Demonstrating Clustering using K-Means Algorithm

Azhar Chowdhury
Statistical Programmer|Data Scientist[SAS, Python, R]

The following meta summary is helpful to follow the problem at hand and it's solution.

Dataset The Portugal Electricity Consumption dataset

Domain Time Series

ML Algorithm K -Means Clustering

Components Cluster. Silhouette Score

Problem Description Perform clustering using K-Means algorithm, Itentify the number of clusters for a given dataset

[1]:
```
import os
os.environ["OMP_NUM_THREADS"] = "1"
```

The above code will prevent from over-consuming CPU or memory, or when we are getting OpenMP-related warnings, setting $OMP\_NUM\_THREADS = 1$ as a safe baseline - as I am running PY programs on Windows 11.

[2]:
```
pathToFile = r'C://Users//azhar//Downloads//'
fileName = 'LD2011_2014.txt'

import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import random
from sklearn.metrics import silhouette_score
from sklearn.cluster import AgglomerativeClustering
random.seed(42)
```

[3]:
```
# Replace "," by ".", otherwise the numbers will be in the form 2,3445 instead␣
↪of 2.3445
import fileinput

with fileinput.FileInput(pathToFile+fileName, inplace=True, backup='.bak') as␣
↪file:
    for line in file:
```

```
        print(line.replace(",", "."), end='')
```

[4]:
```
import pandas as pd
data = pd.read_csv(pathToFile+fileName, sep=";", index_col=0)
```

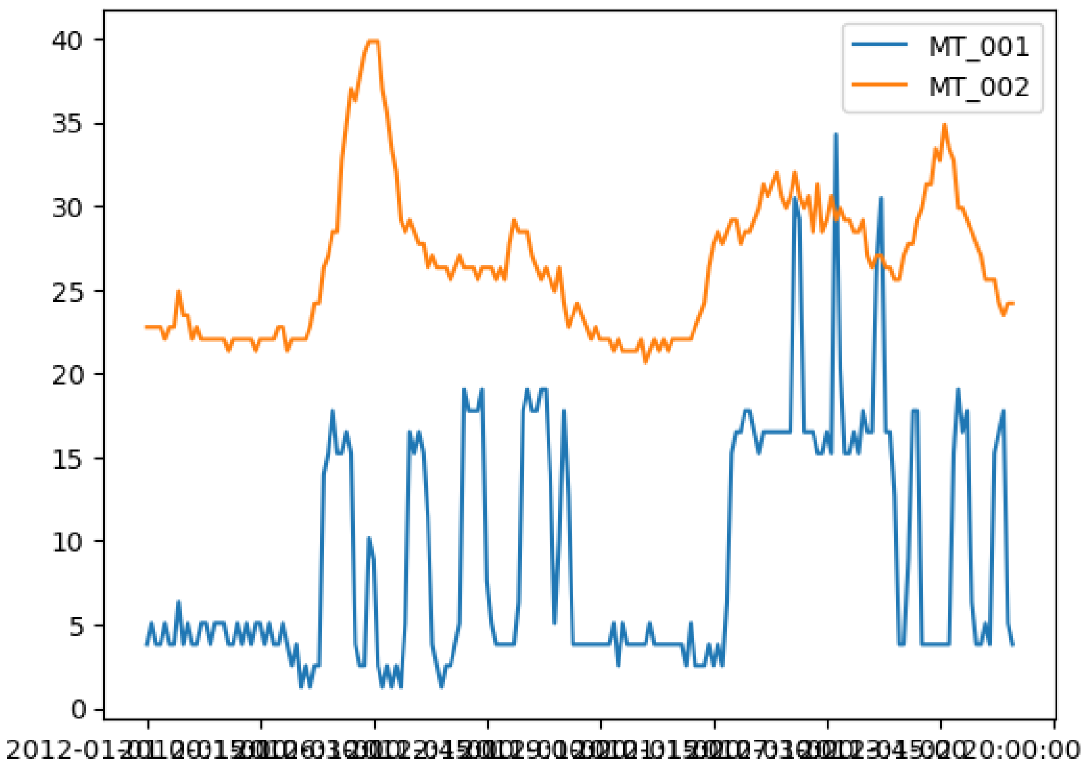
[5]:
```
data.head(2)
data.tail(2)
data.shape
data.info()
data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 140256 entries, 2011-01-01 00:15:00 to 2015-01-01 00:00:00
Columns: 370 entries, MT_001 to MT_370
dtypes: float64(370)
memory usage: 397.0+ MB
```

[5]:
```
              MT_001         MT_002         MT_003         MT_004  \
count  140256.000000  140256.000000  140256.000000  140256.000000
mean        3.970785      20.768480       2.918308      82.184490
std         5.983965      13.272415      11.014456      58.248392
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       2.844950       0.000000      36.585366
50%         1.269036      24.893314       1.737619      87.398374
75%         2.538071      29.871977       1.737619     115.853659
max        48.223350     115.220484     151.172893     321.138211

              MT_005         MT_006         MT_007         MT_008  \
count  140256.000000  140256.000000  140256.000000  140256.000000
mean       37.240309     141.227385       4.521338     191.401476
std        26.461327      98.439984       6.485684     121.981187
min         0.000000       0.000000       0.000000       0.000000
25%        15.853659      71.428571       0.565291     111.111111
50%        39.024390     157.738095       2.826456     222.222222
75%        54.878049     205.357143       4.522329     279.461279
max       150.000000     535.714286      44.657999     552.188552

              MT_009         MT_010  ...         MT_361         MT_362  \
count  140256.000000  140256.000000  ...  140256.000000  140256.000000
mean       39.975354      42.205152  ...     218.213701   37607.987537
std        29.814595      33.401251  ...     204.833532   38691.954832
min         0.000000       0.000000  ...       0.000000       0.000000
25%        13.986014       9.677419  ...       5.710207       0.000000
50%        40.209790      40.860215  ...     131.334761   24100.000000
75%        57.692308      61.290323  ...     403.283369   54800.000000
max       157.342657     198.924731  ...     852.962170  192800.000000

              MT_363         MT_364         MT_365         MT_366  \
```

```
count  140256.000000  140256.000000  140256.000000  140256.000000
mean     1887.427366    2940.031734      65.413150       9.269709
std      1801.486488    2732.251967      65.007818      10.016782
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000      13.037810       0.000000
50%      1050.632911    2136.363636      31.290743       7.021650
75%      3312.236287    5363.636364     108.213820      11.702750
max      7751.054852   12386.363636     335.071708      60.269163

              MT_367         MT_368         MT_369         MT_370
count  140256.000000  140256.000000  140256.000000  140256.000000
mean      424.262904      94.704717     625.251734    8722.355145
std       274.337122      80.297301     380.656042    9195.155777
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000      30.050083      83.944282       0.000000
50%       525.899912      76.794658     758.064516       0.000000
75%       627.743635     151.919866     875.366569   17783.783784
max      1138.718174     362.270451    1549.120235   30918.918919

[8 rows x 370 columns]
```

[6]:
```
data_example = data.loc['2012-01-01 00:15:00':'2012-01-03 00:00:
↪00'][['MT_001','MT_002']]
data_example.plot()
plt.show()
```




```
[7]: data2011 = data.loc['2011-01-01 00:15:00':'2012-01-01 00:00:00']
     data2012 = data.loc['2012-01-01 00:15:00':'2013-01-01 00:00:00']
     data2013 = data.loc['2013-01-01 00:15:00':'2014-01-01 00:00:00']
     data2014 = data.loc['2014-01-01 00:15:00':'2015-01-01 00:00:00']
```

```
[8]: # Check number of days
     print(data2011.shape[0]/96)
     print(data2012.shape[0]/96)
     print(data2013.shape[0]/96)
     print(data2014.shape[0]/96)
```

```
365.0
366.0
365.0
365.0
```

```
[9]: # See number of clients with 0 demand per year
     print(sum(data2011.mean()==0))
     print(sum(data2012.mean()==0))
     print(sum(data2013.mean()==0))
     print(sum(data2014.mean()==0))
```

```
210
```

```
37
21
1
```

[10]:
```python
import pandas as pd
clients = data2011.columns
clients_no_demand = clients[data2013.mean()==0] # clients with 0 demand
#data_13_14 = data2013.concat(data2014) # appending 2013 and 2014
data_13_14 = pd.concat([data2013, data2014], axis=0)
data_13_14 = data_13_14.drop(clients_no_demand, axis=1) # drop clients with 0␣
↪demand
print(data_13_14.shape)
print(sum(data_13_14.mean()==0)) # check that there are no clients with 0 demand
```

```
(70080, 349)
0
```

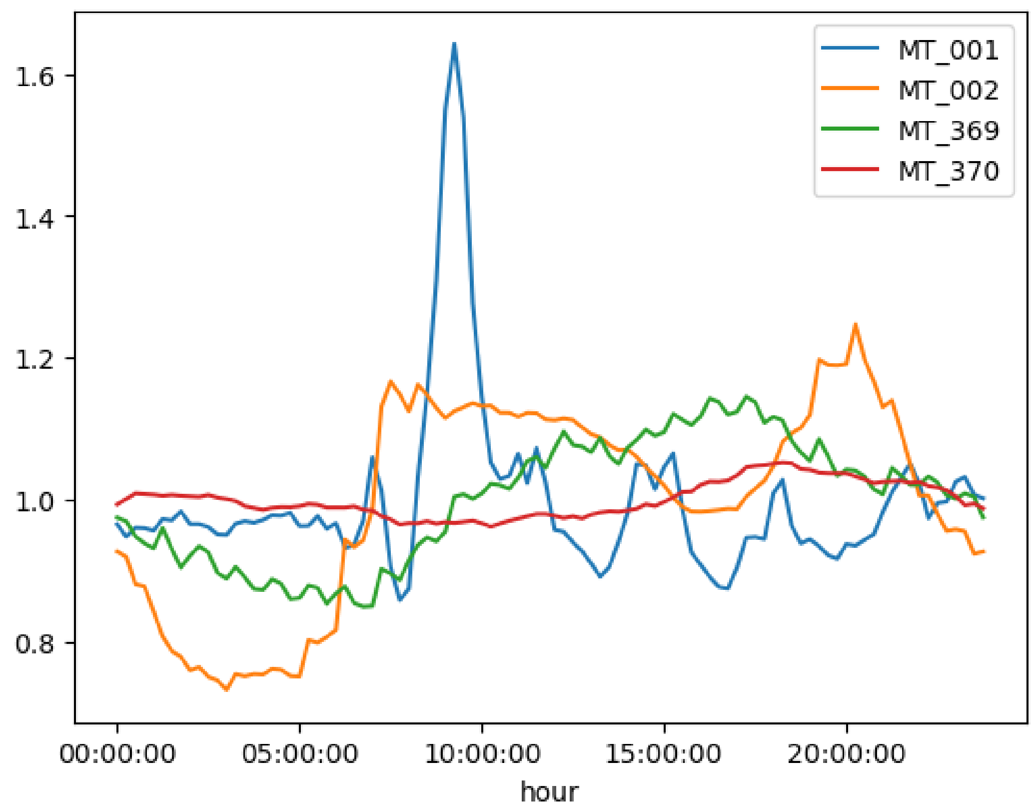
[11]:
```python
data = data_13_14.copy() # weekdays weekends, data2011, data2012, data2013,␣
↪data2014
data['hour'] = data.index.map(lambda x: x[11:])
data.head(5)
```

[11]:
```
                        MT_001     MT_002    MT_003      MT_004     MT_005  \
2013-01-01 00:15:00  2.538071  22.759602  2.606429  138.211382  63.414634
2013-01-01 00:30:00  1.269036  22.759602  2.606429  138.211382  63.414634
2013-01-01 00:45:00  2.538071  22.759602  2.606429  134.146341  60.975610
2013-01-01 01:00:00  1.269036  23.470839  2.606429  130.081301  56.097561
2013-01-01 01:15:00  3.807107  23.470839  2.606429  130.081301  58.536585

                         MT_006    MT_007      MT_008     MT_009     MT_010  \
2013-01-01 00:15:00  255.952381  4.522329  239.057239  57.692308  78.494624
2013-01-01 00:30:00  264.880952  5.652911  228.956229  57.692308  76.344086
2013-01-01 00:45:00  250.000000  5.652911  239.057239  54.195804  76.344086
2013-01-01 01:00:00  226.190476  6.218202  249.158249  50.699301  75.268817
2013-01-01 01:15:00  229.166667  6.783493  239.057239  57.692308  74.193548

                     ...   MT_362      MT_363       MT_364     MT_365  \
2013-01-01 00:15:00  ...  22300.0  886.075949  1000.000000  16.949153
2013-01-01 00:30:00  ...  21000.0  864.978903   909.090909  18.252934
2013-01-01 00:45:00  ...  18200.0  860.759494   840.909091  16.949153
2013-01-01 01:00:00  ...  15800.0  860.759494   840.909091  16.949153
2013-01-01 01:15:00  ...  15000.0  793.248945   818.181818  16.949153

                        MT_366      MT_367     MT_368      MT_369       MT_370  \
2013-01-01 00:15:00  6.436513  616.330114  76.794658  731.671554  8086.486486
2013-01-01 00:30:00  3.510825  564.530290  76.794658  727.272727  8086.486486
2013-01-01 00:45:00  5.851375  590.869183  68.447412  730.205279  7848.648649
2013-01-01 01:00:00  4.095963  575.065847  58.430718  722.873900  7848.648649
```

```
2013-01-01 01:15:00  4.095963  570.676032  60.100167  748.533724  7610.810811

                         hour
2013-01-01 00:15:00  00:15:00
2013-01-01 00:30:00  00:30:00
2013-01-01 00:45:00  00:45:00
2013-01-01 01:00:00  01:00:00
2013-01-01 01:15:00  01:15:00

[5 rows x 350 columns]
```

[12]:

```
# Getting average curves per client
datagrouped = data.groupby("hour")
average_curves = datagrouped.agg("mean")
average_curves.shape
average_curves_norm = average_curves/(average_curves.mean())

average_curves_norm[['MT_001','MT_002','MT_369','MT_370']].plot()
plt.show()
```



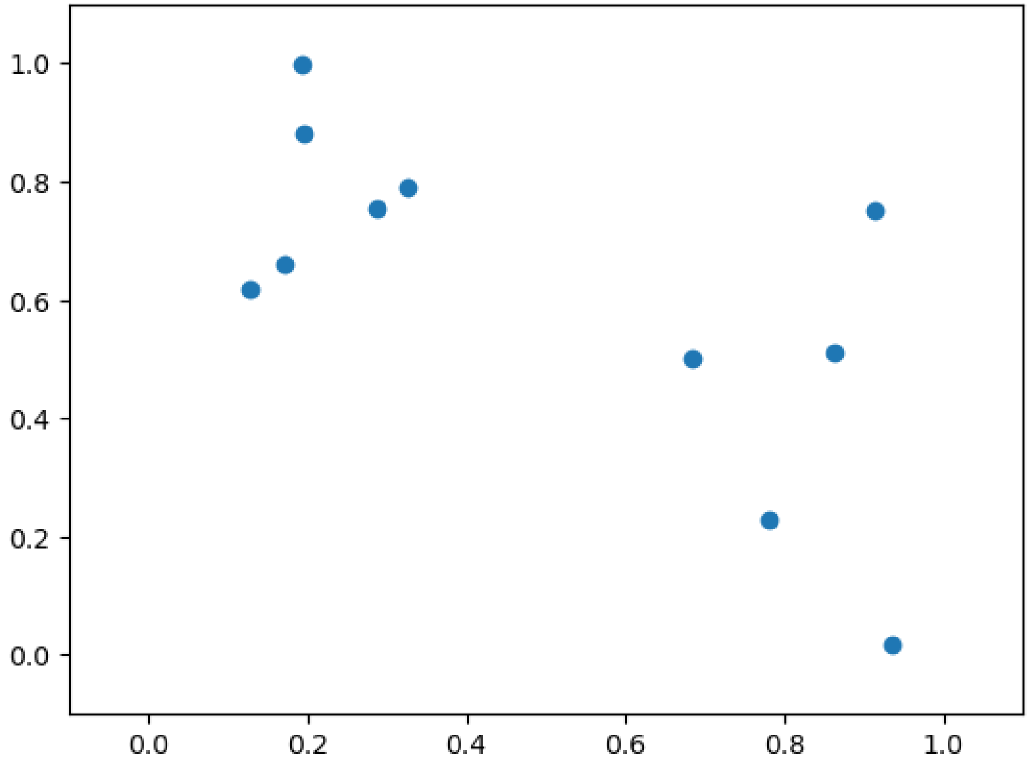
[13]:

```
client = 'MT_022'
oneClient = data_13_14[client]
X = [] # a list of arrays, each array being a normalized curve for a day
for J in range(2*365):
    X.extend([np.array(oneClient[J*96:(J+1)*96])])
```

[14]:

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import random
# from sklearn.metrics import silhouette_score
# from sklearn.cluster import AgglomerativeClustering
random.seed(43)

trueK = 11 # Desired number of clusters
dim = 2 # dimension of feature space
#n = 40 # points per cluster
trueCentroids = np.random.rand(trueK, dim) # generating centroids at random
print(trueCentroids)

if dim==2:
    plt.scatter(trueCentroids[:,0],trueCentroids[:,1],)
    plt.xlim(-0.1,1.1)
    plt.ylim(-0.1,1.1)
    plt.show()
```

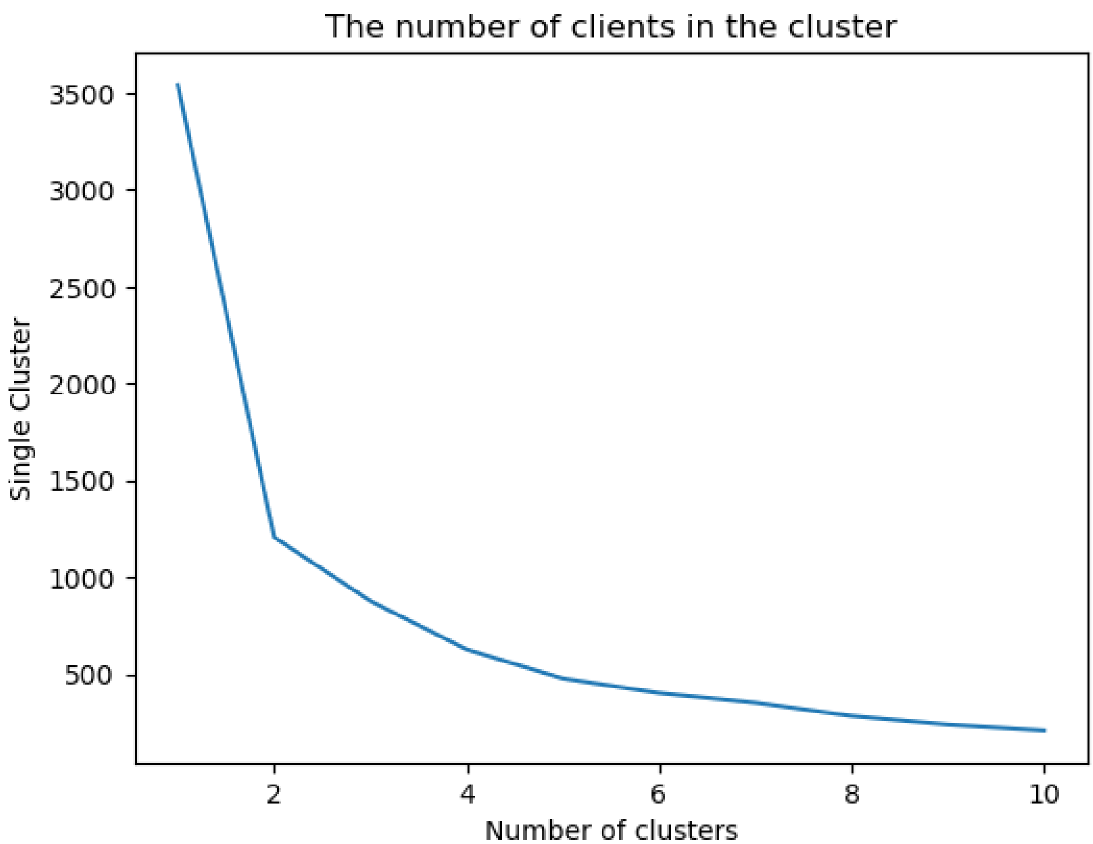
```
[[0.68350152 0.50044166]
 [0.32660083 0.79046578]
 [0.78053065 0.22799233]
 [0.12871697 0.61836057]
 [0.17087974 0.66002271]
 [0.93675917 0.01834244]
 [0.9139862  0.75203116]
 [0.28655132 0.75612555]
 [0.8641899  0.51231176]
 [0.196591   0.88015142]
 [0.19226105 0.99728998]]
```



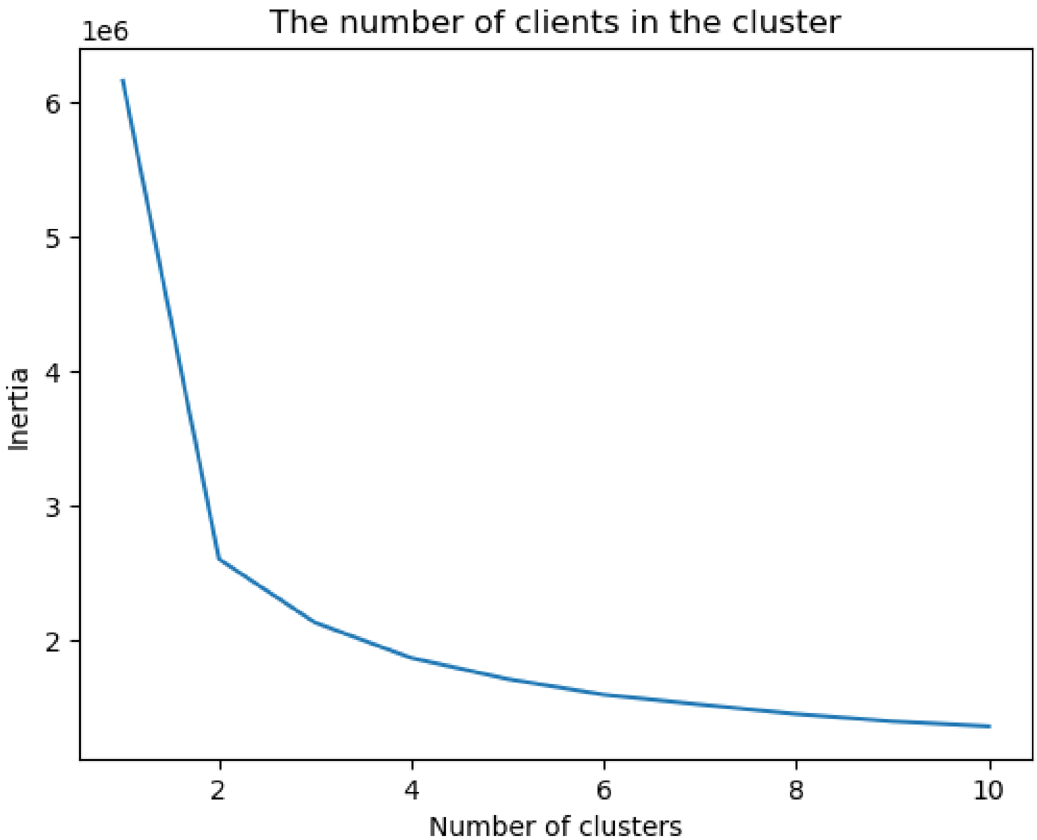
[22]:
```
import os
import numpy as np
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=trueK, n_init=10, random_state=43).fit(X)

print(kmeans.inertia_)
```

```
179.30226039855134
```

[23]:
```
from sklearn.cluster import KMeans
_clusters = []
for i in range(1, trueK):
    kmeans = KMeans(n_clusters = i, n_init=10, init = 'k-means++', random_state␣
 ↪= 43)
    kmeans.fit(X)
    _clusters.append(kmeans.inertia_)

plt.title("The number of clients in the cluster")
plt.plot(range(1, 11), _clusters,)
plt.xlabel('Number of clusters')
plt.ylabel('Single Cluster')
plt.show()
```



[18]:
```
import os
os.environ["OMP_NUM_THREADS"] = "1"  # Avoid MKL memory leak warning on Windows

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

_clusters = []
trueK = 11  # Make sure this is defined appropriately
for i in range(1, trueK):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=43, n_init=10) ␣
↪# Set n_init explicitly
    kmeans.fit(X)
    _clusters.append(kmeans.inertia_)

plt.title("The number of clients in the cluster")
plt.plot(range(1, trueK), _clusters)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



To determine a number of clusters we guess the number of clusters. In a day of 24 hours, usage may vary. Lets assume that the usage varies from 1 to 11 hours in a day. If we fit the KMeans object with the number of cluster (11) and request for inertia_ (KMeans.inertia_) for each cluster we'll get 11 intertia values. Then if we plot them the graph will reveal an elbow shape. The point where the elbow is determined is our desired number of clusters. In our case it is 2.

[19]:
```
from sklearn import metrics
from sklearn.metrics import pairwise_distances

X = average_curves_norm.copy()
```

[26]:
```
import numpy as np
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2,n_init=10,  random_state=1).fit(X)
labels = kmeans.labels_
metrics.silhouette_score(X, labels, metric='euclidean')
```

[26]:
```
0.5676594904032211
```

A Silhouette score close to 1 is a well spreaded cluster. Since our score is positive and close to 1 it is well spreaded, non-overlapping and observations do not seem to be in a wrong cluster. Comparing to inertia score, it is clear that I was not able to set the data properly fit to the model and hence the intertia was not calculated correctly. However, if it were calculated correctly Silhouette method is a better evaluator.