Name:	Saurabhsing Dipaksing Pardeshi
Roll No:	35
Class/Sem:	TE/V
Experiment No.:	8
Title:	Implementation of any one clustering algorithm using languages like JAVA/ python.
Date of Performance:	
Date of Submission:	
Marks:	
Sign of Faculty:	

Aim: To Study and Implement K-Medoids algorithm

Objective: Understand the working of K-Medoids algorithm and its implementation using Python.

Theory:
K-Medoids is a clustering algorithm that is very similar to K-Means. However, instead of choosing means (centroids) as the central point for each cluster, it chooses actual data points (medoids) to minimize the sum of dissimilarities between the data points and the medoids. K-Medoids is more robust to noise and outliers compared to K-Means because it uses actual data points as cluster centers.

Input:
- K: Number of clusters
- D: Dataset containing n objects

Output:
- A set of k clusters

Given k, the K-Medoids algorithm is implemented in 5 steps:
1. Step 1: Arbitrarily choose k objects from D as the initial cluster centers (medoids).
2. Step 2: Find the dissimilarity (e.g., Euclidean distance) between each object in the dataset and the medoids.
3. Step 3: Assign each object to the cluster with the nearest medoid.
4. Step 4: Update the medoids by minimizing the sum of the dissimilarities between all objects in a cluster and the medoid. For each cluster, the object that minimizes this sum becomes the new medoid.
5. Step 5: Repeat the process until there is no change in the medoids.

Example:
Let the dataset D = {2, 4, 10, 12, 3, 20, 30, 11, 25}, and k = 2 clusters.
1. **Randomly assign initial medoids**: Assume m1 = 3 and m2 = 20.
   - Cluster 1 (k1) = {2, 3, 4},
   - Cluster 2 (k2) = {10, 12, 20, 30, 11, 25}.
2. **Calculate distances and reassign medoids**:
   - After calculating the dissimilarities, update medoids to m1 = 4 and m2 = 25.
   - Cluster 1 (k1) = {2, 3, 4, 10, 12, 11},
   - Cluster 2 (k2) = {20, 30, 25}.
3. **Update medoids and repeat**:
   - Continue updating medoids and clusters until the medoids stop changing.
4. **Final Medoids and Clusters**:
   - Cluster 1 (k1) = {2, 3, 4, 10, 12, 11},
   - Cluster 2 (k2) = {20, 30, 25}.

CODE:
```
import numpy as np
from sklearn import datasets
from sklearn_extra.cluster import KMedoids
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
```

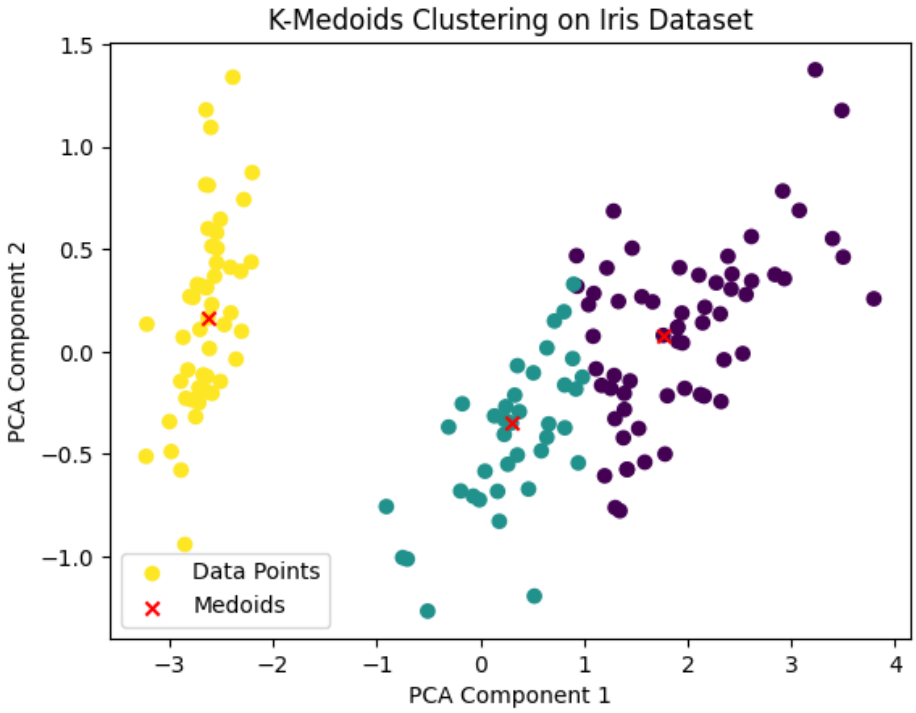
```
iris = datasets.load_iris()
data = iris.data
k = 3

kmedoids = KMedoids(n_clusters=k, random_state=0)
kmedoids.fit(data)
labels = kmedoids.labels_
medoids = kmedoids.cluster_centers_
print("Final medoids (in original feature space):\n", medoids)

pca = PCA(n_components=2)
data_2d = pca.fit_transform(data)
medoids_2d = pca.transform(medoids)
plt.scatter(data_2d[:, 0], data_2d[:, 1], c=labels, cmap='viridis', label="Data Points")
plt.scatter(medoids_2d[:, 0], medoids_2d[:, 1], c='red', label="Medoids", marker='x')
plt.title('K-Medoids Clustering on Iris Dataset')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.show()
```

OUTPUT:

```
Final medoids (in original feature space):
 [[6.5 3.  5.2 2. ]
 [5.7 2.8 4.1 1.3]
 [5.  3.4 1.5 0.2]]
```



CONCLUSION:

The K-Medoids algorithm is a robust clustering method, especially in the presence of noise and outliers, because it uses actual points from the dataset as the medoids. It effectively partitions data into k clusters based on minimizing the sum of dissimilarities between points and their assigned medoids.

What types of data preprocessing are necessary before applying the K-Medoids algorithm?

Before applying the K-Medoids algorithm, several data preprocessing steps are necessary to ensure accurate and meaningful clustering results. These include:

1. Handling missing data: Missing values can distort distance calculations. Impute missing values using techniques like mean/mode imputation, or remove records with missing data to ensure completeness.
2. Data normalization/scaling: Features with different scales can affect distance-based algorithms like K-Medoids. Normalize or scale the data so that all features contribute equally to the distance metric (e.g., using Min-Max Scaling or Standardization).
3. Outlier detection: K-Medoids is more robust than K-Means, but extreme outliers can still skew the results. Identify and handle outliers by removing them or treating them separately.
4. Categorical data encoding: If the dataset includes categorical features, convert them to numerical values using techniques like one-hot encoding, as K-Medoids relies on distance measures that work with numerical data.
5. Dimensionality reduction: If the dataset has many features, consider using dimensionality reduction techniques like Principal Component Analysis (PCA) or manual feature selection to reduce noise and improve performance.
6. Sufficient data size: Ensure the dataset contains a sufficient number of data points relative to the number of clusters. Too few data points can lead to unstable clusters or unrepresentative medoids.