Exp 4: Implementation of Clustering Algorithm (K-Means)

Aim :

To implement the **K-Means clustering algorithm** using Python and visualize the clustered data.

Introduction:

Clustering is an unsupervised learning technique used to group similar data points together.
K-Means is one of the most widely used clustering algorithms. It divides a dataset into K distinct clusters based on the similarity of data points.

The algorithm works by iteratively:

1. Selecting initial centroids.
2. Assigning points to the nearest centroid.
3. Updating centroids as the mean of assigned points.
4. Repeating until centroids do not change.

This experiment demonstrates how K-Means can group data points and visualize clusters.

Procedure:

1. Import required libraries (`random`, `math`, `matplotlib.pyplot`).
2. Define the **Euclidean distance function**.
3. Define the **K-Means function**:
    - Randomly select initial centroids.
    - Assign each point to the nearest centroid.
    - Update centroids based on cluster points.
    - Repeat for a fixed number of iterations.
4. Prepare a sample dataset.

5. Call the K-Means function with **k = 2 or 3**.

6. Print **centroids** and **clusters**.

7. Plot the data points with cluster colors and centroids.

8. Observe the clustering and verify results.

Program Code:

```
import random
import math
import matplotlib.pyplot as plt

# Euclidean distance function
def euclidean(a, b):
    return math.sqrt(sum((a[i] - b[i])**2 for i in range(len(a))))

# K-Means function
def kmeans(data, k, iterations=10):
    centroids = random.sample(data, k)

    for _ in range(iterations):
        clusters = [[] for _ in range(k)]

        # Assign points to nearest centroid
        for point in data:
            distances = [euclidean(point, c) for c in centroids]
            cluster_index = distances.index(min(distances))
            clusters[cluster_index].append(point)

        # Update centroids
        for i in range(k):
            centroids[i] = [sum(dim)/len(dim) for dim in zip(*clusters[i])]

    return centroids, clusters

# Sample data
data = [[1,2], [2,3], [3,3], [8,7], [9,8], [10,9]]

# Run K-Means
centroids, clusters = kmeans(data, k=2)
print("Centroids:", centroids)
print("Clusters:", clusters)
```

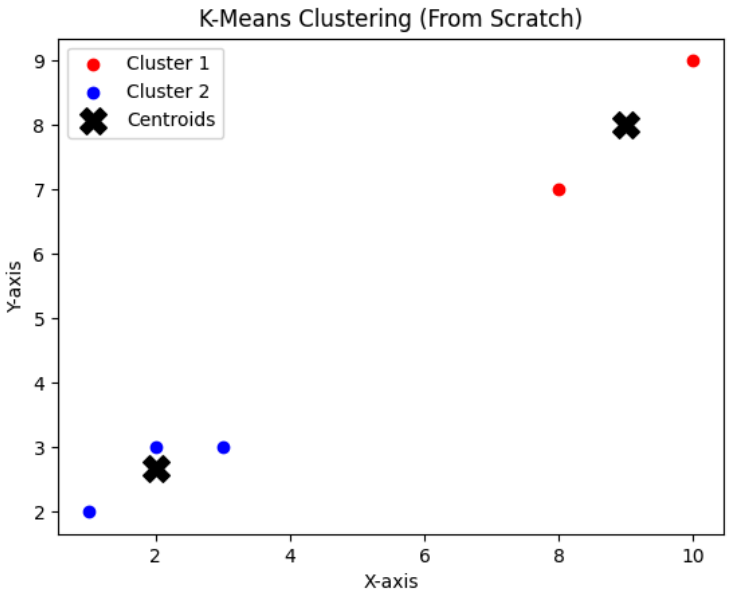
```
# Plotting clusters
colors = ['red', 'blue', 'green', 'purple']
for i, cluster in enumerate(clusters):
    x = [point[0] for point in cluster]
    y = [point[1] for point in cluster]
    plt.scatter(x, y, color=colors[i], label=f'Cluster {i+1}')

# Plot centroids
cx = [c[0] for c in centroids]
cy = [c[1] for c in centroids]
plt.scatter(cx, cy, color='black', marker='X', s=200, label='Centroids')

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("K-Means Clustering")
plt.legend()
plt.show()
```

Output Snapshot:

```
= RESTART: C:/Users/lab703/AppData/Local/Programs/Python/Python310/kmeans cluste
r.py
Centroids: [[9.0, 8.0], [2.0, 2.6666666666666665]]
Clusters: [[[8, 7], [9, 8], [10, 9]], [[1, 2], [2, 3], [3, 3]]]
```



Conclusion:

- K-Means successfully grouped the data into two clusters based on similarity.
- Centroids correctly represent the mean location of each cluster.
- The plotted graph clearly visualizes the separation between clusters.
- K-Means is simple, efficient, and works well with small datasets.

Review Questions:

Q1. What is the K-Means clustering algorithm, and how does it work?
A: K-Means is an unsupervised learning algorithm that divides data into K clusters. It works by selecting initial centroids, assigning points to the nearest centroid, updating centroids as cluster means, and repeating until centroids stabilize.

Q2. How do you determine the optimal number of clusters in K-Means?
A: Common methods include:

- **Elbow method**: Plot SSE (Sum of Squared Errors) vs. K, choose K at “elbow point”.
- **Silhouette score**: Measures how similar a point is to its own cluster vs. other clusters. Higher silhouette → better clustering.

Q3. What are the common distance metrics used in Agglomerative Clustering?
A:

- Euclidean distance
- Manhattan distance
- Cosine distance
- Maximum (Chebyshev) distance

Github link: