```
In [ ]:
```

Applied Problems

Question 1

Generate 50 observations of data in three distinct classes (150 obser- vations total). For observations in Class 1, the 50 variables will be drawn independently from a N (−1, 1) distribution. For observations in Class 2, the 50 variables will be drawn independently from a N (0, 1) distribution. For observations in Class 3, the 50 variables will be drawn indepen- dently from a N (1, 1) distribution. Your final dataset will have 150 total observations and 51 variables (1 variable for the class label, and 50 simulated normal variables).

```
In [1]: import numpy as np
```

```
In [11]: # Generate 50 observations in each of three classes
class1 = np.random.normal(-1, 1, size=(50, 50))
class2 = np.random.normal(0, 1, size=(50, 50))
class3 = np.random.normal(1, 1, size=(50, 50))
classes = np.array([1, 2, 3]).repeat(50, axis=0)
# Combine the three classes into a single dataset
data = np.concatenate([class1, class2, class3], axis=0)

df = np.column_stack((classes, data))
```

```
In [29]: X = df[:,1:]
y = df[:,0]
```

```
In [31]: print(X.shape)
print(y.shape)
```

```
(150, 50)
(150,)
```

(a) Perform PCA on the 150 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. Discuss whether PCA appears to have done a good job separating the three classes.

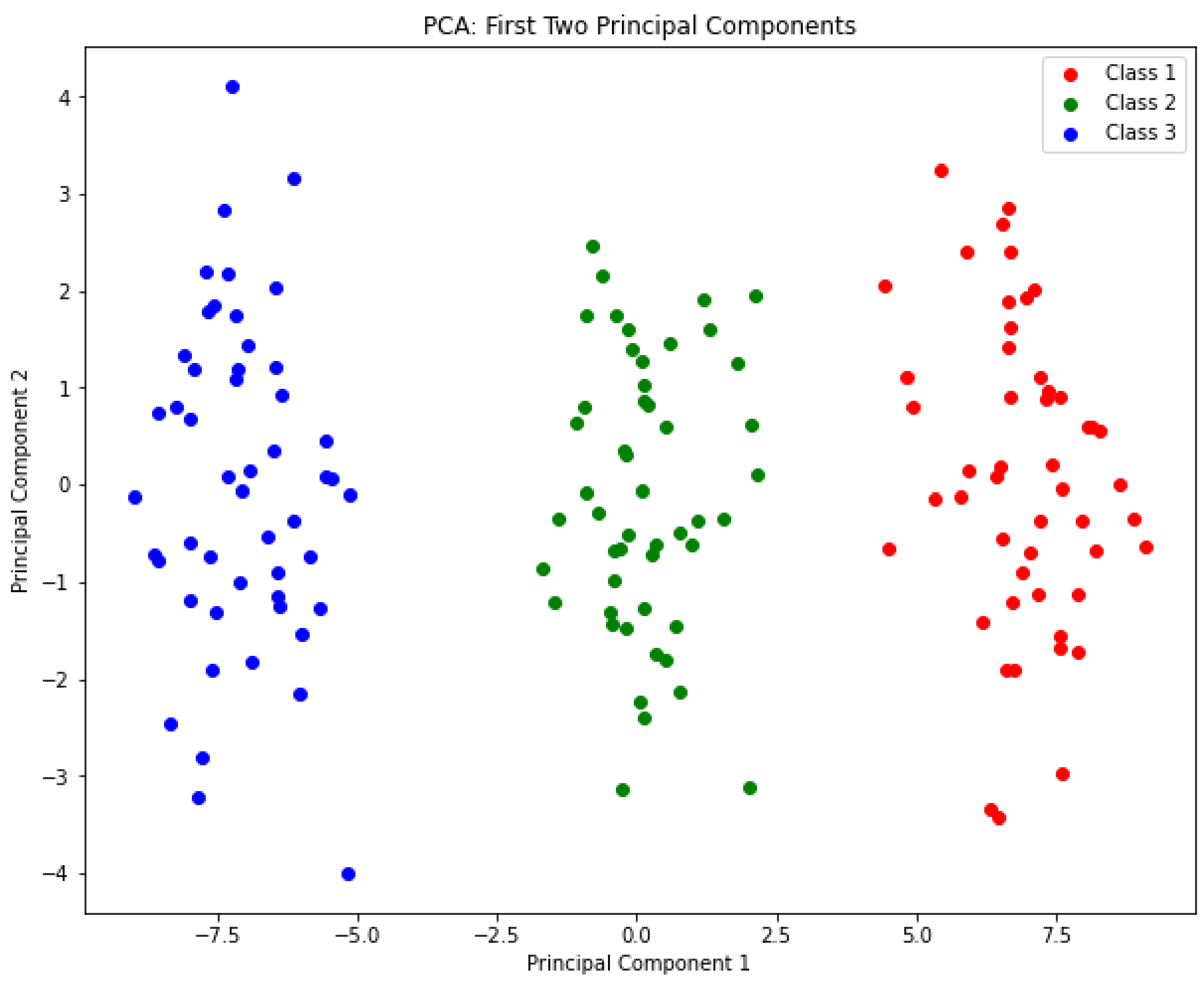
```
In [43]: import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from sklearn.decomposition import PCA
```

```
In [33]: class_color = {1: 'r', 2: 'g', 3: 'b'}
```

```
In [34]: pca = PCA(n_components=2)
```

```
In [39]: principal_components = pca.fit_transform(X)
```

```
In [53]: plt.figure(figsize=(10, 8))
for label in np.unique(classes):
    indices = classes == label
    plt.scatter(principal_components[indices, 0],
                principal_components[indices, 1],
                color=class_color[label],
                label=f"Class {int(label)}")

plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA: First Two Principal Components")
plt.legend()
plt.show()
```



Observation:

It appears as though the 2 principal components have captured sufficient variance in the data to satisfactorily separate the three classes. We see that there is noticeable 'space' between the 3 classes when represented along the 2 principal components

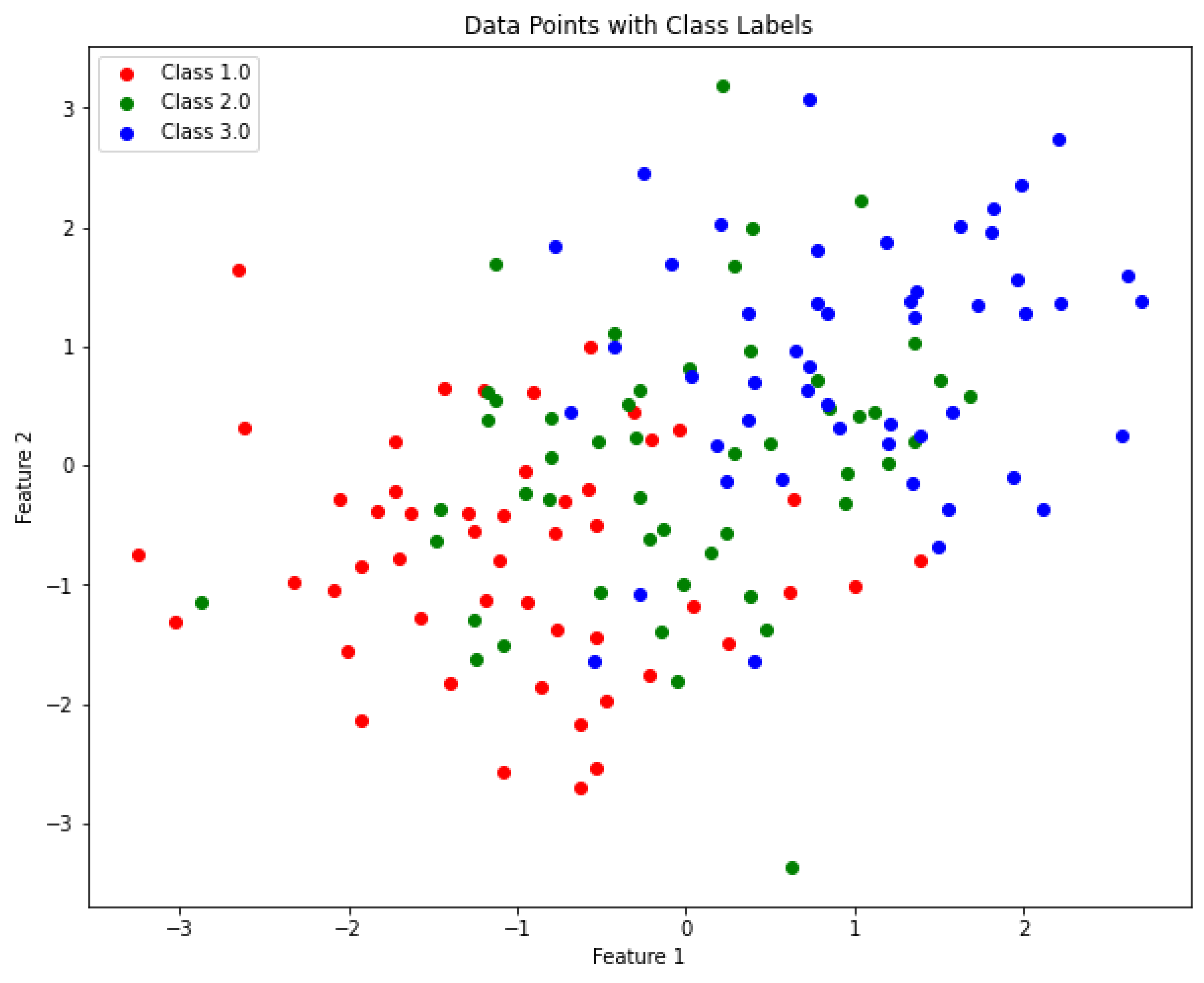
Write your own K-means clustering function that has two arguments (K, data) and returns class labels for each observation in the input dataset. Use your function to perform K-means clustering of the observations with K = 3. How well do the clusters that you obtained in K-means clustering compare to the true class labels?

In [68]:
```
# K-Means
def k_means_clustering(K, data):
    # Randomly initialize cluster centroids
    centroids = data[np.random.choice(range(data.shape[0]), size=K)]

    prev_centroids = np.zeros_like(centroids)
    labels = np.zeros(data.shape[0])

    while not np.array_equal(prev_centroids, centroids):
        prev_centroids = centroids.copy()

        # Assign data points to the nearest centroid
        for i in range(data.shape[0]):
            distances = np.linalg.norm(data[i] - centroids, axis=1)
            labels[i] = np.argmin(distances)

        # Update centroids by calculating the mean of each cluster
        for k in range(K):
            cluster_points = data[labels == k]
            centroids[k] = np.mean(cluster_points, axis=0)
    labels += 1
    return labels
```

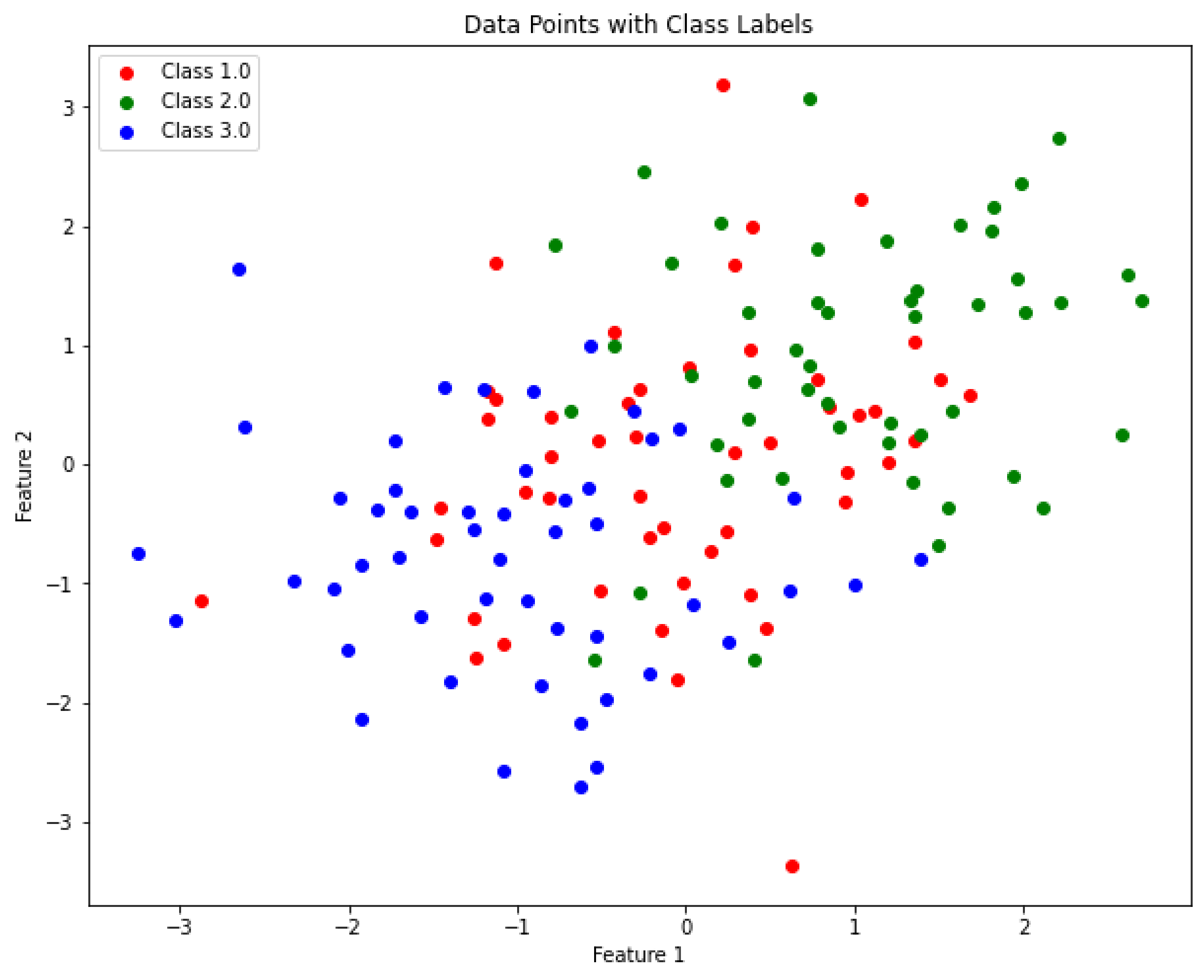
In [87]:
```
def plot_clusters(results):
    class_colors = {1: 'r', 2: 'g', 3: 'b', 4: 'y'}

    plt.figure(figsize=(10, 8))
    for label in np.unique(results):
        indices = results == label
        color = class_colors[label]

        plt.scatter(data[indices, 0], data[indices, 1], color=color, label=f"Class {label}")

    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
    plt.title("Data represented by 2 principal components")
    plt.legend()
    plt.show()
```

In [88]:
```
# For K = 3
test_clusters = k_means_clustering(3, X)
```

In [89]:
```
plot_clusters(y)
```



In [90]:
```
plot_clusters(test_clusters)
```



In [91]:
```
y
```

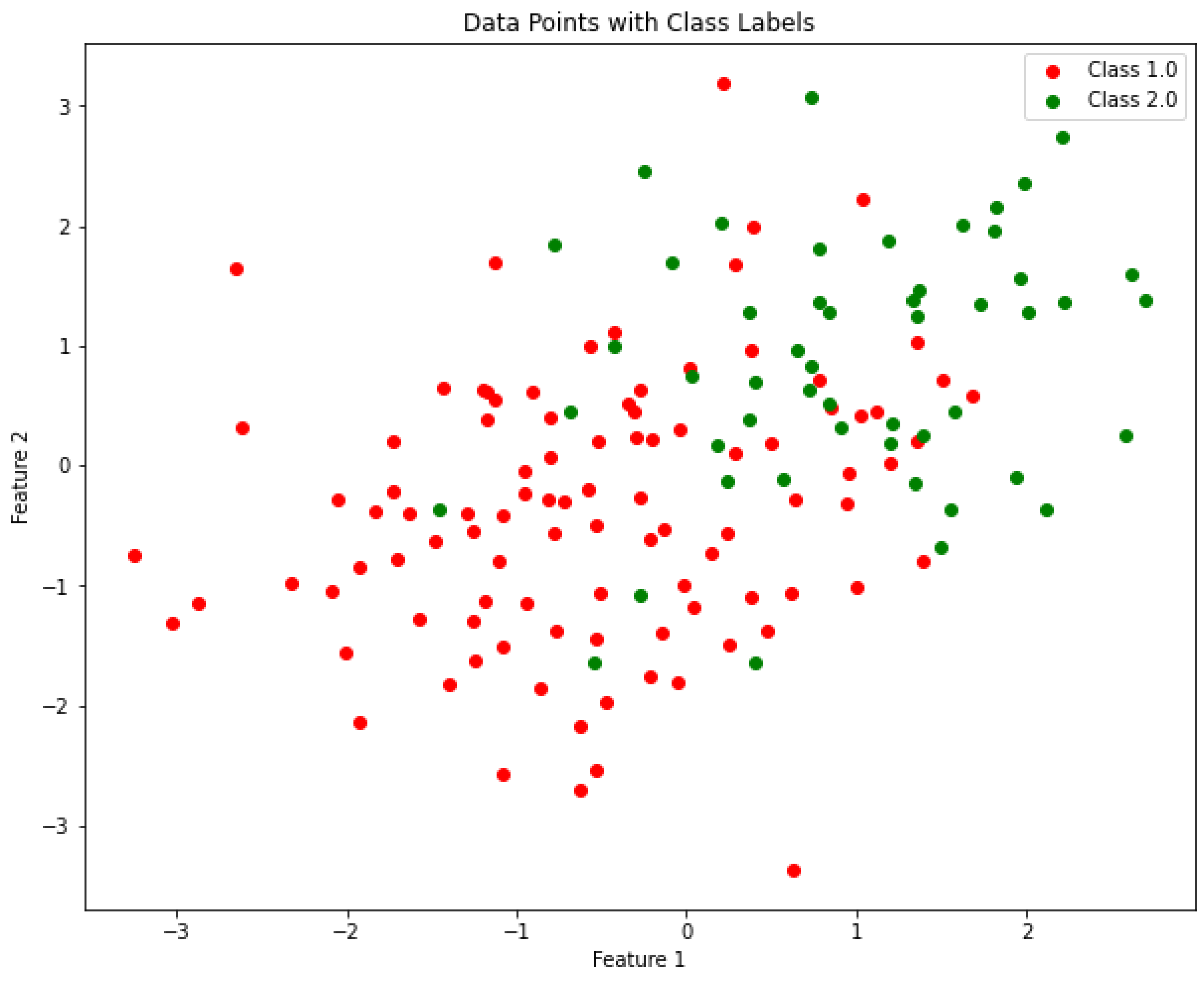
Out[91]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2.,
       2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 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., 3., 3.,
       3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
       3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
       3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.])
```

Observation

It is observed that my implementation is able to exactly separate the datapoints from all three classes.

In [92]:
```
# K = 2
test_clusters_2 = k_means_clustering(2, X)
```

In [93]:
```
plot_clusters(test_clusters_2)
```



```
In [74]: y
```

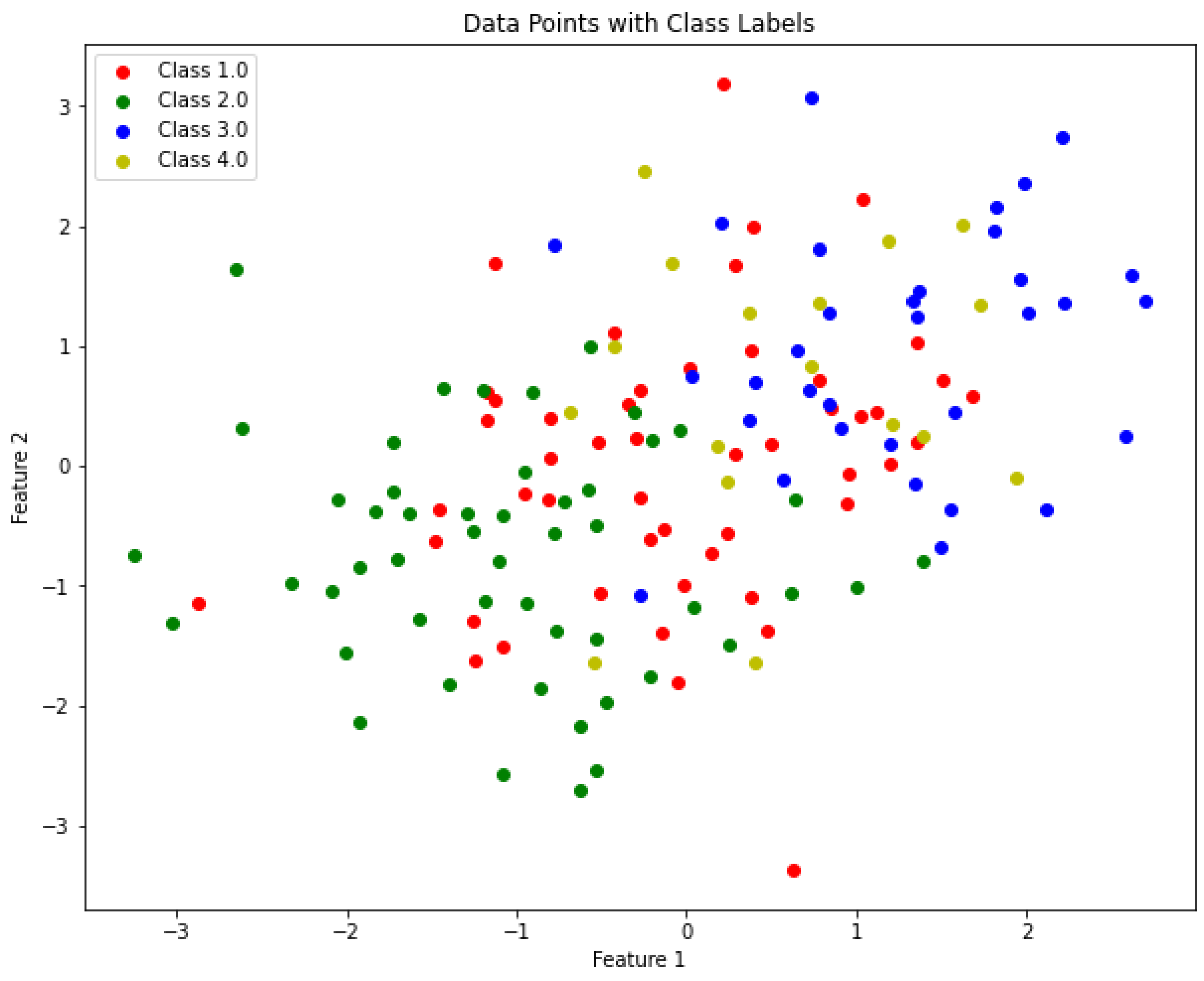
```
Out[74]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
                1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
                1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 2.,
                2., 2., 2., 2., 2., 2., 2., 2., 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., 3., 3.,
                3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
                3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
                3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.])
```

```
In [94]: # K = 4
         test_clusters_4 = k_means_clustering(4, X)
```

```
In [95]: test_clusters_4
```

```
Out[95]: array([2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2., 2.,
                2., 2., 2., 2., 2., 2., 2., 2., 2., 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., 3., 3.,
                3., 3., 4., 3., 4., 3., 4., 3., 3., 3., 3., 3., 4., 3., 3., 3., 3.,
                4., 3., 4., 3., 4., 4., 3., 4., 3., 3., 3., 4., 3., 3., 4., 4., 3.,
                3., 4., 3., 3., 3., 4., 3., 4., 4., 3., 3., 3., 3., 4.])
```

```
In [96]: plot_clusters(test_clusters_4)
```



In [97]:
```python
from scipy.spatial.distance import cdist

def hierarchical_clustering(K, data):
    # Calculate pairwise distances using Euclidean distance
    distances = cdist(data, data, metric='euclidean')

    # Initialize each data point as a separate cluster
    clusters = [[i] for i in range(len(data))]

    while len(clusters) > K:
        # Find the indices of the two clusters with the smallest average distance
        min_dist = np.inf
        merge_indices = None

        for i in range(len(clusters)):
            for j in range(i+1, len(clusters)):
                # Calculate the average distance between the two clusters
                dist = np.mean(distances[clusters[i], :][:, clusters[j]])

                if dist < min_dist:
                    min_dist = dist
                    merge_indices = (i, j)

        # Merge the two clusters with the smallest average distance
        i, j = merge_indices
        clusters[i].extend(clusters[j])
        del clusters[j]

    # Assign class labels to each observation
    labels = np.zeros(len(data), dtype=int)
    for i, cluster in enumerate(clusters):
        labels[cluster] = i + 1

    return labels
```

In [98]:
```python
h_clusters_3 = hierarchical_clustering(3, X)
```

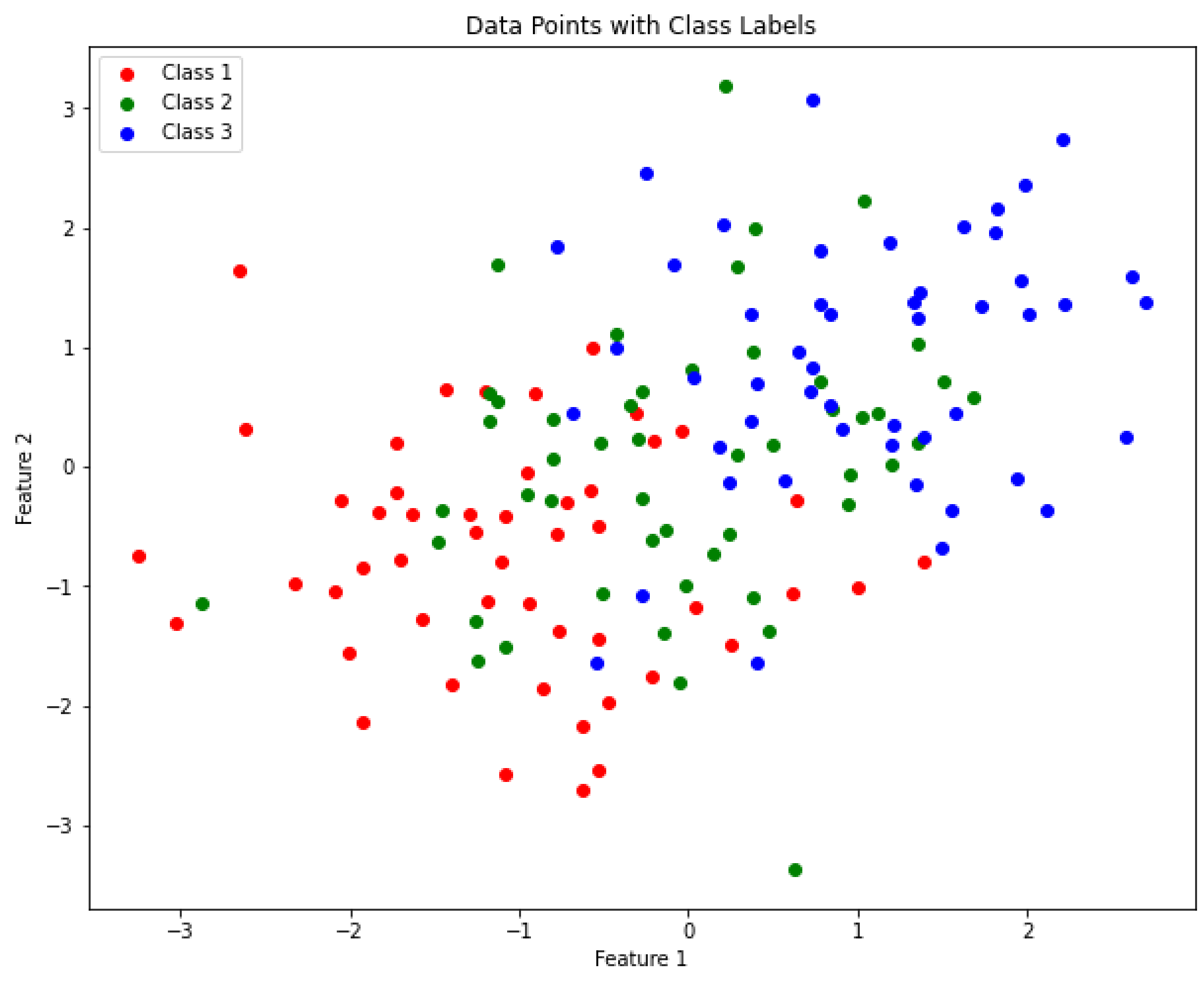
In [99]:
```python
h_clusters_3
```

Out[99]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 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, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3])
```

In [100…
```python
y
```

Out[100]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 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., 3., 3.,
       3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
       3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.,
       3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3., 3.])
```

In [101…
```python
plot_clusters(h_clusters_3)
```



Observation

Problem 2

In question section, we explore the methods PCA, K-means, and SVM’s for unsupervised/supervised learning on the MNIST dataset. The MNIST dataset consists of a large collection of handwritten digits with 60,000 training images.You may load the dataset from here https://search.r-project.org/CRAN/refmans/dslabs/html/read_mnist.html in R or https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html in python.

(a) Perform PCA based on the covariance matrix (after normalizing the variables). Select the number of principle components based on the elbow method by creating a scree plot (number of principle components on the x-axis, and proportion of variance explained on the y-axis).Re- gardless of the number of principle components you selected, project all the data into the two principal components (i.e. principle component scores defined in class). Draw a scatter plot based on the two principle components, Label the points based on the corresponding digits. Do you see any patterns?

In [102…
```
from sklearn.datasets import load_digits
```

In [110…
```
load_digits()
```

```
Out[110]: {'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
                  [ 0.,  0.,  0., ..., 10.,  0.,  0.],
                  [ 0.,  0.,  0., ..., 16.,  9.,  0.],
                  ...,
                  [ 0.,  0.,  1., ...,  6.,  0.,  0.],
                  [ 0.,  0.,  2., ..., 12.,  0.,  0.],
                  [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
           'target': array([0, 1, 2, ..., 8, 9, 8]),
           'frame': None,
           'feature_names': ['pixel_0_0',
            'pixel_0_1',
            'pixel_0_2',
            'pixel_0_3',
            'pixel_0_4',
            'pixel_0_5',
            'pixel_0_6',
            'pixel_0_7',
            'pixel_1_0',
            'pixel_1_1',
            'pixel_1_2',
            'pixel_1_3',
            'pixel_1_4',
            'pixel_1_5',
            'pixel_1_6',
            'pixel_1_7',
            'pixel_2_0',
            'pixel_2_1',
            'pixel_2_2',
            'pixel_2_3',
            'pixel_2_4',
            'pixel_2_5',
            'pixel_2_6',
            'pixel_2_7',
            'pixel_3_0',
            'pixel_3_1',
            'pixel_3_2',
            'pixel_3_3',
            'pixel_3_4',
            'pixel_3_5',
            'pixel_3_6',
            'pixel_3_7',
            'pixel_4_0',
            'pixel_4_1',
            'pixel_4_2',
            'pixel_4_3',
            'pixel_4_4',
            'pixel_4_5',
            'pixel_4_6',
            'pixel_4_7',
            'pixel_5_0',
            'pixel_5_1',
            'pixel_5_2',
            'pixel_5_3',
            'pixel_5_4',
            'pixel_5_5',
            'pixel_5_6',
            'pixel_5_7',
            'pixel_6_0',
            'pixel_6_1',
            'pixel_6_2',
            'pixel_6_3',
            'pixel_6_4',
            'pixel_6_5',
            'pixel_6_6',
            'pixel_6_7',
            'pixel_7_0',
            'pixel_7_1',
            'pixel_7_2',
            'pixel_7_3',
            'pixel_7_4',
            'pixel_7_5',
            'pixel_7_6',
            'pixel_7_7'],
           'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
           'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
                   [ 0.,  0., 13., ..., 15.,  5.,  0.],
                   [ 0.,  3., 15., ..., 11.,  8.,  0.],
                   ...,
                   [ 0.,  4., 11., ..., 12.,  7.,  0.],
                   [ 0.,  2., 14., ..., 12.,  0.,  0.],
                   [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

                  [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
                   [ 0.,  0.,  0., ...,  9.,  0.,  0.],
                   [ 0.,  0.,  3., ...,  6.,  0.,  0.],
                   ...,
                   [ 0.,  0.,  1., ...,  6.,  0.,  0.],
                   [ 0.,  0.,  1., ...,  6.,  0.,  0.],
                   [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

                  [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
                   [ 0.,  0.,  3., ..., 14.,  0.,  0.],
                   [ 0.,  0.,  8., ..., 16.,  0.,  0.],
                   ...,
                   [ 0.,  9., 16., ...,  0.,  0.,  0.],
                   [ 0.,  3., 13., ..., 11.,  5.,  0.],
                   [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

                  ...,

                  [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
                   [ 0.,  0., 13., ...,  2.,  1.,  0.],
                   [ 0.,  0., 16., ..., 16.,  5.,  0.],
                   ...,
                   [ 0.,  0., 16., ..., 15.,  0.,  0.],
                   [ 0.,  0., 15., ..., 16.,  0.,  0.],
                   [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

                  [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
                   [ 0.,  0., 14., ..., 15.,  1.,  0.],
                   [ 0.,  4., 16., ..., 16.,  7.,  0.],
                   ...,
                   [ 0.,  0.,  0., ..., 16.,  2.,  0.],
                   [ 0.,  0.,  4., ..., 16.,  2.,  0.],
                   [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

                  [[ 0.,  0., 10., ...,  1.,  0.,  0.],
                   [ 0.,  2., 16., ...,  1.,  0.,  0.],
                   [ 0.,  0., 15., ..., 15.,  0.,  0.],
                   ...,
                   [ 0.,  4., 16., ..., 16.,  6.,  0.],
                   [ 0.,  8., 16., ..., 16.,  8.,  0.],
                   [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
           'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten digits dataset\n--------------------------------------------
          -------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 1797\n    :Number of Attributes: 64\n    :Attribute Inform
          ation: 8x8 image of integer pixels in the range 0..16.\n    :Missing Attribute Values: None\n    :Creator: E. Alpaydin (alpaydin
          '@' boun.edu.tr)\n    :Date: July; 1998\n\nThis is a copy of the test set of the UCI ML hand-written digits datasets\nhttps://ar
          chive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits\n\nThe data set contains images of hand-written digits:
          10 classes where\neach class refers to a digit.\n\nPreprocessing programs made available by NIST were used to extract\nnormalize
          d bitmaps of handwritten digits from a preprinted form. From a\ntotal of 43 people, 30 contributed to the training set and diffe
          rent 13\nto the test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4 and the number of on pixels are counted i
          n each block. This generates\nan input matrix of 8x8 where each element is an integer in the range\n0..16. This reduces dimensio
          nality and gives invariance to small\ndistortions.\n\nFor info on NIST preprocessing routines, see M. D. Garris, J. L. Blue,
          G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition Sy
          stem, NISTIR 5469,\n1994.\n\n.. topic:: References\n\n  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their\n
          Applications to Handwritten Digit Recognition, MSc Thesis, Institute of\n    Graduate Studies in Science and Engineering, Bogazi
          ci University.\n  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n  - Ken Tang and Ponnuthurai N. Suganthan
          and Xi Yao and A. Kai Qin.\n    Linear dimensionalityreduction using relevance weighted LDA. School of\n    Electrical and Elect
          ronic Engineering Nanyang Technological University.\n    2005.\n  - Claudio Gentile. A New Approximate Maximal Margin Classifica
          tion\n    Algorithm. NIPS. 2000.\n"}
```

In [116…
```
load_digits().data.shape
```

Out[116]:
```
(1797, 64)
```

In [117…
```
load_digits().feature_names
```

Out[117]:
```
['pixel_0_0',
 'pixel_0_1',
 'pixel_0_2',
 'pixel_0_3',
 'pixel_0_4',
 'pixel_0_5',
 'pixel_0_6',
 'pixel_0_7',
 'pixel_1_0',
 'pixel_1_1',
 'pixel_1_2',
 'pixel_1_3',
 'pixel_1_4',
 'pixel_1_5',
 'pixel_1_6',
 'pixel_1_7',
 'pixel_2_0',
 'pixel_2_1',
 'pixel_2_2',
 'pixel_2_3',
 'pixel_2_4',
 'pixel_2_5',
 'pixel_2_6',
 'pixel_2_7',
 'pixel_3_0',
 'pixel_3_1',
 'pixel_3_2',
 'pixel_3_3',
 'pixel_3_4',
 'pixel_3_5',
 'pixel_3_6',
 'pixel_3_7',
 'pixel_4_0',
 'pixel_4_1',
 'pixel_4_2',
 'pixel_4_3',
 'pixel_4_4',
 'pixel_4_5',
 'pixel_4_6',
 'pixel_4_7',
 'pixel_5_0',
 'pixel_5_1',
 'pixel_5_2',
 'pixel_5_3',
 'pixel_5_4',
 'pixel_5_5',
 'pixel_5_6',
 'pixel_5_7',
 'pixel_6_0',
 'pixel_6_1',
 'pixel_6_2',
 'pixel_6_3',
 'pixel_6_4',
 'pixel_6_5',
 'pixel_6_6',
 'pixel_6_7',
 'pixel_7_0',
 'pixel_7_1',
 'pixel_7_2',
 'pixel_7_3',
 'pixel_7_4',
 'pixel_7_5',
 'pixel_7_6',
 'pixel_7_7']
```

In [125…
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
```

In [127…
```
df = pd.DataFrame(data = load_digits().data)
df['target'] = load_digits().target
```

In [128…
```
df.head()
```

Out[128]:

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	target
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0	1
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0	4

5 rows × 65 columns

In [133…
```
df.isna().sum()
```

```
Out[133]: 0         0
          1         0
          2         0
          3         0
          4         0
                   ..
          60        0
          61        0
          62        0
          63        0
          target    0
          Length: 65, dtype: int64
```

```
In [135…  X = df.drop('target', axis = 1)
          y = df['target']
```

```
In [136…  norm = MinMaxScaler()
```

```
In [137…  X_norm = norm.fit_transform(X)
```

```
In [175…  n_pc = np.arange(1,26,1)
```

```
In [176…  n_pc
```

