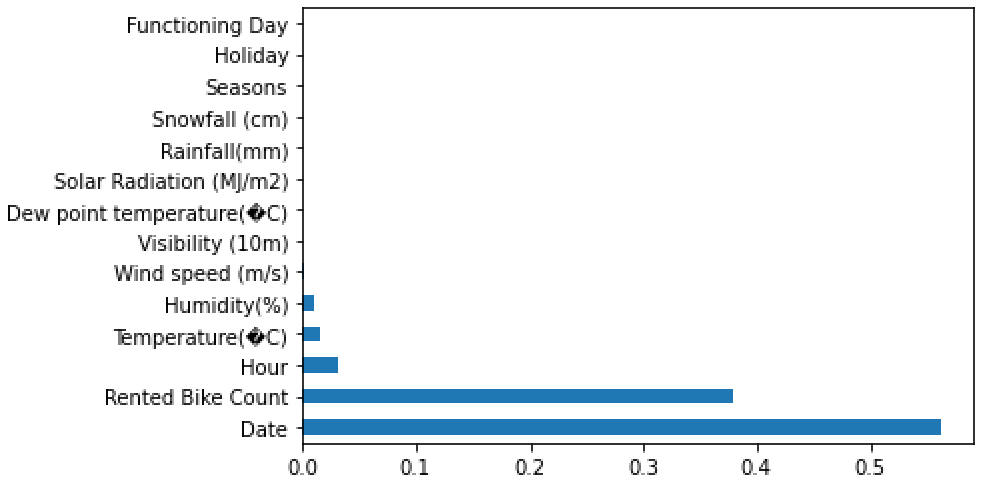
The top three parameters were selected for this data as together they explained most of the variance.



Seoul Bike Data

The Seoul Bike Sharing Demand dataset runs for between start of December 2017 and end of November 2019. There are over 8000 records, which describe the atmospheric conditions on the day and hour of the hirage, how many bikes were rented and whether the day was a holiday.

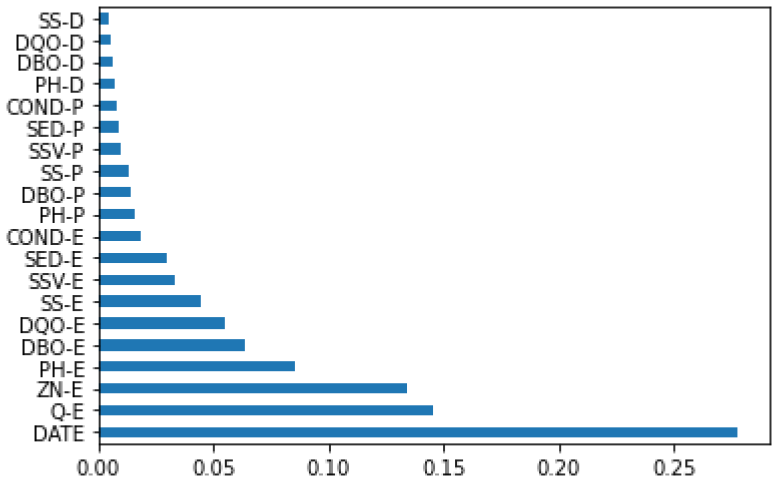
The top two parameters were selected for this data as together they explained most of the variance.



Water Treatment Data

The Water Treatment dataset contains over 500 records and covers a range of parameters that describe the chemical composition of water treatment samples between March 1990 and August 1991 – 18 months worth of data.

The top five parameters were selected for this data as together they explained most of the variance.



Task 1

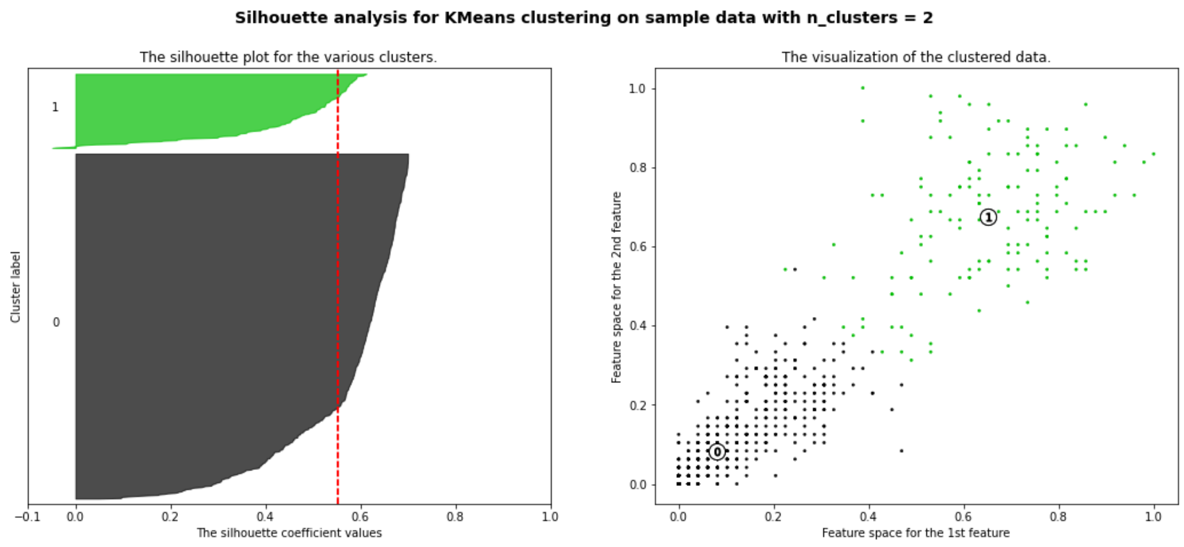
K means algorithm

Table A: K means algorithm

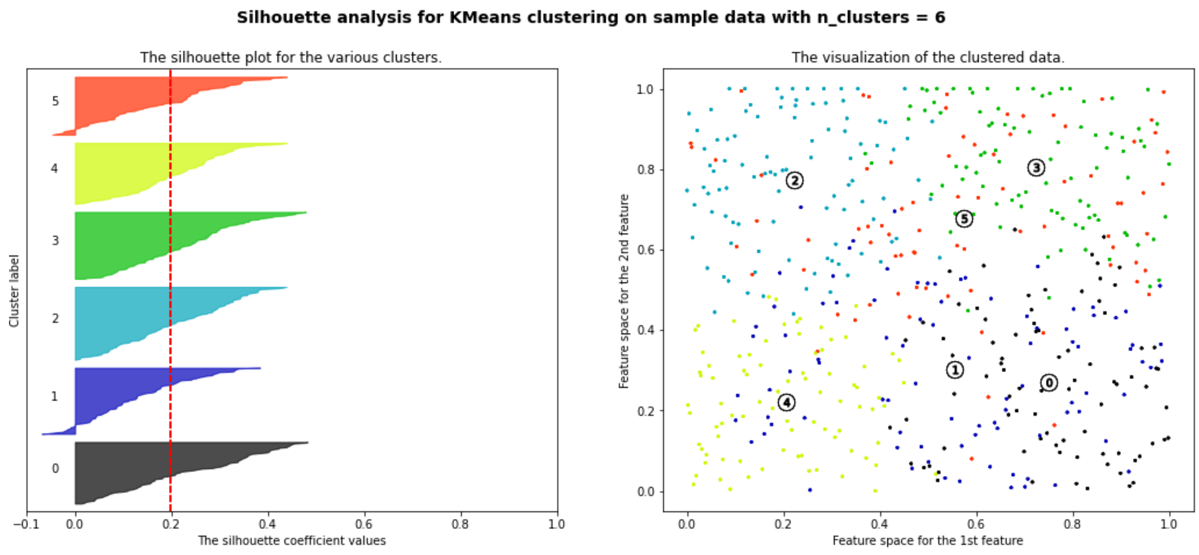
	Time taken (seconds)	CSM / SSE	K parameter
Sales Transactions Data	0.39674830436706543	0.5529922804905653 / 14.437368116537462	2
Water Treatment Data	0.42758846282958984	0.1979742286699438 / 14.219162585579324	6

Seoul Bike Data	5.6746766567230225	0.2736422972611256 / 297.7696053888172	2

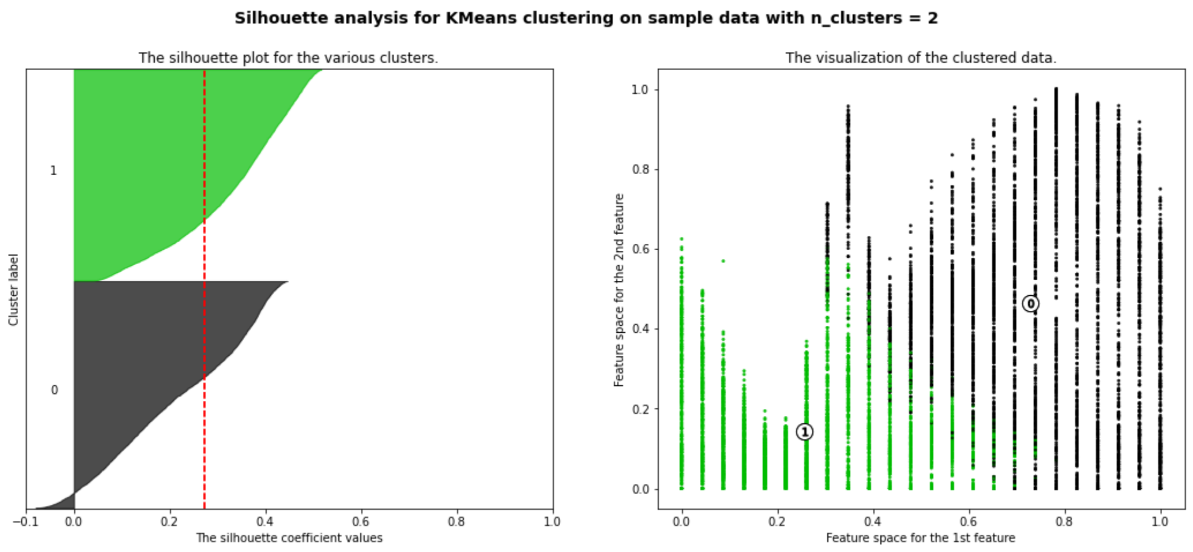
Sales Transactions Data CSM chart for KMeans



Water Treatment Data CSM chart for KMeans



Seoul Bike Data CSM chart for KMeans



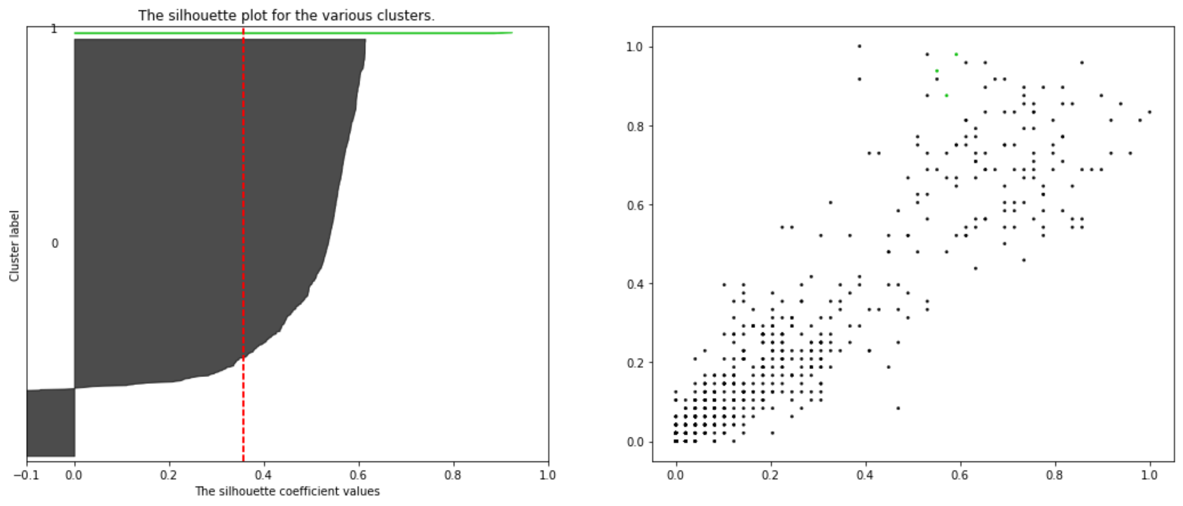
DBSCAN algorithm

Table B: DBSCAN algorithm

For the DBSCAN algorithm I needed to adjust the EPS value also, but unfortunately I was unable to get a correct value of the EPS so that the algorithm would complete correctly for the Water Treatment Data and Seoul Bikes Data.

	Time taken (seconds)	CSM / SSE	EPS parameter
Sales Transactions Data	0.346055269241333	0.35714256858302806 / 27.772162712889894	0.2
Water Treatment Data	[Incomplete]	0.01814249370553166 / [Incomplete]	0.3
Seoul Bike Data	[Incomplete]	-0.48965343976481296 / [Incomplete]	0.1

Sales Transactions Data CSM chart for DBSCAN

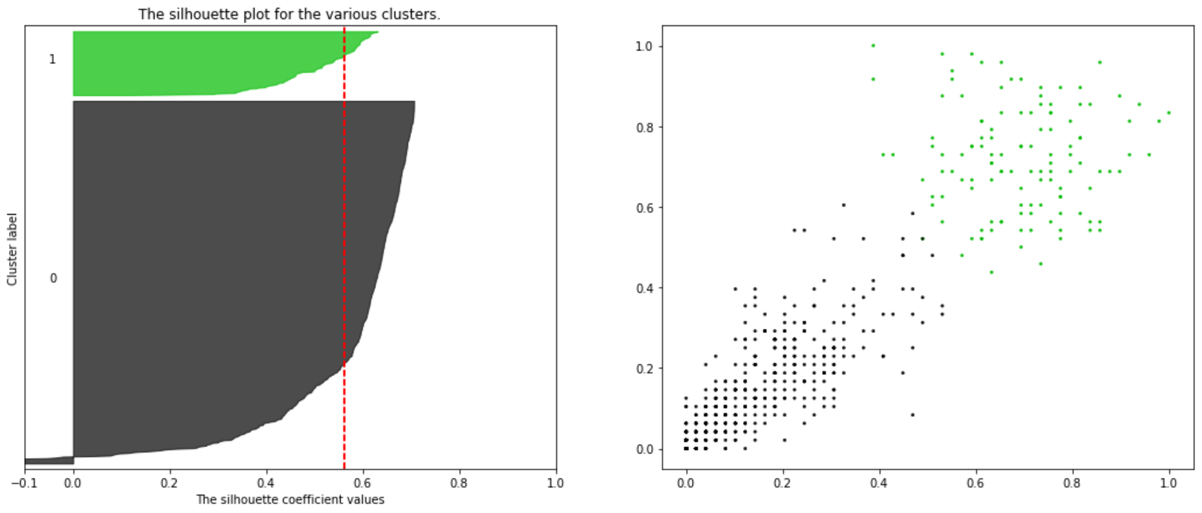


Agglomerative algorithm

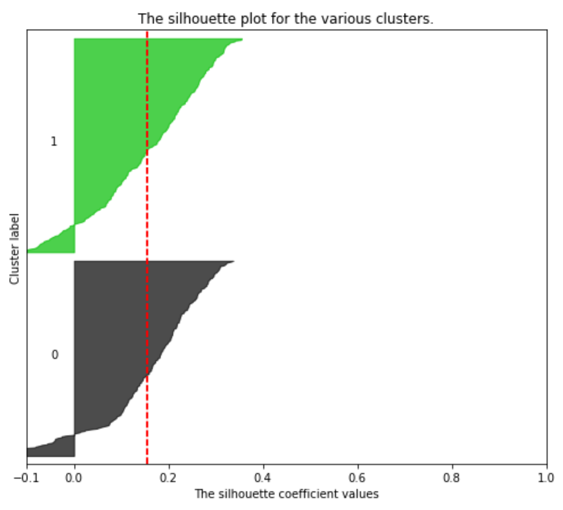
Table C: Agglomerative algorithm

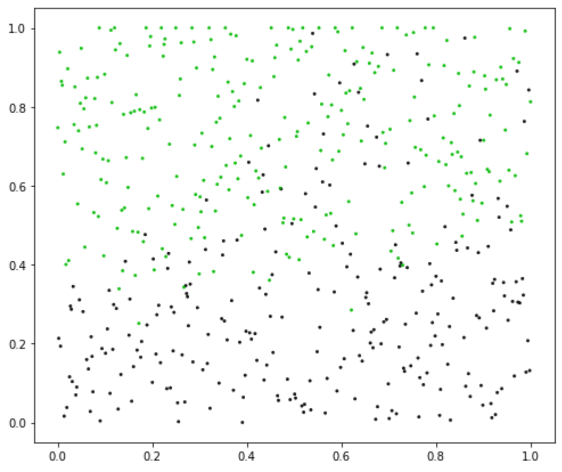
	Time taken (seconds)	CSM / SSE	Clusters parameter
Sales Transactions Data	0.23120903968811035	0.5630813204653953 / 14.907766709758254	2
Water Treatment Data	0.12308955192565918	0.15524541375205828 / 24.894066528279865	2
Seoul Bike Data	8.33217167854309	0.24688690435967905 / 308.6434248773903	2

Sales Transactions Data CSM chart for Agglomerative

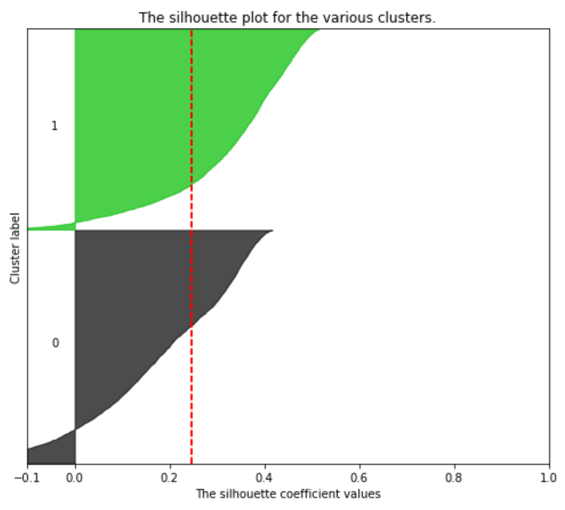


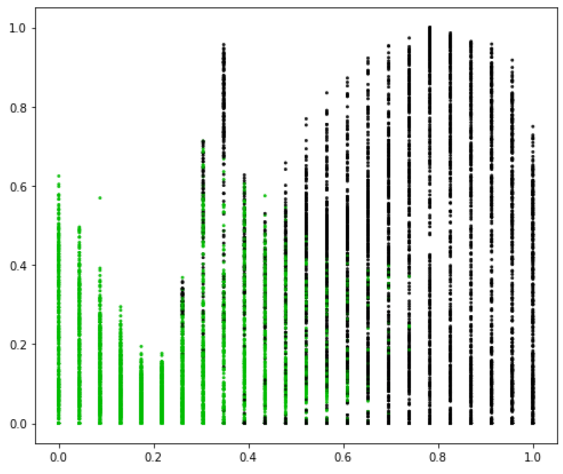
Water Treatment Data CSM chart for Agglomerative





Seoul Bike Data CSM chart for Agglomerative





Task 2

Answer A: which clustering algorithm performed best

The KMeans algorithm appears to perform slower than the aglomorative algorithm in general, although not in all cases. Unfortunately I struggled with the DBSCAN algorithm so have incomplete data for this experiment. For that reason I am only comparing the Aglomorative algorithm and K Means algorithms.

For the Sales Transactions Data the Aglomorative algorithm performed best, with CSM of 0.56 and time taken of 0.23 seconds, for 2 clusters.

For the Water Treatment Data the K Means algorithm performed best, with CSM of 0.197 and time taken of 0.43 seconds, for 2 clusters.

For the Seoul Bike Data the K Means algorithm performed best, with CSM of 0 .27 and time taken of 5.6 seconds, for 2 clusters.

I am using a combination of time taken and CSM score to rank the algorithms. The higher the CSM score, the better the algorithm. If the CSM score is high and the time taken is low, then that algorithm is considered the better one.

Answer B: why did it produced the best value for the CSM measure

Looking at the shape of the data, DBSCAN might have been good to apply to the Seoul Bike dataset as it appears to have some overlap in it's structure. Unfortunately I have have not been able to test this on the dataset. The best performance for the Seoul Bike dataset was K Means.

The result I had for the Water Treatment data was the K Means was the better algorithm. This was somewhat surprising as looking at the dataset, it seems relatively even distributed with no noticeable clustering. I would have thought that the Aglomorative algorithm would have performed better for this dataset.

The result I had for the Sales Transaction data indicated that the Agglomorative algorithm slightly outperformed the K Means algorithm. This dataset is quite neatly dividied in to two clusters, which probably makes it equally good for a hierarchical, a divisive or a partitional algorithm.

Answer C: which clustering algorithm is the overall winner

Of the three algorithms, K Means seemed easiest to work with and to understand. K Means also generally performed the quickest of the three algorithms although I did not get to test DBSCAN to a significant extent.

Part B

Pre-processing

I have used the file Penrose_Hourly_AggregateData_Jan2016Dec2020.csv that was provided for the assignment.

To be able to include the hourly segmentation, I needed to do feature engineering to produce four new columns from the Timestamp column. This was done y concerting the Timestamp type to a datetime and then getting the hour, day, month and year values form the datetime and creating new columns.

I also followed the guidance in the assignment and removed values of greater than 100 from the relatively humidity (%) and temperatures of above 40 degrees Celsius from the Air Temp column.

Finally, all columns were converted to categorical. Initially I wasn't going to do this, but the MLP compained that the target categories were floats. I ended up "binning" these values anyway to reduce the number of possible categories, so this step may not have been necessary.

Feature selection (top 5)

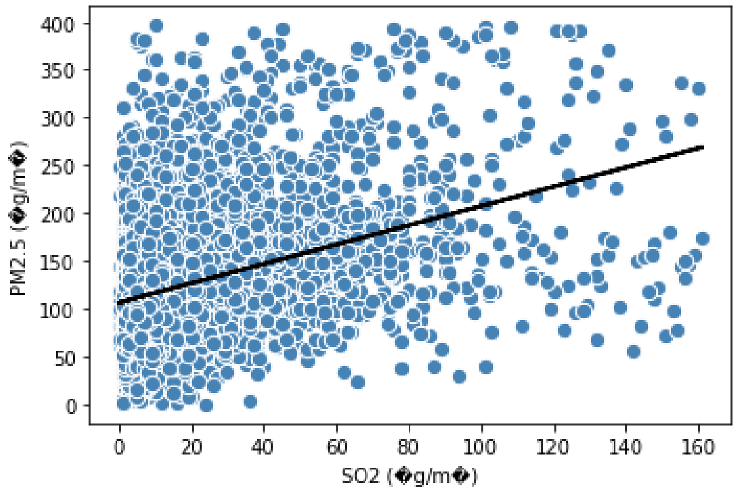
The following sections provide details of the five features that were selected for the predictor variables for the experiment due to their being the variables that had the highest correlation in comparison to the target variable when assessed indicidually against the target (PM2.5 (�g/m�)) using linear regression. A corellation plot for the linear regression model for each predictor is provided.

The statistics for each predictor are provided, including the Pearson Correlation which describes each individual attribute's influence on the target variable.

The reason I used this approach is because it seemed like the most intuitive and simplistic way to get a very clear view of what each predictor variable looked like when it was plotted against the target. An alternative approach would have been to plot all of the predictor vaiables against the target in a single table.

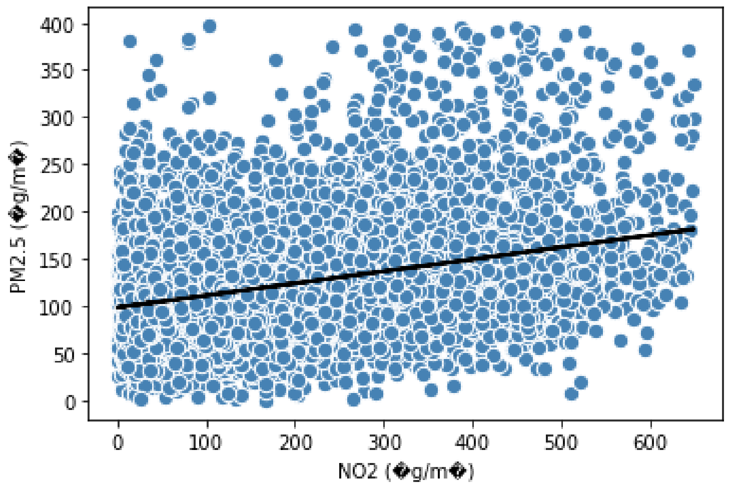
SO2 (�g/m�)

```
Slope: 1.011
Intercept: 106.215
explained_variance:  0.1236
r2:  0.1236
MAE:  36.4505
MSE:  2458.9511
RMSE:  49.5878
Pearsons correlation:  0.3515
```



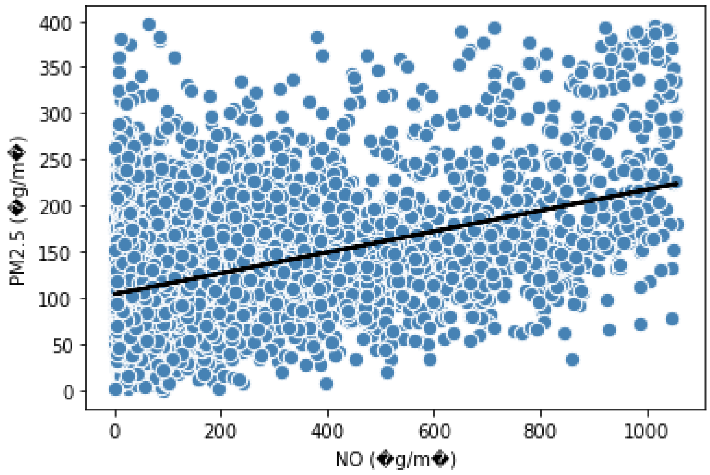
NO2 (�g/m�)

```
Slope: 0.128
Intercept: 98.190
explained_variance:  0.1137
r2:  0.1137
MAE:  36.438
MSE:  2486.5994
RMSE:  49.8658
Pearsons correlation:  0.3372
```



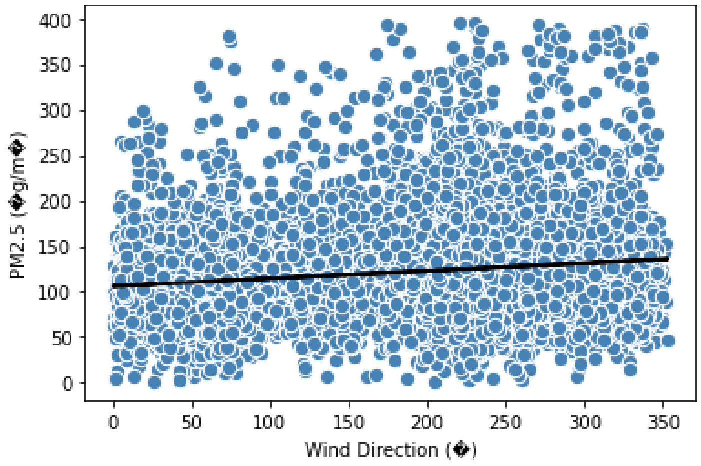
NO (�g/m�)

Slope: 0.113
Intercept: 103.757
explained_variance: 0.1904
r2: 0.1904
MAE: 35.2568
MSE: 2271.5162
RMSE: 47.6604
Pearsons correlation: 0.4363



Wind Direction (�)

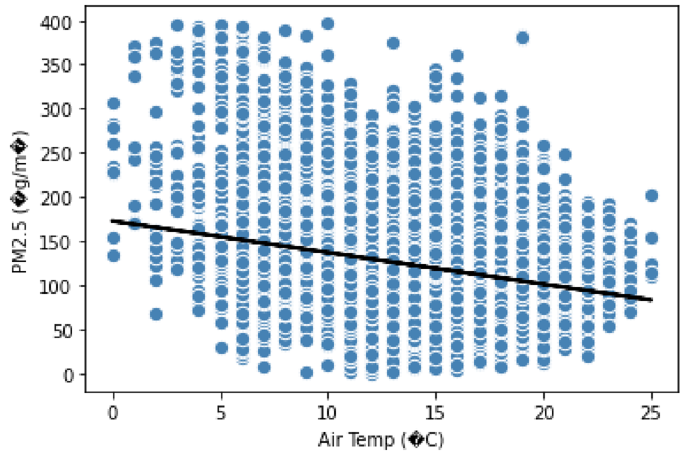
Intercept: 106.028
explained_variance: 0.0211
r2: 0.0211
MAE: 38.016
MSE: 2746.4533
RMSE: 52.4066
Pearsons correlation: 0.1452



Air Temp (�C)

Slope: -3.581
Intercept: 172.655
explained_variance: 0.0722
r2: 0.0722
MAE: 37.9256
MSE: 2603.0704
RMSE: 51.0203

Pearsons correlation: -0.2687



Summary statistics for PM2.5 (�g/m�)

The following summary statistics were attained for the target variable by using the .describe() function against the variable, as is provided by the Pandas package.

```
count   10313.000000
mean      121.846310
std       52.970365
min        0.000000
25%        88.000000
50%       113.000000
75%       145.000000
max       397.000000
Name: PM2.5 (�g/m�), dtype: float64
```

Table of summary stats for predictors

The formula for Pearson Corrleation is as follows:

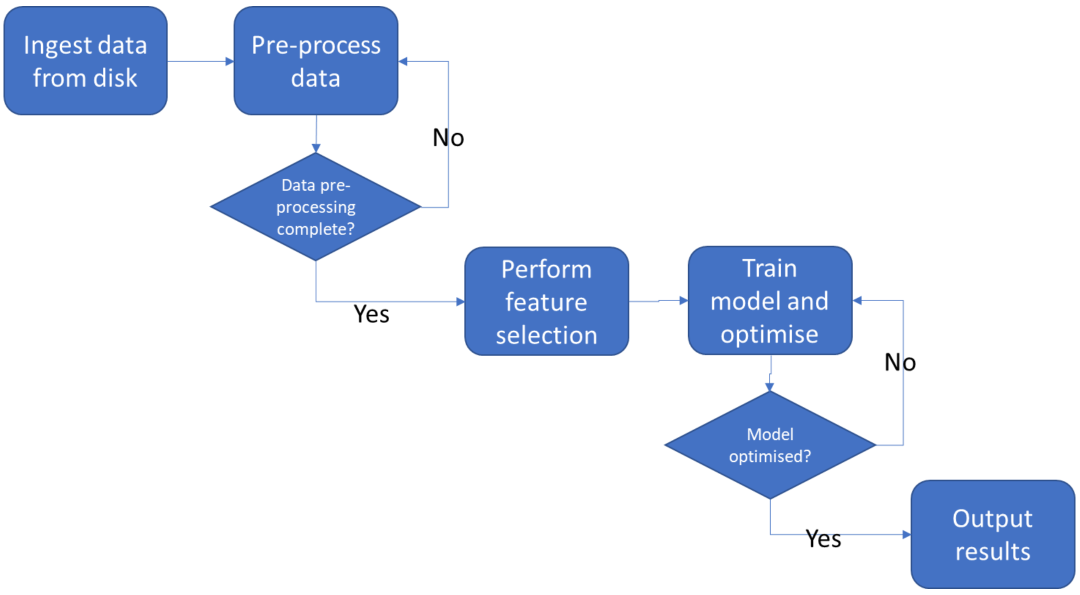
$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

The five best matched variables based on Pearson Correlation are covered in the following table. The Pearson Correlation value provided is the result of a linear regression for each variable against the target. Higher values are better and generally values that are higher in each direction than 0.5 or -0.5 are considered highly correlated.Unfortunately I didn't get any results that were highly correlated.

Predictor variable	Pearson Correlation value
SO2 (�g/m�)	0.3515
NO2 (�g/m�)	0.3372
NO (�g/m�)	0.4363
Wind Direction (�)	0.1452
Air Temp (�C)	-0.2687

Experimental methods

The following diagram issulstrates the process used to generated the model that is created for the MLPClassifier and the LTSM model. This is a high-level overview of the process, and detailed steps such as the train/test-split are incorporated into the bigger parts of the process.

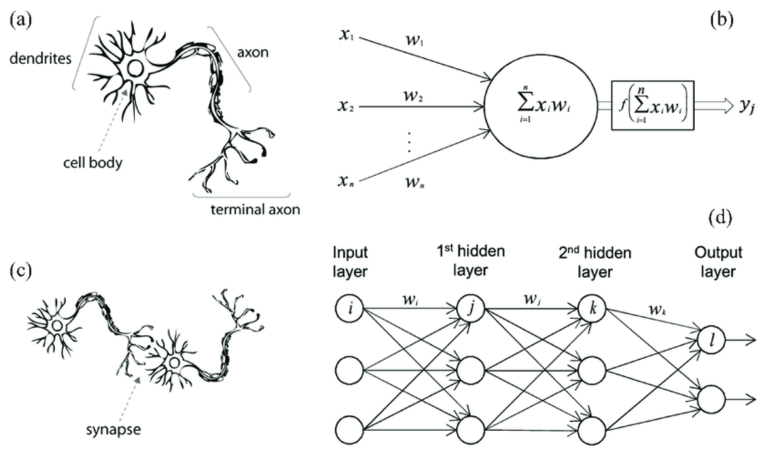


Multilayer Perceptron (MLP)

Answer 1: describe multilayer perceptron

The Multilayer Perceptron is a machine learning approach that tries to mimic the process biological process that is used by brain to achieve predictive capability based on incoming information.

The perceptron element of the model mimics the behaviour of a biological neuron in a brain, and synapses, which link neorons together.



This image is taken from Meng Z., et.al. [1] provides an overview for the model of the synapse and the neuron that the Multi Layer Perceptron borrows from.

The neuron is modelled by the “perceptron” by emulating input signals, applying a weight to each signal, summing the values of the output of the signals multiplied by the respective weights and then deciding of it will in turn output a signal based on whether the sum of the values input and calculated cross a threshold, defined by another function such as the Sigmod function. The neuron in the brain performs this work using input chemical signal, whereas the perceptron performs work work using digital signals. Perceptrons are combined in layers to mimic the function of synapses, which link neurons in the brain.

Artificial neural networks have been around as a concept for a relatively long time, but have become increasingly popular and viable in recent years with the availability of highly parallel computing models such as GPU processors and cloud computing. Because an artificial neural networks mimic the biological brain which is a massively parallel computing process, the ANN also requires platforms that support highly parallel worksloads.

A limitation of the ANN is that it is not inherently explainable in that the process that the training algoritm applies to produce the optimised model is very difficult to interpret in terms of understanding which aspect of the model is responsible for a particular outcome.

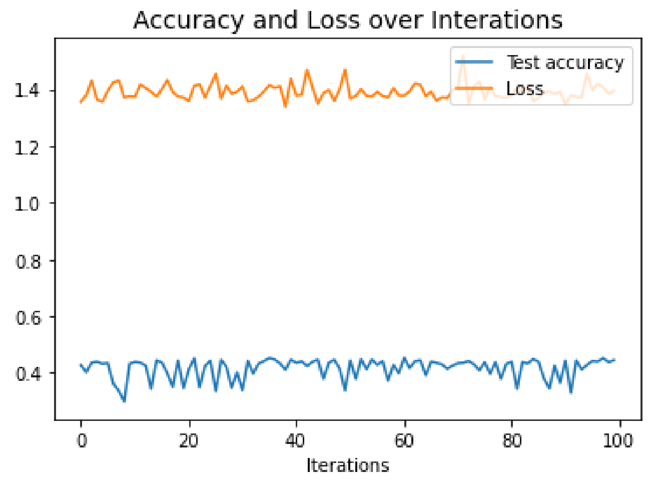
Answer 2: single hidden layer MLP with k= 25 neurons

The single layer network with 25 nerons in the single layer took several minutes to run and yielded a model with accuracy of approximately 0.45 across 7 target categories. The target categories were reached by "binning" the target variable, which was distributed across many values between 0 and 397, resulting initially in a very low accuracy model. I therefore decided to apply binning and split the variable in to 7 evenly split categories with range of 50 values, between 0 and 400. This was achieve as follows:

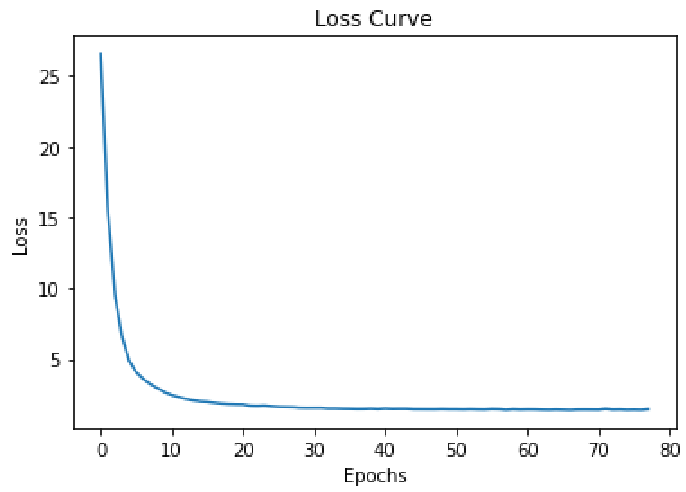
```
y = np.digitize(y,bins=[50,100,150,200,250,300,350])
```

Without bining, the accuracy of the model dropped to 0.01, with binning across 4 categories, the accuracy of the model increases to about 0.6. & seemed like an acceptable compromise.

Details of the model are provided below, along with charts and metrics.



```
Loss 1.4461270468227754
iterations 78
Assigned classes [0 1 2 3 4 5 6 7]
```



Accuracy score: 0.44

Answer 3: two hidden layer MLP with with k= 24 neurons (max)

The following table provides a summary of the result of running the algorithm across 25 iterations of the process of incrementing the first layer of a 2 layer MLP by one and decrementing the second layer by one, with a starting value of 1 for the first layer and 24 for the second layer, until the values are swapped.

Combination of neurons	Accuracy
17,8	0.450226
6,19	0.446025
16,9	0.445055
11,14	0.443439
19,6	0.441823
9,16	0.4415
10,15	0.441176
1,24	0.43956
7,18	0.43956
14,11	0.43245
13,12	0.430834
8,17	0.425339
22,3	0.423077
18,7	0.420491
21,4	0.419522
23,2	0.419198
24,1	0.419198
3,22	0.418875
20,5	0.417582
12,13	0.413704
15,10	0.411441
4,21	0.410472
2,23	0.389787
5,20	0.378474

Answer 4: variation in the obtained performance metrics

As can be seen from the table provided in the (3) part of this section, although there is variation in the results, no obvious pattern has emerged – i.e. there is no marked difference based on the number of perceptrons in the hidden layers being about the same or being very different, the result seems quite random. The two lowest results where where the first hidden layer had low number of perceptrons, but also one of the highest results was where the first layer has 1 perceptron (and the second layer had 24).

It is difficult to say which architecture gives the better performance, or to explain the results, as no pattern has emerged from this experiment and therefore I am unable to make any assumptions about what might (or might not) be happening. What I can say though is that since the pattern is randm, perhaps for this dataset a deeper network is onconsequential.

Long Short-Term Memory (LSTM)

asdf

(1)

DNC

(2)

DNC

(3)

DNC

(4)

DNC

Model Comparison

DNC

(1)

DNC

(2)

DNC

References

[1] Meng Z, Hu Y, Ancey C. "Using a Data Driven Approach to Predict Waves Generated by Gravity Driven Mass Flows." *Water*. 2020; 12(2):600. https://doi.org/10.3390/w12020600

Appendix

Four appendixes are provided

19075153_O'Leary_PartA1_AM: extract from Jupyter-Labs for answers to Part A (Agglomerative)

19075153_O'Leary_PartA1_DB: extract from Jupyter-Labs for answers to Part A (DBSCAN)

19075153_O'Leary_PartA1_KM: extract from Jupyter-Labs for answers to Part A (Kmeans)

19075153_O'Leary_PartB: extract from Jupyter-Labs for answers to Part B

19075153_O'Leary_PartB

October 24, 2021

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import warnings
from pandas.plotting import scatter_matrix
import seaborn as sns
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import tree
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from numpy import set_printoptions
from sklearn.decomposition import PCA
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RANSACRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from scipy.stats import pearsonr
import sklearn.metrics as metrics

warnings.filterwarnings('ignore')
```

1 Data load and pre-processing

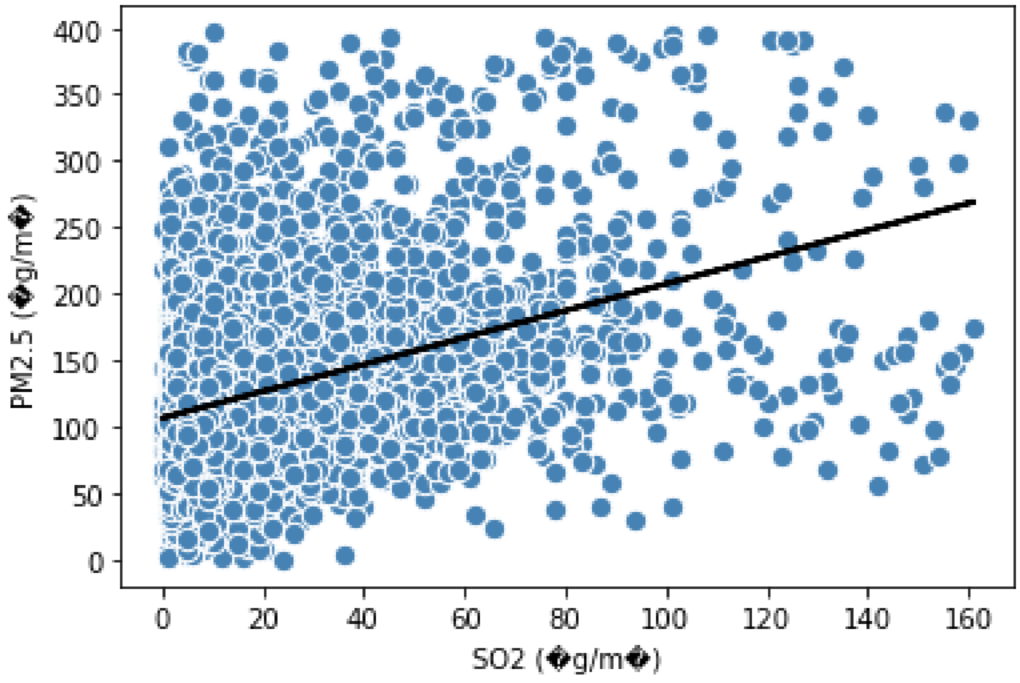
[2]:
```
min_max_scaler = preprocessing.MinMaxScaler()

#################
# load Penrose data
#################
path = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/
↪Penrose_Hourly_AggregateData_Jan2016Dec2020.csv"
rawdata = pd.read_csv(path)
```

```
# Rearrange Timestamp column
# Remove incorrect values or outliers
rawdata = rawdata[rawdata['Relative Humidity (%)'] <= 100]
rawdata = rawdata[rawdata['Air Temp ( C)'] <= 40]
rawdata = rawdata.dropna(how='any',axis=0)
rawdata['Timestamp (UTC+12:00)'] = pd.to_datetime(rawdata['Timestamp (UTC+12:
↪00)'])
rawdata = rawdata.set_index('Timestamp (UTC+12:00)')
rawdata['Hour'] = rawdata.index.hour
rawdata['Day'] = rawdata.index.day
rawdata['Month'] = rawdata.index.month
rawdata['Year'] = rawdata.index.year

# categorise everything and create array
list_of_columns = rawdata.columns
rawdata[list_of_columns] = rawdata[list_of_columns].apply(lambda col:pd.
↪Categorical(col).codes)
```

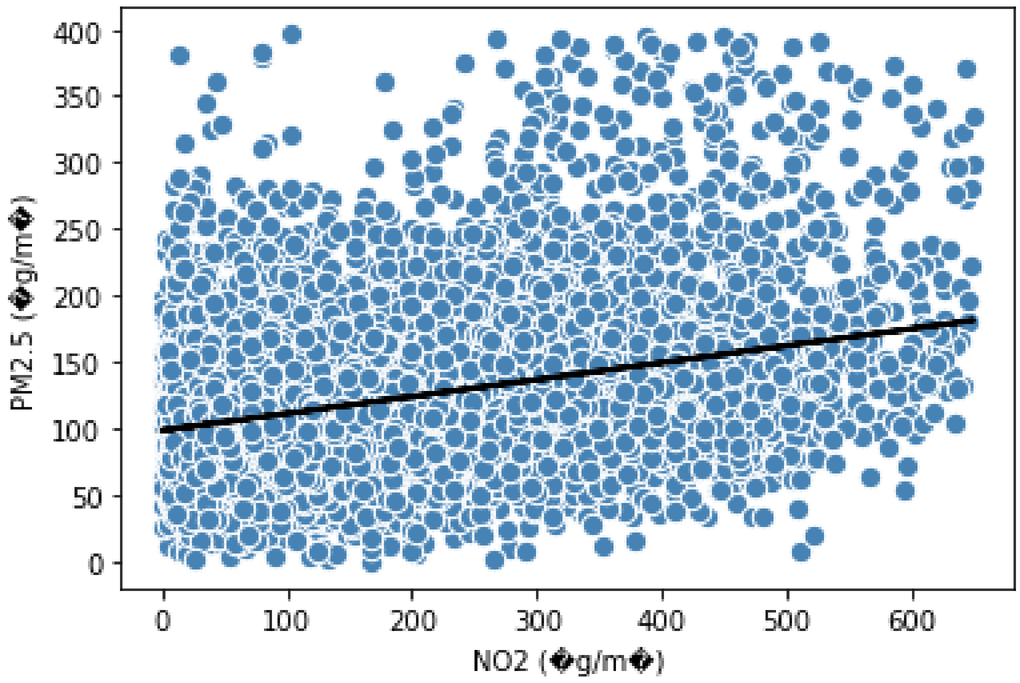
2 Inspecting outliers

[3]:
```
# A function that will plot a scatterplot of the training samples and add the␣
↪regression line
def lin_reg_plot(x_name):

    # Create the model
    slr = LinearRegression()
    slr.fit(X, y)

    # Regression metrics
    explained_variance=metrics.explained_variance_score(y, slr.predict(X))
    mean_absolute_error=metrics.mean_absolute_error(y, slr.predict(X))
    mse=metrics.mean_squared_error(y, slr.predict(X))
    median_absolute_error=metrics.median_absolute_error(y, slr.predict(X))
    r2=metrics.r2_score(y, slr.predict(X))
    corr, _ = pearsonr(rawdata[x_name], y)

    # Print stats
    print('==================================== ' + x_name + '␣
↪====================================')
    print('Slope: %.3f' % slr.coef_[0])
    print('Intercept: %.3f' % slr.intercept_)
    print('explained_variance: ', round(explained_variance,4))
    print('r2: ', round(r2,4))
    print('MAE: ', round(mean_absolute_error,4))
    print('MSE: ', round(mse,4))
    print('RMSE: ', round(np.sqrt(mse),4))
    print('Pearsons correlation: ', round(corr,4)) # Over 0.5 or less than -0.5␣
 ↪signifies strongest correlation

    # Chart
    plt.scatter(X, y, c='steelblue', edgecolor='white', s=70)
    plt.plot(X, slr.predict(X), color='black', lw=2)
    plt.xlabel(x_name)
    plt.ylabel('PM2.5 ( g/m )')
    plt.show()

# Starting point X and y
X = rawdata[['Hour','Air Temp ( C)','Relative Humidity (%)','Solar Radiation (W/
 ↪m )','Wind Direction ( )','Wind Speed (m/s)','NO ( g/m )','NO2 ( g/m )','SO2 ( g/
 ↪m )']].values
y = rawdata['PM2.5 ( g/m )'].values

# Display data in 2D
x_name = 'SO2 ( g/m )'
X = rawdata[[x_name]].values
lin_reg_plot(x_name)
x_name = 'NO2 ( g/m )'
X = rawdata[[x_name]].values
lin_reg_plot(x_name)
x_name = 'NO ( g/m )'
X = rawdata[[x_name]].values
lin_reg_plot(x_name)
x_name = 'Wind Speed (m/s)'
X = rawdata[[x_name]].values
lin_reg_plot(x_name)
x_name = 'Wind Direction ( )'
X = rawdata[[x_name]].values
lin_reg_plot(x_name)
x_name = 'Solar Radiation (W/m )'
X = rawdata[[x_name]].values
lin_reg_plot(x_name)
x_name = 'Relative Humidity (%)'
X = rawdata[[x_name]].values
lin_reg_plot(x_name)
x_name = 'Air Temp ( C)'
X = rawdata[[x_name]].values
lin_reg_plot(x_name)
x_name = 'Hour'
X = rawdata[[x_name]].values
lin_reg_plot(x_name)
```

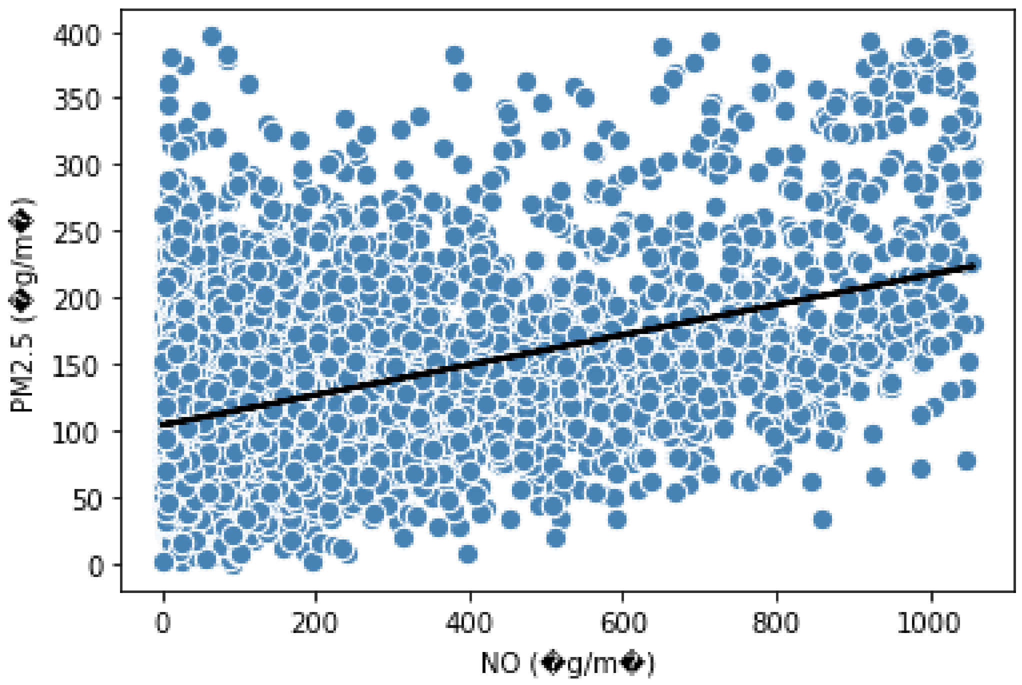
```
==================================== SO2 ( g/m )
====================================
Slope: 1.011
Intercept: 106.215
explained_variance:  0.1236
r2:  0.1236
MAE:  36.4505
MSE:  2458.9511
RMSE:  49.5878
Pearsons correlation:  0.3515
```



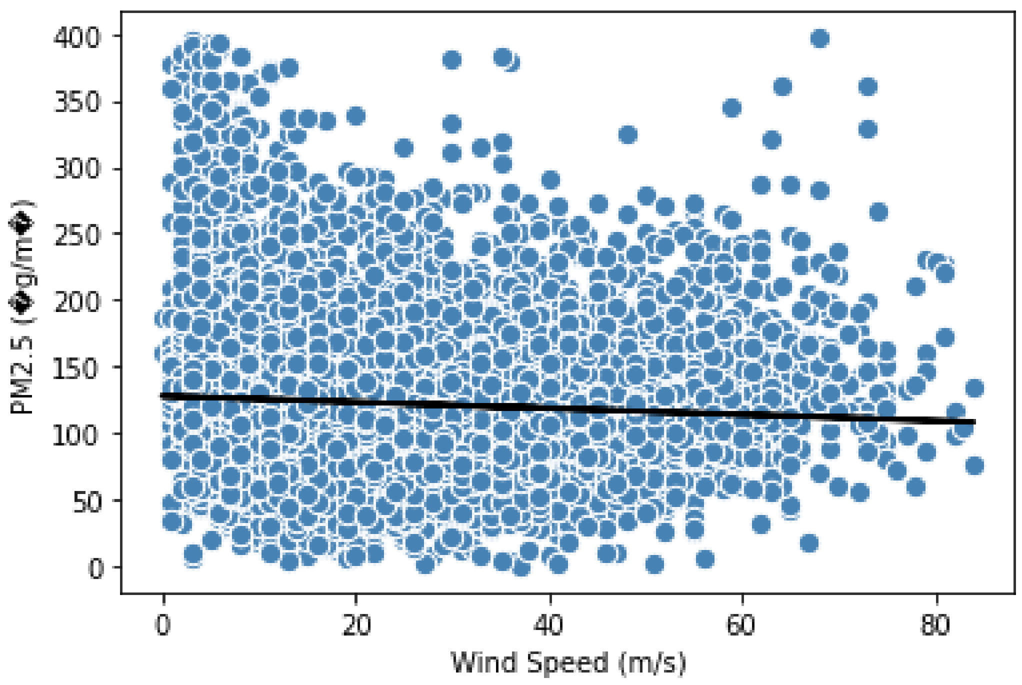
```
==================================== NO2 ( g/m )
====================================
Slope: 0.128
Intercept: 98.190
explained_variance:  0.1137
r2:  0.1137
MAE:  36.438
MSE:  2486.5994
RMSE:  49.8658
Pearsons correlation:  0.3372
```



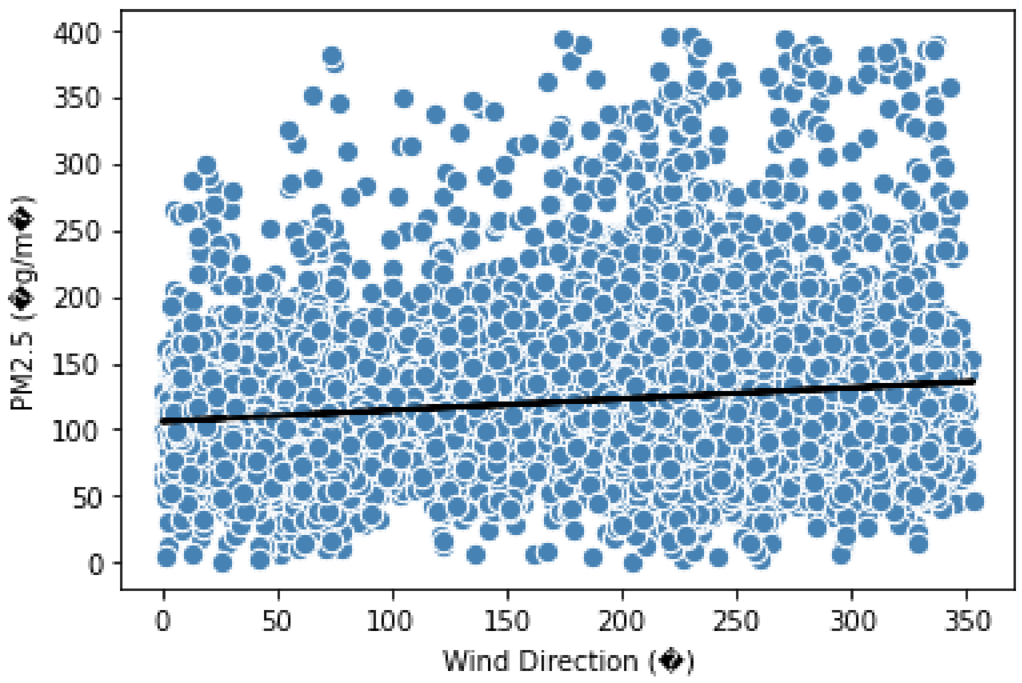
```
==================================== NO ( g/m )
====================================
Slope: 0.113
Intercept: 103.757
explained_variance:  0.1904
r2:  0.1904
MAE:  35.2568
MSE:  2271.5162
RMSE:  47.6604
Pearsons correlation:  0.4363
```



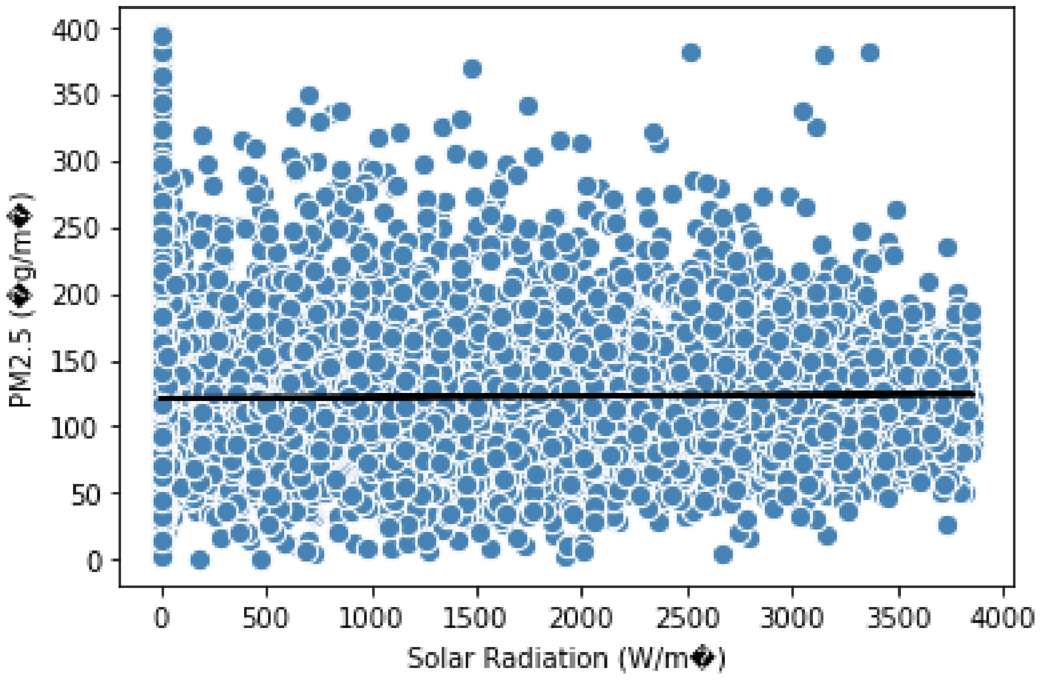
```
==================================== Wind Speed (m/s)
====================================
Slope: -0.233
Intercept: 127.722
explained_variance:  0.0045
r2:  0.0045
MAE:  38.6166
MSE:  2792.8766
RMSE:  52.8477
Pearsons correlation:  -0.0673
```



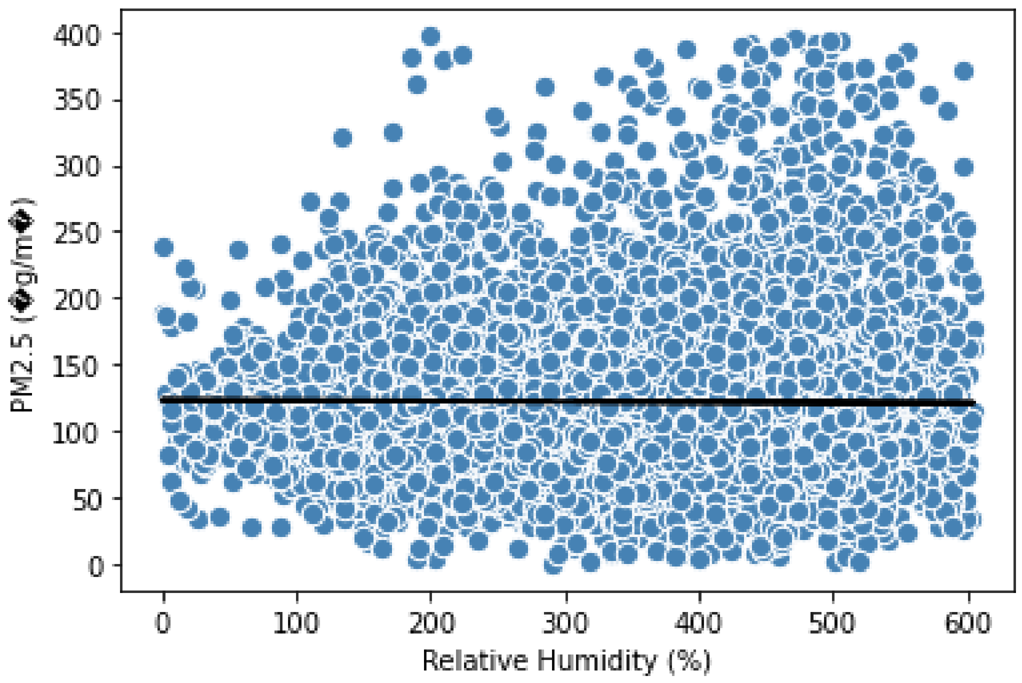
```
==================================== Wind Direction ( )
====================================
Slope: 0.085
Intercept: 106.028
explained_variance:  0.0211
r2:  0.0211
MAE:  38.016
MSE:  2746.4533
RMSE:  52.4066
Pearsons correlation:  0.1452
```



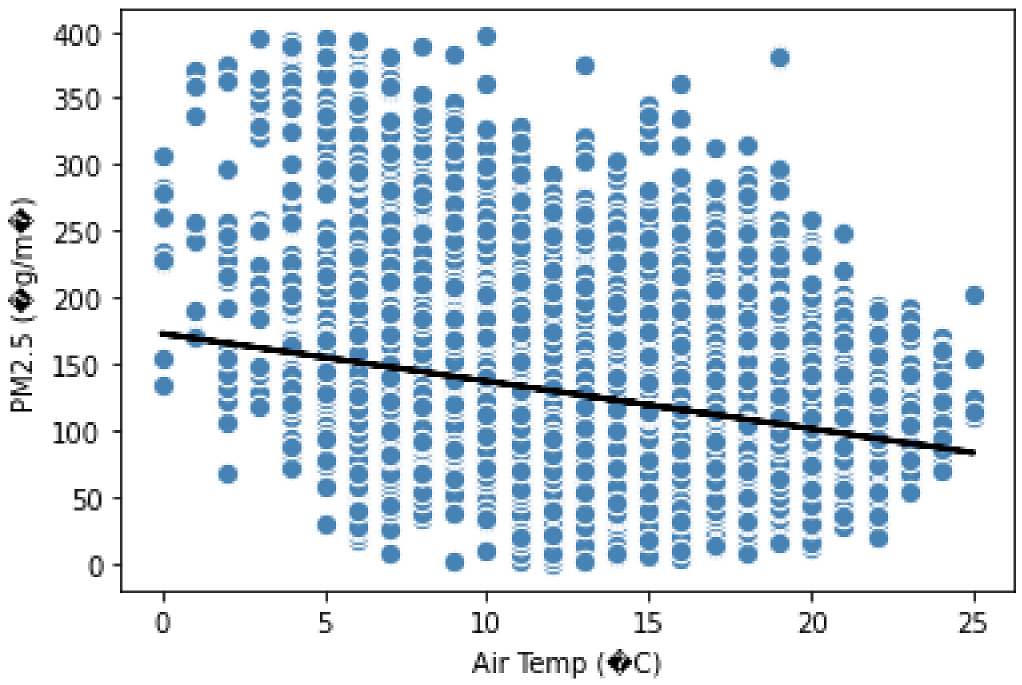
```
===================================== Solar Radiation (W/m )
=====================================
Slope: 0.001
Intercept: 121.037
explained_variance:  0.0004
r2:  0.0004
MAE:  38.47
MSE:  2804.466
RMSE:  52.9572
Pearsons correlation:  0.02
```



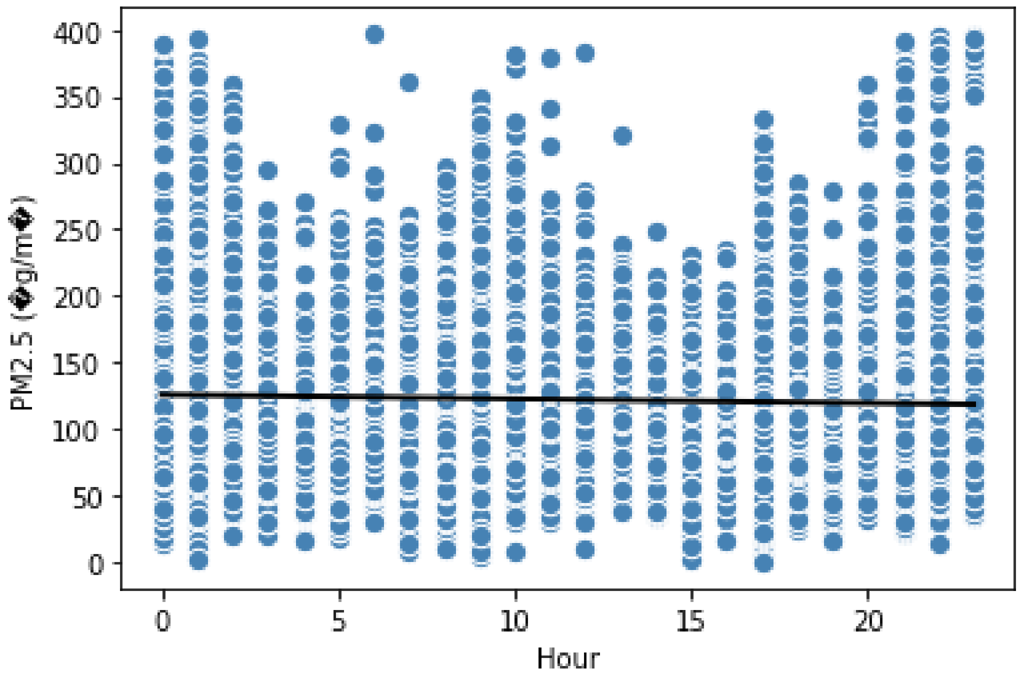
```
==================================== Relative Humidity (%)
====================================
Slope: -0.003
Intercept: 123.050
explained_variance:  0.0001
r2:  0.0001
MAE:  38.4961
MSE:  2805.3804
RMSE:  52.9658
Pearsons correlation:  -0.0086
```



```
==================================== Air Temp ( C)
====================================
Slope: -3.581
Intercept: 172.655
explained_variance:  0.0722
r2:  0.0722
MAE:  37.9256
MSE:  2603.0704
RMSE:  51.0203
Pearsons correlation:  -0.2687
```

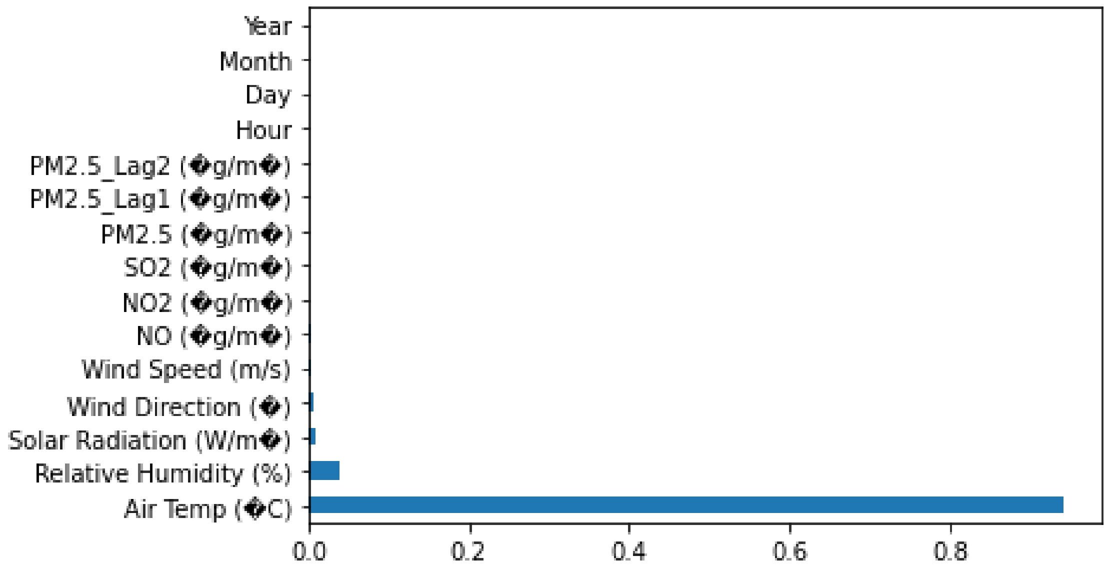


```
==================================== Hour ====================================
Slope: -0.318
Intercept: 125.667
explained_variance:  0.0017
r2:  0.0017
MAE:  38.4984
MSE:  2800.7835
RMSE:  52.9224
Pearsons correlation:  -0.0414
```



3 Feature importance

[4]:
```
# Run PCA
pca = PCA()
pca_fit = pca.fit(rawdata)
# summarize components
feat_importances = pd.Series(pca_fit.explained_variance_ratio_, index=rawdata.
 ↪columns)
feat_importances.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca = PCA(n_components=2)
pca_fit = pca.fit(rawdata)
```



4 Summary stats for PM2.5 (g/m)

[5]:
```
rawdata['PM2.5 ( g/m )'].describe()
```

[5]:
```
count    10313.000000
mean       121.846310
std         52.970365
min          0.000000
25%         88.000000
50%        113.000000
75%        145.000000
max        397.000000
Name: PM2.5 ( g/m ), dtype: float64
```

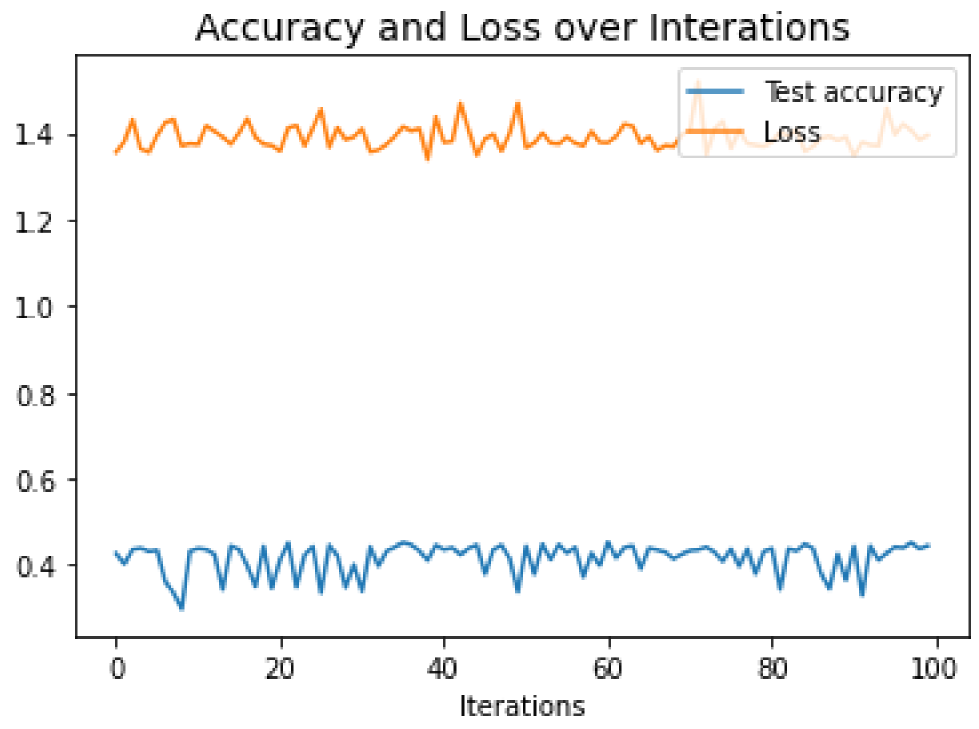
5 Do the train/test split

[9]:
```
# Get final X and y - we want to keep the top five attributes based on Pearsons␣
 ↪correlation
# Five topmost correlated varaiables are 'Air Temp ( C)','Wind Direction␣
 ↪( )','NO ( g/m )','NO2 ( g/m )','SO2 ( g/m )'
X = rawdata[['Air Temp ( C)','Wind Direction ( )','NO ( g/m )','NO2 ( g/m )','SO2␣
 ↪( g/m )']].values
y = rawdata['PM2.5 ( g/m )'].values
```

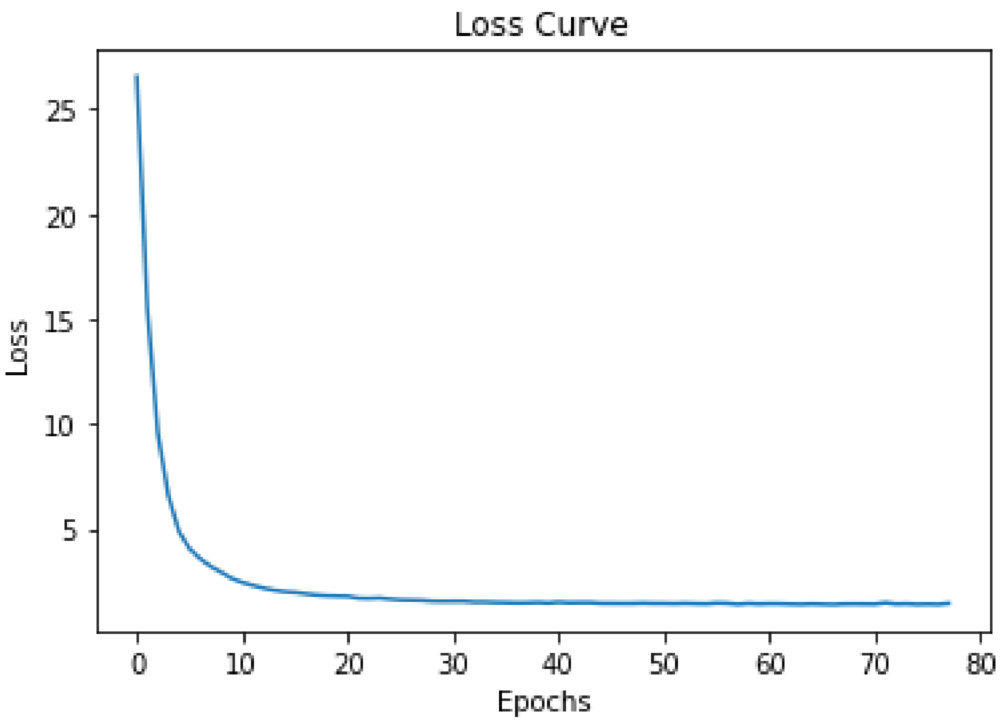
```
# We need to reduce the number of categories for y - max is 397, we can␣
↪probably start with 7 categories
# In this case we get about 0.45 accuracy score
# If we reduce the categories to 4, we get about 0.6 accuracy score
y = np.digitize(y,bins=[50,100,150,200,250,300,350])

# Do the train/test split
pred_train, pred_test, tar_train, tar_test = train_test_split(X, y, test_size=.
↪3, random_state=4)
```

6 MLP classifier

[10]:
```
import pandas
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt

#A function to see some of the attributes of NN
def NN_properties(model):
    loss_values = model.loss_
    print("Loss", loss_values)
    iterations = model.n_iter_
    print("iterations",iterations)
    classes_assigned= model.classes_
    print("Assigned classes", classes_assigned)

#Displaying loss curve using loss_curve method.Note that this only works with␣
↪the MLP default solver "adam"
def make_plots_default(model):
    plt.plot(model.loss_curve_)
    plt.title('Loss Curve')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.show()

#A generic function to display training loss and testing accuracy of␣
↪MLPClassifier
def make_plots_all(mlp, target_train, target_test,␣
↪predictors_test,predictors_train):
    max_iter = 100
    accuracy = []
    losses = []
    for i in range(max_iter):
        mlp.fit(predictors_train, target_train)
        iter_acc = mlp.score(predictors_test, target_test)
        accuracy.append(iter_acc)
        losses.append(mlp.loss_)
    plt.plot(accuracy, label='Test accuracy')
    plt.plot(losses, label='Loss')
    plt.title("Accuracy and Loss over Interations", fontsize=14)
    plt.xlabel('Iterations')
    plt.legend(loc='upper right')
    plt.show()

#A function for model building and calculating accuracy
def get_accuracy(target_train, target_test, predictors_test, predictors_train):
    # Two hidden layers with 10 and 5 neurons - NN
    clf = MLPClassifier(hidden_layer_sizes=(25), max_iter=100)
    #Calling the make_plots_allfunction with unfitted model
    make_plots_all(clf, target_train, target_test, predictors_test,␣
 ↪predictors_train)
    clf.fit(predictors_train, np.ravel(target_train, order='C'))
    predictions = clf.predict(predictors_test)
    NN_properties(clf) ##Calling NN_properties to see the model attributes
    make_plots_default(clf) ##Calling make_plots function to see the error plots
    return accuracy_score(target_test, predictions)

#train-test split
pred_train, pred_test, tar_train, tar_test = train_test_split(X, y, test_size=.
 ↪3, random_state=4)

#Calling get_accuracy function which also invoke other functions NN_properties,␣
 ↪make_plots, make_plots_all
print("Accuracy score: %.2f" % get_accuracy(tar_train, tar_test, pred_test,␣
 ↪pred_train))
```



```
Loss 1.4461270468227754
iterations 78
Assigned classes [0 1 2 3 4 5 6 7]
```



```
Accuracy score: 0.44
```

7 2-Layer MLP

[11]:
```
#train-test split
pred_train, pred_test, tar_train, tar_test = train_test_split(X, y, test_size=.
↪3, random_state=4)

def two_layer_mlp(num_l1):
    tlmlp = MLPClassifier(hidden_layer_sizes=(num_l1, 25-num_l1,), max_iter=84)
    fit_tlmlp = tlmlp.fit(pred_train, np.ravel(tar_train, order='C'))
    pred_tlmlp = tlmlp.predict(pred_test)
    prob_tlmlp = tlmlp.predict_proba(pred_test)
    accuracy_tlmlp = accuracy_score(tar_test, pred_tlmlp)
    return accuracy_tlmlp

results = pd.DataFrame(columns=["Combination of neurons", "Accuracy"])
for i in range(1,25):
    new_row = {"Combination of neurons": str(i)+","+str(25-i), "Accuracy":␣
↪str(two_layer_mlp(i))}
    results = results.append(new_row, ignore_index=True)

# Drop results out to a table
```

```
results
```

```
[11]:    Combination of neurons              Accuracy
     0                     1,24  0.43956043956043955
     1                     2,23  0.38978668390433097
     2                     3,22   0.4188752424046542
     3                     4,21  0.41047188106011634
     4                     5,20   0.3784744667097608
     5                     6,19   0.4460245636716225
     6                     7,18  0.43956043956043955
     7                     8,17   0.4253393665158371
     8                     9,16   0.4414996767937944
     9                    10,15   0.4411764705882353
     10                   11,14   0.4434389140271493
     11                   12,13   0.4137039431157078
     12                   13,12   0.4308338720103426
     13                   14,11  0.43244990303813835
     14                   15,10   0.4114414996767938
     15                    16,9  0.44505494505494503
     16                    17,8   0.4502262443438914
     17                    18,7   0.4204912734324499
     18                    19,6  0.44182288299935357
     19                    20,5   0.4175824175824176
     20                    21,4  0.41952165481577247
     21                    22,3   0.4230769230769231
     22                    23,2   0.4191984486102133
     23                    24,1   0.4191984486102133
```

```
[ ]:
```

19075153_O'Leary_PartA1_KM

October 16, 2021

[1]:
```
import pandas
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import warnings
from pandas.plotting import scatter_matrix
import seaborn as sns
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import tree
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from numpy import set_printoptions
from sklearn.decomposition import PCA
from sklearn import preprocessing

warnings.filterwarnings('ignore')
```

1 Data load and pre-processing

[2]:
```
min_max_scaler = preprocessing.MinMaxScaler()

#################
# load data sales
#################
path_sales = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/
 ↪Sales_Transactions_Dataset_Weekly.csv"
rawdata_sales = pandas.read_csv(path_sales)
# categorise everything and create array
list_of_columns_sales = rawdata_sales.columns
rawdata_sales[list_of_columns_sales] = rawdata_sales[list_of_columns_sales].
 ↪apply(lambda col:pandas.Categorical(col).codes)
# Create array
array_sales = rawdata_sales.values
predictors_sales = array_sales[:, 0:107]
# Print some stats
```

```
print(rawdata_sales.shape)
print(rawdata_sales.head())

#################
# load data seoul
#################
path_seoul = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/SeoulBikeData.csv"
rawdata_seoul = pandas.read_csv(path_seoul)
# categorise everything and create array
list_of_columns_seoul = rawdata_seoul.columns
rawdata_seoul[list_of_columns_seoul] = rawdata_seoul[list_of_columns_seoul].
 ↪apply(lambda col:pandas.Categorical(col).codes)
# Create array
array_seoul = rawdata_seoul.values
predictors_seoul = array_seoul[:, 0:14]
# Print some stats
print(rawdata_seoul.shape)
print(rawdata_seoul.head())

#################
# load data water
#################
path_water = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/water-treatment.data"
rawdata_water = pandas.read_csv(path_water)
rawdata_water.columns =␣
 ↪['DATE','Q-E','ZN-E','PH-E','DBO-E','DQO-E','SS-E','SSV-E','SED-E','COND-E','PH-P','DBO-P',
# categorise everything and create array
list_of_columns_water = rawdata_water.columns
rawdata_water[list_of_columns_water] = rawdata_water[list_of_columns_water].
 ↪apply(lambda col:pandas.Categorical(col).codes)
# Create array
array_water = rawdata_water.values
predictors_water = array_water[:, 0:39]
# Print some stats
print(rawdata_water.shape)
print(rawdata_water.head())
```

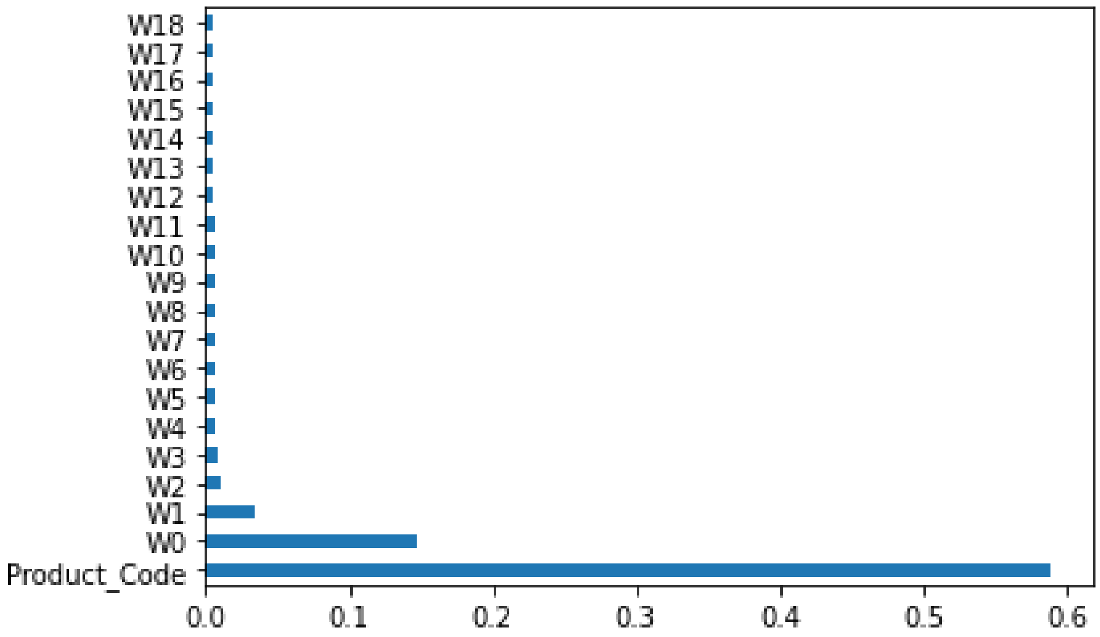
```
(811, 107)
   Product_Code  W0  W1  W2  W3  W4  W5  W6  W7  W8  …  Normalized 42  \
0             0  11  12  10   8  13  12  14  21   6  …              3
1           111   7   6   3   2   7   1   6   3   3  …             17
2           222   7  11   8   9  10   8   7  13  12  …             24
3           331  12   8  13   5   9   6   9  13  13  …             37
4           442   8   5  13  11   6   7   9  14   9  …             24

   Normalized 43  Normalized 44  Normalized 45  Normalized 46  Normalized 47  \
0             20             26             35             46              0
```

```
1            38           47           7           6          35
2            83           16          15          32          40
3            45            4           9          20          30
4            51           25          56          16          15

   Normalized 48  Normalized 49  Normalized 50  Normalized 51
0             16             13              7             35
1             44              7             55              0
2             82             41             41             32
3             65             31             25             31
4              8             49             29             36

[5 rows x 107 columns]
(8760, 14)
   Date  Rented Bike Count  Hour  Temperature( C)  Humidity(%)  \
0    11                253     0              111           28
1    11                203     1              108           29
2    11                172     2              103           30
3    11                106     3              101           31
4    11                 77     4              103           27

   Wind speed (m/s)  Visibility (10m)  Dew point temperature( C)  \
0                22              1788                        114
1                 8              1788                        114
2                10              1788                        113
3                 9              1788                        114
4                23              1788                        104

   Solar Radiation (MJ/m2)  Rainfall(mm)  Snowfall (cm)  Seasons  Holiday  \
0                        0             0              0        3        1
1                        0             0              0        3        1
2                        0             0              0        3        1
3                        0             0              0        3        1
4                        0             0              0        3        1

   Functioning Day
0                1
1                1
2                1
3                1
4                1
(526, 39)
   DATE  Q-E  ZN-E  PH-E  DBO-E  DQO-E  SS-E  SSV-E  SED-E  COND-E  …  \
0   197  330   116     6    204    169    57    201     50     409  …
1   427   99   143     5    204    231    42    208     28     303  …
2   443  219   126     8     93    256    45    171     36     402  …
3   461  282    66     9    126    211    37    164     33     381  …
4   479  319   116     7     90    117    42    197     36     295  …
```

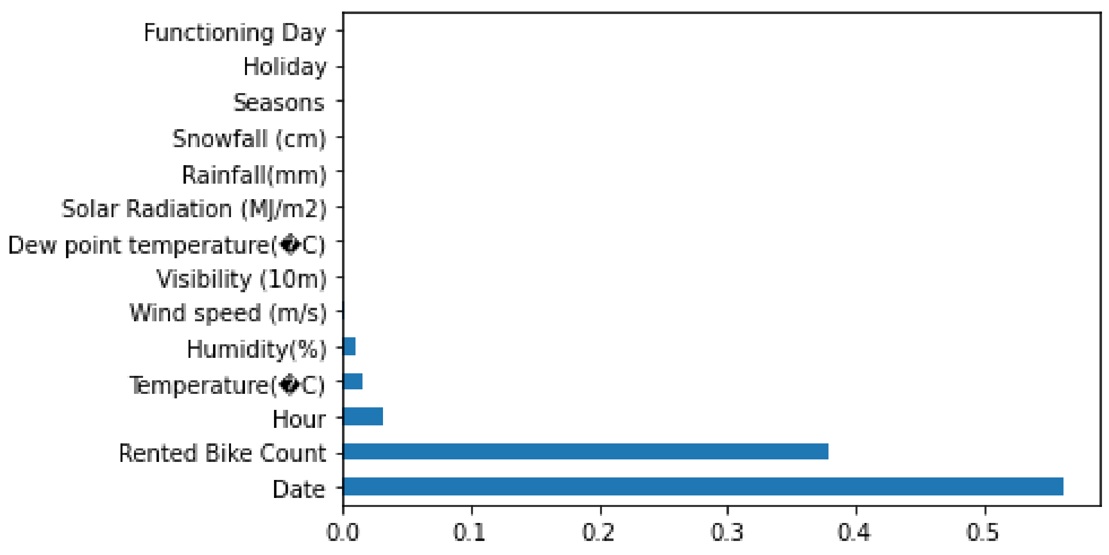
```
   COND-S  RD-DBO-P  RD-SS-P  RD-SED-P  RD-DBO-S  RD-DQO-S  RD-DBO-G  \
0     372       314      165       104       184       233       155
1     334       314      143       111       184        37       155
2     322       100      196       108       131       165        95
3     349       314      183       111       184       153       114
4     301       314      157       118       125       216        94

   RD-DQO-G  RD-SS-G  RD-SED-G
0       134      126         0
1       101       92        26
2       158      101         0
3       121       82        37
4        78       61         0

[5 rows x 39 columns]
```

2 Feature importance

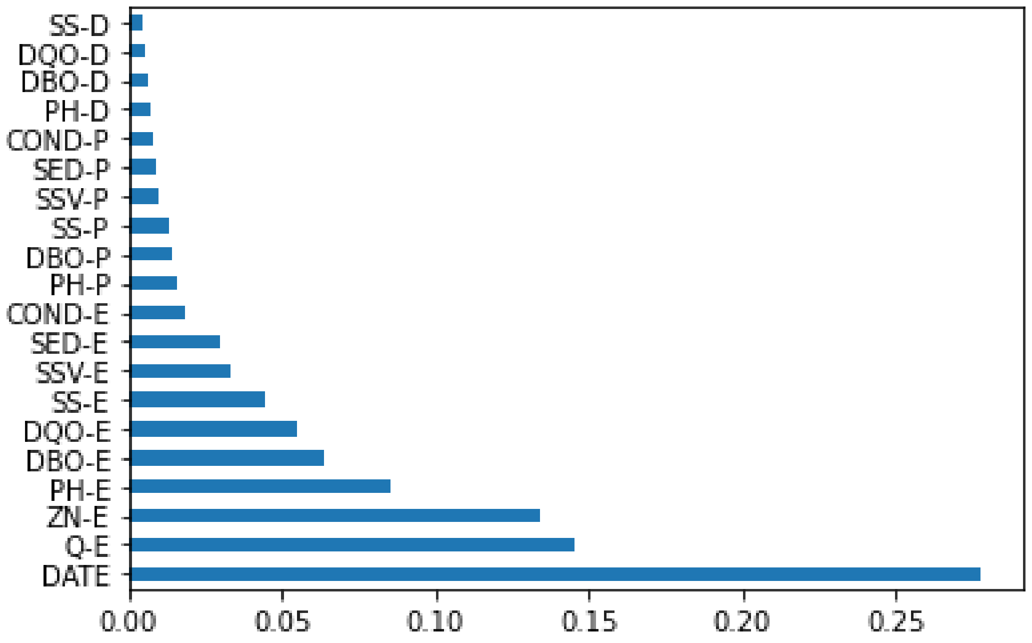
[3]:
```
#################
# Sales
#################
# Run PCA
pca_sales = PCA()
pca_fit_sales = pca_sales.fit(predictors_sales)
# summarize components
feat_importances_sales = pandas.Series(pca_fit_sales.explained_variance_ratio_,␣
 ↪index=rawdata_sales.columns)
feat_importances_sales.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca_sales = PCA(n_components=2)
pca_fit_sales = pca_sales.fit(predictors_sales)
pca_data_sales = rawdata_sales[['W0','W1','Product_Code']]
```



[4]:
```
#################
# Seoul
#################
# Run PCA
pca_seoul = PCA()
pca_fit_seoul = pca_seoul.fit(predictors_seoul)
# summarize components
feat_importances_seoul = pandas.Series(pca_fit_seoul.explained_variance_ratio_,␣
↪index=rawdata_seoul.columns)
feat_importances_seoul.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca_seoul = PCA(n_components=2)
pca_fit_seoul = pca_seoul.fit(predictors_seoul)
pca_data_seoul = rawdata_seoul[['Hour','Rented Bike␣
↪Count','Date','Temperature( C)']]
```



[5]:
```
#################
# Water
#################
# Run PCA
pca_water = PCA()
pca_fit_water = pca_water.fit(predictors_water)
# summarize components
feat_importances_water = pandas.Series(pca_fit_water.explained_variance_ratio_,␣
 ↪index=rawdata_water.columns)
feat_importances_water.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca_water = PCA(n_components=2)
pca_fit_water = pca_water.fit(predictors_water)
pca_data_water = rawdata_water[['DATE','Q-E','ZN-E','PH-E','DBO-E']]
```



3 Clustering

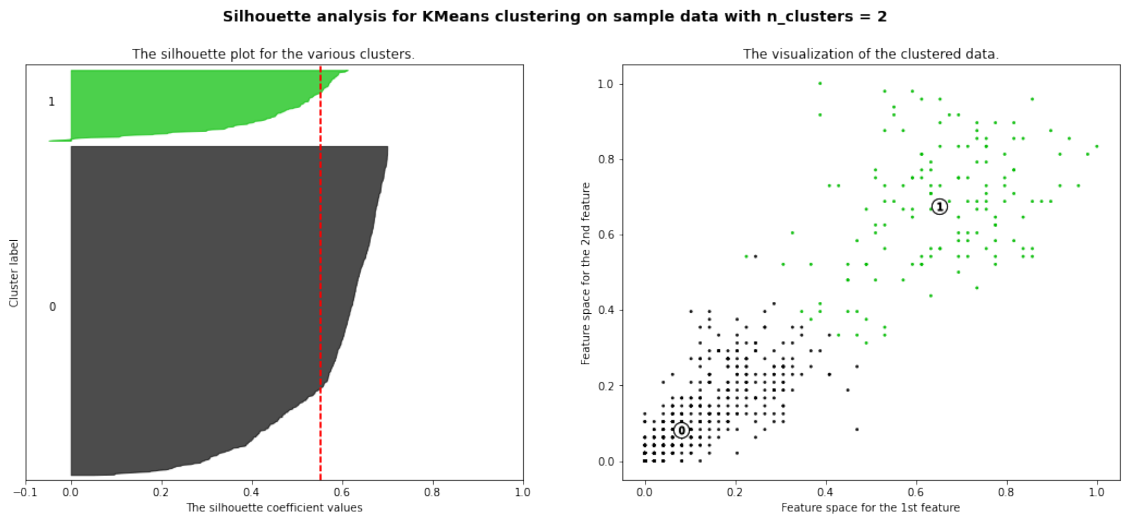
[6]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
import time


def do_sse(X, cluster_labels, n_clusters, model):
    cluster_centers = [X[cluster_labels == i].mean(axis=0) for i in␣
 ↪range(n_clusters)]
    clusterwise_sse = [0, 0, 0, 0, 0, 0]
    for point, label in zip(X, cluster_labels):
        clusterwise_sse[label] += np.square(point - cluster_centers[label]).
 ↪sum()
    clusterwise_sse_avg = np.mean(clusterwise_sse)
    return clusterwise_sse_avg

def do_cluster_analysis(name):
    # To find out the optimal number of clusters we can search through range of␣
  ↪clusters.
    range_n_clusters = [2, 3, 4, 5, 6]
    for n_clusters in range_n_clusters:

       ␣
  ↪print('================================================================================
        print('n_clusters = ', n_clusters)
       ␣
  ↪print('================================================================================
        start_time = time.time()

        # Create a subplot with 1 row and 2 columns
        fig, (ax1, ax2) = plt.subplots(1, 2)
        fig.set_size_inches(18, 7)

        # The 1st subplot is the silhouette plot
        # The silhouette coefficient can range from -1, 1
        # but in this example code all lie within [-0.1, 1]

        ax1.set_xlim([-0.1, 1])

        # # The (n_clusters+1)*10 is for inserting blank space between
        # silhouette plots of individual clusters, to demarcate them
        # clearly.

        ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

        #Apply k-means
        clusterer = KMeans(n_clusters, random_state=0)
        cluster_labels = clusterer.fit_predict(X)

        # The silhouette_score gives the average value for all the
        # samples.This gives a perspective into the density and
        # separation of the formed clusters

        silhouette_avg = silhouette_score(X, cluster_labels)
        #␣
  ↪/////////////////////////////////////////////////////////////////////////////////////
        # Print the values
        #␣
  ↪/////////////////////////////////////////////////////////////////////////////////////
        print("For n_clusters =", n_clusters, "The average silhouette_score is :
  ↪", silhouette_avg)
```

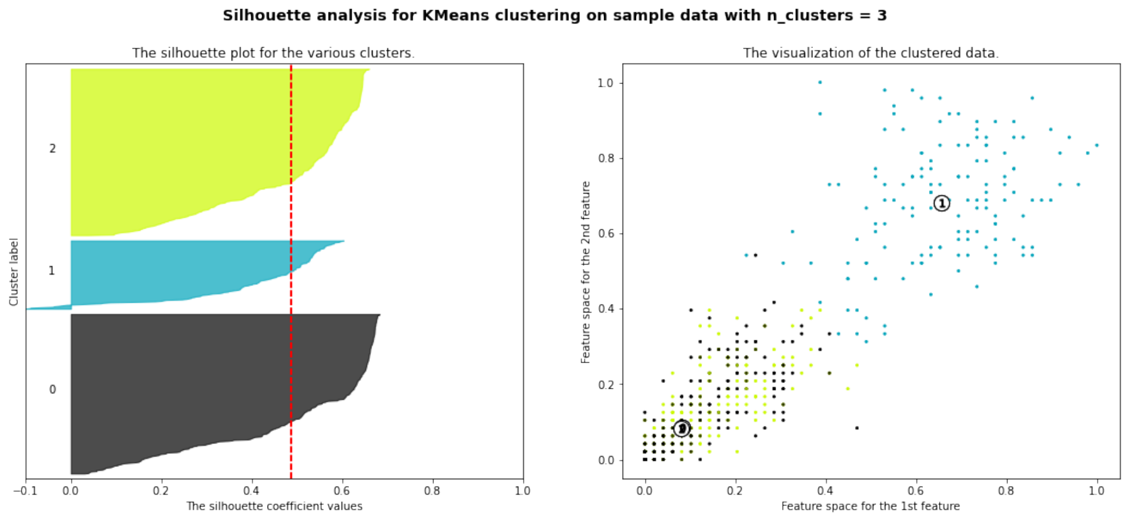
```
        print("For n_clusters =", n_clusters, "The average SSE is :", do_sse(X,␣
↪clusterer.labels_, n_clusters, clusterer))

        # Compute the silhouette scores for each sample
        sample_silhouette_values = silhouette_samples(X, cluster_labels)
        y_lower = 10

        for i in range(n_clusters):
            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them

            #␣
↪//////////////////////////////////////////////////////////////////////////////////////
            # Create the plot
            #␣
↪//////////////////////////////////////////////////////////////////////////////////////

            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them
            ith_cluster_silhouette_values =␣
↪sample_silhouette_values[cluster_labels == i]

            ith_cluster_silhouette_values.sort()
            size_cluster_i = ith_cluster_silhouette_values.shape[0]
            y_upper = y_lower + size_cluster_i
            color = cm.nipy_spectral(float(i) / n_clusters)

            ax1.fill_betweenx(np.arange(y_lower, y_upper),
                              0, ith_cluster_silhouette_values,
                              facecolor=color, edgecolor=color,
                              alpha=0.7)

            # Label the silhouette plots with their cluster numbers at the
            # middle

            ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

            # Compute the new y_lower for next plot

            y_lower = y_upper + 10  # 10 for the 0 samples
            ax1.set_title("The silhouette plot for the various clusters.")
            ax1.set_xlabel("The silhouette coefficient values")
            ax1.set_ylabel("Cluster label")

            # The vertical line for average silhouette score of all the
            # values
            ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
            ax1.set_yticks([])  # Clear the yaxis labels / ticks
            ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

            # 2nd Plot showing the actual clusters formed
            colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
            ax2.scatter(X[:, 0],
                        X[:, 1],
                        marker='.',
                        s=30,
                        lw=0,
                        alpha=0.7,
                        c=colors,
                        edgecolor='k')

            # Labeling the clusters by centers
            centers = clusterer.cluster_centers_

            # Draw white circles at cluster centers
            ax2.scatter(centers[:, 0],
                        centers[:, 1],
                        marker='o',
                        c="white",
                        alpha=1,
                        s=200,
                        edgecolor='k')

            for i, c in enumerate(centers):
                ax2.scatter(c[0],
                            c[1],
                            marker='$%d$' % i,
                            alpha=1,
                            s=50,
                            edgecolor='k')

            ax2.set_title("The visualization of the clustered data.")
            ax2.set_xlabel("Feature space for the 1st feature")
            ax2.set_ylabel("Feature space for the 2nd feature")
            plt.suptitle(("Silhouette analysis for KMeans clustering on sample␣
↪data with n_clusters = %d" % n_clusters),
                         fontsize=14,
                         fontweight='bold')

        # Time to run
        print("--- %s seconds ---" % (time.time() - start_time))

#################
```

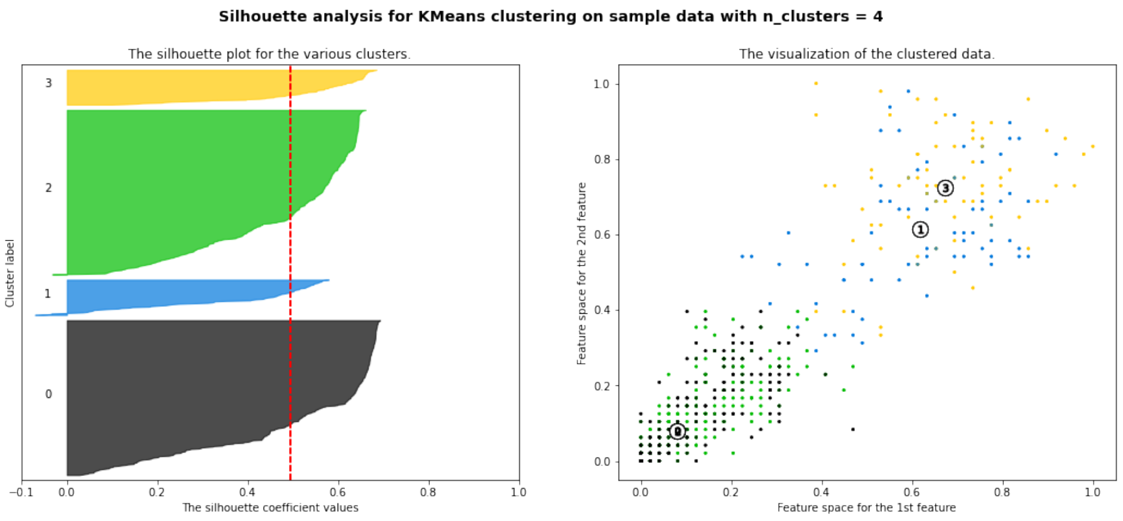
```
# Sales
#################
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('sales')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
X = pca_data_sales
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
X = x_scaled
do_cluster_analysis('sales')

#################
# Water
#################
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('water')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
X = pca_data_water
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
X = x_scaled
do_cluster_analysis('water')

#################
# Seoul
#################
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('seoul')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
X = pca_data_seoul
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
X = x_scaled
do_cluster_analysis('seoul')
```

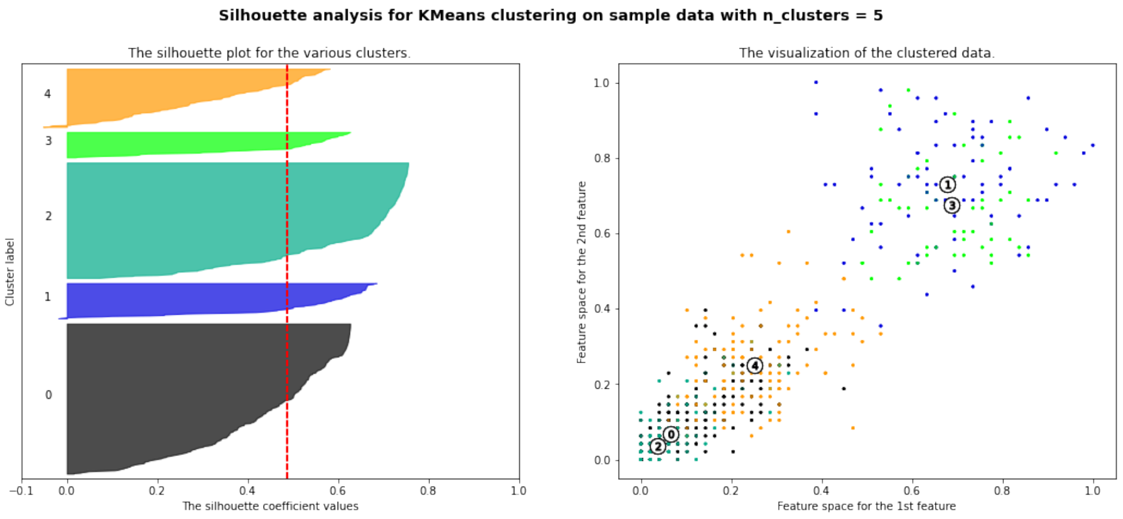
```
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
sales
|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
================================================================================
====
n_clusters =  2
================================================================================
====
For n_clusters = 2 The average silhouette_score is : 0.5529922804905653
For n_clusters = 2 The average SSE is : 14.437368116537462
--- 0.39674830436706543 seconds ---
================================================================================
====
n_clusters =  3
================================================================================
====
For n_clusters = 3 The average silhouette_score is : 0.488051563750343
For n_clusters = 3 The average SSE is : 8.004891084595682
--- 0.3482513427734375 seconds ---
================================================================================
====
n_clusters =  4
================================================================================
====
For n_clusters = 4 The average silhouette_score is : 0.4956690385768799
For n_clusters = 4 The average SSE is : 6.000403406587895
--- 0.36387038230895996 seconds ---
================================================================================
====
n_clusters =  5
================================================================================
====
For n_clusters = 5 The average silhouette_score is : 0.48706090844597216
For n_clusters = 5 The average SSE is : 4.577621604668537
--- 0.3974940776824951 seconds ---
================================================================================
====
n_clusters =  6
================================================================================
====
For n_clusters = 6 The average silhouette_score is : 0.4905602367765982
For n_clusters = 6 The average SSE is : 3.540858712458992
--- 0.4951903820037842 seconds ---
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
water
```

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
================================================================================
====
n_clusters =  2
================================================================================
====
For n_clusters = 2 The average silhouette_score is : 0.19182821192769825
For n_clusters = 2 The average SSE is : 23.774174486620684
--- 0.2271566390991211 seconds ---
================================================================================
====
n_clusters =  3
================================================================================
====
For n_clusters = 3 The average silhouette_score is : 0.18095097792392195
For n_clusters = 3 The average SSE is : 20.271440855256856
--- 0.2882091999053955 seconds ---
================================================================================
====
n_clusters =  4
================================================================================
====
For n_clusters = 4 The average silhouette_score is : 0.1962198321867441
For n_clusters = 4 The average SSE is : 17.68019667536556
--- 0.3564331531524658 seconds ---
================================================================================
====
n_clusters =  5
================================================================================
====
For n_clusters = 5 The average silhouette_score is : 0.19376529853466307
For n_clusters = 5 The average SSE is : 15.661026408267285
--- 0.5259983539581299 seconds ---
================================================================================
====
n_clusters =  6
================================================================================
====
For n_clusters = 6 The average silhouette_score is : 0.1979742286699438
For n_clusters = 6 The average SSE is : 14.219162585579324
--- 0.42758846282958984 seconds ---
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
seoul
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
================================================================================
====
n_clusters =  2
================================================================================
====
For n_clusters = 2 The average silhouette_score is : 0.2736422972611256
For n_clusters = 2 The average SSE is : 297.7696053888172
--- 5.6746766567230225 seconds ---
================================================================================
====
n_clusters =  3
================================================================================
====
For n_clusters = 3 The average silhouette_score is : 0.260409768356324
For n_clusters = 3 The average SSE is : 237.01495785978295
--- 5.756635904312134 seconds ---
================================================================================
====
n_clusters =  4
================================================================================
====
For n_clusters = 4 The average silhouette_score is : 0.2749784228827367
For n_clusters = 4 The average SSE is : 191.5564379329348
--- 7.438884973526001 seconds ---
================================================================================
====
n_clusters =  5
================================================================================
====
For n_clusters = 5 The average silhouette_score is : 0.2715519364075326
For n_clusters = 5 The average SSE is : 163.49507869017359
--- 6.142277956008911 seconds ---
================================================================================
====
n_clusters =  6
================================================================================
====
For n_clusters = 6 The average silhouette_score is : 0.263011224680275
For n_clusters = 6 The average SSE is : 145.85463685892572
--- 6.458942413330078 seconds ---
```

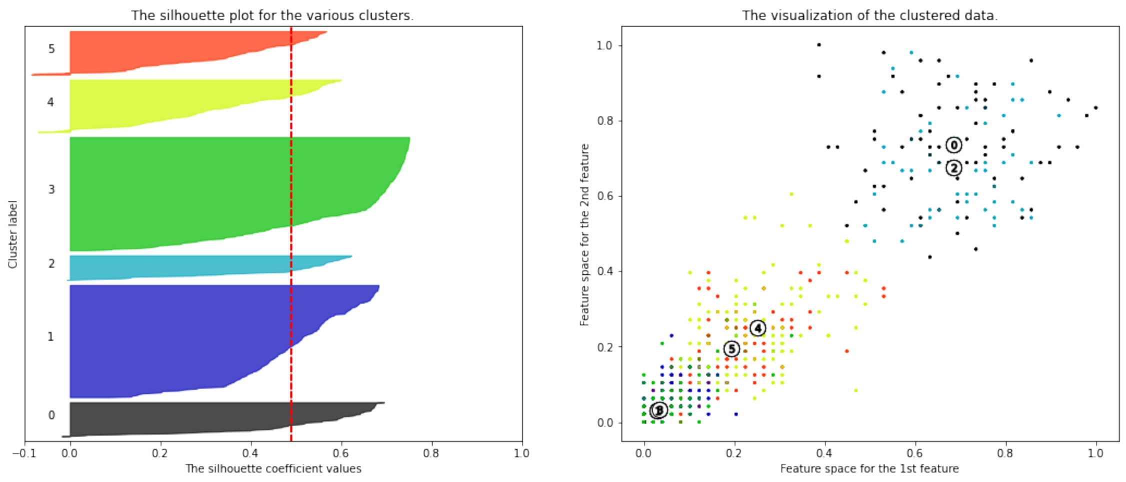

Silhouette analysis for KMeans clustering on sample data with n_clusters = 2
The silhouette plot for the various clusters.
Cluster label
1
0
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature
0
1


Silhouette analysis for KMeans clustering on sample data with n_clusters = 3
The silhouette plot for the various clusters.
Cluster label
2
1
0
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature
1

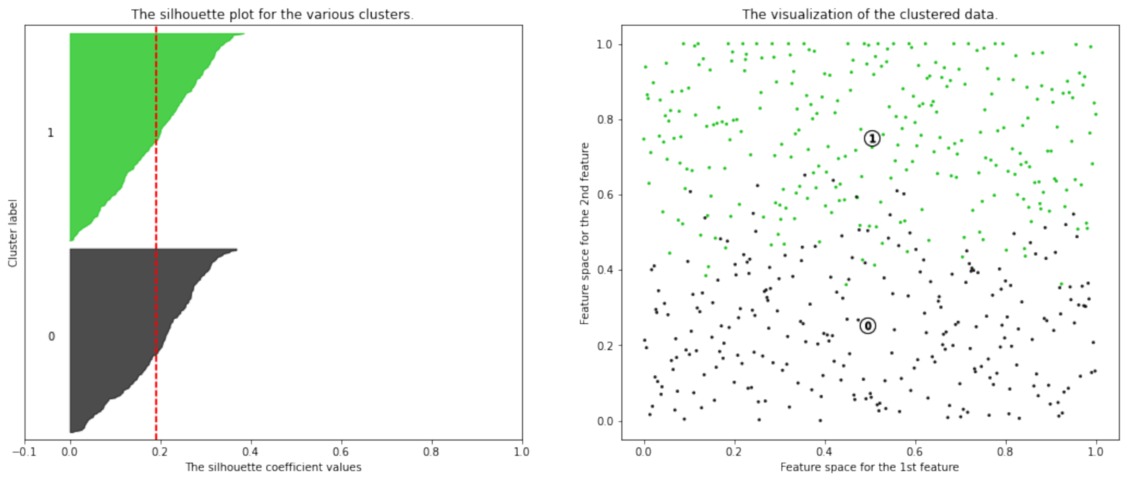

Silhouette analysis for KMeans clustering on sample data with n_clusters = 4
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature


Silhouette analysis for KMeans clustering on sample data with n_clusters = 5
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

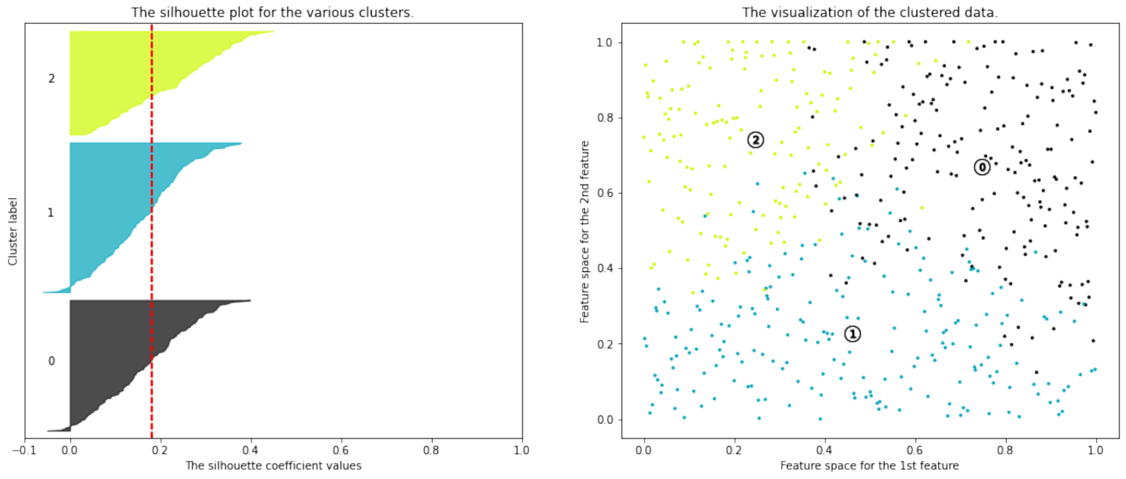
Silhouette analysis for KMeans clustering on sample data with n_clusters = 6



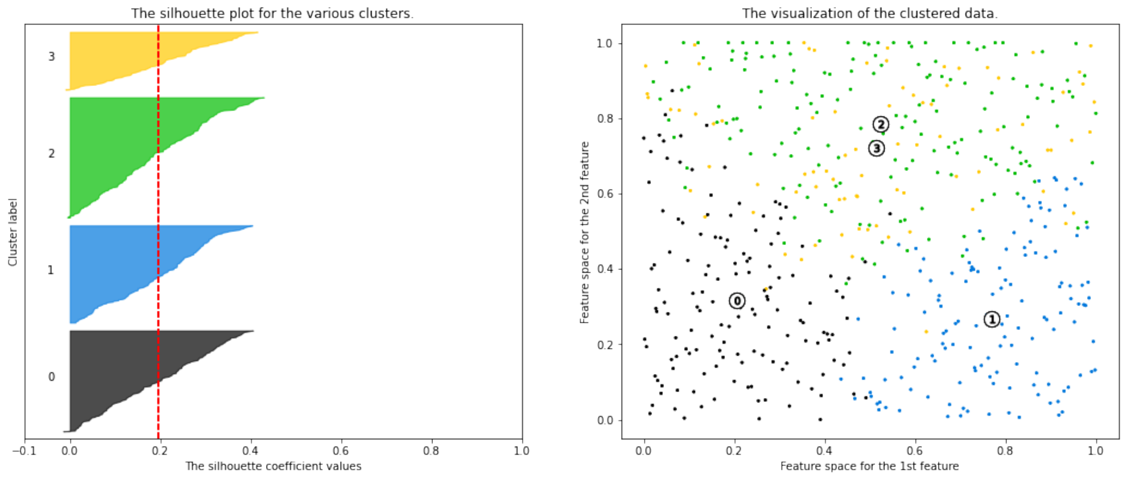
Silhouette analysis for KMeans clustering on sample data with n_clusters = 2

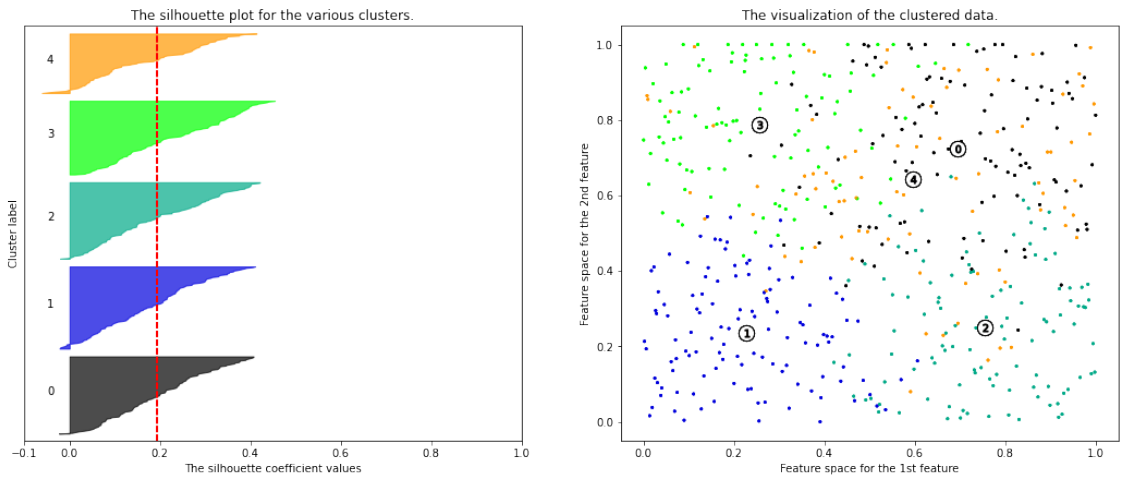


Silhouette analysis for KMeans clustering on sample data with n_clusters = 3

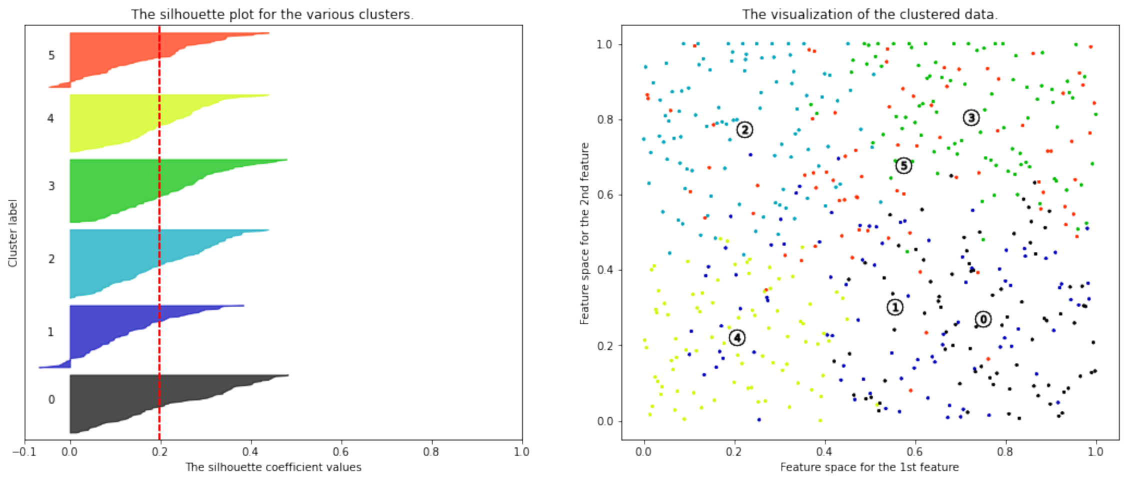


Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



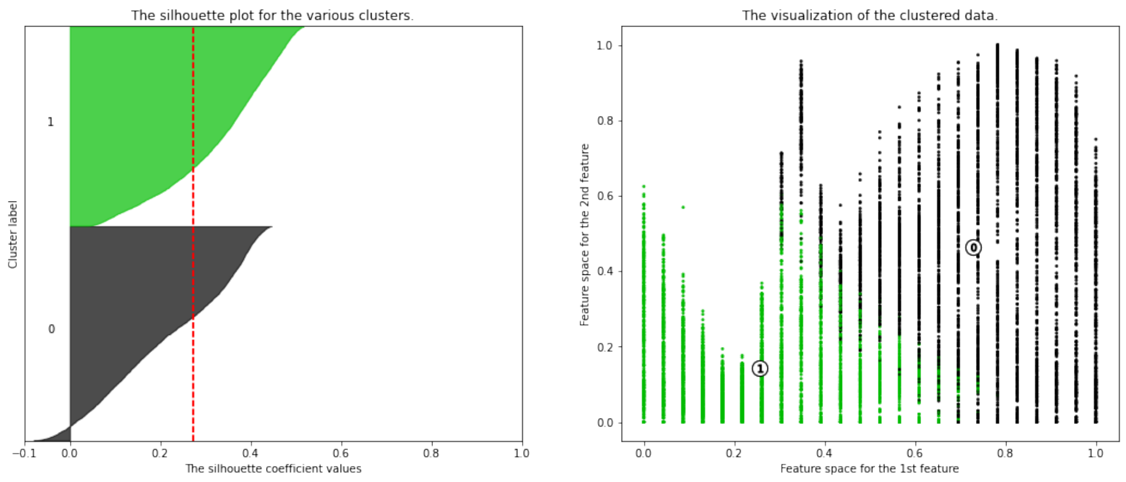


Silhouette analysis for KMeans clustering on sample data with n_clusters = 5

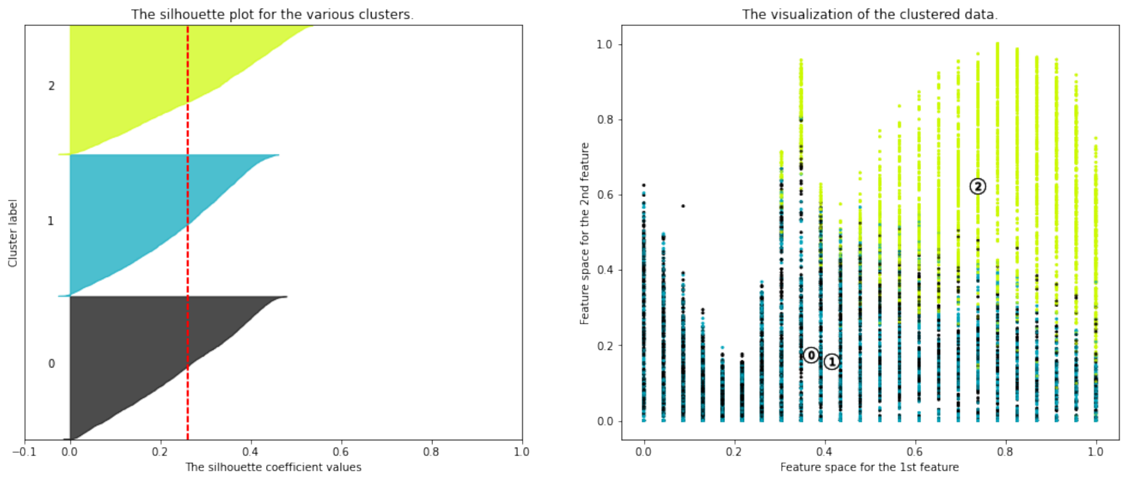


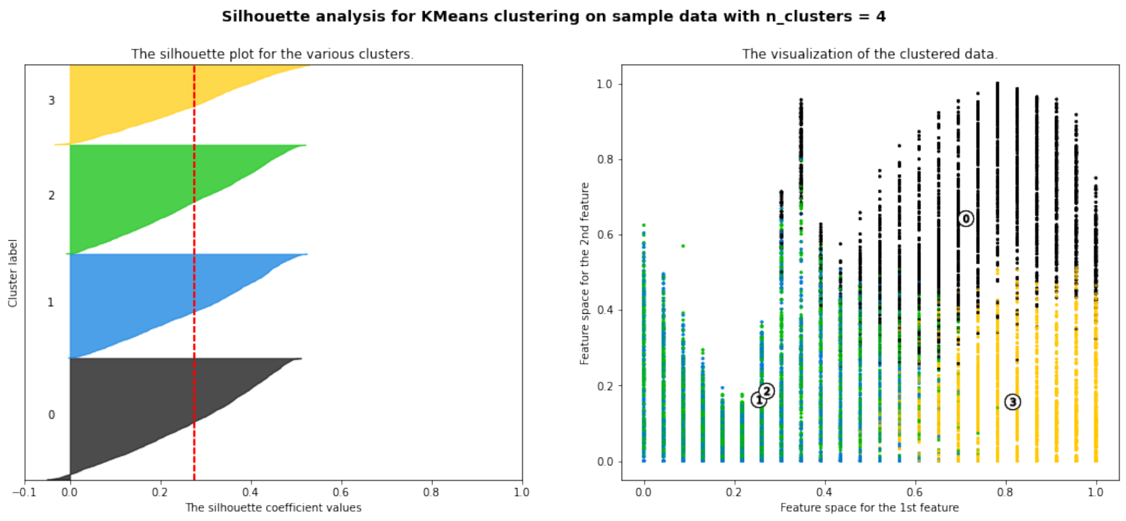
Silhouette analysis for KMeans clustering on sample data with n_clusters = 6

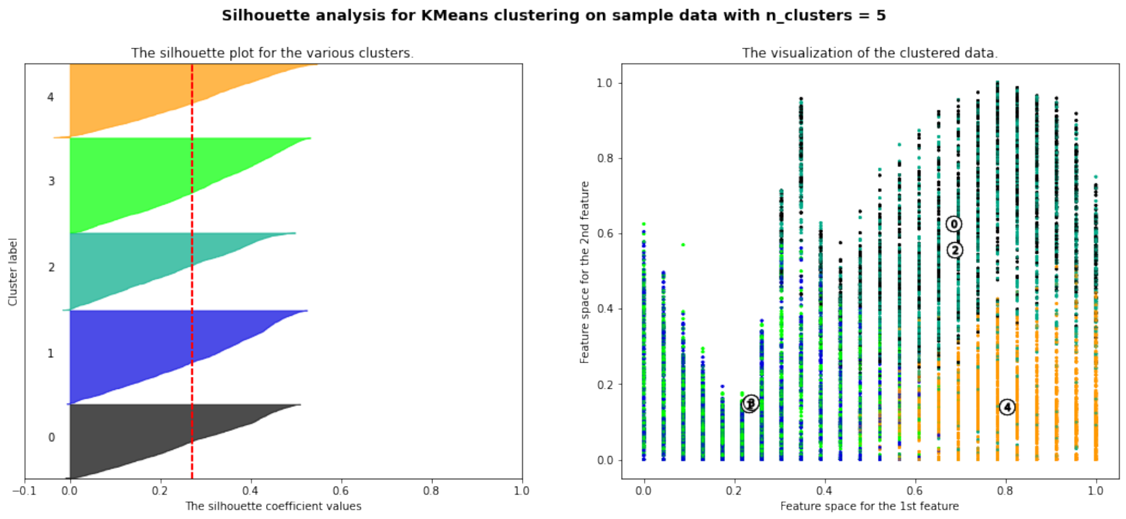
Silhouette analysis for KMeans clustering on sample data with n_clusters = 2

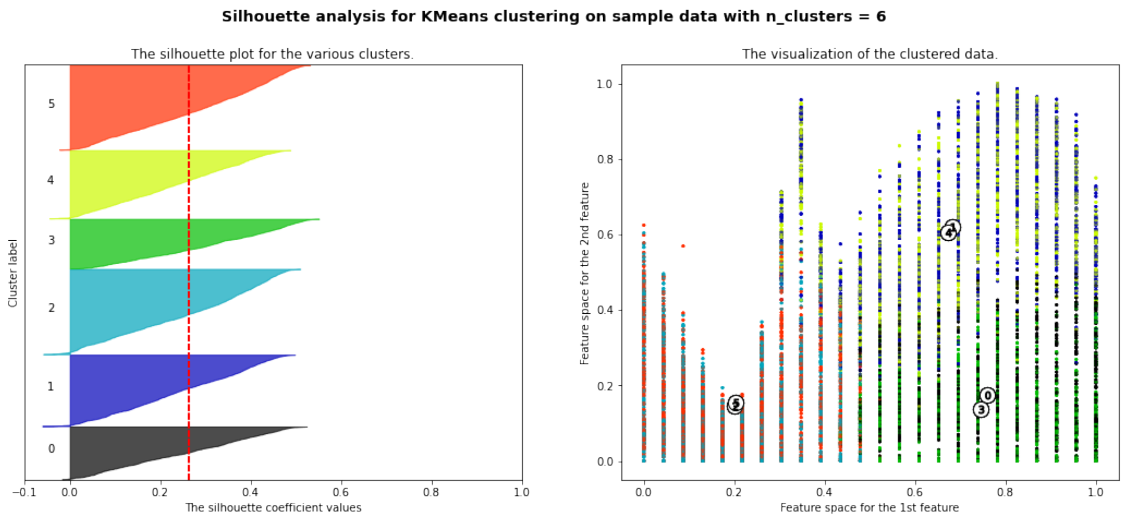


Silhouette analysis for KMeans clustering on sample data with n_clusters = 3




Silhouette analysis for KMeans clustering on sample data with n_clusters = 4
The silhouette plot for the various clusters.
The visualization of the clustered data.
Cluster label
The silhouette coefficient values
Feature space for the 2nd feature
Feature space for the 1st feature


Silhouette analysis for KMeans clustering on sample data with n_clusters = 5
The silhouette plot for the various clusters.
The visualization of the clustered data.
Cluster label
The silhouette coefficient values
Feature space for the 2nd feature
Feature space for the 1st feature


Silhouette analysis for KMeans clustering on sample data with n_clusters = 6
The silhouette plot for the various clusters.
The visualization of the clustered data.
Cluster label
The silhouette coefficient values
Feature space for the 1st feature
Feature space for the 2nd feature

```
[ ]:
```

19075153_O'Leary_PartA1_DB

October 16, 2021

[1]:
```
import pandas
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import warnings
from pandas.plotting import scatter_matrix
import seaborn as sns
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import tree
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from numpy import set_printoptions
from sklearn.decomposition import PCA
from sklearn import preprocessing

warnings.filterwarnings('ignore')
```

1 Data load and pre-processing

[2]:
```
min_max_scaler = preprocessing.MinMaxScaler()

#################
# load data sales
#################
path_sales = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/
 ↪Sales_Transactions_Dataset_Weekly.csv"
rawdata_sales = pandas.read_csv(path_sales)
# categorise everything and create array
list_of_columns_sales = rawdata_sales.columns
rawdata_sales[list_of_columns_sales] = rawdata_sales[list_of_columns_sales].
 ↪apply(lambda col:pandas.Categorical(col).codes)
# Create array
array_sales = rawdata_sales.values
predictors_sales = array_sales[:, 0:107]
# Print some stats
```

```
print(rawdata_sales.shape)
print(rawdata_sales.head())

#################
# load data seoul
#################
path_seoul = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/SeoulBikeData.csv"
rawdata_seoul = pandas.read_csv(path_seoul)
# categorise everything and create array
list_of_columns_seoul = rawdata_seoul.columns
rawdata_seoul[list_of_columns_seoul] = rawdata_seoul[list_of_columns_seoul].
 ↪apply(lambda col:pandas.Categorical(col).codes)
# Create array
array_seoul = rawdata_seoul.values
predictors_seoul = array_seoul[:, 0:14]
# Print some stats
print(rawdata_seoul.shape)
print(rawdata_seoul.head())

#################
# load data water
#################
path_water = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/water-treatment.data"
rawdata_water = pandas.read_csv(path_water)
rawdata_water.columns =␣
 ↪['DATE','Q-E','ZN-E','PH-E','DBO-E','DQO-E','SS-E','SSV-E','SED-E','COND-E','PH-P','DBO-P',
# categorise everything and create array
list_of_columns_water = rawdata_water.columns
rawdata_water[list_of_columns_water] = rawdata_water[list_of_columns_water].
 ↪apply(lambda col:pandas.Categorical(col).codes)
# Create array
array_water = rawdata_water.values
predictors_water = array_water[:, 0:39]
# Print some stats
print(rawdata_water.shape)
print(rawdata_water.head())
```

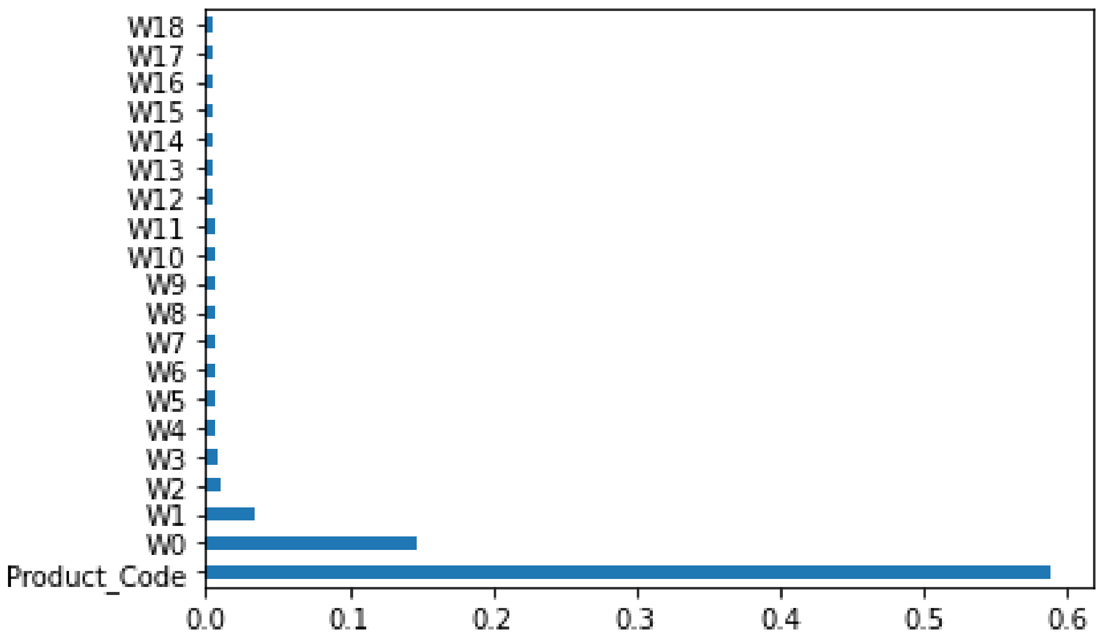
```
(811, 107)
   Product_Code  W0  W1  W2  W3  W4  W5  W6  W7  W8  …  Normalized 42  \
0             0  11  12  10   8  13  12  14  21   6  …              3
1           111   7   6   3   2   7   1   6   3   3  …             17
2           222   7  11   8   9  10   8   7  13  12  …             24
3           331  12   8  13   5   9   6   9  13  13  …             37
4           442   8   5  13  11   6   7   9  14   9  …             24

   Normalized 43  Normalized 44  Normalized 45  Normalized 46  Normalized 47  \
0             20             26             35             46              0
```

```
1             38            47            7             6            35
2             83            16           15            32            40
3             45             4            9            20            30
4             51            25           56            16            15

   Normalized 48  Normalized 49  Normalized 50  Normalized 51
0             16             13              7             35
1             44              7             55              0
2             82             41             41             32
3             65             31             25             31
4              8             49             29             36

[5 rows x 107 columns]
(8760, 14)
   Date  Rented Bike Count  Hour  Temperature( C)  Humidity(%)  \
0    11                253     0              111           28
1    11                203     1              108           29
2    11                172     2              103           30
3    11                106     3              101           31
4    11                 77     4              103           27

   Wind speed (m/s)  Visibility (10m)  Dew point temperature( C)  \
0                22              1788                        114
1                 8              1788                        114
2                10              1788                        113
3                 9              1788                        114
4                23              1788                        104

   Solar Radiation (MJ/m2)  Rainfall(mm)  Snowfall (cm)  Seasons  Holiday  \
0                        0             0              0        3        1
1                        0             0              0        3        1
2                        0             0              0        3        1
3                        0             0              0        3        1
4                        0             0              0        3        1

   Functioning Day
0                1
1                1
2                1
3                1
4                1
(526, 39)
   DATE  Q-E  ZN-E  PH-E  DBO-E  DQO-E  SS-E  SSV-E  SED-E  COND-E  …  \
0   197  330   116     6    204    169    57    201     50     409  …
1   427   99   143     5    204    231    42    208     28     303  …
2   443  219   126     8     93    256    45    171     36     402  …
3   461  282    66     9    126    211    37    164     33     381  …
4   479  319   116     7     90    117    42    197     36     295  …
```

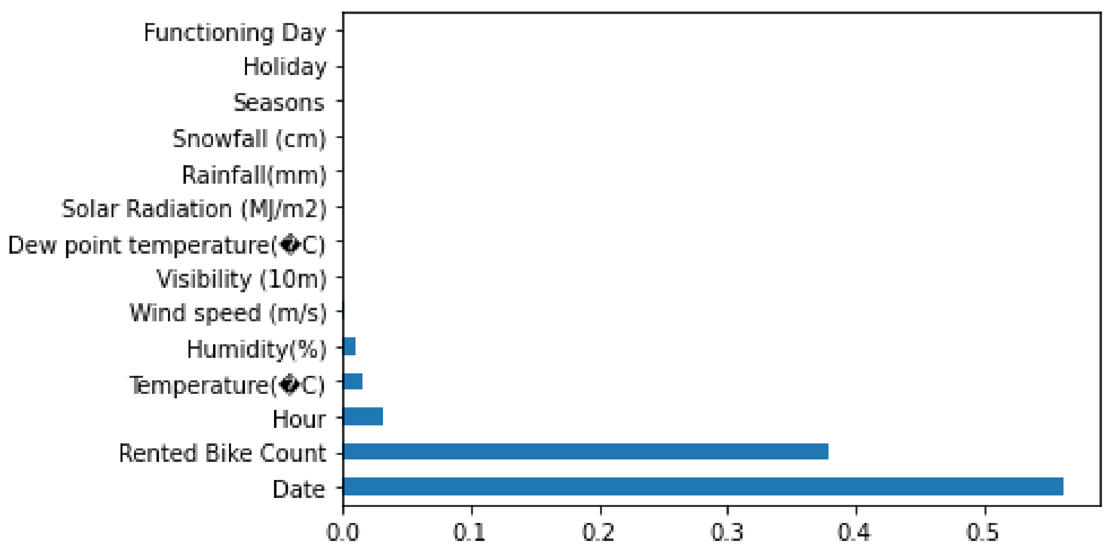
```
   COND-S  RD-DBO-P  RD-SS-P  RD-SED-P  RD-DBO-S  RD-DQO-S  RD-DBO-G  \
0     372       314      165       104       184       233       155
1     334       314      143       111       184        37       155
2     322       100      196       108       131       165        95
3     349       314      183       111       184       153       114
4     301       314      157       118       125       216        94

   RD-DQO-G  RD-SS-G  RD-SED-G
0       134      126         0
1       101       92        26
2       158      101         0
3       121       82        37
4        78       61         0

[5 rows x 39 columns]
```

2 Feature importance

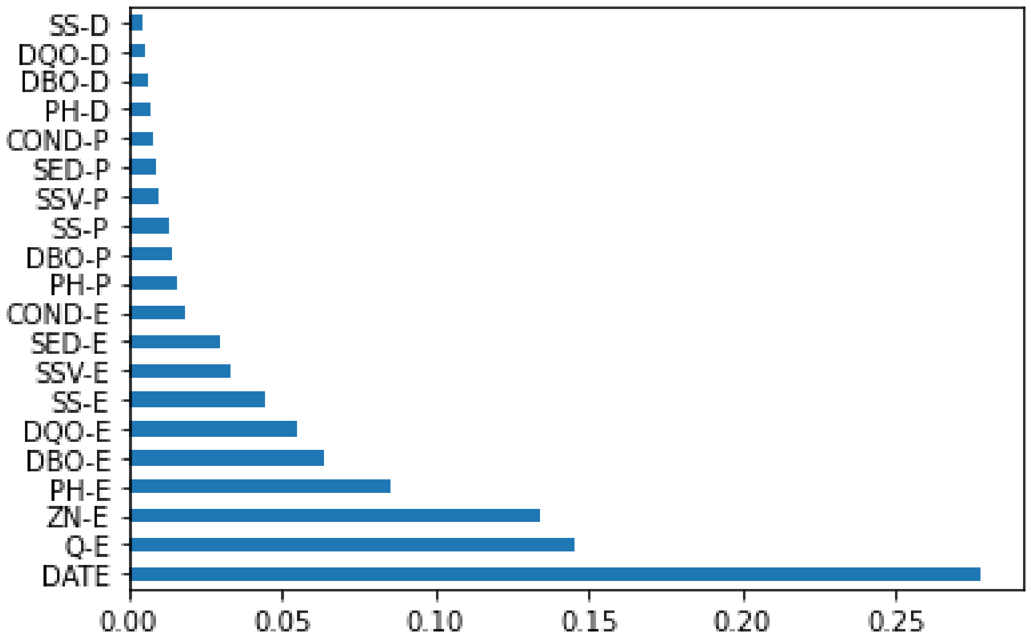
[3]:
```
#################
# Sales
#################
# Run PCA
pca_sales = PCA()
pca_fit_sales = pca_sales.fit(predictors_sales)
# summarize components
feat_importances_sales = pandas.Series(pca_fit_sales.explained_variance_ratio_,␣
 ↪index=rawdata_sales.columns)
feat_importances_sales.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca_sales = PCA(n_components=2)
pca_fit_sales = pca_sales.fit(predictors_sales)
pca_data_sales = rawdata_sales[['W0','W1','Product_Code']]
```



[4]:
```
#################
# Seoul
#################
# Run PCA
pca_seoul = PCA()
pca_fit_seoul = pca_seoul.fit(predictors_seoul)
# summarize components
feat_importances_seoul = pandas.Series(pca_fit_seoul.explained_variance_ratio_,␣
 ↪index=rawdata_seoul.columns)
feat_importances_seoul.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca_seoul = PCA(n_components=2)
pca_fit_seoul = pca_seoul.fit(predictors_seoul)
pca_data_seoul = rawdata_seoul[['Hour','Rented Bike␣
 ↪Count','Date','Temperature( C)']]
```



[5]:
```
#################
# Water
#################
# Run PCA
pca_water = PCA()
pca_fit_water = pca_water.fit(predictors_water)
# summarize components
feat_importances_water = pandas.Series(pca_fit_water.explained_variance_ratio_,␣
 ↪index=rawdata_water.columns)
feat_importances_water.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca_water = PCA(n_components=2)
pca_fit_water = pca_water.fit(predictors_water)
pca_data_water = rawdata_water[['DATE','Q-E','ZN-E','PH-E','DBO-E']]
```



3 Clustering

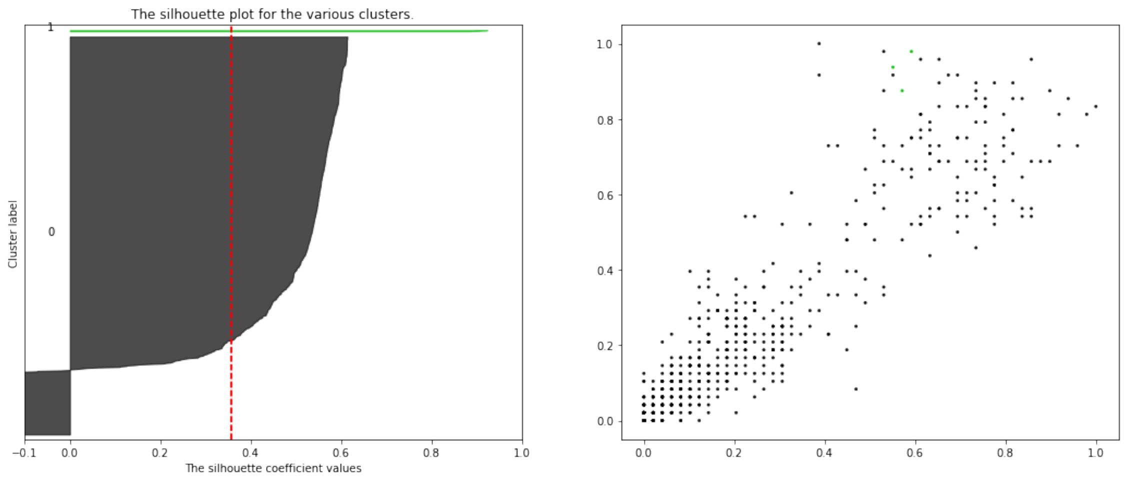
[12]:
```python
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
import time


def do_sse(X, cluster_labels, n_clusters, model):
    cluster_centers = [X[cluster_labels == i].mean(axis=0) for i in␣
 ↪range(n_clusters)]
    clusterwise_sse = [0, 0, 0, 0, 0, 0]
    for point, label in zip(X, cluster_labels):
        clusterwise_sse[label] += np.square(point - cluster_centers[label]).
 ↪sum()
    clusterwise_sse_avg = np.mean(clusterwise_sse)
    return clusterwise_sse_avg

def do_cluster_analysis(name):
    # To find out the optimal number of clusters we can search through range of␣
  ↪clusters.
    range_n_clusters = [2, 3, 4, 5, 6]
    for n_clusters in range_n_clusters:

       ␣
  ↪print('==========================================================================
        print('n_clusters = ', n_clusters)
       ␣
  ↪print('==========================================================================
        start_time = time.time()

        # Create a subplot with 1 row and 2 columns
        fig, (ax1, ax2) = plt.subplots(1, 2)
        fig.set_size_inches(18, 7)

        # The 1st subplot is the silhouette plot
        # The silhouette coefficient can range from -1, 1
        # but in this example code all lie within [-0.1, 1]

        ax1.set_xlim([-0.1, 1])

        # # The (n_clusters+1)*10 is for inserting blank space between
        # silhouette plots of individual clusters, to demarcate them
        # clearly.

        ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

        #Apply DBSCAN and set the EPS
        eps = 0
        if name == 'sales':
            eps = 0.2
        elif name == 'water':
            eps = 0.3
        elif name == 'seoul':
            eps = 0.1
        clusterer = DBSCAN(eps, min_samples=n_clusters, metric='euclidean')
        cluster_labels = clusterer.fit_predict(X)

        # The silhouette_score gives the average value for all the
        # samples.This gives a perspective into the density and
        # separation of the formed clusters

        try:
            silhouette_avg = silhouette_score(X, cluster_labels)
```

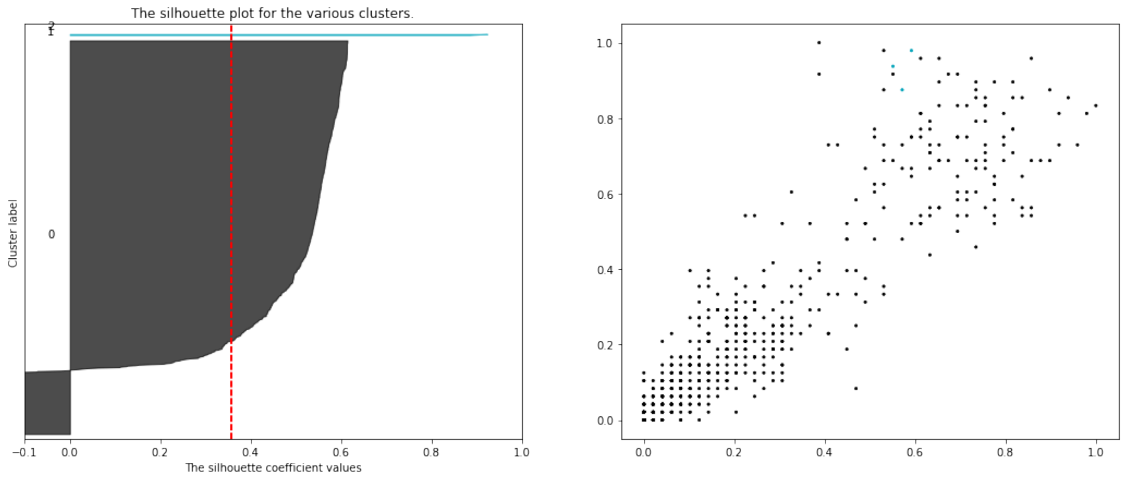
```
            #␣
↪////////////////////////////////////////////////////////////////////////////////////
            # Print the values
            #␣
↪////////////////////////////////////////////////////////////////////////////////////
            print("For n_clusters =", n_clusters, "The average silhouette_score␣
↪is :", silhouette_avg)
            print("For n_clusters =", n_clusters, "The average SSE is :",␣
↪do_sse(X, clusterer.labels_, n_clusters, clusterer))
        except:
            print('DBSCAN EXCEPTION')
            break

        # Compute the silhouette scores for each sample
        sample_silhouette_values = silhouette_samples(X, cluster_labels)
        y_lower = 10

        for i in range(n_clusters):
            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them

            #␣
↪////////////////////////////////////////////////////////////////////////////////////
            # Create the plot
            #␣
↪////////////////////////////////////////////////////////////////////////////////////

            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them
            ith_cluster_silhouette_values =␣
↪sample_silhouette_values[cluster_labels == i]

            ith_cluster_silhouette_values.sort()
            size_cluster_i = ith_cluster_silhouette_values.shape[0]
            y_upper = y_lower + size_cluster_i
            color = cm.nipy_spectral(float(i) / n_clusters)

            ax1.fill_betweenx(np.arange(y_lower, y_upper),
                              0, ith_cluster_silhouette_values,
                              facecolor=color, edgecolor=color,
                              alpha=0.7)

            # Label the silhouette plots with their cluster numbers at the
            # middle

            ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
            # Compute the new y_lower for next plot

            y_lower = y_upper + 10  # 10 for the 0 samples
            ax1.set_title("The silhouette plot for the various clusters.")
            ax1.set_xlabel("The silhouette coefficient values")
            ax1.set_ylabel("Cluster label")

            # The vertical line for average silhouette score of all the
            # values

            ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
            ax1.set_yticks([])  # Clear the yaxis labels / ticks
            ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

            # 2nd Plot showing the actual clusters formed
            colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
            ax2.scatter(X[:, 0],
                        X[:, 1],
                        marker='.',
                        s=30,
                        lw=0,
                        alpha=0.7,
                        c=colors,
                        edgecolor='k')

            # Labeling the clusters by centers
            centers = clusterer.labels_

        # Time to run
        print("--- %s seconds ---" % (time.time() - start_time))

#################
# Sales
#################
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('sales')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
X = pca_data_sales
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
X = x_scaled
do_cluster_analysis('sales')

#################
```

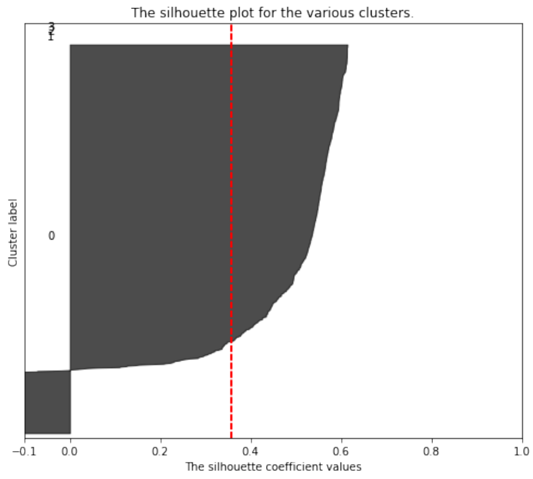
```
# Water
#################
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('water')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
X = pca_data_water
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
X = x_scaled
do_cluster_analysis('water')

#################
# Seoul
#################
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('seoul')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
X = pca_data_seoul
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
X = x_scaled
do_cluster_analysis('seoul')
```

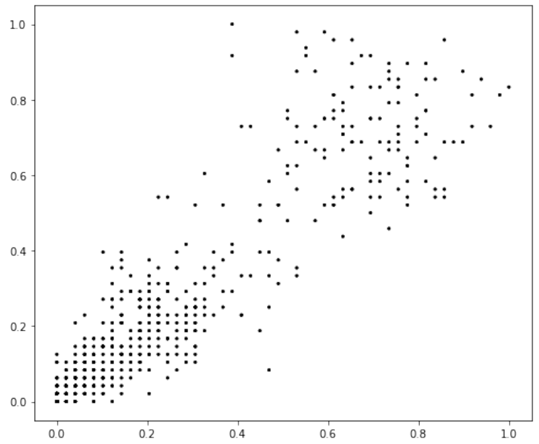
```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
sales
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
================================================================================
====
n_clusters =  2
================================================================================
====
For n_clusters = 2 The average silhouette_score is : 0.35714256858302806
For n_clusters = 2 The average SSE is : 27.772162712889894
--- 0.346055269241333 seconds ---
================================================================================
====
n_clusters =  3
```

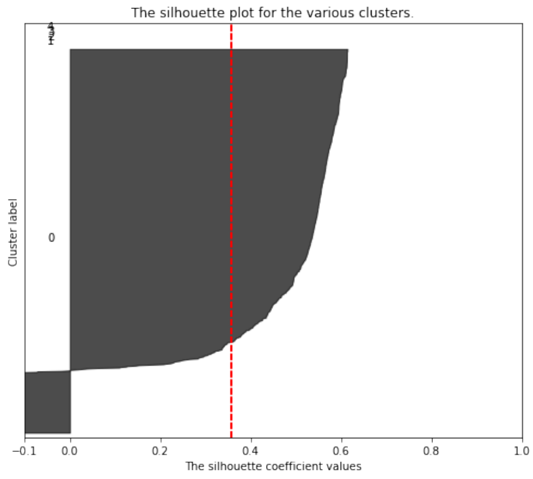
```
================================================================================
====
For n_clusters = 3 The average silhouette_score is : 0.35714256858302806
For n_clusters = 3 The average SSE is : 27.772162712889894
--- 0.23154139518737793 seconds ---
================================================================================
====
n_clusters =  4
================================================================================
====
For n_clusters = 4 The average silhouette_score is : 0.35714256858302806
For n_clusters = 4 The average SSE is : nan
--- 0.17214298248291016 seconds ---
================================================================================
====
n_clusters =  5
================================================================================
====
For n_clusters = 5 The average silhouette_score is : 0.35714256858302806
For n_clusters = 5 The average SSE is : nan
--- 0.14641571044921875 seconds ---
================================================================================
====
n_clusters =  6
================================================================================
====
For n_clusters = 6 The average silhouette_score is : 0.35714256858302806
For n_clusters = 6 The average SSE is : nan
--- 0.1602458953857422 seconds ---
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
water
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
================================================================================
====
n_clusters =  2
================================================================================
====
For n_clusters = 2 The average silhouette_score is : 0.01814249370553166
DBSCAN EXCEPTION
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
seoul
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
================================================================================
====
n_clusters =  2
================================================================================
====
For n_clusters = 2 The average silhouette_score is : -0.48965343976481296
DBSCAN EXCEPTION
```

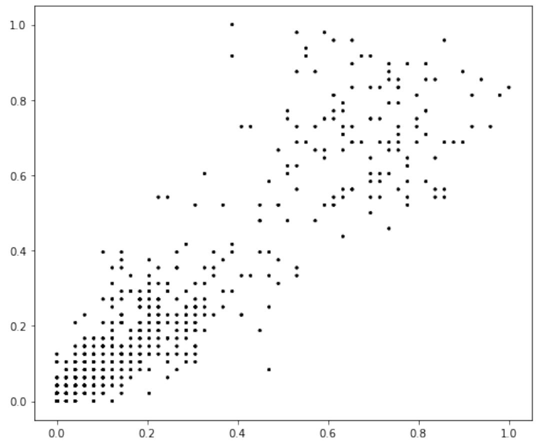


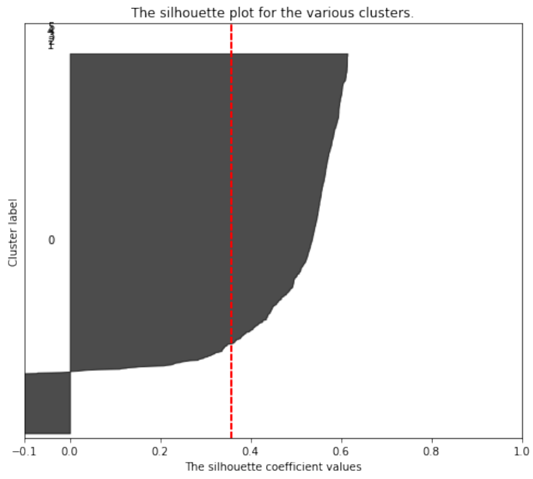


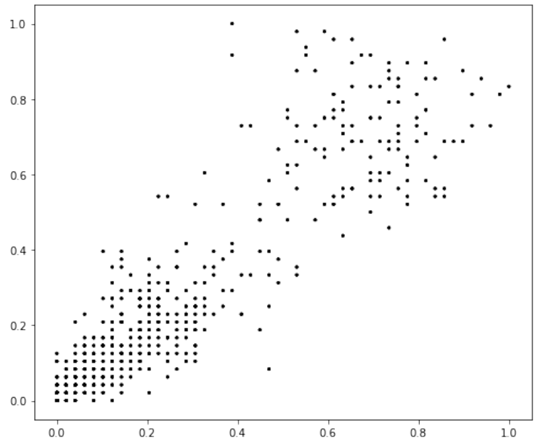

The silhouette plot for the various clusters.
Cluster label
0
The silhouette coefficient values

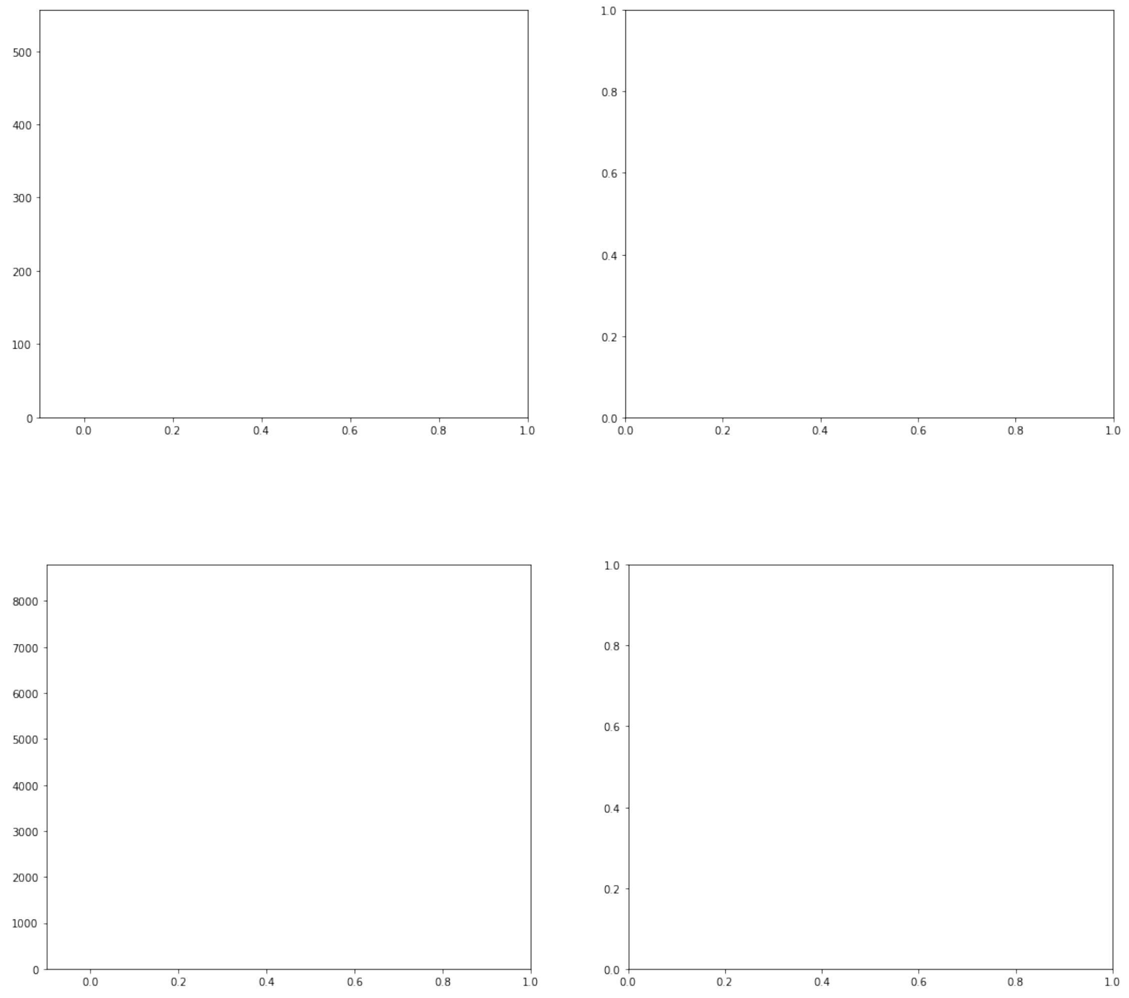



The silhouette plot for the various clusters.
Cluster label
0
The silhouette coefficient values




The silhouette plot for the various clusters.
Cluster label
0
The silhouette coefficient values




0
100
200
300
400
500
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
5000
6000
7000
8000

```
[ ]:
```

19075153_O'Leary_PartA1_AM

October 16, 2021

[1]:
```
import pandas
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import warnings
from pandas.plotting import scatter_matrix
import seaborn as sns
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import tree
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from numpy import set_printoptions
from sklearn.decomposition import PCA
from sklearn import preprocessing

warnings.filterwarnings('ignore')
```

1 Data load and pre-processing

[2]:
```
min_max_scaler = preprocessing.MinMaxScaler()

#################
# load data sales
#################
path_sales = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/
 ↪Sales_Transactions_Dataset_Weekly.csv"
rawdata_sales = pandas.read_csv(path_sales)
# categorise everything and create array
list_of_columns_sales = rawdata_sales.columns
rawdata_sales[list_of_columns_sales] = rawdata_sales[list_of_columns_sales].
 ↪apply(lambda col:pandas.Categorical(col).codes)
# Create array
array_sales = rawdata_sales.values
predictors_sales = array_sales[:, 0:107]
# Print some stats
```

```
print(rawdata_sales.shape)
print(rawdata_sales.head())

#################
# load data seoul
#################
path_seoul = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/SeoulBikeData.csv"
rawdata_seoul = pandas.read_csv(path_seoul)
# categorise everything and create array
list_of_columns_seoul = rawdata_seoul.columns
rawdata_seoul[list_of_columns_seoul] = rawdata_seoul[list_of_columns_seoul].
 ↪apply(lambda col:pandas.Categorical(col).codes)
# Create array
array_seoul = rawdata_seoul.values
predictors_seoul = array_seoul[:, 0:14]
# Print some stats
print(rawdata_seoul.shape)
print(rawdata_seoul.head())

#################
# load data water
#################
path_water = "/mnt/c/Users/T828808/Study/AUT/COMP809/Ass2/water-treatment.data"
rawdata_water = pandas.read_csv(path_water)
rawdata_water.columns =␣
 ↪['DATE','Q-E','ZN-E','PH-E','DBO-E','DQO-E','SS-E','SSV-E','SED-E','COND-E','PH-P','DBO-P',
# categorise everything and create array
list_of_columns_water = rawdata_water.columns
rawdata_water[list_of_columns_water] = rawdata_water[list_of_columns_water].
 ↪apply(lambda col:pandas.Categorical(col).codes)
# Create array
array_water = rawdata_water.values
predictors_water = array_water[:, 0:39]
# Print some stats
print(rawdata_water.shape)
print(rawdata_water.head())
```

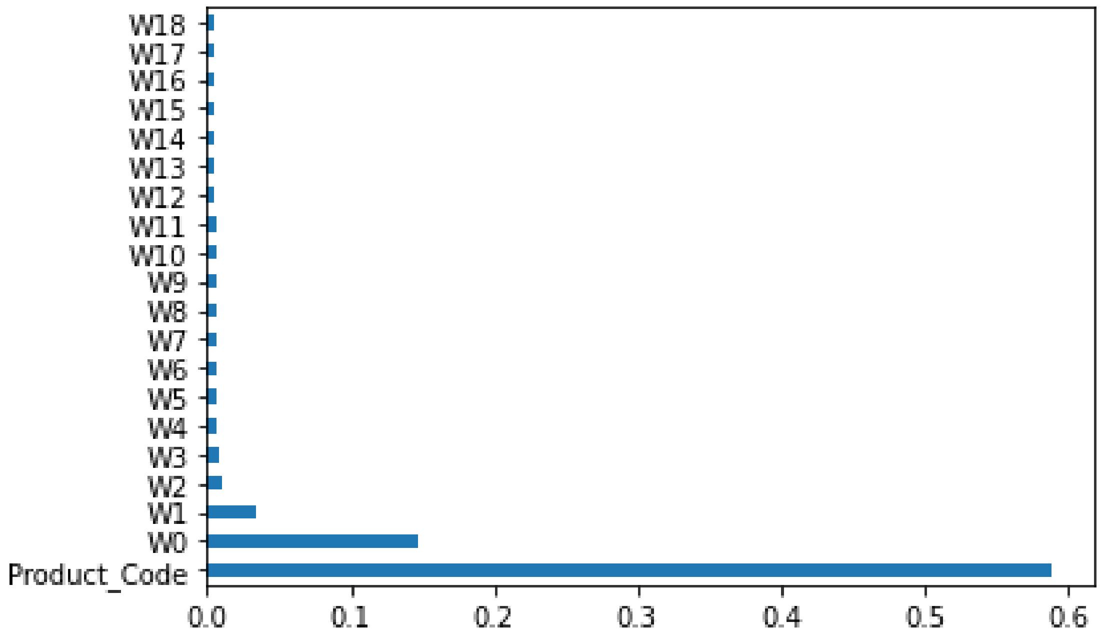
```
(811, 107)
   Product_Code  W0  W1  W2  W3  W4  W5  W6  W7  W8  …  Normalized 42  \
0             0  11  12  10   8  13  12  14  21   6  …              3
1           111   7   6   3   2   7   1   6   3   3  …             17
2           222   7  11   8   9  10   8   7  13  12  …             24
3           331  12   8  13   5   9   6   9  13  13  …             37
4           442   8   5  13  11   6   7   9  14   9  …             24

   Normalized 43  Normalized 44  Normalized 45  Normalized 46  Normalized 47  \
0             20             26             35             46              0
```

```
1            38           47           7            6           35
2            83           16          15           32           40
3            45            4           9           20           30
4            51           25          56           16           15

   Normalized 48  Normalized 49  Normalized 50  Normalized 51
0             16             13              7             35
1             44              7             55              0
2             82             41             41             32
3             65             31             25             31
4              8             49             29             36

[5 rows x 107 columns]
(8760, 14)
   Date  Rented Bike Count  Hour  Temperature( C)  Humidity(%)  \
0    11                253     0              111           28
1    11                203     1              108           29
2    11                172     2              103           30
3    11                106     3              101           31
4    11                 77     4              103           27

   Wind speed (m/s)  Visibility (10m)  Dew point temperature( C)  \
0                22              1788                        114
1                 8              1788                        114
2                10              1788                        113
3                 9              1788                        114
4                23              1788                        104

   Solar Radiation (MJ/m2)  Rainfall(mm)  Snowfall (cm)  Seasons  Holiday  \
0                        0             0              0        3        1
1                        0             0              0        3        1
2                        0             0              0        3        1
3                        0             0              0        3        1
4                        0             0              0        3        1

   Functioning Day
0                1
1                1
2                1
3                1
4                1
(526, 39)
   DATE  Q-E  ZN-E  PH-E  DBO-E  DQO-E  SS-E  SSV-E  SED-E  COND-E  …  \
0   197  330   116     6    204    169    57    201     50     409  …
1   427   99   143     5    204    231    42    208     28     303  …
2   443  219   126     8     93    256    45    171     36     402  …
3   461  282    66     9    126    211    37    164     33     381  …
4   479  319   116     7     90    117    42    197     36     295  …
```

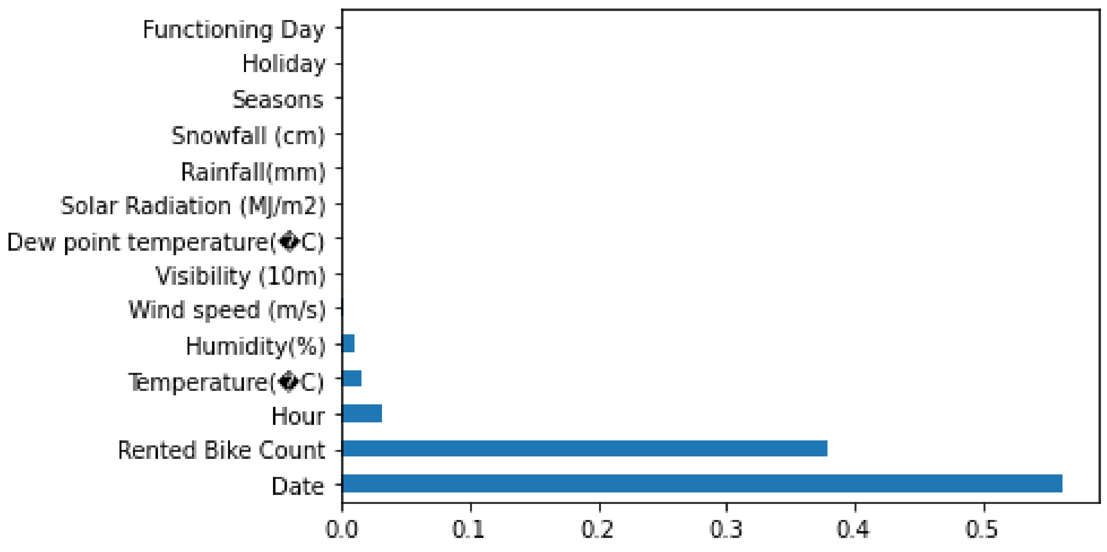
```
   COND-S  RD-DBO-P  RD-SS-P  RD-SED-P  RD-DBO-S  RD-DQO-S  RD-DBO-G  \
0     372       314      165       104       184       233       155
1     334       314      143       111       184        37       155
2     322       100      196       108       131       165        95
3     349       314      183       111       184       153       114
4     301       314      157       118       125       216        94

   RD-DQO-G  RD-SS-G  RD-SED-G
0       134      126         0
1       101       92        26
2       158      101         0
3       121       82        37
4        78       61         0

[5 rows x 39 columns]
```

2 Feature importance

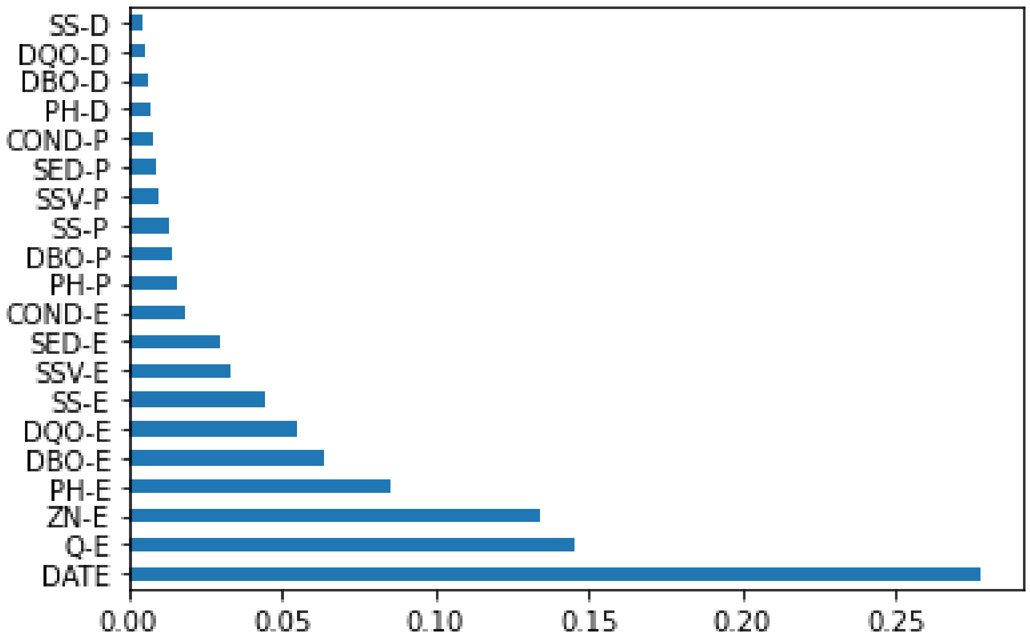
[3]:
```
#################
# Sales
#################
# Run PCA
pca_sales = PCA()
pca_fit_sales = pca_sales.fit(predictors_sales)
# summarize components
feat_importances_sales = pandas.Series(pca_fit_sales.explained_variance_ratio_,␣
 ↪index=rawdata_sales.columns)
feat_importances_sales.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca_sales = PCA(n_components=2)
pca_fit_sales = pca_sales.fit(predictors_sales)
pca_data_sales = rawdata_sales[['W0','W1','Product_Code']]
```



[4]:
```
#################
# Seoul
#################
# Run PCA
pca_seoul = PCA()
pca_fit_seoul = pca_seoul.fit(predictors_seoul)
# summarize components
feat_importances_seoul = pandas.Series(pca_fit_seoul.explained_variance_ratio_,␣
 ↪index=rawdata_seoul.columns)
feat_importances_seoul.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca_seoul = PCA(n_components=2)
pca_fit_seoul = pca_seoul.fit(predictors_seoul)
pca_data_seoul = rawdata_seoul[['Hour','Rented Bike␣
 ↪Count','Date','Temperature( C)']]
```



[5]:
```
#################
# Water
#################
# Run PCA
pca_water = PCA()
pca_fit_water = pca_water.fit(predictors_water)
# summarize components
feat_importances_water = pandas.Series(pca_fit_water.explained_variance_ratio_,␣
↪index=rawdata_water.columns)
feat_importances_water.nlargest(20).plot(kind='barh')
# Get only the first two components as they explain almost all of the variance
pca_water = PCA(n_components=2)
pca_fit_water = pca_water.fit(predictors_water)
pca_data_water = rawdata_water[['DATE','Q-E','ZN-E','PH-E','DBO-E']]
```



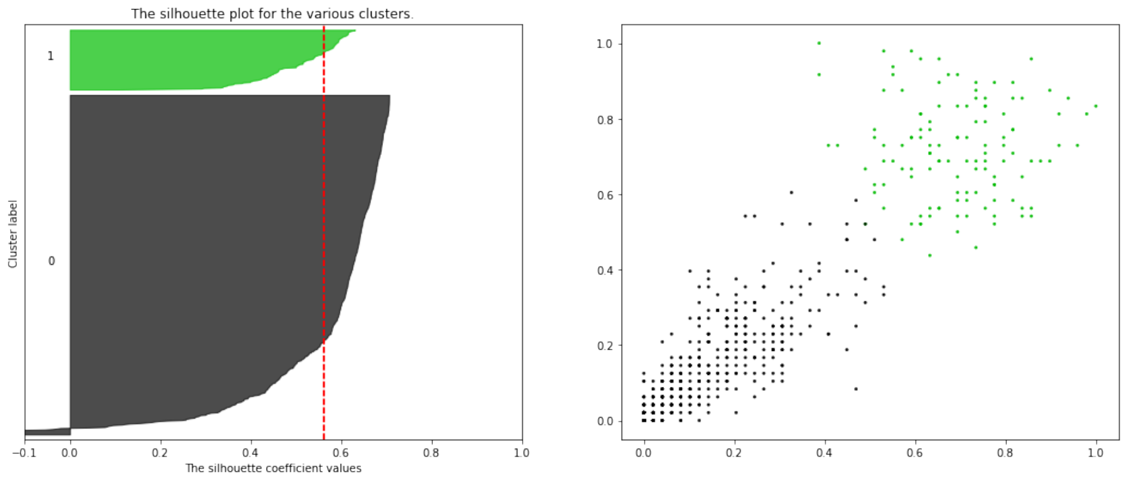
3 Clustering

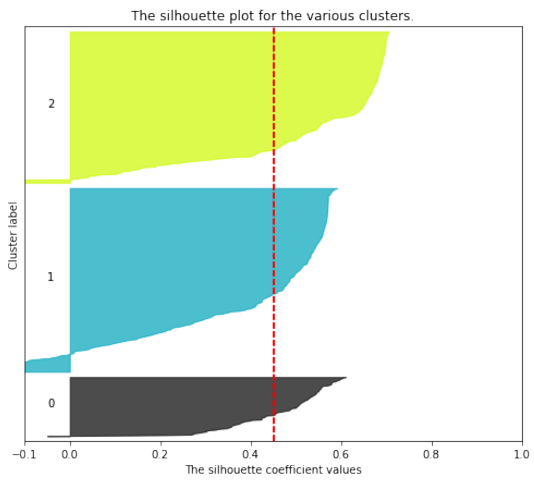
[9]:
```python
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
import time


def do_sse(X, cluster_labels, n_clusters, model):
    cluster_centers = [X[cluster_labels == i].mean(axis=0) for i in␣
 ↪range(n_clusters)]
    clusterwise_sse = [0, 0, 0, 0, 0, 0]
    for point, label in zip(X, cluster_labels):
        clusterwise_sse[label] += np.square(point - cluster_centers[label]).
 ↪sum()
    clusterwise_sse_avg = np.mean(clusterwise_sse)
    return clusterwise_sse_avg

def do_cluster_analysis(name):
    # To find out the optimal number of clusters we can search through range of␣
↪clusters.
    range_n_clusters = [2, 3, 4, 5, 6]
    for n_clusters in range_n_clusters:

        ␣
↪print('================================================================================
        print('n_clusters = ', n_clusters)
        ␣
↪print('================================================================================
        start_time = time.time()

        # Create a subplot with 1 row and 2 columns
        fig, (ax1, ax2) = plt.subplots(1, 2)
        fig.set_size_inches(18, 7)

        # The 1st subplot is the silhouette plot
        # The silhouette coefficient can range from -1, 1
        # but in this example code all lie within [-0.1, 1]

        ax1.set_xlim([-0.1, 1])

        # # The (n_clusters+1)*10 is for inserting blank space between
        # silhouette plots of individual clusters, to demarcate them
        # clearly.

        ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

        #Apply AgglomerativeClustering
        clusterer = AgglomerativeClustering(n_clusters, affinity='euclidean',␣
↪linkage='complete')
        cluster_labels = clusterer.fit_predict(X)

        # The silhouette_score gives the average value for all the
        # samples.This gives a perspective into the density and
        # separation of the formed clusters

        silhouette_avg = silhouette_score(X, cluster_labels)
        #␣
↪////////////////////////////////////////////////////////////////////////////////////
        # Print the values
        #␣
↪////////////////////////////////////////////////////////////////////////////////////
        print("For n_clusters =", n_clusters, "The average silhouette_score is :
↪", silhouette_avg)
```

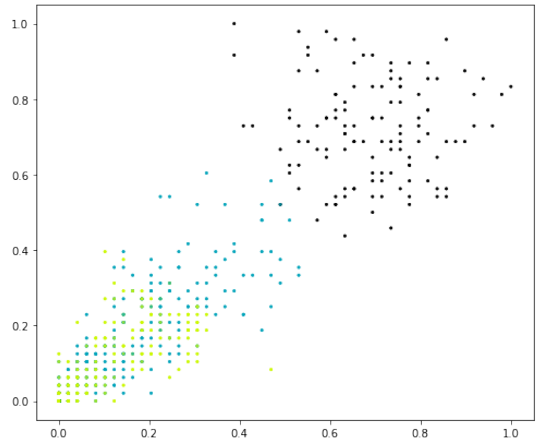
```
        print("For n_clusters =", n_clusters, "The average SSE is :", do_sse(X,␣
↪clusterer.labels_, n_clusters, clusterer))

        # Compute the silhouette scores for each sample
        sample_silhouette_values = silhouette_samples(X, cluster_labels)
        y_lower = 10

        for i in range(n_clusters):
            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them

            #␣
↪/////////////////////////////////////////////////////////////////////////////////
            # Create the plot
            #␣
↪/////////////////////////////////////////////////////////////////////////////////

            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them
            ith_cluster_silhouette_values =␣
↪sample_silhouette_values[cluster_labels == i]

            ith_cluster_silhouette_values.sort()
            size_cluster_i = ith_cluster_silhouette_values.shape[0]
            y_upper = y_lower + size_cluster_i
            color = cm.nipy_spectral(float(i) / n_clusters)

            ax1.fill_betweenx(np.arange(y_lower, y_upper),
                              0, ith_cluster_silhouette_values,
                              facecolor=color, edgecolor=color,
                              alpha=0.7)

            # Label the silhouette plots with their cluster numbers at the
            # middle

            ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

            # Compute the new y_lower for next plot

            y_lower = y_upper + 10  # 10 for the 0 samples
            ax1.set_title("The silhouette plot for the various clusters.")
            ax1.set_xlabel("The silhouette coefficient values")
            ax1.set_ylabel("Cluster label")

            # The vertical line for average silhouette score of all the
            # values
            ax1.axvline(x=silhouette_avg, color="red", linestyle="--")
            ax1.set_yticks([])  # Clear the yaxis labels / ticks
            ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

            # 2nd Plot showing the actual clusters formed
            colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
            ax2.scatter(X[:, 0],
                        X[:, 1],
                        marker='.',
                        s=30,
                        lw=0,
                        alpha=0.7,
                        c=colors,
                        edgecolor='k')

            # Labeling the clusters by centers
            centers = clusterer.labels_

        # Time to run
        print("--- %s seconds ---" % (time.time() - start_time))

#################
# Sales
#################
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('sales')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
X = pca_data_sales
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
X = x_scaled
do_cluster_analysis('sales')

#################
# Water
#################
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('water')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
X = pca_data_water
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
X = x_scaled
```

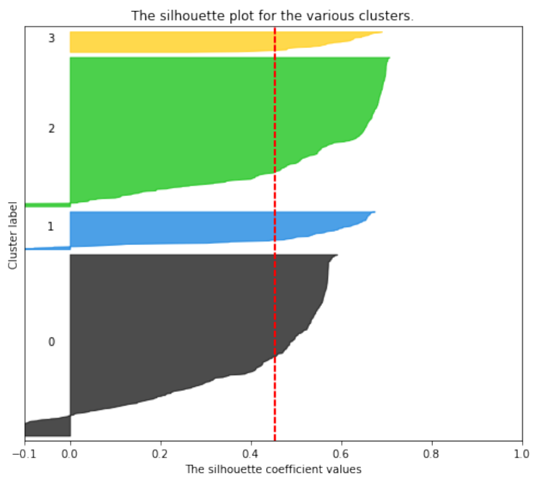
```python
do_cluster_analysis('water')

#################
# Seoul
#################
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('seoul')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
print('||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||')
X = pca_data_seoul
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
X = x_scaled
do_cluster_analysis('seoul')
```

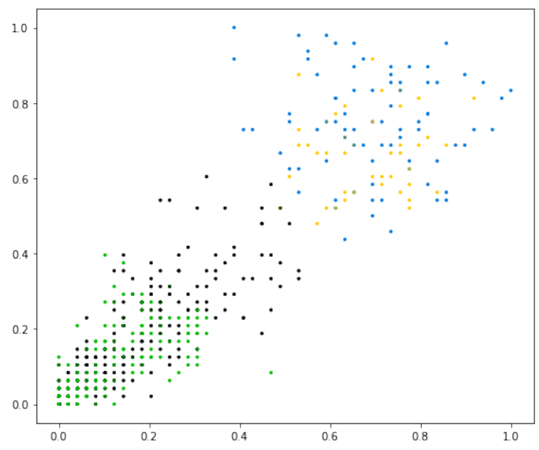
```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
sales
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
================================================================================
====
n_clusters =  2
================================================================================
====
For n_clusters = 2 The average silhouette_score is : 0.5630813204653953
For n_clusters = 2 The average SSE is : 14.907766709758254
--- 0.23120903968811035 seconds ---
================================================================================
====
n_clusters =  3
================================================================================
====
For n_clusters = 3 The average silhouette_score is : 0.4517887084052137
For n_clusters = 3 The average SSE is : 8.755768394980814
--- 0.15660643577575684 seconds ---
================================================================================
====
n_clusters =  4
================================================================================
====
For n_clusters = 4 The average silhouette_score is : 0.4537820793098297
For n_clusters = 4 The average SSE is : 6.962470134634124
--- 0.22800803184509277 seconds ---
================================================================================
====
n_clusters =  5
================================================================================
====
For n_clusters = 5 The average silhouette_score is : 0.45281621283033274
For n_clusters = 5 The average SSE is : 5.376949572658198
--- 0.23021531105041504 seconds ---
================================================================================
====
n_clusters =  6
================================================================================
====
For n_clusters = 6 The average silhouette_score is : 0.427459801328905
For n_clusters = 6 The average SSE is : 5.164992693901558
--- 0.5414533615112305 seconds ---
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
water
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
================================================================================
====
n_clusters =  2
================================================================================
====
For n_clusters = 2 The average silhouette_score is : 0.15524541375205828
For n_clusters = 2 The average SSE is : 24.894066528279865
--- 0.12308955192565918 seconds ---
================================================================================
====
n_clusters =  3
================================================================================
====
For n_clusters = 3 The average silhouette_score is : 0.1227273605961021
For n_clusters = 3 The average SSE is : 22.316291857947885
--- 0.12271738052368164 seconds ---
================================================================================
====
n_clusters =  4
================================================================================
====
For n_clusters = 4 The average silhouette_score is : 0.13617035449634593
For n_clusters = 4 The average SSE is : 20.056887615553745
--- 0.12484121322631836 seconds ---
================================================================================
====
n_clusters =  5
================================================================================
====
For n_clusters = 5 The average silhouette_score is : 0.12186405302481013
For n_clusters = 5 The average SSE is : 18.775880768282853
--- 0.12403750419616699 seconds ---
================================================================================
====
n_clusters =  6
================================================================================
====
For n_clusters = 6 The average silhouette_score is : 0.1411477652760043
For n_clusters = 6 The average SSE is : 16.26094953544457
--- 0.13375568389892578 seconds ---
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
seoul
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||
================================================================================
====
n_clusters =  2
================================================================================
====
For n_clusters = 2 The average silhouette_score is : 0.24688690435967905
For n_clusters = 2 The average SSE is : 308.6434248773903
--- 8.33217167854309 seconds ---
================================================================================
====
n_clusters =  3
================================================================================
====
For n_clusters = 3 The average silhouette_score is : 0.23314084347983294
For n_clusters = 3 The average SSE is : 253.88981532100146
--- 7.827468156814575 seconds ---
================================================================================
====
n_clusters =  4
================================================================================
====
For n_clusters = 4 The average silhouette_score is : 0.2239192996437419
For n_clusters = 4 The average SSE is : 223.3114014879127
--- 7.871132850646973 seconds ---
================================================================================
====
n_clusters =  5
================================================================================
====
For n_clusters = 5 The average silhouette_score is : 0.2127806695469617
For n_clusters = 5 The average SSE is : 186.38571116085453
--- 8.309257984161377 seconds ---
================================================================================
====
n_clusters =  6
================================================================================
====
For n_clusters = 6 The average silhouette_score is : 0.20436498901055802
For n_clusters = 6 The average SSE is : 168.10840550212362
--- 9.070887804031372 seconds ---
```

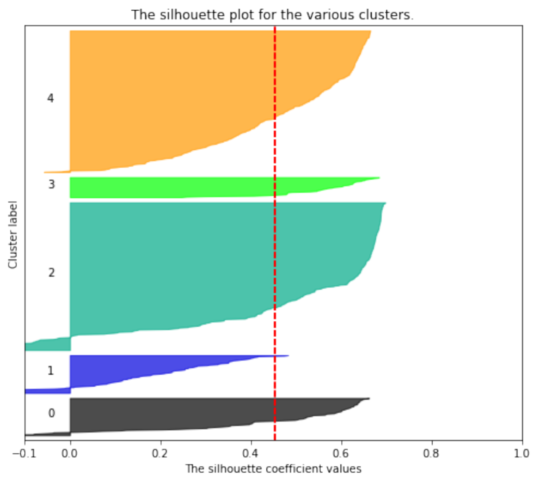


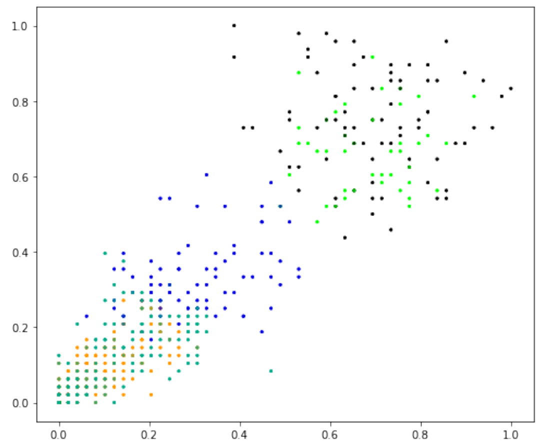

The silhouette plot for the various clusters.
Cluster label
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

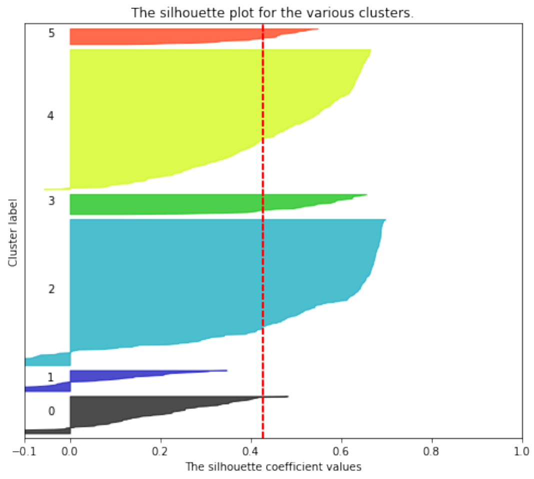

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

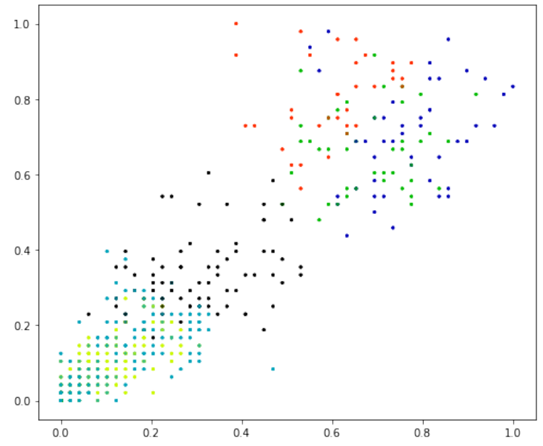

The silhouette plot for the various clusters.
Cluster label
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

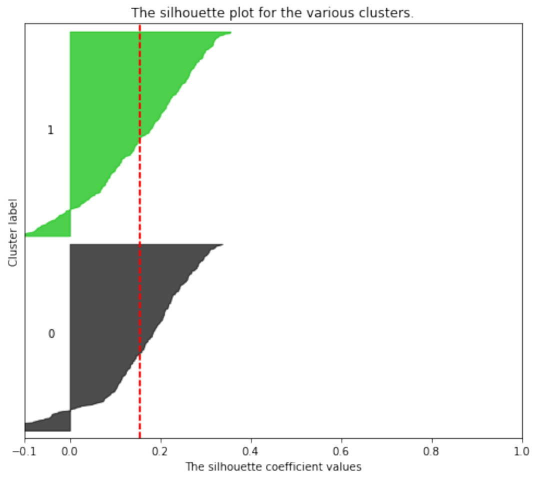

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

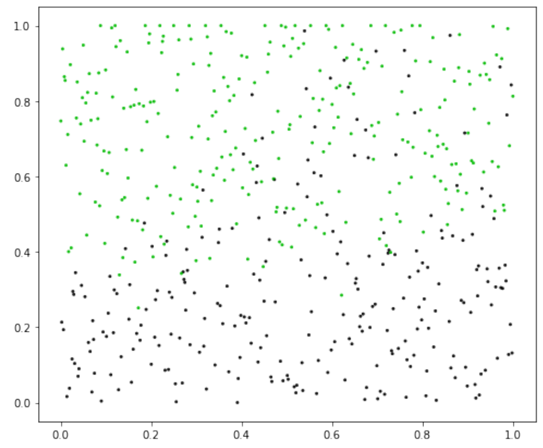

The silhouette plot for the various clusters.
Cluster label
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

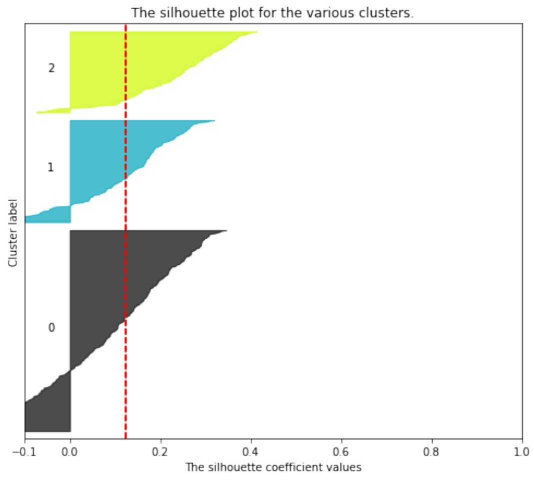

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

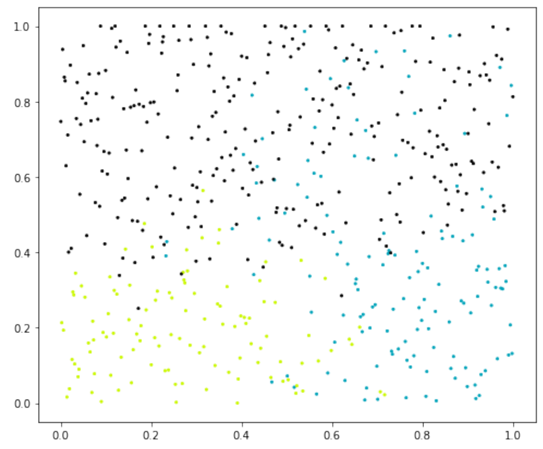

The silhouette plot for the various clusters.
Cluster label
5
4
3
2
1
0
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

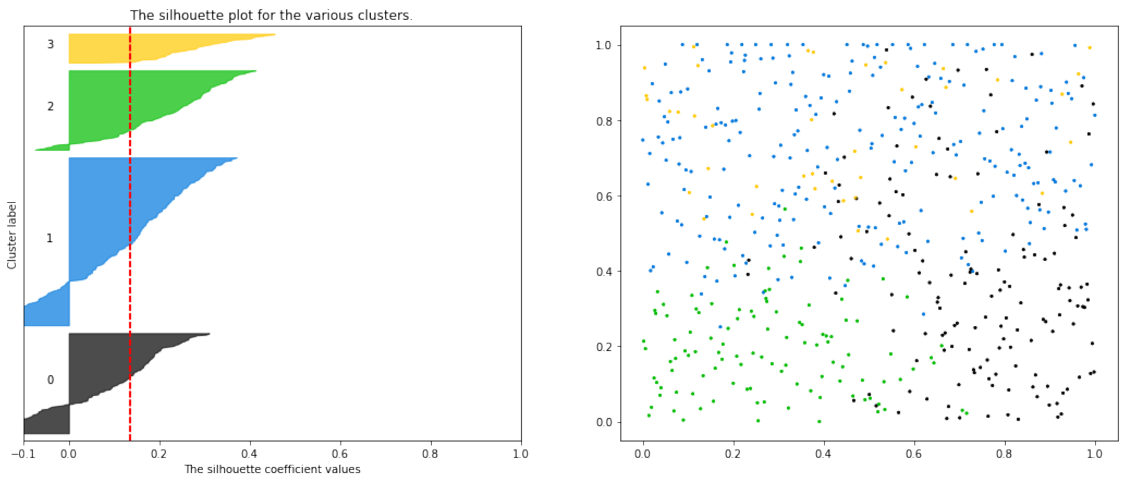

0.0
0.2
0.4
0.6
0.8
1.0


The silhouette plot for the various clusters.
Cluster label
1
0
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

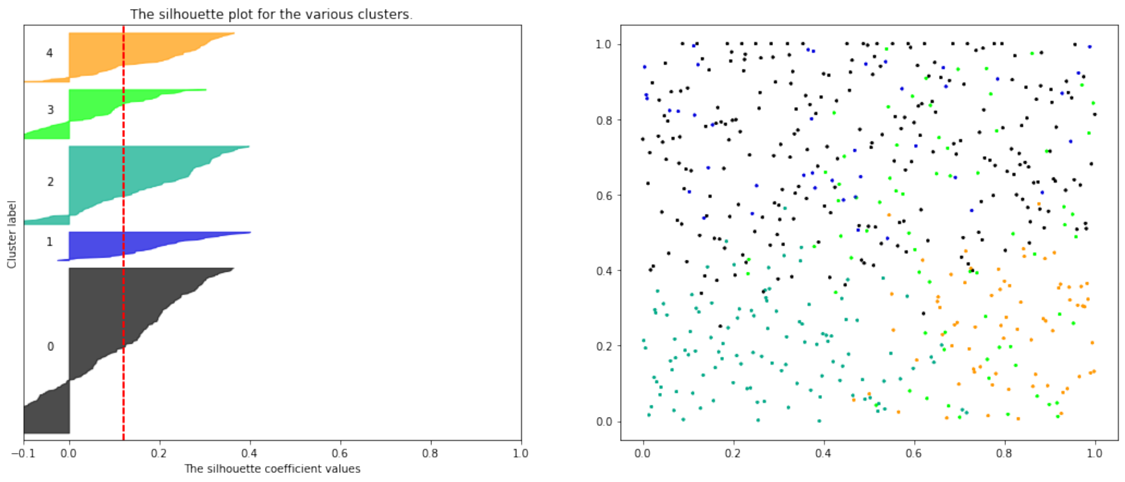

0.0
0.2
0.4
0.6
0.8
1.0


The silhouette plot for the various clusters.
Cluster label
2
1
0
The silhouette coefficient values
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

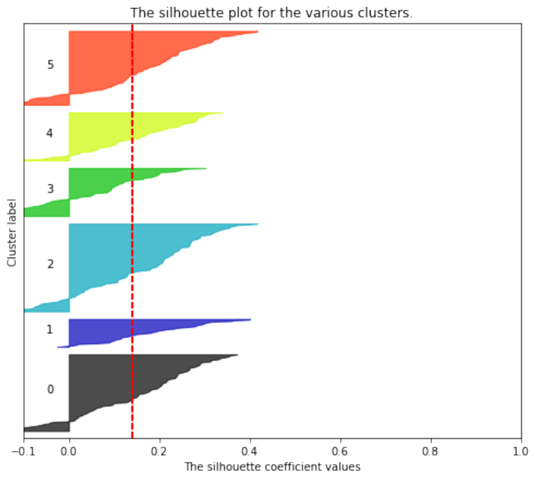

0.0
0.2
0.4
0.6
0.8
1.0

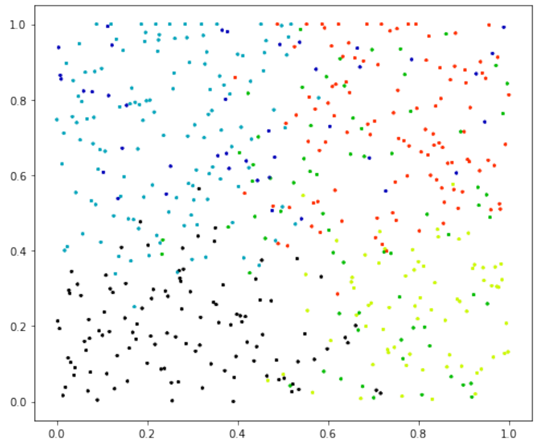


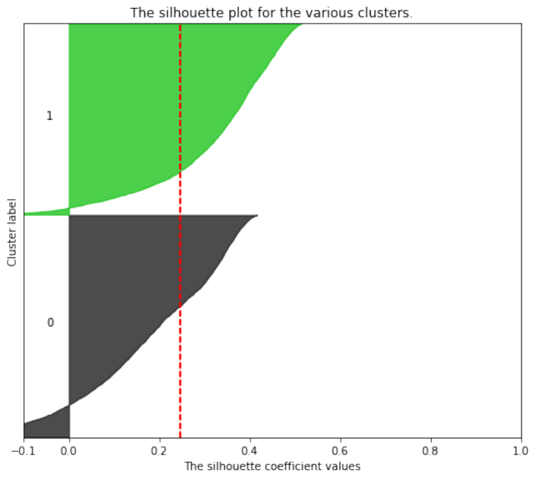
The silhouette plot for the various clusters.
Cluster label
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

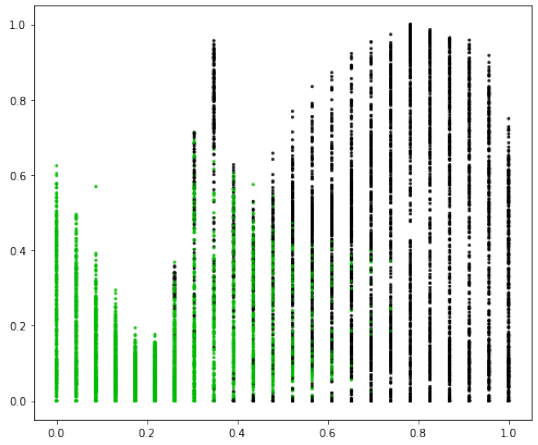


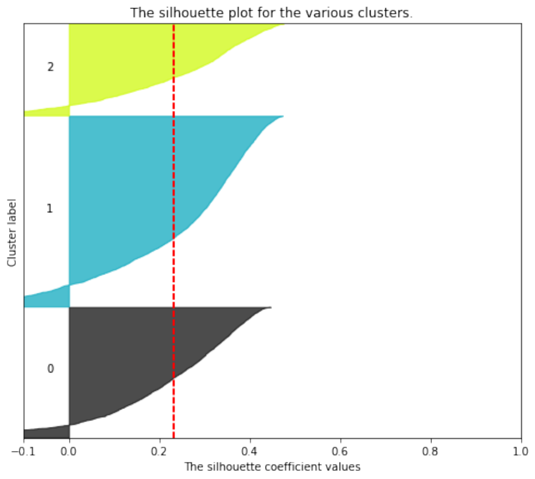
The silhouette plot for the various clusters.
Cluster label
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

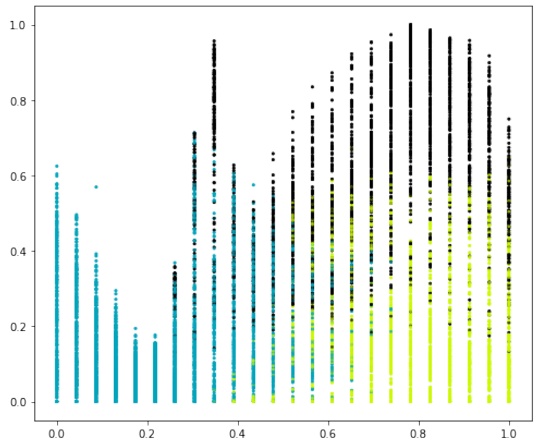

The silhouette plot for the various clusters.
Cluster label
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

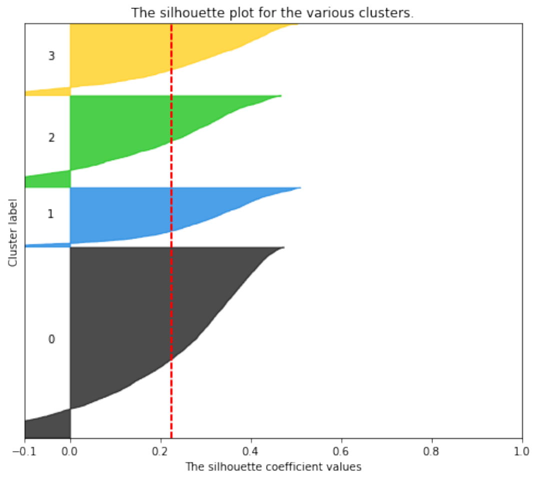

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

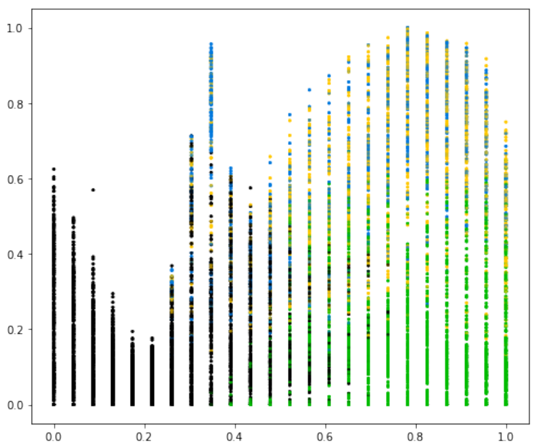

The silhouette plot for the various clusters.
Cluster label
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

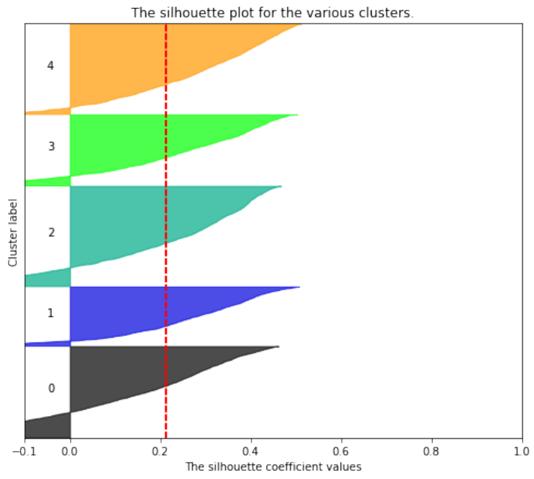

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

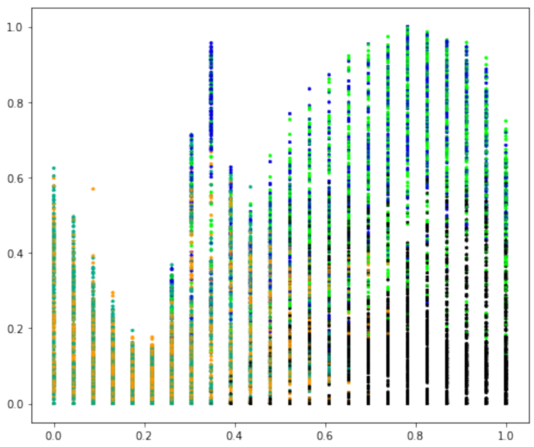

The silhouette plot for the various clusters.
Cluster label
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

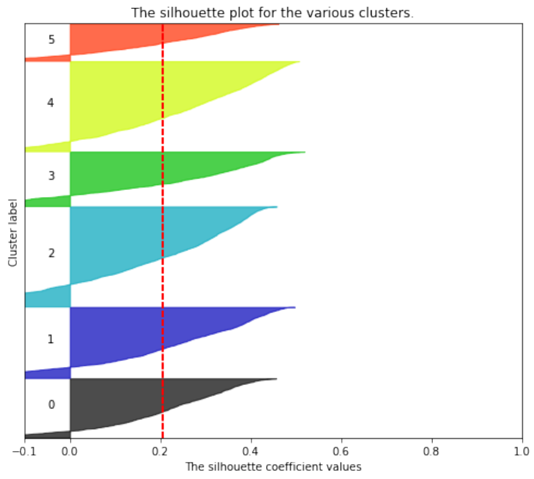

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

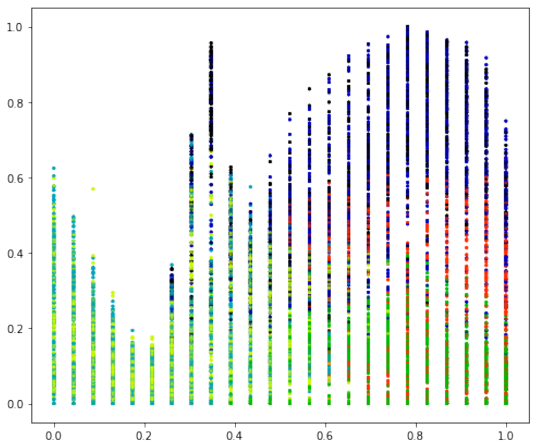

The silhouette plot for the various clusters.
Cluster label
3
2
1
0
The silhouette coefficient values




The silhouette plot for the various clusters.
Cluster label
4
3
2
1
0
The silhouette coefficient values




The silhouette plot for the various clusters.
Cluster label
5
4
3
2
1
0
The silhouette coefficient values



```
[ ]:
```