Semester	T.E. Semester VI – Computer Engineering
Subject	Data Warehousing and Mining
Subject Professor In-charge	Prof. Kavita Shirsat
Assisting Teachers	Prof. Kavita Shirsat
Laboratory	Lab 312 A

Student Name	Deep Salunkhe	
Roll Number	21102A0014	
Grade and Subject Teacher's Signature		

Experiment Number	05	
Experiment Title	K-Means Algorithm to find clusters	
Resources / Apparatus Required	Hardware: Computer system	Software: Python
Description	**Basic Idea:** • The K-Means algorithm partitions a dataset into K distinct, non-overlapping clusters. • Each data point belongs to the cluster with the nearest mean (centroid). **Steps of the K-Means Algorithm:** 1. **Initialization**: Choose K initial centroids. This can be done randomly or by selecting K data points from the dataset. 2. **Assignment**: Assign each data point to the nearest centroid. The assignment is typically based on a distance metric, such as Euclidean distance or Manhattan distance. 3. **Update Centroids**: Recalculate the centroids for each cluster as the mean (average) of all data points assigned to that cluster. 4. **Repeat**: Steps 2 and 3 are repeated until a stopping criterion is met. Common stopping criteria include a maximum number of iterations or when centroids no longer change significantly. **Key Concepts:**	

- **Centroid**: Each cluster is represented by its centroid, which is the mean of all data points in that cluster.
- **Cluster Variance**: K-Means aims to minimize the within-cluster variance. It tries to ensure that data points within the same cluster are close to each other in terms of distance.

Challenges and Considerations:

- **Initial Centroid Selection**: The choice of initial centroids can affect the final clustering result. Different initializations may lead to different cluster assignments.
- **Number of Clusters (K)**: You often need to specify the number of clusters (K) in advance. Selecting an inappropriate K value can result in suboptimal clustering.
- **Convergence**: K-Means may converge to a local minimum, meaning it might not find the best clustering solution. Running the algorithm multiple times with different initializations can mitigate this issue.
- **Scalability**: For large datasets, K-Means can be computationally expensive. There are variants like Mini-Batch K-Means for handling large datasets.

Use Cases:

- K-Means is commonly used for customer segmentation, image compression, anomaly detection, and recommendation systems.

Advantages:

- Simple and easy to implement.
- Scales well to large datasets.
- Often provides meaningful results.

Disadvantages:

- Requires specifying the number of clusters (K).
- Sensitive to initial centroid selection.
- May not perform well with non-spherical or unevenly sized clusters.

Program	

```
import math

# Define a function to calculate the Manhattan distance between two points
def mh(x, y):
    return abs(x - y)
```

```
# Data points to cluster
data = [22, 9, 12, 15, 10, 27, 35, 18, 36, 11]

# Number of clusters
k = 3

# Lists to store cluster points and centroids at each iteration
all_clusters = []
all_centroids = []

# Initialize cluster centroids
c = [22, 9, 12]
```

```
# Main loop to perform k-means clustering
while True:
    # Store the current clusters
    current_clusters = [[] for _ in range(k)]

    # Assign each data point to the nearest cluster based on Manhattan distance
    for i in data:
        mini = 100000
        for j in c:
            if mh(i, j) < mini:
                mini = mh(i, j)
                temp = j
        current_clusters[c.index(temp)].append(i)

    # Recalculate centroids as the mean of each cluster
    current_centroids = [sum(cluster) / len(cluster) for cluster in current_clusters]

    # Store current clusters and centroids
    all_clusters.append(current_clusters)
    all_centroids.append(current_centroids)

    # If the centroids haven't changed, exit the loop
    if c == current_centroids:
        break

    # Update centroids for the next iteration
    c = current_centroids

# Print all clusters and centroids at each iteration
for i, (clusters, centroids) in enumerate(zip(all_clusters, all_centroids)):
    print(f"Iteration {i + 1}")
    print("Clusters:", clusters)
    print("Centroids:", centroids)
    print()

# Print the final clusters and centroids
print("Final Clusters:", all_clusters[-1])
print("Final Centroids:", all_centroids[-1])
```

Output	Iteration 1 Clusters: [[22, 27, 35, 18, 36], [9, 10], [12, 15, 11]] Centroids: [27.6, 9.5, 12.666666666666666] Iteration 2 Clusters: [[22, 27, 35, 36], [9, 10, 11], [12, 15, 18]] Centroids: [30.0, 10.0, 15.0] Iteration 3 Clusters: [[27, 35, 36], [9, 12, 10, 11], [22, 15, 18]] Centroids: [32.666666666666664, 10.5, 18.333333333333332] Iteration 4 Clusters: [[27, 35, 36], [9, 12, 10, 11], [22, 15, 18]] Centroids: [32.666666666666664, 10.5, 18.333333333333332] Final Clusters: [[27, 35, 36], [9, 12, 10, 11], [22, 15, 18]] Final Centroids: [32.666666666666664, 10.5, 18.333333333333332]
Conclusion:	K-Means is a versatile and widely used clustering algorithm that finds natural groupings in data by iteratively optimizing cluster centroids. While it has its limitations, it remains a valuable tool in data analysis and machine learning.