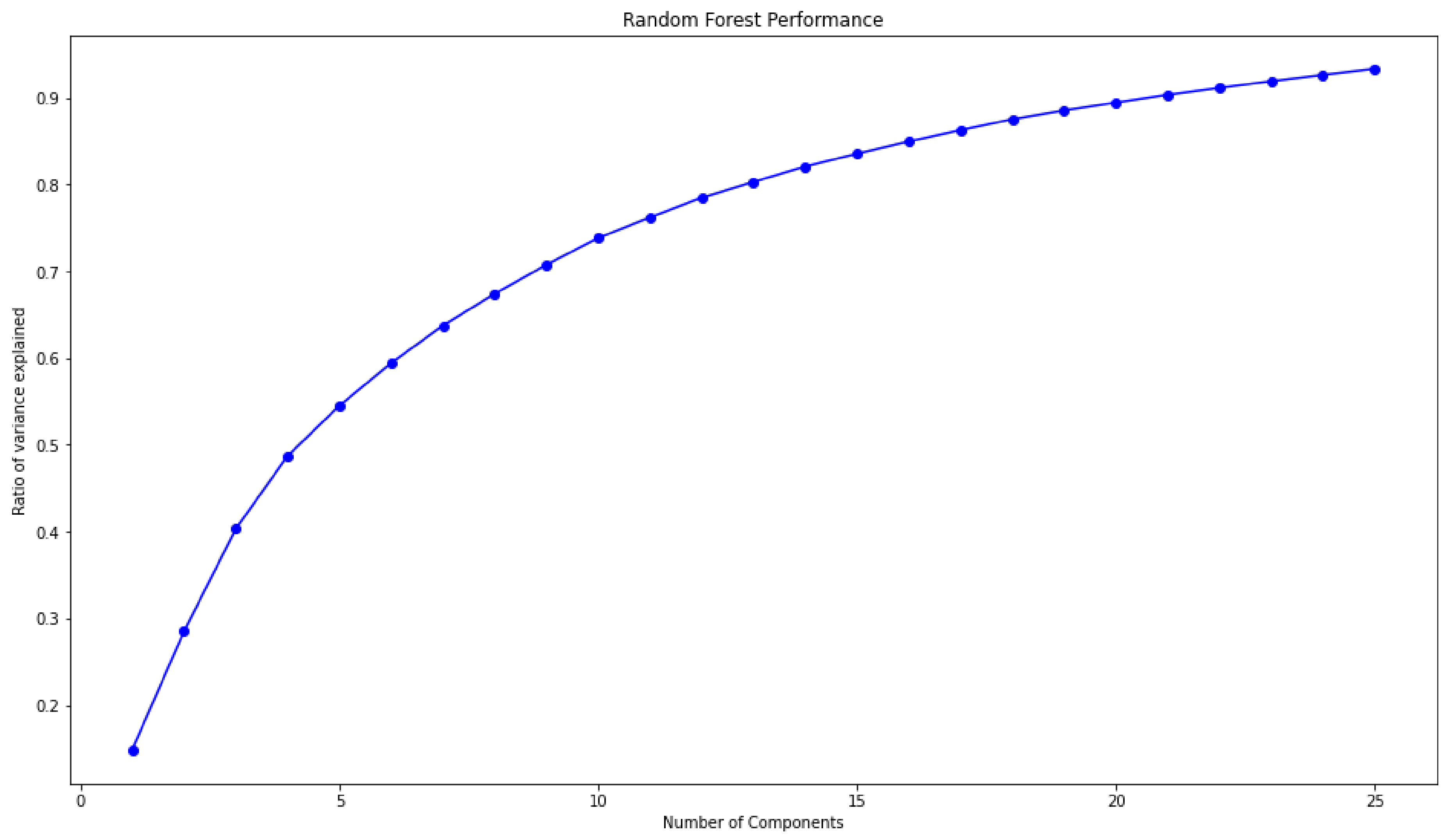
```
Out[176]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
                 18, 19, 20, 21, 22, 23, 24, 25])
```

```
In [177…  from sklearn.decomposition import PCA
```

```
In [178…  pca = PCA()
          principal_components = pca.fit_transform(X)
          variance_sum = {}
```

```
In [179…  for i in n_pc:
              pca = PCA(n_components = i)
              X_fit = pca.fit_transform(X)
              variance_sum[i] = pca.explained_variance_ratio_.cumsum()[-1]
```

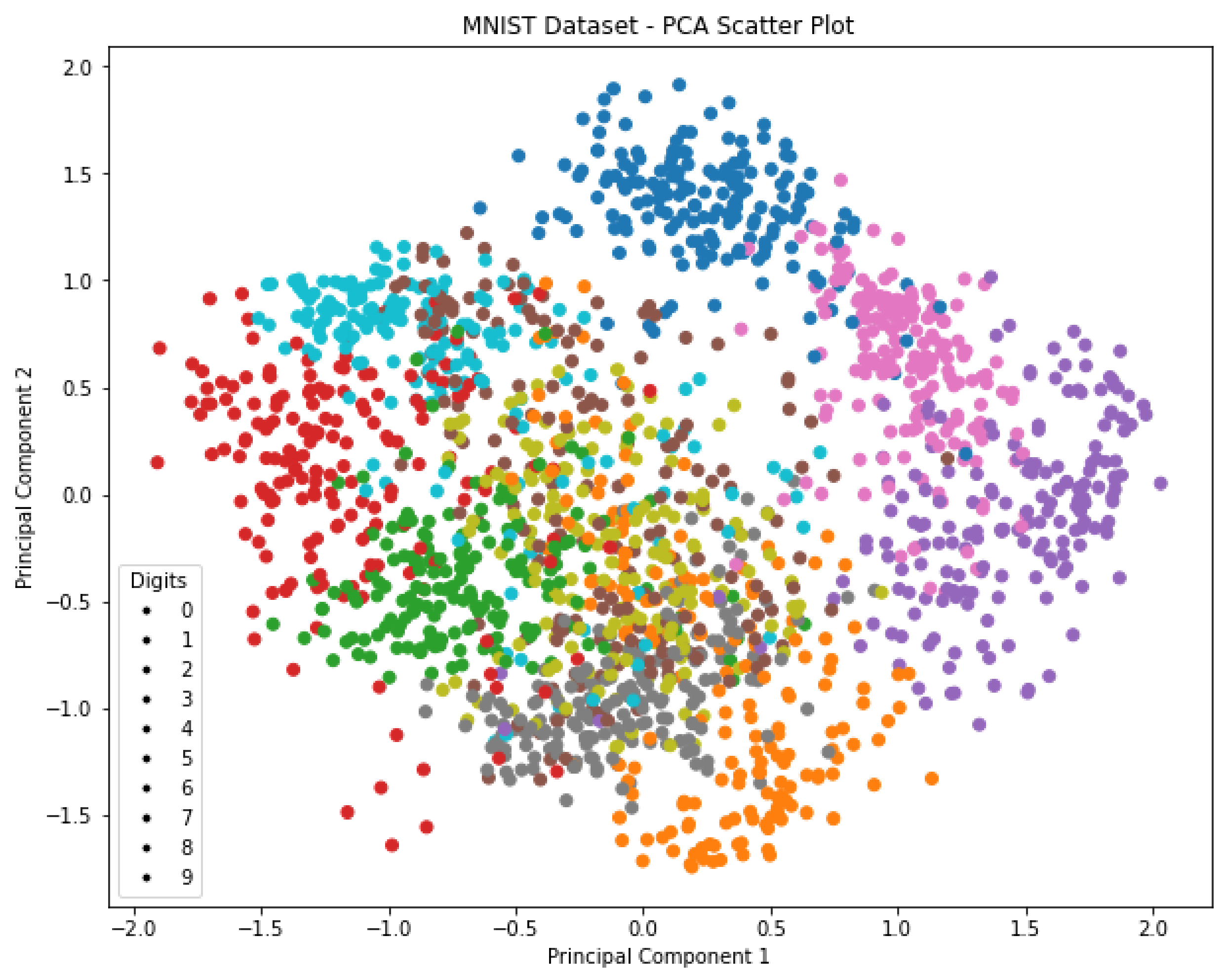
```
In [187…  plt.figure(figsize = (16,9))
          plt.plot(variance_sum.keys(), variance_sum.values(), color = 'blue', marker='o')
          plt.xlabel('Number of Components')
          plt.ylabel('Ratio of variance explained')
          plt.title('Random Forest Performance')
          plt.show()
```



```
In [188…  pca_2 = PCA(n_components = 2)
          X_fit = pca.fit_transform(X_norm)
```

```
In [189…  plt.figure(figsize=(10, 8))
          scatter = plt.scatter(X_fit[:, 0], X_fit[:, 1], c=y, cmap='tab10')
          plt.xlabel('Principal Component 1')
          plt.ylabel('Principal Component 2')
          plt.title('MNIST Dataset - PCA Scatter Plot')

          # Add legend based on the corresponding digits
          legend_elements = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='k', markersize=5, label=str(i))
```

```
                    for i in range(10)]
plt.legend(handles=legend_elements, title='Digits')

plt.show()
```

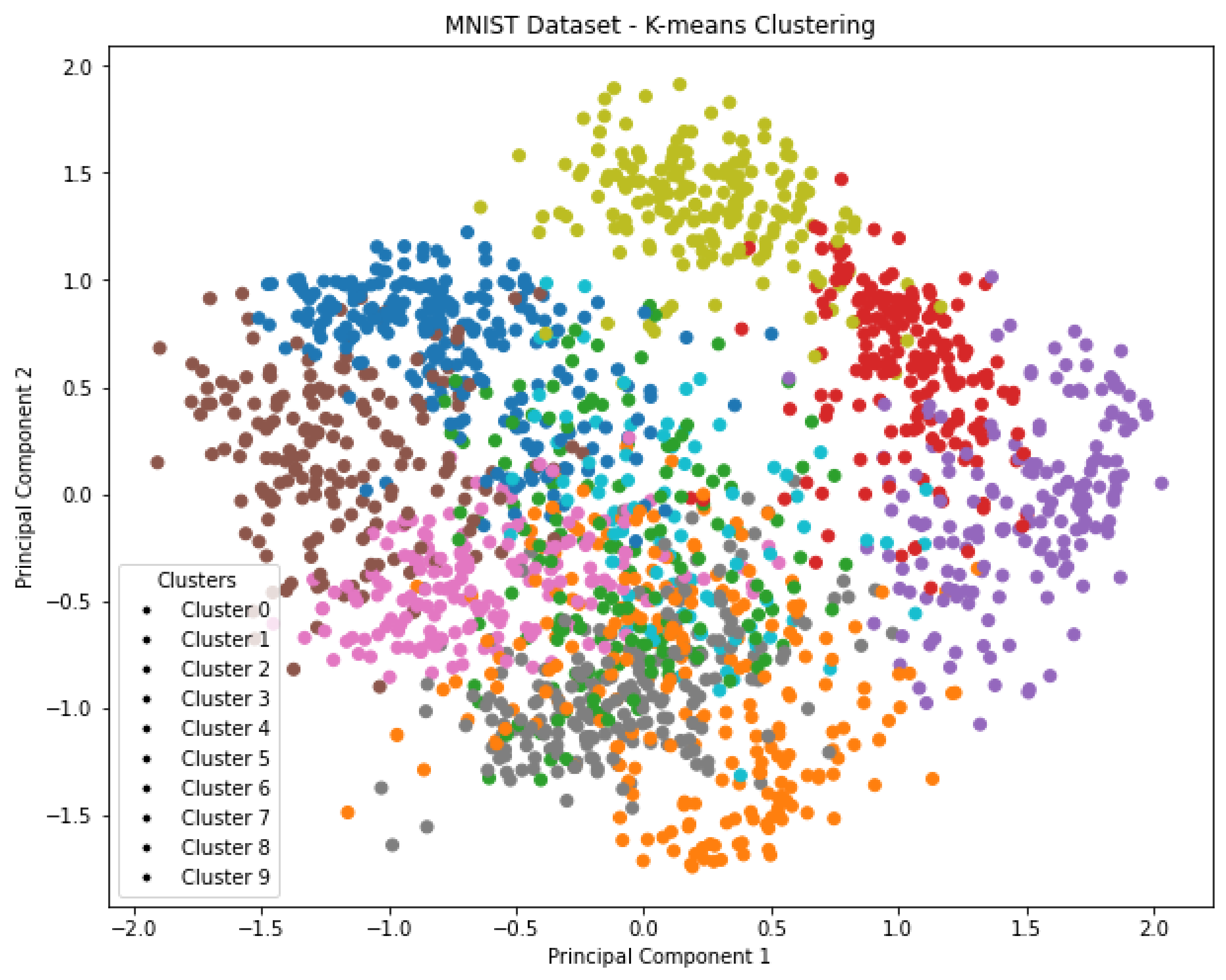


In [191…

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

In [195…

```
kmeans = KMeans(n_clusters=10)
labels = kmeans.fit_predict(X_fit)

# Plot the scatter plot based on the two principal components
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_fit[:, 0], X_fit[:, 1], c=labels, cmap='tab10')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('MNIST Dataset - K-means Clustering')

# Add legend based on the clustering results
legend_elements = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='k', markersize=5, label=f'Cluster {i}')
                    for i in range(10)]
plt.legend(handles=legend_elements, title='Clusters')

plt.show()
```



In [208…

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

In [209…
```
X_train, X_test, y_train, y_test = train_test_split(X_fit, y, test_size=0.2)
```

In [210…
```
# Set up parameter grid for GridSearchCV
param_grid = {
    'kernel': ['poly', 'rbf'],
    'degree': [1, 2, 3],
    'C': [0.1, 1, 10]
}
```

In [211…
```
# Perform grid search using cross-validation
svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

Out[211]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'degree': [1, 2, 3],
                         'kernel': ['poly', 'rbf']})
```

In [212…
```
# Get the optimal model
best_model = grid_search.best_estimator_
```

In [213…
```
# Evaluate the optimal model on the test set
accuracy = best_model.score(X_test, y_test)

# Report the optimal model and its accuracy
print("Optimal Model:")
print(best_model)
print("Accuracy:", accuracy)
```

```
Optimal Model:
SVC(C=10, degree=1)
Accuracy: 0.9944444444444445
```

In [222…
```
from sklearn.metrics import confusion_matrix
```

In [216…
```
y_pred = best_model.predict(X_test)
```

In [223…
```
print(confusion_matrix(y_test, y_pred))
```

```
[[39  0  0  0  0  0  0  0  0  0]
 [ 0 28  0  0  0  0  0  0  0  0]
 [ 0  0 32  0  0  0  0  0  0  0]
 [ 0  0  0 30  0  0  0  0  0  0]
 [ 0  0  0  0 37  0  0  0  0  0]
 [ 0  0  0  0  0 37  0  0  0  0]
 [ 0  0  0  0  0  0 32  0  0  0]
 [ 0  0  0  0  0  0  0 34  0  0]
 [ 0  0  0  0  0  0  0  0 43  0]
 [ 0  0  0  1  0  0  0  1  0 46]]
```

In [ ]: