Mini Project 3-Copy1

March 2, 2024

0.1 Task 1 – Your main task is to use K-Means and DBSCAN to do clustering on the given dataset. Your code needs to consider the following aspects, and this also should be reflected in your final report.

[1]:
```
#LIbraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, DBSCAN
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from sklearn.neighbors import NearestNeighbors
```

[2]:
```
# Load data
train_data_path = 'Data/UCI HAR Dataset/UCI HAR Dataset/train/'
test_data_path = 'Data/UCI HAR Dataset/UCI HAR Dataset/test/'
X_train = pd.read_csv('Data/UCI HAR Dataset/UCI HAR Dataset/train/X_train.txt',␣
 ↪delim_whitespace=True, header=None)
y_train = pd.read_csv('Data/UCI HAR Dataset/UCI HAR Dataset/train/y_train.txt',␣
 ↪delim_whitespace=True, header=None)
X_test = pd.read_csv('Data/UCI HAR Dataset/UCI HAR Dataset/test/X_test.txt',␣
 ↪delim_whitespace=True, header=None)
y_test = pd.read_csv('Data/UCI HAR Dataset/UCI HAR Dataset/test/y_test.txt',␣
 ↪delim_whitespace=True, header=None)

print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
print("Training labels shape:", y_train.shape)
print("Testing labels shape:", y_test.shape)
```

```
Training data shape: (7352, 561)
Testing data shape: (2947, 561)
Training labels shape: (7352, 1)
Testing labels shape: (2947, 1)
```

[3]:
```
#Show data
print("Train Data:")
display(pd.DataFrame(X_train).head().style)
```

```
print("\nTest Data:")
display(pd.DataFrame(X_test).head().style)
```

```
Train Data:

<pandas.io.formats.style.Styler at 0x1df93f51f90>


Test Data:

<pandas.io.formats.style.Styler at 0x1df8ea55b50>
```

[4]:
```
# Check for missing values
print("Number of missing values in training data:", X_train.isnull().sum().
    ↪sum())
print("Number of missing values in testing data:", X_test.isnull().sum().sum())
```

```
Number of missing values in training data: 0
Number of missing values in testing data: 0
```

[5]:
```
#Data Types
print('Features\n', X_train.dtypes)
print('\nTest\n', X_test.dtypes)
```

```
Features
 0      float64
1      float64
2      float64
3      float64
4      float64
        …
556    float64
557    float64
558    float64
559    float64
560    float64
Length: 561, dtype: object

Test
 0      float64
1      float64
2      float64
3      float64
4      float64
        …
556    float64
557    float64
558    float64
559    float64
```

```
560    float64
Length: 561, dtype: object
```

[6]:
```python
# Summary of data
display('Train:',X_train.describe())
print('\n')
display('Test',X_test.describe())
```

```
'Train:'

                 0            1            2            3            4  \
count  7352.000000  7352.000000  7352.000000  7352.000000  7352.000000
mean      0.274488    -0.017695    -0.109141    -0.605438    -0.510938
std       0.070261     0.040811     0.056635     0.448734     0.502645
min      -1.000000    -1.000000    -1.000000    -1.000000    -0.999873
25%       0.262975    -0.024863    -0.120993    -0.992754    -0.978129
50%       0.277193    -0.017219    -0.108676    -0.946196    -0.851897
75%       0.288461    -0.010783    -0.097794    -0.242813    -0.034231
max       1.000000     1.000000     1.000000     1.000000     0.916238

                 5            6            7            8            9  …  \
count  7352.000000  7352.000000  7352.000000  7352.000000  7352.000000  …
mean     -0.604754    -0.630512    -0.526907    -0.606150    -0.468604  …
std       0.418687     0.424073     0.485942     0.414122     0.544547  …
min      -1.000000    -1.000000    -1.000000    -1.000000    -1.000000  …
25%      -0.980233    -0.993591    -0.978162    -0.980251    -0.936219  …
50%      -0.859365    -0.950709    -0.857328    -0.857143    -0.881637  …
75%      -0.262415    -0.292680    -0.066701    -0.265671    -0.017129  …
max       1.000000     1.000000     0.967664     1.000000     1.000000  …

               551          552          553          554          555  \
count  7352.000000  7352.000000  7352.000000  7352.000000  7352.000000
mean      0.125293    -0.307009    -0.625294     0.008684     0.002186
std       0.250994     0.321011     0.307584     0.336787     0.448306
min      -1.000000    -0.995357    -0.999765    -0.976580    -1.000000
25%      -0.023692    -0.542602    -0.845573    -0.121527    -0.289549
50%       0.134000    -0.343685    -0.711692     0.009509     0.008943
75%       0.289096    -0.126979    -0.503878     0.150865     0.292861
max       0.946700     0.989538     0.956845     1.000000     1.000000

               556          557          558          559          560
count  7352.000000  7352.000000  7352.000000  7352.000000  7352.000000
mean      0.008726    -0.005981    -0.489547     0.058593    -0.056515
std       0.608303     0.477975     0.511807     0.297480     0.279122
min      -1.000000    -1.000000    -1.000000    -1.000000    -1.000000
25%      -0.482273    -0.376341    -0.812065    -0.017885    -0.143414
50%       0.008735    -0.000368    -0.709417     0.182071     0.003181
75%       0.506187     0.359368    -0.509079     0.248353     0.107659
```

```
max        0.998702     0.996078     1.000000     0.478157     1.000000

[8 rows x 561 columns]



'Test'
                 0            1            2            3            4    \
count  2947.000000  2947.000000  2947.000000  2947.000000  2947.000000
mean      0.273996    -0.017863    -0.108386    -0.613635    -0.508330
std       0.060570     0.025745     0.042747     0.412597     0.494269
min      -0.592004    -0.362884    -0.576184    -0.999606    -1.000000
25%       0.262075    -0.024961    -0.121162    -0.990914    -0.973664
50%       0.277113    -0.016967    -0.108458    -0.931214    -0.790972
75%       0.288097    -0.010143    -0.097123    -0.267395    -0.105919
max       0.671887     0.246106     0.494114     0.465299     1.000000

                 5            6            7            8            9    …  \
count  2947.000000  2947.000000  2947.000000  2947.000000  2947.000000  …
mean     -0.633797    -0.641278    -0.522676    -0.637038    -0.462063  …
std       0.362699     0.385199     0.479899     0.357753     0.523916  …
min      -0.998955    -0.999417    -0.999914    -0.998899    -0.952357  …
25%      -0.976122    -0.992333    -0.974131    -0.975352    -0.934447  …
50%      -0.827534    -0.937664    -0.799907    -0.817005    -0.852659  …
75%      -0.311432    -0.321719    -0.133488    -0.322771    -0.009965  …
max       0.489703     0.439657     1.000000     0.427958     0.786436  …

               551          552          553          554          555  \
count  2947.000000  2947.000000  2947.000000  2947.000000  2947.000000
mean      0.130236    -0.277593    -0.598756     0.005264     0.003799
std       0.231018     0.317245     0.311042     0.336147     0.445077
min      -0.785543    -1.000000    -1.000000    -1.000000    -0.993402
25%      -0.008433    -0.517494    -0.829593    -0.130541    -0.282600
50%       0.142676    -0.311023    -0.683672     0.005188     0.006767
75%       0.288320    -0.083559    -0.458332     0.146200     0.288113
max       1.000000     1.000000     1.000000     0.998898     0.986347

               556          557          558          559          560
count  2947.000000  2947.000000  2947.000000  2947.000000  2947.000000
mean      0.040029    -0.017298    -0.513923     0.074886    -0.048720
std       0.634989     0.501311     0.509205     0.324300     0.241467
min      -0.998898    -0.991096    -0.984195    -0.913704    -0.949228
25%      -0.518924    -0.428375    -0.829722     0.022140    -0.098485
50%       0.047113    -0.026726    -0.729648     0.181563    -0.010671
75%       0.622151     0.394387    -0.545939     0.260252     0.092373
max       1.000000     1.000000     0.833180     1.000000     0.973113
```

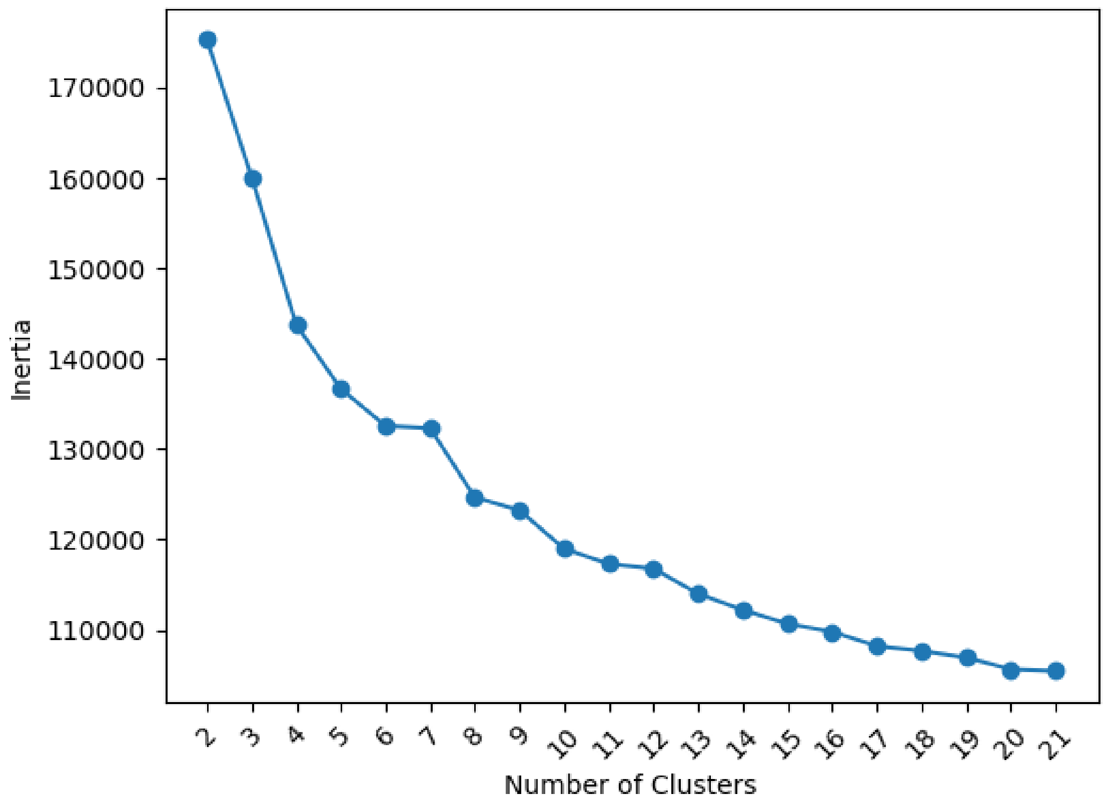
```
[8 rows x 561 columns]
```

[7]:
```
# COnsidering various clusters' number for k-mean
clusters = range(2, 22)
inertia_values = [] #sum of squared distances between each data point and its␣
↪nearest cluster center
for cluster in clusters:
    kmeans = KMeans(n_clusters=cluster, n_init='auto')
    kmeans.fit(X_train)
    inertia_values.append(kmeans.inertia_)
```

[8]:
```
plt.plot(clusters, inertia_values, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.xticks(rotation=45)
plt.xticks(clusters)
```

[8]:
```
([<matplotlib.axis.XTick at 0x1df94bb6910>,
  <matplotlib.axis.XTick at 0x1df94bb4050>,
  <matplotlib.axis.XTick at 0x1df94be5690>,
  <matplotlib.axis.XTick at 0x1df94d28d10>,
  <matplotlib.axis.XTick at 0x1df94d2bdd0>,
  <matplotlib.axis.XTick at 0x1df94bc1350>,
  <matplotlib.axis.XTick at 0x1df94d1f210>,
  <matplotlib.axis.XTick at 0x1df94d16950>,
  <matplotlib.axis.XTick at 0x1df94d14cd0>,
  <matplotlib.axis.XTick at 0x1df94d04bd0>,
  <matplotlib.axis.XTick at 0x1df8ea9b490>,
  <matplotlib.axis.XTick at 0x1df94d06910>,
  <matplotlib.axis.XTick at 0x1df94d02810>,
  <matplotlib.axis.XTick at 0x1df94d017d0>,
  <matplotlib.axis.XTick at 0x1df94cf5590>,
  <matplotlib.axis.XTick at 0x1df94cef510>,
  <matplotlib.axis.XTick at 0x1df94d01510>,
  <matplotlib.axis.XTick at 0x1df94cec110>,
  <matplotlib.axis.XTick at 0x1df94ced2d0>,
  <matplotlib.axis.XTick at 0x1df94ce7f50>],
 [Text(2, 0, '2'),
  Text(3, 0, '3'),
  Text(4, 0, '4'),
  Text(5, 0, '5'),
  Text(6, 0, '6'),
  Text(7, 0, '7'),
  Text(8, 0, '8'),
  Text(9, 0, '9'),
  Text(10, 0, '10'),
  Text(11, 0, '11'),
```

```
 Text(12, 0, '12'),
 Text(13, 0, '13'),
 Text(14, 0, '14'),
 Text(15, 0, '15'),
 Text(16, 0, '16'),
 Text(17, 0, '17'),
 Text(18, 0, '18'),
 Text(19, 0, '19'),
 Text(20, 0, '20'),
 Text(21, 0, '21')])
```



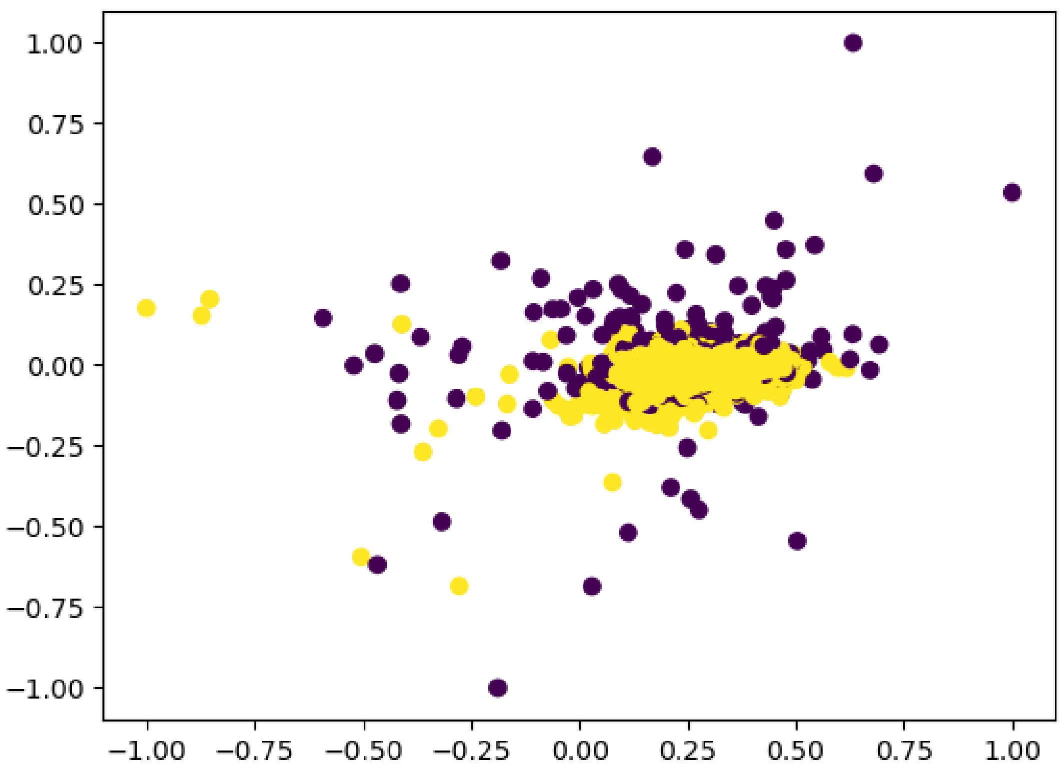
[9]:
```
#Find the optimum number of clusters which is where the inertia does not change more than 10%
inertia_values = []
prev_inertia = float('inf')
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, n_init='auto')
    kmeans.fit(X_train)
    inertia = kmeans.inertia_
    inertia_values.append(inertia)
    if k > 1:
```

```
        relative_change = (prev_inertia - inertia) / prev_inertia
        if relative_change < 0.1:
            break
    prev_inertia = inertia

optimal_num_clusters = k-1
print("Optimal clusters:", optimal_num_clusters)
```

```
Optimal clusters: 2
```

[10]:
```
# Calculation and Visualization of k-means with optimum cluster
X = np.concatenate((X_train, X_test), axis=0)
kmeans = KMeans(n_clusters=optimal_num_clusters,  n_init='auto')
kmeans.fit(X)
cluster_labels = kmeans.labels_

plt.scatter(X[:, 0], X[:, 1], c=cluster_labels)
```

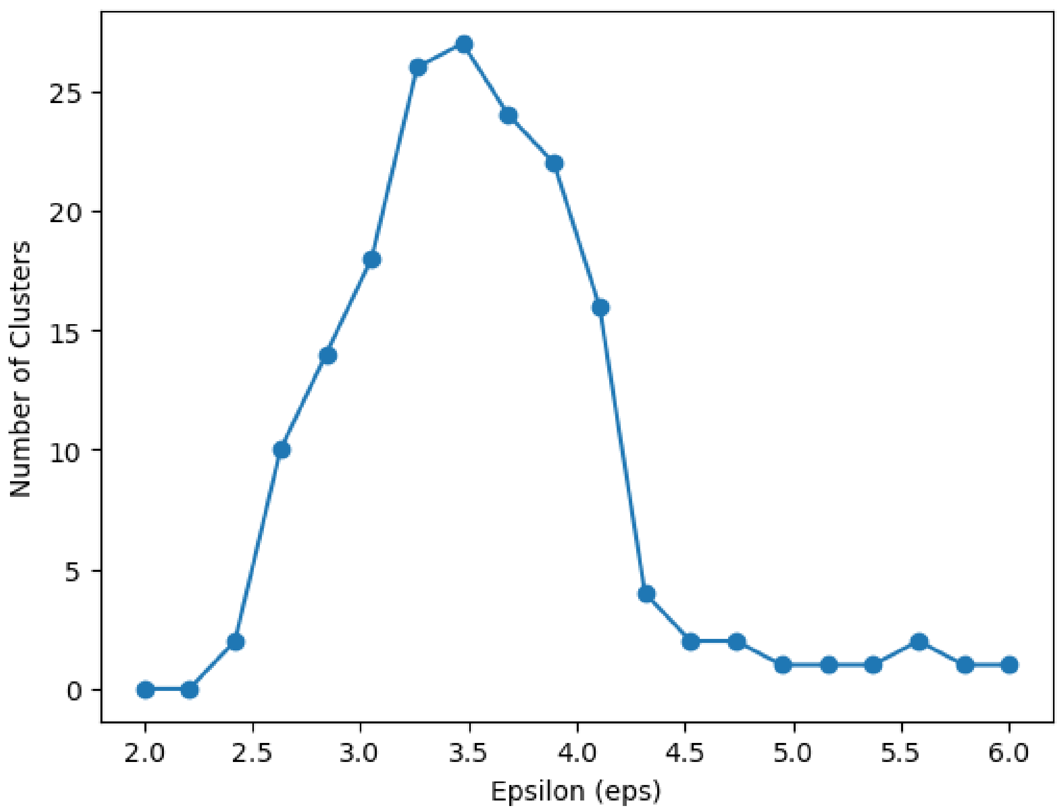
[10]: <matplotlib.collections.PathCollection at 0x1df94bce750>



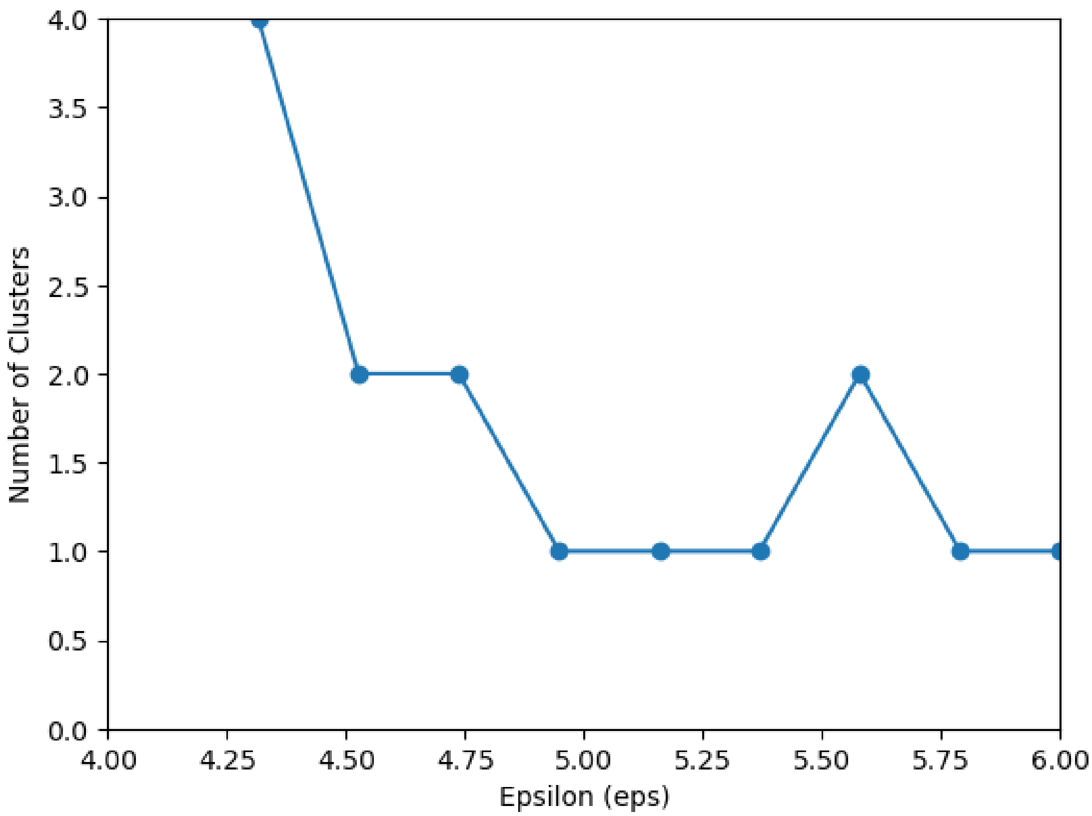
[11]:
```
#dbscan
```

[12]:
```
# Try various epsilon to find optimum one
eps_values = np.linspace(2, 6, 20)
num_clusters = []
for eps8 in eps_values:
    dbscan = DBSCAN(eps=eps8, min_samples=10)
    cluster_labels = dbscan.fit_predict(X)
    n_clusters = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
    num_clusters.append(n_clusters)
```

[13]:
```
plt.plot(eps_values, num_clusters, marker='o')
plt.xlabel('Epsilon (eps)')
plt.ylabel('Number of Clusters')
plt.show()

plt.plot(eps_values, num_clusters, marker='o')
plt.xlabel('Epsilon (eps)')
plt.ylabel('Number of Clusters')
plt.xlim(4, 6)
plt.ylim(0, 4)
```

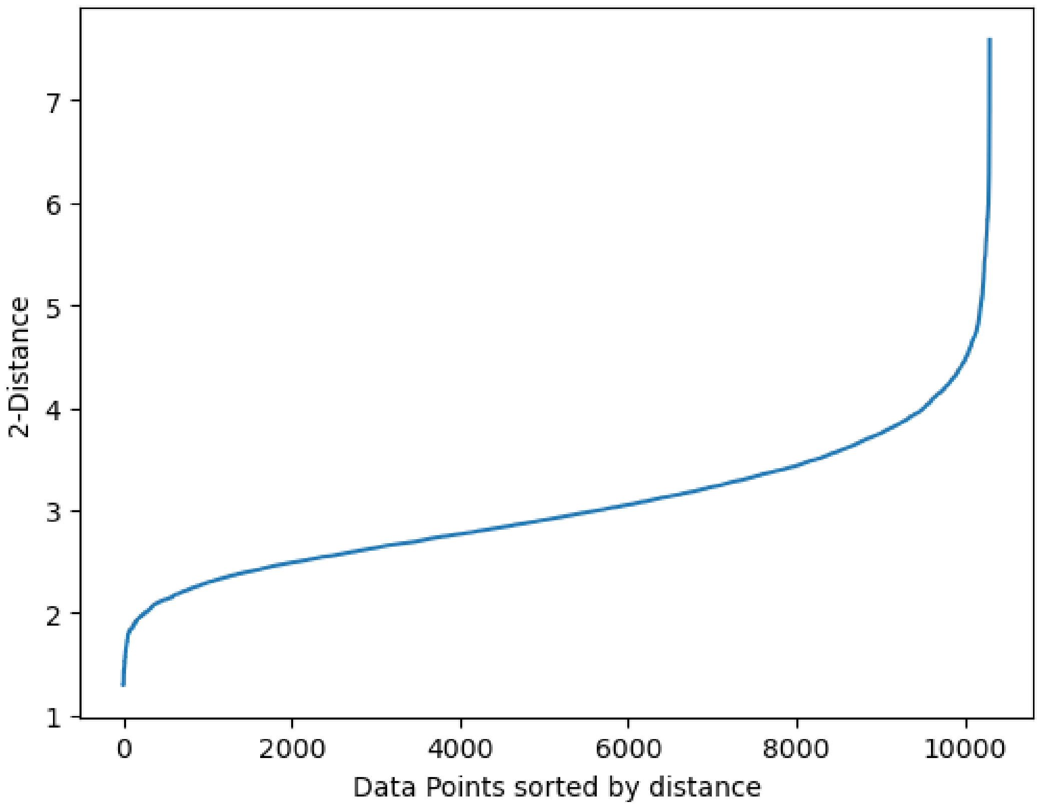


[13]: (0.0, 4.0)



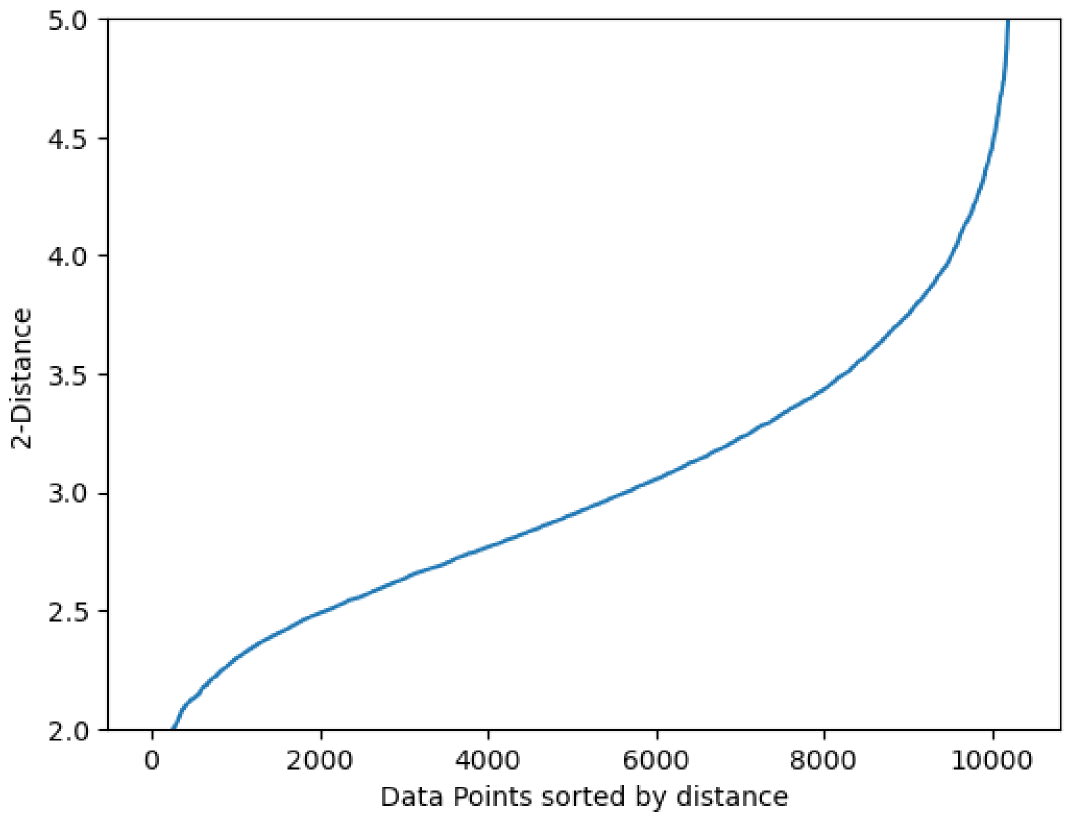
[14]:
```
X = np.concatenate((X_train, X_test), axis=0)
k = 2
#distances to the k-nearest neighbors
nn = NearestNeighbors(n_neighbors=k)
nn.fit(X)
distances, _ = nn.kneighbors(X)
sorted_distances = np.sort(distances[:, -1])
```

[15]:
```
plt.plot(np.arange(len(X)), sorted_distances)
plt.xlabel('Data Points sorted by distance')
plt.ylabel(f'{k}-Distance')
plt.show()
plt.plot(np.arange(len(X)), sorted_distances)
plt.xlabel('Data Points sorted by distance')
plt.ylabel(f'{k}-Distance')
plt.ylim(2,5)
```



2-Distance
Data Points sorted by distance
1
2
3
4
5
6
7
0
2000
4000
6000
8000
10000

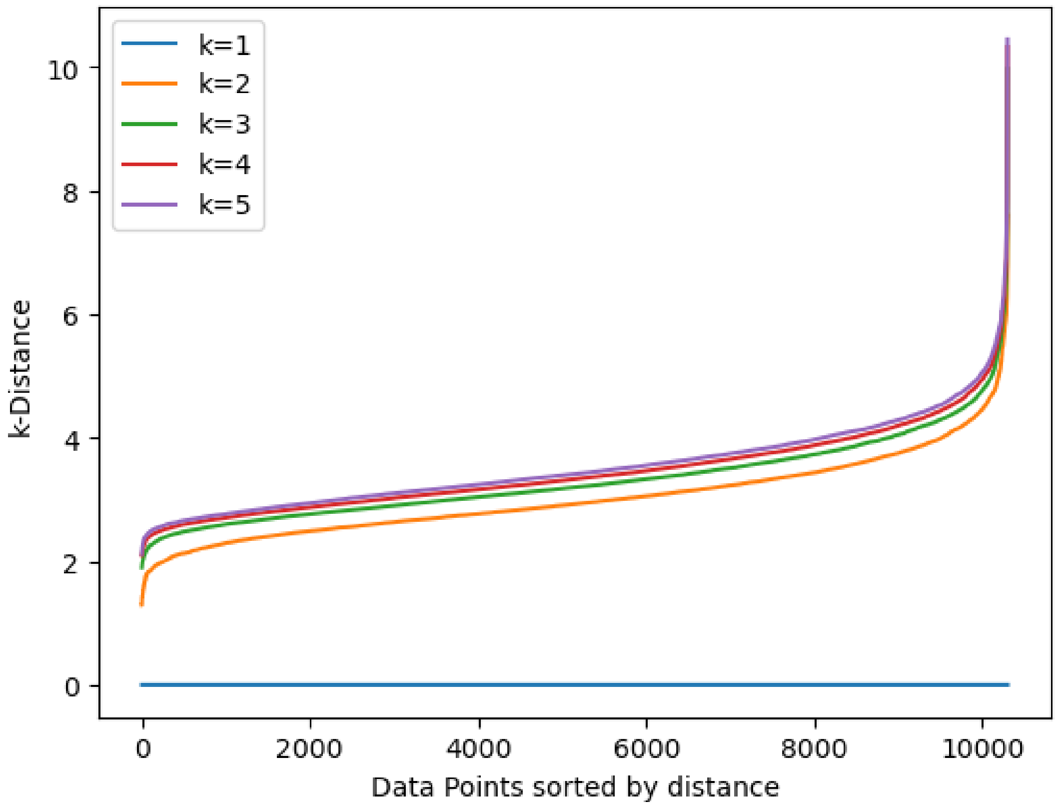

```
[15]: (2.0, 5.0)
```



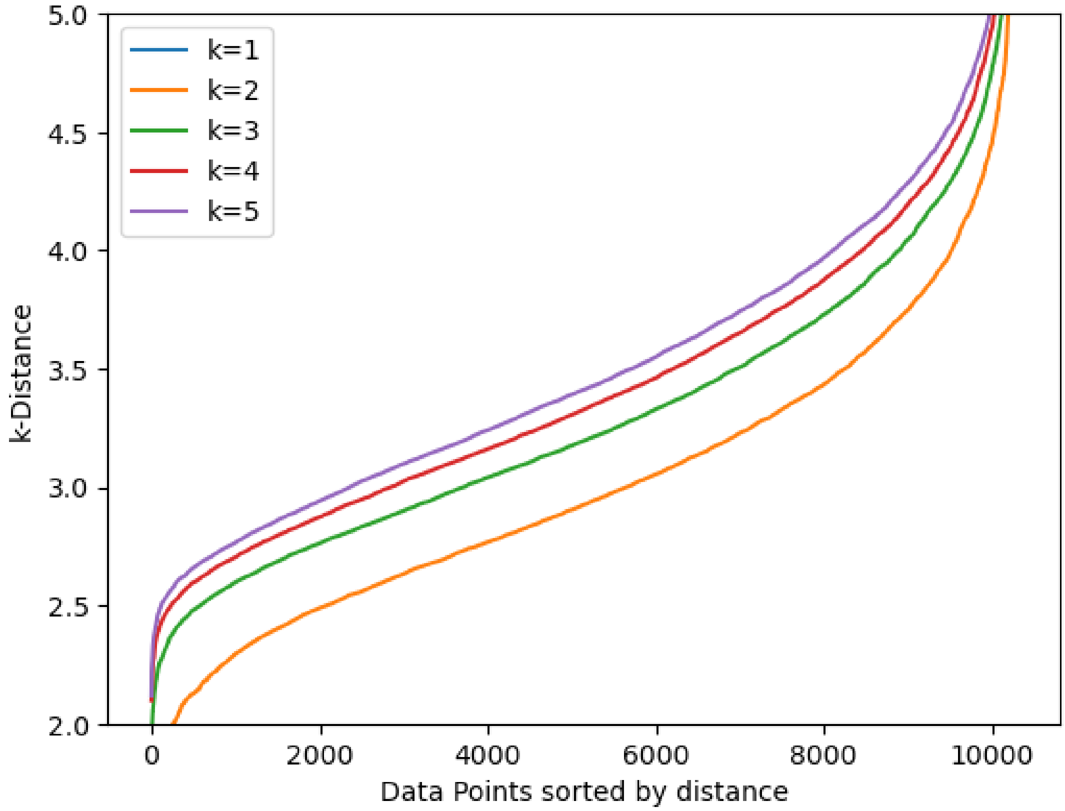
[16]:
```
#Various k
X = np.concatenate((X_train, X_test), axis=0)
k_values = range(1, 6)
sorted_distances_dict = {}
for k in k_values:
    nn = NearestNeighbors(n_neighbors=k)
    nn.fit(X)
    distances, _ = nn.kneighbors(X)
    sorted_distances = np.sort(distances[:, -1])
    sorted_distances_dict[k] = sorted_distances
```

[17]:
```
#plt.figure(figsize=(12, 8))
for k in sorted_distances_dict:
    plt.plot(np.arange(len(X)), sorted_distances_dict[k], label=f'k={k}')

plt.xlabel('Data Points sorted by distance')
plt.ylabel('k-Distance')
plt.legend()
plt.show()
for k in sorted_distances_dict:
    plt.plot(np.arange(len(X)), sorted_distances_dict[k], label=f'k={k}')
```

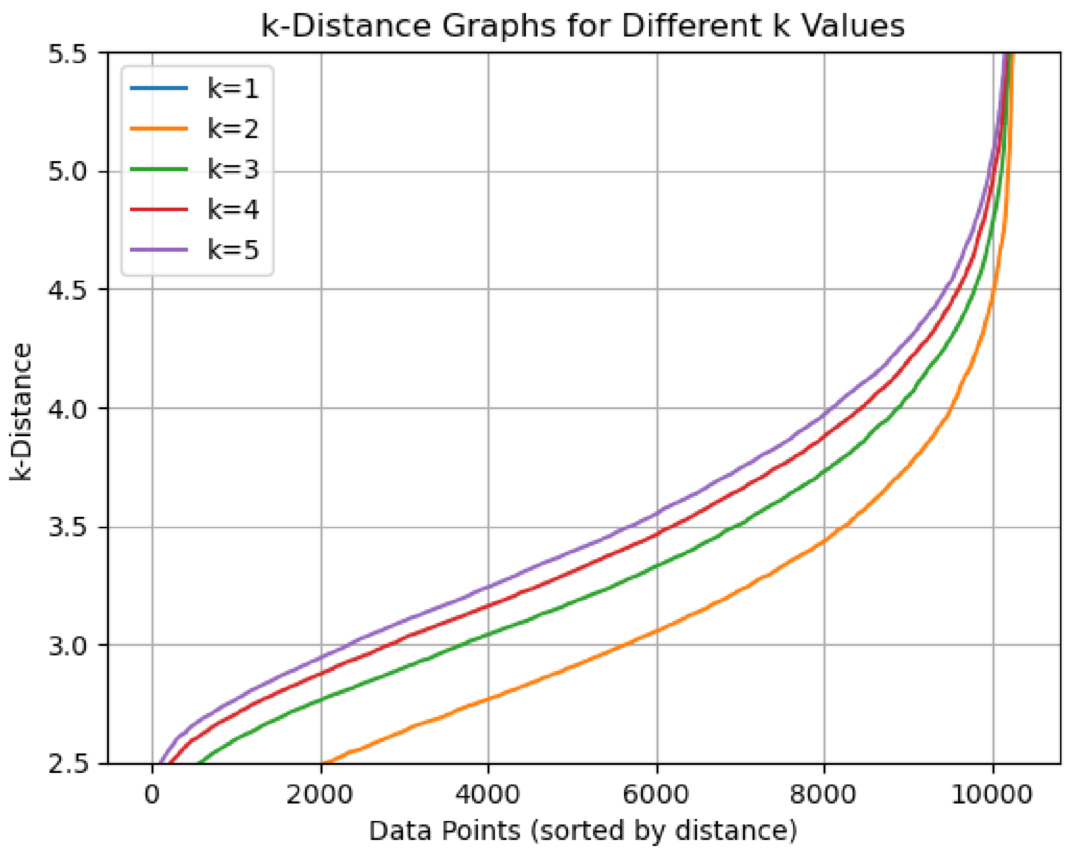
```
plt.xlabel('Data Points sorted by distance')
plt.ylabel('k-Distance')
plt.legend()
plt.ylim(2,5)
```



[17]: (2.0, 5.0)



[18]:
```python
# Plot k-distance graphs
for k in sorted_distances_dict:
    plt.plot(np.arange(len(X)), sorted_distances_dict[k], label=f'k={k}')

plt.xlabel('Data Points (sorted by distance)')
plt.ylabel('k-Distance')
plt.title('k-Distance Graphs for Different k Values')
plt.ylim(2.5, 5.5)  # Set y-axis limit between 0 and 3
plt.legend()
plt.grid(True)
plt.show()
```

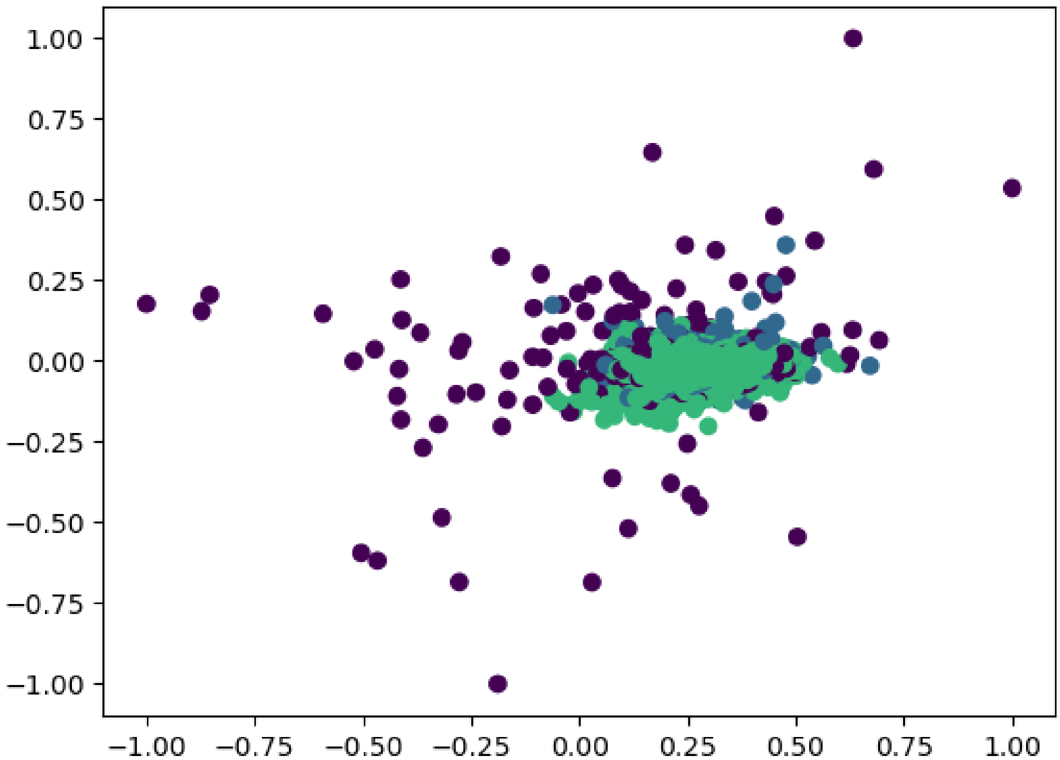


[19]:
```
#DBSCAN
dbscan = DBSCAN(eps=4.5, min_samples=10)
cluster_labels = dbscan.fit_predict(X)
n_clusters_ = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
n_noise_ = list(cluster_labels).count(-1)
print("Estimated number of clusters:", n_clusters_)
print("Estimated number of noise points:", n_noise_)
```

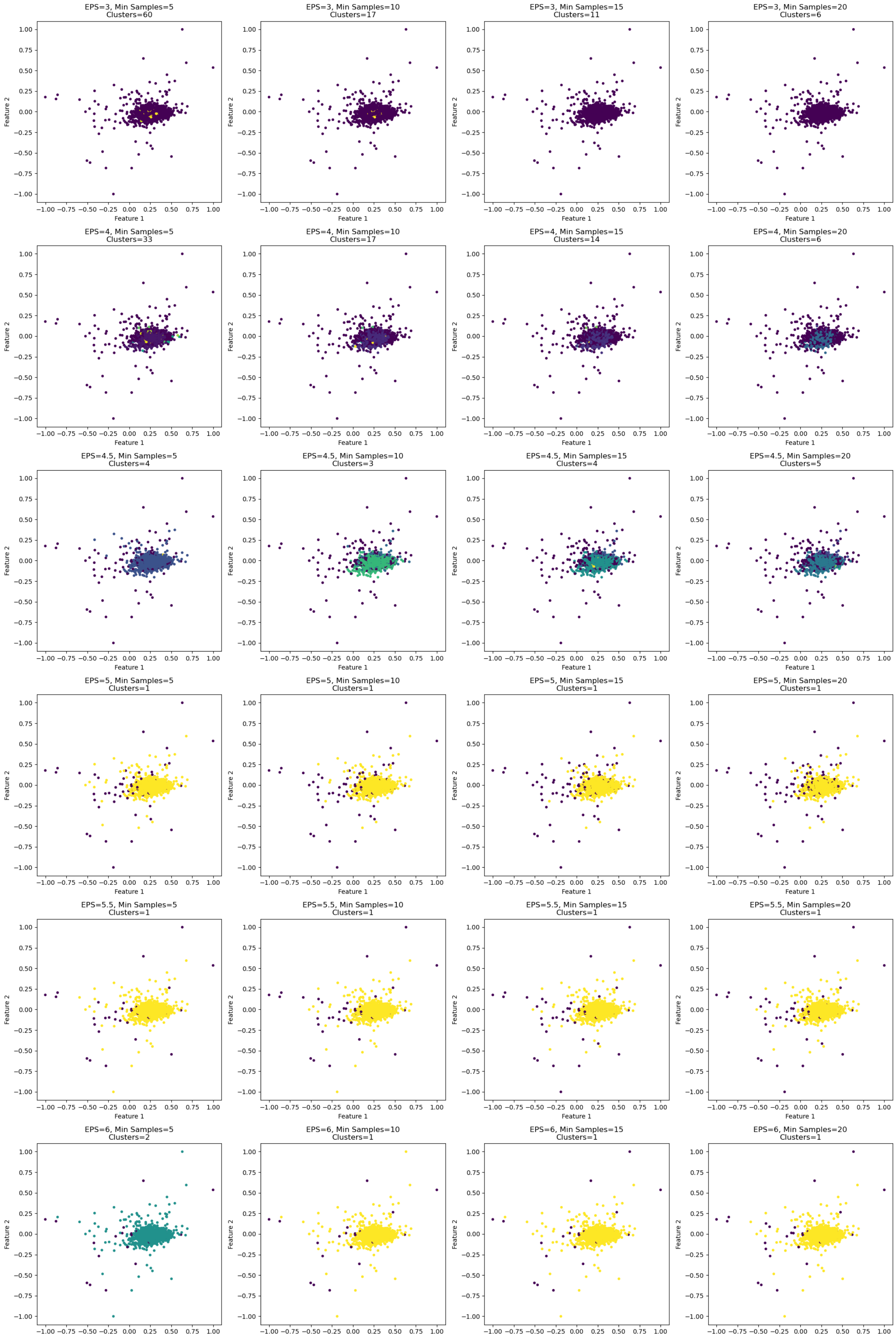
```
Estimated number of clusters: 3
Estimated number of noise points: 654
```

[20]:
```
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels)
```

[20]:
```
<matplotlib.collections.PathCollection at 0x1df94e72410>
```



[21]:
```
#Various parameters
eps_values = [3,4,4.5, 5,5.5, 6]
min_samples_values = [5, 10, 15, 20]
fig, axs = plt.subplots(len(eps_values), len(min_samples_values), figsize=(20,␣
↪30))
for i, eps in enumerate(eps_values):
    for j, min_samples in enumerate(min_samples_values):
        # Perform DBSCAN clustering
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        cluster_labels = dbscan.fit_predict(X)
        n_clusters_ = len(set(cluster_labels)) - (1 if -1 in cluster_labels␣
↪else 0)
        axs[i, j].scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis',␣
↪s=10)
        axs[i, j].set_title(f'EPS={eps}, Min␣
↪Samples={min_samples}\nClusters={n_clusters_}')
        axs[i, j].set_xlabel('Feature 1')
        axs[i, j].set_ylabel('Feature 2')
plt.tight_layout()
plt.show()
```

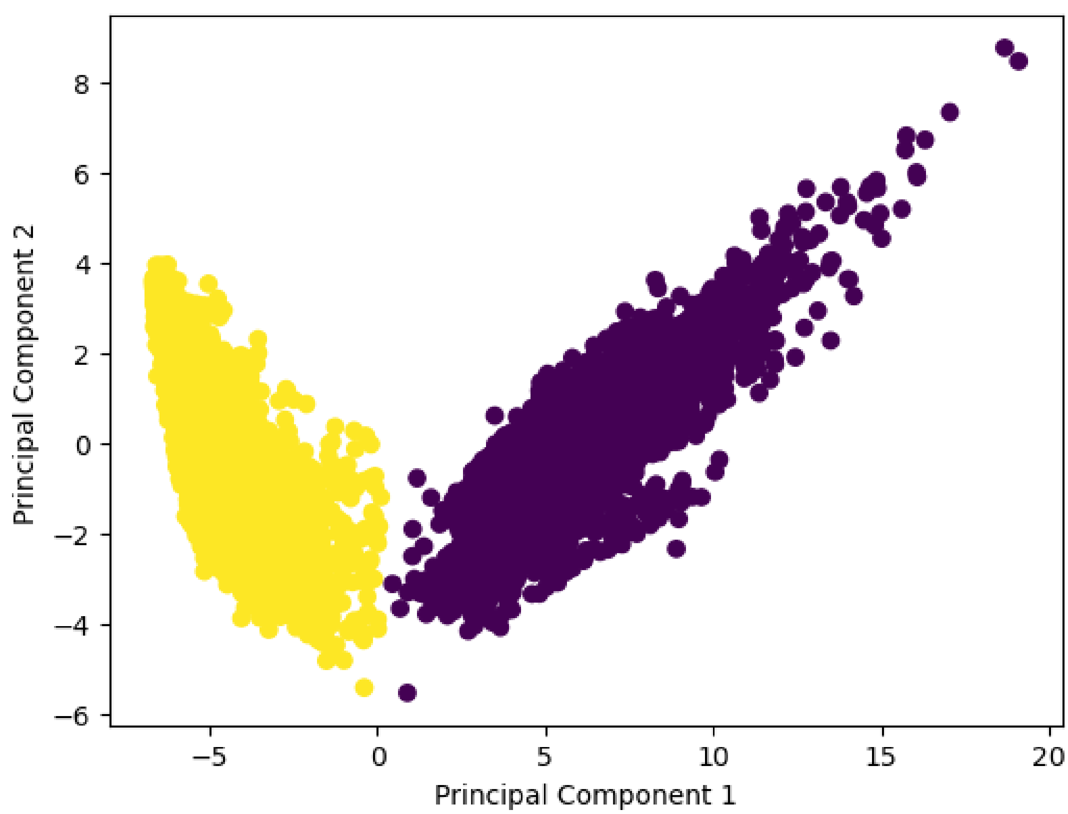

EPS=3, Min Samples=5
Clusters=60
EPS=3, Min Samples=10
Clusters=17
EPS=3, Min Samples=15
Clusters=11
EPS=3, Min Samples=20
Clusters=6
EPS=4, Min Samples=5
Clusters=33
EPS=4, Min Samples=10
Clusters=17
EPS=4, Min Samples=15
Clusters=14
EPS=4, Min Samples=20
Clusters=6
EPS=4.5, Min Samples=5
Clusters=4
EPS=4.5, Min Samples=10
Clusters=3
EPS=4.5, Min Samples=15
Clusters=4
EPS=4.5, Min Samples=20
Clusters=5
EPS=5, Min Samples=5
Clusters=1
EPS=5, Min Samples=10
Clusters=1
EPS=5, Min Samples=15
Clusters=1
EPS=5, Min Samples=20
Clusters=1
EPS=5.5, Min Samples=5
Clusters=1
EPS=5.5, Min Samples=10
Clusters=1
EPS=5.5, Min Samples=15
Clusters=1
EPS=5.5, Min Samples=20
Clusters=1
EPS=6, Min Samples=5
Clusters=2
EPS=6, Min Samples=10
Clusters=1
EPS=6, Min Samples=15
Clusters=1
EPS=6, Min Samples=20
Clusters=1
Feature 1
Feature 2

0.2 Task 2 – Use a dimensionality reduction technique before using K-Means and DBSCAN on the dataset.

[22]:
```
X = np.concatenate((X_train, X_test), axis=0)
# PCA to reduce dimensionality
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
kmeans = KMeans(n_clusters=2, n_init='auto')
cluster_labels = kmeans.fit_predict(X_pca)
```

[23]:
```
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

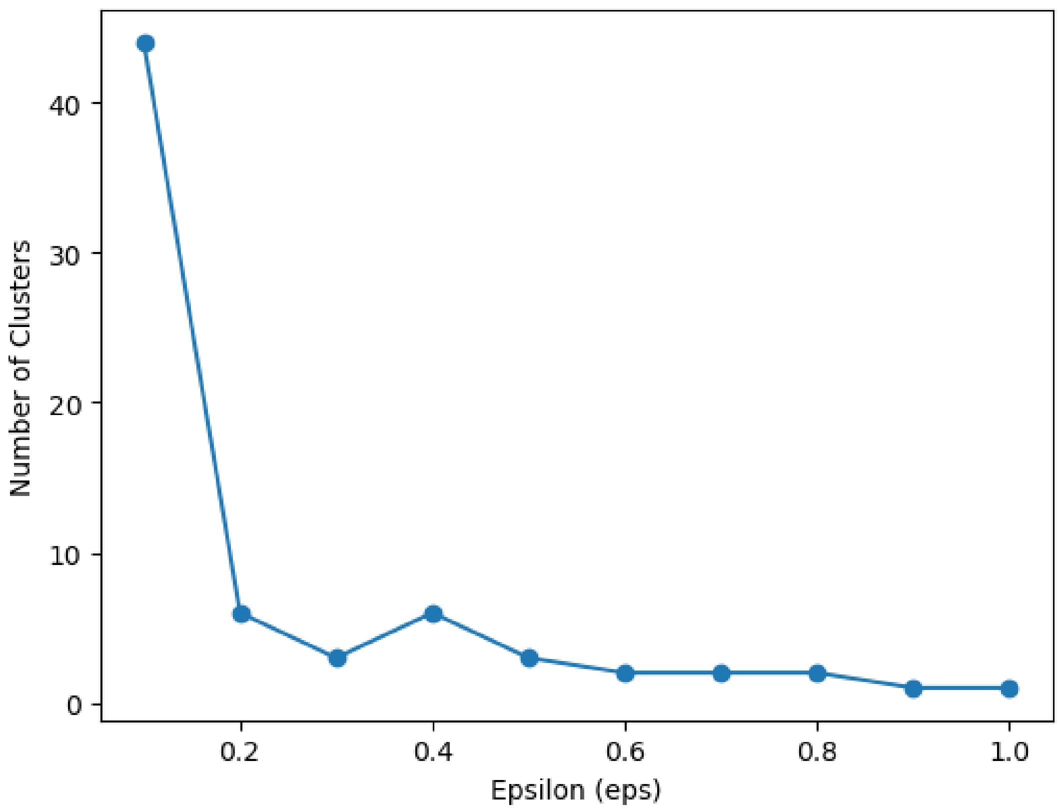
[23]: Text(0, 0.5, 'Principal Component 2')



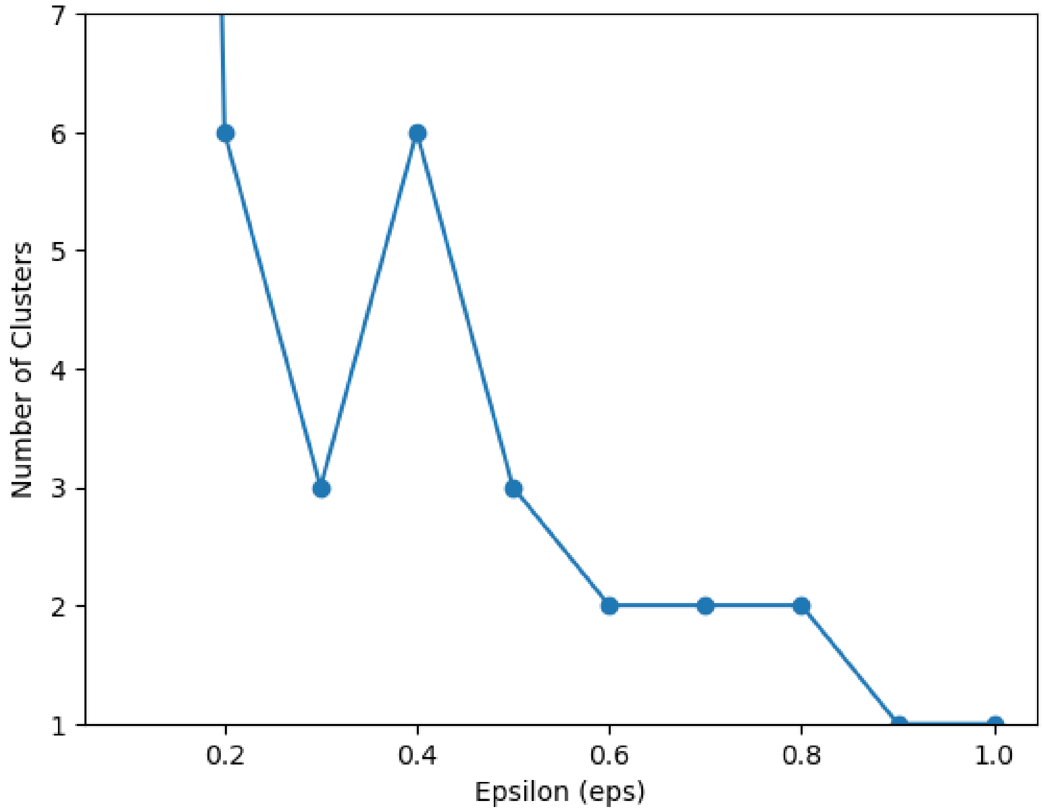
[24]:
```
# Various epsilon
eps_values = np.linspace(0.1, 1.0, 10)
```

```
num_clusters = []
for eps in eps_values:
    dbscan = DBSCAN(eps=eps, min_samples=15)
    cluster_labels = dbscan.fit_predict(X_pca)
    n_clusters = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
    num_clusters.append(n_clusters)
```

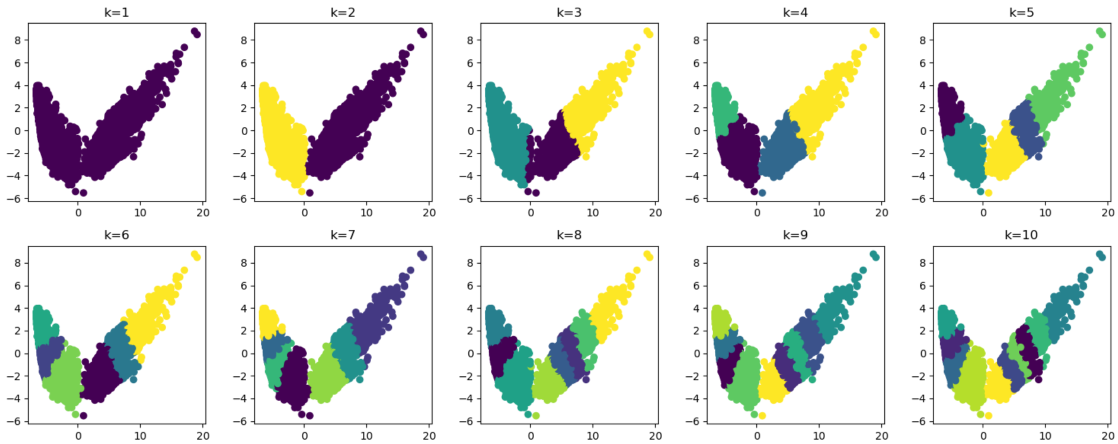
[25]:
```
plt.plot(eps_values, num_clusters, marker='o')
plt.xlabel('Epsilon (eps)')
plt.ylabel('Number of Clusters')
plt.show()

plt.plot(eps_values, num_clusters, marker='o')
plt.xlabel('Epsilon (eps)')
plt.ylabel('Number of Clusters')
plt.ylim(1, 7)
```



[25]: (1.0, 7.0)



[26]:
```
#Various k
X = np.concatenate((X_train, X_test), axis=0)
k_values = range(1, 11)
subplot_index = 1
plt.figure(figsize=(15, 6))
for k in k_values:
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)
    kmeans = KMeans(n_clusters=k, n_init='auto')
    cluster_labels = kmeans.fit_predict(X_pca)
    plt.subplot(2, 5, subplot_index)
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels)
    plt.title(f'k={k}')
    subplot_index += 1
plt.tight_layout()
```



[27]:
```
#Various samples
X = np.concatenate((X_train, X_test), axis=0)
n_clusters = 2
plt.figure(figsize=(20, 15))
subplot_index = 1
min_samples_range = range(5, 21, 5)

# silhouette scores for comparison
silhouette_scores_kmeans = []
silhouette_scores_kmeans_pca = []
silhouette_scores_dbscan = []
silhouette_scores_dbscan_pca = []
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

for min_samples in min_samples_range:
    kmeans = KMeans(n_clusters=n_clusters, n_init='auto')
    cluster_labels_kmeans = kmeans.fit_predict(X)
    silhouette_scores_kmeans.append(silhouette_score(X, cluster_labels_kmeans))
    kmeans_pca = KMeans(n_clusters=n_clusters, n_init='auto')
    cluster_labels_kmeans_pca = kmeans_pca.fit_predict(X_pca)
    silhouette_scores_kmeans_pca.append(silhouette_score(X_pca,
 ↪cluster_labels_kmeans_pca))

    dbscan = DBSCAN(eps=4.5, min_samples=min_samples)
    cluster_labels_dbscan = dbscan.fit_predict(X)
    silhouette_scores_dbscan.append(silhouette_score(X, cluster_labels_dbscan))
    dbscan_pca = DBSCAN(eps=0.6, min_samples=min_samples)
    cluster_labels_dbscan_pca = dbscan_pca.fit_predict(X_pca)

    if len(np.unique(cluster_labels_dbscan_pca)) > 1:
```

```
        silhouette_scores_dbscan_pca.append(silhouette_score(X_pca,␣
↪cluster_labels_dbscan_pca))
    else:
        silhouette_scores_dbscan_pca.append(None)

    plt.subplot(4, 4, subplot_index)
    plt.scatter(X[:, 0], X[:, 1], c=cluster_labels_dbscan)
    plt.title(f'DBSCAN without PCA (Min Samples={min_samples})')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    subplot_index += 1
    plt.subplot(4, 4, subplot_index)
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cluster_labels_dbscan_pca)
    plt.title(f'DBSCAN with PCA (Min Samples={min_samples})')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    subplot_index += 1

plt.tight_layout()
print("Silhouette Scores:")
print("K-Means without PCA:", silhouette_scores_kmeans)
print("K-Means with PCA:", silhouette_scores_kmeans_pca)
print("DBSCAN without PCA:", silhouette_scores_dbscan)
print("DBSCAN with PCA:", silhouette_scores_dbscan_pca)
```

```
Silhouette Scores:
K-Means without PCA: [0.48107627299425504, 0.48107627299425504,
0.48107627299425504, 0.48107627299425504]
K-Means with PCA: [0.7553142586315335, 0.7553142586315335, 0.7553142586315335,
0.7553142586315335]
DBSCAN without PCA: [-0.20844844080567973, 0.19351593638437486,
0.13572259357059774, 0.07467116570943297]
DBSCAN with PCA: [0.6995498624322384, 0.7119896931125205, 0.7104702967350589,
0.7017838874945116]
```