EXPERIMENT - 4

AIM:

To understand K-means clustering algorithm and write the code for the same in python.

THEORY:

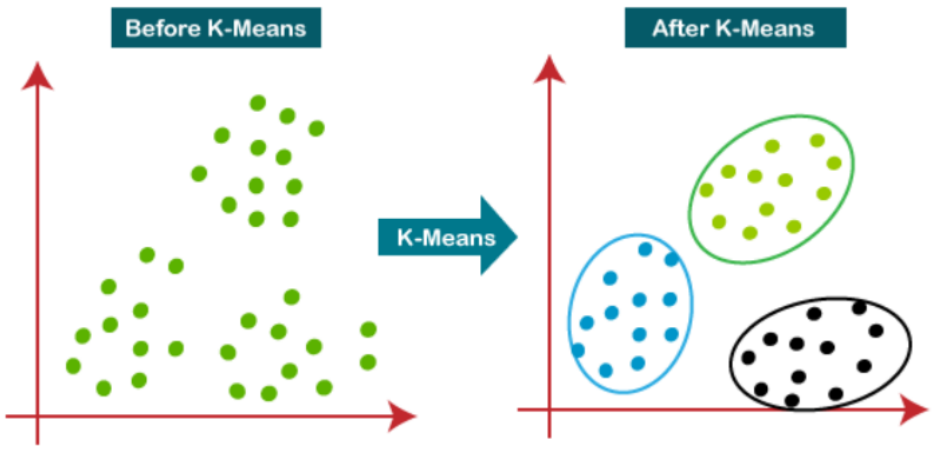
K-Means Clustering is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs to only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.



The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.
Step-2: Select random K points or centroids. (It can be other from the input dataset).
Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.
Step-5: Repeat the third steps, which means re-assign each datapoint to the new closest centroid of each cluster.
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
Step-7: The model is ready.

DATASET:

Mall_Customers.csv

The excel sheet Mall_customers.csv has information about customers shopping in a mall with information about their gender, age, annual income and spending score.

CODE and OUTPUT:

CONCLUSION:

From this experiment, we learn in great detail about K-Means clustering and its implementation in python. We also learn how K-means helps us make complex analysis simpler. For instance, in the mall dataset, we can see that the middle cluster is average earning people with average spending score so they come under the category of normal customers. Bottom right cluster shows the customer has a high income but low spending, so we can categorize them as careful. Bottom left shows the low income and also low spending so they can be categorized as sensible. Top Left shows the customers with low income with very high spending so they can be categorized as careless. Top Right shows the customers with high income and high spending so they can be categorized as targets, and these customers can be the most profitable customers for the mall owner.

60004190013
Aryan Parekh
Computer Engineering
Data Mining and Warehouse

PART C

K-Means Clustering (Elbow Comparison)

Importing the libraries

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

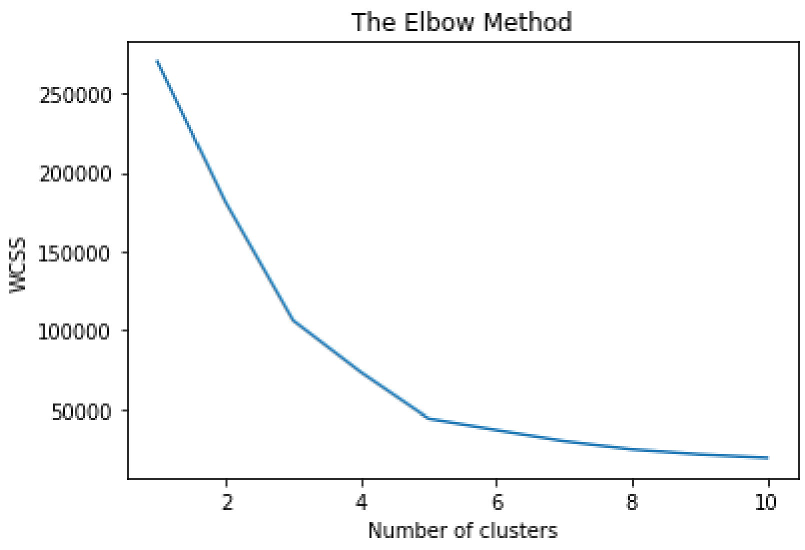
In [2]:

```
dataset = pd.read_csv('Mall_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()
```



Conclusion: The change after n=5 is not significant. Hence, best number of clusters for the given dataset = 5.

