In this project we will design k-means algorithm from scratch, to cluster data and visuzlize it and make conclusion from results.

Task:

```
1) Clusterize data and visualize it
```

Methods:

```
1) K-means clustering algorithm
2) PCA for dimensionality reduction
```

K-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

Given a set of observations (x1, x2, ..., xn), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k (≤ n) sets S = {S1, S2, ..., Sk} so as to minimize the within-cluster sum of squares (WCSS) (i.e. variance).

Formally, the objective is to find

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2 = \underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} |S_i| \operatorname{Var} S_i$$

where μ_1 is the mean (also called centroid) of points in S_i, i.e.d

$$\boldsymbol{\mu}_i = \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \mathbf{x},$$

:

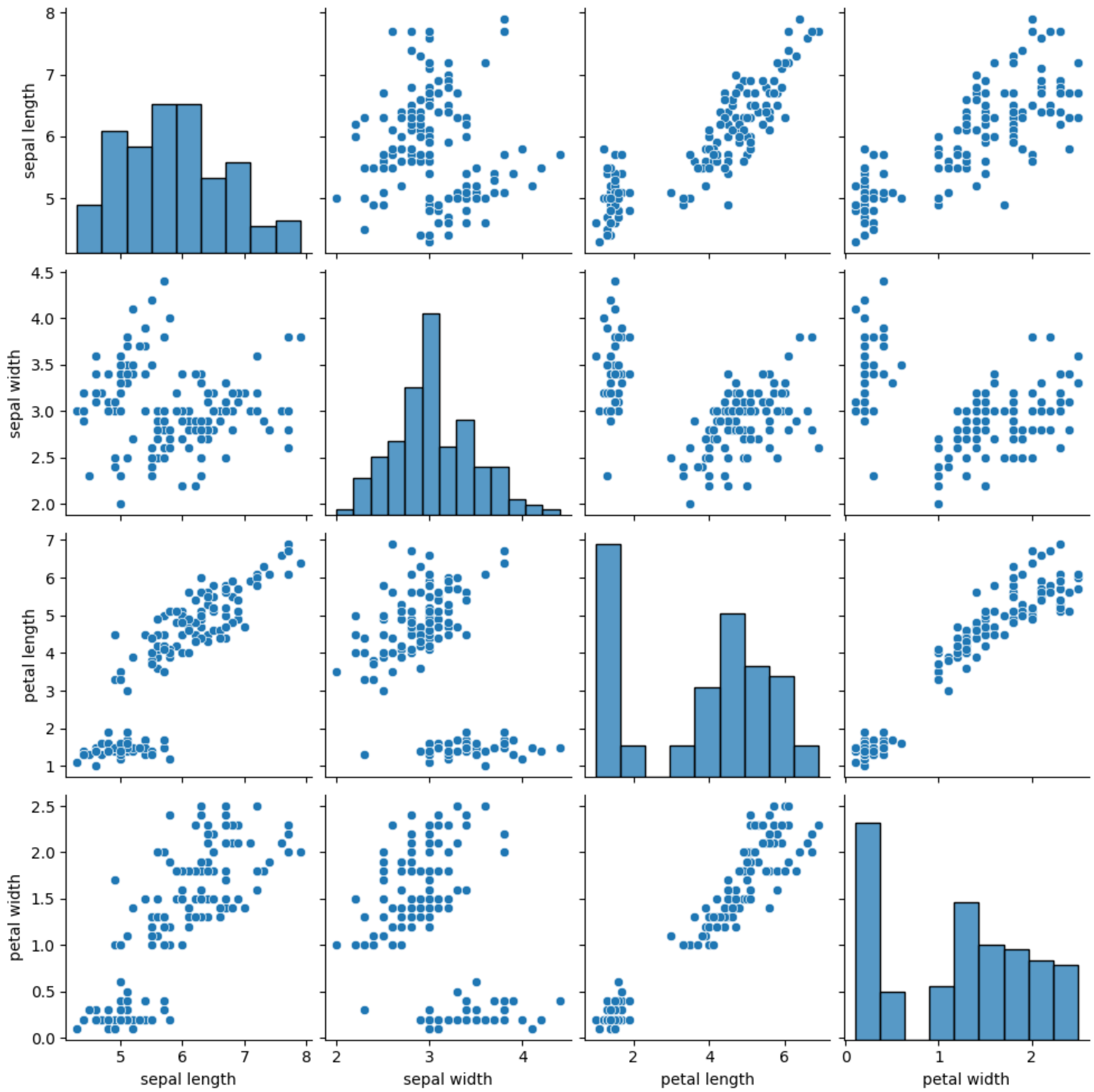
In [9]:
```
from ucimlrepo import fetch_ucirepo
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# fetch dataset
iris = fetch_ucirepo(id=53)

# data (as pandas dataframes)
X = iris.data.features
y = iris.data.targets

print(X)
print(y)
```

```
     sepal length  sepal width  petal length  petal width
0             5.1          3.5           1.4          0.2
1             4.9          3.0           1.4          0.2
2             4.7          3.2           1.3          0.2
3             4.6          3.1           1.5          0.2
4             5.0          3.6           1.4          0.2
..            ...          ...           ...          ...
145           6.7          3.0           5.2          2.3
146           6.3          2.5           5.0          1.9
147           6.5          3.0           5.2          2.0
148           6.2          3.4           5.4          2.3
149           5.9          3.0           5.1          1.8

[150 rows x 4 columns]
              class
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..              ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 1 columns]
```

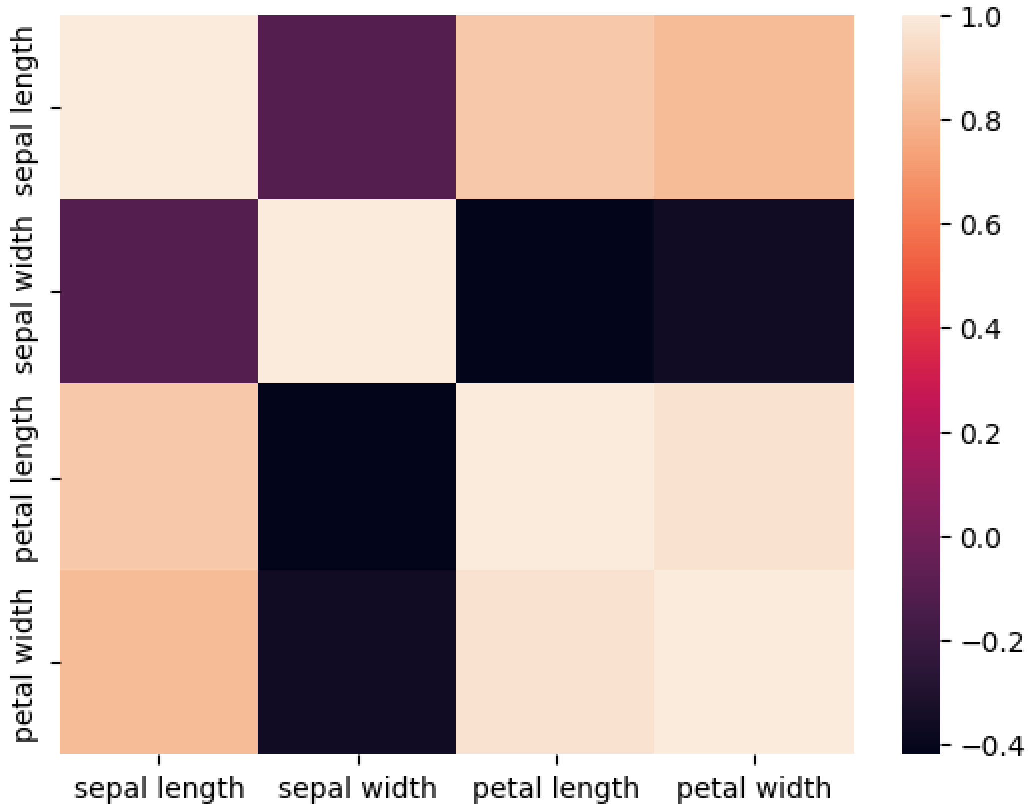
This dataset ussualy is used for classification problems, but here we will use it for clustering, therefore we don't need y`s

Let's explore the dataset:

In [10]:
```
sns.pairplot(data=X)
plt.show()
```

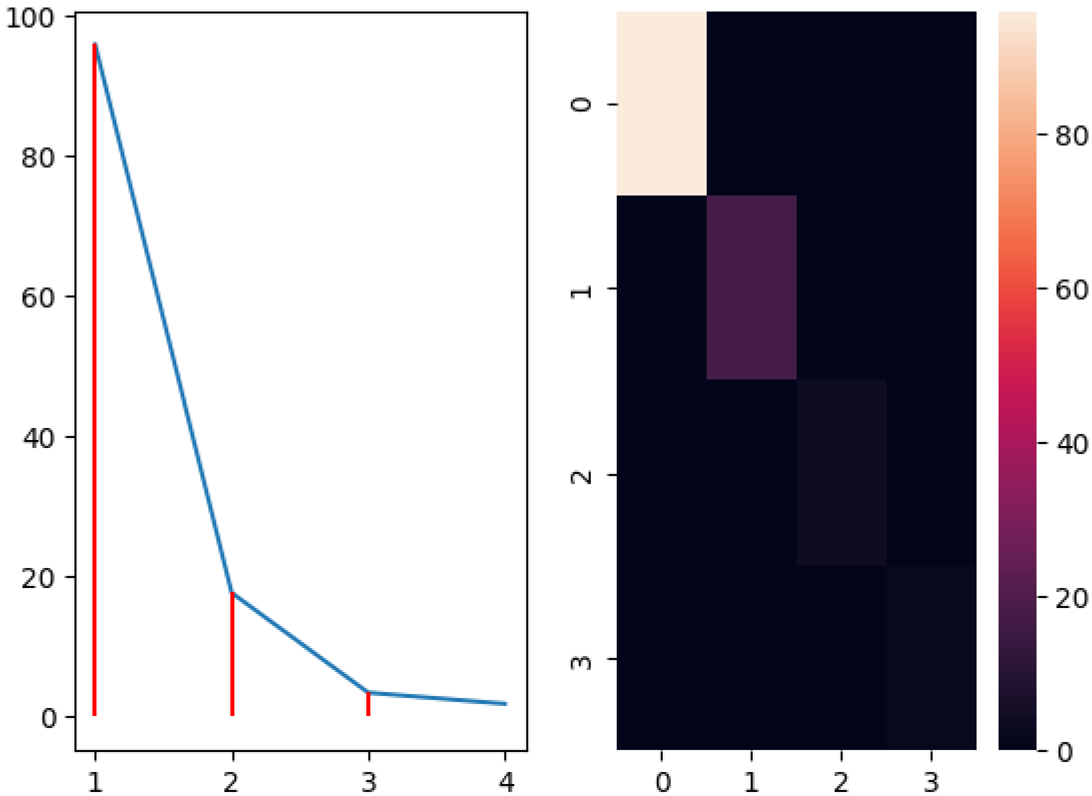


In [11]:
```
sns.heatmap(X.corr())
plt.show()
```



In [12]:

```
# PCA for reducing dimension

lsv, sv, rsv = np.linalg.svd(X)

diag_sv = np.diag(sv)

fig, ax = plt.subplots(nrows=1,ncols=2)

sns.heatmap(data=diag_sv)
sns.lineplot(x=list(range(1, len(sv)+1)), y = sv, ax = ax[0])

ax[0].set_xticks(list(range(1, len(sv)+1)))

for i in range(1, len(sv)):
    ax[0].vlines(i, ymin=0, ymax=sv[i-1], color='red')
plt.show()
```



We can use 3 clusters for our task, and then reduce dimension to visualize our results

Steps of k-means algorithm:

```
1) Create k-centroids
2) Determine distances from centroids
3) Assert data point to cluster with lowest distance
4) Recalculate centroids as mean their clusters
5) Repeat N times or until there will be no changes
```

In [13]:
```
def cluster(data, k=2):

    # Step 1 Creating k-centroids
    centroids = []
    clusters = [[] for _ in range(k)]
    for i in range(k):
        centroid = []
        for feature in data:
            feature_min = data[feature].min()
            feature_max = data[feature].max()
            centroid.append(np.random.uniform(low=feature_min, high=feature_max))
        centroids.append(centroid)

    # Step 5 recalculating clusters k times
    for _ in range(k):
        # Step 2 Calculating point distances
        distances = []
        clusters = [[] for _ in range(k)]
        for index, point in data.iterrows():
```

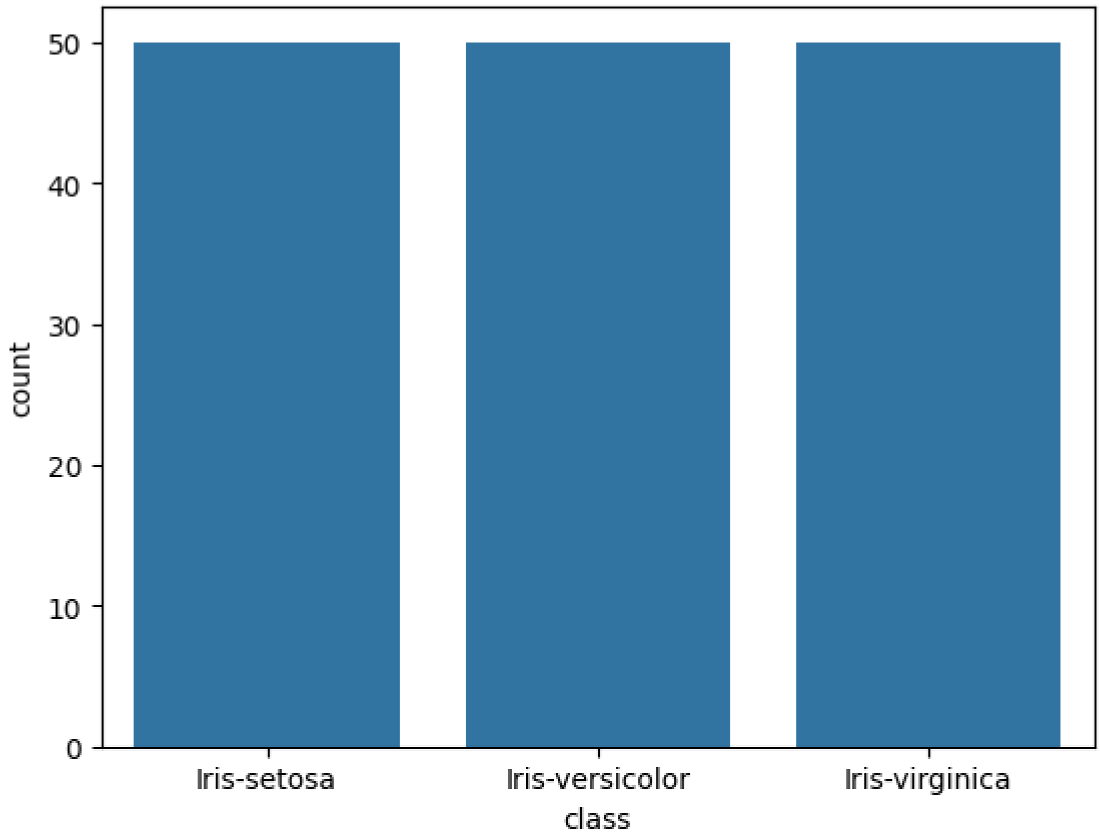
```
            point_centroid = []
            for centroid in centroids:
                dist = np.linalg.norm(point - centroid)
                point_centroid.append(dist)
            distances.append(point_centroid)

        # Step 3 Assign points to clusters
        for i, point_distances in enumerate(distances):
            cluster_index = np.argmin(point_distances)
            clusters[cluster_index].append(data.iloc[i].values)

        # Step 4 Recalculating centroids
        for i, cluster_points in enumerate(clusters):
            if len(cluster_points) != 0:
                centroids[i] = np.mean(cluster_points, axis=0)

    return clusters
```

In [15]:
```
k = 3
clusters = cluster(X, k)
for c in clusters:
    print(len(c))
```

```
30
50
70
```

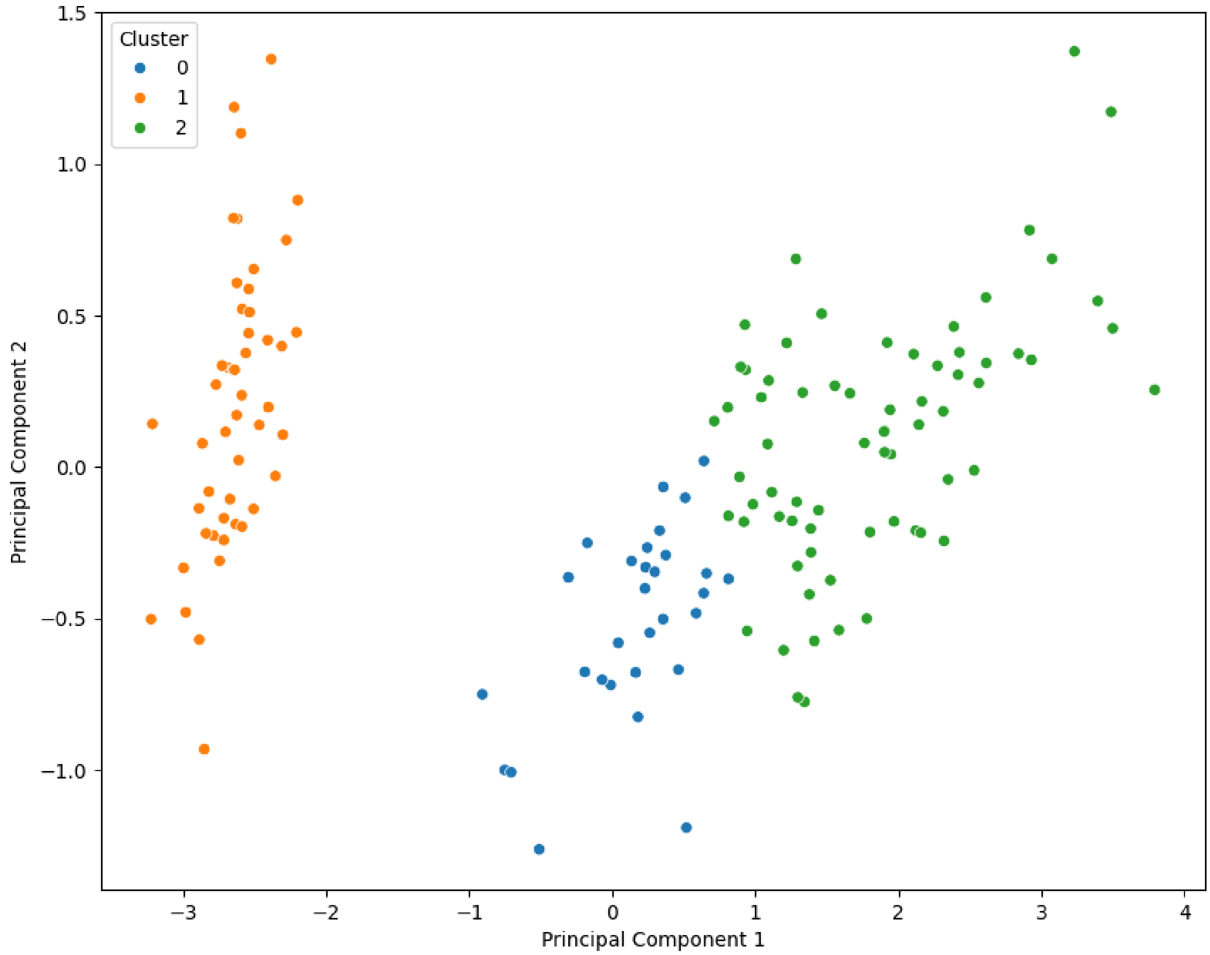
We have 3 different clusters with different sizes:

In [16]:
```
sns.countplot(data=y, x='class')
plt.show()
```



In [17]:

```
from sklearn.decomposition import PCA

def visualize_clusters(clusters):
    clustered_data = np.concatenate(clusters)


    pca = PCA(n_components=2)
    clustered_data_pca = pca.fit_transform(clustered_data)

    df = pd.DataFrame(data=clustered_data_pca, columns=['PC1', 'PC2'])

    cluster_labels = []
    for i, cluster in enumerate(clusters):
        cluster_labels.extend([i]*len(cluster))
    df['Cluster'] = cluster_labels

    plt.figure(figsize=(10, 8))
    sns.scatterplot(x='PC1', y='PC2', hue='Cluster', data=df, palette='tab10')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()

visualize_clusters(clusters)
```

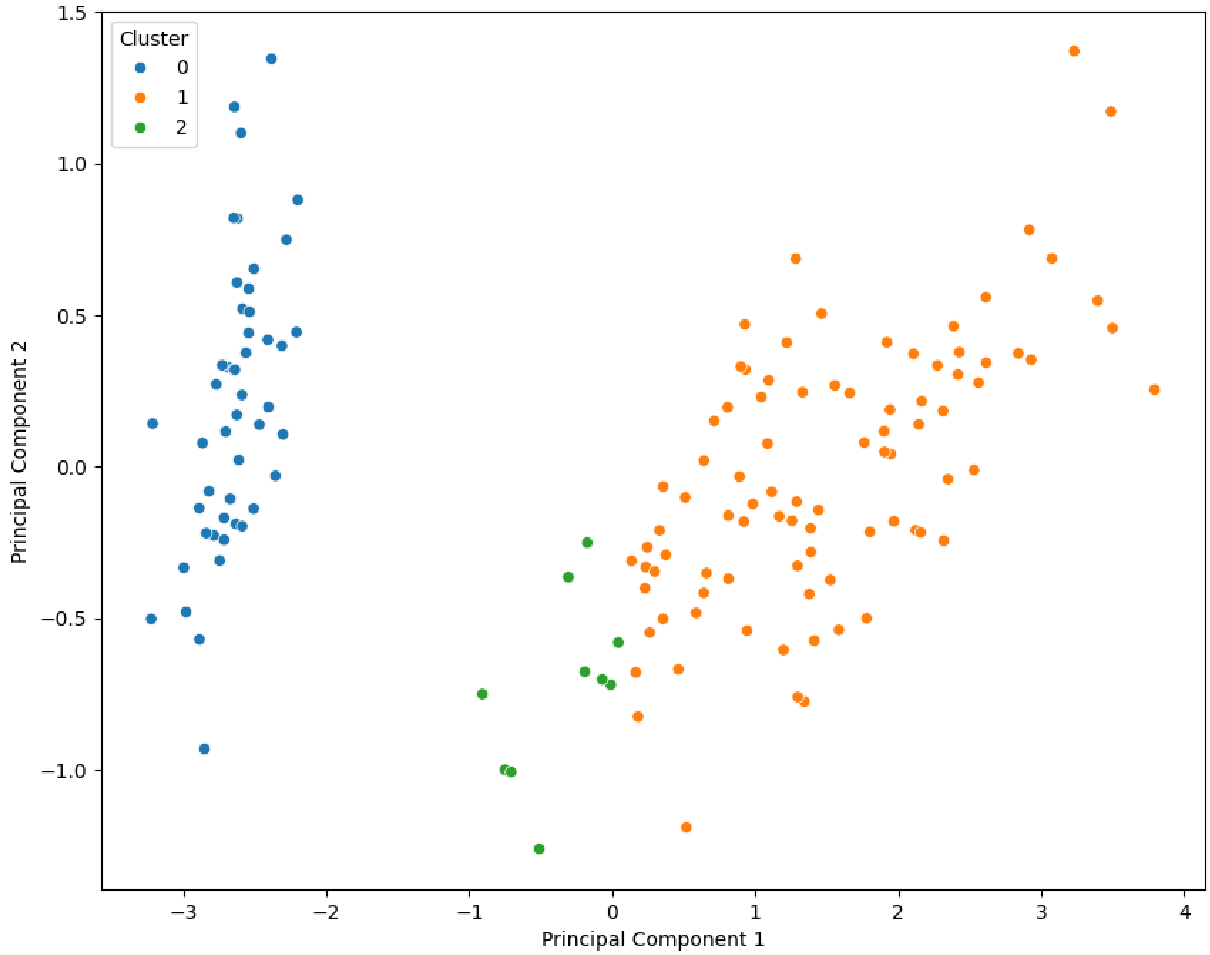


Some data can be missed because of:

```
1) PCA, we recuced from 4D to 2D
2) k-means algorithm depends on random selected clusters
```

We can prove second paragraph by using this algorithm several times:

In [19]:
```
clusters = cluster(X, k)
visualize_clusters(clusters)
```



Calling again this function we can see that result is different (third cluster is gone), and now we have almost evenly distibuted data.

In conclussion:

Advantages of K-means:

```
1) Simple Implementation: K-means is straightforward and easy to
implement, making it suitable for large datasets and real-time
applications
2) Efficiency: It is computationally efficient, particularly for
large datasets, as its time complexity is linear with the number of
data point.
3) Scalability: K-means can handle a large number of variables or
features, making it suitable for high-dimensional d.le for such
cases.
```

Disadvantages of K-means:

```
1) Dependence on Initial Centroids: The final clustering outcome
may vary depending on the initial selection of centroids, and a
poor initial selection can lead to suboptimal clustering results.
2) Sensitive to Outliers: K-means is sensitive to outliers, as they
```

can significantly influence the positions of centroids and the clustering outcome.

3) Assumption of Spherical Clusters: K-means assumes that clusters are spherical and of similar size, which may not hold true for all datasets.

4) Fixed Number of Clusters: The number of clusters (k) needs to be specified beforehand, which can be challenging to determine, especially when dealing with unknown or complex dataset..