Exercise 10 : Clustering using kMeans

Step 1 : Read sampleDataset.csv provided

In [95]:
```
import pandas as pd

df = pd.read_csv('sampleDataSet.csv')
```

Step 2 : Check percentage of missing values in each column and drop the columns with missing pixels percentage greater than 95%

In [96]:
```
nan_percentage = (df.isna().sum() / len(df)) * 100
threshold = 95

columns_to_drop = nan_percentage[nan_percentage > threshold].index
df = df.drop(columns=columns_to_drop, axis=1)

df
```

Out[96]:

	Unnamed: 0	TimeStamp	(140.7468006 41.8188869)	(140.957261 37.6422006)	(140.4021967 36.555081)	(139.1103334 36.2974922)	(140.13 35.61
0	0	2022-04-01 01:00:00	9.0	3.0	6.0	5.0	
1	1	2022-04-01 02:00:00	0.0	0.0	8.0	10.0	
2	2	2022-04-01 03:00:00	2.0	NaN	7.0	-2.0	
3	3	2022-04-01 04:00:00	4.0	NaN	9.0	1.0	
4	4	2022-04-01 05:00:00	4.0	NaN	3.0	6.0	
5	5	2022-04-01 06:00:00	-1.0	4.0	4.0	-8.0	
6	6	2022-04-01 07:00:00	6.0	4.0	4.0	14.0	
7	7	2022-04-01 08:00:00	10.0	4.0	3.0	-1.0	
8	8	2022-04-01 09:00:00	3.0	0.0	4.0	-5.0	
9	9	2022-04-01 10:00:00	6.0	9.0	4.0	-3.0	
10	10	2022-04-01 11:00:00	6.0	9.0	3.0	0.0	
11	11	2022-04-01 12:00:00	1.0	7.0	NaN	-3.0	
12	12	2022-04-01 13:00:00	-1.0	11.0	6.0	5.0	
13	13	2022-04-01 14:00:00	2.0	9.0	8.0	7.0	
14	14	2022-04-01 15:00:00	4.0	10.0	13.0	3.0	
15	15	2022-04-01 16:00:00	5.0	11.0	7.0	17.0	
16	16	2022-04-01 17:00:00	10.0	5.0	8.0	-2.0	
17	17	2022-04-01 18:00:00	9.0	2.0	7.0	7.0	
18	18	2022-04-01 19:00:00	1.0	NaN	7.0	10.0	
19	19	2022-04-01 20:00:00	8.0	NaN	7.0	-3.0	
20	20	2022-04-01 21:00:00	2.0	5.0	8.0	1.0	
21	21	2022-04-01 22:00:00	7.0	4.0	5.0	0.0	

	Unnamed: 0	TimeStamp	(140.7468006 41.8188869)	(140.957261 37.6422006)	(140.4021967 36.555081)	(139.1103334 36.2974922)	(140.13 35.61
22	22	2022-04-01 23:00:00	1.0	3.0	6.0	10.0	
23	23	2022-04-02 00:00:00	7.0	5.0	6.0	-3.0	
24	24	2022-04-02 01:00:00	7.0	5.0	6.0	11.0	
25	25	2022-04-02 02:00:00	14.0	6.0	5.0	-4.0	
26	26	2022-04-02 03:00:00	11.0	6.0	7.0	2.0	
27	27	2022-04-02 04:00:00	14.0	6.0	5.0	6.0	
28	28	2022-04-02 05:00:00	14.0	8.0	9.0	3.0	
29	29	2022-04-02 06:00:00	9.0	4.0	8.0	13.0	
30	30	2022-04-02 07:00:00	8.0	12.0	4.0	4.0	
31	31	2022-04-02 08:00:00	3.0	17.0	3.0	11.0	
32	32	2022-04-02 09:00:00	9.0	20.0	2.0	8.0	
33	33	2022-04-02 10:00:00	12.0	20.0	4.0	-2.0	
34	34	2022-04-02 11:00:00	3.0	15.0	NaN	6.0	
35	35	2022-04-02 12:00:00	4.0	10.0	2.0	10.0	
36	36	2022-04-02 13:00:00	7.0	12.0	3.0	1.0	
37	37	2022-04-02 14:00:00	5.0	10.0	8.0	0.0	
38	38	2022-04-02 15:00:00	0.0	8.0	1.0	7.0	
39	39	2022-04-02 16:00:00	5.0	10.0	9.0	4.0	

40 rows × 24 columns

Step 3 : Perform mean Imputation if missing values are present in the data

In [97]:
```
columns_with_nan = df.columns[df.isna().any()].tolist()


for column in columns_with_nan:
    df[column] = df[column].fillna(df[column].mean())
df
```

Out[97]:

	Unnamed: 0	TimeStamp	(140.7468006 41.8188869)	(140.957261 37.6422006)	(140.4021967 36.555081)	(139.1103334 36.2974922)	(140.13 35.61
0	0	2022-04-01 01:00:00	9.0	3.000000	6.000000	5.0	
1	1	2022-04-01 02:00:00	0.0	0.000000	8.000000	10.0	
2	2	2022-04-01 03:00:00	2.0	7.828571	7.000000	-2.0	
3	3	2022-04-01 04:00:00	4.0	7.828571	9.000000	1.0	
4	4	2022-04-01 05:00:00	4.0	7.828571	3.000000	6.0	
5	5	2022-04-01 06:00:00	-1.0	4.000000	4.000000	-8.0	
6	6	2022-04-01 07:00:00	6.0	4.000000	4.000000	14.0	
7	7	2022-04-01 08:00:00	10.0	4.000000	3.000000	-1.0	
8	8	2022-04-01 09:00:00	3.0	0.000000	4.000000	-5.0	
9	9	2022-04-01 10:00:00	6.0	9.000000	4.000000	-3.0	
10	10	2022-04-01 11:00:00	6.0	9.000000	3.000000	0.0	
11	11	2022-04-01 12:00:00	1.0	7.000000	5.763158	-3.0	
12	12	2022-04-01 13:00:00	-1.0	11.000000	6.000000	5.0	
13	13	2022-04-01 14:00:00	2.0	9.000000	8.000000	7.0	
14	14	2022-04-01 15:00:00	4.0	10.000000	13.000000	3.0	
15	15	2022-04-01 16:00:00	5.0	11.000000	7.000000	17.0	
16	16	2022-04-01 17:00:00	10.0	5.000000	8.000000	-2.0	
17	17	2022-04-01 18:00:00	9.0	2.000000	7.000000	7.0	
18	18	2022-04-01 19:00:00	1.0	7.828571	7.000000	10.0	
19	19	2022-04-01 20:00:00	8.0	7.828571	7.000000	-3.0	
20	20	2022-04-01 21:00:00	2.0	5.000000	8.000000	1.0	
21	21	2022-04-01 22:00:00	7.0	4.000000	5.000000	0.0	

	Unnamed: 0	TimeStamp	(140.746800641.8188869)	(140.95726137.6422006)	(140.402196736.555081)	(139.110333436.2974922)	(140.1335.61
22	22	2022-04-01 23:00:00	1.0	3.000000	6.000000	10.0	
23	23	2022-04-02 00:00:00	7.0	5.000000	6.000000	-3.0	
24	24	2022-04-02 01:00:00	7.0	5.000000	6.000000	11.0	
25	25	2022-04-02 02:00:00	14.0	6.000000	5.000000	-4.0	
26	26	2022-04-02 03:00:00	11.0	6.000000	7.000000	2.0	
27	27	2022-04-02 04:00:00	14.0	6.000000	5.000000	6.0	
28	28	2022-04-02 05:00:00	14.0	8.000000	9.000000	3.0	
29	29	2022-04-02 06:00:00	9.0	4.000000	8.000000	13.0	
30	30	2022-04-02 07:00:00	8.0	12.000000	4.000000	4.0	
31	31	2022-04-02 08:00:00	3.0	17.000000	3.000000	11.0	
32	32	2022-04-02 09:00:00	9.0	20.000000	2.000000	8.0	
33	33	2022-04-02 10:00:00	12.0	20.000000	4.000000	-2.0	
34	34	2022-04-02 11:00:00	3.0	15.000000	5.763158	6.0	
35	35	2022-04-02 12:00:00	4.0	10.000000	2.000000	10.0	
36	36	2022-04-02 13:00:00	7.0	12.000000	3.000000	1.0	
37	37	2022-04-02 14:00:00	5.0	10.000000	8.000000	0.0	
38	38	2022-04-02 15:00:00	0.0	8.000000	1.000000	7.0	
39	39	2022-04-02 16:00:00	5.0	10.000000	9.000000	4.0	

40 rows × 24 columns

Step 4 : Using Elbow k Means find best k value

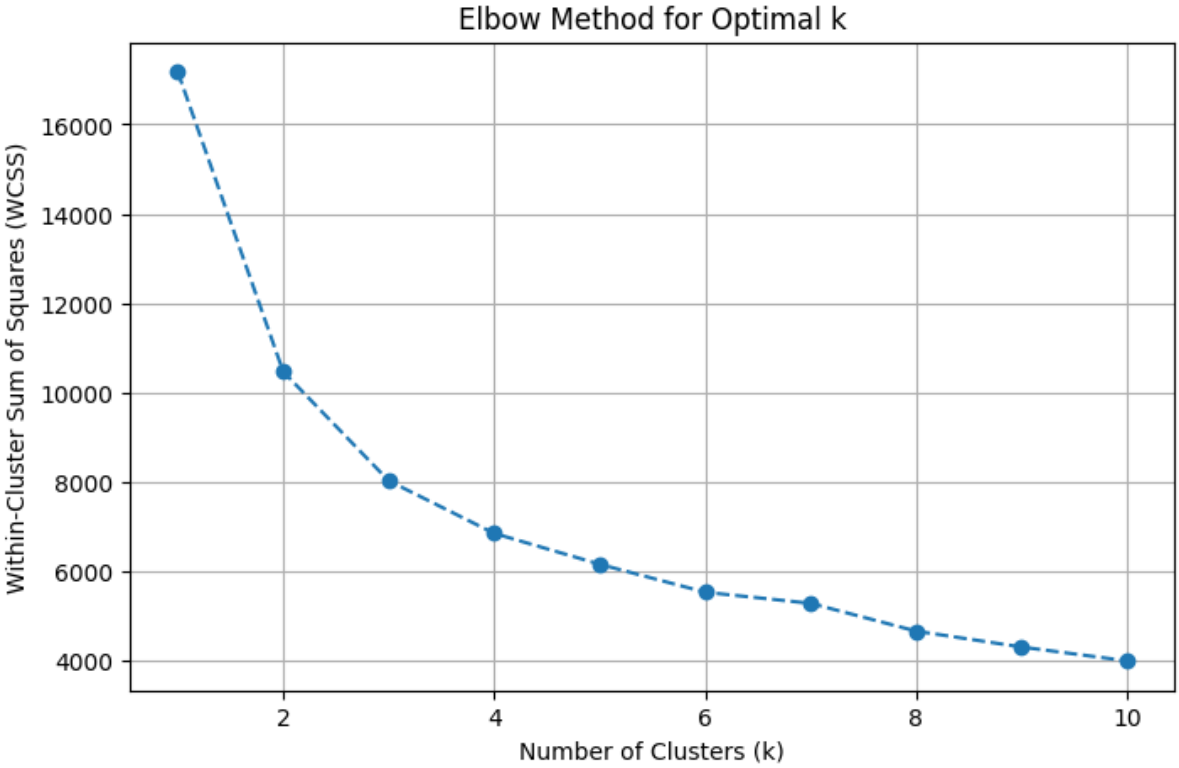
In [98]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

```
columns_to_exclude = ['Unnamed: 0', 'TimeStamp']
df = df.drop(columns=columns_to_exclude)


wcss = []  # Within-cluster sum of squares

for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=0, n_init=10)
    kmeans.fit(df)
    wcss.append(kmeans.inertia_)

# Visualize the Elbow method
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.title('Elbow Method for Optimal k')
plt.grid()
plt.show()
```



Step 5 : Apply K Means using best k-value

In [99]:
```
optimal_k = 3

kmeans = KMeans(n_clusters=optimal_k, random_state=0, n_init=10)
df['cluster'] = kmeans.fit_predict(df)
cluster_column = df['cluster']
df.drop(columns=['cluster'], inplace=True)
df.insert(0, 'cluster', cluster_column)
df
```

Out[99]:

	cluster	Unnamed: 0	(140.7468006 41.8188869)	(140.957261 37.6422006)	(140.4021967 36.555081)	(139.1103334 36.2974922)	(140.13855 35.611391
0	0	0	9.0	3.000000	6.000000	5.0	4.(
1	0	1	0.0	0.000000	8.000000	10.0	-4.(
2	0	2	2.0	7.828571	7.000000	-2.0	2.(
3	0	3	4.0	7.828571	9.000000	1.0	1.(
4	0	4	4.0	7.828571	3.000000	6.0	0.(
5	0	5	-1.0	4.000000	4.000000	-8.0	3.(
6	0	6	6.0	4.000000	4.000000	14.0	2.(
7	0	7	10.0	4.000000	3.000000	-1.0	6.(
8	0	8	3.0	0.000000	4.000000	-5.0	3.(
9	0	9	6.0	9.000000	4.000000	-3.0	2.(
10	0	10	6.0	9.000000	3.000000	0.0	0.(
11	0	11	1.0	7.000000	5.763158	-3.0	6.(
12	0	12	-1.0	11.000000	6.000000	5.0	2.(
13	0	13	2.0	9.000000	8.000000	7.0	6.(
14	0	14	4.0	10.000000	13.000000	3.0	7.(
15	0	15	5.0	11.000000	7.000000	17.0	5.(
16	0	16	10.0	5.000000	8.000000	-2.0	7.(
17	2	17	9.0	2.000000	7.000000	7.0	8.(
18	2	18	1.0	7.828571	7.000000	10.0	9.(
19	2	19	8.0	7.828571	7.000000	-3.0	7.(
20	2	20	2.0	5.000000	8.000000	1.0	8.(
21	2	21	7.0	4.000000	5.000000	0.0	7.(
22	2	22	1.0	3.000000	6.000000	10.0	6.(
23	2	23	7.0	5.000000	6.000000	-3.0	8.(
24	2	24	7.0	5.000000	6.000000	11.0	7.(
25	2	25	14.0	6.000000	5.000000	-4.0	3.(
26	2	26	11.0	6.000000	7.000000	2.0	10.(
27	2	27	14.0	6.000000	5.000000	6.0	9.(
28	2	28	14.0	8.000000	9.000000	3.0	10.(
29	2	29	9.0	4.000000	8.000000	13.0	9.(
30	2	30	8.0	12.000000	4.000000	4.0	1.(
31	2	31	3.0	17.000000	3.000000	11.0	5.(
32	2	32	9.0	20.000000	2.000000	8.0	8.(
33	1	33	12.0	20.000000	4.000000	-2.0	10.(
34	1	34	3.0	15.000000	5.763158	6.0	11.(
35	1	35	4.0	10.000000	2.000000	10.0	9.(
36	1	36	7.0	12.000000	3.000000	1.0	8.(
37	1	37	5.0	10.000000	8.000000	0.0	9.(

	cluster	Unnamed: 0	(140.7468006 41.8188869)	(140.957261 37.6422006)	(140.4021967 36.555081)	(139.1103334 36.2974922)	(140.13855 35.611391
38	1	38	0.0	8.000000	1.000000	7.0	8.(
39	1	39	5.0	10.000000	9.000000	4.0	11.(

40 rows × 24 columns

Step 7 :(Optional) Create geopandas dataframe with geometry and kmeans labels

In [ ]:
```
loc = list(dataframe.dropna(axis=1, how='all').columns[1:])

x_cord = []
y_cord = []
for l in loc:
    x,y = l.replace("(","").replace(")","").split(" ")
    x_cord.append(float(x))
    y_cord.append(float(y))

import geopandas

geometry = geopandas.points_from_xy(x_cord, y_cord)
geo_df = geopandas.GeoDataFrame(kmeans.labels_, geometry=geometry)
geo_df.columns = ['label','geometry']
geo_df.head()
```

Step 8 :(Optional) Create Folium map to view clusters

In [ ]:
```
import folium

map = folium.Map(location = [13.406,80.110], tiles = "Stamen Terrain", zoom_start =
```

Assign colors for each label on the map and add cluster labels to the plot

In [ ]:
```
# Create a geometry list from the GeoDataFrame
geo_df_list = [[point.xy[1][0], point.xy[0][0]] for point in geo_dataframe.geometry


# Iterate through list and add a marker for each unique label.
i = 0
for coordinates in geo_df_list:
    #assign a color marker
    if geo_df.label[i] == 0:
        type_color = "green"
    elif geo_df.label[i] == 1:
        type_color = "blue"
    elif geo_df.label[i] == 2:
        type_color = "orange"
    else:
        type_color = "purple"

     # Place the markers with the popup labels and data
    map.add_child(folium.Marker(location = coordinates,
                                icon = folium.Icon(color = "%s" % type_color)))
    i = i + 1
```