**Experiment NO: 8** **Date:**

**Aim:** To implement K-Means clustering algorithm.

**Theory:** The K-Means clustering algorithm is a popular unsupervised machine learning technique used for partitioning a dataset into groups or clusters. The goal of K-Means is to assign data points to clusters in a way that minimizes the sum of squared distances between data points and the centroid of their assigned cluster. Here's the theoretical overview of the K-Means clustering algorithm:

**Basics:**

* Data Representation:

The dataset consists of data points, each represented by a set of features.

Clusters are formed based on the similarity of data points in the feature space.

* Objective Function:

K-Means minimizes the sum of squared distances between data points and the centroid of their assigned cluster.

The objective function is to minimize the within-cluster sum of squares (WCSS):

WCSS = ∑(i=1 to k) ∑(j=1 to ni) ||xj – cj||2

Where:

k is the number of clusters.

ni is the number of data points in cluster

xj is the jth data point in the cluster i.

ci is the centroid of cluster i.

**K-Means Algorithm Steps:**

1. Initialization:

Randomly initialize k cluster centroids.

Centroids can be set as randomly selected data points or based on some heuristic.

1. Assignment Step:

Assign each data point to the cluster with the nearest centroid.

The distance metric is typically Euclidean distance.

1. Update Step:

Recalculate the centroids of each cluster by taking the mean of all data points assigned to that cluster.

1. Repeat Steps 2 and 3:

Iteratively repeat the assignment and update steps until convergence.

Convergence occurs when the centroids no longer change significantly or a maximum number of iterations is reached.

**Elbow Method for Determining k:**

The choice of the number of clusters (k) is crucial. The Elbow Method is a common technique to find the optimal k.

WCSS is calculated for different values of k, and the "elbow" in the plot of WCSS against k is considered as the optimal number of clusters.

**Initialization Sensitivity:**

The performance of K-Means can be sensitive to the initial placement of centroids. Random initialization may lead to different results in different runs.

K-Means++ is an improvement that uses a smarter initialization strategy to mitigate this issue.

**Scaling and Standardization:**

K-Means is sensitive to the scale of features. It is recommended to standardize or normalize the features before applying the algorithm.

**Limitations:**

Assumes spherical, equally sized clusters, which may not reflect the true nature of all datasets.

Sensitive to outliers, as they can significantly affect the centroid positions.

**Application Areas:**

Image compression.

Customer segmentation.

Anomaly detection.

Document clustering.

Genetic clustering in bioinformatics.

K-Means is a widely used clustering algorithm due to its simplicity and efficiency. It is effective in partitioning data into distinct groups, and its simplicity makes it computationally efficient for large datasets.

**Code:**

# K-Means Clustering

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the mall dataset with pandas

dataset = pd.read\_csv('C:\P Jeevesh Naidu\college\honours\sem 5\k-clustering\Mall\_Customers.csv')

X = dataset.iloc[:, [3, 4]].values

# Using the elbow method to find the optimal number of clusters

from sklearn.cluster import KMeans

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

# Plot the graph to visualize the Elbow Method to find the optimal number of clusters

plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

# Applying KMeans to the dataset with the optimal number of clusters

kmeans = KMeans(n\_clusters=5, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

y\_kmeans = kmeans.fit\_predict(X)

# Visualising the clusters

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s=100, c='red', label='Cluster 1')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s=100, c='blue', label='Cluster 2')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s=100, c='green', label='Cluster 3')

plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s=100, c='cyan', label='Cluster 4')

plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s=100, c='magenta', label='Cluster 5')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='yellow', label='Centroids')

# Annotate the centroid values

for i, centroid in enumerate(kmeans.cluster\_centers\_):

plt.annotate(f'Centroid {i+1}\n({centroid[0]:.2f}, {centroid[1]:.2f})',

(centroid[0], centroid[1]),

textcoords="offset points",

xytext=(0,10),

ha='center')

plt.title('Clusters of clients')

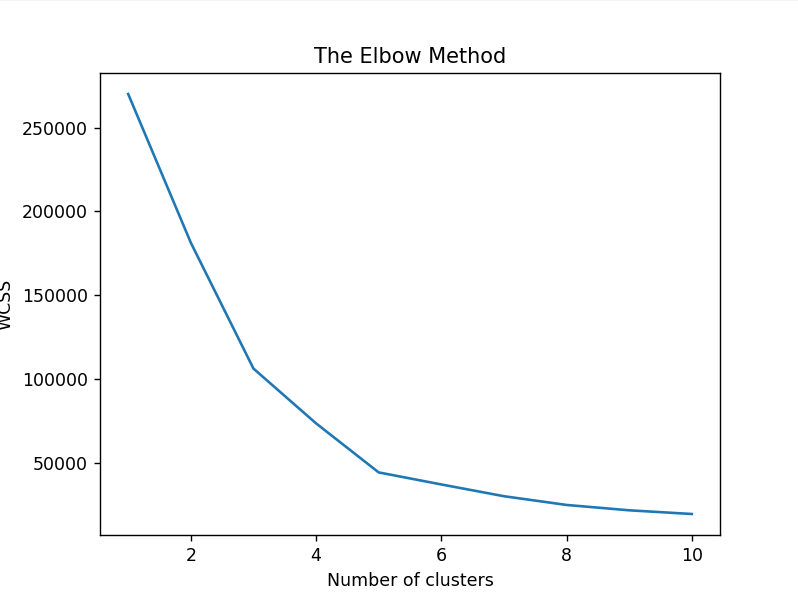
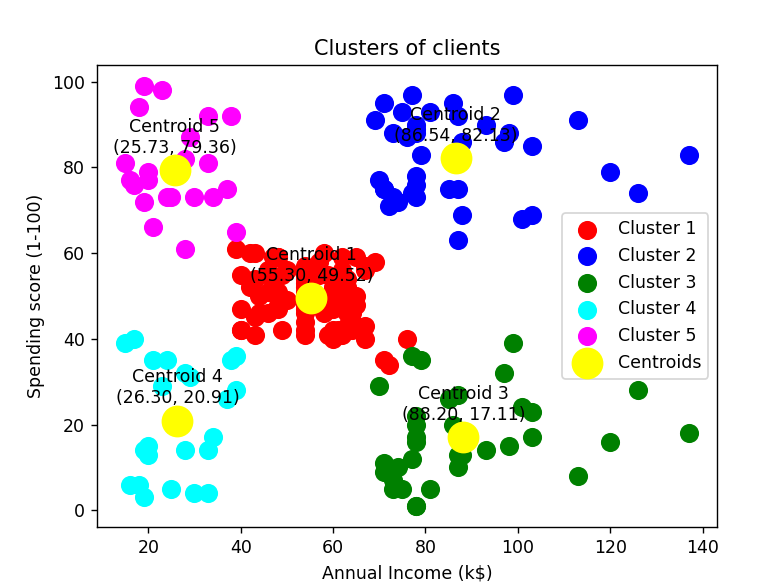
plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending score (1-100)')

plt.legend()

plt.show()

**Output:**

** **

**Conclusion:**

K-Clustering algorithm was studied and successfully implemented.