PART B

K-Means Clustering (from scratch)

Importing the libraries

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Importing the dataset

In [2]:

```
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values

# setting the number of training examples
m=X.shape[0]
n=X.shape[1]
n_iter=50
```

Code

Creating Random Centroids

In [3]:

```
# computing the initial centroids randomly
K=5
import random

# creating an empty centroid array
centroids=np.array([]).reshape(n,0)

# creating 5 random centroids
for k in range(K):
    centroids=np.c_[centroids,X[random.randint(0,m-1)]]
```

Finding Euclidean distance between centroids and storing the minimum ones

In [4]:

```
output={}

# creating an empty array
euclid=np.array([]).reshape(m,0)

# finding distance between for each centroid
for k in range(K):
       dist=np.sum((X-centroids[:,k])**2,axis=1)
       euclid=np.c_[euclid,dist]

# storing the minimum value we have computed
minimum=np.argmin(euclid,axis=1)+1
```

Regrouping dataset based on minimum values and calculating centroid value

In [5]:

```
# computing the mean of separated clusters
cent={}
for k in range(K):
    cent[k+1]=np.array([]).reshape(2,0)

# assigning of clusters to points
for k in range(m):
    cent[minimum[k]]=np.c_[cent[minimum[k]],X[k]]
for k in range(K):
    cent[k+1]=cent[k+1].T

# computing mean and updating it
for k in range(K):
     centroids[:,k]=np.mean(cent[k+1],axis=0)
```

Repeating the above steps over and over until we reach a convergence

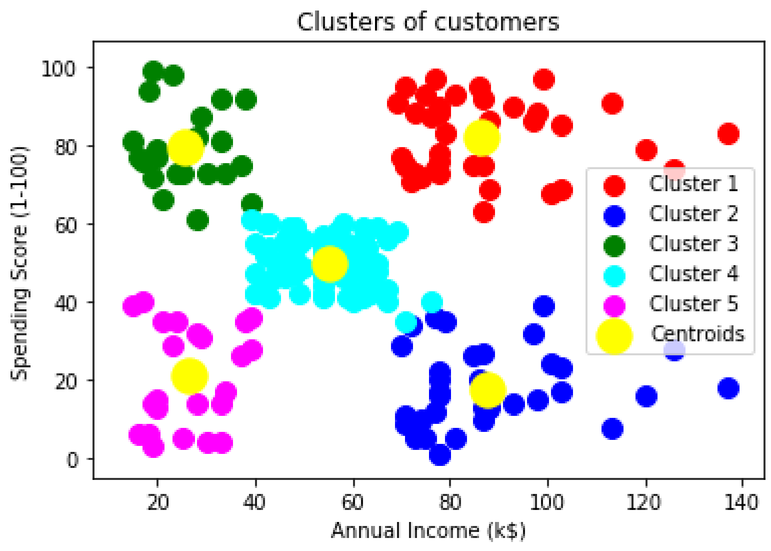
In [6]:

```
# repeating the above steps again and again
for i in range(n_iter):
      euclid=np.array([]).reshape(m,0)
      for k in range(K):
          dist=np.sum((X-centroids[:,k])**2,axis=1)
          euclid=np.c_[euclid,dist]
      C=np.argmin(euclid,axis=1)+1
      cent={}
      for k in range(K):
           cent[k+1]=np.array([]).reshape(2,0)
      for k in range(m):
           cent[C[k]]=np.c_[cent[C[k]],X[k]]
      for k in range(K):
           cent[k+1]=cent[k+1].T
      for k in range(K):
           centroids[:,k]=np.mean(cent[k+1],axis=0)
      final=cent
```

Visualising the clusters

In [7]:

```
plt.scatter(final[1][:,0],final[1][:,1], s=100, c = 'red', label = 'Cluster 1')
plt.scatter(final[2][:,0],final[2][:,1], s=100, c = 'blue', label = 'Cluster 2')
plt.scatter(final[3][:,0],final[3][:,1], s=100, c = 'green', label = 'Cluster 3')
plt.scatter(final[4][:,0],final[4][:,1], s=100, c = 'cyan', label = 'Cluster 4')
plt.scatter(final[5][:,0],final[5][:,1], s=100, c = 'magenta', label = 'Cluster 5')
plt.scatter(centroids[0,:],centroids[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()
```



PART A

K-Means Clustering (inbuilt)

Importing the libraries

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

In [2]:

```
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
```

Training the K-Means model on the dataset

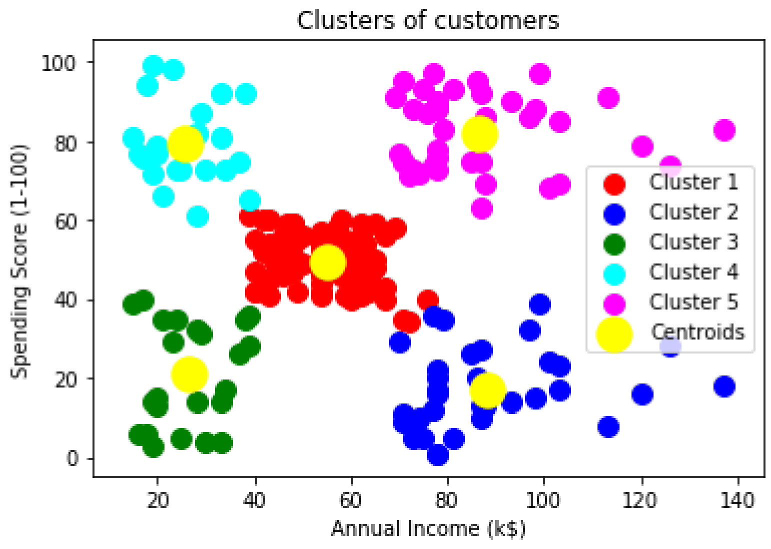
In [4]:

```
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
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluste
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Clus
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yel
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```


Clusters of customers
Spending Score (1-100)
Annual Income (k$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids