Q1

May 10, 2024

0.1 Mahdi Anvari 610700002 Homework 2 of Machine Learning Question 1

[187]:
```
# importing libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import linkage, fcluster
from sklearn.cluster import SpectralClustering
from sklearn import metrics
import pandas as pd
from sklearn.metrics import adjusted_rand_score
import seaborn as sbn
```

a. Cluster the data using the following methods. Use the same number of clusters as specified in the dataset. For each method, describe what distance or similarity metric you have used.

b. K-means

ii. Hierarchical clustering (average linkage)

iii. Hierarchical clustering (single linkage)

iv. Hierarchical clustering (complete linkage)

v. Spectral clustering. Describe how you defined the graph.

[24]:
```
# Pathbased dataset
# N=300, k=3, D=2
```

[25]:
```
PathbasedDataset = np.loadtxt('pathbased.txt')
PathbasedScaler = StandardScaler()
ScaledPathbasedDataset = PathbasedScaler.fit_transform(PathbasedDataset[:,:2])
print(PathbasedDataset[:,2])
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 3. 1. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
```

```
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
```

[26]:
```
# K-means
PathbasedKmeans = KMeans(n_clusters=3, n_init = 10)
PathbasedKmeans.fit(ScaledPathbasedDataset)
PathbasedClusters = PathbasedKmeans.labels_
print(PathbasedClusters)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1]
```

[27]:
```
# Hierarchical clustering (average linkage)
PathbasedAverageLinkage = linkage(ScaledPathbasedDataset, method='average',␣
 ↪metric='euclidean')
PathbasedClusters2 = fcluster(PathbasedAverageLinkage, t=3 ,␣
 ↪criterion='maxclust')
print(PathbasedClusters2)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2]
```

[28]:
```
# Hierarchical clustering (single linkage)
PathbasedSingleLinkage = linkage(ScaledPathbasedDataset, method='single',␣
 ↪metric='euclidean')
PathbasedClusters3 = fcluster(PathbasedSingleLinkage, t=3 ,␣
 ↪criterion='maxclust')
print(PathbasedClusters3)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1]
```

[29]:
```
# Hierarchical clustering (complete linkage)
PathbasedCompleteLinkage = linkage(ScaledPathbasedDataset, method='complete',␣
 ↪metric='euclidean')
PathbasedClusters4 = fcluster(PathbasedCompleteLinkage, t=3 ,␣
 ↪criterion='maxclust')
print(PathbasedClusters4)
```

```
[3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3]
```

[30]:
```
# Spectral clustering
PathbasedSpectral = SpectralClustering(n_clusters= 3)
PathbasedClusters5 = PathbasedSpectral.fit_predict(ScaledPathbasedDataset)
print(PathbasedClusters5)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2]
```

[31]:
```
# Spiral dataset
# N=312, k=3, D=2
```

[32]:
```
SpiralDataset = np.loadtxt('spiral.txt')
SpiralScaler = StandardScaler()
ScaledSpiralDataset = SpiralScaler.fit_transform(SpiralDataset[:,:2])
print(SpiralDataset[:,2])
```

```
[3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
```

```
3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 3.
3. 3. 3. 3. 3. 3. 3. 3. 3. 3. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2. 2. 2.
2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
```

[33]:
```
# K-means
SpiralKmeans = KMeans(n_clusters=3 , n_init=10)
SpiralKmeans.fit(ScaledSpiralDataset)
SpiralClusters = SpiralKmeans.labels_
print(SpiralClusters)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1]
```

[34]:
```
# Hierarchical clustering (average linkage)
SpiralAverageLinkage = linkage(ScaledSpiralDataset, method='average',␣
 ↪metric='euclidean')
SpiralClusters2 = fcluster(SpiralAverageLinkage, t=3 , criterion='maxclust')
print(SpiralClusters2)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
```

[35]:
```
# Hierarchical clustering (single linkage)
SpiralSingleLinkage = linkage(ScaledSpiralDataset, method='single',␣
 ↪metric='euclidean')
SpiralClusters3 = fcluster(SpiralSingleLinkage, t=3 , criterion='maxclust')
print(SpiralClusters3)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
```

[36]:
```
# Hierarchical clustering (complete linkage)
SpiralCompleteLinkage = linkage(ScaledSpiralDataset, method='complete',␣
 ↪metric='euclidean')
SpiralClusters4 = fcluster(SpiralCompleteLinkage, t=3 , criterion='maxclust')
print(SpiralClusters4)
```

```
[3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2
 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

[37]:
```
# Spectral clustering
SpiralSpectral = SpectralClustering(n_clusters= 3)
SpiralClusters5 = SpiralSpectral.fit_predict(ScaledSpiralDataset)
print(SpiralClusters5)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2]
```

[38]:
```
# Jain dataset
# N=373, k=2, D=2
```

[39]:
```
JainDataset = np.loadtxt('jain.txt')
JainScaler = StandardScaler()
ScaledJainDataset = JainScaler.fit_transform(JainDataset[:,:2])
print(JainDataset[:,2])
```

```
[2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

[40]:
```python
# K-means
JainKmeans = KMeans(n_clusters=2 , n_init=10)
JainKmeans.fit(ScaledJainDataset)
JainClusters = JainKmeans.labels_
print(JainClusters)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```

[41]:
```python
# Hierarchical clustering (average linkage)
JainAverageLinkage = linkage(ScaledJainDataset, method='average',␣
 ↪metric='euclidean')
JainClusters2 = fcluster(JainAverageLinkage, t=2 , criterion='maxclust')
print(JainClusters2)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2]
```

[42]:
```
# Hierarchical clustering (single linkage)
JainSingleLinkage = linkage(ScaledJainDataset, method='single',␣
 ↪metric='euclidean')
JainClusters3 = fcluster(JainSingleLinkage, t=2 , criterion='maxclust')
print(JainClusters3)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
```

[44]:
```
# Hierarchical clustering (complete linkage)
JainCompleteLinkage = linkage(ScaledJainDataset, method='complete',␣
 ↪metric='euclidean')
JainClusters4 = fcluster(JainCompleteLinkage, t=2 , criterion='maxclust')
print(JainClusters4)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1]
```

[45]:
```
# Spectral clustering
JainSpectral = SpectralClustering(n_clusters= 2)
JainClusters5 = JainSpectral.fit_predict(ScaledJainDataset)
print(JainClusters5)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
```

[46]:
```
# Flame dataset
# N=240, k=2, D=2
```

[47]:
```
FlameDataset = np.loadtxt('flame.txt')
FlameScaler = StandardScaler()
ScaledFlameDataset = FlameScaler.fit_transform(FlameDataset[:,:2])
print(FlameDataset[:,2])
```

```
[1. 1. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 2.
 2. 2. 2. 2. 2. 2. 2. 2. 2. 2. 1. 1. 1. 1. 1. 1. 1. 1. 1. 2. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

[48]:
```
# K-means
FlameKmeans = KMeans(n_clusters=2 , n_init=10)
FlameKmeans.fit(ScaledFlameDataset)
FlameClusters = FlameKmeans.labels_
print(FlameClusters)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

[49]:
```
# Hierarchical clustering (average linkage)
FlameAverageLinkage = linkage(ScaledFlameDataset, method='average',␣
 ↪metric='euclidean')
FlameClusters2 = fcluster(FlameAverageLinkage, t=2 , criterion='maxclust')
print(FlameClusters2)
```

```
[2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 2 1 1 1 1 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 1 1 1 1 1 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

[50]:

```python
# Hierarchical clustering (single linkage)
FlameSingleLinkage = linkage(ScaledFlameDataset, method='single',␣
 ↪metric='euclidean')
FlameClusters3 = fcluster(FlameSingleLinkage, t=2 , criterion='maxclust')
print(FlameClusters3)
```

```
[2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

[51]:

```python
# Hierarchical clustering (complete linkage)
FlameCompleteLinkage = linkage(ScaledFlameDataset, method='complete',␣
 ↪metric='euclidean')
FlameClusters4 = fcluster(FlameCompleteLinkage, t=2 , criterion='maxclust')
print(FlameClusters4)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

[52]:

```python
# Spectral clustering
FlameSpectral = SpectralClustering(n_clusters= 2)
FlameClusters5 = FlameSpectral.fit_predict(ScaledFlameDataset)
print(FlameClusters5)
```

```
[1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

[53]:

```python
# S1 dataset
```

```python
# Synthetic 2-d data with N=5000 vectors and k=15 Gaussian clusters with␣
↪different degree of cluster overlap
```

[70]:
```python
S1Dataset = np.loadtxt('s1.txt')
S1Scaler = StandardScaler()
ScaledS1Dataset = S1Scaler.fit_transform(S1Dataset)
```

[63]:
```python
# K-means
S1Kmeans = KMeans(n_clusters=15 , n_init=10)
S1Kmeans.fit(ScaledS1Dataset)
S1Clusters = S1Kmeans.labels_
print(S1Clusters[:1000])
```

```
[10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10  7 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10  7 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10  7 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  5  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  5  0  0  6  6
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
```

```
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 11 11 11 11 11 11
 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11]
```

[64]:
```
# Hierarchical clustering (average linkage)
S1AverageLinkage = linkage(ScaledS1Dataset, method='average',␣
 ↪metric='euclidean')
S1Clusters2 = fcluster(S1AverageLinkage, t=15 , criterion='maxclust')
print(S1Clusters2[:1000])
```

```
[4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 2 2 2 2 2 7 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 7 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 7 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 7 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
 6]
```

[65]:
```
# Hierarchical clustering (single linkage)
S1SingleLinkage = linkage(ScaledS1Dataset, method='single', metric='euclidean')
S1Clusters3 = fcluster(S1SingleLinkage, t=15 , criterion='maxclust')
print(S1Clusters3[:1000])
```

```
[10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
```

```
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 11
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12]
```

[66]:

```python
# Hierarchical clustering (complete linkage)
S1CompleteLinkage = linkage(ScaledS1Dataset, method='complete',␣
 ↪metric='euclidean')
S1Clusters4 = fcluster(S1CompleteLinkage, t=15 , criterion='maxclust')
print(S1Clusters4[:1000])
```

```
[9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
 9 9 9 9 5 5 5 5 5 6 5 5 5 5 5 5 6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 7 6 5 5 5 5
 5 5 5 5 6 5 5 5 5 5 6 6 5 5 5 5 5 5 5 5 5 6 5 5 5 5 6 5 5 5 5 5 5 5 5 5 5
 5 5 5 6 6 5 5 5 5 7 5 6 5 5 5 5 5 5 5 5 5 5 6 5 5 6 5 5 5 5 6 5 5 5 5 6 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 5 5 5 5 6
 6 6 6 5 5 5 5 5 6 5 5 5 5 5 5 5 5 5 5 6 5 5 5 5 6 5 5 6 5 5 5 5 5 6 5 5 5
 5 6 5 5 5 5 5 5 5 5 5 5 5 5 6 5 6 5 5 5 5 5 5 5 5 6 6 6 5 5 5 5 5 6 5 5 5
 5 5 5 6 5 5 5 5 5 5 6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 5 5 6 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 5 5 5 5 5 5 5 5 6 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 6 5 5 5 5 5 5 5 5 5 5 5 5 6 5 5 5 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2]
```

[68]:

```python
# Spectral clustering
S1Spectral = SpectralClustering(n_clusters= 15)
S1Clusters5 = S1Spectral.fit_predict(ScaledS1Dataset)
print(S1Clusters5[:1000])
```

```
[12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 13 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 13 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 13 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  8  6  6  6  6  6  6  6  6
```

```
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  8  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6
  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6  6 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10  1 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10  1 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10  1 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14]
```

[69]:
```
# S4 dataset
# Synthetic 2-d data with N=5000 vectors and k=15 Gaussian clusters with␣
↪different degree of cluster overlap
```

[71]:
```
S4Dataset = np.loadtxt('s4.txt')
S4Scaler = StandardScaler()
ScaledS4Dataset = S4Scaler.fit_transform(S4Dataset)
```

[74]:
```
# K-means
S4Kmeans = KMeans(n_clusters=15 , n_init=10)
S4Kmeans.fit(ScaledS4Dataset)
S4Clusters = S4Kmeans.labels_
print(S4Clusters[:1000])
```

```
[ 4  4  7  9  4  4  4  9  4  4  4  4  7  4  4  4  7  4  7  9  6 10  4  4
  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  7  4 10  4
  4  4  7  6  7  4  4  4  9  4  7  4  4  4  4  4  4  4  4  7  4  6  4  4
  4  4  4  4  4  4  4  4  4  7  6  4  4  9  4 10  4  9  4  4  4  4  4  4
  9  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4 10  4  4  4  4  4  4  4
```

```
 9  4  4  4  4  4  4  4  4  4  9  7  4  4  4  7  4  7  9  4  9  4  4  4
 4  4  4  4  4  4  9  4  4 10  4  4 10  4  4  4  7  4 10  4  4  4  4  9
 4  4  4  4  4  4  4  4  4  4  7  4  4  4  4  4  4  4  4  4  4  4  4  9
 7  4  4 10  9  4  4  4  4 10  4  9  4  7  4  4  4  4  4  4  4  4  4  4
 4  4  4  4  4  4  4  4  4  4  7  4  4  4  4  4  4  4  4  4  4  4  4  4
 4  4  4  7  4  4  4  4  4  4  4  4  4  4  4  4  7  4  4 10  7  4  4  4
 4  7  4  7  4  4  4  4  4  4  4  4  9  6  4  4  4  4  4  4  4  7  4  4
 4  7  7  4  4  4  4  4  6  4  4  4 13 13 13 13 13 13 13  9 11 13 13 13
 6 13  3  0 13 13 13 13 13 13 13 13 13 13 13 13 13  6 13  6 13 13 13  9
13  6 13 13 13 13 13  0 13 13 13  9  6 13  0 13 13 13  6 13 13 13  9 13
13 13  3  9 13 13  3 13 13 13 13 13 13 13  6 13  3 13  3  3 13 13 13 13
 9 13  0 13 13 13 13 13 13 13 13 13 13 13 13  0 13 13 13 13  6 13  0 13
 3  3 13 13 13  6 13 13 13 13 13  3 13 13  6 13 13 13 13 13 13 13 13 13
13 13  9 13 13 13 13 13 13 13  9  6  3 13  3 13 13  3 13 13  3 13 13 13
 9 13  3 13 13 13 13 13 13 13 13  3 13 13 13 13 13 13 13 13 13 13 11 13
13 13 13  0 13 13 13 13  3 13 13 13 13 13 13 13 13 13 13 13 13 13 13  9
13  3 13 13  6 13 13  6 13  6 13 13  6 13 13 13 13  3  9 13  3 13 13 13
13 13 13 13 13 13 13  6  9 13  3 13 13 13  0  6 13 13 13  3 13 13 13 13
13 13 13 13 13 13 13 13  9  3 13 13 13  3 13 13  6 13 13 13  0  0 13 13
13 11 13 13  6 13  6 13 13 13 13 13  6 13 13 13 13 13 13  0 13 13  6 13
13 13 13 13 13 13 13 13 13 13  6  0  3 13 13 13  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2 11  2  2  2  2 11  2  2  2  2  2  2  2
 2  2  2  2 11  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2 11  2 11  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2 11  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2 11  2 11  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2 11  2  2  2
 2  2  2  2  2  2  2 10 10 10 10 10  7 10 10 10  2 10 10 10 10 10 10 10
10 10 10  4 10 10 10 10 10 10  2 10  4 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10  2 10  2]
```

[75]:
```
# Hierarchical clustering (average linkage)
S4AverageLinkage = linkage(ScaledS4Dataset, method='average',␣
 ↪metric='euclidean')
S4Clusters2 = fcluster(S4AverageLinkage, t=15 , criterion='maxclust')
print(S4Clusters2[:1000])
```

```
[12 13 13 12 13 13 13 12 13 12 13 13 13 13 13 13 13 13 12 13 10 13 13
 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 10 13
 13 12 13 15 13 13 13 13 12 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
 13 13 13 12 13 13 13 13 13 13 13 13 13 12 13 10 13 12 13 13 13 12 13 13
```

```
12 13 13 13 13 13 13 13 13 13 13 13 13 13 13 10 13 13 13 13 13 13 13
12 13 13 13 13 13 13 13 13 13 12 13 13 13 13 13 13 13 13 13 12 13 13 13
13 13 13 13 13 13 12 13 13 10 13 13 10 13 13 13 13 13 10 13 13 13 13 12
13 13 13 13 13 13 13 13 13 13 13 13 12 13 13 13 13 13 13 13 13 13 13 12
13 13 13 10 13 13 13 13 13 10 13 13 13 13 13 13 12 13 13 13 13 13 13 13
12 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 10 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 13 13 13 13 15 13 13 13  4  4  4  4  4  4  4 12 15  4  4  4
15  4  2 15  4  4  4  4  4  4  4  4  4  4  4  4  4  4 15  4 15  4  4  4 12
 4 15  4  4  4  4  4 15  4  5  4 12 15  4 15  4 15 15 15  4  4  4  4  4
15  4  2 12  4  2  2  4  4  4  4  4  4  4 15 15  5  4  5  2  4  4  4  4
12  4 15  4  4 15  4  4 15  4  4  4  4  4  4 15  4  4  4  4 15  4 15  4
 2  5  4  4  4 13  4  4 15  4  4  2  4  5 15  4  4  4  4  4  4  4  4  4
 4 15 12  4  4  4  4  4  4  5 12 15  2  4  2  4  4  2  4  4  2  4  4  4
 4  4  2  4  6  4  4  4  4  4  4  2  4  4 15  4  4  4  4  4 15  4 14  4
 4  4  4 15 15  5  4  4  2  4  4 15  4  4  4  2  4  4  4 15  5  4  4  4
 4  2  4  4 15  4  4 15  4 15  4  4 15  4  4  4 15  2 15  4  2  4  4  4
 4  4  4  4  4  4  4 15 12  4  2  4  4  4  2 15  5  4  4  5  4 15 15  4
 4  4  5  4  4  5  4 15 12  5  4  4  4  5 15  4 15  4  4  4 15 15  4  5
 4 14  4  4 15 15 15  4  4  4  4  4 15  4  5  4  4  4  4 15  4  4 13  4
 4  4  4  4  4  4  4  4  4  4 15  2  5  4  4  4 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 14 11 11 11 11 14 11 11 11 11 11 11 11
11 11 11 14 14 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 14 11 14 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 14 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 14 11 14 11 11 11 11 11 11 11 14 11 11 11 11 11 11
11 11 11 11 11 11 14 11 11 11 14 14 11 14 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 14
11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 14 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 14 11 11 11
11 11 11 11 11 11 11 10 10 10 10 10 13 10 10 10 10 10 10 10 10 10 11 10
10 10 10 13 10 10 10 10 10 10 10 10 13 10 10 10 10 10 10 10 11 10 10 10
10 10 11 10 10 10 10 10 10 10 10 10 10 11 10 11]
```

[76]:
```
# Hierarchical clustering (single linkage)
S4SingleLinkage = linkage(ScaledS4Dataset, method='single', metric='euclidean')
S4Clusters3 = fcluster(S4SingleLinkage, t=15 , criterion='maxclust')
print(S4Clusters3[:1000])
```

```
[ 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
```

```
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5 11  5  5  5  5  5  3  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  4  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5 15  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  4  5  5  5  5  5  5  5  5  5  5  5  5  5  5  4  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  3  5  5  5  5
 5  5  4  5  5  5  5  5  5  3  5  5  5  5  5  5  5  5  5  5  5  5  5  4
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  4  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  3  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5
 5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5]
```

[77]:

```
# Hierarchical clustering (complete linkage)
S4CompleteLinkage = linkage(ScaledS4Dataset, method='complete',␣
 ↪metric='euclidean')
S4Clusters4 = fcluster(S4CompleteLinkage, t=15 , criterion='maxclust')
print(S4Clusters4[:1000])
```

```
[ 8  8  9  1  8  8  8  1  8  8  8  9  9  9  8  8  4  4  4  1  4  9  9  8
  8  8  8  8  8  8  8  8  8  4  8  8  8  8  9  8  8  8  8  8  9  8  9  8
  8  8  4  4  9  8  4  8  1  8  4  8  8  8  8  8  4  8  9  9  8  4  8  8
```

```
 4  8  9  1  8  4  8  8  8  4  4  4  8  1  8  9  8  1  1  8  8  1  8  8
 1  8  8  8  8  9  8  8  8  8  8  8  8  8  8  8  9  9  8  8  8  1  8  8  8
 1  8  8  8  8  8  8  8  8  8  1  9  8  8  8  9  8  9  1  8  1  8  8  8
 8  8  8  9  8  8  1  8  9  9  8  8  9  8  9  8  4  8  9  8  8  8  8  1
 8  8  8  8  8  8  4  8  8  8  9  9  8  8  8  8  8  8  9  8  8  8  4  1
 4  8  1  9  4  8  8  8  8  9  4  1  8  4  8  8  8  8  9  8  9  8  8  1
 8  4  8  8  8  8  9  8  8  8  9  8  8  8  8  1  4  4  1  8  8  8  8  8
 8  8  8  4  9  8  8  9  8  8  8  8  8  8  8  8  9  4  8  9  4  8  8  8
 8  4  8  9  8  8  8  8  8  8  8  8  1  4  8  1  8  1  8  4  8  4  8  8
 8  9  4  8  8  4  8  8  4  1  8  8  2  2  2  2  2  2  2  1  5  2  2  2
 4  2 10  4  2  4  2  3  2  2  3  2  2  2  2  4  2  4  2  4  2  2  2  1
 2  2  2  3  2  2  2  4  2  3  3  1  4  2  5  2  4  2  4  2  3  2  2  2
 4  2 10  1  2 10 10  2  2  2  2  2  2  2  4  4  3  2 10 10  2  2  2  2
 1  2  5  2  2  4  2  2  2  2  2  2  3  2  2  5  2  2  2  2  4  3  4  2
10 10  3  2  4  4  2  2  2  2  2 10  2  3  4  2  2  2  2  2  2  2  3  2
 2  4  1  2  2  3  2  2  2  3  1  4 10  2 10  2  2 10  2  2 10  2  2  2
 2  2 10  2  3  2  2  2  2  2  2 10  2  2  2  2  3  2  2  2  4  2  4  2
 2  2  2  4  2  3  2  2 10  4  2  4  2  2  2 10  2  2  2  4  3  2  2  2
 2 10  2  2  4  2  2  4  2  4  2  2  4  2  3  2  2 10  1  2 10  2  2  2
 2  2  4  2  2  2  3  4  1  2 10  2  2  2 10  4  3  2  2  3  2  2  2  2
 2  2  3  3  2  3  2  4  1  3  4  2  2 10  4  2  4  2  4  2  5  5  2  3
 2  6  3  4  4  2  4  2  2  2  2  2  4  2  3  2  2  3  2  4  2  2  4  2
 3  2  2  3  2  2  2  4  2  2  4 10  3  2  2  2  7  7  7  7  7  7  7  7
 7  7  7  7  7  7  7  7  7  7  7  6  7  7  7  7  6  7  7  7  7  7  7  7
 7  7  7  6  6  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  6  7  6  7  7  7  7  7
 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  6  7
 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
 7  7  7  7  7  7  7  6  7  6  7  7  7  7  7  7  7  6  7  7  7  7  7  7
 7  7  7  7  7  7  7  7  7  7  7  6  7  6  7  7  7  7  7  7  7  7  7  7
 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
 7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
 7  7  6  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  6  7  7  7
 7  7  7  7  7  7  7  9  9  9  9  9  9  9  9  9  9  9  9  9  9  7  9
 9  9  9  9  9  9  9  9  9  9  9  9  8  9  9  9  9  9  9  9  7  9  9  9
 9  9  7  7  9  9  9  9  9  9  9  9  9  7  9  7]
```

[78]:
```
# Spectral clustering
S4Spectral = SpectralClustering(n_clusters= 15)
S4Clusters5 = S4Spectral.fit_predict(ScaledS4Dataset)
print(S4Clusters5[1:1000])
```

