Executive Summary:

This project involved clustering the electricity demand patterns of clients based on 15-minute interval data from 2011 to 2014, sourced from the UCI Machine Learning Repository. Using KMeans clustering and silhouette analysis, I identified optimal cluster counts to categorize clients' consumption behaviors. The goal was to uncover distinct usage patterns for energy management, predictive modeling, and personalized recommendations. The findings showed two primary clusters: one representing regular weekday consumption and the other indicating lower or irregular patterns, likely weekends or holidays. This analysis provided actionable insights for tailored energy-saving strategies.

1. Data:

https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014#

2. Electric Demand:

http://www.think-energy.net/KWvsKWH.htm

Data Preparation

Start by downloading the data to a local directory and modify the "pathToFile" and "fileName" variables, if needed

In [1]:
```
pathToFile = r"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 2,3445 instead of 2
import fileinput

with fileinput.FileInput(pathToFile+fileName, inplace=True, backup='.bak') as file:
    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 [26]:
```
data.head(2)
```

Out[26]:

	MT_001	MT_002	MT_003	MT_004	MT_005	MT_006	MT_007	MT_008	MT_00
2011-01-01 00:15:00	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.

2 rows × 370 columns

In [27]:
```
data.tail(2)
```

Out[27]:

	MT_001	MT_002	MT_003	MT_004	MT_005	MT_006	MT_007	M
2014-12-31 23:45:00	1.269036	21.337127	1.737619	166.666667	85.365854	285.714286	10.17524	225.5
2015-01-01 00:00:00	2.538071	19.914651	1.737619	178.861789	84.146341	279.761905	10.17524	249.1

2 rows × 370 columns

In [28]:
```
data.shape
```

Out[28]:
```
(140256, 370)
```

As 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 [29]:
```
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: 401.0+ MB
```

In [30]:
```
data.describe()
```

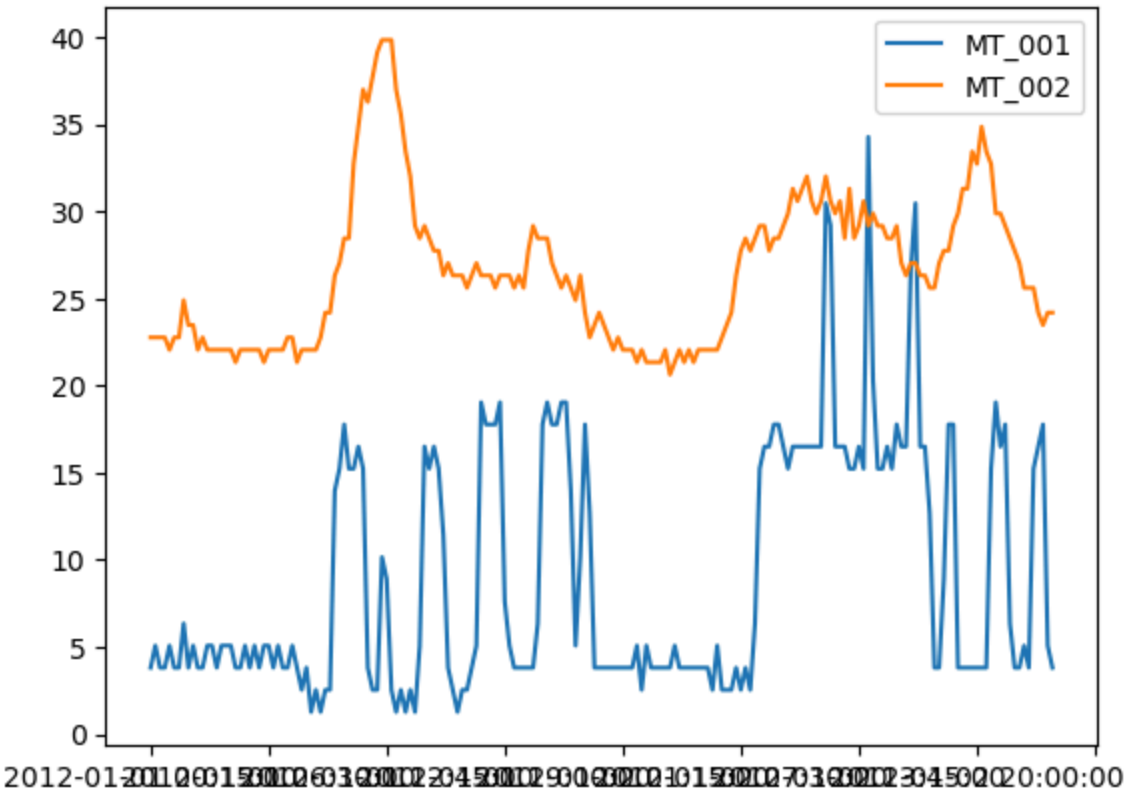
Out[30]:

	MT_001	MT_002	MT_003	MT_004	MT_005	M
count	140256.000000	140256.000000	140256.000000	140256.000000	140256.000000	140256.
mean	3.970785	20.768480	2.918308	82.184490	37.240309	141.
std	5.983965	13.272415	11.014456	58.248392	26.461327	98.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.000000	2.844950	0.000000	36.585366	15.853659	71.
50%	1.269036	24.893314	1.737619	87.398374	39.024390	157.
75%	2.538071	29.871977	1.737619	115.853659	54.878049	205.
max	48.223350	115.220484	151.172893	321.138211	150.000000	535.

8 rows × 370 columns

Plot the 2 days of 2012 for the first 2 clients

In [31]:
```
data_example = data.loc['2012-01-01 00:15:00':'2012-01-03 00:00:00'][['MT_001','MT_
data_example.plot()
plt.show()
```



We quickly note that the main difference between the curves is the level (also seen on the means in df.describe()), so 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, rather than the overall level.

Data for the analysis

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

In [32]:
```
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 [33]:
```
# 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 [34]:
```
# 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 [36]:
```
clients = data2011.columns
clients_no_demand = clients[data2013.mean()==0] # clients with 0 demand
data_13_14 = pd.concat([data2013, data2014], axis=0)  # concatenating 2013 and 2014
data_13_14 = data_13_14.drop(clients_no_demand, axis=1)  # drop clients with 0 dema
print(data_13_14.shape)
print(sum(data_13_14.mean()==0)) # check that there are no clients with 0 demand
```

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

Getting average curves per client

In [37]:
```
data = data_13_14.copy() # weekdays weekends, data2011, data2012, data2013, data201
```

In [38]:
```
data['hour'] = data.index.map(lambda x: x[11:])
```

In [48]:
```
data.head(5)
```

Out[48]:

	MT_001	MT_002	MT_003	MT_004	MT_005	MT_006	MT_007	M
2013-01-01 00:15:00	2.538071	22.759602	2.606429	138.211382	63.414634	255.952381	4.522329	239.0
2013-01-01 00:30:00	1.269036	22.759602	2.606429	138.211382	63.414634	264.880952	5.652911	228.9
2013-01-01 00:45:00	2.538071	22.759602	2.606429	134.146341	60.975610	250.000000	5.652911	239.0
2013-01-01 01:00:00	1.269036	23.470839	2.606429	130.081301	56.097561	226.190476	6.218202	249.1
2013-01-01 01:15:00	3.807107	23.470839	2.606429	130.081301	58.536585	229.166667	6.783493	239.0

5 rows × 350 columns

Getting average curves per client

In [40]:
```
datagrouped = data.groupby("hour")
average_curves = datagrouped.agg("mean")
average_curves.shape
```

Out[40]: (96, 349)

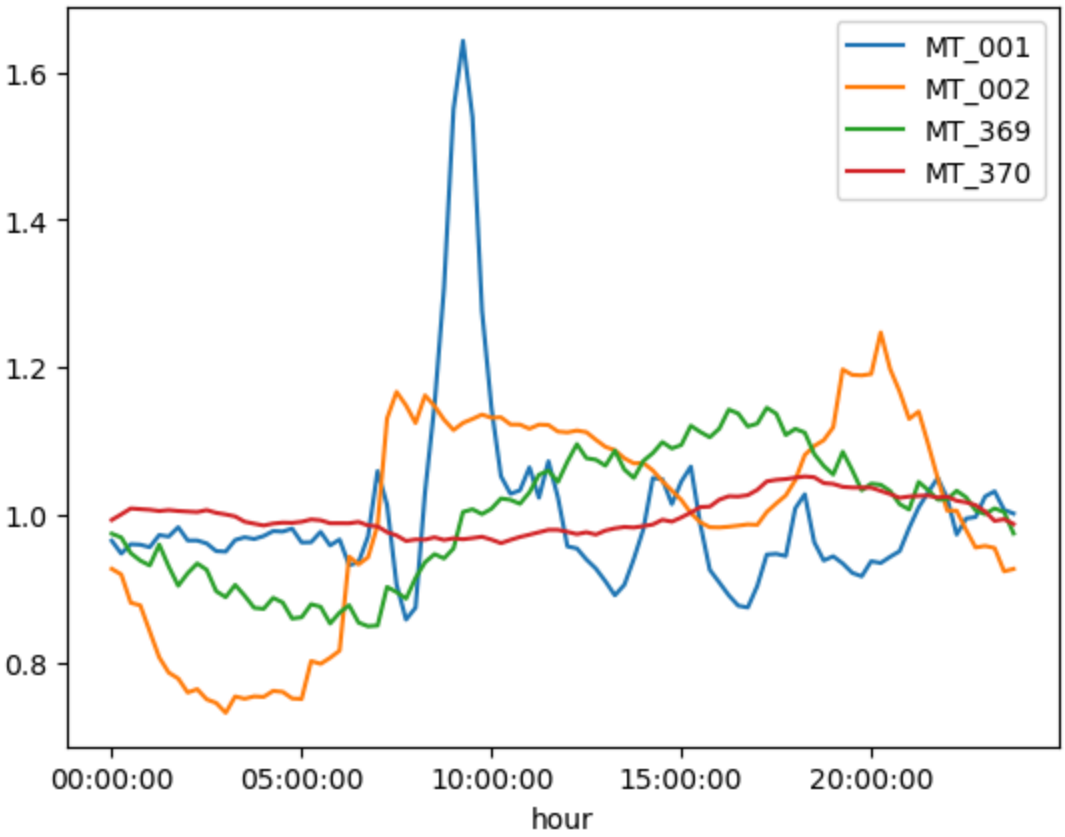
The dataframe average_curves contains the 349 typical weekday electric demands

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

In [41]:
```
average_curves_norm = average_curves/(average_curves.mean())
```

Plot the first 2 and last 2 clients

In [42]:
```
average_curves_norm[['MT_001','MT_002','MT_369','MT_370']].plot()
plt.show()
```



Clustering Analysis on the average normalized curves

In [43]:
```
X = average_curves_norm.copy()
X = np.array(X.T) # put it in the right format
```

Exercise 1

a. Using the functions described in Module 8, determine a convenient number of clusters based on *inertia* and based on *silhouette*. Justify your choice.

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 centroid of the cluster (make this curve thicker to distinguish it from the individual curves)

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

In [ ]:
```

```

In [51]:
```
# X is a NumPy array where each row corresponds to a client and columns correspond
X = average_curves_norm.copy()
```

```
X = np.array(X.T)
print("Shape of X:", X.shape)
```

```
Shape of X: (349, 96)
```

Computing Inertia and Silhouette Scores

We will compute inertia and silhouette scores for a range of cluster numbers (k) to identify the optimal number of clusters.

In [52]:
```
max_k = 15  # Maximum number of clusters to test
inertias = []
silhouette_avgs = []

# Since silhouette_score requires at least 2 clusters, we start from k=2
for k in range(2, max_k+1):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)
    silhouette_avg = silhouette_score(X, kmeans.labels_)
    silhouette_avgs.append(silhouette_avg)
```

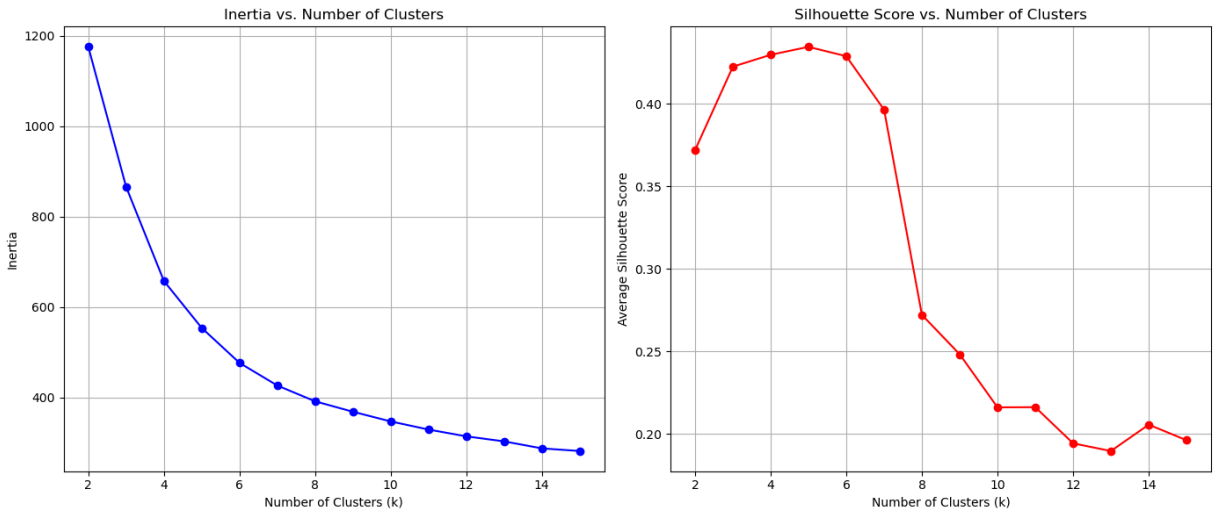
```
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
```

```
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
```

Plotting Inertia and Silhouette Scores

In [53]:
```
plt.figure(figsize=(14, 6))

# Plot Inertia
plt.subplot(1, 2, 1)
plt.plot(range(2, max_k+1), inertias, 'bo-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Inertia vs. Number of Clusters')
plt.grid(True)

# Plot Silhouette Score
plt.subplot(1, 2, 2)
plt.plot(range(2, max_k+1), silhouette_avgs, 'ro-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Average Silhouette Score')
plt.title('Silhouette Score vs. Number of Clusters')
plt.grid(True)

plt.tight_layout()
plt.show()
```



Choice of Number of Clusters

Inertia Plot Analysis:

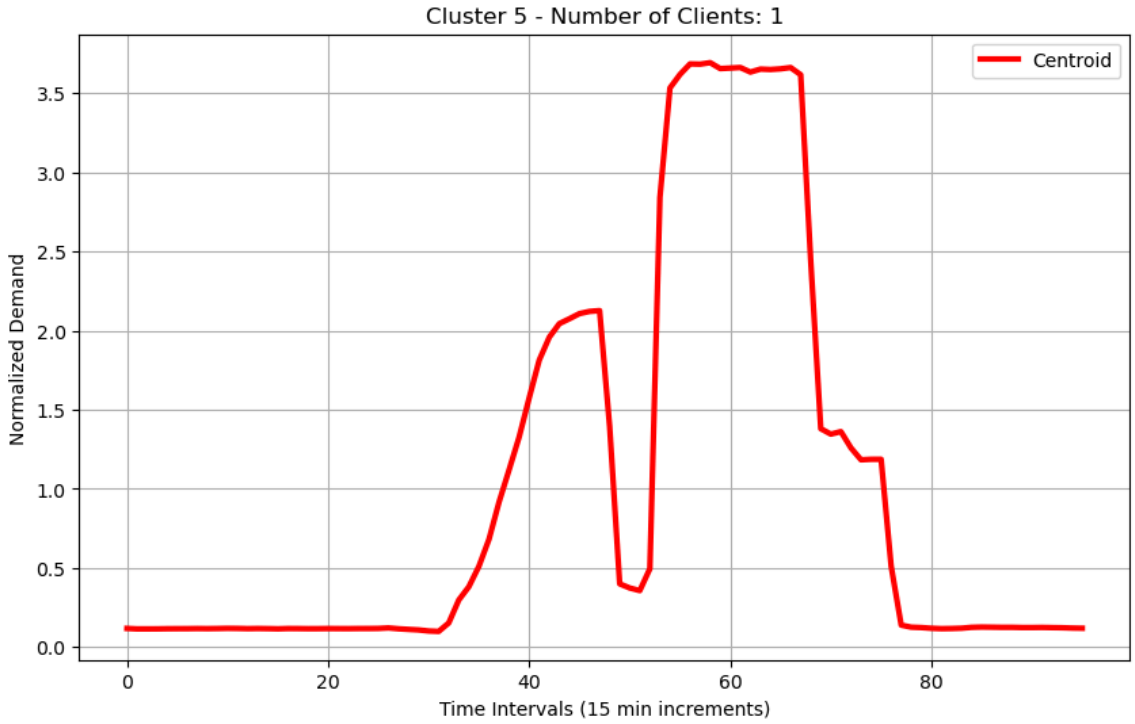
The inertia decreases as the number of clusters increases, which is expected. The "elbow" point, where the rate of decrease sharply changes, is around k = 5. After k = 5, the decrease in inertia becomes more gradual, indicating that adding more clusters doesn't significantly improve the model.

Silhouette Score Analysis:The silhouette score peaks at k = 5, suggesting that the clusters are well-defined at this point.

The score starts to decrease after k = 5, indicating that the clusters become less distinct.

Conclusion:

Based on both the inertia plot and the silhouette score plot, the optimal number of clusters appears to be k = 5. *However, after plotting all the clusters, I found that the last cluster has only 1 data point in it. Which is why I reduced the optimal clusters k = 4*. Plot shown below for the 5th cluster.



Part (b): Plotting Clusters and Centroids

Now, we will perform KMeans clustering with k = 6 and plot the results.

Performing KMeans Clustering

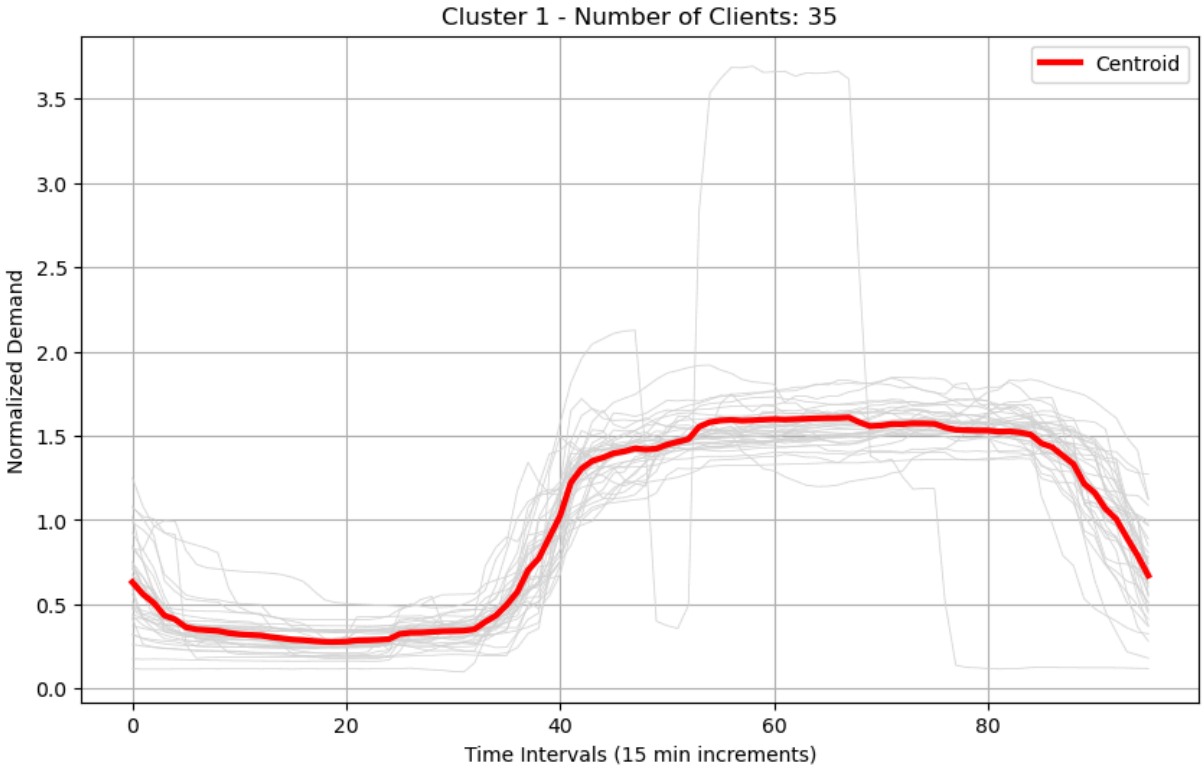
In [56]:
```
k_optimal = 4 # optimal clusters
kmeans_optimal = KMeans(n_clusters=k_optimal, random_state=42)
kmeans_optimal.fit(X)
labels = kmeans_optimal.labels_
centroids = kmeans_optimal.cluster_centers_
```

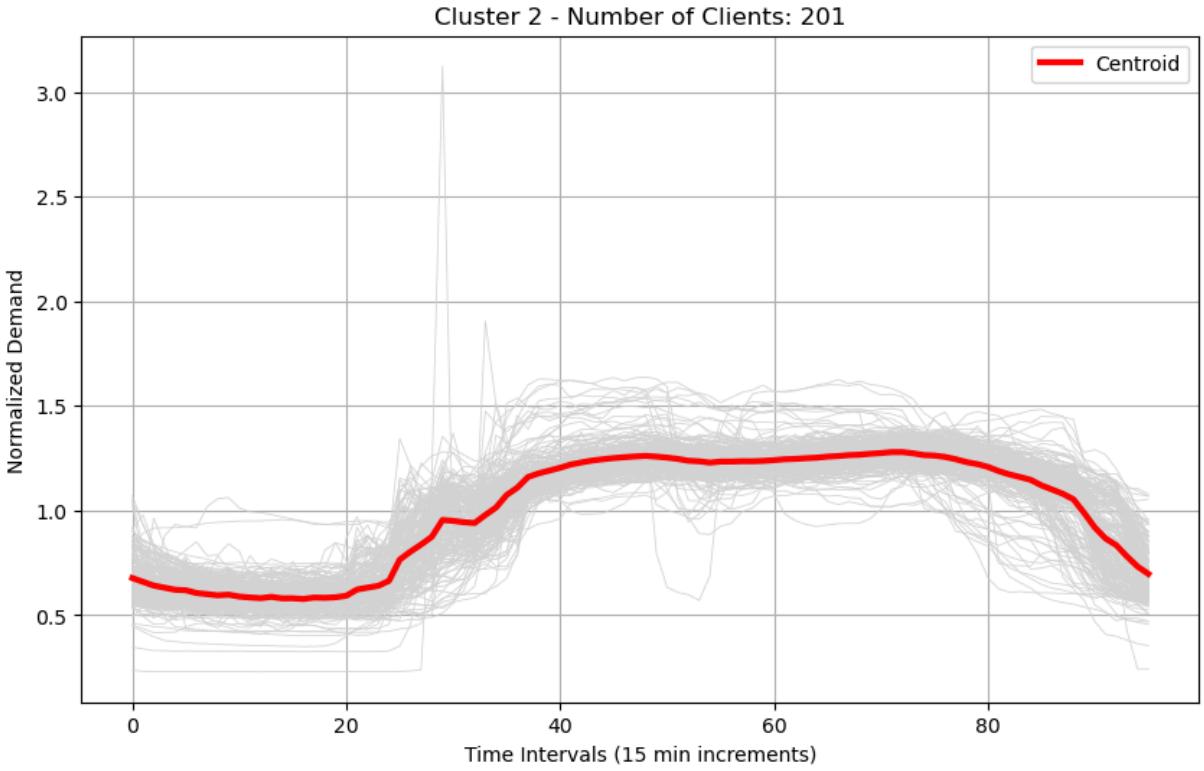
```
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=2.
  warnings.warn(
```

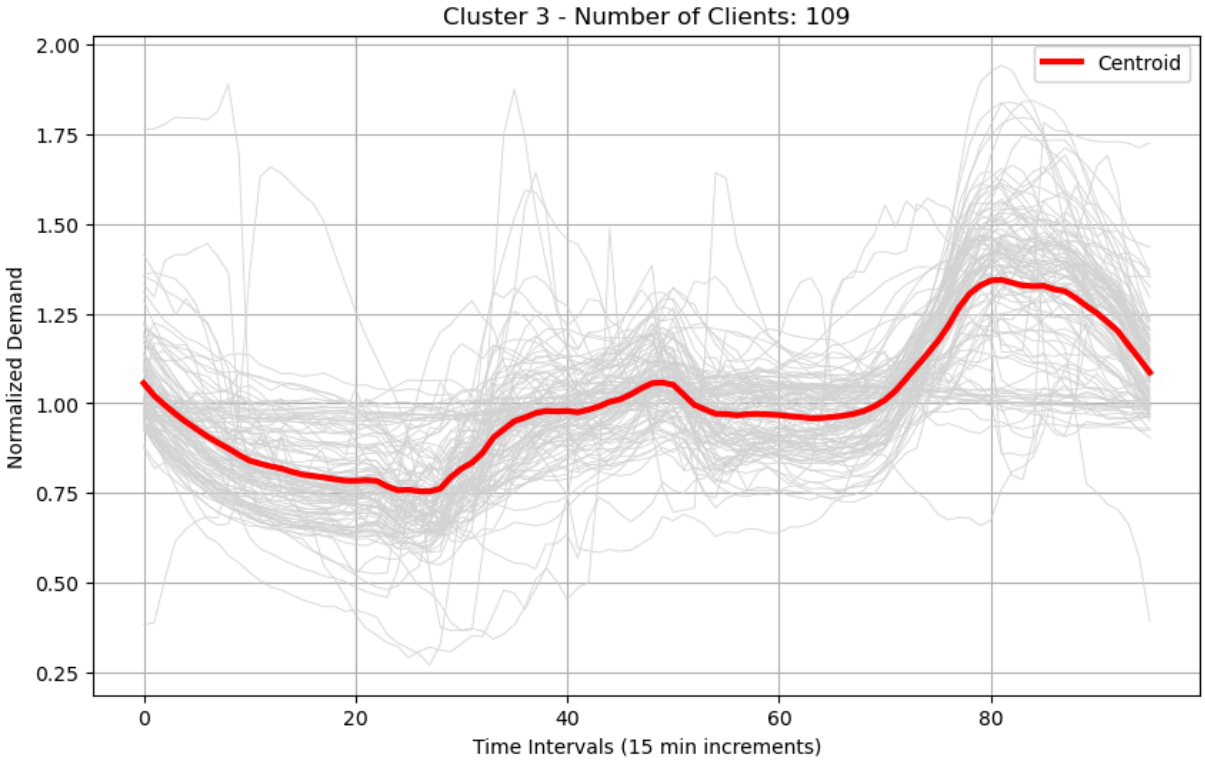
Plotting the Clusters

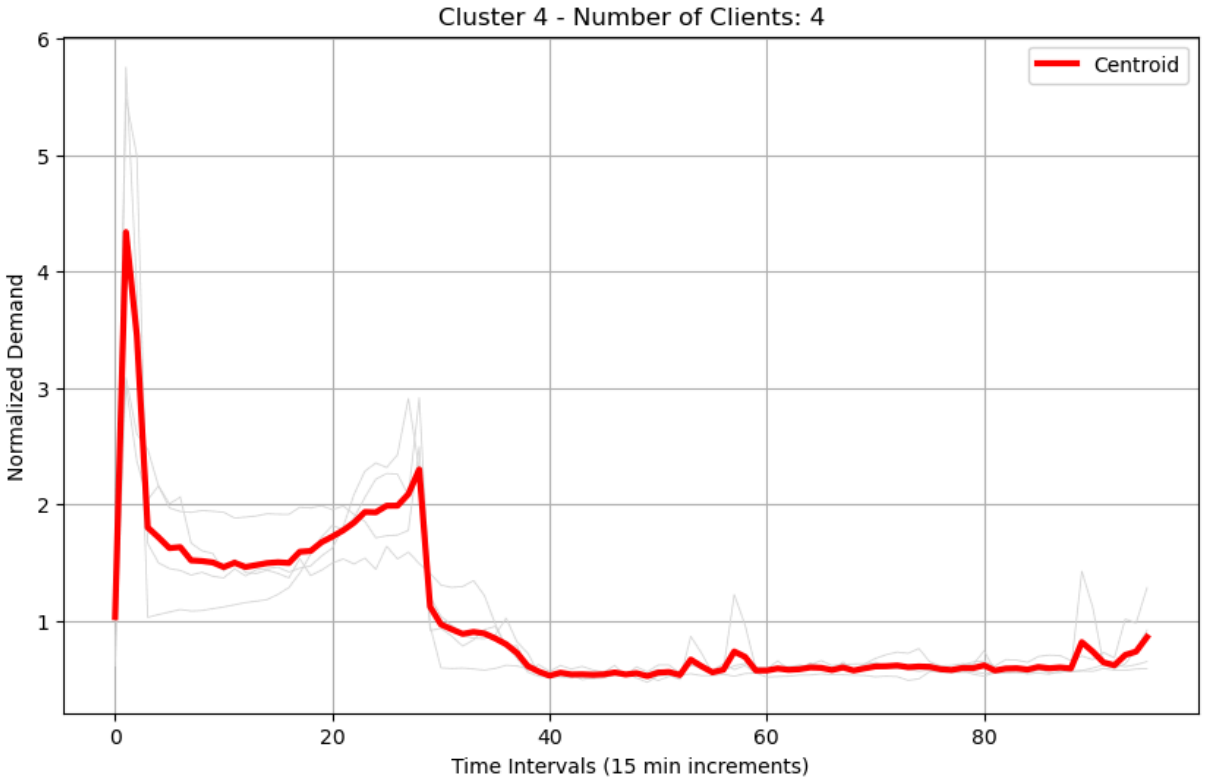
We will plot the normalized average demand curves for all clients in each cluster and overlay the centroid curve.

In [57]:
```
for cluster_num in range(k_optimal):
    plt.figure(figsize=(10, 6))

    # Get indices of clients in the cluster
    cluster_indices = np.where(labels == cluster_num)[0]
    cluster_curves = X[cluster_indices]
    num_clients = len(cluster_indices)

    # Plot individual client curves
    for curve in cluster_curves:
        plt.plot(curve, color='lightgray', linewidth=0.5)

    # Plot the centroid curve
    plt.plot(centroids[cluster_num], color='red', linewidth=3, label='Centroid')

    plt.title(f'Cluster {cluster_num+1} - Number of Clients: {num_clients}')
    plt.xlabel('Time Intervals (15 min increments)')
    plt.ylabel('Normalized Demand')
    plt.legend()
    plt.grid(True)
    plt.show()
```


Cluster 1 - Number of Clients: 35
Centroid
Normalized Demand
Time Intervals (15 min increments)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
20
40
60
80


Cluster 2 - Number of Clients: 201
Centroid
Normalized Demand
Time Intervals (15 min increments)
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80





Interpretation

Each plot represents one of the six clusters. The light gray lines are the normalized demand curves of individual clients in that cluster. The red line is the centroid of the cluster, representing the typical behavior. The number of clients in each cluster is displayed in the title.

Observations:

Clusters differ in the shape of their demand curves. Some clusters may represent clients with peak demand in the morning, while others in the evening.

Exercise 2

In this exercise you work with the daily curves of 1 single client.

First you create a list of arrays, each array containing a normalized curve for a day. The list contains 730 arrays, one for each of the days of 2013 and 2014.

a. Determine the optimal value for k-means.

b. What do the clusters that you obtained represent?

Part (a): Determine the Optimal Value for k-means

Preparing the Data

In [59]:
```
client = 'MT_022'
oneClient = data_13_14[client]
X_daily = []  # List of daily demand curves

num_days = int(len(oneClient) / 96)  # Each day has 96 data points (15-minute inter

for day in range(num_days):
    daily_data = np.array(oneClient[day*96:(day+1)*96])
    daily_data_norm = daily_data / np.mean(daily_data)  # Normalize the curve
    X_daily.append(daily_data_norm)

X_daily = np.array(X_daily)
print("Shape of X_daily:", X_daily.shape)
```

```
Shape of X_daily: (730, 96)
```

Computing Inertia and Silhouette Scores

We will compute inertia and silhouette scores for different values of k to find the optimal number of clusters.

In [60]:
```
max_k = 15
inertias_daily = []
silhouette_avgs_daily = []

for k in range(2, max_k+1):
    kmeans_daily = KMeans(n_clusters=k, random_state=42)
    kmeans_daily.fit(X_daily)
    inertias_daily.append(kmeans_daily.inertia_)
```

```
    silhouette_avg_daily = silhouette_score(X_daily, kmeans_daily.labels_)
    silhouette_avgs_daily.append(silhouette_avg_daily)
```

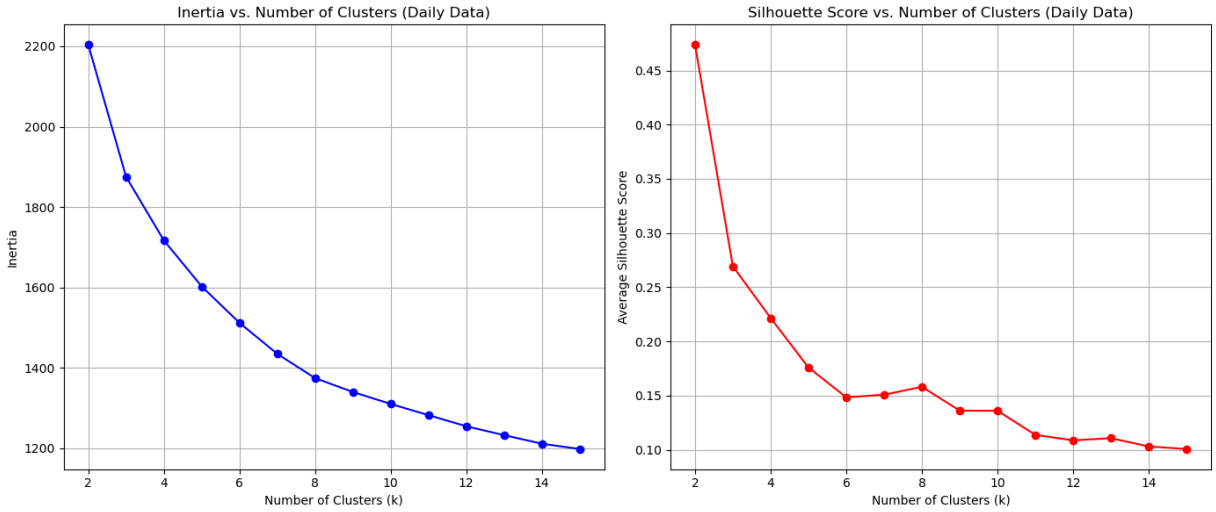
```
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
```

```
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
```

Plotting Inertia and Silhouette Scores

In [61]:
```
plt.figure(figsize=(14, 6))

# Plot Inertia
plt.subplot(1, 2, 1)
plt.plot(range(2, max_k+1), inertias_daily, 'bo-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Inertia vs. Number of Clusters (Daily Data)')
plt.grid(True)

# Plot Silhouette Score
plt.subplot(1, 2, 2)
plt.plot(range(2, max_k+1), silhouette_avgs_daily, 'ro-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Average Silhouette Score')
plt.title('Silhouette Score vs. Number of Clusters (Daily Data)')
plt.grid(True)

plt.tight_layout()
plt.show()
```



Commentary on the Choice of Number of Clusters

Inertia Plot Analysis:

The inertia decreases with increasing k, as expected. There is a noticeable "elbow" at k = 2, where the rate of decrease in inertia changes significantly.

Silhouette Score Analysis:

The silhouette score peaks at k = 2, indicating the clusters are well-defined. After k = 2, the silhouette score decreases, suggesting that adding more clusters doesn't improve the clustering structure.

Conclusion:

Based on both plots, the optimal number of clusters for the daily curves of client 'MT_022' is k = 2.

Part (b): Interpretation of the Clusters

Performing KMeans Clustering with k = 2

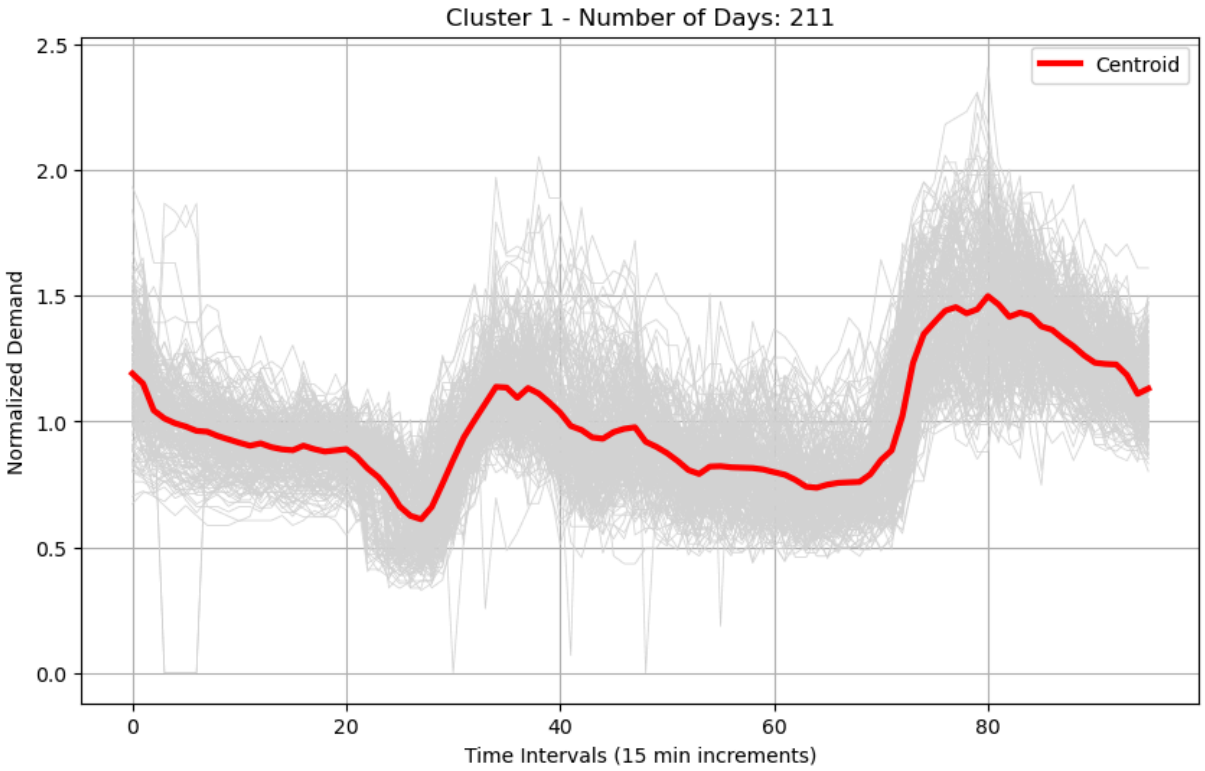
In [62]:
```
k_daily_optimal = 2
kmeans_daily_optimal = KMeans(n_clusters=k_daily_optimal, random_state=42)
kmeans_daily_optimal.fit(X_daily)
daily_labels = kmeans_daily_optimal.labels_
daily_centroids = kmeans_daily_optimal.cluster_centers_
```

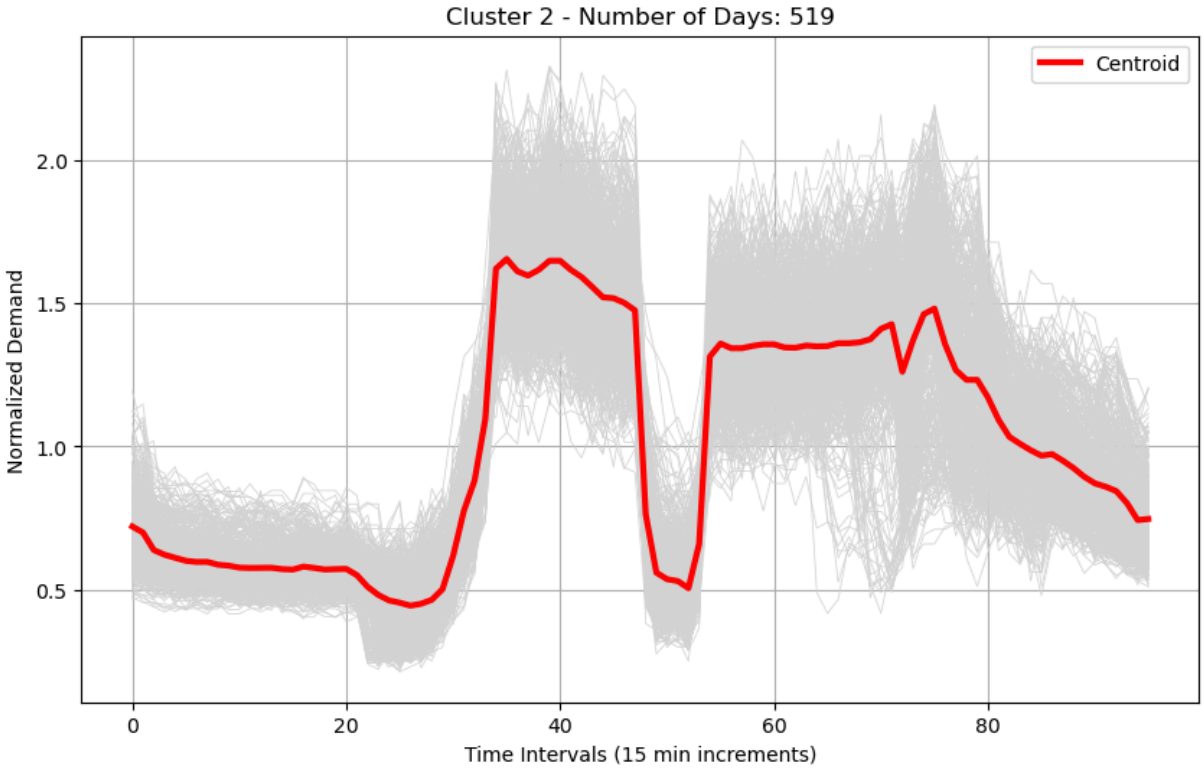
```
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWar
ning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the va
lue of `n_init` explicitly to suppress the warning
  warnings.warn(
c:\Users\dawoo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarn
ing: KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=3.
  warnings.warn(
```

Plotting the Clusters

In [63]:
```
for cluster_num in range(k_daily_optimal):
    plt.figure(figsize=(10, 6))

    # Get indices of days in the cluster
    cluster_indices = np.where(daily_labels == cluster_num)[0]
    cluster_curves = X_daily[cluster_indices]
    num_days_in_cluster = len(cluster_indices)
```

```
    # Plot individual daily curves
    for curve in cluster_curves:
        plt.plot(curve, color='lightgray', linewidth=0.5)

    # Plot the centroid curve
    plt.plot(daily_centroids[cluster_num], color='red', linewidth=3, label='Centroi

    plt.title(f'Cluster {cluster_num+1} - Number of Days: {num_days_in_cluster}')
    plt.xlabel('Time Intervals (15 min increments)')
    plt.ylabel('Normalized Demand')
    plt.legend()
    plt.grid(True)
    plt.show()
```





The following cell generates a list of all the days in the two years 2013-2014, which is helpful to answer part (b)

In [64]:
```
from datetime import date, timedelta

d1 = date(2013, 1, 1)  # start date
d2 = date(2014, 12, 31)  # end date
delta = d2 - d1         # timedelta
daysyear = []
weekdays = []
weekday_names = {0:'mon', 1:'tue', 2:'wed', 3:'thu', 4:'fri', 5:'sat', 6:'sun'}

for i in range(delta.days + 1):
    current_date = d1 + timedelta(days=i)
    daysyear.append(str(current_date))
    weekdays.append(weekday_names[current_date.weekday()])
```

Assigning Days to Clusters

In [65]:
```
from itertools import compress

# Get the list of days for each cluster
cluster_1_days = list(compress(daysyear, daily_labels == 0))
cluster_1_weekdays = list(compress(weekdays, daily_labels == 0))

cluster_2_days = list(compress(daysyear, daily_labels == 1))
cluster_2_weekdays = list(compress(weekdays, daily_labels == 1))
```

Counting the Days of the Week in Each Cluster

In [66]:
```
from collections import Counter

cluster_1_weekday_counts = Counter(cluster_1_weekdays)
cluster_2_weekday_counts = Counter(cluster_2_weekdays)

# Display the counts
print("Cluster 1 Weekday Counts:")
print(cluster_1_weekday_counts)
print("\nCluster 2 Weekday Counts:")
print(cluster_2_weekday_counts)
```

```
Cluster 1 Weekday Counts:
Counter({'sun': 104, 'sat': 52, 'mon': 12, 'tue': 11, 'fri': 11, 'thu': 11, 'wed': 1
0})

Cluster 2 Weekday Counts:
Counter({'wed': 95, 'tue': 94, 'thu': 93, 'fri': 93, 'mon': 92, 'sat': 52})
```

The following cell shows you how to do the following: given a list L, and a list of booleans B (True, False), get the sublist of L that corresponds to the True values in B. For example if L=[1,2,3,4] and B=[False,True,True,False], then the results is [2,3]. This is also helpful for pard (b)

In [67]:
```
from itertools import compress
L = [1,2,3,4]
B = [False, True, True, False]
list(compress(L, B))
```

Out[67]:
```
[2, 3]
```

Conclusion for Exercise 2

For this exercise, we analyzed the daily electricity demand of a single client ('MT_022') over 2013-2014, clustering the curves to identify consumption patterns.

Optimal Clusters

Number of Clusters: Based on inertia and silhouette scores, the optimal number of clusters is 2, where we observed an "elbow" in inertia and the highest silhouette score.

Cluster Interpretation

Cluster 1: Primarily Sundays and half the Saturdays, with some weekdays, indicating lower or irregular consumption patterns, likely due to weekends, holidays, or special events. Cluster 2: Mostly regular weekdays and remaining Saturdays, showing consistent, higher consumption linked to typical routines.

Key Findings

Cluster 1: Reflects days with atypical patterns, possibly weekends or holidays with lower demand and delayed peaks. Cluster 2: Displays steady weekday patterns, with regular morning and evening peaks.

Implications

Energy Management: Insights can support tailored energy-saving strategies. Predictive Modeling: Improved demand forecasting based on the day. Customized Services: Potential for personalized recommendations matching usage habits. In summary, these clusters reveal two main consumption patterns, offering a better understanding of the client's energy usage, which could lead to more targeted energy solutions.