**Problem Statement:**

K-medoids Clustering on the Iris Dataset.

**Data Description:**

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

The columns in this dataset are:

1. SepalLengthCm
2. SepalWidthCm
3. PetalLengthCm
4. PetalWidthCm

**Algorithm:**

1. Start with an initial set of medoids (selected randomly or using an initialization method).
2. Iteratively:
   * Replace a medoid with a non-medoid data point if it reduces the total sum of distances (sum of squared errors, SSE) within the resulting cluster.
   * Continue this process until no further improvement can be made.

The objective is to minimize the Sum of Squared Errors (SSE), which is defined as:

**Program:**

import numpy as np

from sklearn\_extra.cluster import KMedoids

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

import pandas as pd

# Sample data with 4 columns

dataset = pd.read\_csv("/content/Iris.csv")

X\_data = dataset.iloc[:,1:-1]

data = X\_data.to\_numpy()

# Step 1: Normalize the data

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

clustering\_score = []

for i in range(1, 11):

kmedoids = KMedoids(n\_clusters = i, init = 'random', random\_state = 42)

kmedoids.fit(data\_scaled)

clustering\_score.append(kmedoids.inertia\_) # inertia\_ = Sum of squared distances of samples to their closest cluster center.

plt.figure(figsize=(10,6))

plt.plot(range(1, 11), clustering\_score)

plt.scatter(5,clustering\_score[4], s = 200, c = 'red', marker='\*')

plt.title('The Elbow Method')

plt.xlabel('No. of Clusters')

plt.ylabel('Clustering Score')

plt.show()

# Step 2: Apply K-medoids

k = 5 # Choose the number of clusters

kmedoids = KMedoids(n\_clusters=k, metric='euclidean', random\_state=42)

labels = kmedoids.fit\_predict(data\_scaled)

# Step 3: Evaluate the clustering

print("Cluster labels:", labels)

print("Medoids:", kmedoids.cluster\_centers\_)

print("Silhouette Score:", silhouette\_score(data\_scaled, labels))

# Step 4: Reduce dimensions for visualization

pca = PCA(n\_components=2)

data\_2d = pca.fit\_transform(data\_scaled)

# Step 5: Plot the clusters

plt.figure(figsize=(8, 6))

for cluster in np.unique(labels):

plt.scatter(data\_2d[labels == cluster, 0], data\_2d[labels == cluster, 1], label=f'Cluster {cluster}')

# Plot the medoids

medoid\_2d = pca.transform(kmedoids.cluster\_centers\_)

plt.scatter(medoid\_2d[:, 0], medoid\_2d[:, 1], c='red', marker='X', s=200, label='Medoids')

plt.title('K-Medoids Clustering Visualization')

plt.xlabel('PCA Component 1')

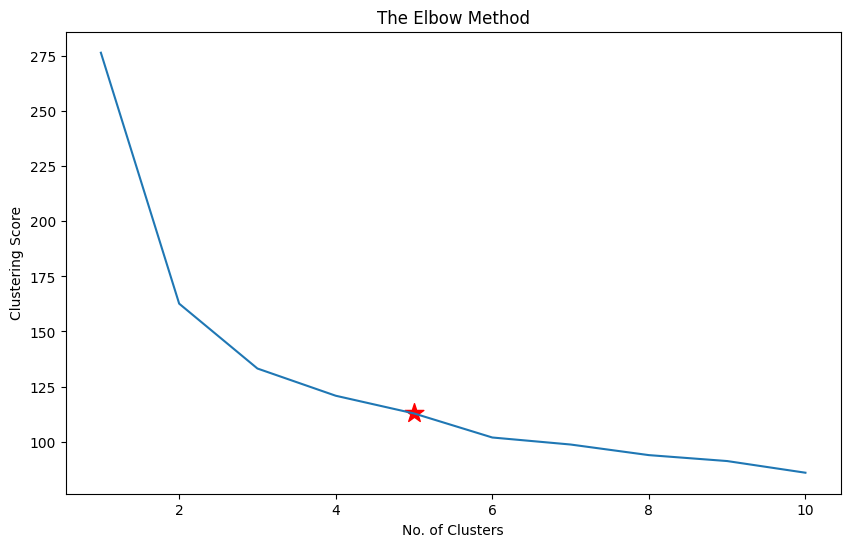
plt.ylabel('PCA Component 2')

plt.legend()

plt.grid()

plt.show()

**Output:**

****

Cluster labels: [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 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 2 0 2 3 1 0 0 3 0 3 3 0 3 0 0 0 0 3 3 3 0 0 1 0

0 0 1 2 0 3 3 3 3 1 0 0 0 3 0 3 3 0 3 3 3 0 0 0 3 0 2 1 2 1 2 2 3 2 1 2 2

1 2 1 1 2 2 2 2 3 2 1 2 1 2 2 1 1 1 2 2 2 1 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2

2 1]

Medoids: [[ 0.31099753 -0.1249576 0.47843012 0.26469891]

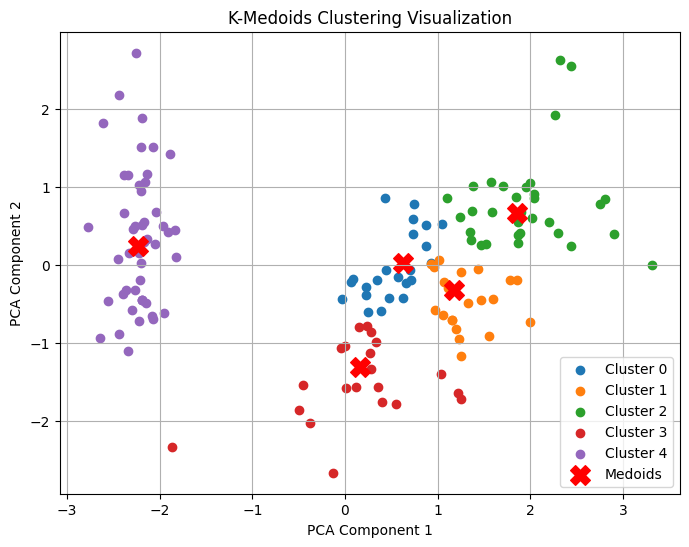
[ 0.4321654 -0.58776353 0.59216153 0.79059079]

[ 1.2803405 0.10644536 0.93335575 1.1850097 ]

[-0.29484182 -1.28197243 0.08037019 -0.12972 ]

[-1.02184904 0.80065426 -1.2844067 -1.31297673]]

Silhouette Score: 0.3759993156661742

****

**Conclusion:**

K-Medoids clustering mechanism in Partition Clustering. First, Clustering is the process of breaking down an abstract group of data points/ objects into classes of similar objects such that all the objects in one cluster have similar traits. A group of n objects is broken down into k number of clusters based on their similarities. Here I have use PCA(Principal Component Analysis) for dimension reduction, which makes it easier to represent in 2D space.