```
[14  3  5 14 14 14  5 14 14  3  3  3  3 14 14  3  3  3  5  5 12  3 14 14
 14 14 14 14 14 14 14 14  3 14 14 14 14  3 14 14 14 14 14  3 14 12  3 14
 14  5  5  3 14  3 14  5 14  5  3 14 14 14 14  3 14  3  3 14  5 14 14  3
```

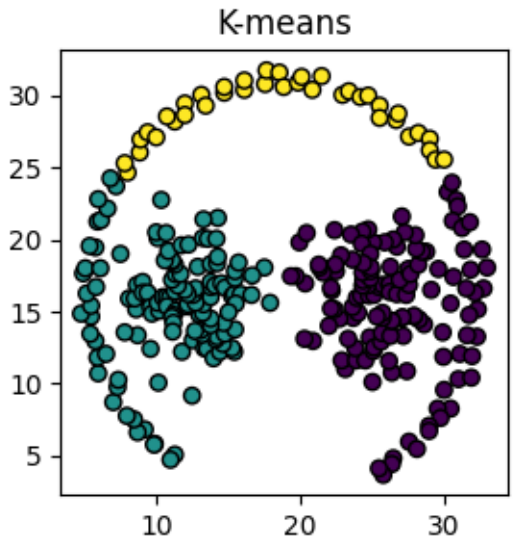
```
14  3 14 14  5 14 14 14  3  5  3  3 13 14  3 14 14  5 14 14 14 14  3  5
14 14 14 14 14 14 14 14 14 14 14 14 14 14  3  3 14 14 14 14 14 14 14  5
14 14 14 14 14 14 14  3 14 13  3 14 14 14  3 14  3  5 14 14 14 14 14 14
14 14  3 14 14  5 14  3 12 14 14  3 14  3 14  3 14  3 14 14 14 14 13 14
14  3 14 14 14  3 14 14 14  3  3 14 14 14 14 14 14  3 14 14 14  3  5  3
14  5  3  5 14 14 14 14 12  5  5 14  5 14 14 14 14 14 14  3 14 14 14 14
 5 14 14 14 14  3 14 14 14  3 14 14 14 14 14  5  3 14 14 14 14 14 14 14
14 14  3 14 14 14  3 14 14 14 14 14 14 14 14  3  3 14  3  5 14 14 14 14
 3 14  3 14  3 14 14 14 14 14  3  5  5 14  5 14  5 14  5 14  3 14 14 14
 3  3 14 14  3 14 14  5 14 14 14  9  9  9  9  9  9  9  5  2  9  9  9  5
13  7  2  9  9  9  9  9  9  9  9  9  9  9  9 11  5  9  5  9  9  9  5 11
 5  9 11 11  9  9  2  9  4  9  5  5  9  2  9  9  5  5  9  9  9 13  9  9
 9  7  5  9  9  7  9  9  9 11 13  9  9  5  5  7  9  7  9  9  9  9  9  5
 9  2  9  9  5  9  9  9  9  9  9 11  9  9  2  9  9  9 11  5 11  2  9  2
 7 11  9  9  5  9  9  5  9  9  2  9 11  5  9  9  9  9  9  9 11  9  9 13
 9  5  9  9  9  9  9  9  4 13  5  2 13  7  9  9  2  9  9  7  9 13  9 13
 9  7 11  4  9  9  9  9  9  9  7  9  9  5  9 11  9 11  9  9 13  2 13  9
 9  9  5  9  4  9  9  7  9  9  5  9  9  9  2  9 11  9  9  4  9  9 13  9
 7  9  9  5 11  9  5  9  5  9 11  5  9  9 13  9  7 13  9  7  9  9  9  9
 9  9  9  9  9 11  5 13  9  2  9  9  9  2  5  4  9 13  7  9  5  5  9  9
 9  4  9  9  9  9  5  5  9  9  9  9  7  5  9  5  9  9  9  2  2  9  4  9
 6 11  9  5  9  5  9  9  9  9  9  5  9  4  9  9 11  9  2  9  9  5  9 11
 9  9  9  9  9  9  9  9  9  5  2  4  9  9  9  1  1  1  1  1  1  1  1  1
 1  1  1  1  1  1  1  1  1  1  6  1  1  1  6  6  6  1  1  1  1  1  1  1
 1  1  6  6  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  6  1  6  1  1  1  1  1  1
 1  1  1  1  1  1  1  6  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 1  1  1  1  6  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  6  1  1
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 1  1  1  1  1  1  6  1  6  1  1  1  1  1  1  1  6  1  1  1  1  1  1  1
 1  1  1  1  1  6  1  1  1  6  6  1  6  1  1  1  1  1  1  1  1  1  1  1
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  6  1
 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 1  6  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  6  1  1  1  1
 1  1  1  1  1  1 12 12 12 12 12  3 12 12  3  1 12  1 12 12 12 12 12 12
12  3  3 12 12 12 12 12 12  1 12 14 12  3 12 12 12 12 12 12 12 12 12 12
 3 12 12 12  3 12 12 12 12 12 12 12 12 12  1]
```

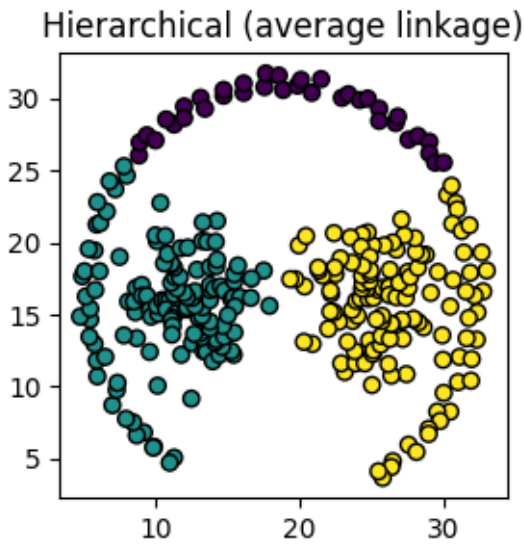
b. Visualize the resulting clustering (use the scatter plot of the data points and color the points by cluster assignment).

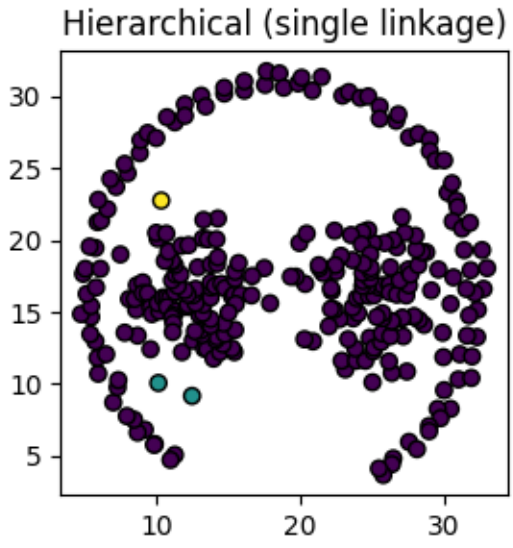
[99]:
```
# Pathbased dataset
# N=300, k=3, D=2
```

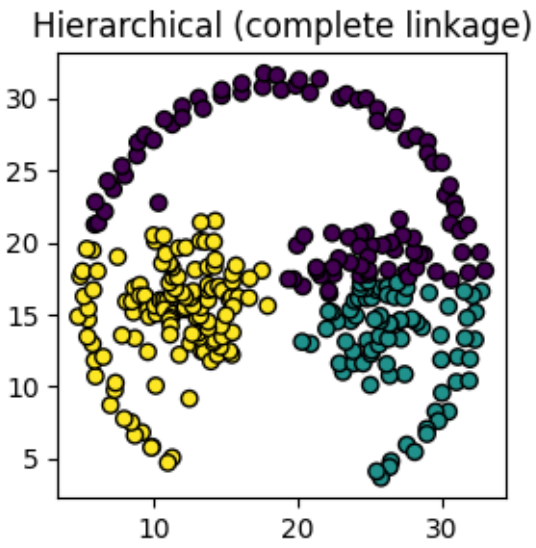
[122]:
```
plt.figure(figsize=(3, 3))
```

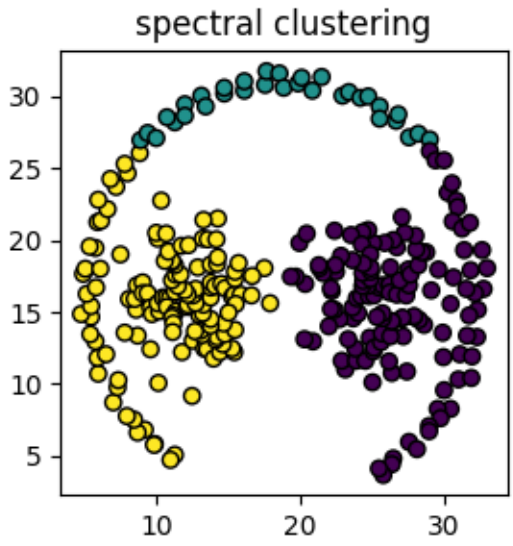
```
plt.scatter(PathbasedDataset[:,0], PathbasedDataset[:,1], c=PathbasedClusters,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("K-means")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(PathbasedDataset[:,0], PathbasedDataset[:,1], c=PathbasedClusters2,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (average linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(PathbasedDataset[:,0], PathbasedDataset[:,1], c=PathbasedClusters3,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (single linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(PathbasedDataset[:,0], PathbasedDataset[:,1], c=PathbasedClusters4,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (complete linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(PathbasedDataset[:,0], PathbasedDataset[:,1], c=PathbasedClusters5,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("spectral clustering")
plt.show()
```




Hierarchical (average linkage)
30
25
20
15
10
5
10
20
30


Hierarchical (single linkage)
30
25
20
15
10
5
10
20
30

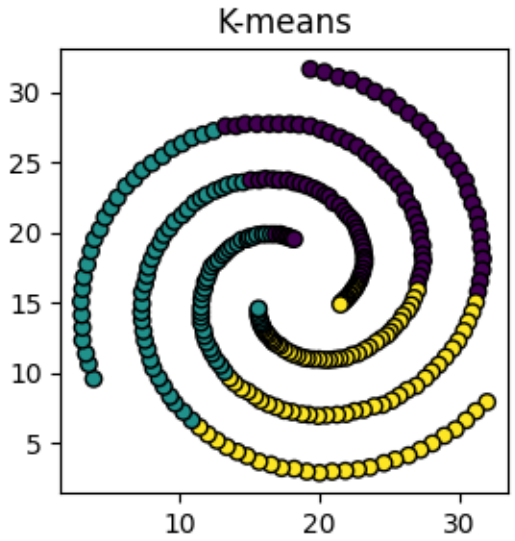


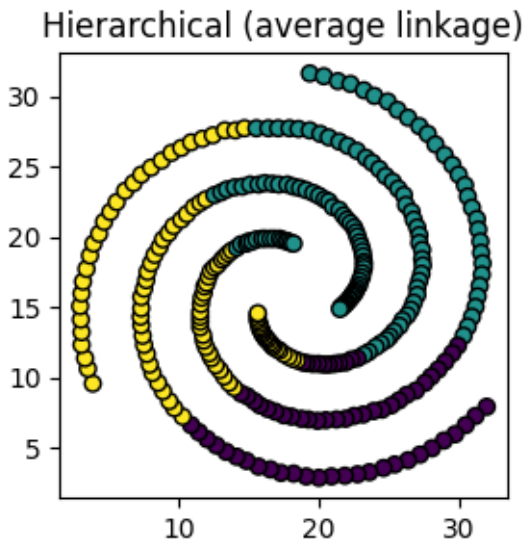


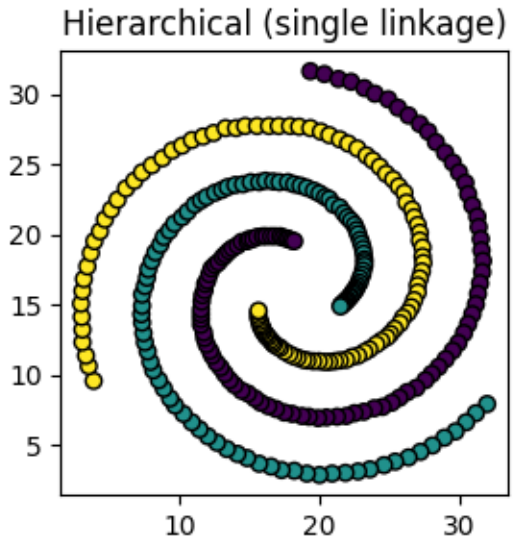
[103]:
```
# Spiral dataset
# N=312, k=3, D=2
```

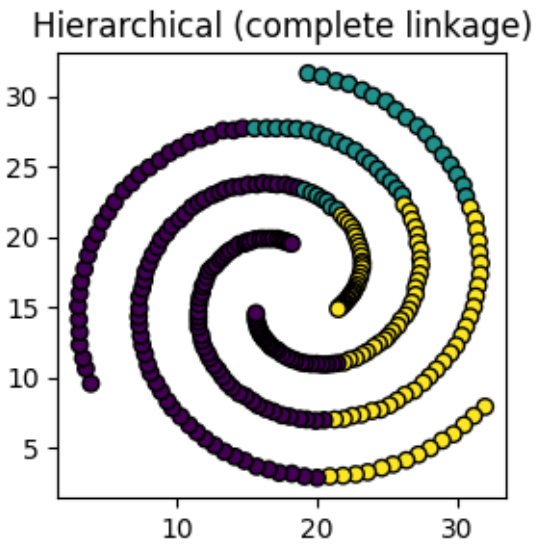
[123]:
```
plt.figure(figsize=(3, 3))
plt.scatter(SpiralDataset[:,0], SpiralDataset[:,1], c=SpiralClusters,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("K-means")
plt.show()
```

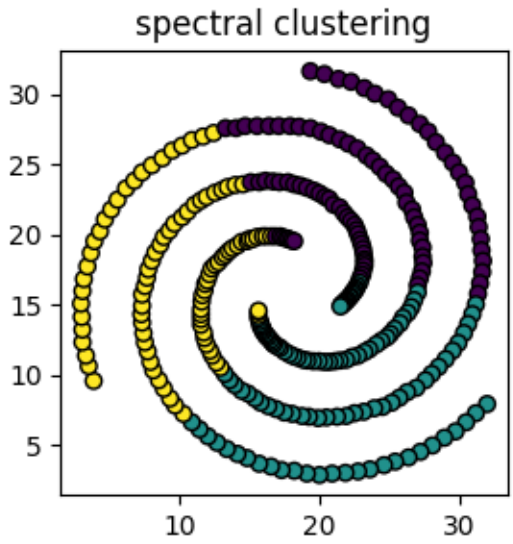
```
plt.figure(figsize=(3, 3))
plt.scatter(SpiralDataset[:,0], SpiralDataset[:,1], c=SpiralClusters2,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (average linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(SpiralDataset[:,0], SpiralDataset[:,1], c=SpiralClusters3,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (single linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(SpiralDataset[:,0], SpiralDataset[:,1], c=SpiralClusters4,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (complete linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(SpiralDataset[:,0], SpiralDataset[:,1], c=SpiralClusters5,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("spectral clustering")
plt.show()
```




Hierarchical (average linkage)
30
25
20
15
10
5
10
20
30


Hierarchical (single linkage)
30
25
20
15
10
5
10
20
30

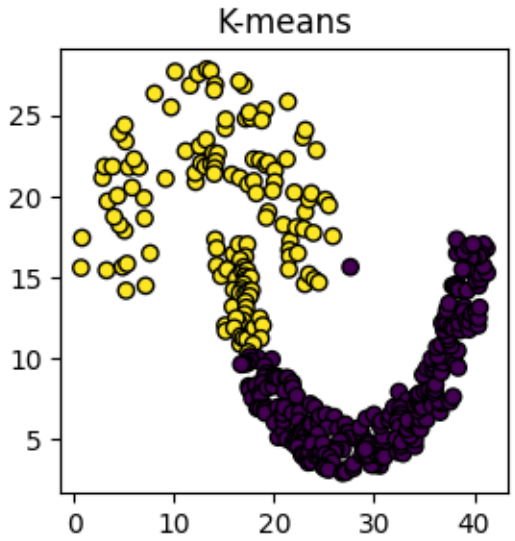


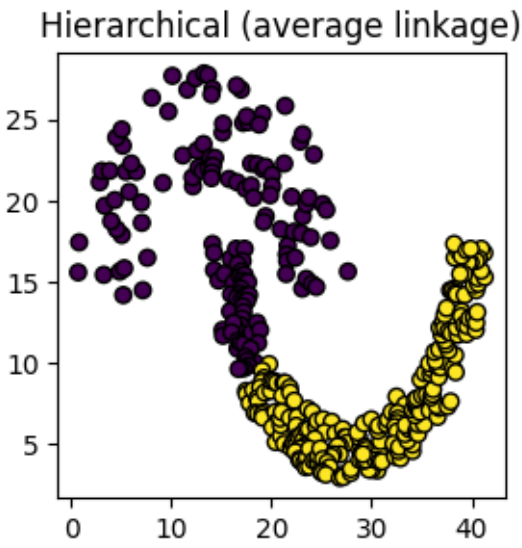


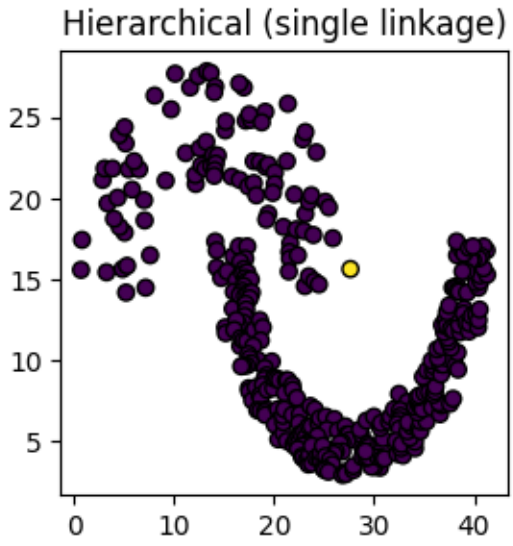
[108]:
```
# Jain dataset
# N=373, k=2, D=2
```

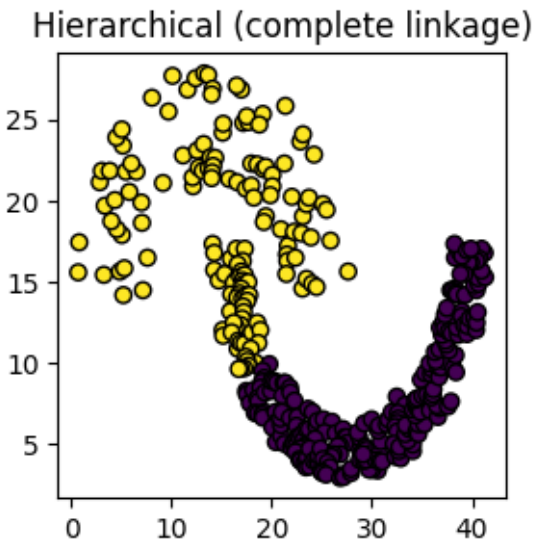
[126]:
```
plt.figure(figsize=(3, 3))
plt.scatter(JainDataset[:,0], JainDataset[:,1], c=JainClusters, cmap='viridis',␣
↪edgecolor='k')
plt.title("K-means")
plt.show()
```

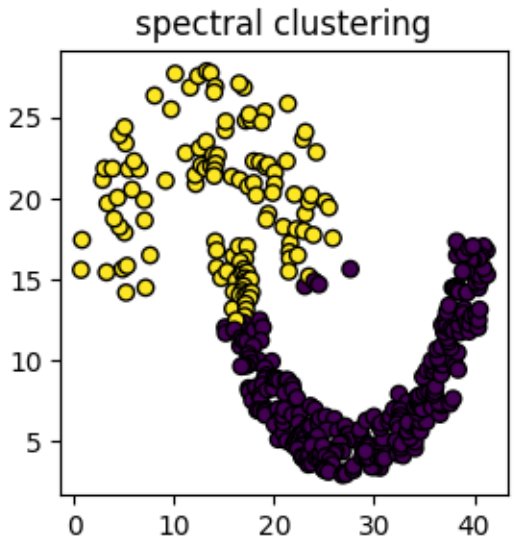
```
plt.figure(figsize=(3, 3))
plt.scatter(JainDataset[:,0], JainDataset[:,1], c=JainClusters2,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (average linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(JainDataset[:,0], JainDataset[:,1], c=JainClusters3,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (single linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(JainDataset[:,0], JainDataset[:,1], c=JainClusters4,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (complete linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(JainDataset[:,0], JainDataset[:,1], c=JainClusters5,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("spectral clustering")
plt.show()
```




Hierarchical (average linkage)
25
20
15
10
5
0
10
20
30
40


Hierarchical (single linkage)
25
20
15
10
5
0
10
20
30
40

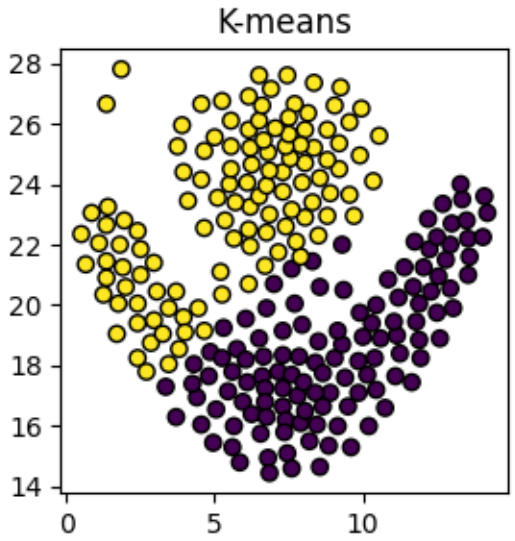


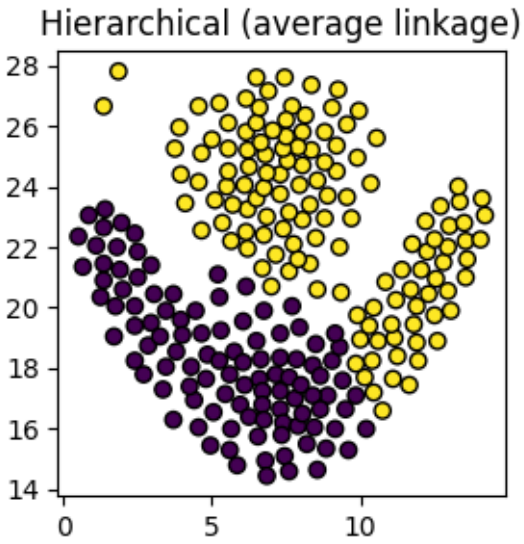


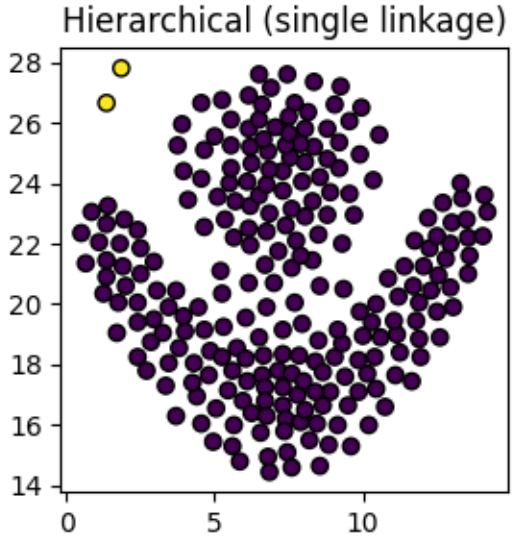
[110]:
```
# Flame dataset
# N=240, k=2, D=2
```

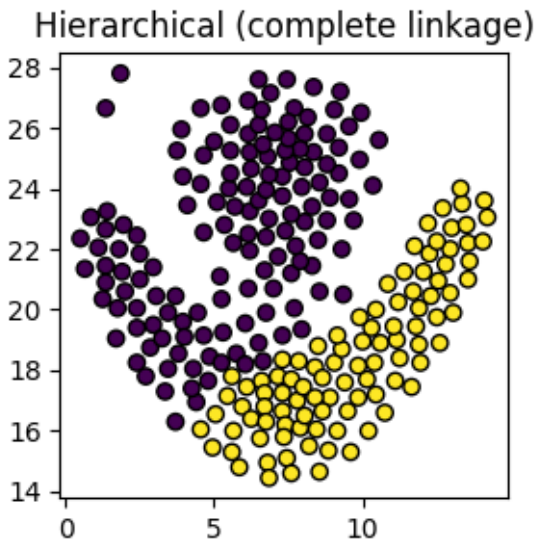
[127]:
```
plt.figure(figsize=(3, 3))
plt.scatter(FlameDataset[:,0], FlameDataset[:,1], c=FlameClusters,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("K-means")
plt.show()
```

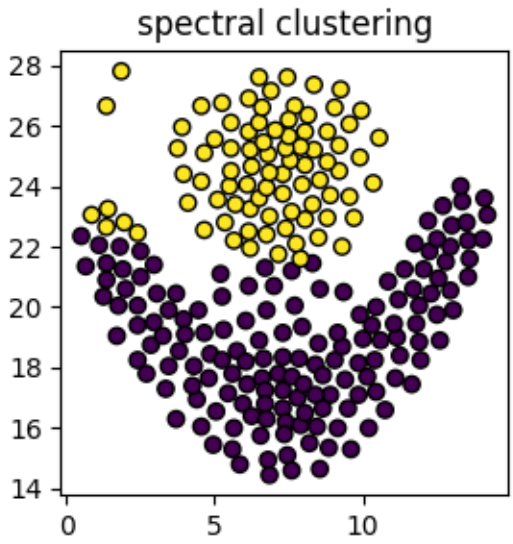
```
plt.figure(figsize=(3, 3))
plt.scatter(FlameDataset[:,0], FlameDataset[:,1], c=FlameClusters2,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (average linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(FlameDataset[:,0], FlameDataset[:,1], c=FlameClusters3,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (single linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(FlameDataset[:,0], FlameDataset[:,1], c=FlameClusters4,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("Hierarchical (complete linkage)")
plt.show()

plt.figure(figsize=(3, 3))
plt.scatter(FlameDataset[:,0], FlameDataset[:,1], c=FlameClusters5,␣
 ↪cmap='viridis', edgecolor='k')
plt.title("spectral clustering")
plt.show()
```




Hierarchical (average linkage)
28
26
24
22
20
18
16
14
0
5
10

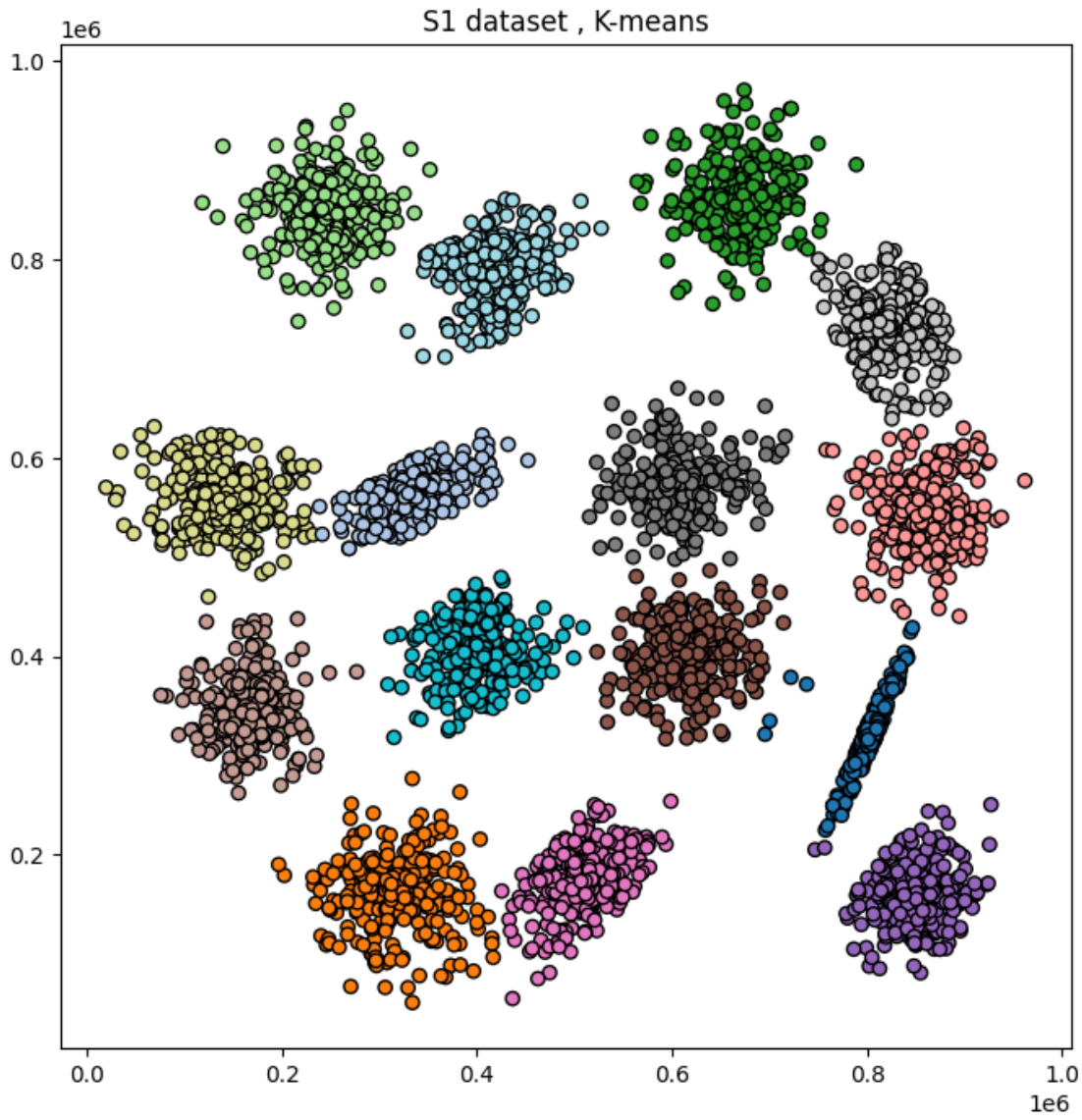

Hierarchical (single linkage)
28
26
24
22
20
18
16
14
0
5
10



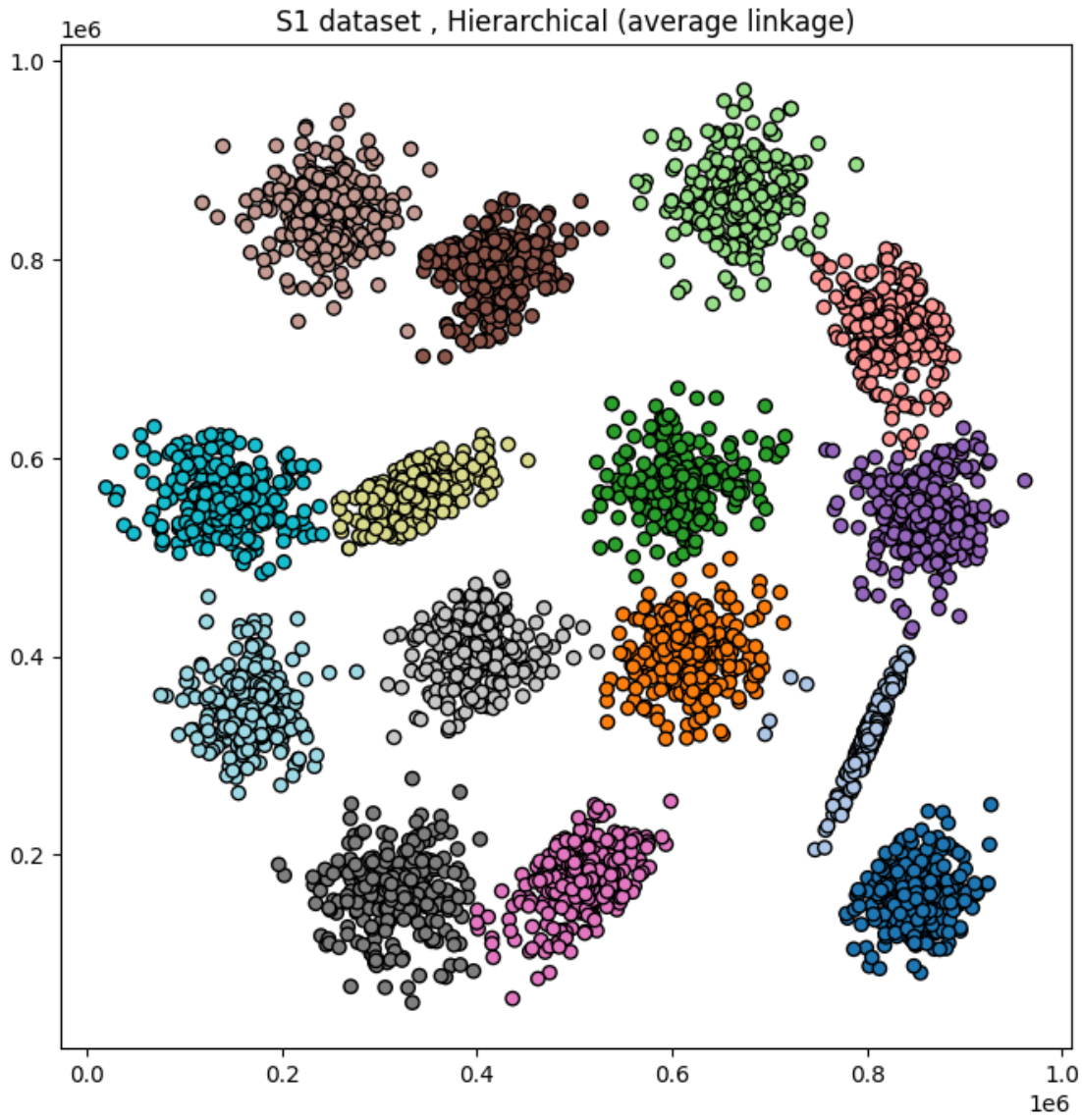


[120]:
```
# S1 dataset
# Synthetic 2-d data with N=5000 vectors and k=15 Gaussian clusters with␣
 ↪different degree of cluster overlap
```

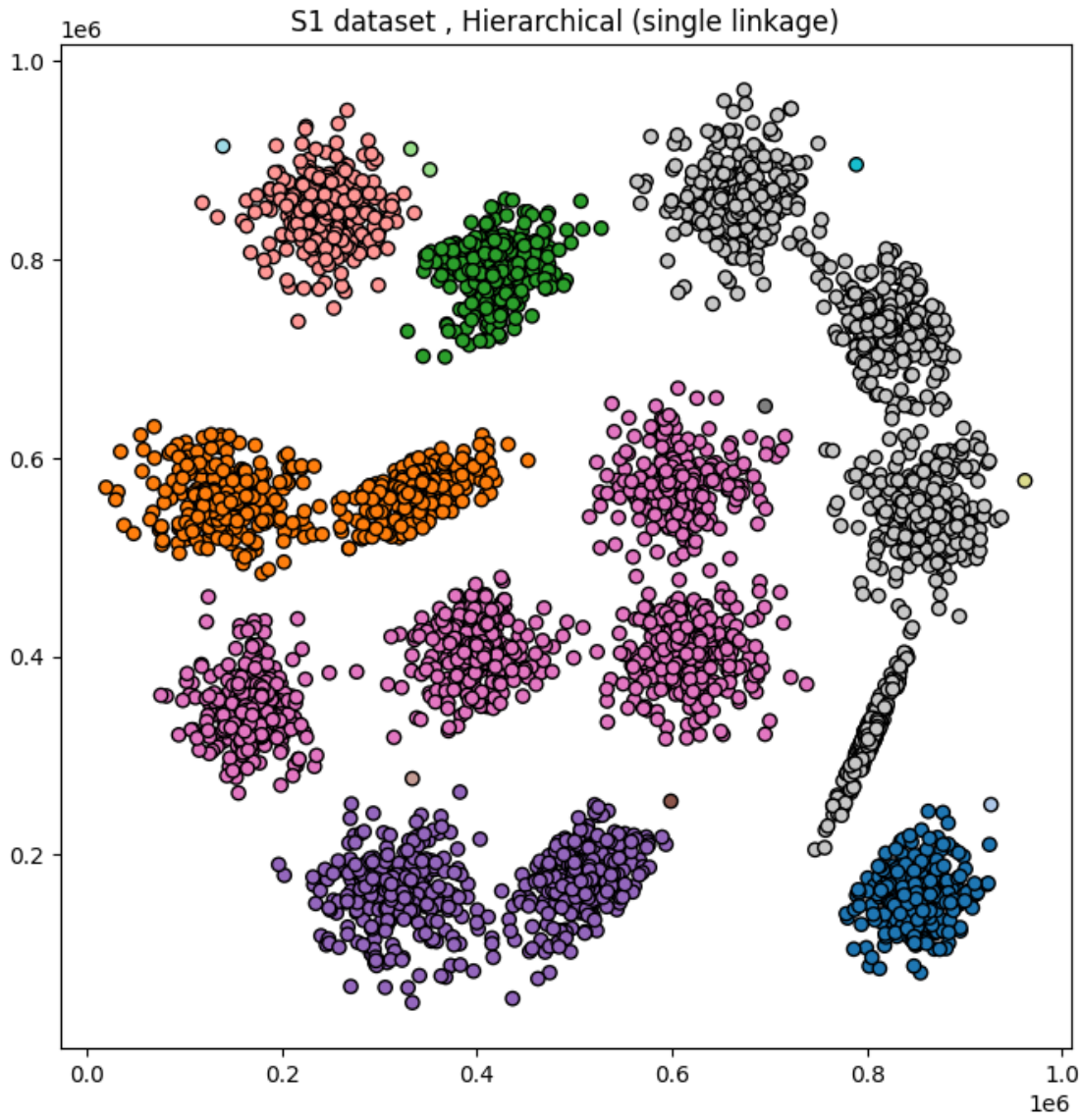
[144]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S1Dataset[:,0], S1Dataset[:,1], c=S1Clusters, cmap='tab20',␣
 ↪edgecolor='k')
plt.title("S1 dataset , K-means")
plt.show()
```



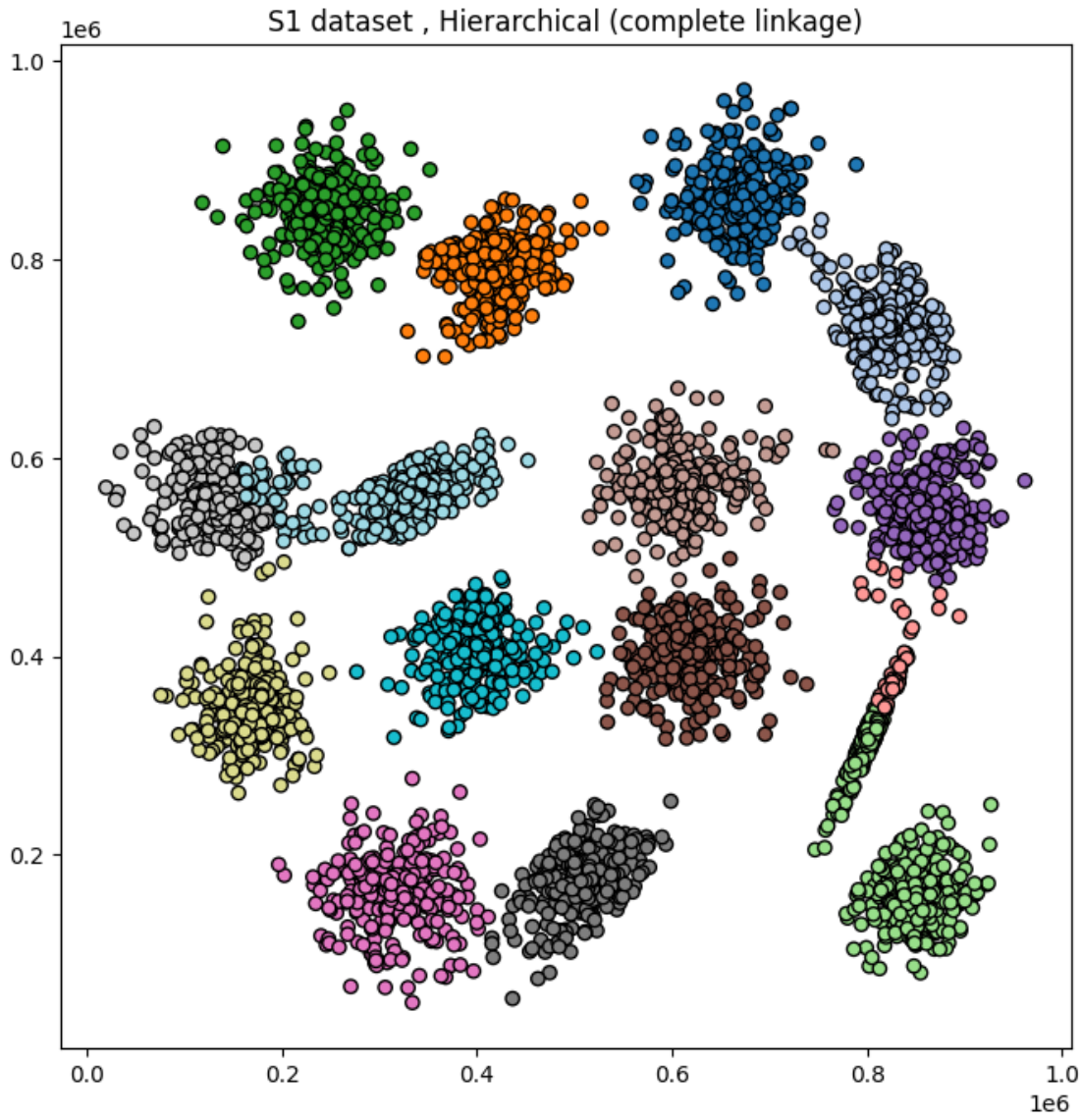
[145]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S1Dataset[:,0], S1Dataset[:,1], c=S1Clusters2, cmap='tab20',␣
↪edgecolor='k')
plt.title("S1 dataset , Hierarchical (average linkage)")
plt.show()
```



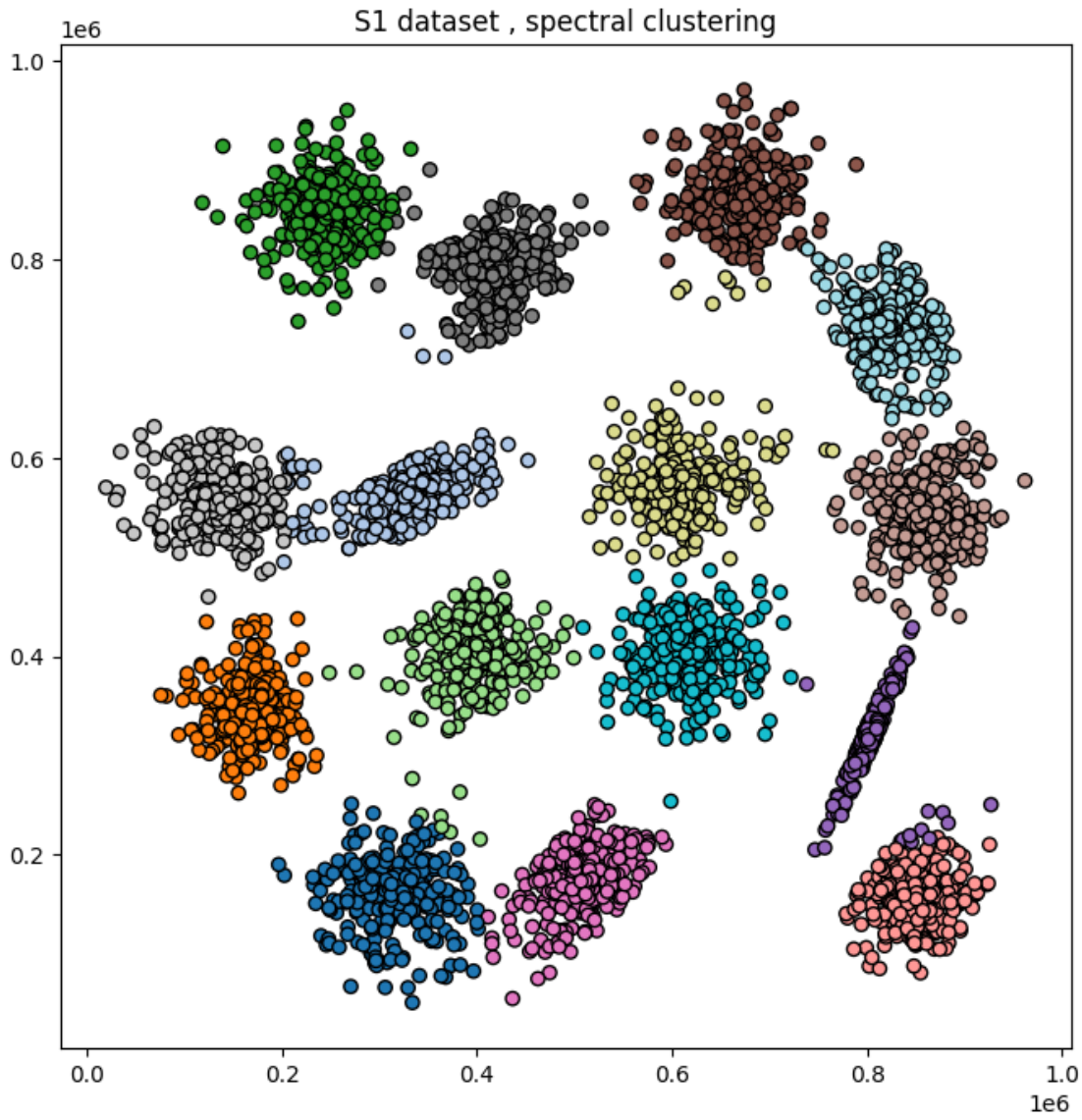
[146]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S1Dataset[:,0], S1Dataset[:,1], c=S1Clusters3, cmap='tab20',␣
↪edgecolor='k')
plt.title("S1 dataset , Hierarchical (single linkage)")
plt.show()
```



[147]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S1Dataset[:,0], S1Dataset[:,1], c=S1Clusters4, cmap='tab20',␣
↪edgecolor='k')
plt.title("S1 dataset , Hierarchical (complete linkage)")
plt.show()
```

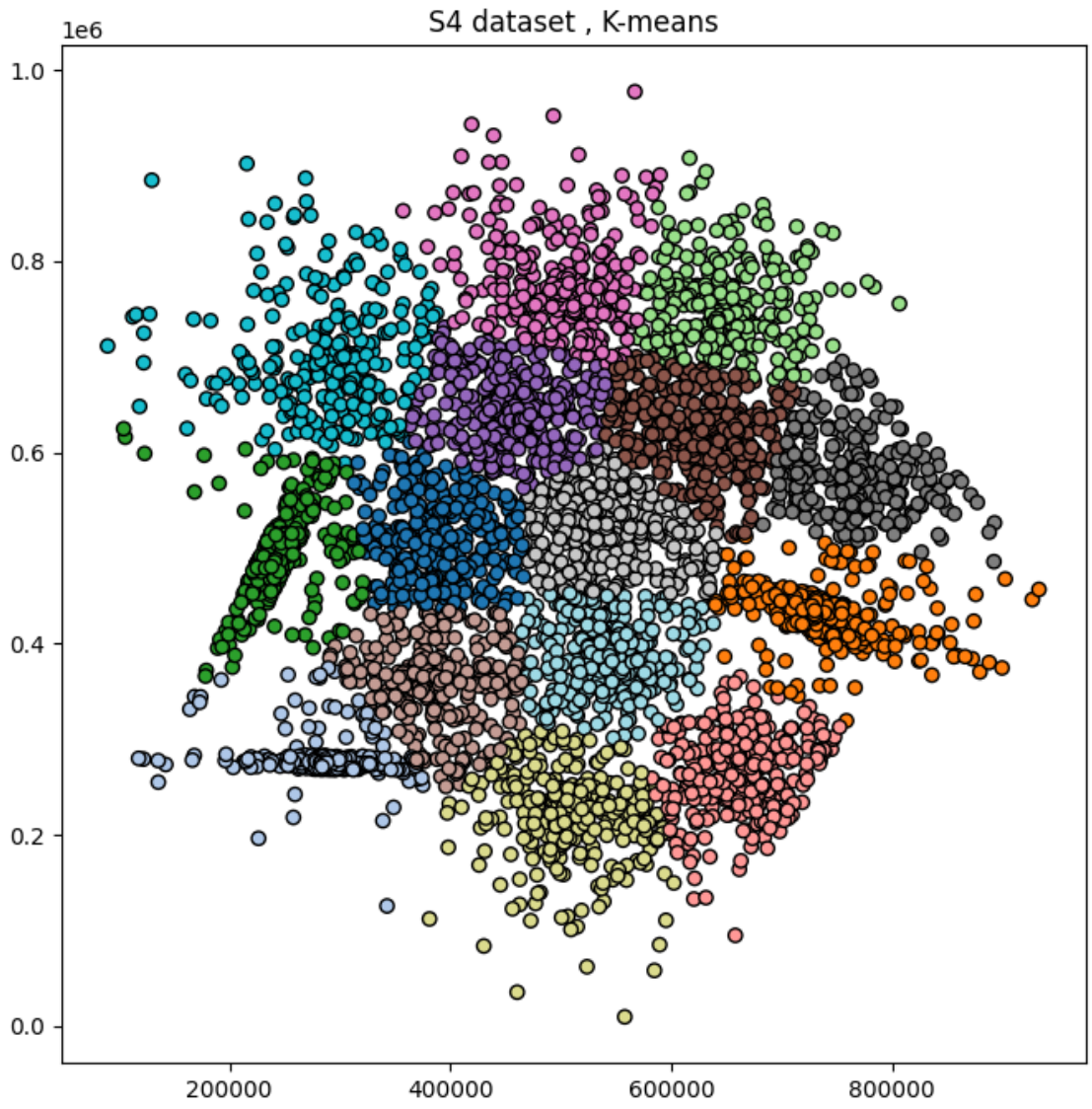


[148]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S1Dataset[:,0], S1Dataset[:,1], c=S1Clusters5, cmap='tab20',␣
↪edgecolor='k')
plt.title("S1 dataset , spectral clustering")
plt.show()
```

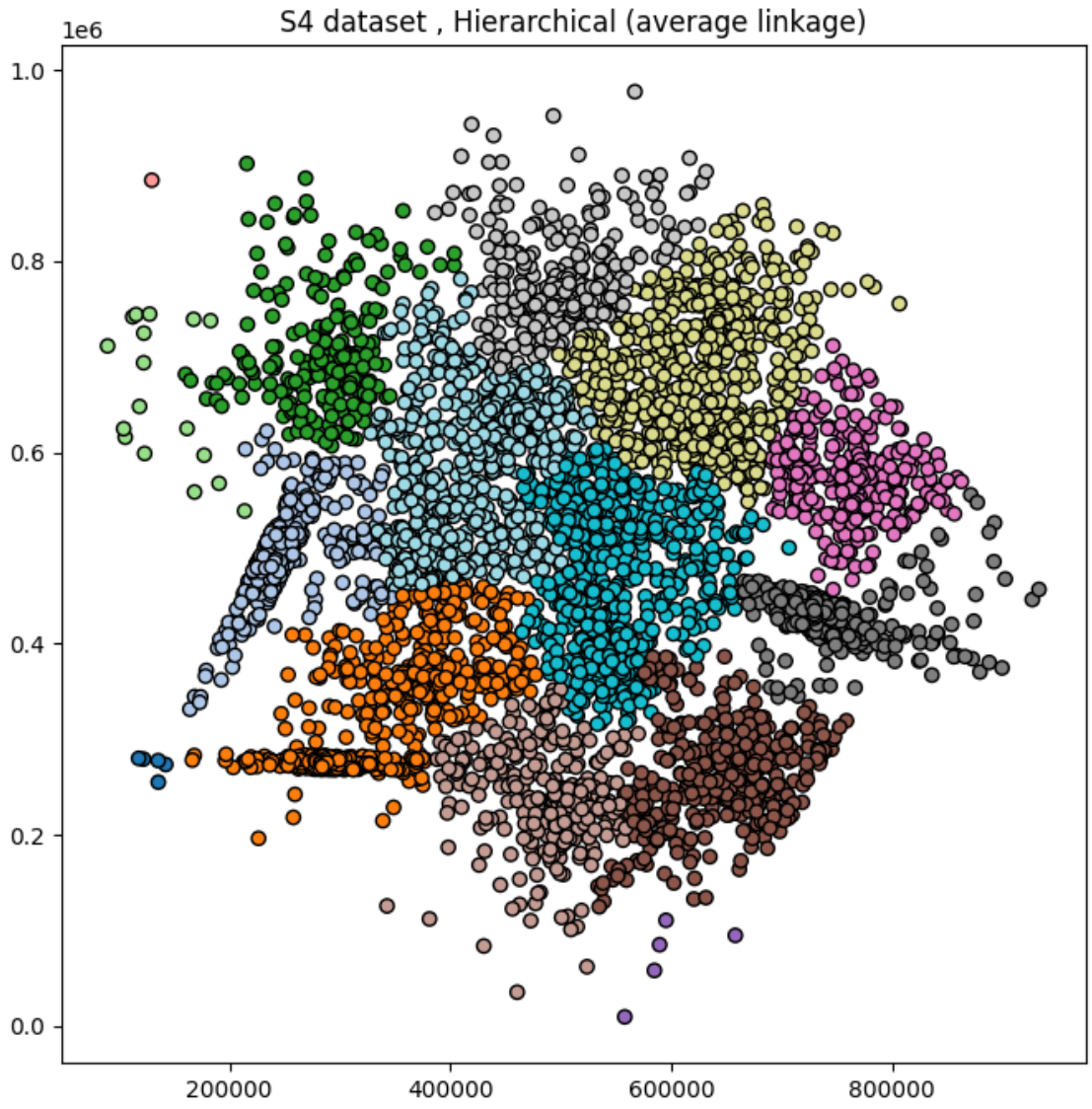


[124]:
```
# S4 dataset
# Synthetic 2-d data with N=5000 vectors and k=15 Gaussian clusters with␣
 ↪different degree of cluster overlap
```

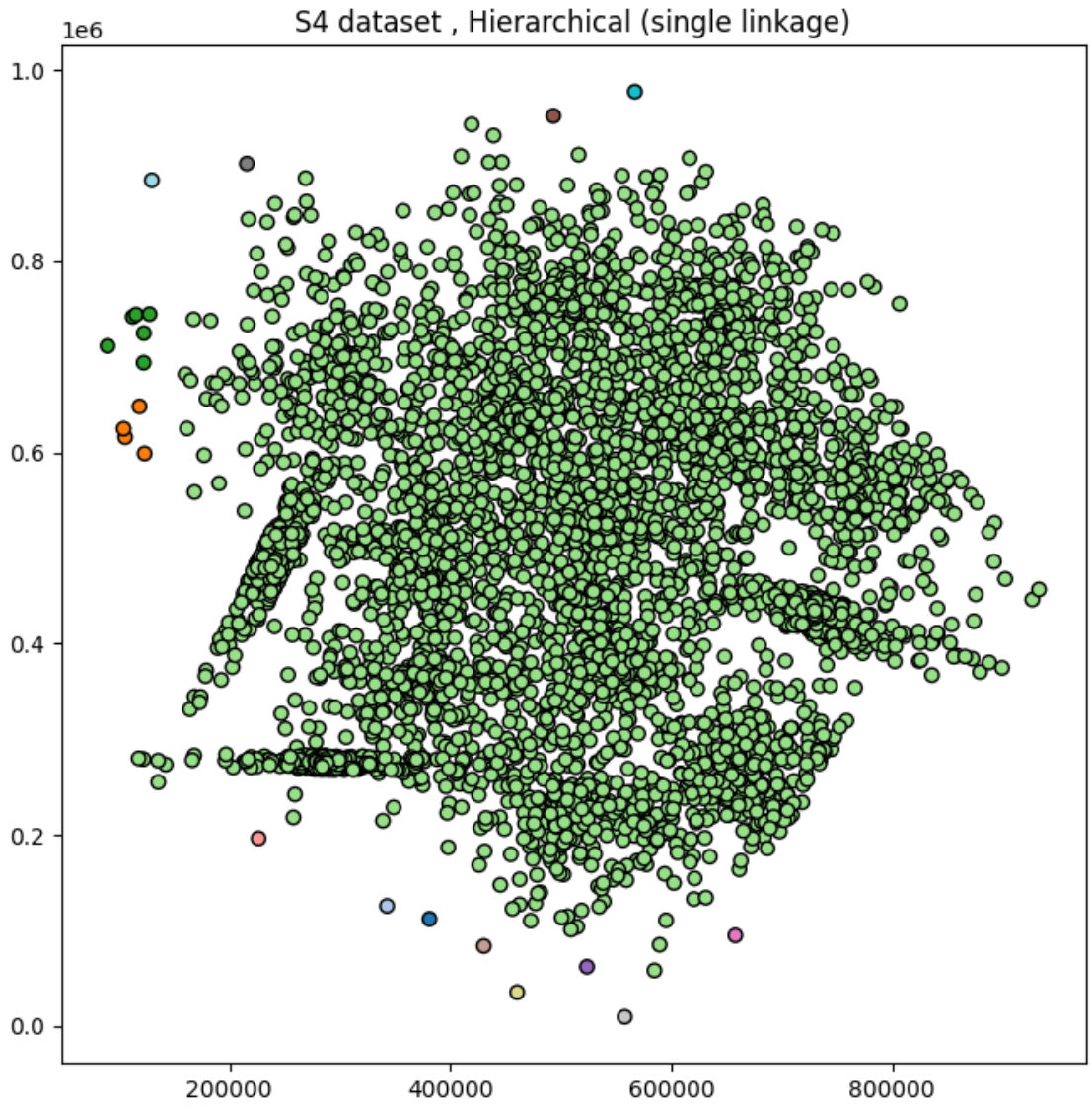
[149]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S4Dataset[:,0], S4Dataset[:,1], c=S4Clusters, cmap='tab20',␣
 ↪edgecolor='k')
plt.title("S4 dataset , K-means")
plt.show()
```



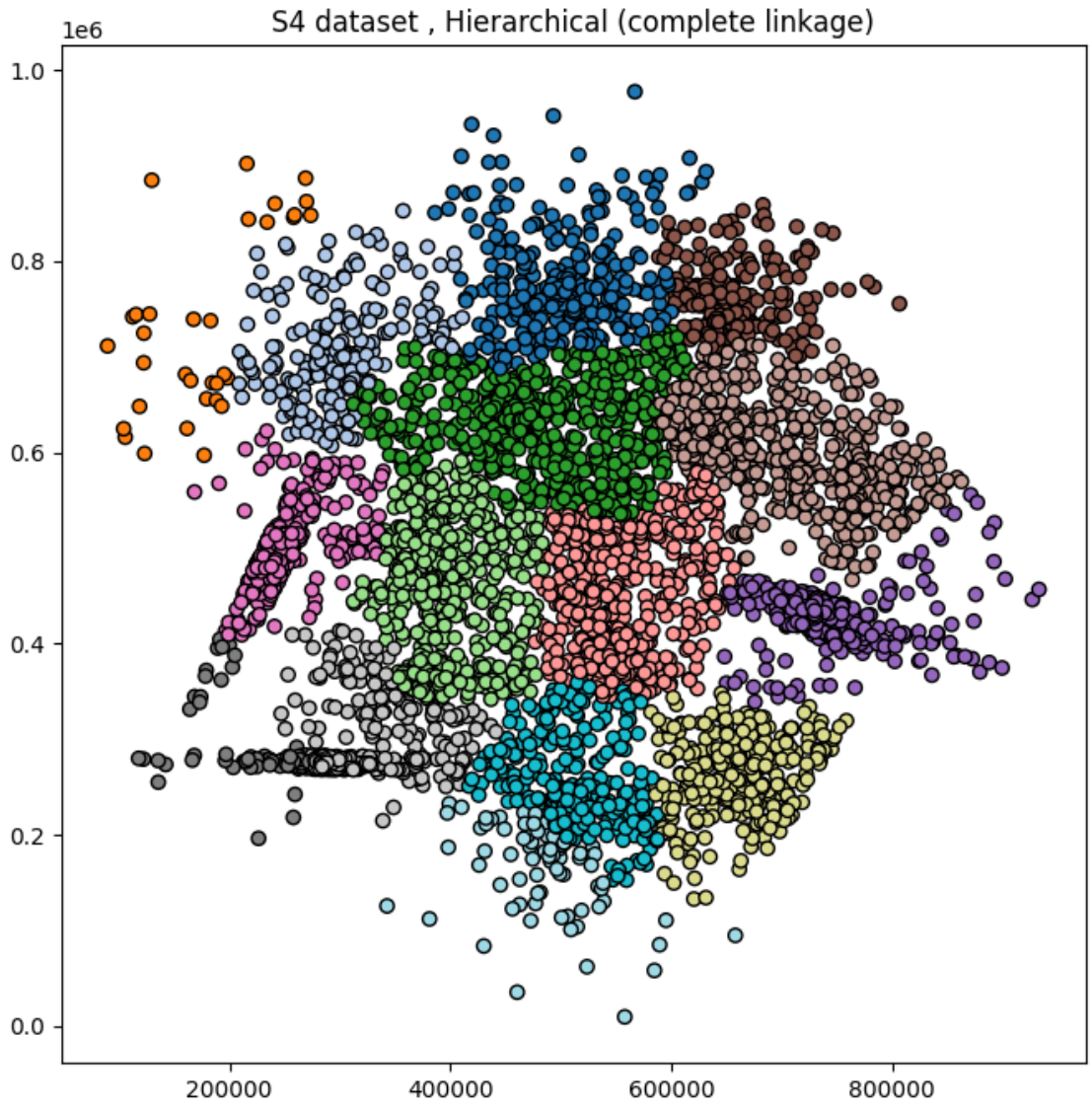
[150]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S4Dataset[:,0], S4Dataset[:,1], c=S4Clusters2, cmap='tab20',␣
↪edgecolor='k')
plt.title("S4 dataset , Hierarchical (average linkage)")
plt.show()
```



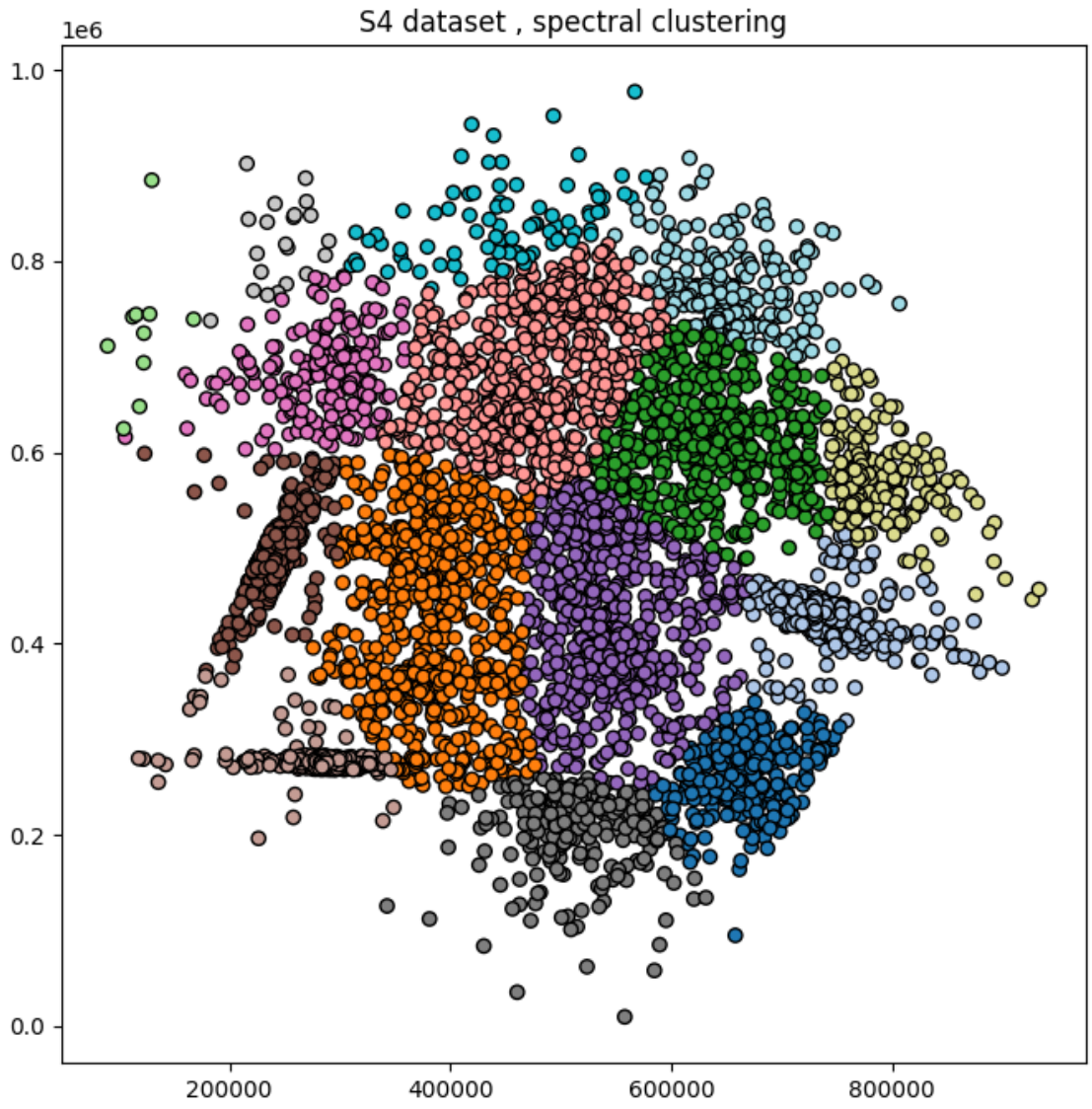
[151]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S4Dataset[:,0], S4Dataset[:,1], c=S4Clusters3, cmap='tab20',␣
↪edgecolor='k')
plt.title("S4 dataset , Hierarchical (single linkage)")
plt.show()
```



[152]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S4Dataset[:,0], S4Dataset[:,1], c=S4Clusters4, cmap='tab20',␣
↪edgecolor='k')
plt.title("S4 dataset , Hierarchical (complete linkage)")
plt.show()
```



[143]:
```
plt.figure(figsize=(8, 8))
plt.scatter(S4Dataset[:,0], S4Dataset[:,1], c=S4Clusters5, cmap='tab20',␣
↪edgecolor='k')
plt.title("S4 dataset , spectral clustering")
plt.show()
```



c. For datasets from the "Shape Set" where the true cluster is known, evaluate each clustering result using the purity index.

[168]:
```
def PurityScore(y_true, y_pred):
    contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
    return np.sum(np.amax(contingency_matrix, axis=0)) / np.
 ↪sum(contingency_matrix)

rows = ['K-means','Hierarchical(average linkage)','Hierarchical(single␣
 ↪linkage)','Hierarchical(complete linkage)','Spectral']
columns = ['Pathbased','Spiral','Jain','Flame']
PurityIndex = pd.DataFrame(index=rows , columns=columns)
```

```
for i in range(len(rows)):
    for j in range(len(columns)):
        if j==0:
            TrueLabels = PathbasedDataset[:,2]
            if i==0:
                PredictedLabels = PathbasedClusters
            if i==1:
                PredictedLabels = PathbasedClusters2
            if i==2:
                PredictedLabels = PathbasedClusters3
            if i==3:
                PredictedLabels = PathbasedClusters4
            if i==4:
                PredictedLabels = PathbasedClusters5

        if j==1:
            TrueLabels = SpiralDataset[:,2]
            if i==0:
                PredictedLabels = SpiralClusters
            if i==1:
                PredictedLabels = SpiralClusters2
            if i==2:
                PredictedLabels = SpiralClusters3
            if i==3:
                PredictedLabels = SpiralClusters4
            if i==4:
                PredictedLabels = SpiralClusters5

        if j==2:
            TrueLabels = JainDataset[:,2]
            if i==0:
                PredictedLabels = JainClusters
            if i==1:
                PredictedLabels = JainClusters2
            if i==2:
                PredictedLabels = JainClusters3
            if i==3:
                PredictedLabels = JainClusters4
            if i==4:
                PredictedLabels = JainClusters5

        if j==3:
            TrueLabels = FlameDataset[:,2]
            if i==0:
                PredictedLabels = FlameClusters
            if i==1:
```

```
                PredictedLabels = FlameClusters2
            if i==2:
                PredictedLabels = FlameClusters3
            if i==3:
                PredictedLabels = FlameClusters4
            if i==4:
                PredictedLabels = FlameClusters5

        Purity = PurityScore(TrueLabels, PredictedLabels)
        PurityIndex.iloc[i,j] = Purity

print(PurityIndex)
```

```
                            Pathbased    Spiral      Jain     Flame
K-means                          0.76  0.339744  0.873995  0.820833
Hierarchical(average linkage)   0.753333  0.416667   0.86059       0.8
Hierarchical(single linkage)    0.376667       1.0  0.742627  0.645833
Hierarchical(complete linkage)  0.693333  0.391026   0.86059  0.770833
Spectral                         0.74  0.349359  0.906166      0.95
```

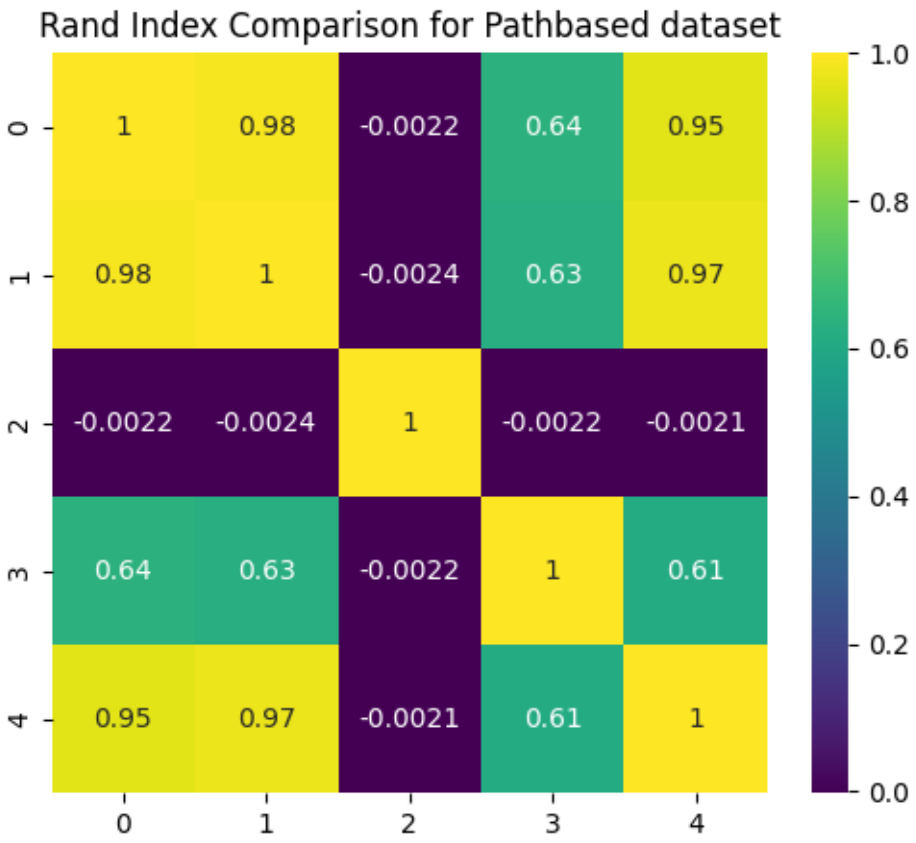
d. For each dataset, compare the results of each pair of clustering methods using the Rand index. Visualize the results in a 5x5 heatmap.

[ ]:
```
# Pathbased dataset
# N=300, k=3, D=2
```

[219]:
```
methods = ['K-means','Average','Single','Complete','Spectral']
PathbasedRandIndex = pd.DataFrame(index=methods , columns=methods)
PathbasedRandIndexNP = np.zeros((5,5))

for i in range(len(methods)):
    for j in range(len(methods)):

        if i==0:
            PredictedLabels1 = PathbasedClusters
        if i==1:
            PredictedLabels1 = PathbasedClusters2
        if i==2:
            PredictedLabels1 = PathbasedClusters3
        if i==3:
            PredictedLabels1 = PathbasedClusters4
        if i==4:
            PredictedLabels1 = PathbasedClusters5

        if j==0:
            PredictedLabels2 = PathbasedClusters
        if j==1:
            PredictedLabels2 = PathbasedClusters2
```

```
        if j==2:
            PredictedLabels2 = PathbasedClusters3
        if j==3:
            PredictedLabels2 = PathbasedClusters4
        if j==4:
            PredictedLabels2 = PathbasedClusters5

        PathbasedRand = adjusted_rand_score(PredictedLabels1,PredictedLabels2)
        PathbasedRandIndex.iloc[i,j] = PathbasedRand
        PathbasedRandIndexNP[i,j] = PathbasedRand

print(PathbasedRandIndex)

plt.figure(figsize=(6,5))
sbn.heatmap(PathbasedRandIndexNP, annot=True, cmap="viridis")
plt.title("Rand Index Comparison for Pathbased dataset")
plt.show()
```

```
           K-means   Average    Single  Complete  Spectral
K-means        1.0  0.984546 -0.002221   0.63922  0.954234
Average   0.984546       1.0 -0.002404  0.625766  0.969491
Single   -0.002221 -0.002404       1.0 -0.002208 -0.002067
Complete   0.63922  0.625766 -0.002208       1.0  0.612139
Spectral  0.954234  0.969491 -0.002067  0.612139       1.0
```

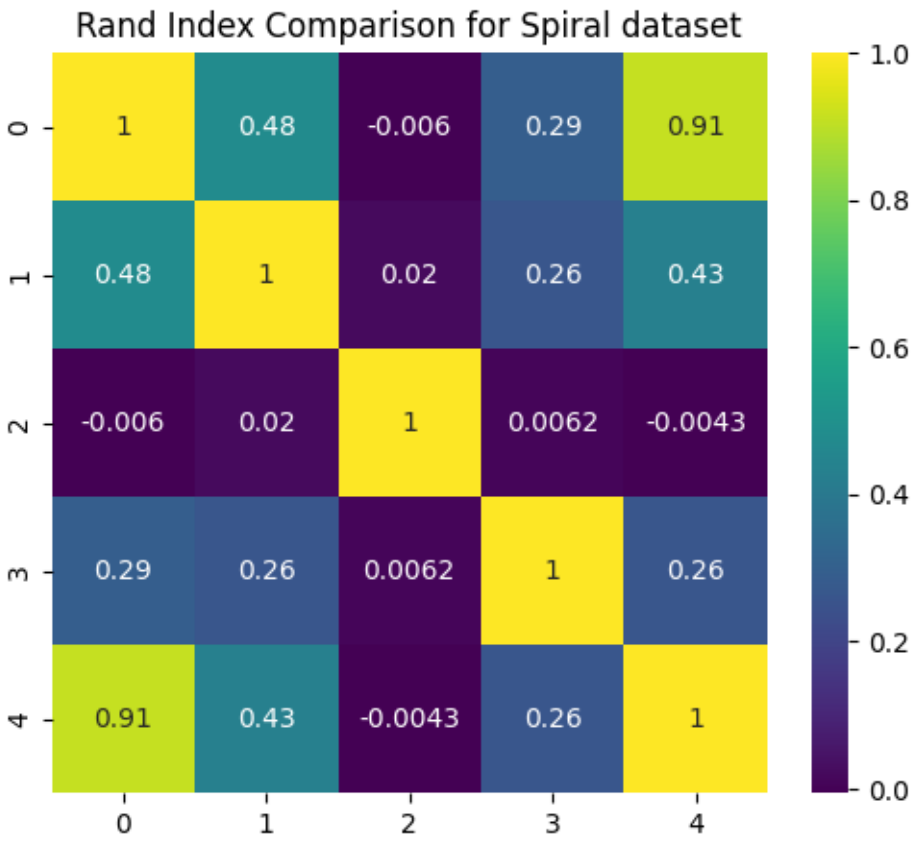


[176]:
```
# Spiral dataset
# N=312, k=3, D=2
```

[220]:
```
SpiralRandIndex = pd.DataFrame(index=methods , columns=methods)
SpiralRandIndexNP = np.zeros((5,5))

for i in range(len(methods)):
    for j in range(len(methods)):

        if i==0:
            PredictedLabels1 = SpiralClusters
        if i==1:
            PredictedLabels1 = SpiralClusters2
        if i==2:
            PredictedLabels1 = SpiralClusters3
        if i==3:
            PredictedLabels1 = SpiralClusters4
        if i==4:
            PredictedLabels1 = SpiralClusters5
```

```
        if j==0:
            PredictedLabels2 = SpiralClusters
        if j==1:
            PredictedLabels2 = SpiralClusters2
        if j==2:
            PredictedLabels2 = SpiralClusters3
        if j==3:
            PredictedLabels2 = SpiralClusters4
        if j==4:
            PredictedLabels2 = SpiralClusters5

        SpiralRand = adjusted_rand_score(PredictedLabels1,PredictedLabels2)
        SpiralRandIndex.iloc[i,j] = SpiralRand
        SpiralRandIndexNP[i,j] = SpiralRand

print(SpiralRandIndex)

plt.figure(figsize=(6,5))
sbn.heatmap(SpiralRandIndexNP, annot=True, cmap="viridis")
plt.title("Rand Index Comparison for Spiral dataset")
plt.show()
```

```
           K-means   Average    Single  Complete  Spectral
K-means        1.0  0.477081 -0.006025  0.291922  0.913216
Average   0.477081       1.0  0.020269  0.262671   0.42995
Single   -0.006025  0.020269       1.0  0.006224 -0.004251
Complete  0.291922  0.262671  0.006224       1.0   0.26179
Spectral  0.913216   0.42995 -0.004251   0.26179       1.0
```

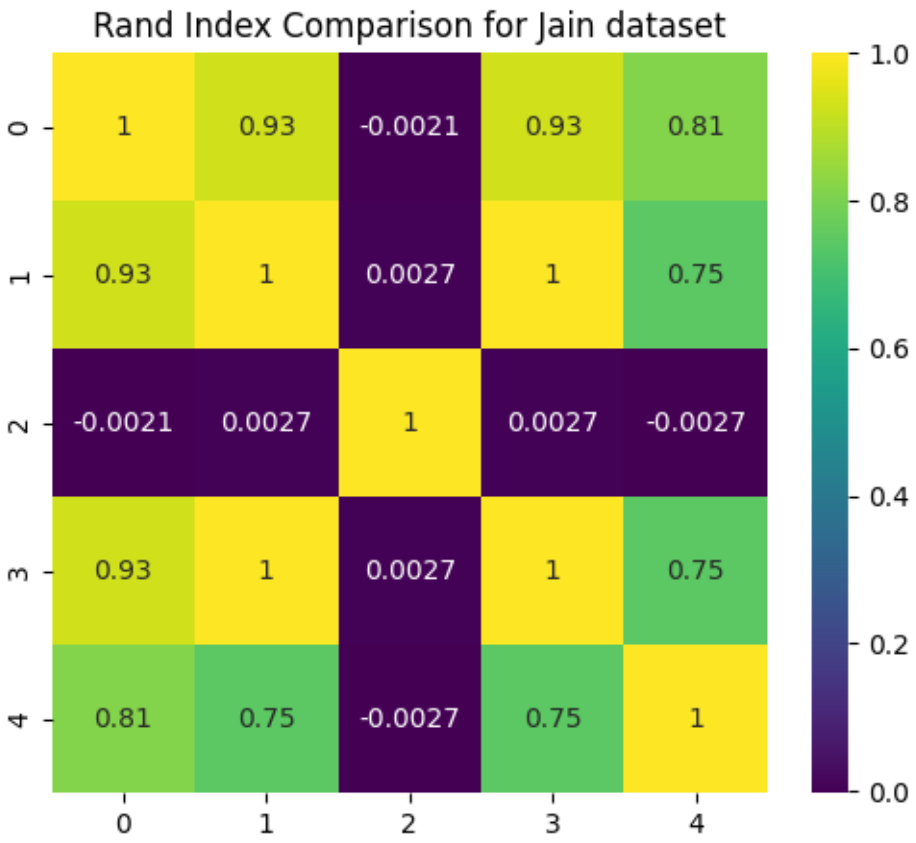


[178]:
```
# Jain dataset
# N=373, k=2, D=2
```

[221]:
```
JainRandIndex = pd.DataFrame(index=methods , columns=methods)
JainRandIndexNP = np.zeros((5,5))

for i in range(len(methods)):
    for j in range(len(methods)):

        if i==0:
            PredictedLabels1 = JainClusters
        if i==1:
            PredictedLabels1 = JainClusters2
        if i==2:
            PredictedLabels1 = JainClusters3
        if i==3:
            PredictedLabels1 = JainClusters4
        if i==4:
            PredictedLabels1 = JainClusters5
```

```
        if j==0:
            PredictedLabels2 = JainClusters
        if j==1:
            PredictedLabels2 = JainClusters2
        if j==2:
            PredictedLabels2 = JainClusters3
        if j==3:
            PredictedLabels2 = JainClusters4
        if j==4:
            PredictedLabels2 = JainClusters5

        JainRand = adjusted_rand_score(PredictedLabels1,PredictedLabels2)
        JainRandIndex.iloc[i,j] = JainRand
        JainRandIndexNP[i,j] = JainRand

print(JainRandIndex)

plt.figure(figsize=(6,5))
sbn.heatmap(JainRandIndexNP, annot=True, cmap="viridis")
plt.title("Rand Index Comparison for Jain dataset")
plt.show()
```

```
            K-means   Average    Single  Complete  Spectral
K-means         1.0  0.925991 -0.002065  0.925991  0.814684
Average    0.925991       1.0  0.002698       1.0  0.748152
Single    -0.002065  0.002698       1.0  0.002698 -0.002688
Complete   0.925991       1.0  0.002698       1.0  0.748152
Spectral   0.814684  0.748152 -0.002688  0.748152       1.0
```

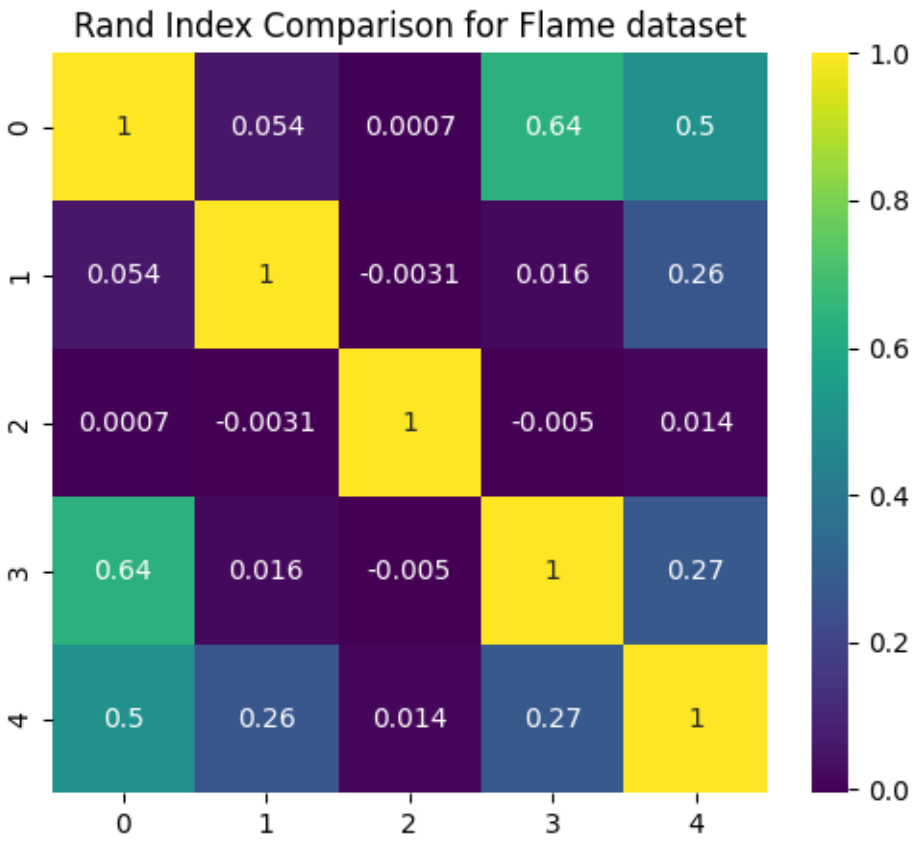


[180]:
```
# Flame dataset
# N=240, k=2, D=2
```

[222]:
```
FlameRandIndex = pd.DataFrame(index=methods , columns=methods)
FlameRandIndexNP = np.zeros((5,5))

for i in range(len(methods)):
    for j in range(len(methods)):

        if i==0:
            PredictedLabels1 = FlameClusters
        if i==1:
            PredictedLabels1 = FlameClusters2
        if i==2:
            PredictedLabels1 = FlameClusters3
        if i==3:
            PredictedLabels1 = FlameClusters4
        if i==4:
            PredictedLabels1 = FlameClusters5
```

```
        if j==0:
            PredictedLabels2 = FlameClusters
        if j==1:
            PredictedLabels2 = FlameClusters2
        if j==2:
            PredictedLabels2 = FlameClusters3
        if j==3:
            PredictedLabels2 = FlameClusters4
        if j==4:
            PredictedLabels2 = FlameClusters5

        FlameRand = adjusted_rand_score(PredictedLabels1,PredictedLabels2)
        FlameRandIndex.iloc[i,j] = FlameRand
        FlameRandIndexNP[i,j] = FlameRand

print(FlameRandIndex)

plt.figure(figsize=(6,5))
sbn.heatmap(FlameRandIndexNP, annot=True, cmap="viridis")
plt.title("Rand Index Comparison for Flame dataset")
plt.show()
```

```
           K-means   Average    Single  Complete  Spectral
K-means        1.0  0.054491  0.000705  0.638535   0.49981
Average   0.054491       1.0 -0.003118  0.015748  0.263422
Single    0.000705 -0.003118       1.0  -0.00502  0.013824
Complete  0.638535  0.015748  -0.00502       1.0  0.270843
Spectral   0.49981  0.263422  0.013824  0.270843       1.0
```

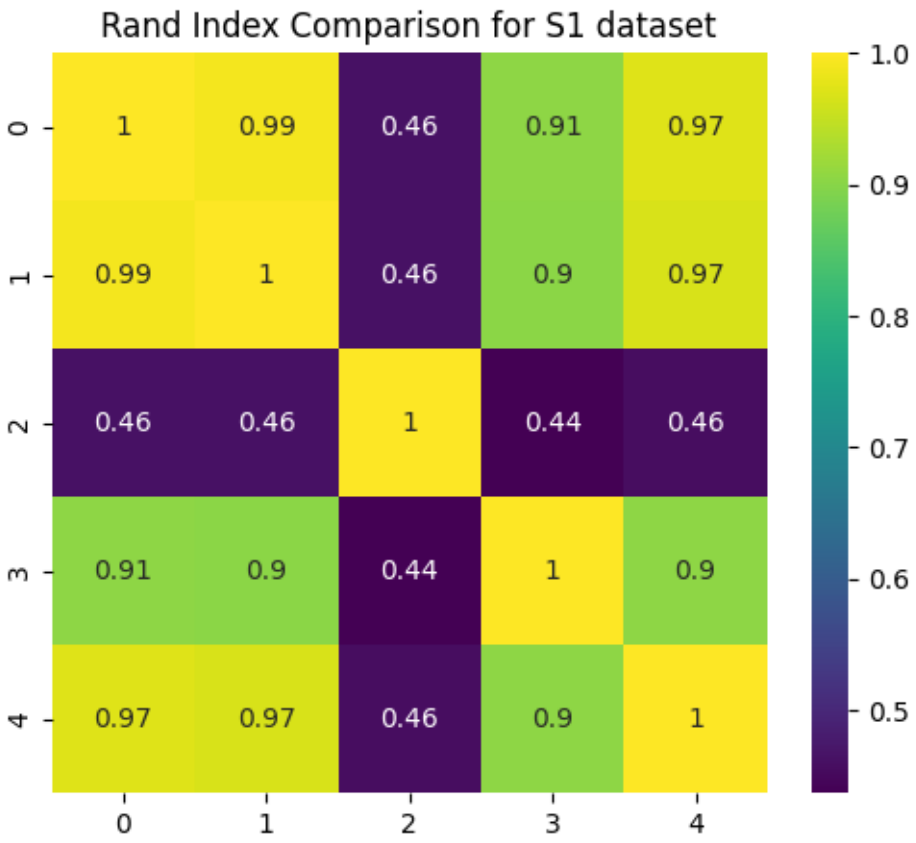


[183]:
```
# S1 dataset
# Synthetic 2-d data with N=5000 vectors and k=15 Gaussian clusters with␣
 ↪different degree of cluster overlap
```

[223]:
```
S1RandIndex = pd.DataFrame(index=methods , columns=methods)
S1RandIndexNP = np.zeros((5,5))

for i in range(len(methods)):
    for j in range(len(methods)):

        if i==0:
            PredictedLabels1 = S1Clusters
        if i==1:
            PredictedLabels1 = S1Clusters2
        if i==2:
            PredictedLabels1 = S1Clusters3
        if i==3:
            PredictedLabels1 = S1Clusters4
        if i==4:
            PredictedLabels1 = S1Clusters5
```

```
        if j==0:
            PredictedLabels2 = S1Clusters
        if j==1:
            PredictedLabels2 = S1Clusters2
        if j==2:
            PredictedLabels2 = S1Clusters3
        if j==3:
            PredictedLabels2 = S1Clusters4
        if j==4:
            PredictedLabels2 = S1Clusters5

        S1Rand = adjusted_rand_score(PredictedLabels1,PredictedLabels2)
        S1RandIndex.iloc[i,j] = S1Rand
        S1RandIndexNP[i,j] = S1Rand

print(S1RandIndex)

plt.figure(figsize=(6,5))
sbn.heatmap(S1RandIndexNP, annot=True, cmap="viridis")
plt.title("Rand Index Comparison for S1 dataset")
plt.show()
```

```
           K-means   Average    Single  Complete  Spectral
K-means        1.0  0.989727  0.462011  0.905319  0.971393
Average   0.989727       1.0  0.462504  0.902849  0.966765
Single    0.462011  0.462504       1.0  0.436653  0.455791
Complete  0.905319  0.902849  0.436653       1.0  0.903843
Spectral  0.971393  0.966765  0.455791  0.903843       1.0
```

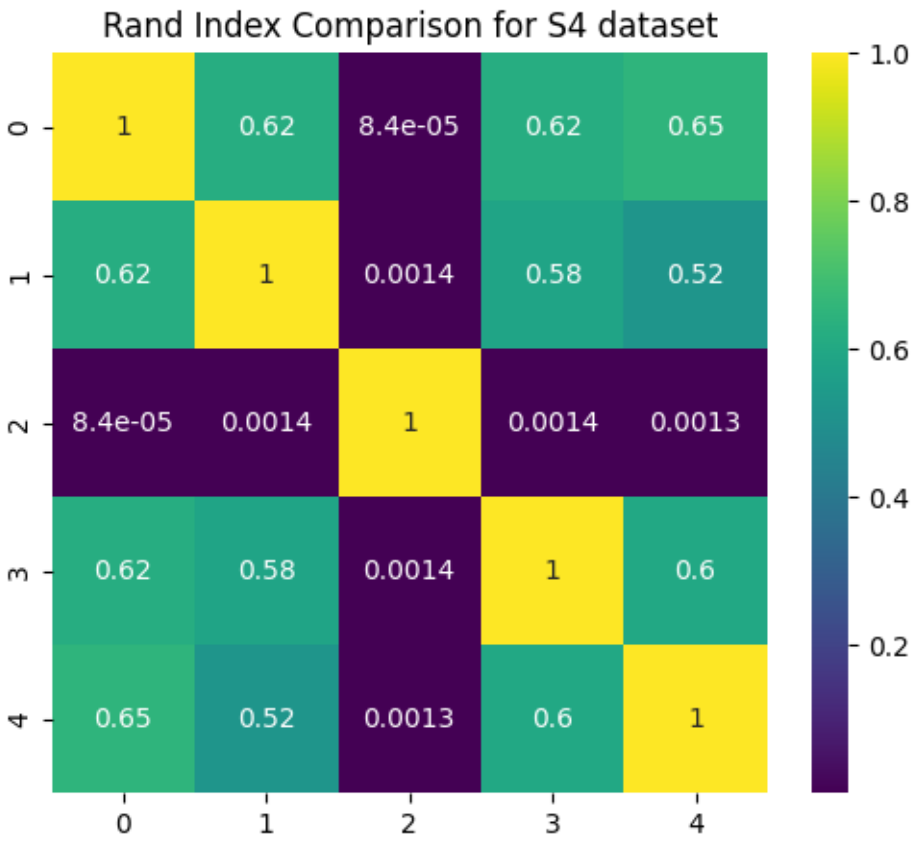


[185]:
```
# S4 dataset
# Synthetic 2-d data with N=5000 vectors and k=15 Gaussian clusters with␣
 ↪different degree of cluster overlap
```

[218]:
```
S4RandIndex = pd.DataFrame(index=methods , columns=methods)
S4RandIndexNP = np.zeros((5,5))

for i in range(len(methods)):
    for j in range(len(methods)):

        if i==0:
            PredictedLabels1 = S4Clusters
        if i==1:
            PredictedLabels1 = S4Clusters2
        if i==2:
            PredictedLabels1 = S4Clusters3
        if i==3:
            PredictedLabels1 = S4Clusters4
        if i==4:
            PredictedLabels1 = S4Clusters5
```

```
        if j==0:
            PredictedLabels2 = S4Clusters
        if j==1:
            PredictedLabels2 = S4Clusters2
        if j==2:
            PredictedLabels2 = S4Clusters3
        if j==3:
            PredictedLabels2 = S4Clusters4
        if j==4:
            PredictedLabels2 = S4Clusters5

        S4Rand = adjusted_rand_score(PredictedLabels1,PredictedLabels2)
        S4RandIndex.iloc[i,j] = S4Rand
        S4RandIndexNP[i,j] = S4Rand

print(S4RandIndex)

plt.figure(figsize=(6,5))
sbn.heatmap(S4RandIndexNP, annot=True, cmap="viridis")
plt.title("Rand Index Comparison for S4 dataset")
plt.show()
```

```
           K-means   Average    Single  Complete  Spectral
K-means        1.0  0.624955  0.000084  0.621982  0.649843
Average   0.624955       1.0  0.001441  0.584414  0.523268
Single    0.000084  0.001441       1.0  0.001359  0.001327
Complete  0.621982  0.584414  0.001359       1.0  0.597986
Spectral  0.649843  0.523268  0.001327  0.597986       1.0
```


Rand Index Comparison for S4 dataset
1
0.62
8.4e-05
0.62
0.65
0.62
1
0.0014
0.58
0.52
8.4e-05
0.0014
1
0.0014
0.0013
0.62
0.58
0.0014
1
0.6
0.65
0.52
0.0013
0.6
1
0
1
2
3
4
1.0
0.8
0.6
0.4
0.2

[ ]:

Q2

May 10, 2024

0.1 Mahdi Anvari 610700002 Homework 2 of Machine Learning Question 2

[100]:
```
# importing libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

Dataset: • Load MNIST dataset (could be accessed using from keras.datasets in python) • Separate them by label into 10 smaller sets

[116]:
```
mnist_path = 'c:/Users/M/Downloads/mnist.npz'
with np.load(mnist_path, allow_pickle=True) as f:
    X_train, Y_train = f['x_train'], f['y_train']
    X_test, Y_test = f['x_test'], f['y_test']

X_all = np.concatenate((X_train, X_test), axis=0)
Y_all = np.concatenate((Y_train, Y_test), axis=0)
print((X_all.shape))
print((Y_all.shape))

Sets = [[] for _ in range(10)]
Sets2 = [[] for _ in range(10)]
for i in range(len(X_all)):
    label = Y_all[i]
    Sets[label].append(X_all[i])
    Sets2[label].append(X_all[i])

for i in range(10):
    print(f"Length of label {i} is: ",len(Sets[i]))
```

```
(70000, 28, 28)
(70000,)
Length of label 0 is:  6903
Length of label 1 is:  7877
Length of label 2 is:  6990
Length of label 3 is:  7141
```

```
Length of label 4 is:  6824
Length of label 5 is:  6313
Length of label 6 is:  6876
Length of label 7 is:  7293
Length of label 8 is:  6825
Length of label 9 is:  6958
```

a. Flatten the pictures and apply PCA

[117]:
```
for i in range(10):
    Sets[i] = np.array(Sets[i])
    Sets2[i] = np.array(Sets[i])
print(Sets[0].shape)
for i in range(10):
    Sets[i] = (Sets[i]).reshape(Sets[i].shape[0],-1)
print(Sets[0].shape)
```

```
(6903, 28, 28)
(6903, 784)
```

[103]:
```
NormalizedSets = [[] for _ in range(10)]
for i in range(10):
    scaler = StandardScaler()
    NormalizedSets[i] = scaler.fit_transform(Sets[i])
print(NormalizedSets[0].shape)
```

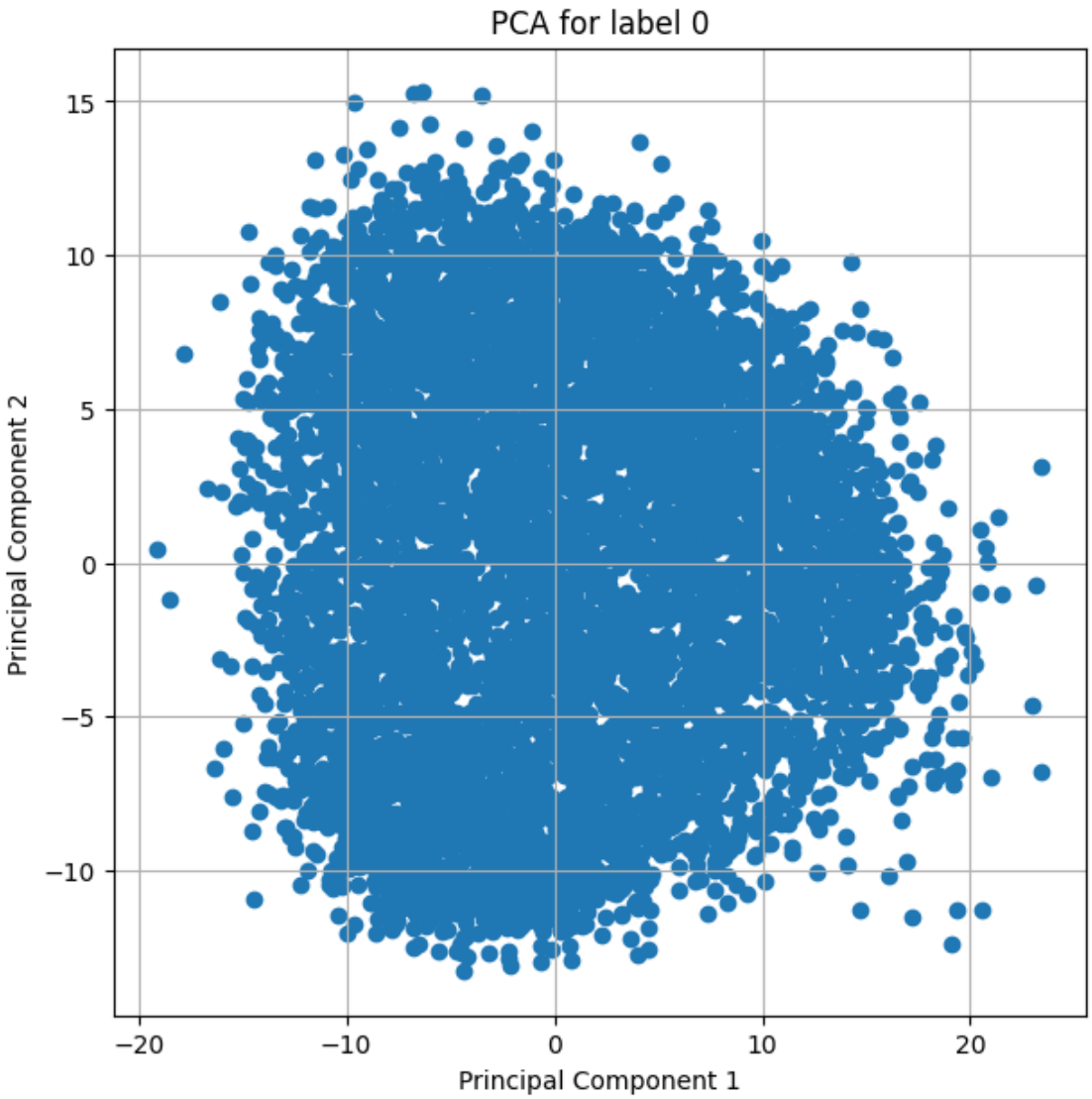
```
(6903, 784)
```

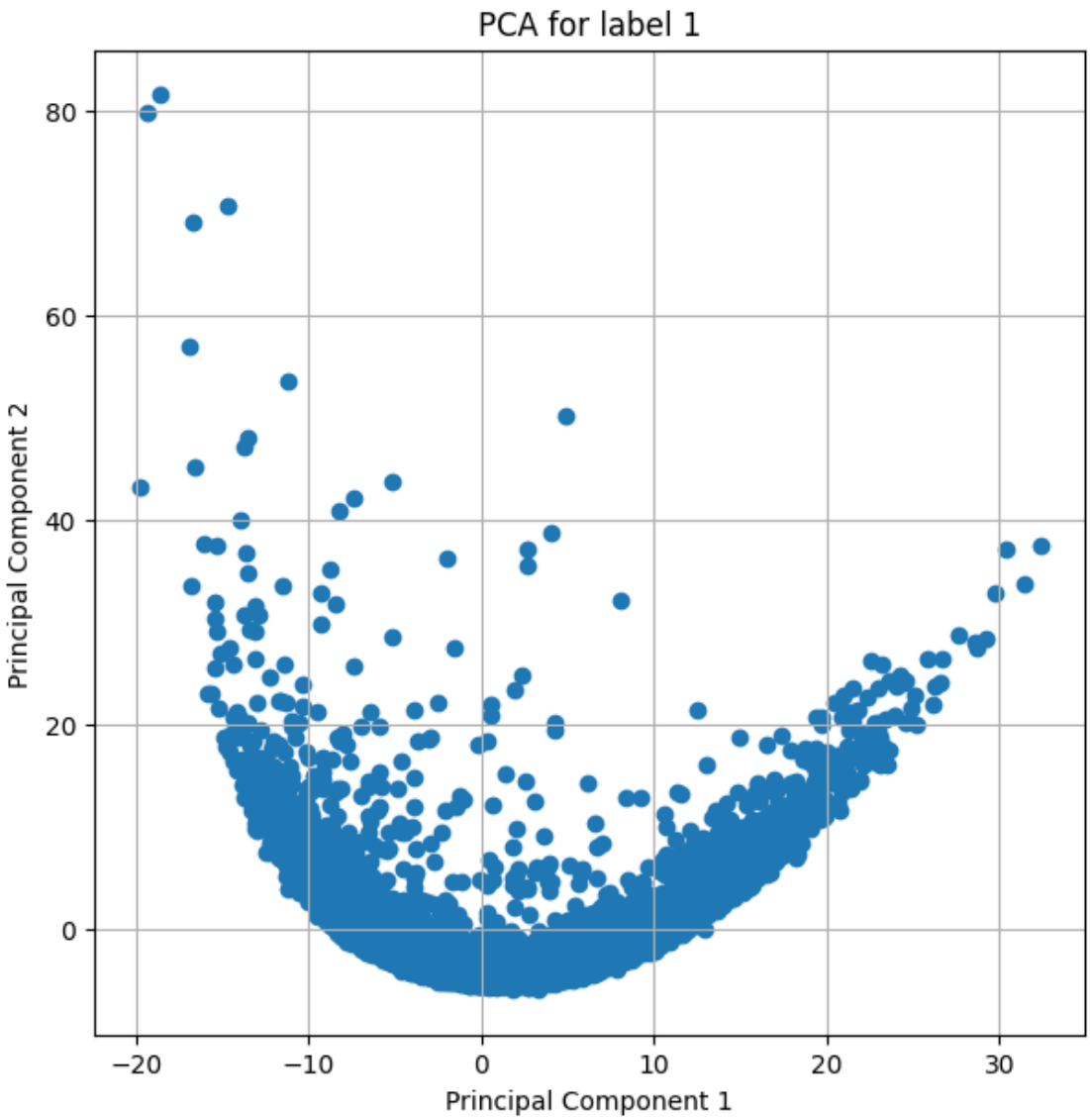
[104]:
```
PCASets = [[] for _ in range(10)]
for i in range(10):
    NormalizedSets[i] = scaler.fit_transform(Sets[i])
    pca = PCA(n_components=50)
    pca.fit(NormalizedSets[i])
    PCASets[i] = pca.transform(NormalizedSets[i])
print(PCASets[0].shape)
```

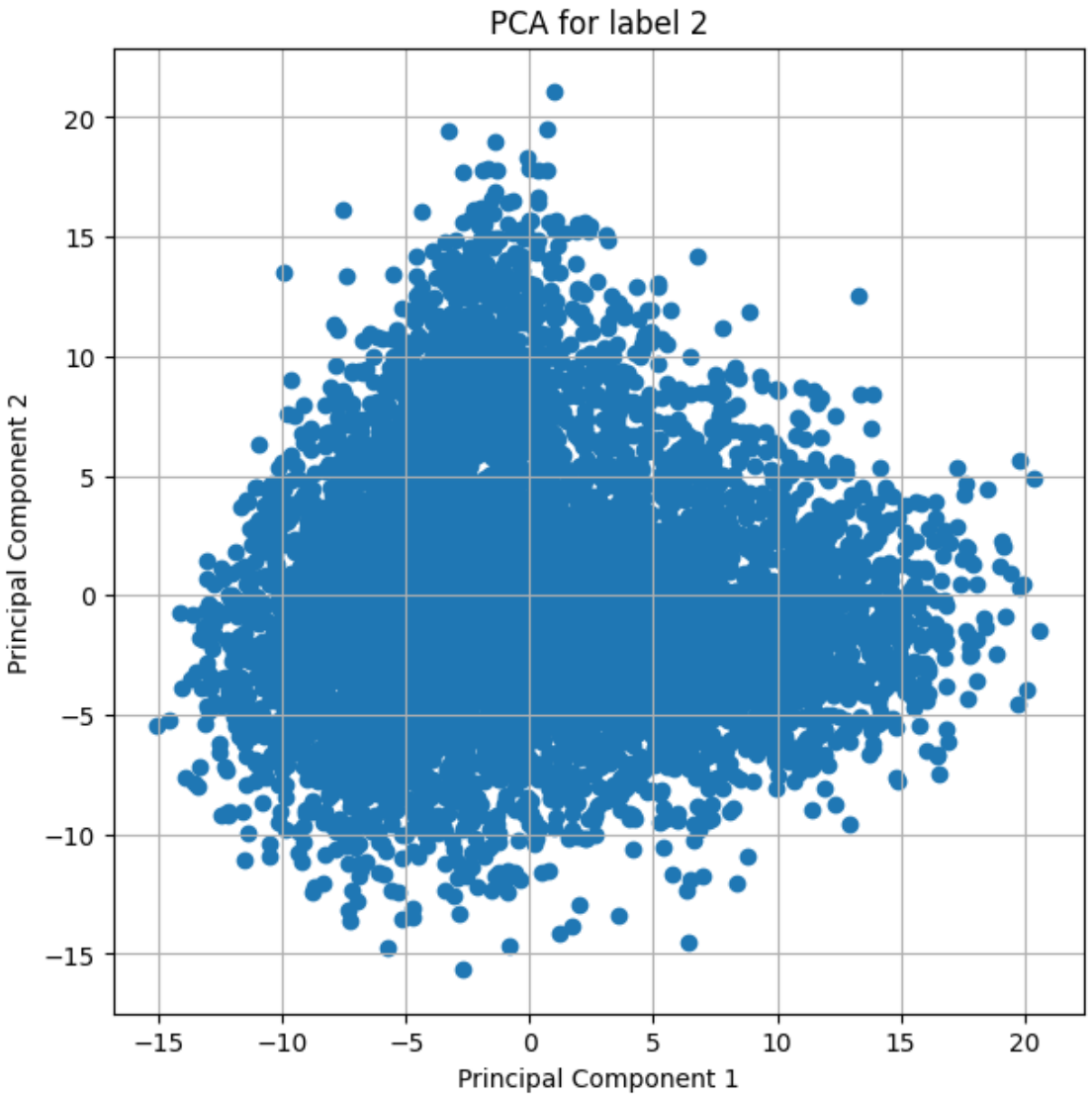
```
(6903, 50)
```

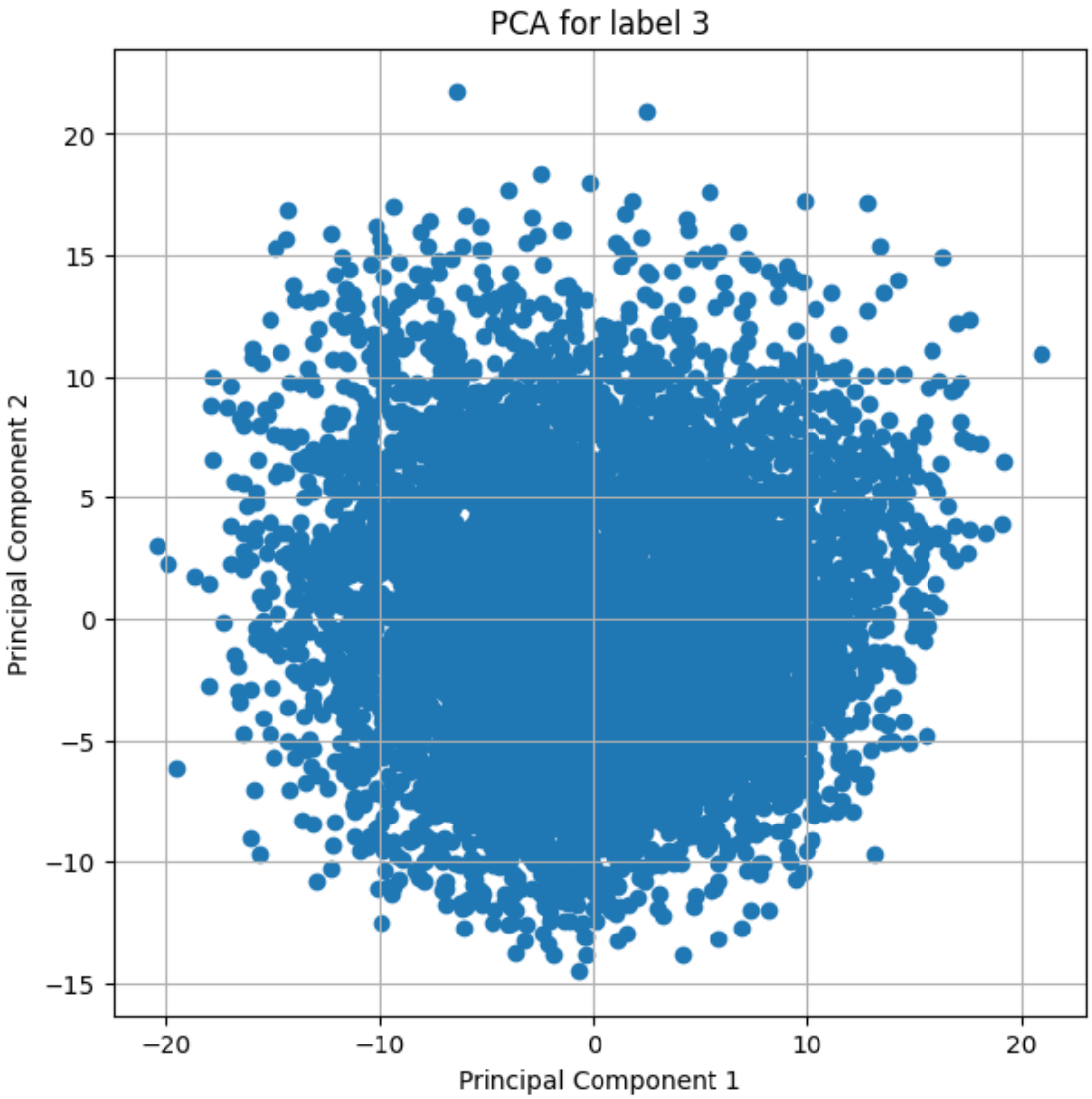
b. Plot first PC vs. Second PC

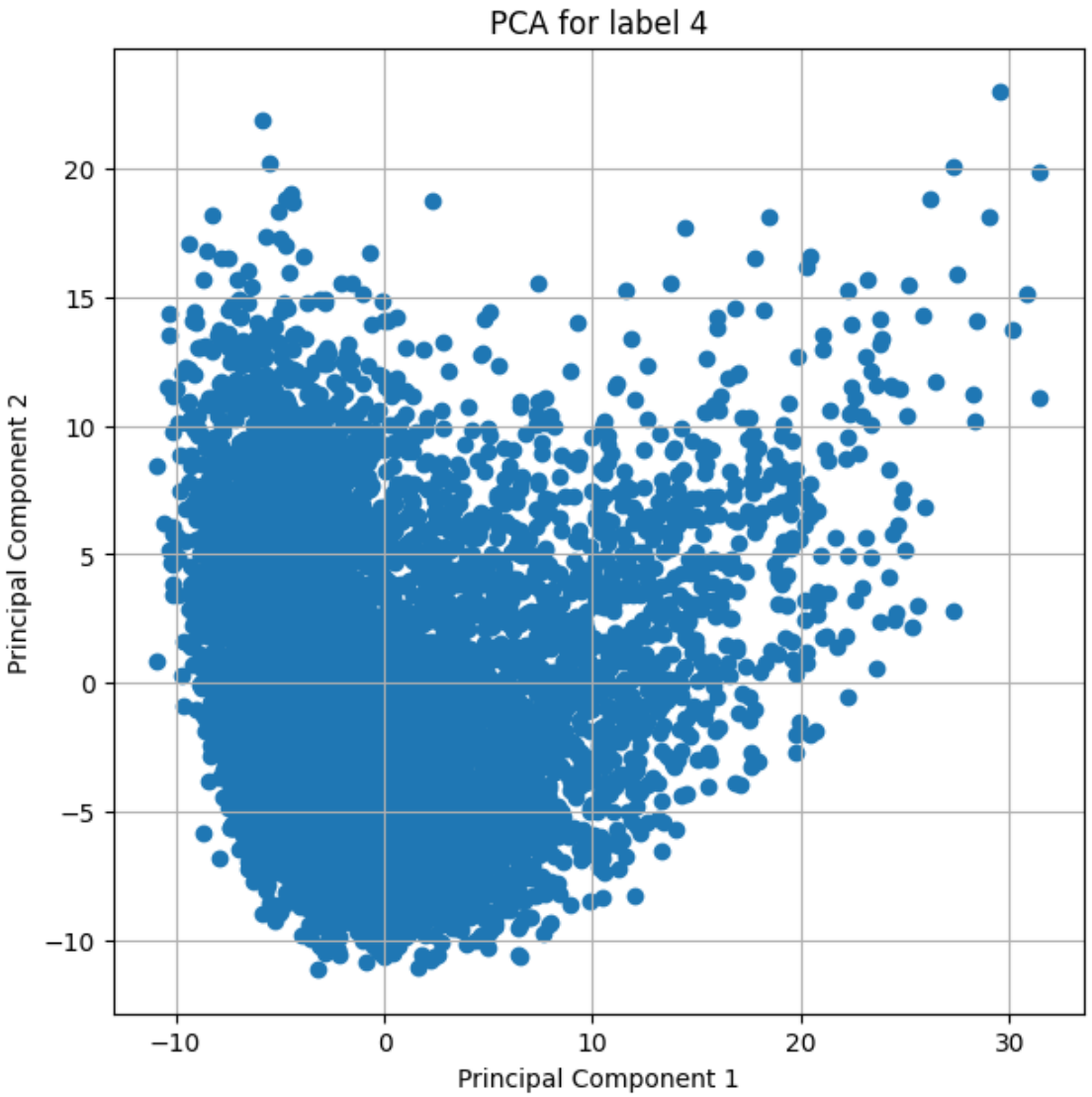
[105]:
```
for i in range(10):
    plt.figure(figsize=(7, 7))
    plt.scatter(PCASets[i][:, 0], PCASets[i][:, 1])
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title(f'PCA for label {i}')
    plt.grid(True)
    plt.show()
```

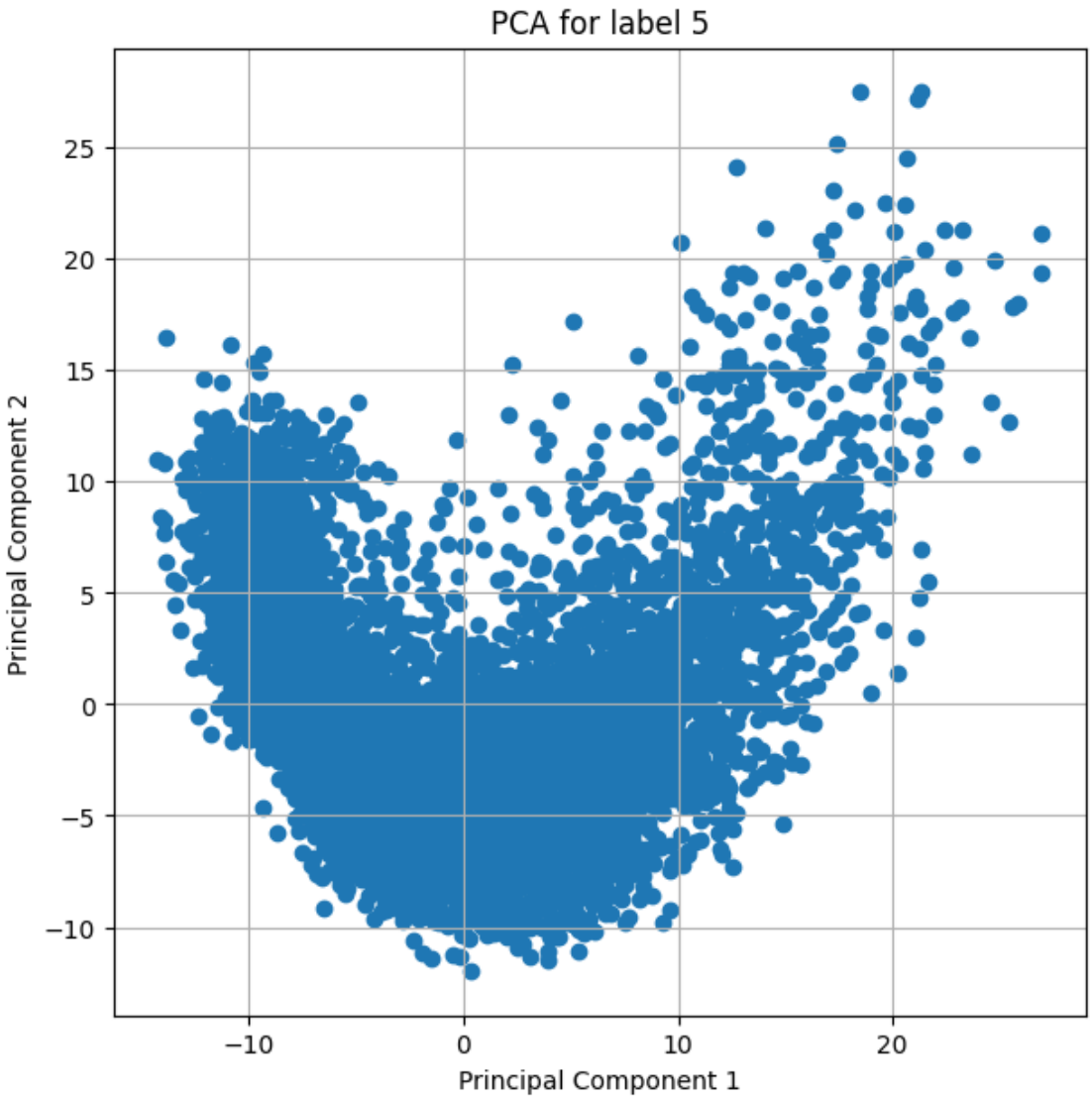

PCA for label 0
Principal Component 2
Principal Component 1
15
10
5
0
−5
−10
−20
−10
0
10
20

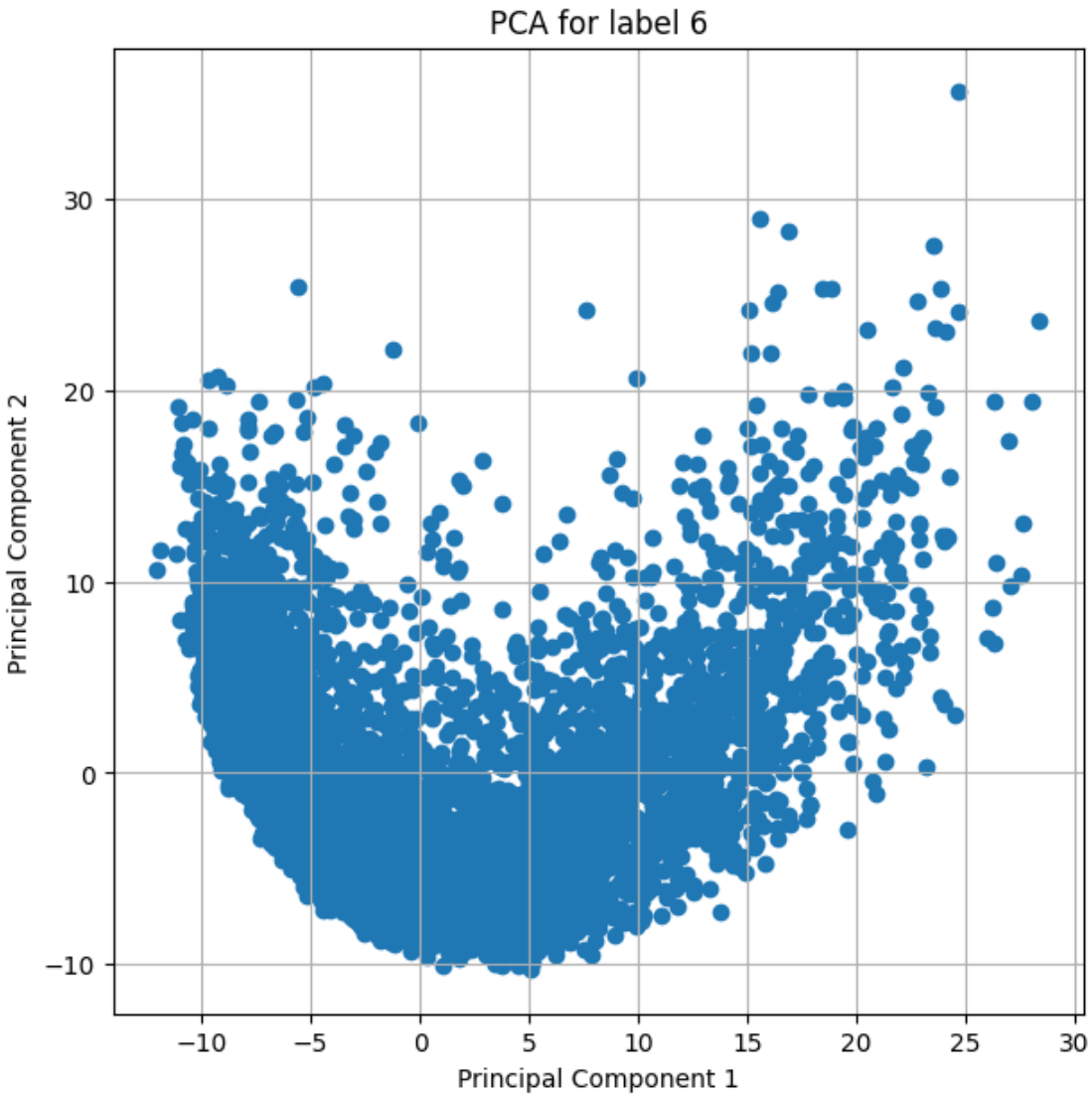

PCA for label 1
Principal Component 2
Principal Component 1
80
60
40
20
0
−20
−10
0
10
20
30

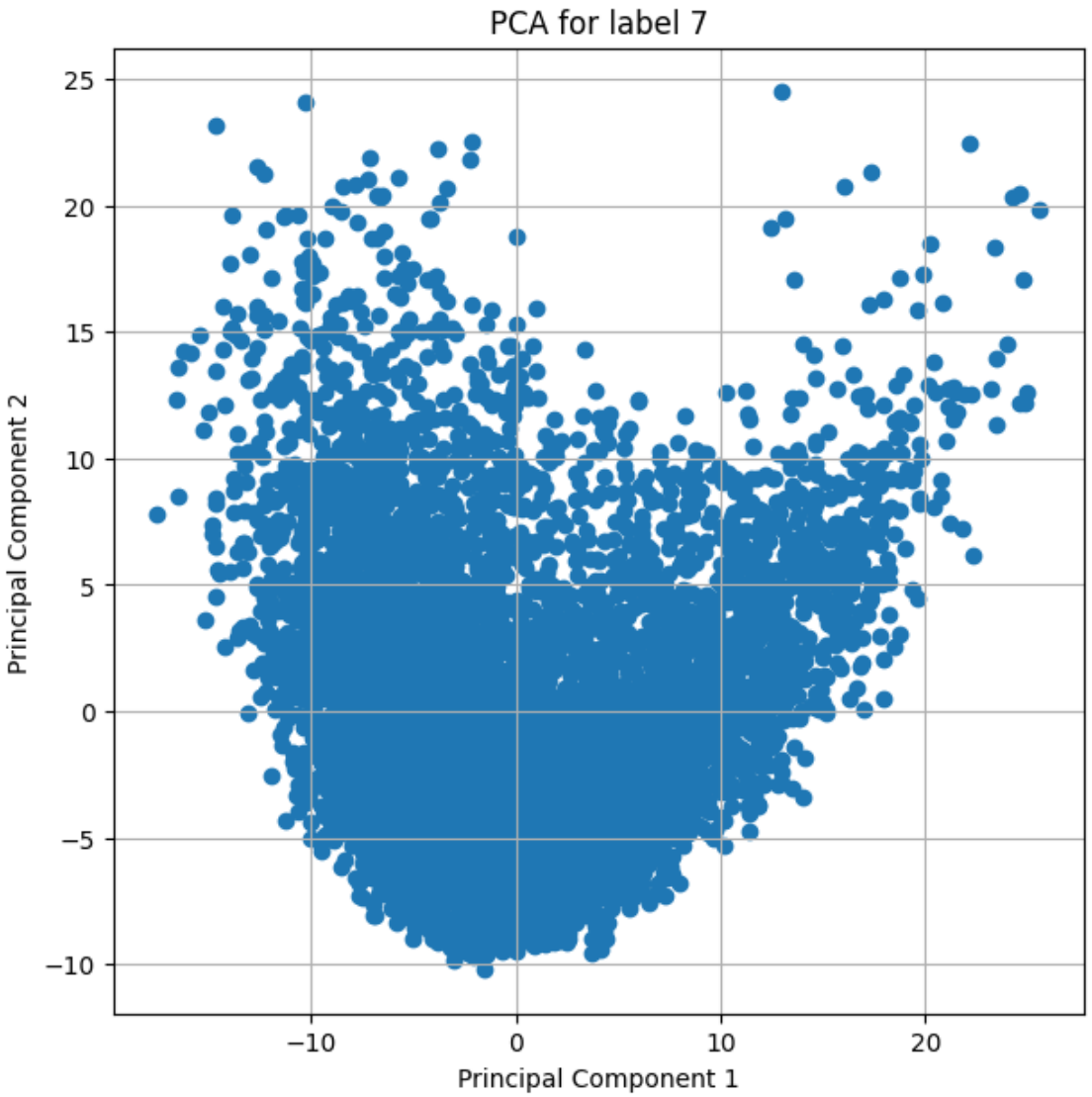

PCA for label 2
Principal Component 2
Principal Component 1
20
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15
20

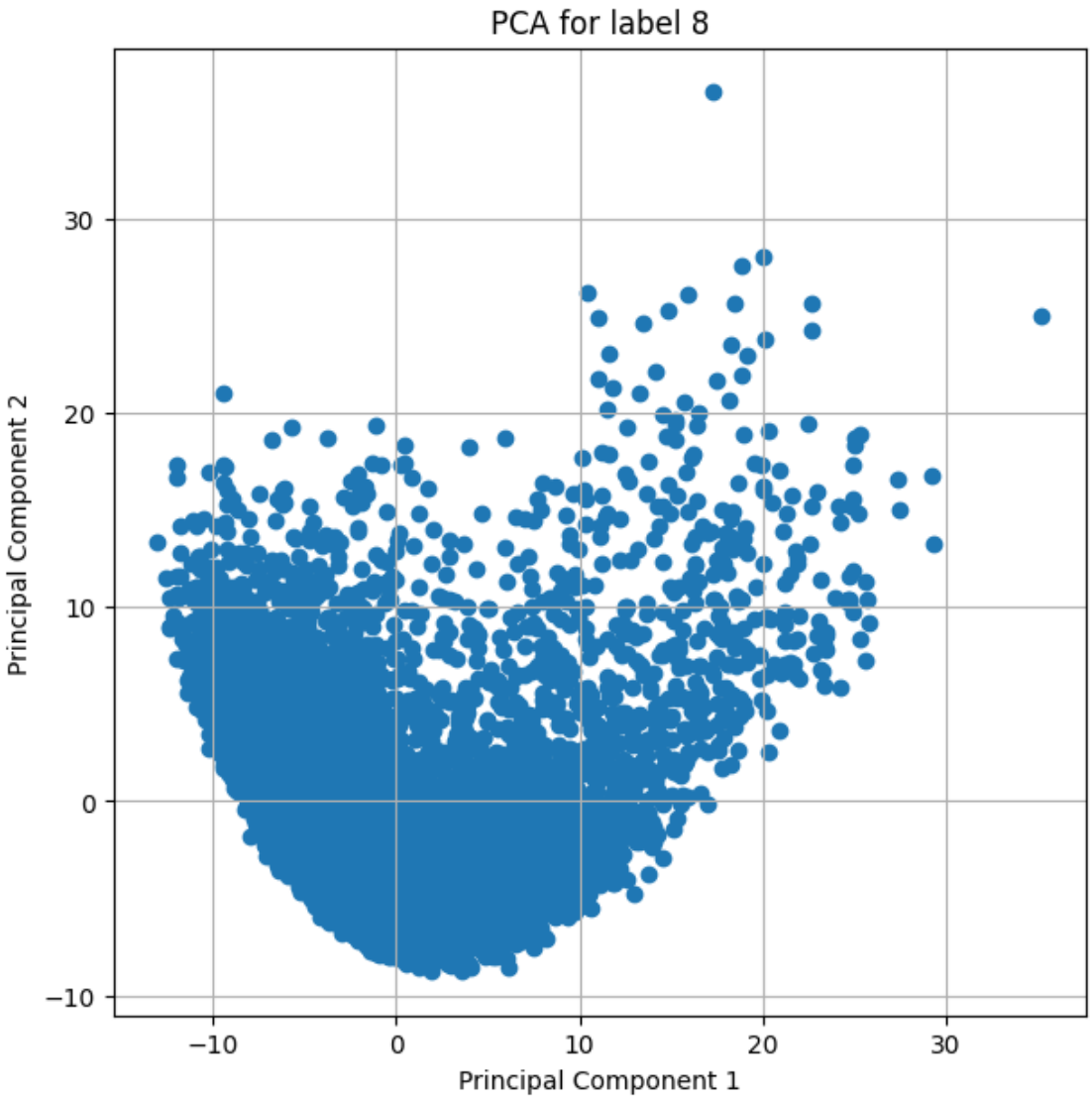

PCA for label 3
Principal Component 2
Principal Component 1
20
15
10
5
0
−5
−10
−15
−20
−10
0
10
20

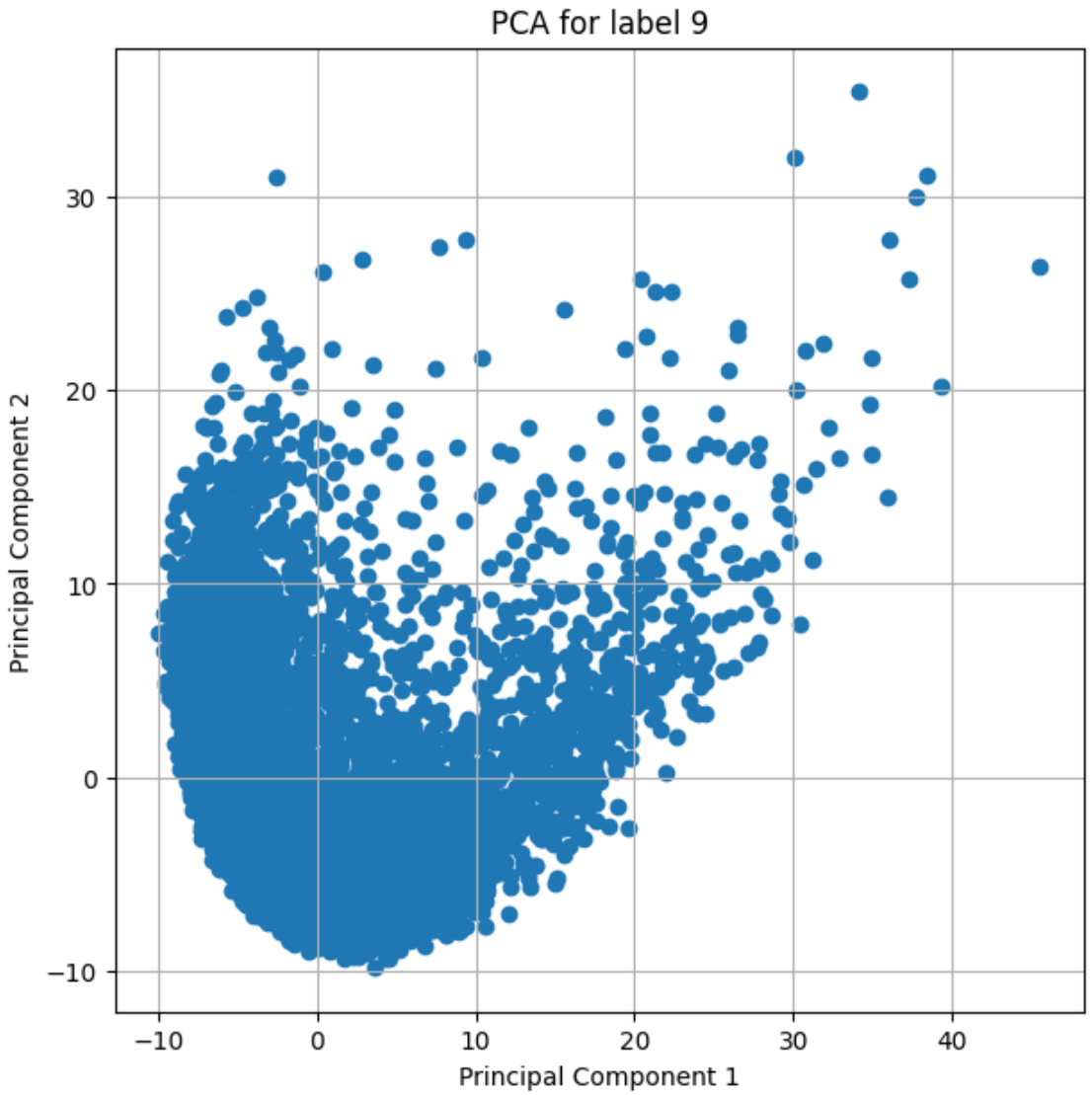

PCA for label 4
Principal Component 2
Principal Component 1
20
15
10
5
0
−5
−10
−10
0
10
20
30


PCA for label 5
Principal Component 2
Principal Component 1
25
20
15
10
5
0
−5
−10
−10
0
10
20


PCA for label 6
Principal Component 2
Principal Component 1
30
20
10
0
−10
−10
−5
0
5
10
15
20
25
30


PCA for label 7
Principal Component 2
Principal Component 1
25
20
15
10
5
0
−5
−10
−10
0
10
20

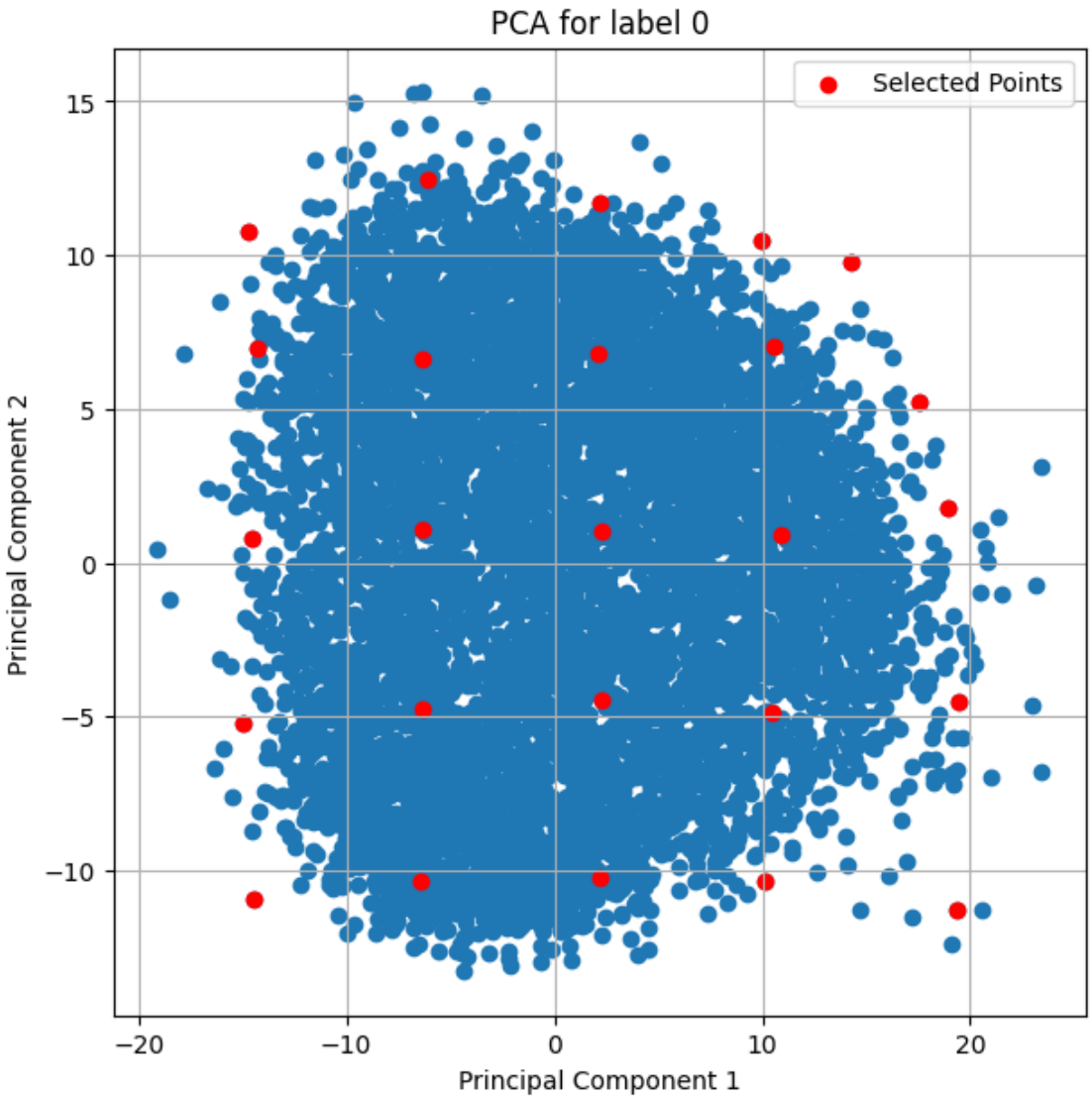

PCA for label 8
Principal Component 2
Principal Component 1
30
20
10
0
−10
−10
0
10
20
30

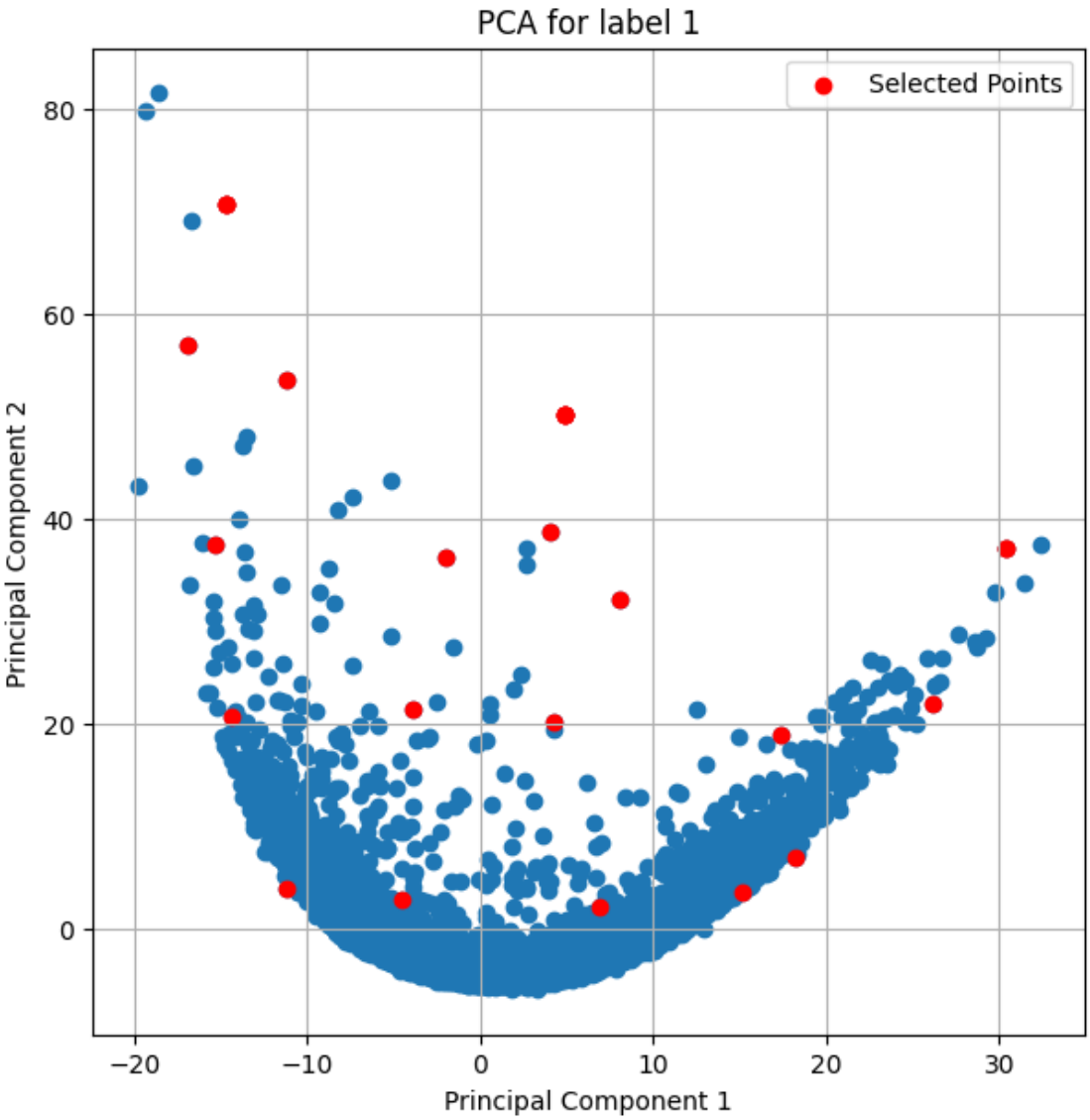


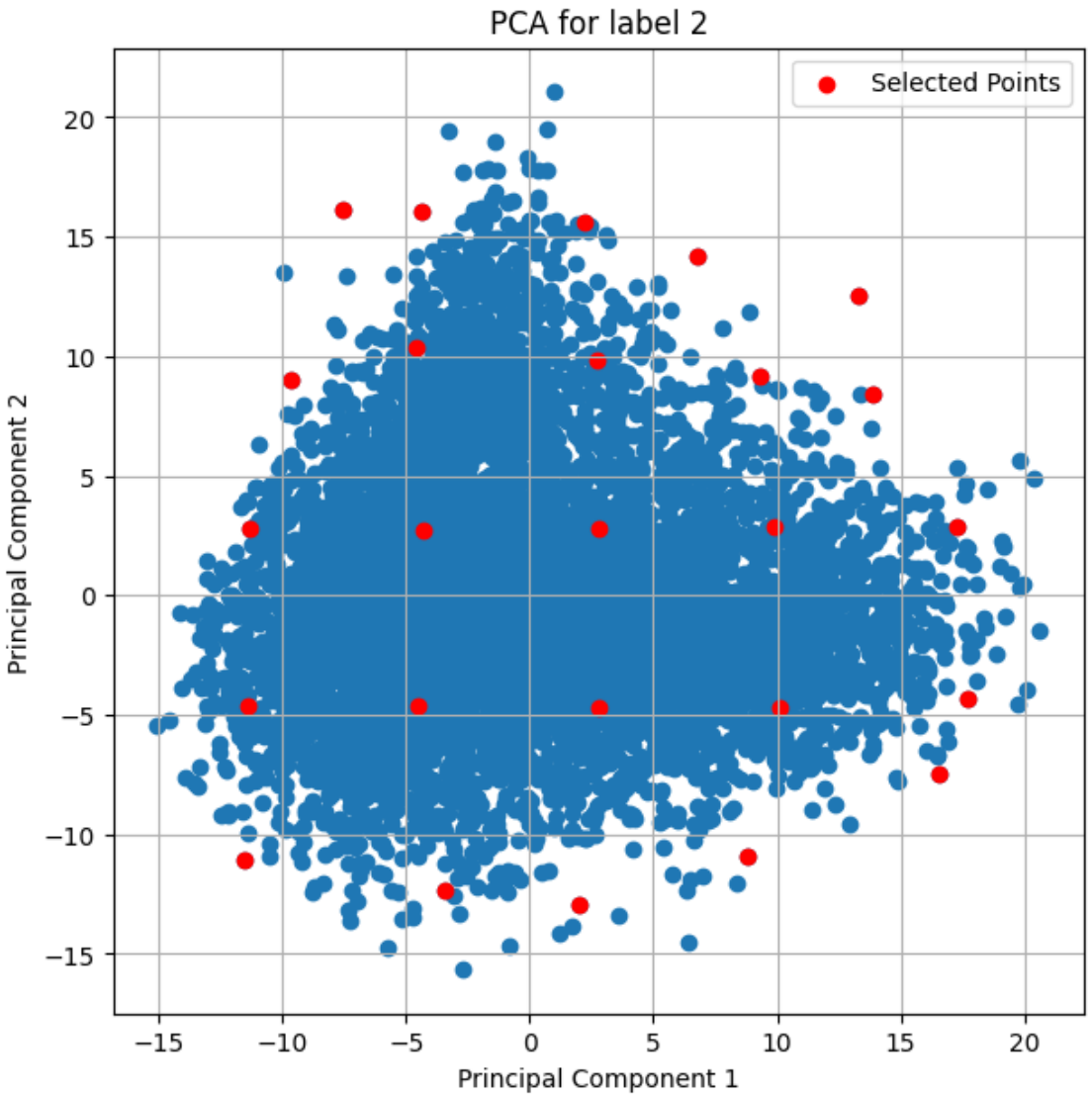
c. Assume the points in this scatter plot are spread between (1, 2) and (1, 2) (which are the min and max of PC1 and PC2). Split this space into a 5x5 grid, and for each cell select a point that is closest to the center of that cell. Highlight these points in the scatter plot from the previous step.

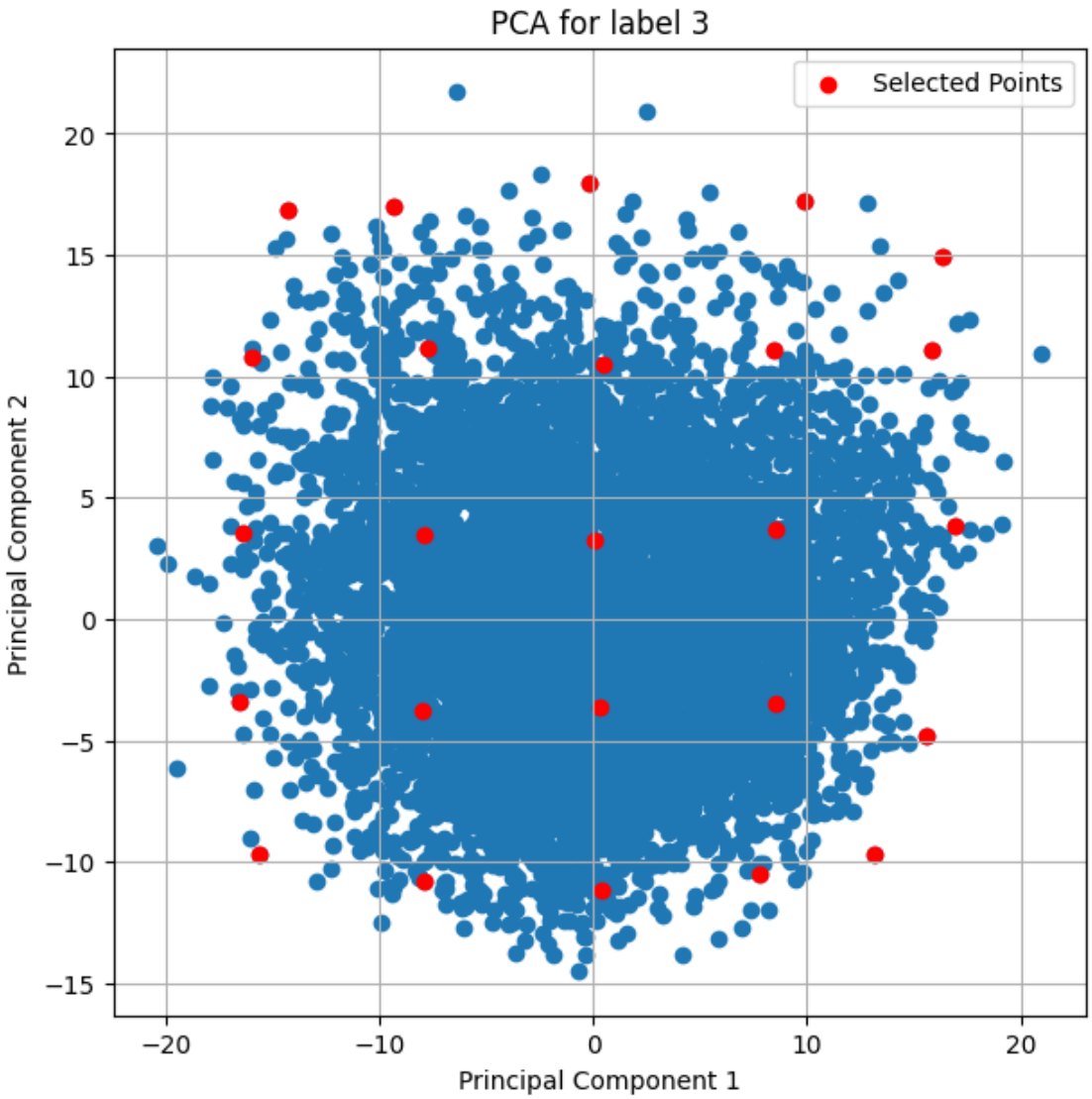
[106]:
```
PointsList = []
for s in range(10):
    x1, x2 = np.min(PCASets[s][:, 0]), np.max(PCASets[s][:, 0])
    y1, y2 = np.min(PCASets[s][:, 1]), np.max(PCASets[s][:, 1])
    #print(x1,x2)
    #print(y1,y2)

    gridX = np.linspace(x1, x2, 6)
```

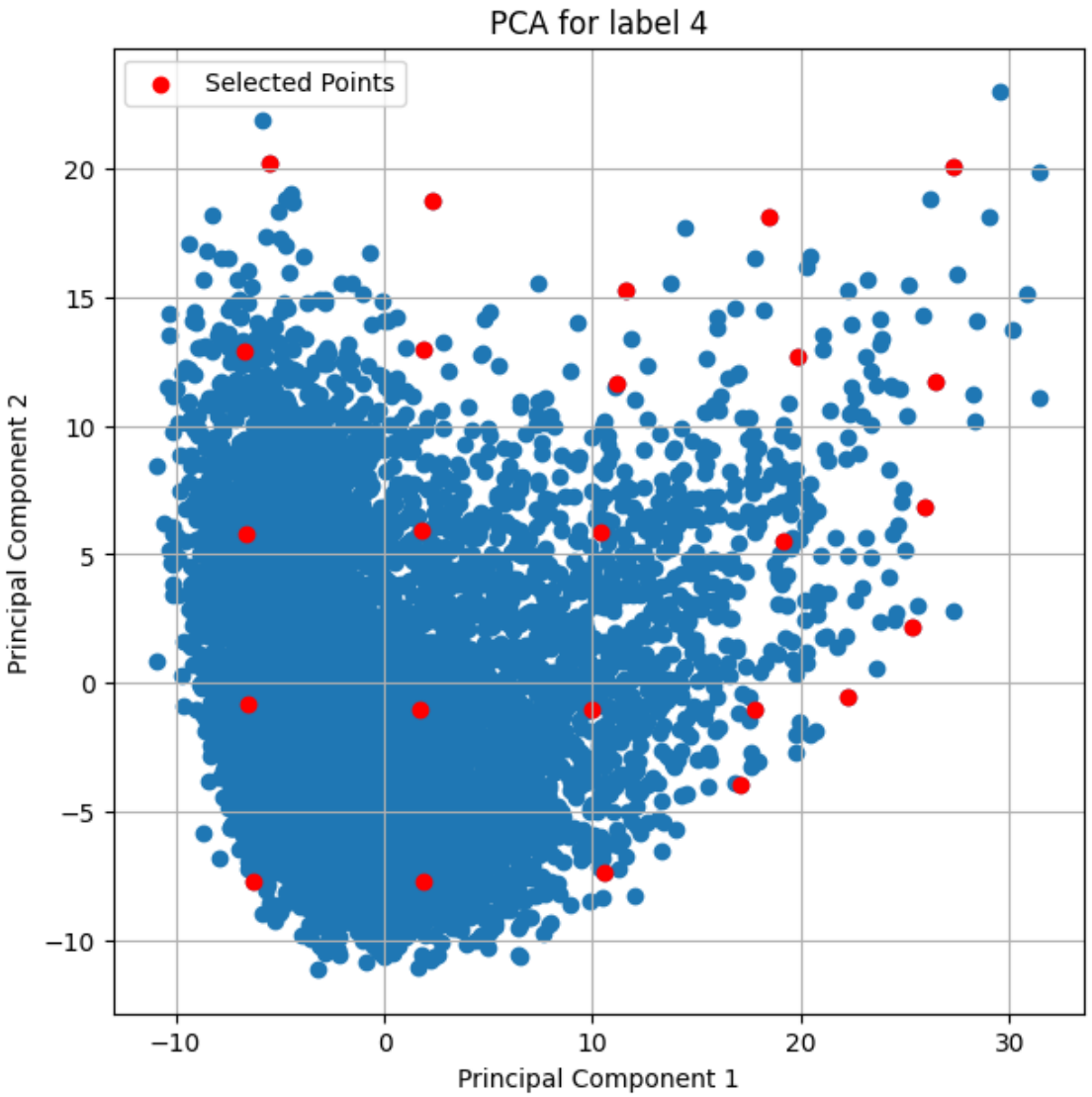
```
    gridY = np.linspace(y1, y2, 6)
    #print(gridX)
    #print(gridY)

    Points = []
    for i in range(len(gridX)-1):
        for j in range(len(gridY)-1):
            centerX = (gridX[i] + gridX[i+1]) / 2
            centerY = (gridY[j] + gridY[j+1]) / 2
            distance = np.sqrt((PCASets[s][:, 0] - centerX)**2 + (PCASets[s][:,␣
↪1] - centerY)**2)
            closest = np.argmin(distance)
            Points.append(closest)
    PointsList.append(Points)

    plt.figure(figsize=(7, 7))
    plt.scatter(PCASets[s][:, 0], PCASets[s][:, 1])
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title(f'PCA for label {s}')

    Points = np.array(Points)
    plt.scatter(PCASets[s][Points, 0], PCASets[s][Points, 1], color='red',␣
↪marker='o', label='Selected Points')
    plt.legend()

    plt.grid(True)
    plt.show()
```

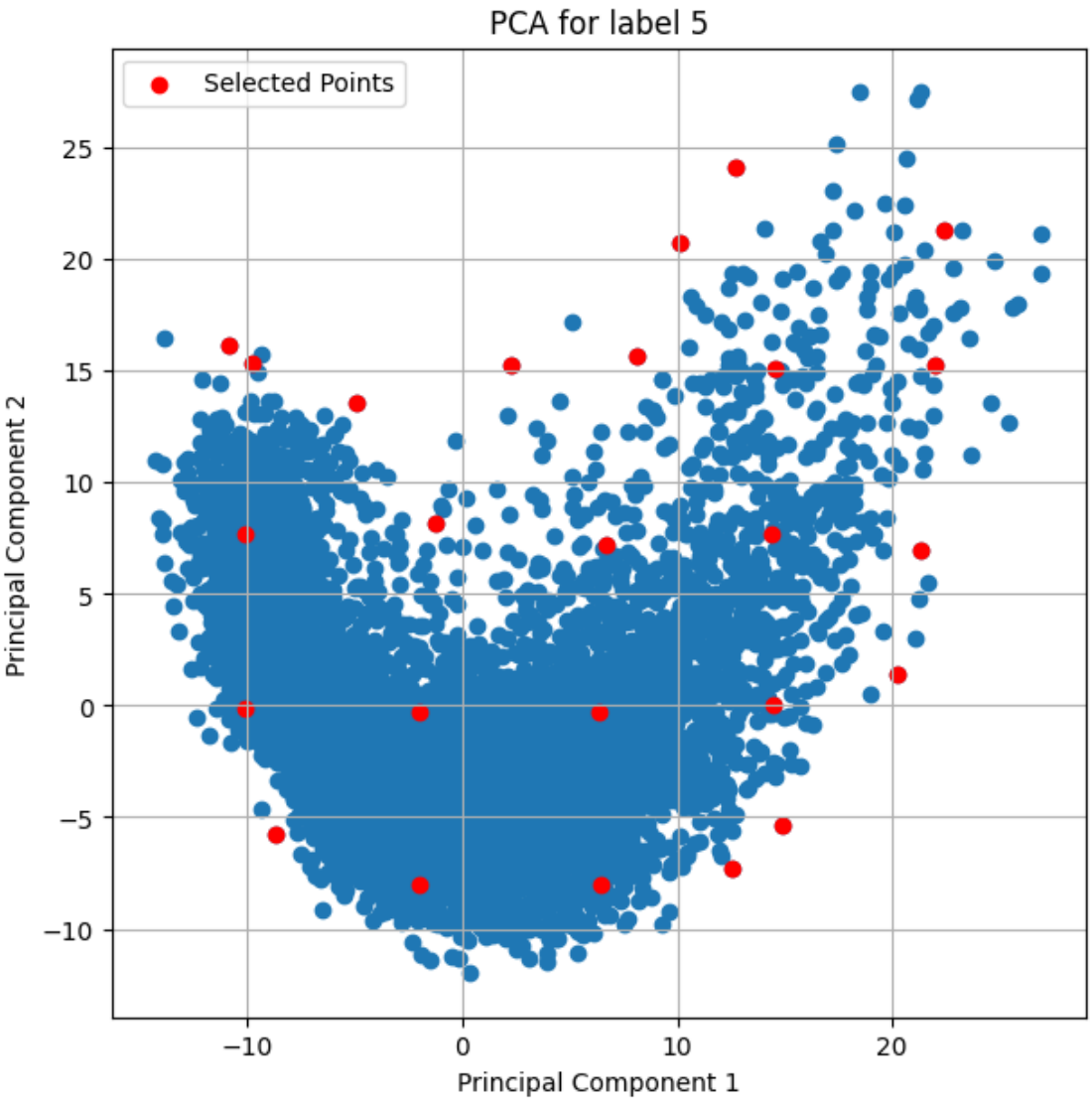

PCA for label 0
Selected Points
Principal Component 2
Principal Component 1
15
10
5
0
−5
−10
−20
−10
0
10
20

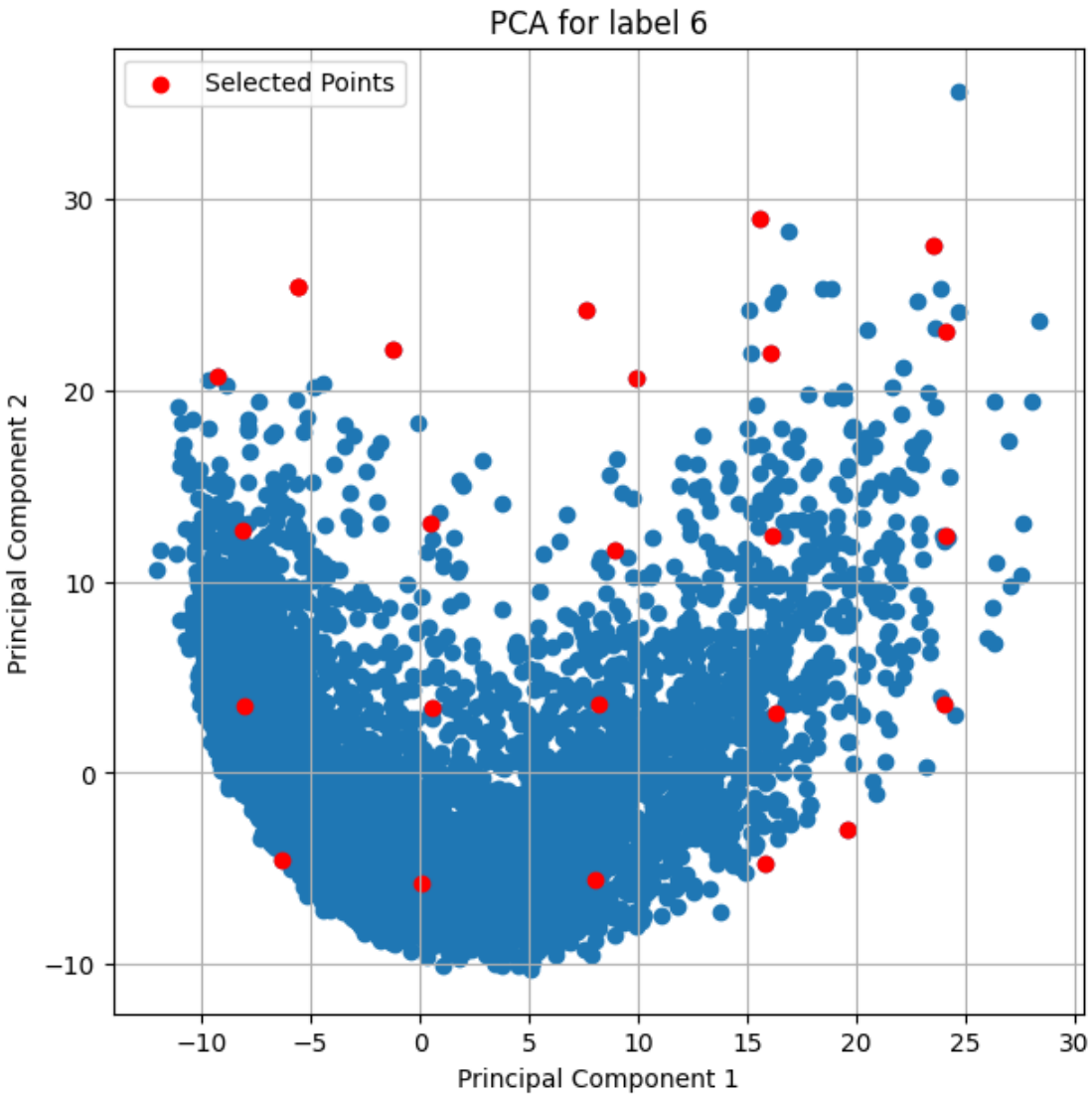

PCA for label 1
Selected Points
Principal Component 2
Principal Component 1
80
60
40
20
0
−20
−10
0
10
20
30

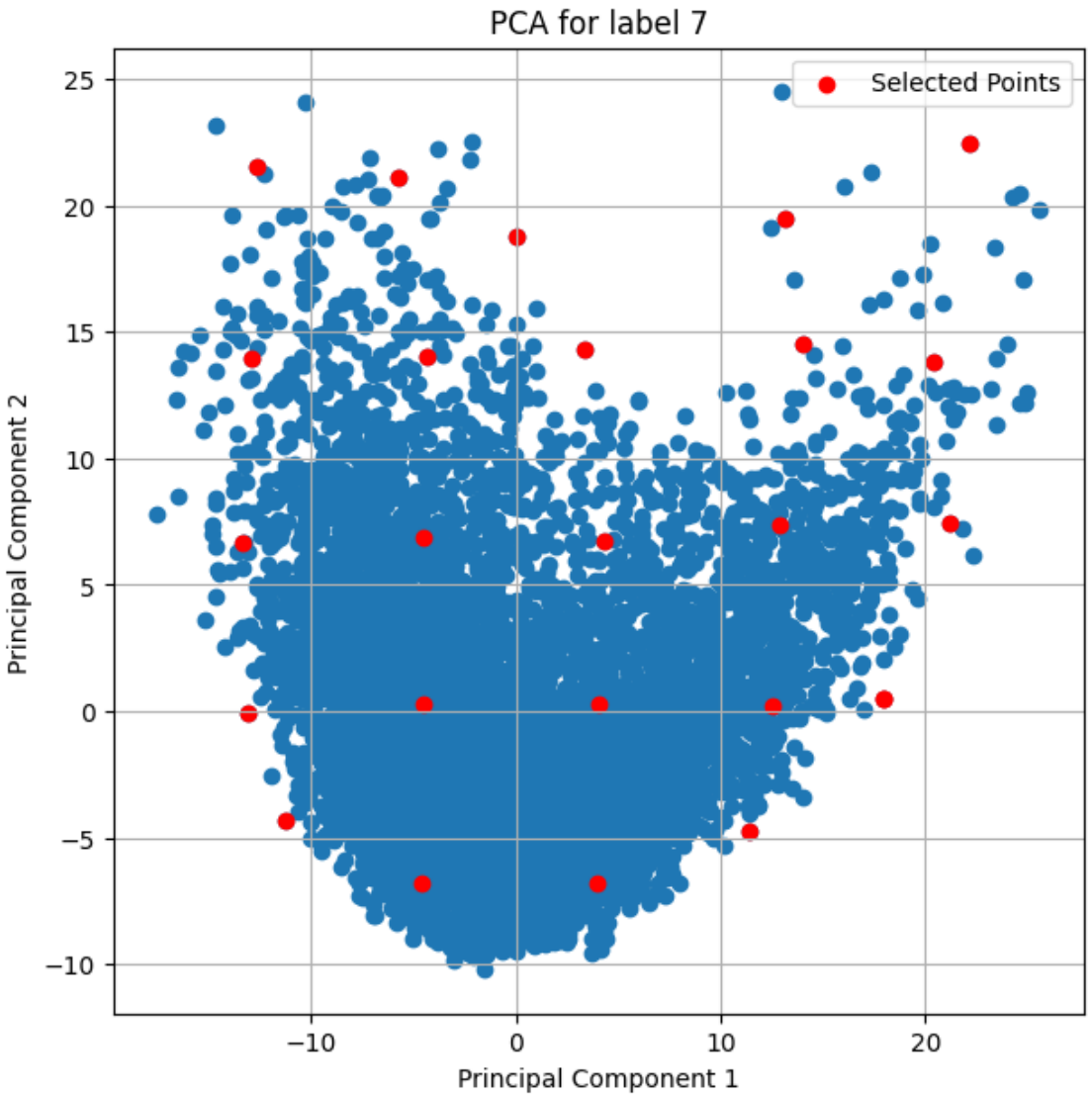

PCA for label 2
Selected Points
Principal Component 2
Principal Component 1
20
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15
20

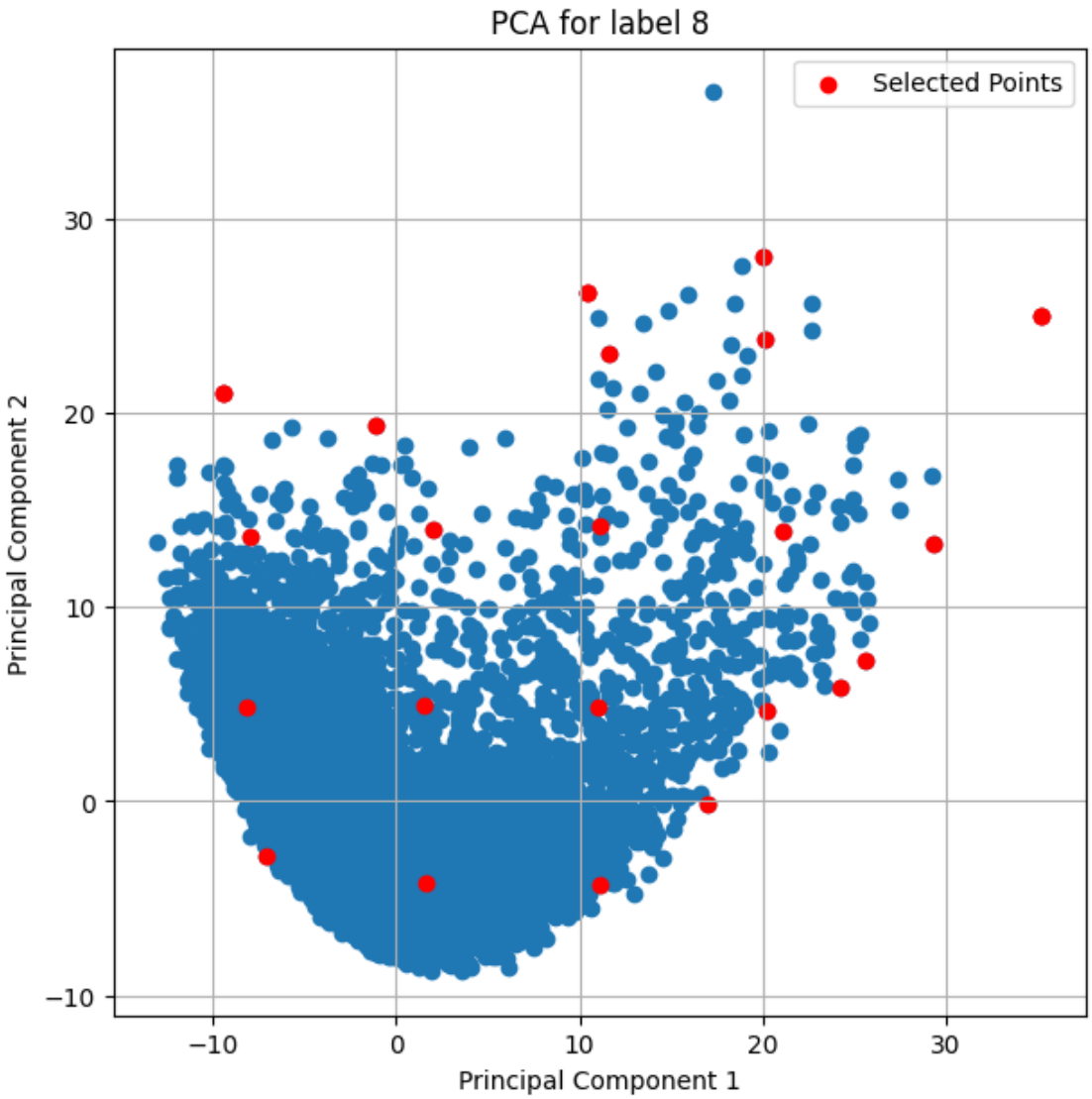

PCA for label 3
Selected Points
Principal Component 2
Principal Component 1
20
15
10
5
0
−5
−10
−15
−20
−10
0
10
20

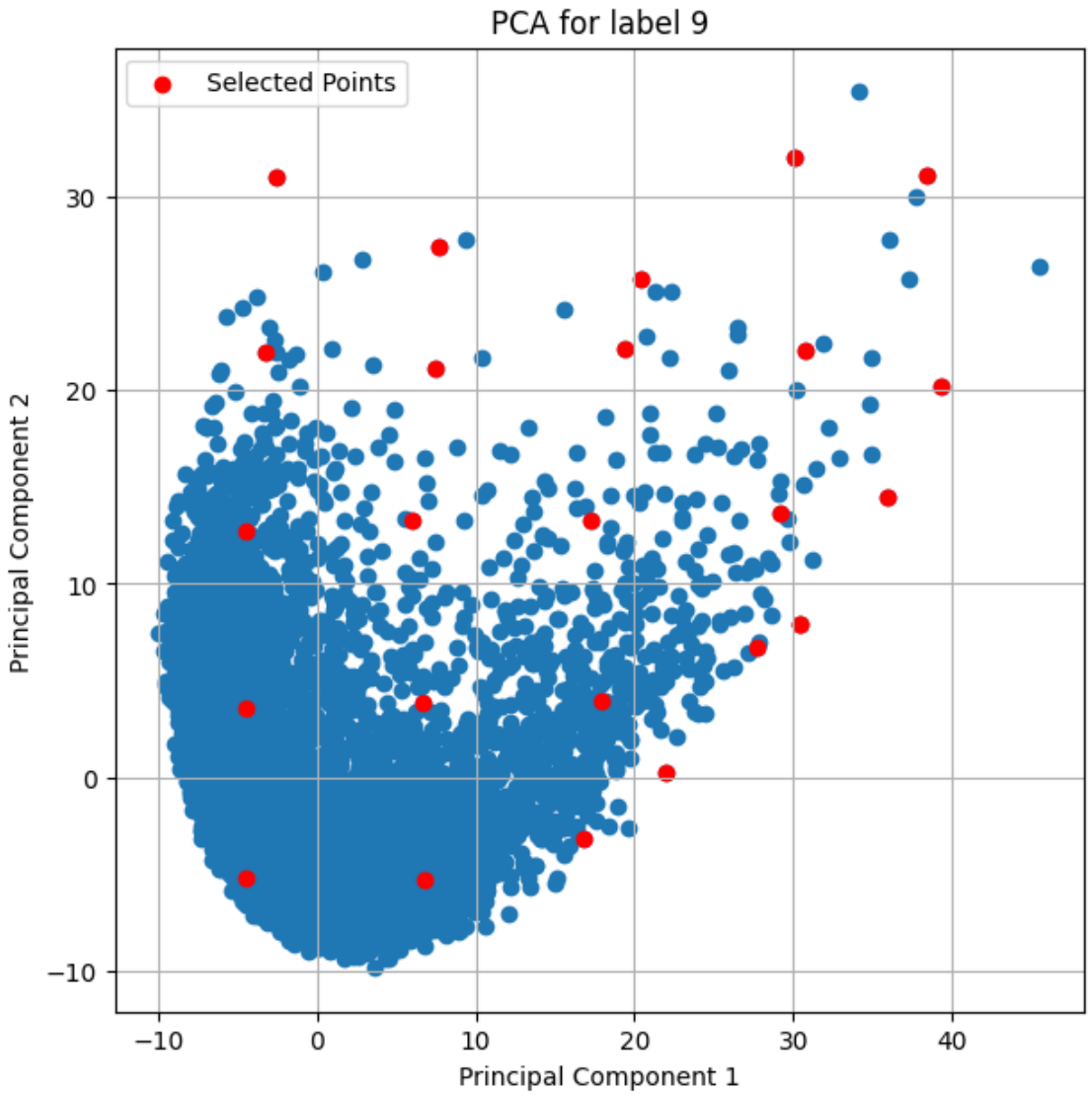

PCA for label 4
Selected Points
Principal Component 2
Principal Component 1
20
15
10
5
0
−5
−10
−10
0
10
20
30


PCA for label 5
Selected Points
Principal Component 2
Principal Component 1
25
20
15
10
5
0
−5
−10
−10
0
10
20


PCA for label 6
Selected Points
Principal Component 2
Principal Component 1
30
20
10
0
−10
−10
−5
0
5
10
15
20
25
30

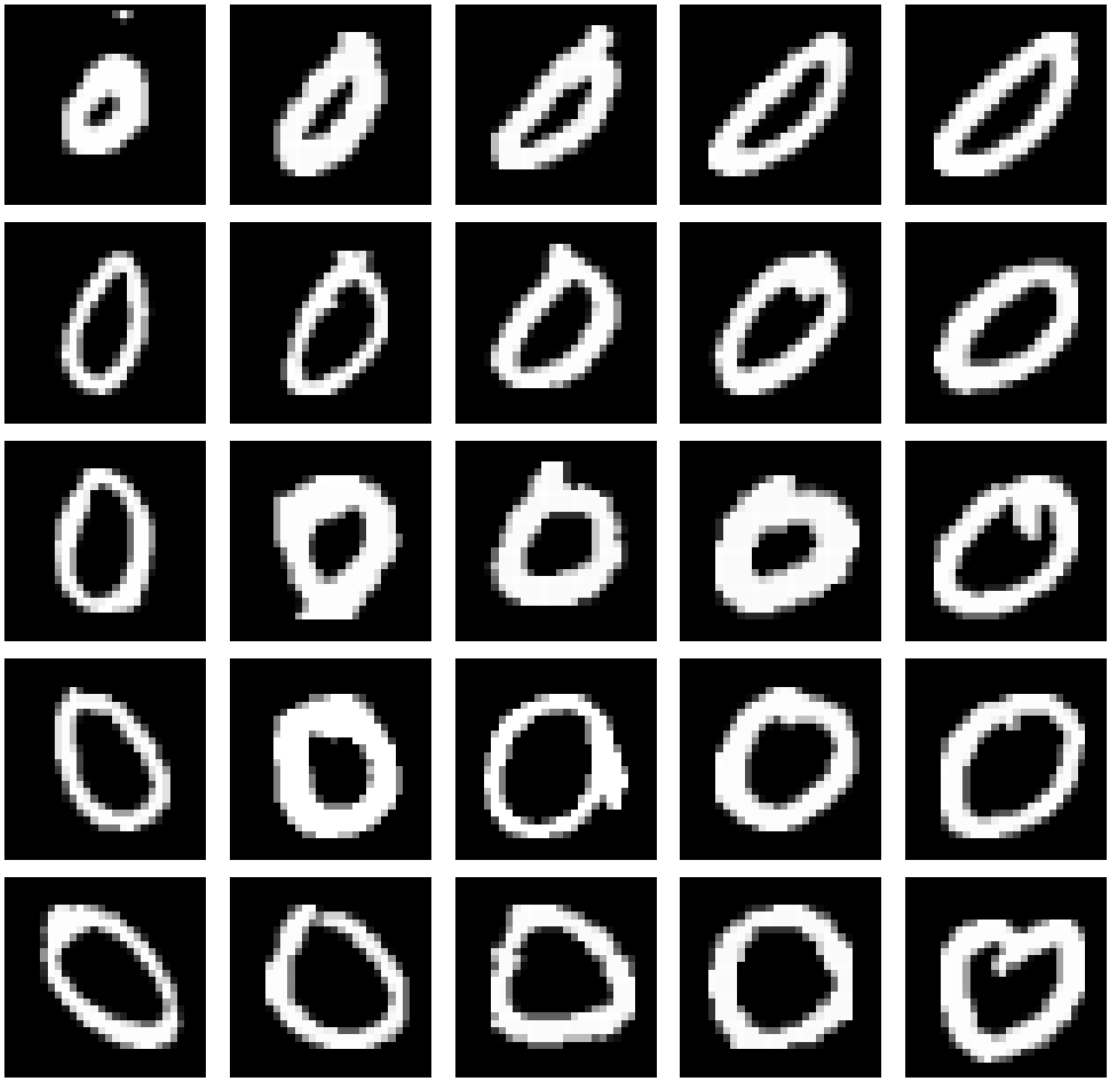

PCA for label 7
Selected Points
Principal Component 2
Principal Component 1
25
20
15
10
5
0
−5
−10
−10
0
10
20


PCA for label 8
Selected Points
Principal Component 2
Principal Component 1
30
20
10
0
−10
−10
0
10
20
30

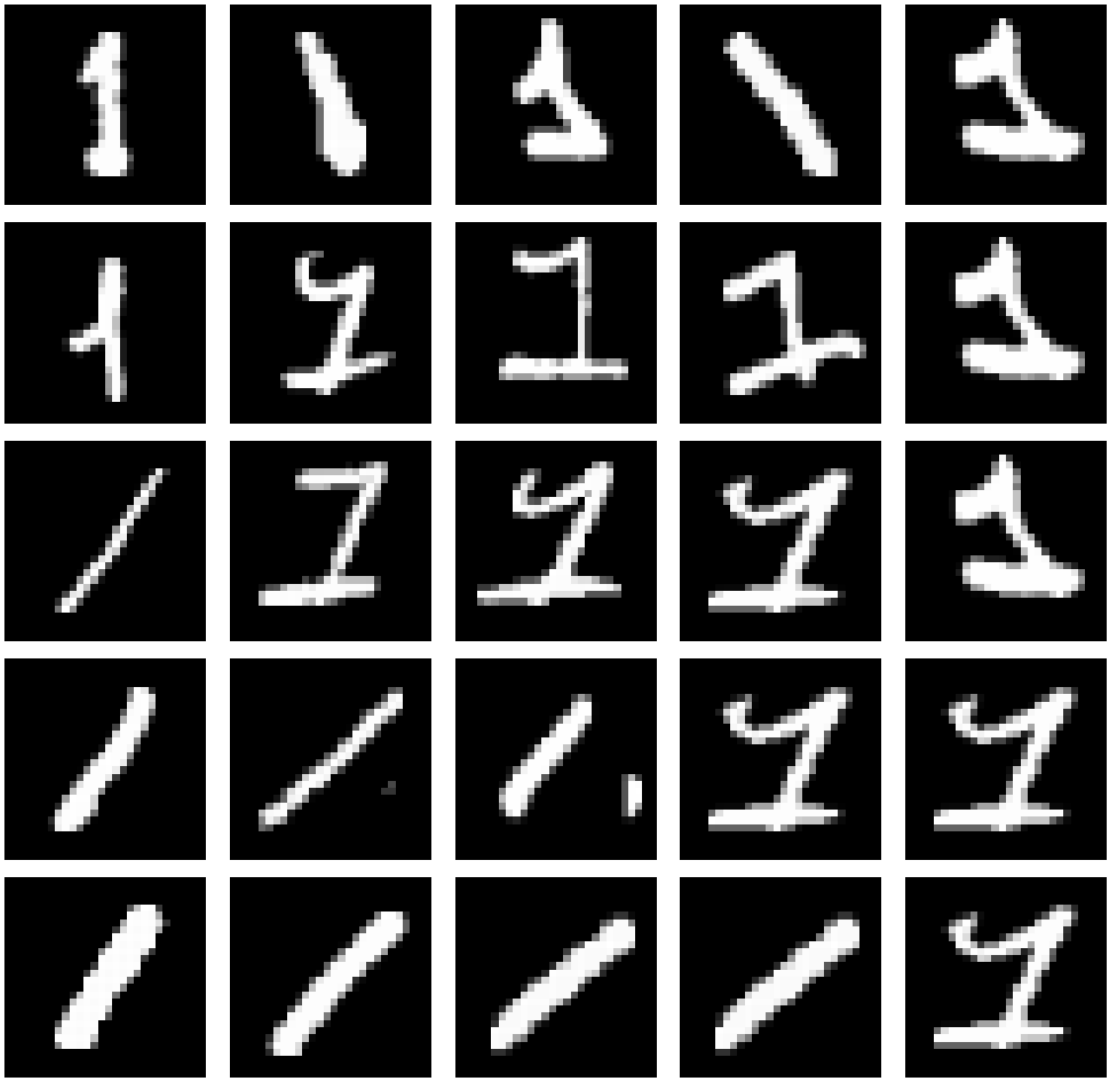


d. Draw the original pictures corresponding to the 25 selecting points. (See figure 14.23 of Element of Statistical Learning for an example)

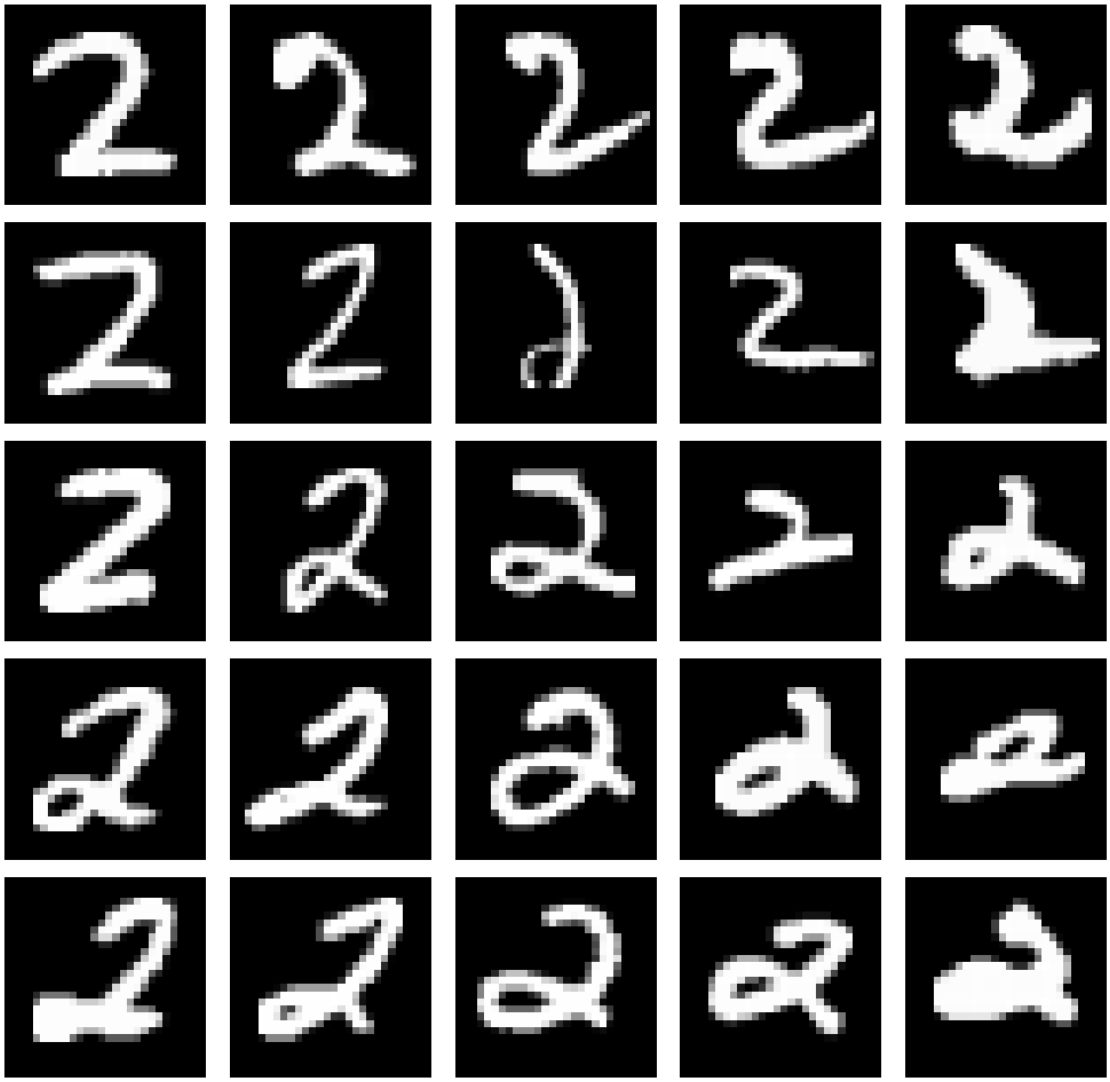
[121]:
```
for u in range(10):
    plt.figure(figsize=(10, 10))
    for i, idx in enumerate(PointsList[u]):
        plt.subplot(5, 5, i + 1)
        plt.imshow(Sets2[u][idx], cmap='gray')
        plt.axis('off')

    plt.suptitle(f'Original pictures for label {u}')
    plt.tight_layout()
    plt.show()
```


Original pictures for label 0

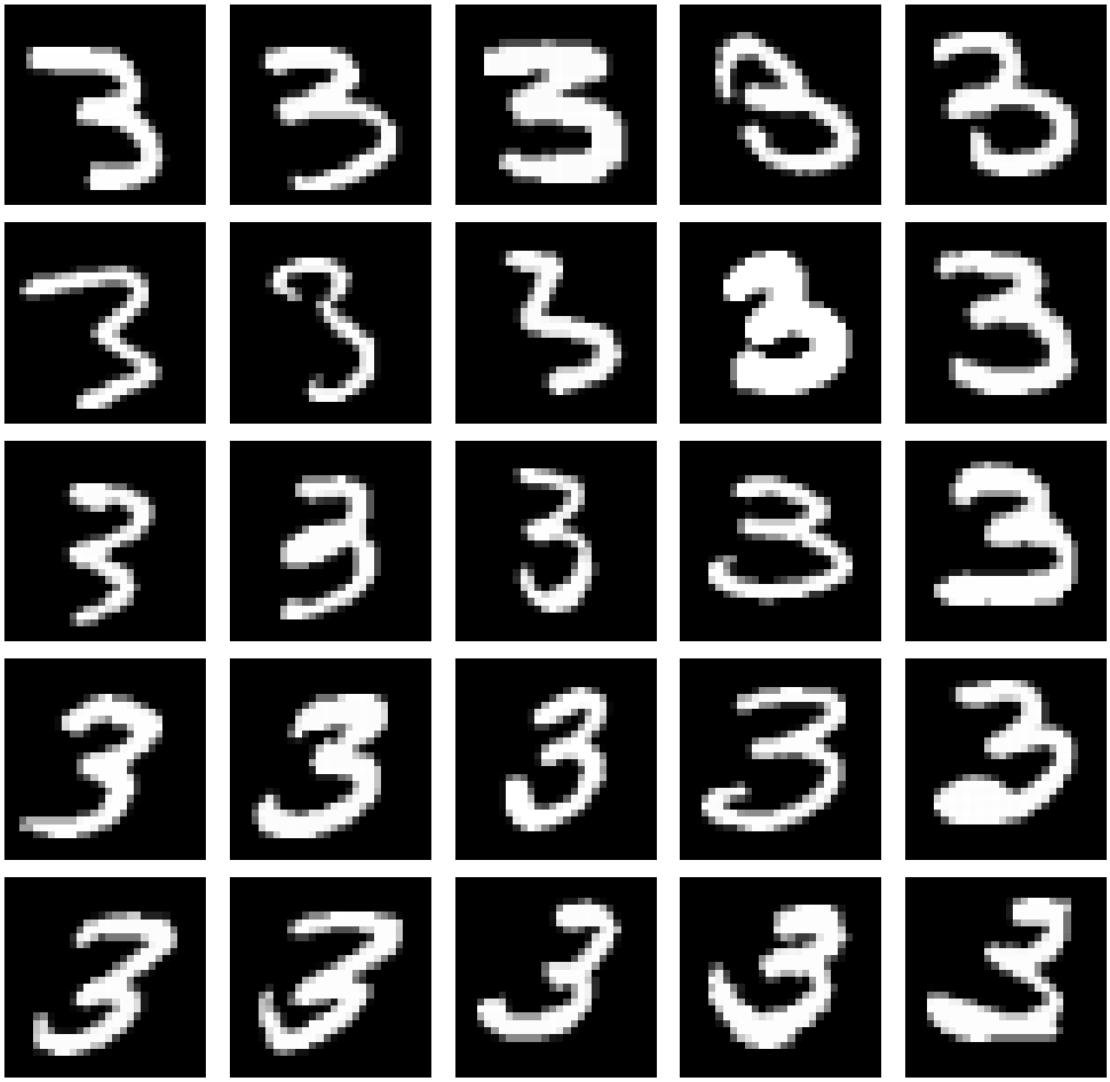
Original pictures for label 1



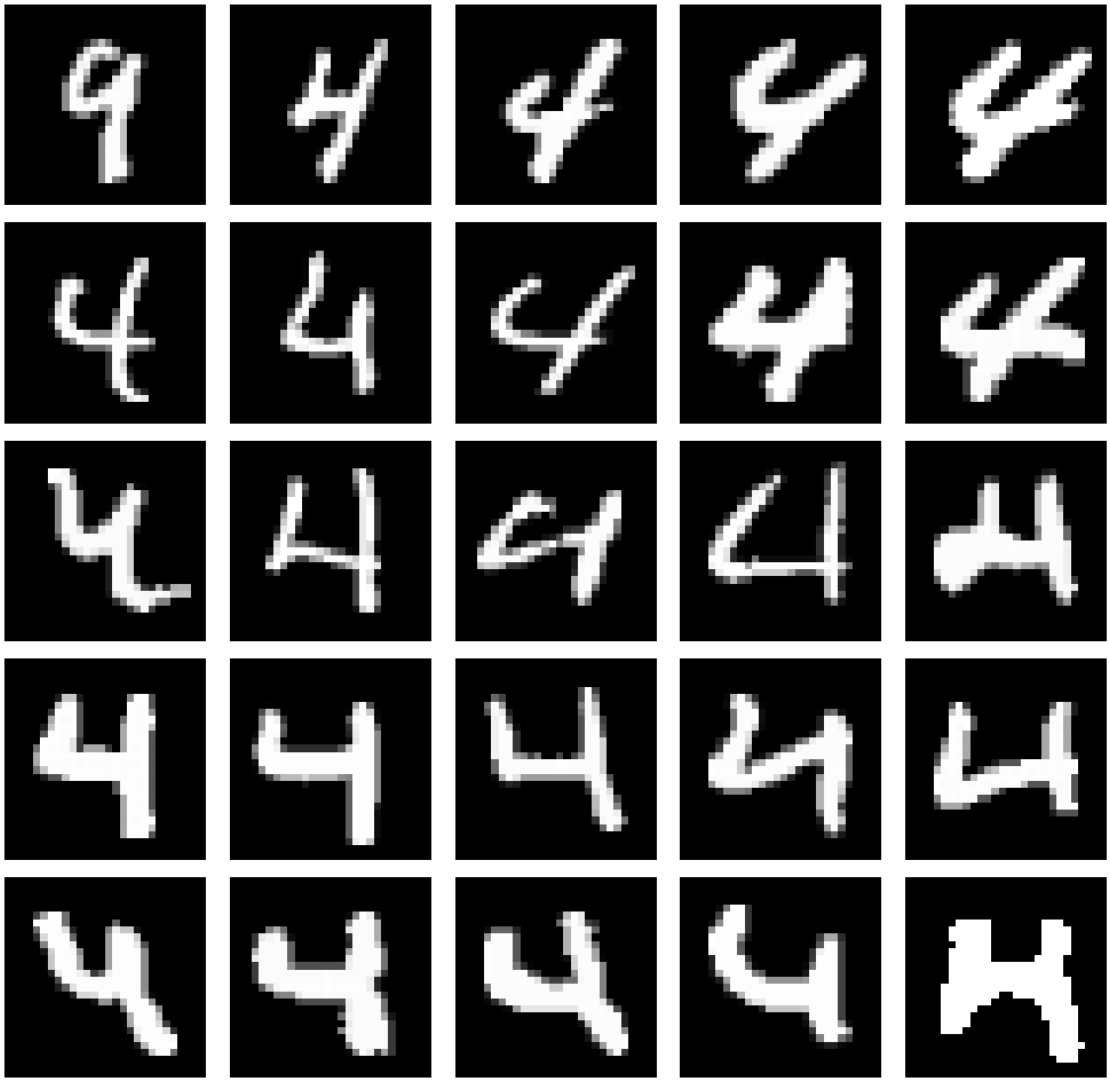
Original pictures for label 2



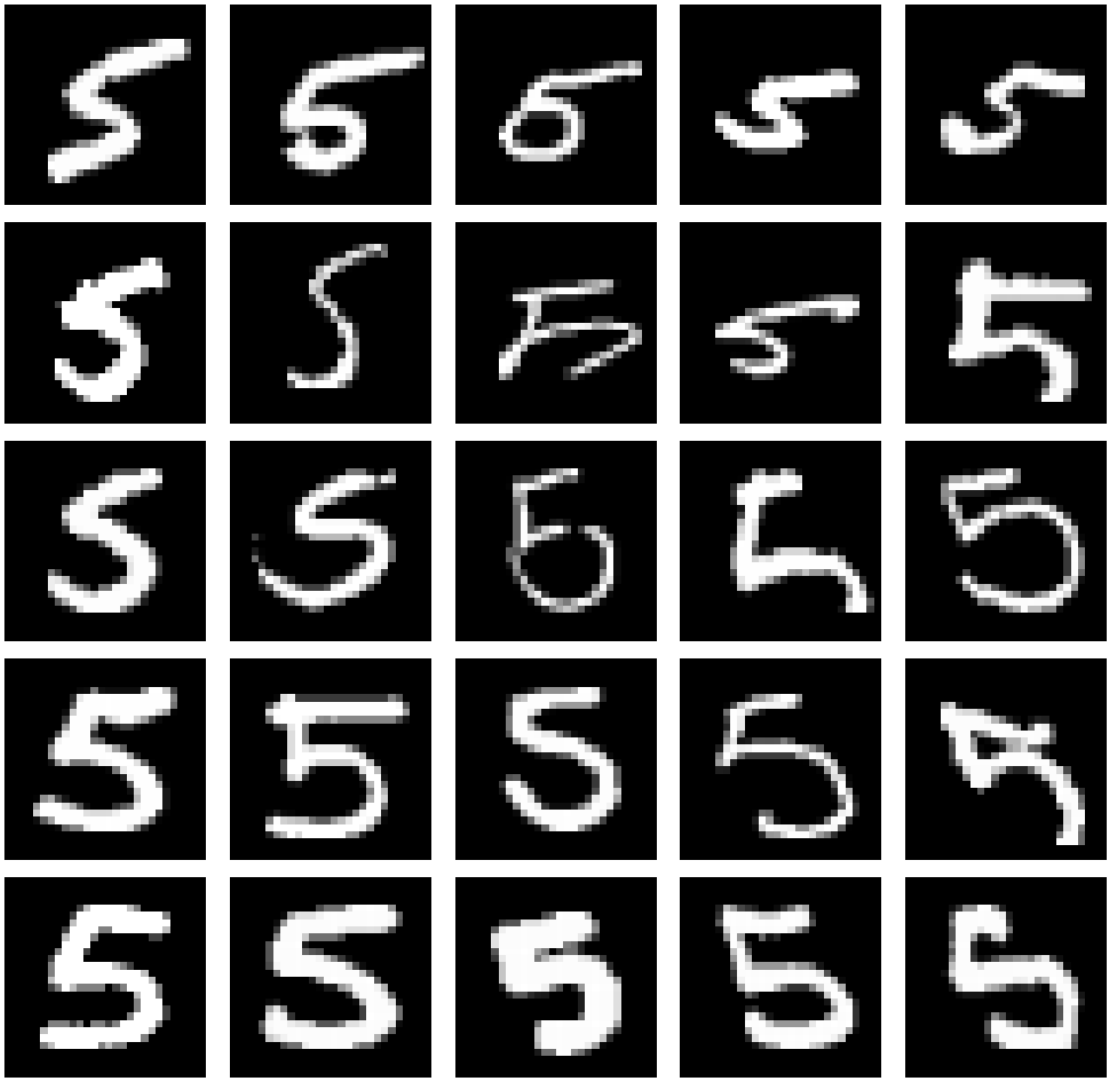
Original pictures for label 3



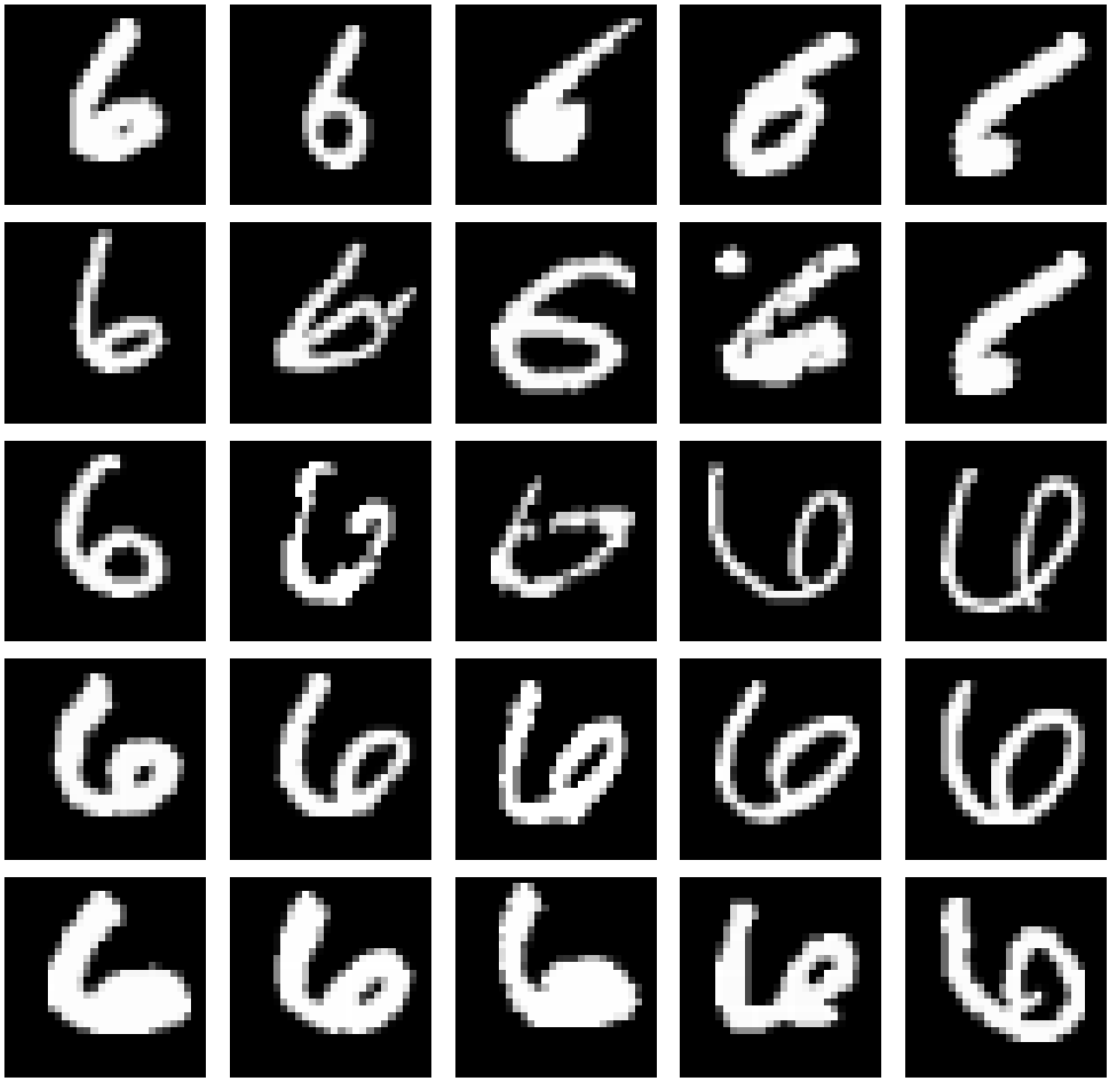
Original pictures for label 4



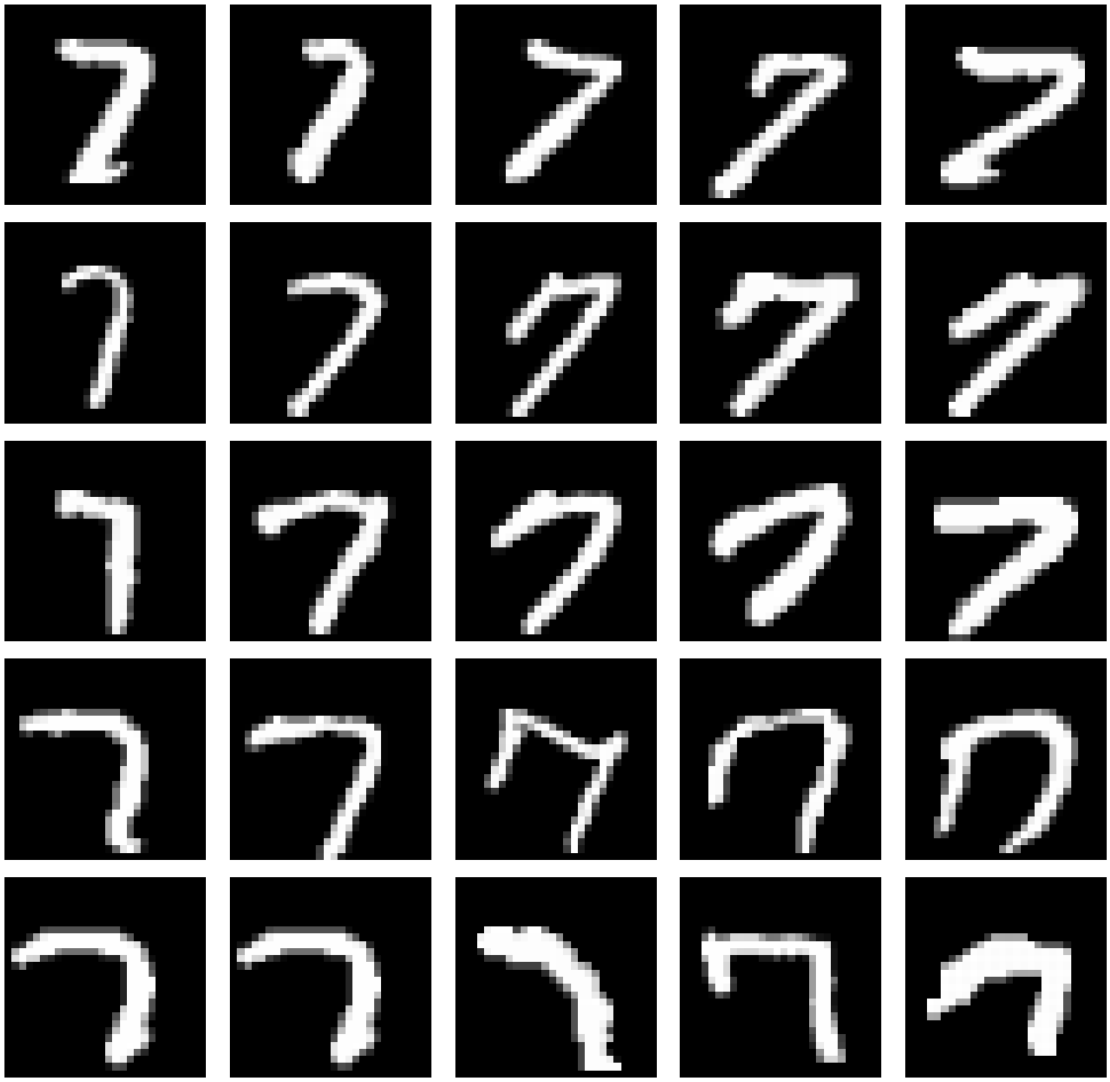
Original pictures for label 5



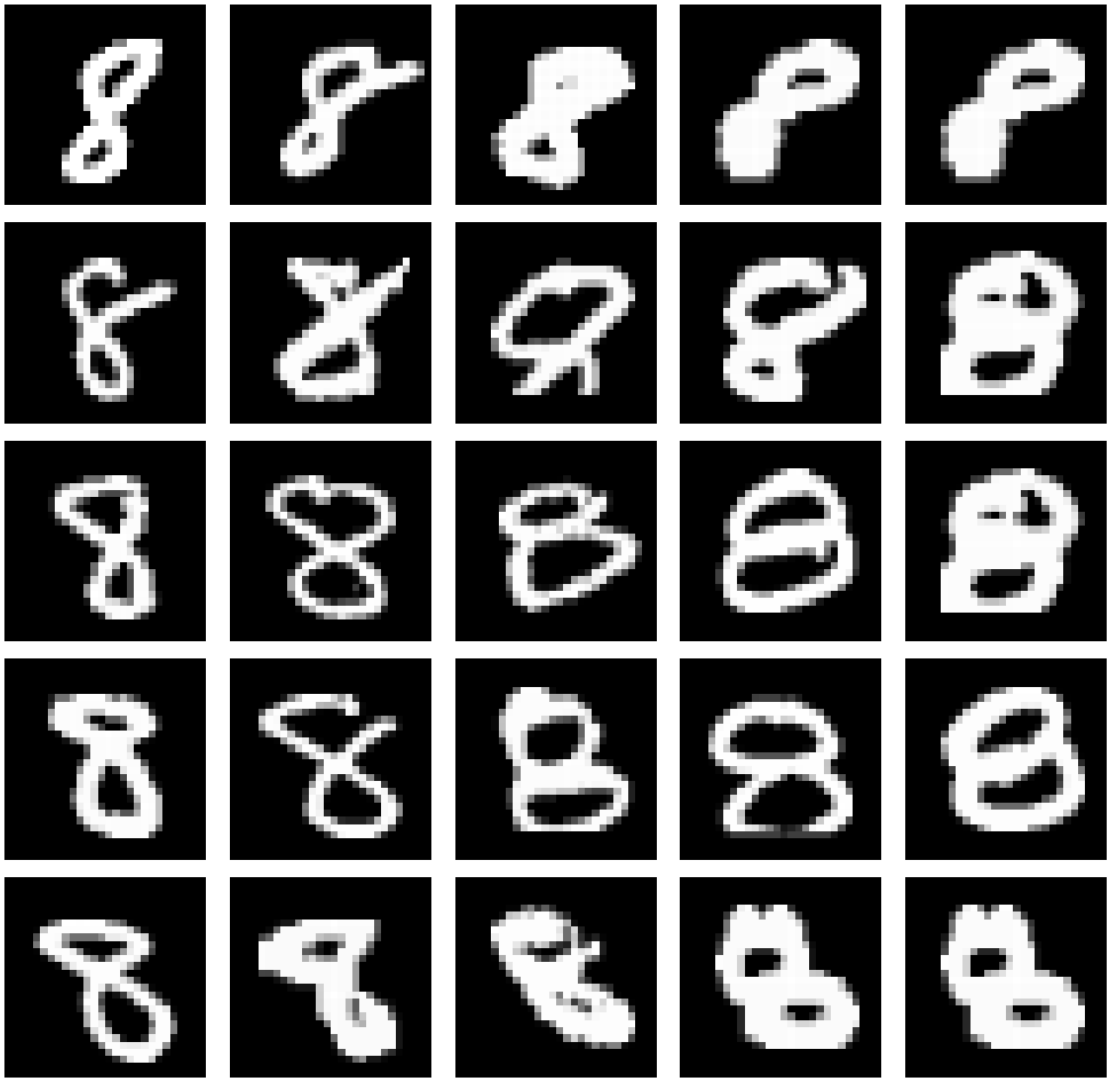
Original pictures for label 6



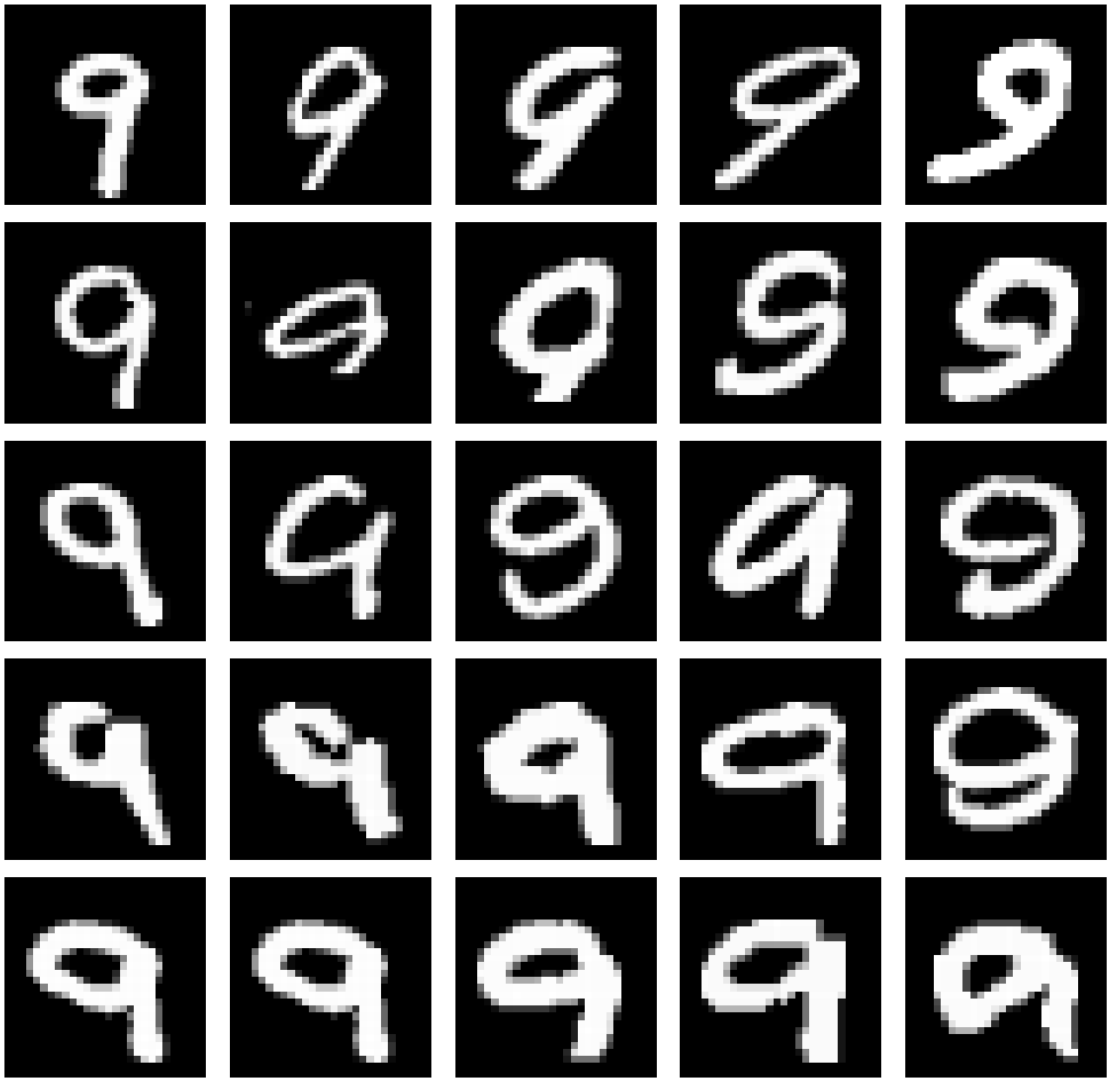
Original pictures for label 7



Original pictures for label 8



Original pictures for label 9



[ ]:

Q3

May 10, 2024

0.1 Mahdi Anvari 610700002 Homework 2 of Machine Learning Question 3

[96]:
```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
from sklearn import metrics
from sklearn.decomposition import PCA
```

a. Perform clustering (with 2 and 3 clusters) on the samples and compare the results to Project and Tissue label in the annotation file.

[97]:
```
AnnotationDF = pd.read_csv("Leaf_Root_annotation.csv")
RawDF = pd.read_csv("Leaf_Root_raw_data.csv")
NormalizedDF = pd.read_csv("Leaf_Root_normalized_data.csv")
#print(AnnotationDF.shape)
#print(RawDF.shape)
#print(NormalizedDF.shape)

Projects = AnnotationDF.values[:,1]
Tissues = AnnotationDF.values[:,4]
print(Tissues)
print(Projects)

RawNP = RawDF.values
RawData = np.transpose(RawNP[:,1:])
NormalizedNP = NormalizedDF.values
NormalizedData = np.transpose(NormalizedNP[:,1:])
```

```
['Leaf' 'Leaf' 'Leaf' 'Leaf' 'Root' 'Root' 'Root' 'Root' 'Leaf' 'Root'
 'Leaf' 'Leaf' 'Root' 'Root' 'Root' 'Root' 'Root' 'Leaf' 'Leaf' 'Leaf'
 'Leaf' 'Leaf' 'Leaf' 'Root' 'Root' 'Root']
['PRJNA493167' 'PRJNA493167' 'PRJNA493167' 'PRJNA493167' 'PRJNA493167'
 'PRJNA493167' 'PRJNA493167' 'PRJNA493167' 'PRJNA661543' 'PRJNA661543'
 'PRJNA661543' 'PRJNA661543' 'PRJNA661543' 'PRJNA661543' 'PRJNA730337'
 'PRJNA730337' 'PRJNA730337' 'PRJNA730337' 'PRJNA730337' 'PRJNA730337'
 'PRJNA730337' 'PRJNA730337' 'PRJNA730337' 'PRJNA730337' 'PRJNA730337'
```

```
 'PRJNA730337']
```

[98]:
```
# for raw data
Kmeans2 = KMeans(n_clusters=2, n_init=10)
Kmeans3 = KMeans(n_clusters=3, n_init=10)
Clusters2 = Kmeans2.fit_predict(RawData)
Clusters3 = Kmeans3.fit_predict(RawData)
print(Clusters2)
print(Clusters3)
ari2 = adjusted_rand_score(Tissues,Clusters2)
ari3 = adjusted_rand_score(Projects,Clusters3)
print("For raw data:")
print("Adjusted Rand Index of Kmeans based on tissues is",ari2)
print("Adjusted Rand Index of Kmeans based on projects is",ari3)
```

```
[0 0 0 0 1 1 1 1 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 1]
[2 0 2 2 2 2 2 2 0 2 0 0 2 2 1 1 1 0 0 0 0 0 0 1 1 1]
For raw data:
Adjusted Rand Index of Kmeans based on tissues is 1.0
Adjusted Rand Index of Kmeans based on projects is 0.30347192456065153
```

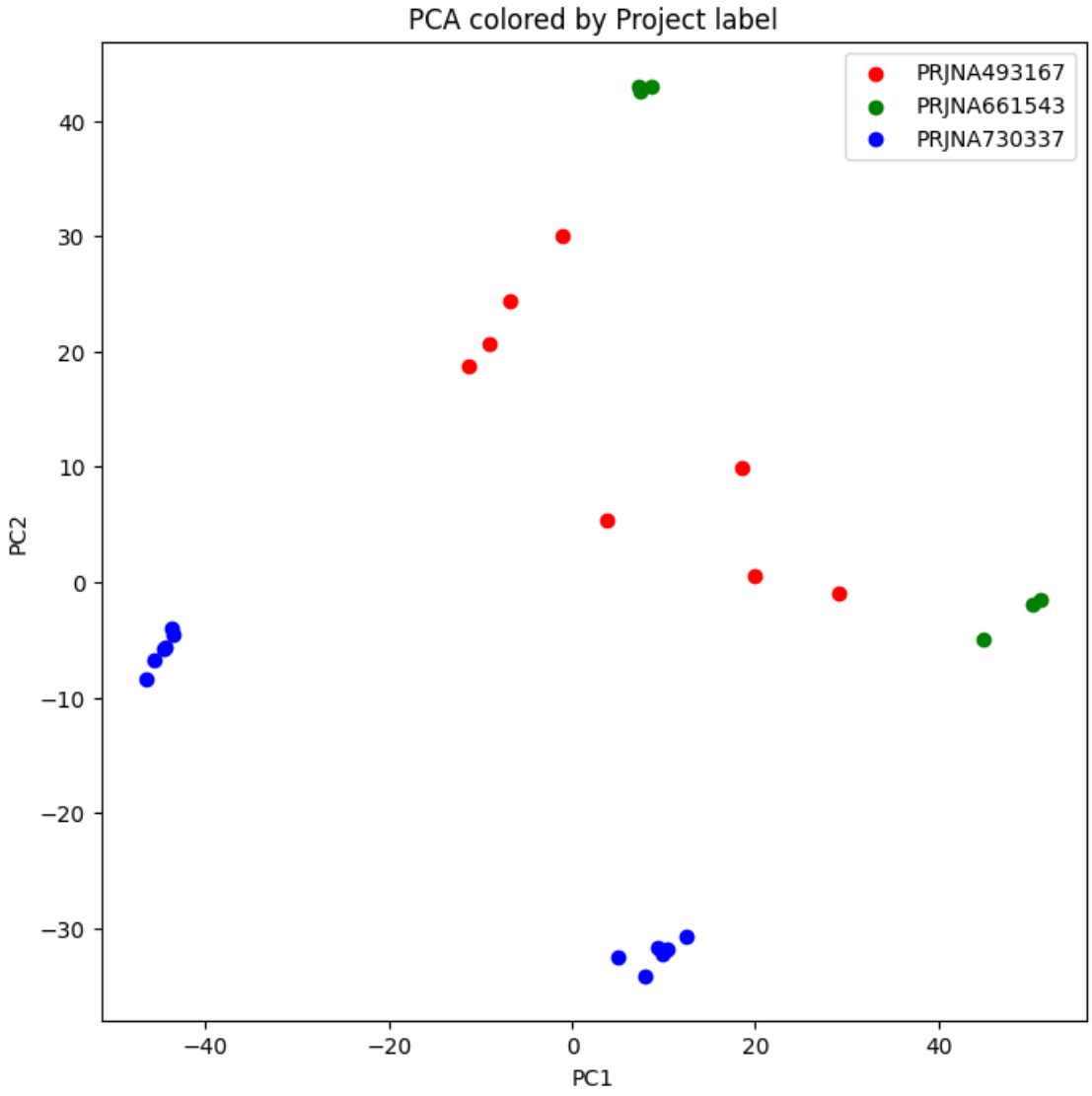
[99]:
```
# for normalized data
Kmeans2 = KMeans(n_clusters=2, n_init=10)
Kmeans3 = KMeans(n_clusters=3, n_init=10)
Clusters2 = Kmeans2.fit_predict(NormalizedData)
Clusters3 = Kmeans3.fit_predict(NormalizedData)
print(Clusters2)
print(Clusters3)
ari2 = adjusted_rand_score(Tissues,Clusters2)
ari3 = adjusted_rand_score(Projects,Clusters3)
print("For normalized data:")
print("Adjusted Rand Index of Kmeans based on tissues is",ari2)
print("Adjusted Rand Indexof Kmeans based on projects is",ari3)
```

```
[1 1 1 1 0 0 0 0 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0]
[0 0 0 0 1 1 1 1 0 1 0 0 1 1 2 2 2 0 0 0 0 0 0 2 2 2]
For normalized data:
Adjusted Rand Index of Kmeans based on tissues is 1.0
Adjusted Rand Indexof Kmeans based on projects is 0.1332969363542826
```

b. Perform PCA, and color the samples once with Project label and once with the Tissue label.

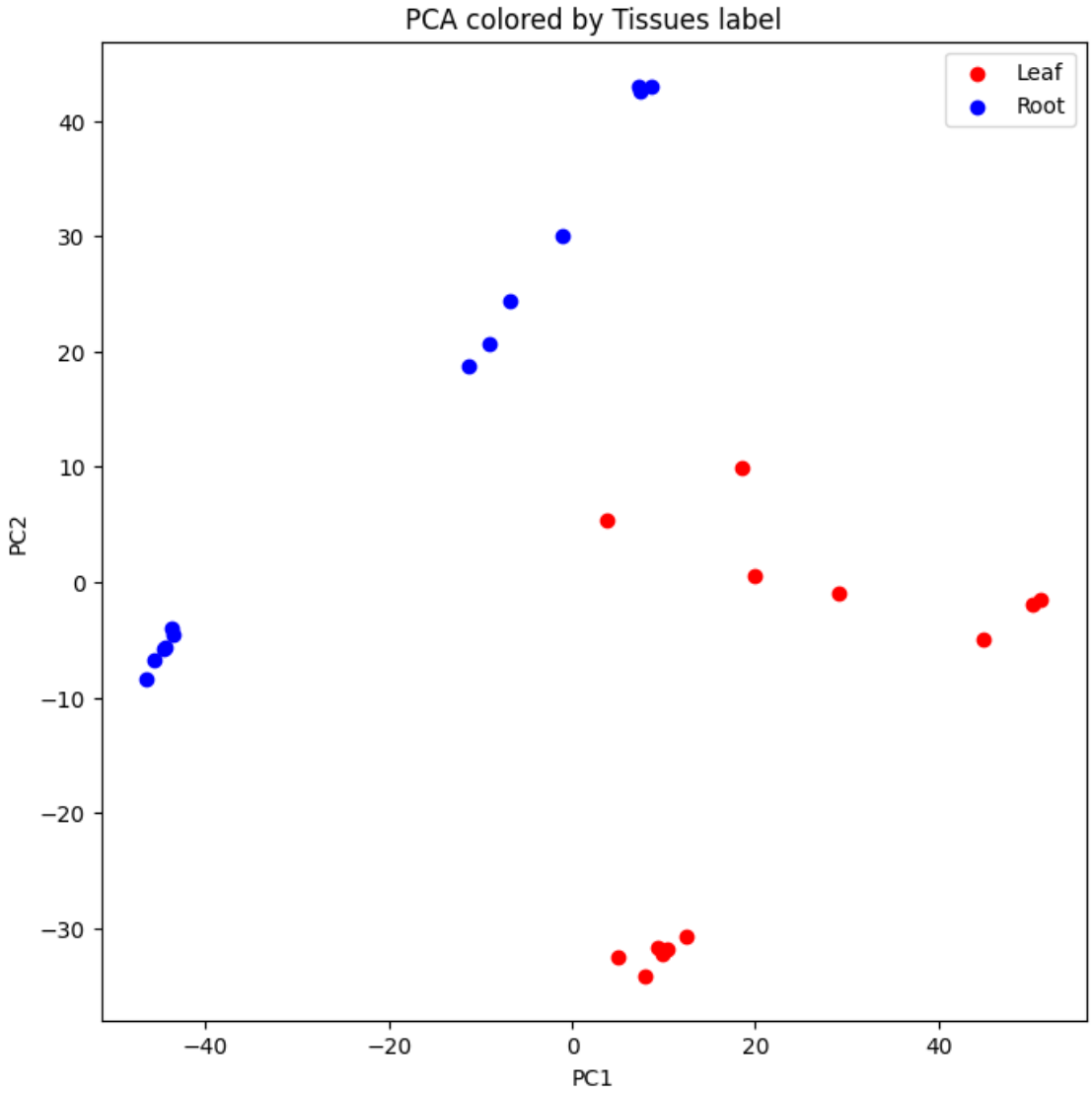
[100]:
```
# for raw data
pca = PCA(n_components=20)
pcaResult = pca.fit_transform(RawData)
#print(pcaResult.shape)

plt.figure(figsize=(8, 8))
for project, color in zip(np.unique(Projects), ['r', 'g', 'b']):
```

```
    idx = AnnotationDF['Project'] == project
    plt.scatter(pcaResult[idx, 0], pcaResult[idx, 1], c=color, label=project)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA colored by Project label')
plt.legend()
plt.show()
```



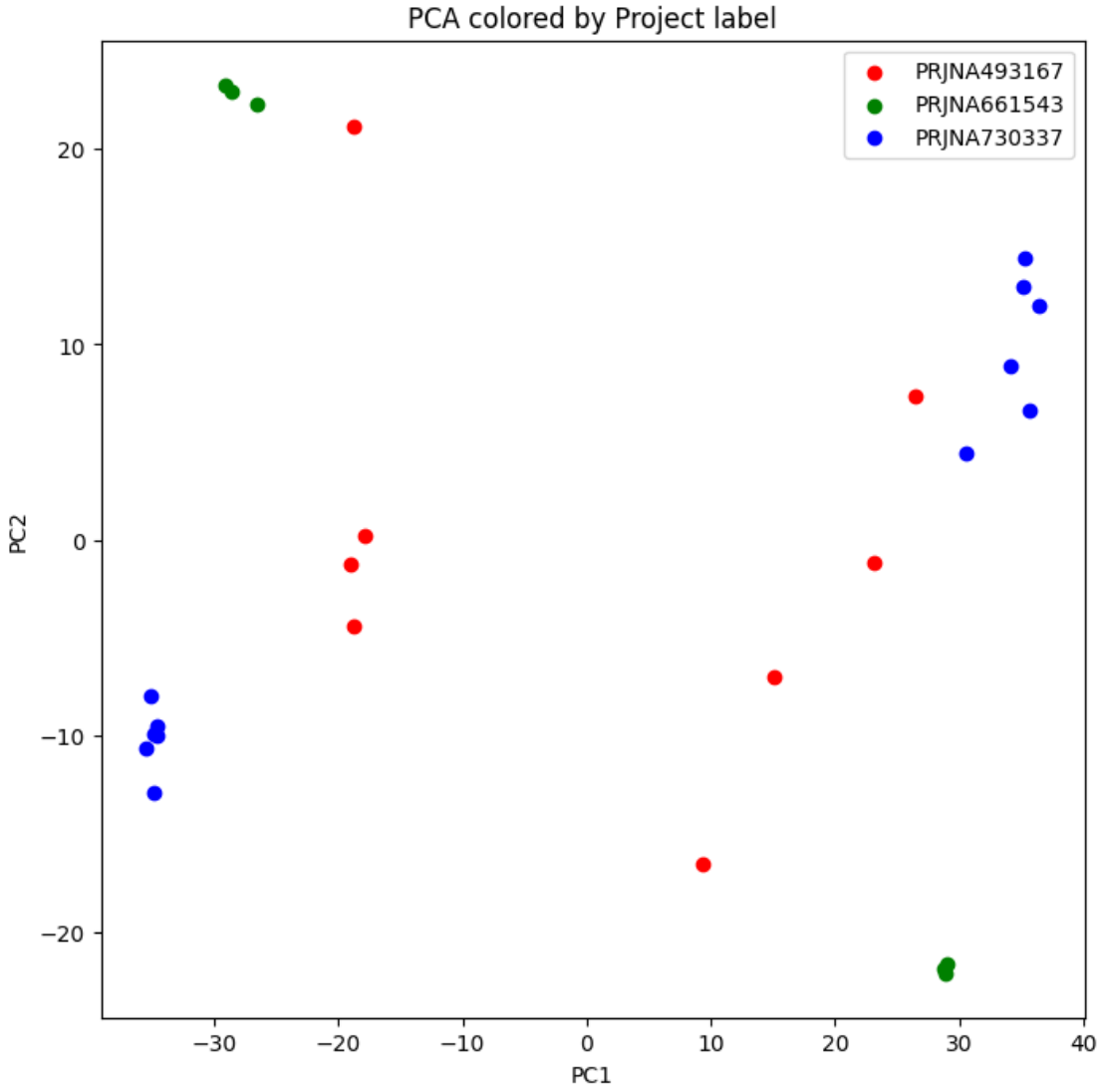
[101]:
```
# for raw data
plt.figure(figsize=(8, 8))
for tissue, color in zip(np.unique(Tissues), ['r', 'b']):
    idx = AnnotationDF['Tissue'] == tissue
```

```
    plt.scatter(pcaResult[idx, 0], pcaResult[idx, 1], c=color, label=tissue)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA colored by Tissues label')
plt.legend()
plt.show()
```



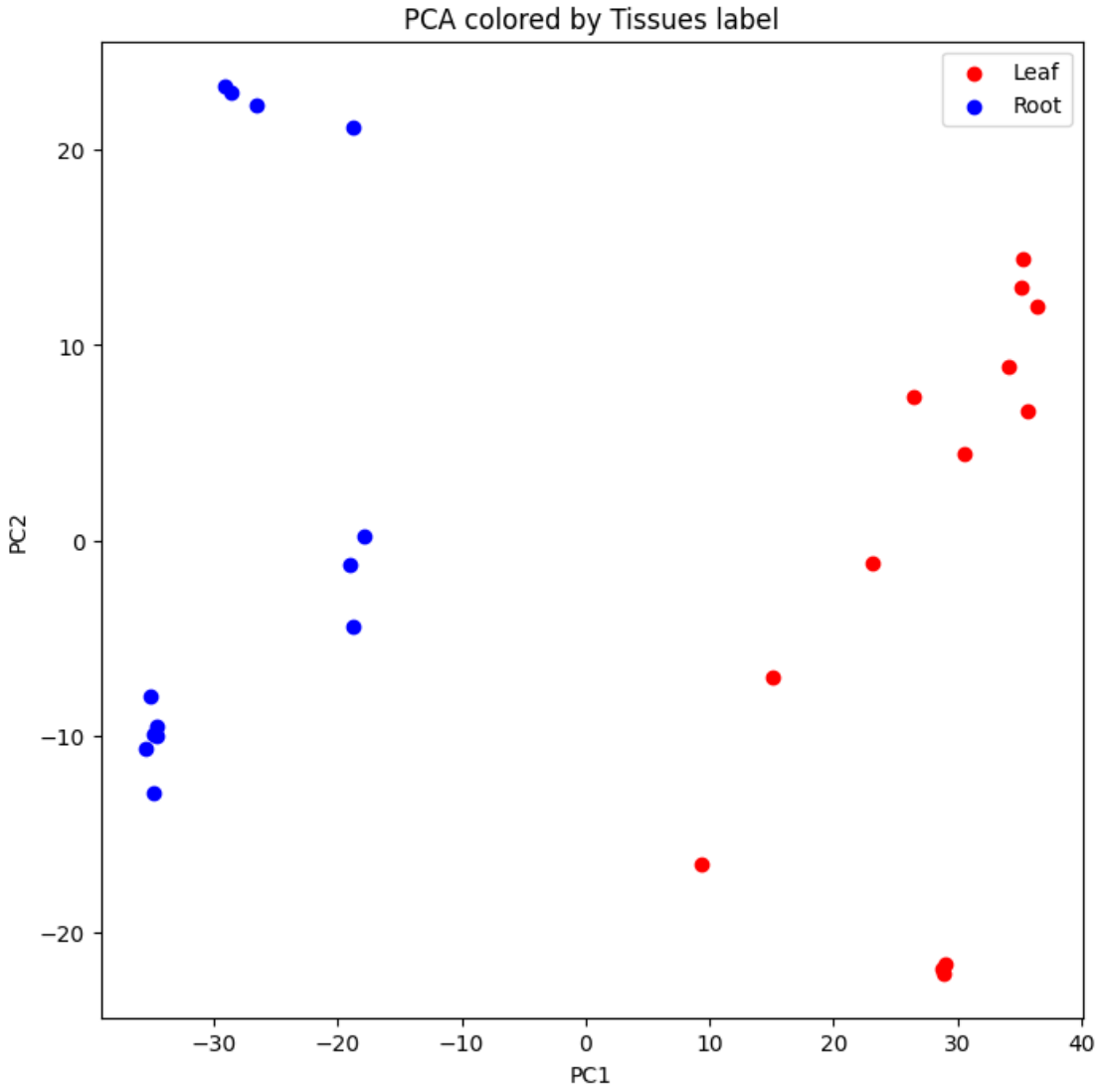
[102]:
```
# for normalized data
pca = PCA(n_components=20)
pcaResult = pca.fit_transform(NormalizedData)
#print(pcaResult.shape)
```

```
plt.figure(figsize=(8, 8))
for project, color in zip(np.unique(Projects), ['r', 'g', 'b']):
    idx = AnnotationDF['Project'] == project
    plt.scatter(pcaResult[idx, 0], pcaResult[idx, 1], c=color, label=project)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA colored by Project label')
plt.legend()
plt.show()
```



[103]:
```
# for normalized data
plt.figure(figsize=(8, 8))
for tissue, color in zip(np.unique(Tissues), ['r', 'b']):
```

```
    idx = AnnotationDF['Tissue'] == tissue
    plt.scatter(pcaResult[idx, 0], pcaResult[idx, 1], c=color, label=tissue)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA colored by Tissues label')
plt.legend()
plt.show()
```



[ ]:

Q4

May 12, 2024

0.1 Mahdi Anvari 610700002 Homework 2 of Machine Learning Question 4

[2]:
```
# importing libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sbn
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D
```

```
WARNING:tensorflow:From
C:\Users\M\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\losses.py:2664: The name tf.losses.sparse_softmax_cross_entropy
is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy
instead.
```

- Load MNIST dataset (could be accessed using from keras.datasets in python)

[12]:
```
mnist_path = 'c:/Users/M/Downloads/mnist.npz'
with np.load(mnist_path, allow_pickle=True) as f:
    X_train, Y_train = f['x_train'], f['y_train']
    X_test, Y_test = f['x_test'], f['y_test']
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

[13]:
```
X_train = X_train.reshape(X_train.shape[0],-1)
X_test = X_test.reshape(X_test.shape[0],-1)
```

```
print(X_train.shape)
print(X_test.shape)

scaler = StandardScaler()
NormalizedXtrain = scaler.fit_transform(X_train)
NormalizedXtest = scaler.fit_transform(X_test)
```

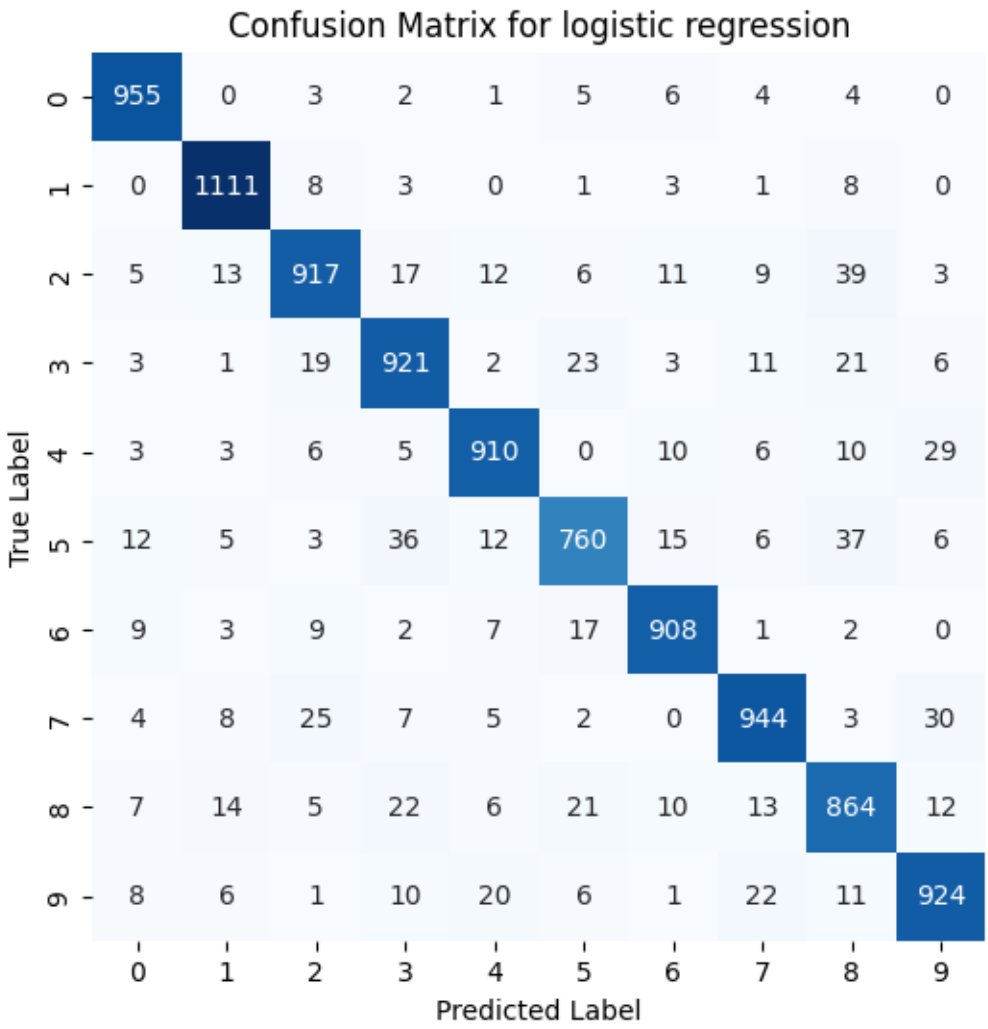
```
(60000, 784)
(10000, 784)
```

a. Logistic Regression

[23]:
```
LogReg = LogisticRegression(max_iter=1000)
LogReg.fit(X_train, Y_train)
Y_pred = LogReg.predict(X_test)
LogAccuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy for logistic regression:", LogAccuracy)
```

```
Accuracy for logistic regression: 0.9214

C:\Users\M\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[29]:
```
LogCM = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(6, 6))
sbn.heatmap(LogCM, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for logistic regression')
plt.show()
```



b. MLP with one hidden layer of size 128

[52]:
```
modelB = Sequential(name='modelB')
modelB.add(Dense(units=128, activation='relu', input_shape=(784,)␣
 ↪,name='hidden_layer'))
modelB.add(Dense(units=10, activation='softmax', name='output_layer'))
modelB.compile(optimizer=tf.optimizers.Adam(),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
modelB.summary()
```

```
Model: "modelB"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 hidden_layer (Dense)         (None, 128)               100480
```

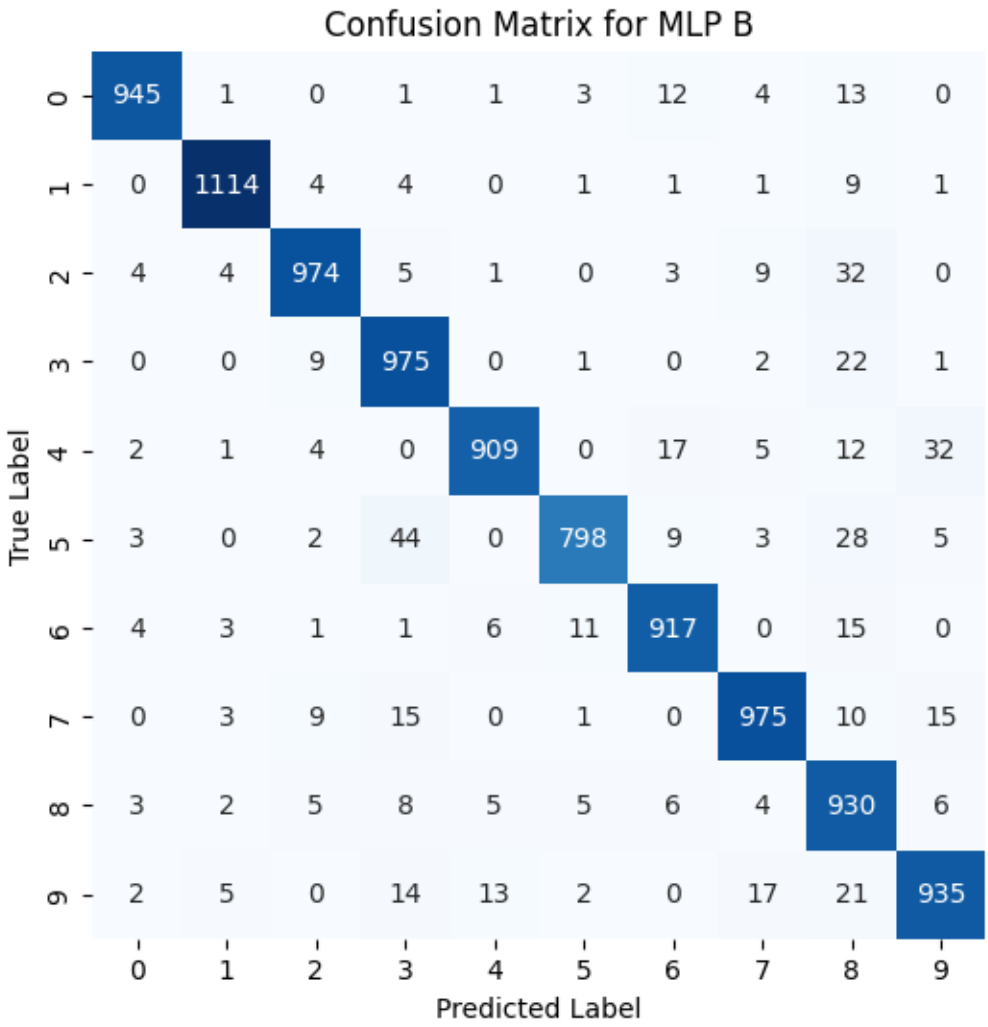
```
 output_layer (Dense)        (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

[55]:
```
modelB.fit(X_train, Y_train, epochs=10, batch_size=32, validation_split=0.2)
loss, accuracy = modelB.evaluate(X_test, Y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1760 -
accuracy: 0.9571 - val_loss: 0.2726 - val_accuracy: 0.9454
Epoch 2/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1693 -
accuracy: 0.9594 - val_loss: 0.3052 - val_accuracy: 0.9431
Epoch 3/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1688 -
accuracy: 0.9591 - val_loss: 0.2883 - val_accuracy: 0.9471
Epoch 4/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1683 -
accuracy: 0.9614 - val_loss: 0.2503 - val_accuracy: 0.9507
Epoch 5/10
1500/1500 [==============================] - 7s 4ms/step - loss: 0.1550 -
accuracy: 0.9623 - val_loss: 0.2626 - val_accuracy: 0.9504
Epoch 6/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1514 -
accuracy: 0.9624 - val_loss: 0.3094 - val_accuracy: 0.9448
Epoch 7/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1561 -
accuracy: 0.9629 - val_loss: 0.3498 - val_accuracy: 0.9437
Epoch 8/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1581 -
accuracy: 0.9639 - val_loss: 0.3044 - val_accuracy: 0.9467
Epoch 9/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1450 -
accuracy: 0.9649 - val_loss: 0.3202 - val_accuracy: 0.9392
Epoch 10/10
1500/1500 [==============================] - 6s 4ms/step - loss: 0.1398 -
accuracy: 0.9662 - val_loss: 0.3565 - val_accuracy: 0.9493
313/313 [==============================] - 1s 2ms/step - loss: 0.3818 -
accuracy: 0.9472
Test Loss: 0.3817773759365082
Test Accuracy: 0.9472000002861023
```

[56]:
```
Y_pred = np.argmax(modelB.predict(X_test), axis=-1)
B_CM = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(6, 6))
sbn.heatmap(B_CM, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for MLP B')
plt.show()
```

```
313/313 [==============================] - 1s 2ms/step
```



c. MLP with two hidden layers of sizes 256 and 128.

[57]:
```
modelC = Sequential(name='modelC')
modelC.add(Dense(units=256, activation='relu', input_shape=(784,)␣
↪,name='hidden_layer1'))
```

```
modelC.add(Dense(units=128, activation='relu' ,name='hidden_layer2'))
modelC.add(Dense(units=10, activation='softmax', name='output_layer'))
modelC.compile(optimizer=tf.optimizers.Adam(),␣
  ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
modelC.summary()
```

```
Model: "modelC"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 hidden_layer1 (Dense)       (None, 256)               200960

 hidden_layer2 (Dense)       (None, 128)               32896

 output_layer (Dense)        (None, 10)                1290

=================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
_________________________________________________________________
```

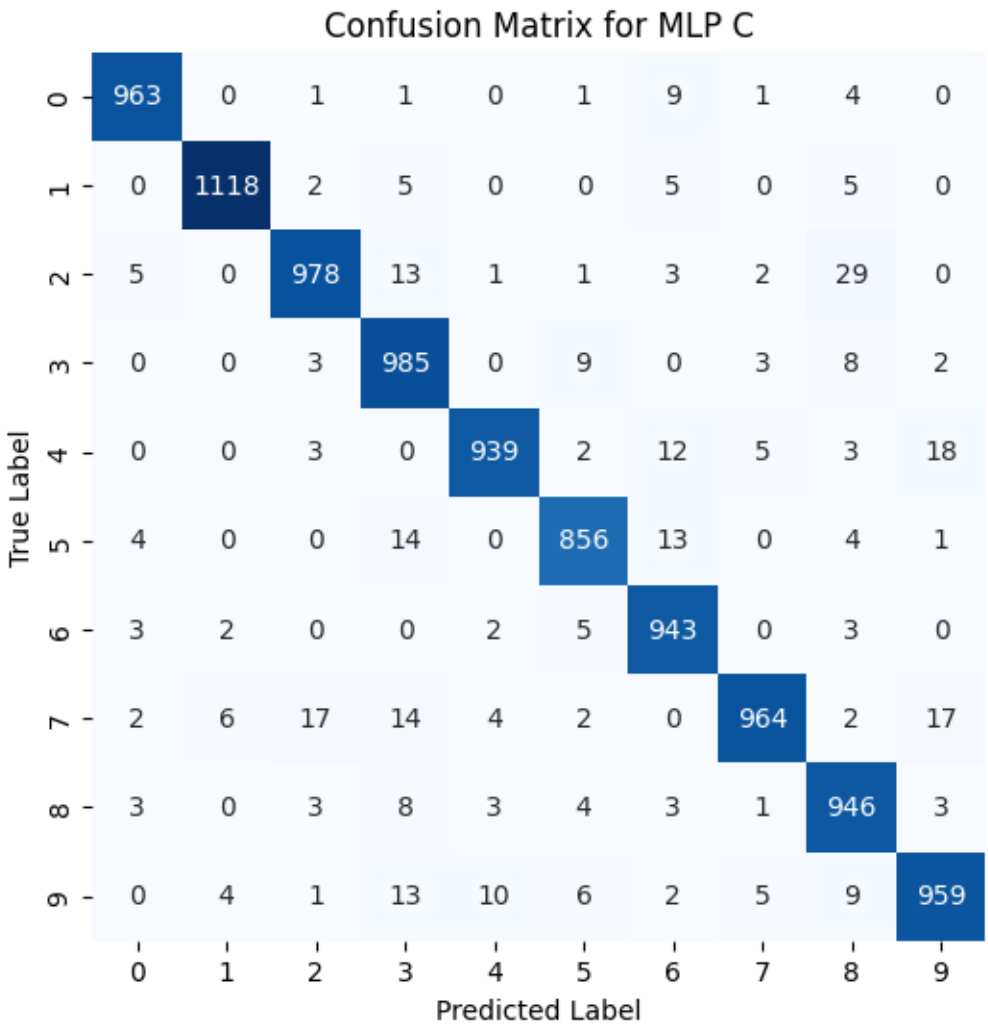
[58]:
```
modelC.fit(X_train, Y_train, epochs=10, batch_size=32, validation_split=0.2)
loss, accuracy = modelC.evaluate(X_test, Y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
1500/1500 [==============================] - 8s 5ms/step - loss: 1.9098 -
accuracy: 0.8776 - val_loss: 0.3792 - val_accuracy: 0.9073
Epoch 2/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.2613 -
accuracy: 0.9338 - val_loss: 0.2744 - val_accuracy: 0.9303
Epoch 3/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.1915 -
accuracy: 0.9483 - val_loss: 0.2091 - val_accuracy: 0.9453
Epoch 4/10
1500/1500 [==============================] - 8s 6ms/step - loss: 0.1589 -
accuracy: 0.9565 - val_loss: 0.1656 - val_accuracy: 0.9578
Epoch 5/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1468 -
accuracy: 0.9595 - val_loss: 0.1998 - val_accuracy: 0.9540
Epoch 6/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.1249 -
accuracy: 0.9657 - val_loss: 0.1580 - val_accuracy: 0.9590
Epoch 7/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.1206 -
accuracy: 0.9674 - val_loss: 0.1565 - val_accuracy: 0.9606
```

```
Epoch 8/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.1074 -
accuracy: 0.9711 - val_loss: 0.1549 - val_accuracy: 0.9647
Epoch 9/10
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0937 -
accuracy: 0.9754 - val_loss: 0.1802 - val_accuracy: 0.9617
Epoch 10/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0929 -
accuracy: 0.9766 - val_loss: 0.1472 - val_accuracy: 0.9653
313/313 [==============================] - 1s 3ms/step - loss: 0.1425 -
accuracy: 0.9651
Test Loss: 0.14246425032615662
Test Accuracy: 0.9650999903678894
```

[59]:
```
Y_pred = np.argmax(modelC.predict(X_test), axis=-1)
C_CM = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(6, 6))
sbn.heatmap(C_CM, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for MLP C')
plt.show()
```

```
313/313 [==============================] - 1s 3ms/step
```



d. CNN with two "convolution + max pooling" blocks and a dense network with one hidden layer of size 128.

[11]:
```
modelD = Sequential(name='modelD')
modelD.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu',␣
 ↪input_shape=(28,28,1), name='CNN1'))
modelD.add(MaxPooling2D(pool_size=(2,2), name='Pooling1'))
modelD.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu',␣
 ↪name='CNN2'))
modelD.add(MaxPooling2D(pool_size=(2,2), name='Pooling2'))
modelD.add(Flatten())
modelD.add(Dense(units=128, activation='relu' ,name='hidden_layer'))
modelD.add(Dense(units=10, activation='softmax', name='output_layer'))
modelD.compile(optimizer=tf.optimizers.Adam(),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
modelD.summary()
```

```
Model: "modelD"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 CNN1 (Conv2D)               (None, 26, 26, 32)        320

 Pooling1 (MaxPooling2D)     (None, 13, 13, 32)        0

 CNN2 (Conv2D)               (None, 11, 11, 64)        18496

 Pooling2 (MaxPooling2D)     (None, 5, 5, 64)          0

 flatten (Flatten)           (None, 1600)              0

 hidden_layer (Dense)        (None, 128)               204928

 output_layer (Dense)        (None, 10)                1290

=================================================================
Total params: 225,034
Trainable params: 225,034
Non-trainable params: 0
_________________________________________________________________
```

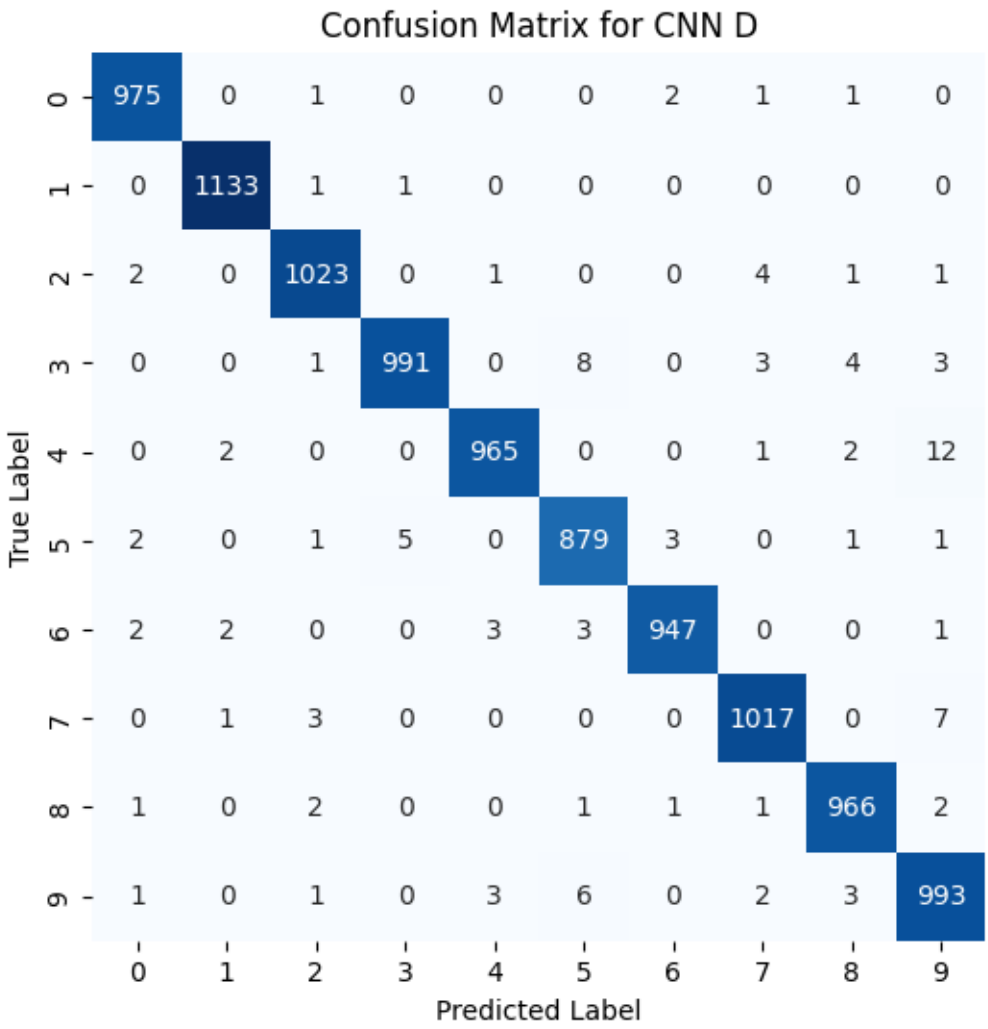
[15]:
```
ReshapedNormalizedXtrain = NormalizedXtrain.reshape(NormalizedXtrain.
 ↪shape[0],28,28,1)
ReshapedNormalizedXtest = NormalizedXtest.reshape(NormalizedXtest.
 ↪shape[0],28,28,1)

modelD.fit(ReshapedNormalizedXtrain, Y_train, epochs=10, batch_size=32,␣
 ↪validation_split=0.2)
loss, accuracy = modelD.evaluate(ReshapedNormalizedXtest, Y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
WARNING:tensorflow:From
C:\Users\M\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\utils\tf_utils.py:490: The name tf.ragged.RaggedTensorValue is
deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From
C:\Users\M\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\engine\base_layer_utils.py:380: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.
```

```
1500/1500 [==============================] - 20s 12ms/step - loss: 0.1380 -
accuracy: 0.9576 - val_loss: 0.0637 - val_accuracy: 0.9817
Epoch 2/10
1500/1500 [==============================] - 19s 13ms/step - loss: 0.0452 -
accuracy: 0.9852 - val_loss: 0.0451 - val_accuracy: 0.9879
Epoch 3/10
1500/1500 [==============================] - 17s 11ms/step - loss: 0.0334 -
accuracy: 0.9894 - val_loss: 0.0396 - val_accuracy: 0.9887
Epoch 4/10
1500/1500 [==============================] - 16s 10ms/step - loss: 0.0220 -
accuracy: 0.9926 - val_loss: 0.0430 - val_accuracy: 0.9873
Epoch 5/10
1500/1500 [==============================] - 16s 11ms/step - loss: 0.0171 -
accuracy: 0.9944 - val_loss: 0.0439 - val_accuracy: 0.9888
Epoch 6/10
1500/1500 [==============================] - 18s 12ms/step - loss: 0.0164 -
accuracy: 0.9947 - val_loss: 0.0483 - val_accuracy: 0.9877
Epoch 7/10
1500/1500 [==============================] - 19s 13ms/step - loss: 0.0121 -
accuracy: 0.9963 - val_loss: 0.0579 - val_accuracy: 0.9868
Epoch 8/10
1500/1500 [==============================] - 19s 13ms/step - loss: 0.0091 -
accuracy: 0.9969 - val_loss: 0.0467 - val_accuracy: 0.9897
Epoch 9/10
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0078 -
accuracy: 0.9974 - val_loss: 0.0527 - val_accuracy: 0.9893
Epoch 10/10
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0078 -
accuracy: 0.9976 - val_loss: 0.0599 - val_accuracy: 0.9894
313/313 [==============================] - 2s 6ms/step - loss: 0.0506 -
accuracy: 0.9889
Test Loss: 0.050574351102113724
Test Accuracy: 0.9889000058174133
```

[16]:
```
Y_pred = np.argmax(modelD.predict(ReshapedNormalizedXtest), axis=-1)
D_CM = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(6, 6))
sbn.heatmap(D_CM, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for CNN D')
plt.show()
```

```
313/313 [==============================] - 2s 5ms/step
```


Confusion Matrix for CNN D
True Label
Predicted Label
0 975 0 1 0 0 0 2 1 1 0
1 0 1133 1 1 0 0 0 0 0 0
2 2 0 1023 0 1 0 0 4 1 1
3 0 0 1 991 0 8 0 3 4 3
4 0 2 0 0 965 0 0 1 2 12
5 2 0 1 5 0 879 3 0 1 1
6 2 2 0 0 3 3 947 0 0 1
7 0 1 3 0 0 0 0 1017 0 7
8 1 0 2 0 0 1 1 1 966 2
9 1 0 1 0 3 6 0 2 3 993
0 1 2 3 4 5 6 7 8 9

[ ]: