EXPERIMENT NO. 8

AIM: Study and Implement K-means Clustering to Find Natural Patterns in Data.

Theory:

K-means clustering is an unsupervised machine learning algorithm commonly used for identifying natural patterns in data by grouping similar data points into clusters. It relies on calculating the distance between data points and iteratively refines the clusters to minimize within-cluster variance.

The algorithm works as follows:

1. **Initialization**: Define the number of clusters, kkk, and randomly select kkk data points as initial centroids.
2. **Assignment Step**: Assign each data point to the nearest centroid based on a distance metric (commonly Euclidean distance), creating clusters.
3. **Update Step**: Recalculate the centroid of each cluster by finding the mean of the data points in the cluster.
4. **Iteration**: Repeat the assignment and update steps until convergence, where the centroids stabilize and there is minimal or no change in the assignments.

K-means aims to minimize the **intra-cluster variance** (the sum of squared distances between data points and their cluster centroid) while maximizing **inter-cluster variance** (distance between centroids of different clusters). However, the quality of clustering depends on factors like the number of clusters k, data scaling, and initial centroid selection. The algorithm is relatively efficient but may converge to local minima, so it's typically run multiple times with different initializations to improve results.

Code:

```
# In[1]:
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# In[2]:
dataset = pd.read_csv('Customers.csv')
X = dataset.iloc[:, [3, 4]].values
```

Using the elbow method to find the optimal number of clusters

```
# In[3]:
from sklearn.cluster import KMeans
```

```
wcss = []
for i in range(1, 11):
kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
        kmeans.fit(X)
        wcss.append(kmeans.inertia_)
        plt.plot(range(1, 11), wcss)
        plt.title('The Elbow Method')
        plt.xlabel('Number of clusters')
        plt.ylabel('WCSS')
        plt.show()
```

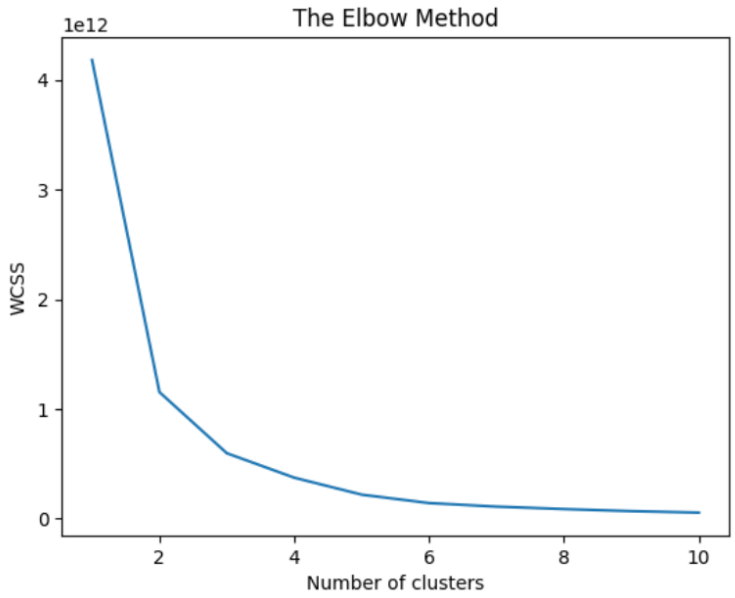
Training the K-Means model on the dataset

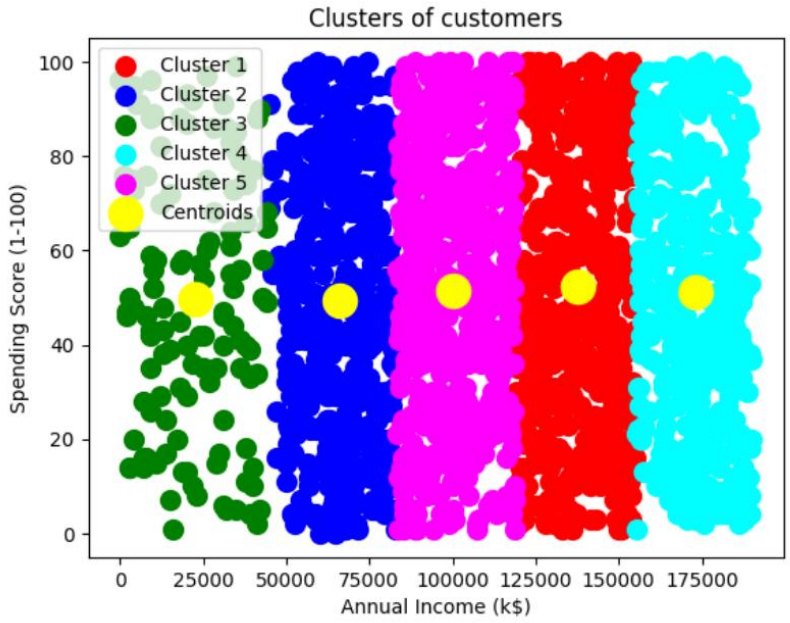
```
#In[4]:
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

Visualising the clusters

```
#In[5]:
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')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Output Snapshots:





Learning Outcome: