Experiment No: 07	TE AI&DS
Date of Performance:	Roll No: 9696
Aim: To Implement K-mean clustering algorithm to group similar data objects	
Related CO4: Implement Classification, Clustering and Association mining techniques to extract knowledge	
Objective: To learn clustering techniques	

Rubrics for assessment of Experiment:

Sr. No	Parameters	Exceed Expectations(EE)	Meet Expectations (ME)	Below Expectations (BE)
1	Timeline (2)	Early or on time (2)	One session late (1)	More than one session late (0)
2	Preparedness (2)	Knows the basic theory related to the experiment very well. (2)	Managed to explain the theory related to the experiment. (1)	Not aware of the theory to the point. (1)
3	Effort (3)	Done expt on their own. (3)	Done expt with help from other. (2)	Just managed. (1)
4	Documentation(2)	Lab experiment is documented in proper format and maintained neatly. (2)	Documented in proper format but some formatting guidelines are missed. (1)	Experiments not written in proper format (0.5)
5	Result (1)	Specific conclusion.(1)	Partially specific conclusion. (0.5)	Not specific at all. (0)

Assessment Marks:

Timeline(2)	Preparedness(2)	Effort(3)	Documentation(2)	Result(1)	Total(10)

Theory:

What is Clustering?

Clustering is dividing data points into homogeneous classes or clusters:

- Points in the same group are as similar as possible
- Points in different group are as dissimilar as possible

When a collection of objects is given, we put objects into group based on similarity.

Application of Clustering:

Clustering is used in almost all the fields. You can infer some ideas from Example 1 to come up with lot of clustering applications that you would have come across.
Listed here are few more applications, which would add to what you have learnt.

- Clustering helps marketers improve their customer base and work on the target areas. It helps group people (according to different criteria's such as willingness, purchasing power etc.) based on their similarity in many ways related to the product under consideration.
- Clustering helps in identification of groups of houses on the basis of their value, type and geographical locations.
- Clustering is used to study earth-quake. Based on the areas hit by an earthquake in a region, clustering can help analyse the next probable location where earthquake can occur.

Clustering Algorithms:

A Clustering Algorithm tries to analyse natural groups of data on the basis of some similarity. It locates the centroid of the group of data points. To carry out effective clustering, the algorithm evaluates the distance between each point from the centroid of the cluster.
The goal of clustering is to determine the intrinsic grouping in a set of unlabelled data.

K-Means Clustering Algorithm-

K-Means Clustering Algorithm involves the following steps-

Step-01:

- Choose the number of clusters K.

Step-02:

- Randomly select any K data points as cluster centers.
- Select cluster centers in such a way that they are as farther as possible from each other.

Step-03:

- Calculate the distance between each data point and each cluster center.

- The distance may be calculated either by using given distance function or by using euclidean distance formula.

Step-04:

- Assign each data point to some cluster.
- A data point is assigned to that cluster whose center is nearest to that data point.

Step-05:

- Re-compute the center of newly formed clusters.
- The center of a cluster is computed by taking mean of all the data points contained in that cluster.

Step-06:

Keep repeating the procedure from Step-03 to Step-05 until any of the following stopping criteria is met-

- Center of newly formed clusters do not change
- Data points remain present in the same cluster
- Maximum number of iterations are reached

Implementation:

Part 1: WAP in Python/java to implement K-means clustering algorithm

Suppose That The Data Mining Task Is To Cluster Points (With *.X, Y/* Representing Location)
Into Three Clusters, Where The Points Are
A(2,10),B(2,5),C(8,4),D(5,8),*E(*7,5),*F(*6,4),G(1,2),*H(*4,9).

Read No of Objects
Read all Object Details
Read K
Give Output In Step Wise/Iteration Wise (distance matrix with cluster assignment)

Part 2: using standard libraries / packages

K-Means Clustering with scikit-learn - DataCamp

Post lab:

1. Differentiate K-means and K-medoid algorithms with one example
2. List the packages/ libraries of python used in experiments.

Other resources:
sklearn.cluster.KMeans — scikit-learn 0.24.1 documentation (scikit-learn.org)
Build K-Means from scratch in Python | by Rishit Dagli | Medium
K-Means Clustering Algorithm from Scratch - ML+ (machinelearningplus.com)
10 Clustering Algorithms With Python (machinelearningmastery.com)

Code with output:

Part 1:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import style
# Points
# A(2,10),B(2,5),C(8,4),D(5,8),E(7,5),F(6,4),G(1,2),H(4,9).
style.use('ggplot')
X = np.array([[2, 10], # A
              [2, 5], # B
              [8, 4], # C
              [5, 8], # D
              [7, 5], # E
              [6, 4], # F
              [1, 2], # G
              [4, 9]  # H
               ])

plt.scatter(X[:,0], X[:,1], s=150)
plt.show()

#defining functions
class K_Means:
    def __init__(self, k=2, tol=0.001, max_iter=300):
        # Initialize K-Means parameters: number of clusters (k), tolerance, and maximum
iterations
        self.k = k
        self.tol = tol
        self.max_iter = max_iter

    def fit(self, data):
        # Initialize centroids for each cluster
        # Randomly assign the first k data points as centroids
        self.centroids = {}
        for i in range(self.k):
            self.centroids[i] = data[i]

        # Iterate to update centroids and cluster assignments
        for i in range(self.max_iter):
            self.classifications = {}  # Assignments of data points to clusters

            # Initialize classifications
            for i in range(self.k):
                self.classifications[i] = []

            # Assign each data point to the nearest centroid
            for featureset in data:
```

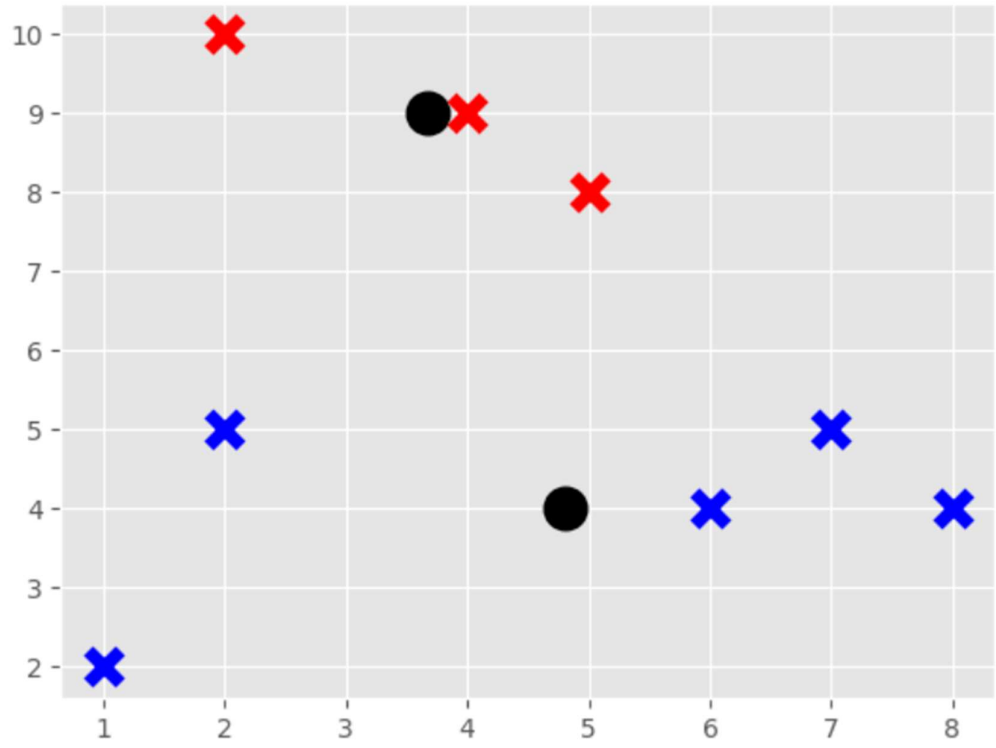
```
            distances = [np.linalg.norm(featureset - self.centroids[centroid]) for centroid in
self.centroids]
            classification = distances.index(min(distances))
            self.classifications[classification].append(featureset)

        prev_centroids = dict(self.centroids)

        # Update centroids based on the mean of points in each cluster
        for classification in self.classifications:
            self.centroids[classification] = np.average(self.classifications[classification],
axis=0)

        # Calculate the distance matrix for each centroid
        dist_matrix = np.zeros((self.k, len(data)))
        for centroid_idx in range (self.k):
            for i, feature_set in enumerate(data):
                dist_matrix[centroid_idx][i] = np.linalg.norm(feature_set -
self.centroids[centroid_idx])
                print(f"Distance Matrix for Iteration {i + 1}:")
                print(dist_matrix)

        # Print the distance matrix
#           print(f"Distance Matrix for Iteration {i + 1}:")
#           print(dist_matrix)

        # Check for convergence based on centroid movement
        optimized = True
        for c in self.centroids:
            original_centroid = prev_centroids[c]
            current_centroid = self.centroids[c]
            if np.sum((current_centroid - original_centroid) / original_centroid * 100.0) >
self.tol:
                optimized = False

        # Break if centroids have converged
        if optimized:
            break

    def predict(self, data):
        # Predict the cluster for a given data point (find the nearest centroid)
        distances = [np.linalg.norm(data - self.centroids[centroid]) for centroid in self.centroids]
        classification = distances.index(min(distances))
        return classification

#main function
model = K_Means()  # Create a K-Means model instance
model.fit(X)  # Fit the model to the data

# Plot the centroids of each cluster
for centroid in model.centroids:
```

```
    plt.scatter(model.centroids[centroid][0], model.centroids[centroid][1],
                marker="o", color="k", s=150, linewidths=5)

# Plot the data points for each cluster
for classification in model.classifications:
    # Assign a different color to each cluster
    color = 'r' if classification == 0 else 'b'
    for featureset in model.classifications[classification]:
        plt.scatter(featureset[0], featureset[1], marker="x", color=color, s=150, linewidths=5)

plt.show()


# Output:
# distance matrix calculation for each iterations
Distance Matrix for Iteration 1:
[[1.94365063 0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 2:
[[1.94365063 4.33333333 0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 3:
[[1.94365063 4.33333333 6.61647775 0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 4:
[[1.94365063 4.33333333 6.61647775 1.66666667 0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 5:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 6:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 7:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
```

```
Distance Matrix for Iteration 8:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 1:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 2:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 3:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2        0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 4:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2        4.00499688 0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 5:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2        4.00499688 2.41660919 0.
  0.         0.        ]]
Distance Matrix for Iteration 6:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2        4.00499688 2.41660919 1.2
  0.         0.        ]]
Distance Matrix for Iteration 7:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2        4.00499688 2.41660919 1.2
  4.29418211 0.        ]]
Distance Matrix for Iteration 8:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2        4.00499688 2.41660919 1.2
  4.29418211 5.06359556]]
Distance Matrix for Iteration 1:
[[1.94365063 0.         0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
```

```
Distance Matrix for Iteration 2:
[[1.94365063 4.33333333 0.         0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 3:
[[1.94365063 4.33333333 6.61647775 0.         0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 4:
[[1.94365063 4.33333333 6.61647775 1.66666667 0.         0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 5:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 0.
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 6:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 7:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 8:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [0.         0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 1:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 0.         0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 2:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 0.         0.         0.         0.
  0.         0.        ]]
Distance Matrix for Iteration 3:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2        0.         0.         0.
  0.         0.        ]]
```

Distance Matrix for Iteration 4:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2   4.00499688 0.   0.
  0.   0.   ]]
Distance Matrix for Iteration 5:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2   4.00499688 2.41660919 0.
  0.   0.   ]]
Distance Matrix for Iteration 6:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2   4.00499688 2.41660919 1.2
  0.   0.   ]]
Distance Matrix for Iteration 7:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2   4.00499688 2.41660919 1.2
  4.29418211 0.   ]]
Distance Matrix for Iteration 8:
[[1.94365063 4.33333333 6.61647775 1.66666667 5.20683312 5.51764845
  7.49073502 0.33333333]
 [6.62117814 2.97321375 3.2   4.00499688 2.41660919 1.2
  4.29418211 5.06359556]]

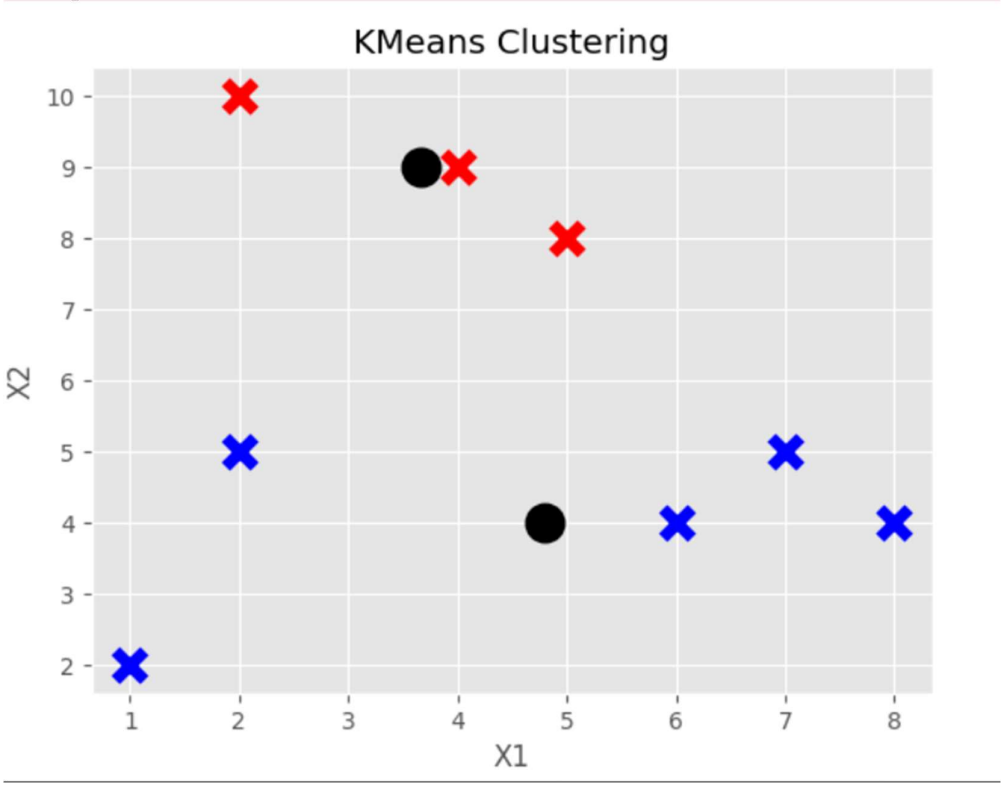


Part 2:

Code:

```
# Using scikit-learn's KMeans
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Plot the centroids of each cluster
plt.scatter(centroids[:, 0], centroids[:, 1], marker="o", color="k", s=150, linewidths=5)

# Plot the data points for each cluster
for cluster_num in range(kmeans.n_clusters):
    color = 'r' if cluster_num == 0 else 'b'
    plt.scatter(X[labels == cluster_num][:, 0], X[labels == cluster_num][:, 1],
                marker="x", color=color, s=150, linewidths=5)

plt.xlabel('X1')
plt.ylabel('X2')
plt.title('KMeans Clustering')
plt.show()
```

Output:



Post lab:

1. Differentiate K-means and K-medoid algorithms with one example

Ans:

1. K-Means:

• Centroid-Based Clustering: In order to reduce the sum of squared distances, K-Means divides data into K clusters according to the centroids.
• Centroid as Representative: The centroid of each cluster—which might not actually be a data point—serves as its representative.
• Sensitive to Outliers: Because squared distances are used, it is sensitive to outliers.
• Computational Efficiency: For large datasets, generally faster and more scalable.
One example would be to identify market segments by classifying customers based on t heir past purchases.

2. K-Medoid:

• Medoids-Based Clustering: Using real data points (medoids) as cluster representatives, K-Medoid clusters data into K groups.
• Using a Medium as a Representative: Real data points, which are more resilient to outliers, are used to represent each cluster.
• Robust to Outliers: Because absolute distances are used, it is less susceptible to outliers.
• Computational Complexity: When dealing with large datasets, computational complexity is higher than with K-Means.
• As an illustration, consider grouping cities according to travel times, with the medoid designating the location with the shortest overall distance to other cities.

3. List the packages/ libraries of python used in experiments.

Ans:

The following packages and libraries for Python were used in the code that you submitt ed:

1. matplotlib.pyplot: This programme is used to create plots and data visualisations, incl uding scatter plots.
2. numpy (np): Used to manipulate arrays and perform numerical operations.
3. matplotlib.style: This specifies the plot style, which is 'ggplot' in this example.
4. KMeans from sklearn.cluster: This K-Means clustering tool was imported from Scikit-Learn.

Conclusion:

We learnt to Implement K-mean clustering algorithm to group similar data objects.