Name:	Pranita Kumbhar
Roll No:	70
Class/Sem:	TE/V
Experiment No.:	8
Title:	Implementation of any one clustering algorithm using languages like JAVA/ python.
Date of Performance:	11/9/25
Date of Submission:	18/9/25
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.

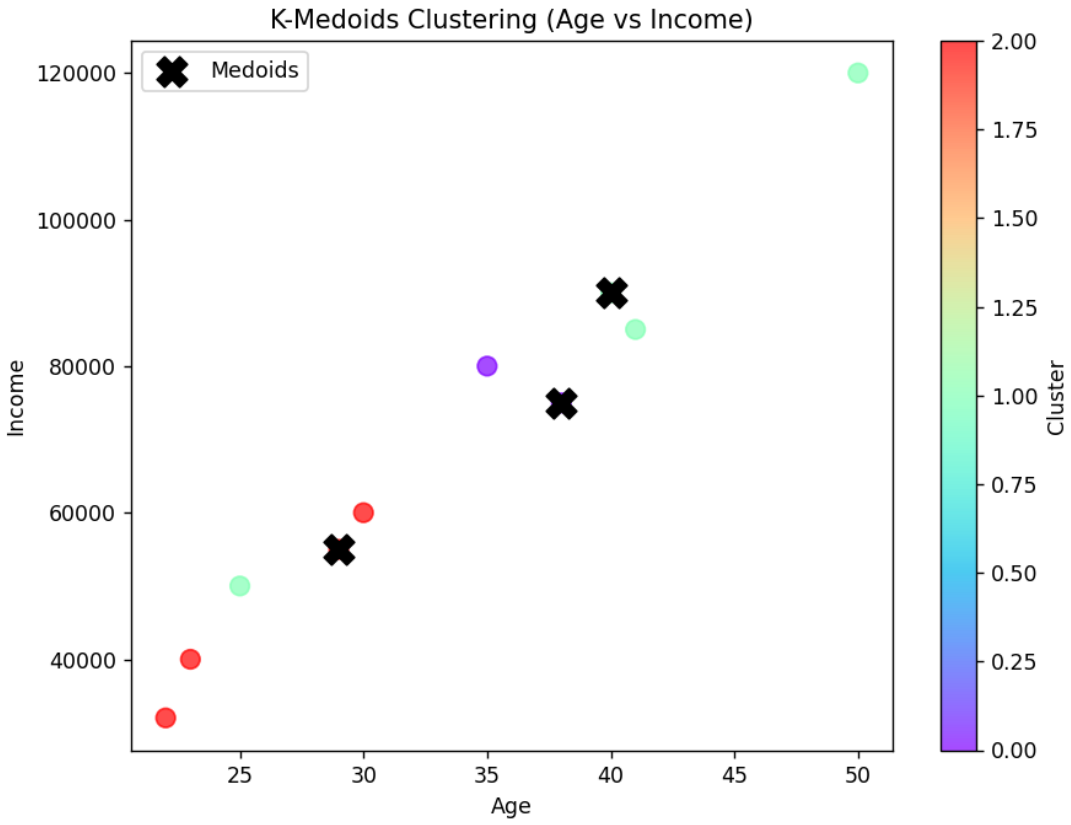
Example:
Let the dataset D = {2, 4, 10, 12, 3, 20, 30, 11, 25}, and k = 2 clusters.
1. **Randomly assign initial medoids**: Assume m1 = 3 and m2 = 20.
   - Cluster 1 (k1) = {2, 3, 4},
   - Cluster 2 (k2) = {10, 12, 20, 30, 11, 25}.
2. **Calculate distances and reassign medoids**:
   - After calculating 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 pandas as pd
import gower
from sklearn_extra.cluster import KMedoids
```

```
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# 1. Load dataset
df = pd.read_csv("C:/Users/Pranita Kumbhar/Downloads/sample_kmedoids.csv")
print("Original Dataset:")
print(df)

# 2. Compute Gower distance (handles mixed numeric + categorical data)
D = gower.gower_matrix(df)

# 3. Apply K-Medoids
k = 3
kmed = KMedoids(n_clusters=k, metric="precomputed", random_state=42)
kmed.fit(D)

# 4. Add cluster labels to dataset
df["Cluster"] = kmed.labels_
print("\nDataset with Clusters:")
print(df)

# 5. Show medoids
print("\nMedoid indices:", kmed.medoid_indices_)
print("Medoid points:\n", df.iloc[kmed.medoid_indices_])

# 6. Silhouette score (cluster quality)
score = silhouette_score(D, kmed.labels_, metric="precomputed")
print("\nSilhouette Score:", score)

# 7. Scatter plot (using Age vs Income)
plt.figure(figsize=(8,6))
scatter = plt.scatter(df["Age"], df["Income"], c=df["Cluster"], cmap="rainbow", s=80,
alpha=0.7)
plt.scatter(df.iloc[kmed.medoid_indices_]["Age"],
        df.iloc[kmed.medoid_indices_]["Income"],
        c="black", marker="X", s=200, label="Medoids")
plt.xlabel("Age")
plt.ylabel("Income")
plt.title("K-Medoids Clustering (Age vs Income)")
plt.legend()
plt.colorbar(scatter, label="Cluster")
plt.show()
```

OUTPUT:


K-Medoids Clustering (Age vs Income)
Medoids
120000
100000
80000
60000
40000
Income
25
30
35
40
45
50
Age
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Cluster

```
(base) C:\Users\Pranita Kumbhar>python decision_tree_experiment.py
Original Dataset:
   Age  Income  Gender Purchased
0   25   50000    Male       Yes
1   30   60000  Female        No
2   22   32000  Female       Yes
3   35   80000    Male        No
4   40   90000    Male       Yes
5   29   55000  Female        No
6   50  120000    Male       Yes
7   41   85000  Female       Yes
8   23   40000    Male        No
9   38   75000  Female        No

Dataset with Clusters:
   Age  Income  Gender Purchased  Cluster
0   25   50000    Male       Yes        1
1   30   60000  Female        No        2
2   22   32000  Female       Yes        2
3   35   80000    Male        No        0
4   40   90000    Male       Yes        1
5   29   55000  Female        No        2
6   50  120000    Male       Yes        1
7   41   85000  Female       Yes        1
8   23   40000    Male        No        2
9   38   75000  Female        No        0

Medoid indices: [9 4 5]
Medoid points:
    Age  Income  Gender Purchased  Cluster
9   38   75000  Female        No        0
4   40   90000    Male       Yes        1
5   29   55000  Female        No        2

Silhouette Score: 0.13726659
```

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?

➢

K-Medoids is a clustering algorithm based on pairwise **distances/dissimilarities**, the preprocessing step is **critical** for getting meaningful clusters.

Here's a clear breakdown of the **data preprocessing steps** :

- **1. Handling Missing Values**
  - **Why:** Distance measures (like Euclidean, Manhattan, or Gower) cannot work with NaN values.
  - **How:**
    - Numeric: impute with mean/median or drop if very few.
    - Categorical: impute with mode or add a category "missing".
- **2. Removing Duplicates & Irrelevant Features**
  - **Why:** Duplicate rows can bias clustering, and irrelevant ID-like columns (e.g., roll numbers) add noise.
  - **How:** Drop exact duplicates, remove or transform non-informative columns.
- **3. Encoding Categorical Features**
  - **Why:** K-Medoids works on distances; categorical values must be converted.
  - **Options:**
    - **One-hot encoding** for categorical data (but may increase dimensionality).
    - **Label encoding** only for ordinal categories.
    - **Better:** Use a **mixed-type distance measure** like **Gower distance**, which handles numeric + categorical directly.

◇ **4. Feature Scaling (Normalization/Standardization)**

- **Why:** Numeric features with large ranges dominate distance calculations.
- **How:**
    - Standardization (Z-score): (x - mean) / std
    - Min-Max scaling: (x - min) / (max - min)

◇ **5. Outlier Detection and Treatment**

- **Why:** K-Medoids is more robust to outliers than K-Means, but extreme outliers can still distort clusters by becoming medoids.
- **How:** Identify with boxplots/z-scores and decide whether to remove or keep based on domain knowledge.

◇ **6. Feature Selection / Dimensionality Reduction (Optional)**

- **Why:** Too many irrelevant/noisy features dilute distance calculations.
- **How:** Remove highly correlated or uninformative features, or apply PCA (for numeric features).

◇ **7. Choosing the Right Distance Metric**

- **Numeric data only:** Euclidean or Manhattan distance.
- **Mixed data (numeric + categorical):** Gower distance (commonly used with K-Medoids).