Assignment 2 - Clustering

Negin Baghbanzadeh

Learning Outcomes

In this assignment, you will do the following:

- Explore a dataset and carry out clustering using k-means algorithm
- Identify the optimum number of clusters for a given dataset

Problem Description

In this assignment, you will study the electricity demand from clients in Portugal, during 2013 and 2014. You have been provided with the data file, which you should download when you download this assignment file.

The data[1] available contains 370 time series, corresponding to the electric demand[2] for 370 clients, between 2011 and 2014.

In this guided exercise, you will use clustering techniques to understand the typical usage behaviour during 2013-2014.

Both these datasets are publicly available, and can be used to carry out experiments. Their source is below:

1. Data: https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014# (https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014#)
2. Electric Demand: http://www.think-energy.net/KWvsKWH.htm (http://www.think-energy.net/KWvsKWH.htm)

We will start by exploring the data set and continue on to the assignment. Consider this as a working notebook, you will add your work to the same notebook.

In this assignment we will use the sklearn package for k-means. Please refer here for the documentation https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html) (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)).

The sklearn package for k-means is one of the many clustering algorithms found in the module "sklearn.cluster". These come with a variety of functions that you can call by importing the package.

For example

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans
```

Data Preparation

Start by downloading the data to a local directory and modify the "pathToFile" and "fileName" variables, if needed. The data file has been provided with this assignment. It is also available at the links provided above.

In [1]:
```
pathToFile = r"/Users/Negin/Downloads/assignment2-data/"

fileName = 'LD2011_2014.txt'
```

In [2]:
```
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(42)
```

In [3]:
```
# Replace "," by ".", otherwise the numbers will be in the form 2,
import fileinput

with fileinput.FileInput(pathToFile+fileName, inplace=True, backup
    for line in file:
        print(line.replace(",", "."), end='')
```

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

Quick data inspection

In [5]:
```
data.head(2)
```

Out[5]:

	MT_001	MT_002	MT_003	MT_004	MT_005	MT_006	MT_007	MT_008	MT_009	MT
2011-01-01 00:15:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2011-01-01 00:30:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

2 rows × 370 columns

In [6]:
```
data.tail(2)
```

Out[6]:

	MT_001	MT_002	MT_003	MT_004	MT_005	MT_006	MT_007	MT_0
2014-12-31 23:45:00	1.269036	21.337127	1.737619	166.666667	85.365854	285.714286	10.17524	225.5892
2015-01-01 00:00:00	2.538071	19.914651	1.737619	178.861789	84.146341	279.761905	10.17524	249.1582

2 rows × 370 columns

In [7]:
```
data.shape
```

Out[7]:
```
(140256, 370)
```

As it can be seen, the dataframe contains a row for each interval of 15 minutes between Jan 1, 2011 to Dec 31 2014. There are 370 columns corresponding 370 clients. The dataframe is indexed by the timestamp.

Since the frequency is 15 minutes, each day provides $24 \times 4 = 96$ datapoints, which multiplied by 365 days and 4 years (plus 1 day in Feb 29, 2012) gives:
$96 \times 365 \times 4 + 96 = 140256$, as observed in data.shape

In [8]:
```
data.info()
```

```
<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
```

In [9]:
```
data.describe()
```

Out[9]:

	MT_001	MT_002	MT_003	MT_004	MT_005	MT_
count	140256.000000	140256.000000	140256.000000	140256.000000	140256.000000	140256.000
mean	3.970785	20.768480	2.918308	82.184490	37.240309	141.227
std	5.983965	13.272415	11.014456	58.248392	26.461327	98.439
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.000000	2.844950	0.000000	36.585366	15.853659	71.428
50%	1.269036	24.893314	1.737619	87.398374	39.024390	157.738
75%	2.538071	29.871977	1.737619	115.853659	54.878049	205.357
max	48.223350	115.220484	151.172893	321.138211	150.000000	535.714

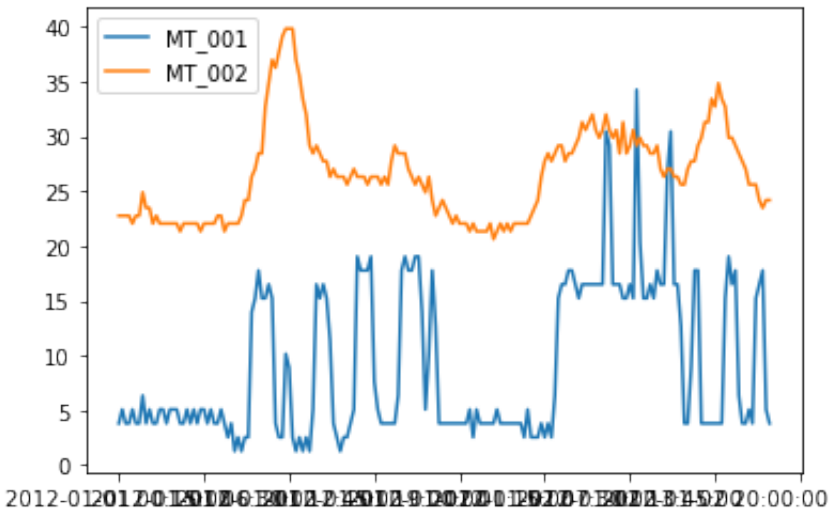
8 rows × 370 columns

In [10]:
```
data.isna().any().all()
```

Out[10]: False

Plot the first 2 days of 2012 for the first 2 clients

In [11]:
```
data_example = data.loc['2012-01-01 00:15:00':'2012-01-03 00:00:00
data_example.plot()
plt.show()
```



We note that the main difference between the curves is the level (also seen on the means in df.describe()). We know we will have to somehow normalize the curves, in order for the clustering technique to capture the behaviour of the consumption throughout the day.

Data for the analysis

We focus on 2013 and 2014 because these are the years with low number of clients having zero demand

In [12]:
```
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']
```

In [13]:
```
# 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
```

In [14]:
```
# 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
```

In [15]:
```
clients = data2011.columns
clients_no_demand = clients[data2013.mean()==0] # clients with 0 c
data_13_14 = data2013.append(data2014) # appending 2013 and 2014
data_13_14 = data_13_14.drop(clients_no_demand, axis=1) # drop cli
print(data_13_14.shape)
print(sum(data_13_14.mean()==0)) # check that there are no clients
```

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

Getting average curves per client

In [16]:
```
data = data_13_14.copy() # weekdays weekends, data2011, data2012,
```

In [17]:
```
data['hour'] = data.index.map(lambda x: x[11:])
```

In [18]:
```
data.head(3)
```

Out[18]:

	MT_001	MT_002	MT_003	MT_004	MT_005	MT_006	MT_007	MT_0
2013-01-01 00:15:00	2.538071	22.759602	2.606429	138.211382	63.414634	255.952381	4.522329	239.0572
2013-01-01 00:30:00	1.269036	22.759602	2.606429	138.211382	63.414634	264.880952	5.652911	228.9562
2013-01-01 00:45:00	2.538071	22.759602	2.606429	134.146341	60.975610	250.000000	5.652911	239.0572

3 rows × 350 columns

Getting average curves per client

In [19]:
```
datagrouped = data.groupby("hour")
average_curves = datagrouped.agg("mean")
average_curves.shape
```

Out[19]: (96, 349)

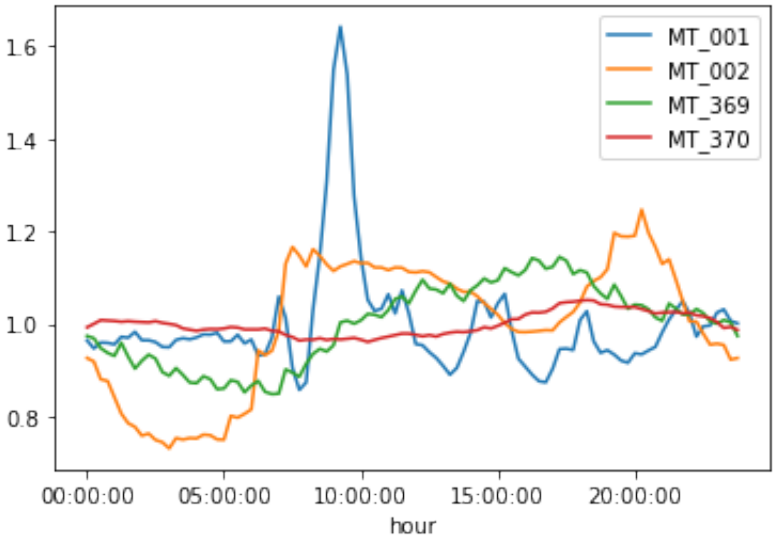
The dataframe average_curves contains the 349 typical weekday electric demands

We now divide each curve by its mean, so that all curves have mean 1

In [20]:
```
average_curves_norm = average_curves/(average_curves.mean())
```

Plot the first 2 and last 2 clients

In [21]:
```
average_curves_norm[['MT_001','MT_002','MT_369','MT_370']].plot()
plt.show()
```



Clustering Analysis on the average normalized curves

In [22]:
```
X = average_curves_norm.copy()  # We call this normalized curve
X = np.array(X.T) # put it on the right format
```

Questions (15 marks total)

Q1: (7 marks)

a. Determine what a convenient number of clusters. Justify your choice. Make use of the sklearn's package for k-means for this. You may refer to the module to figure out how to come up with the optimal number of clusters.

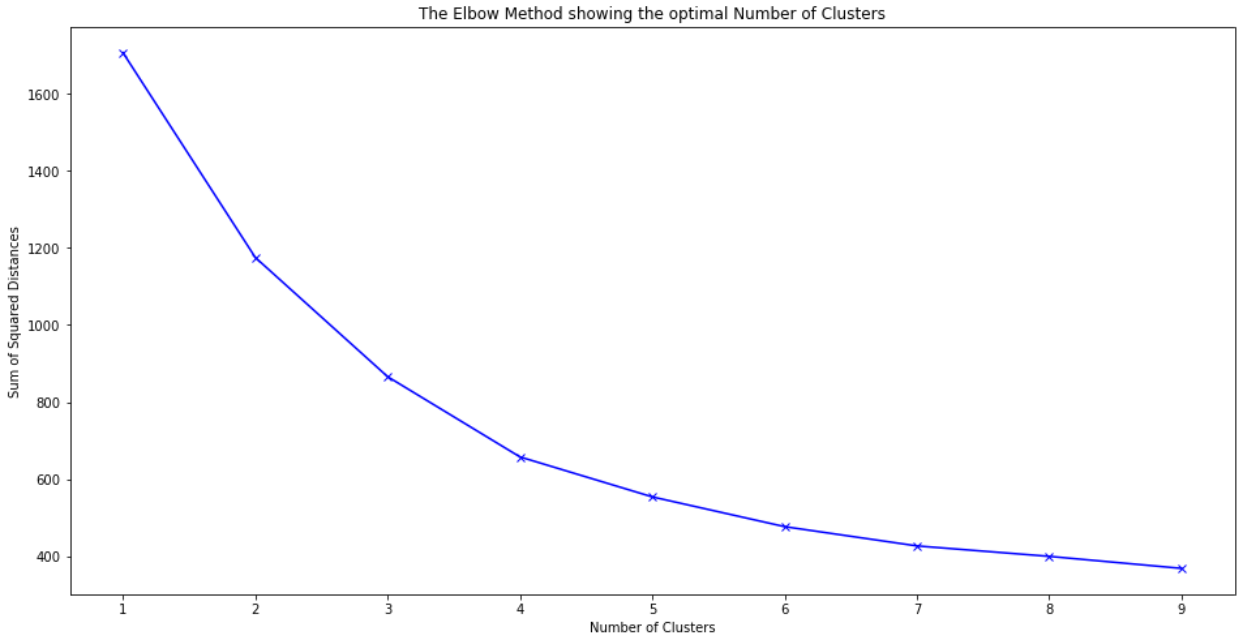
b. Make a plot for each cluster, that includes: - The number of clients in the cluster (you can put this in the title of the plot) - All the curves in the cluster - The curve corresponding to the center of the cluster (make this curve thicker to distinguish it from the individual curves). The center is also sometimes referred to as "centroid".

You have 2 separate plots for each cluster if you prefer (one for the individual curves, one for the centroid)

Q1 - Part a

In [23]:
```
sum_squared_dists = []
number_of_clusters = range(1,10)
for k in number_of_clusters:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(X)
    sum_squared_dists.append(kmeanModel.inertia_)
```

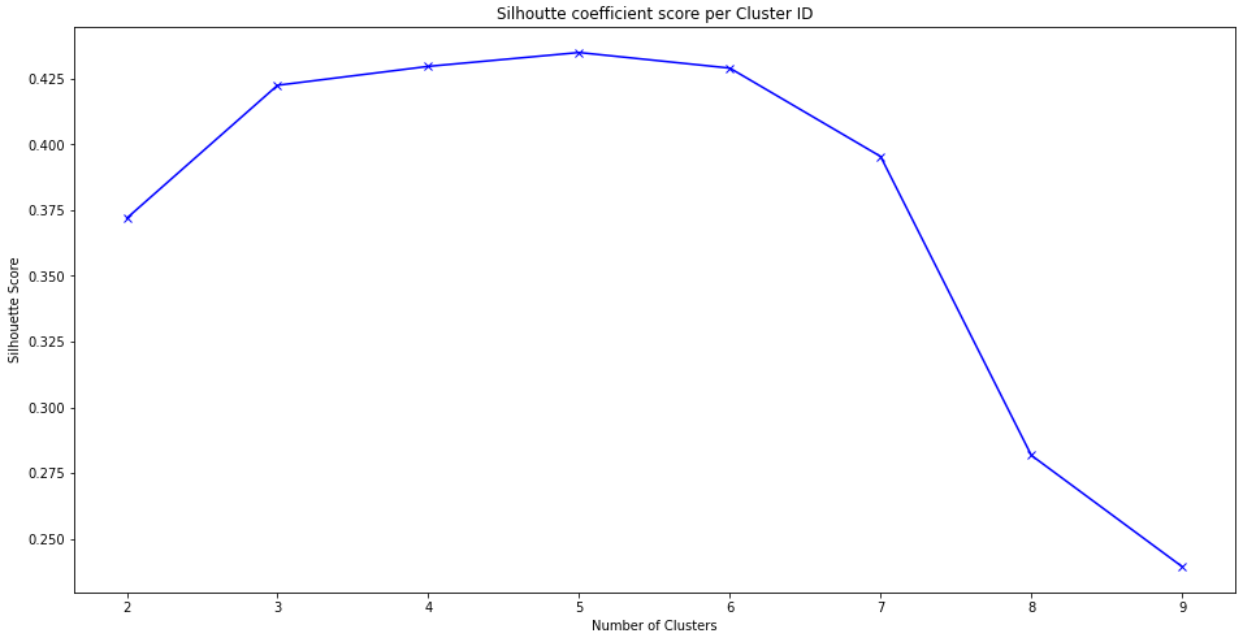
In [24]:
```
plt.figure(figsize=(16,8))
plt.plot(number_of_clusters, sum_squared_dists, 'bx-')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Distances')
plt.title('The Elbow Method showing the optimal Number of Clusters
plt.show()
```



Based on the elbow graph above, having 6 clusters, doesn't make the much diffrence in making sum of squared distaances less than 7 clusters (making the model better). So a convenient number of clusters is **4** clusters.

In [25]:
```
from sklearn import metrics
silhouette_score = []
number_of_clusters = range(2,10)
for k in number_of_clusters:
    kmeans_model_one_client = KMeans(n_clusters=k).fit(X)
    labels = kmeans_model_one_client.labels_
    silhouette_score.append(metrics.silhouette_score(X, labels, me
```

In [26]:
```
plt.figure(figsize=(16,8))
plt.plot(number_of_clusters, silhouette_score, 'bx-')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhoutte coefficient score per Cluster ID')
plt.show()
```



The Silhoutte coefficient score confirm k=5 too.

Q1 - Part b

In [27]:
```
optimal_clusters = 5
optimal_model = KMeans(n_clusters=optimal_clusters).fit(X)
y_pred = optimal_model.labels_
```

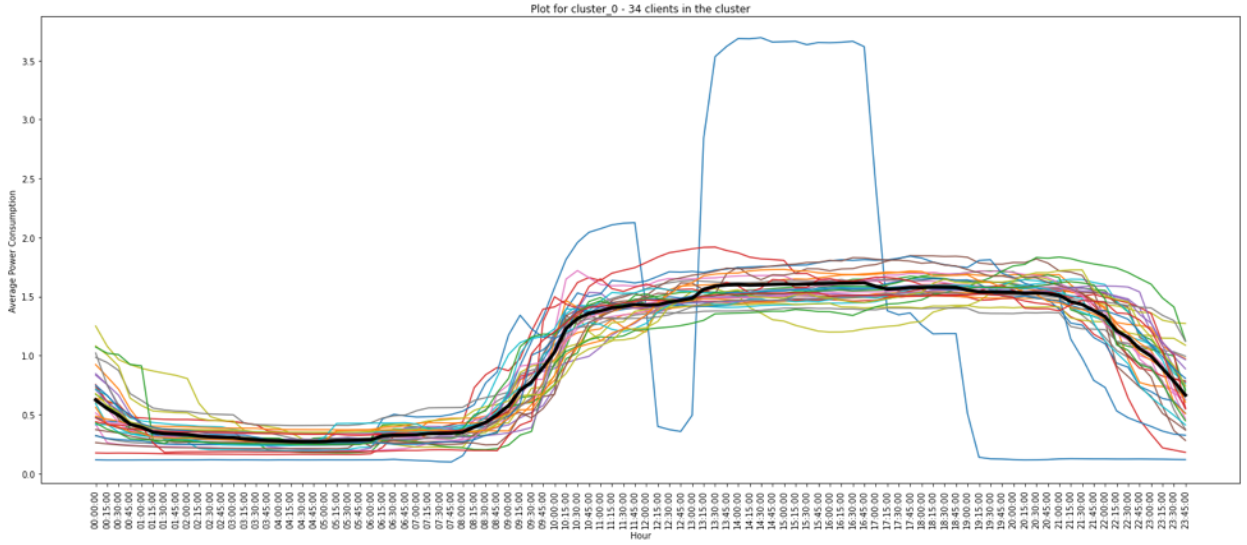
In [28]:
```
from datetime import datetime

hours = average_curves_norm.index.map(lambda x: datetime.strptime(
```

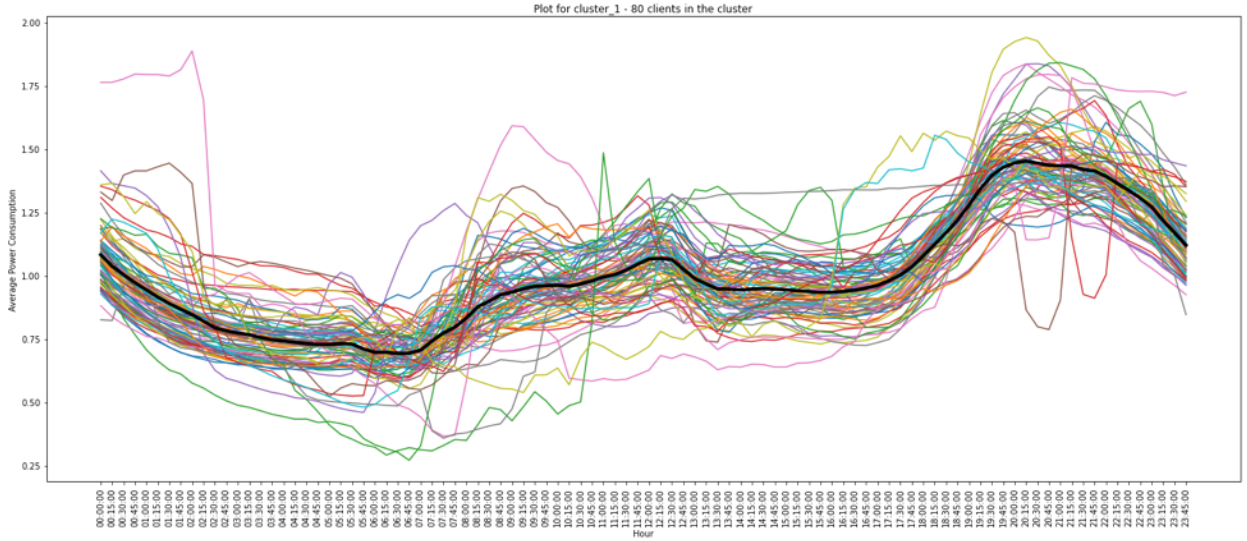
In [29]:
```
# Finding the data in each cluster
clusters = []
for i in range(optimal_clusters):
    clusters.append(X[y_pred == i])
```

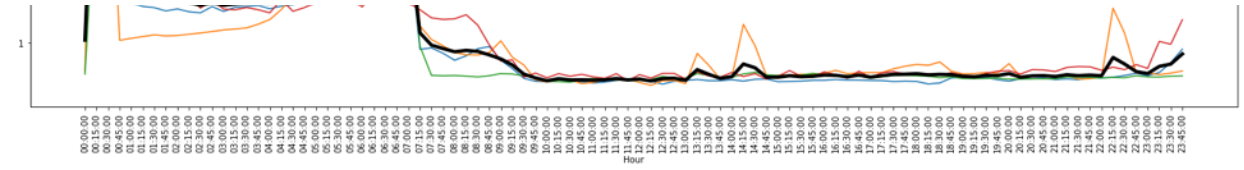
The centroids is shown by the black curve.

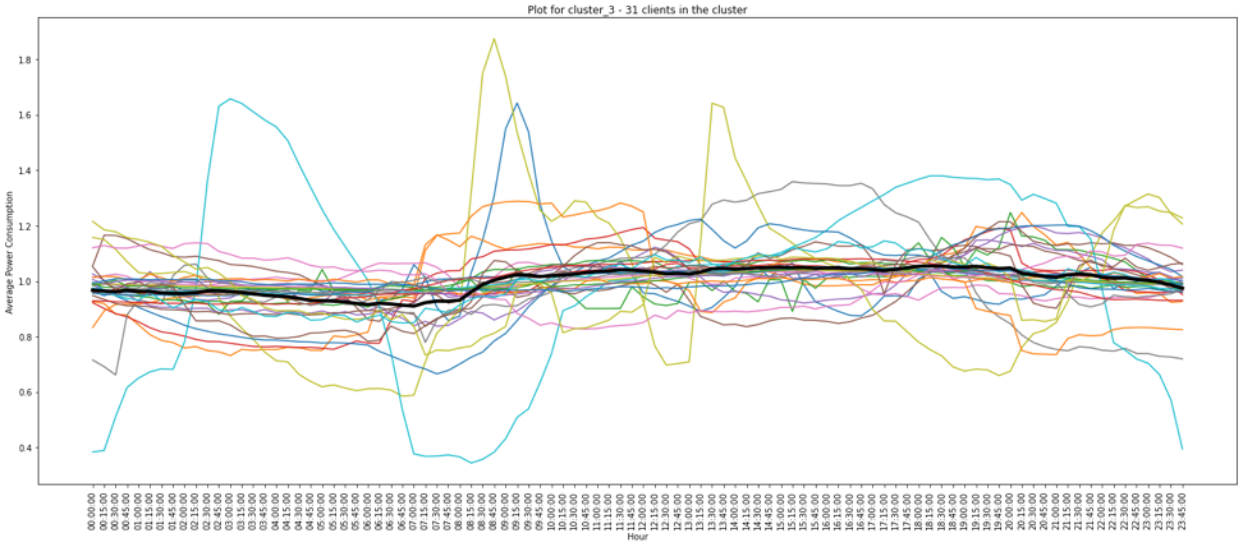
In [30]:
```
for i in range(optimal_clusters):
```

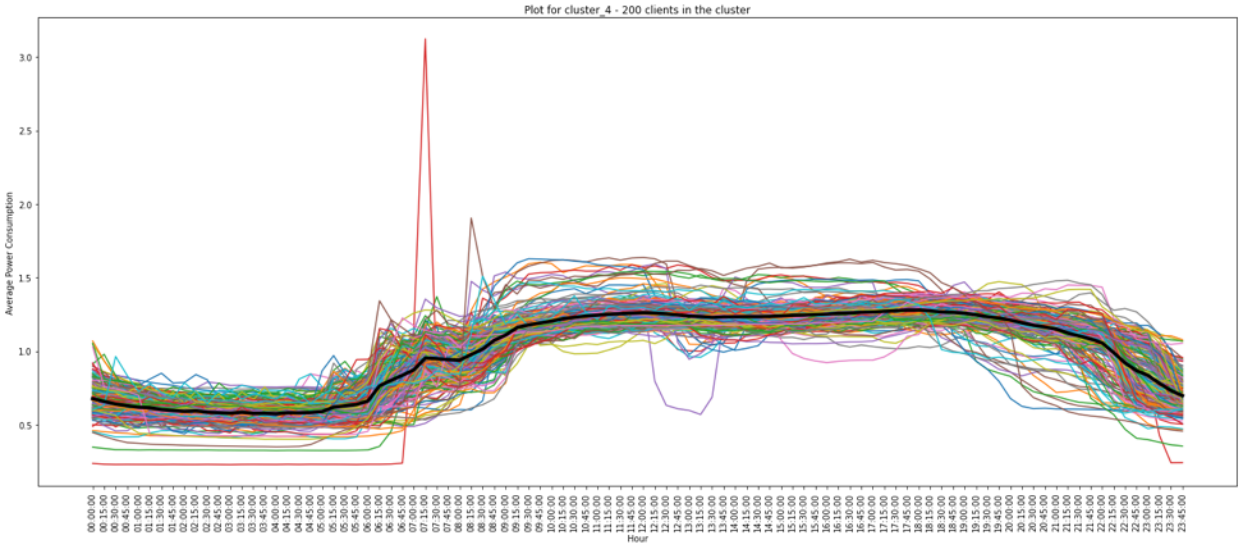
In [30]:
```
for i in range(optimal_clusters):
    plt.figure(figsize=(25,10))
    plt.xticks(rotation=90)
    plt.plot(average_curves_norm.index, clusters[i].T)
    plt.plot(average_curves_norm.index, optimal_model.cluster_cent
    plt.title("Plot for cluster_" + str(i) + " - " + str(len(clust
    plt.xlabel("Hour")
    plt.ylabel("Average Power Consumption")
```






1
Hour


Plot for cluster_3 - 31 clients in the cluster
Average Power Consumption
Hour


Plot for cluster_4 - 200 clients in the cluster
Average Power Consumption
Hour

Q2: (8 marks)

In this exercise you work with the daily curves of 1 single client. First, create a list of arrays, each array containing a curve for a day. You may use X from the cells above. X = average_curves_norm.copy() The list contains 730 arrays, one for each of the days of 2013 and 2014.

a. Determine the optimal value of k (number of clusters). This time you may also perform silhoutte analysis as stated in the module. Carrying out silhoutte analysis is left as an exercise. What do you understand about the clusters?

b. Based on your results from your analyses of both methods, what do understand? Interpret it perhaps with different perspectives of timelines like weeks or months.

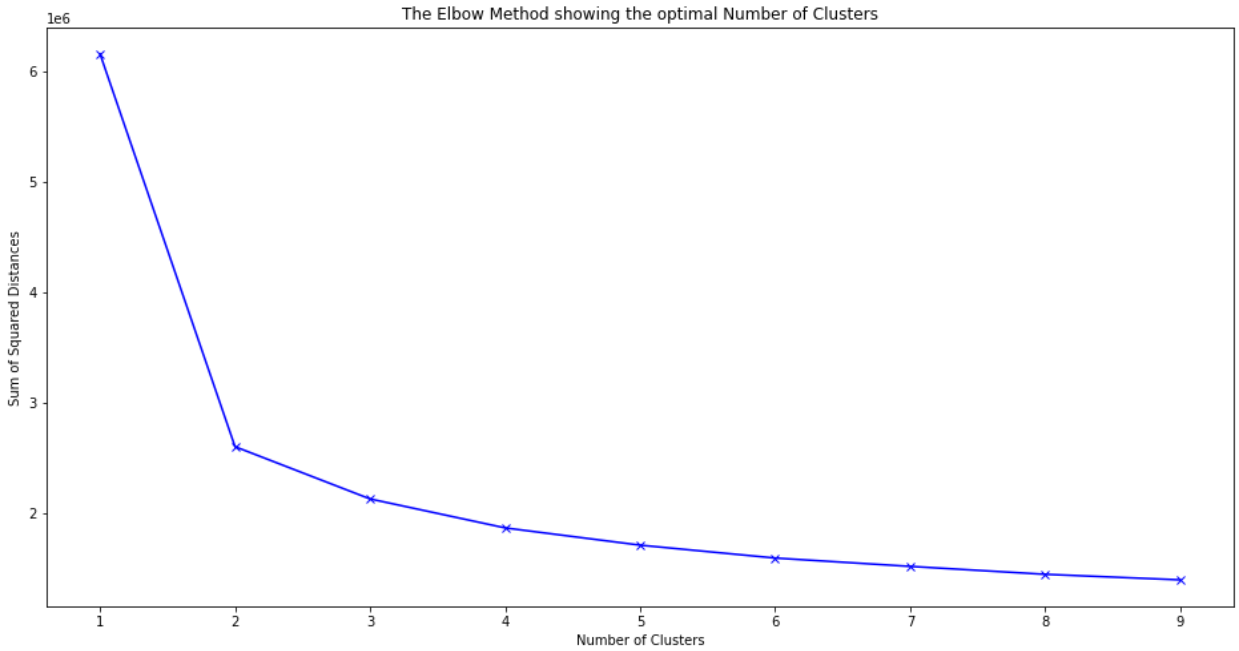
Q2 - Part a

In [31]:
```
client = 'MT_022'
oneClient = data_13_14[client]
X_one_client = [] # a list of arrays, each array being a normalize
for J in range(2*365):
    X_one_client.extend([np.array(oneClient[J*96:(J+1)*96])])#/np.
```

First I'm going to use Elbow Graph.

In [32]:
```
sum_squared_dists = []
number_of_clusters = range(1,10)
for k in number_of_clusters:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(X_one_client)
    sum_squared_dists.append(kmeanModel.inertia_)
```

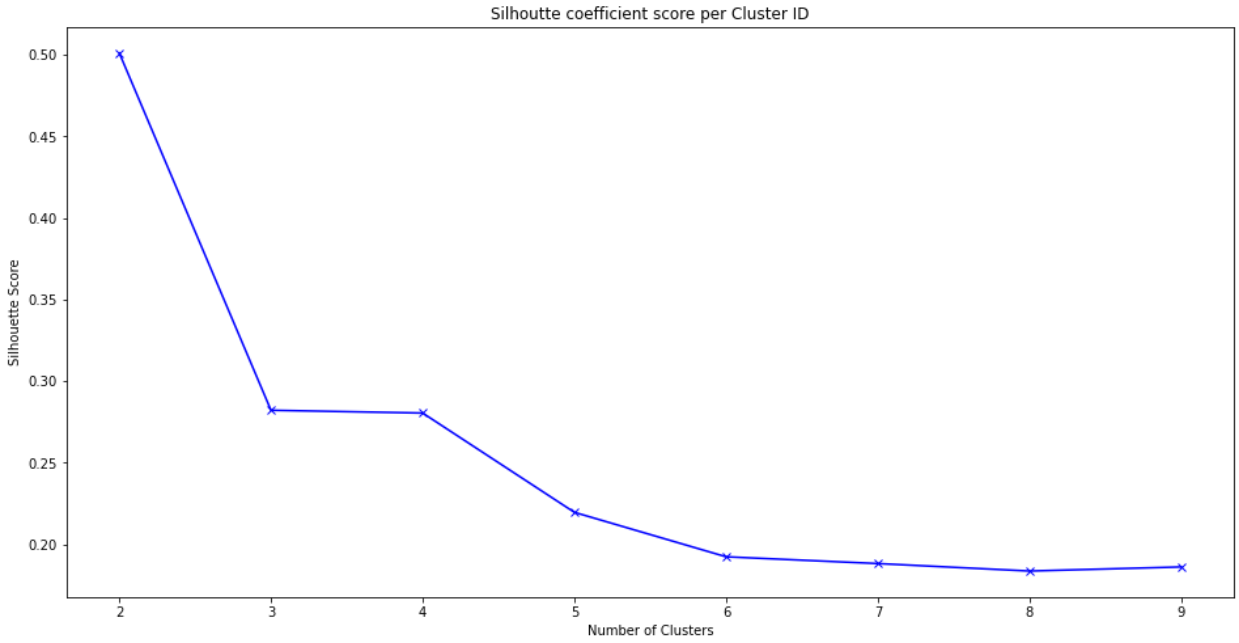
In [33]:
```
plt.figure(figsize=(16,8))
plt.plot(number_of_clusters, sum_squared_dists, 'bx-')
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared Distances')
plt.title('The Elbow Method showing the optimal Number of Clusters
plt.show()
```



Based on the graph, I think k=3 is a good number for the clusters. (but k=2 also seems like good number of clusters)

In [34]:
```
silhouette_score = []
number_of_clusters = range(2,10)
for k in number_of_clusters:
    kmeans_model_one_client = KMeans(n_clusters=k).fit(X_one_clier
    labels = kmeans_model_one_client.labels_
    silhouette_score.append(metrics.silhouette_score(X_one_client,
```

In [35]:
```
plt.figure(figsize=(16,8))
plt.plot(number_of_clusters, silhouette_score, 'bx-')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhoutte coefficient score per Cluster ID')
plt.show()
```



$s(x)$ is a quantity in $[-1, 1]$. A larger Silhoutte coefficient score relates to a model with better defined clusters. Based on the graph, the best number of clusters is 2 clusters.

In the Elbow graph, I had doubt between 2 and 3(Although the change from 1 to 2 was way more than the change between 2 and 3), but based on the Silhoutte coefficient score, k=2,3,4 are better choices than the rest but k=2 is way better than k=3 and k=4. So we Can say that these two methodds confirm each other.

Q2 - Part b

The following cell generates a list of all the days in the two years 2013-2014, which is helpful to answer part (b). The cells below are only to give you a headstart. You may or may not use these and come up with your own interpretation.

In [36]:
```
from datetime import date, timedelta

d1 = date(2013, 1, 1)  # start date
d2 = date(2014, 12, 31)  # end date
delta = d2 - d1         # timedelta
daysyear = []
D = {0:'mon', 1:'tue', 2:'wed', 3:'thu', 4:'fri', 5:'sat', 6:'sun'
for i in range(delta.days + 1):
    daysyear.extend([D[(d1 + timedelta(days=i)).weekday()]+"-"+str
```

Since we have 2 clusters, one easy guess is that the clussters represent weekadys and weekends. So we're going to plot days of the year vs cluster ID to see if our alternitive hypothesis is right or not.

In [37]:
```
X_one_client_df = pd.DataFrame(X_one_client)
X_one_client_df['daysyear'] = daysyear #[i[0:3] for i in daysyear]
optimal_k=2
optimal_model_one_client = KMeans(n_clusters=optimal_k).fit(X_one_
X_one_client_df['cluster'] = optimal_model_one_client.labels_
```

In [38]:
```
plt.figure(figsize=(500,10))
plt.xticks(rotation=90)
plt.plot(X_one_client_df['daysyear'], X_one_client_df['cluster']==
plt.title("Days of Year VS. Cluster ID")
plt.xlabel("Days in Year")
plt.ylabel("Cluster ID")
```

Out[38]:
```
Text(0, 0.5, 'Cluster ID')
```


If you look at the above graph(double click on it) you can see that most of the times, on weekends, the cluster ID is 0 and on weekdays it is 1. So we can say our alternitive hypothesis is right and the clusters represent week days and weekends.