**EXPT NO: 9A A python program to implement**

**DATE: 25.10.2024 KNN MODEL .**

# AIM:

TowriteapythonprogramtoimplementKNNModel.

# PROCEDURE:

ImplementingKNNModelusingthemall\_customerdatasetinvolvethefollowing steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

import numpy as np

importmatplotlib.pyplotasplt import pandas as pd

fromsklearn.model\_selectionimporttrain\_test\_split from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

fromsklearn.metricsimportclassification\_report,confusion\_matrix from sklearn.cluster import KMeans

**Step2:LoadtheDataset**

Themall\_customerdatasetcanbeloadedanddisplaythefirstfewrowsofthedataset.

# Load the dataset

dataset = pd.read\_csv('/content/Mall\_Customers.csv')

#Displaythefirstfewrowsofthedataset print(dataset.head())

# Display the dimensions of the dataset print(f"Datasetshape:{dataset.shape}")

#Displaydescriptivestatisticsofthedataset print(dataset.describe())

**Step3:Separatethefeatures(x)andtargetvariable (y)**

# Separate the features (X) and the target variable (y)

X = dataset.iloc[:, [3, 4]].values #Weuse'AnnualIncome'and'Spending Score'

#Standardizethefeatures scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

**Step4:Visualizingtheclusterofcustomer**

#ApplyKMeansclusteringusingtheElbowMethodtofindtheoptimalnumber of clusters

wcss = [] #Within-clustersumofsquares 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\_scaled) wcss.append(kmeans.inertia\_)**  **# Plot the Elbow Method graph plt.plot(range(1, 11), wcss) plt.title('The Elbow Method') plt.xlabel('Numberofclusters') plt.ylabel('WCSS')**  **plt.show()**  **#Fromtheplot,wecanobservethattheoptimalnumberofclustersis5 (elbow point)**  **kmeans=KMeans(n\_clusters=5,init='k-means++',max\_iter=300,n\_init=10, random\_state=0)**  **y\_kmeans = kmeans.fit\_predict(X\_scaled)**  **# Visualizing the clusters of customers** | | | | | | | | |
| **plt.scatter(X\_scaled[y\_kmeans c='red', label='Cluster 1')** | **==** | **0,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **0,** | **1],** | **s=100,** |
| **plt.scatter(X\_scaled[y\_kmeans c='blue', label='Cluster 2')** | **==** | **1,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **1,** | **1],** | **s=100,** |
| **plt.scatter(X\_scaled[y\_kmeans c='green', label='Cluster 3')** | **==** | **2,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **2,** | **1],** | **s=100,** |
| **plt.scatter(X\_scaled[y\_kmeans c='cyan', label='Cluster 4')** | **==** | **3,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **3,** | **1],** | **s=100,** |
| **plt.scatter(X\_scaled[y\_kmeans** | **==** | **4,** | **0],** | **X\_scaled[y\_kmeans** | **==** | **4,** | **1],** | **s=100,** |
| **c='magenta', label='Cluster 5')** | | | | | | | | |

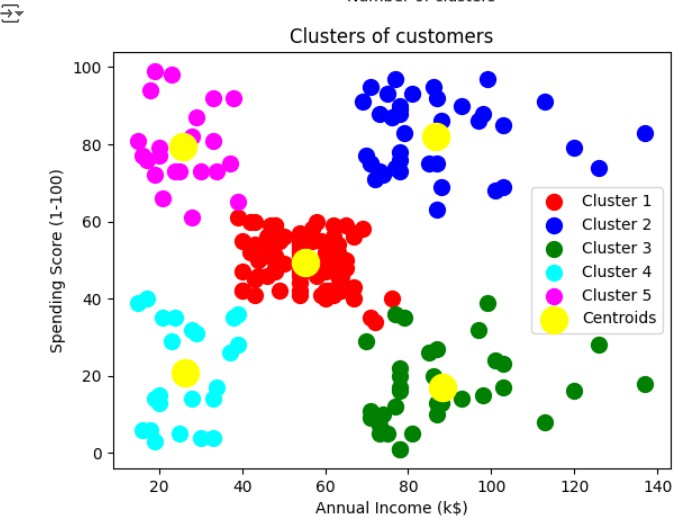
# Plot the centroids

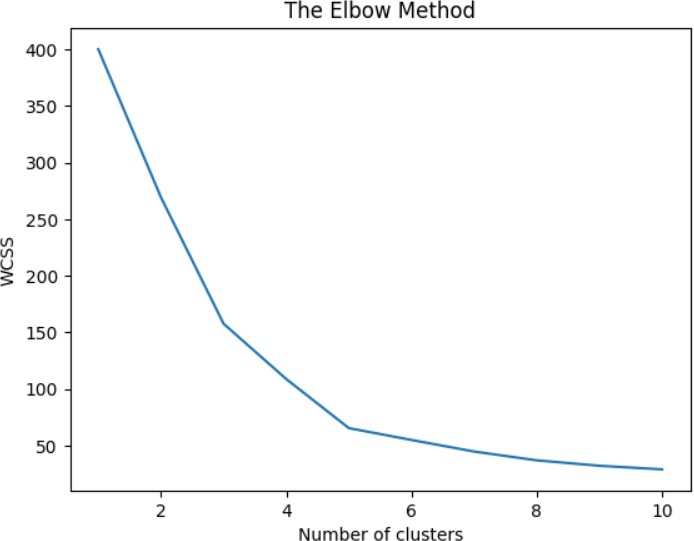
plt.scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1], s=300, c='yellow', label='Centroids')

plt.title('Clusters of customers') plt.xlabel('Annual Income (k$)') plt.ylabel('SpendingScore(1-100)') plt.legend()

plt.show()

# OUTPUT:





**RESULT:**

ThusthepythonprogramtoimplementKNNmodelhasbeensuccessfully implemented and the results have been verified.

**EXPT NO: 9B Apythonprogramtoimplement DATE: 25.10.2024 K-Means Model**

# AIM:

TowriteapythonprogramtoimplementtheK-meansModel.

# PROCEDURE:

ImplementingK-meansModelusingthemall\_customerdatasetinvolvethe following steps:

**Step1:ImportNecessaryLibraries**

First,importthelibrariesthatareessentialfordatamanipulation,visualization,and model building.

import numpy as np importpandasaspd

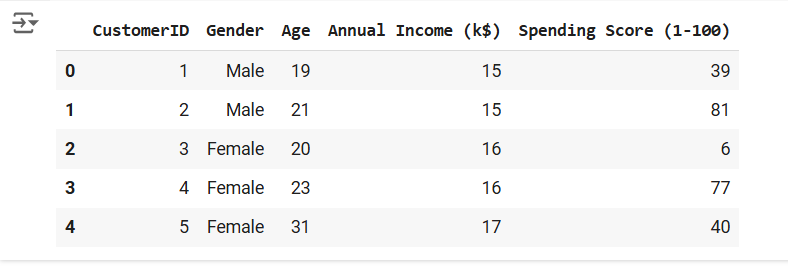
from math import sqrt

**Step2:loadthe Dataset**

data = pd.read\_csv('/content/Mall\_Customers.csv')

data.head(5)

# OUTPUT:

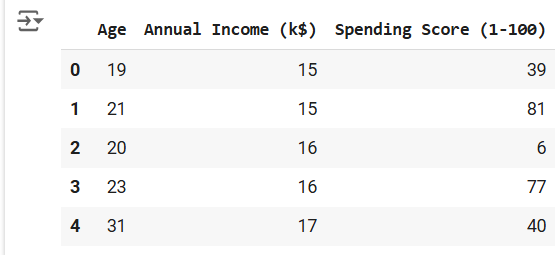


**Step3:Preprocessthedata**

req\_data = data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

req\_data.head(5)

# OUTPUT:



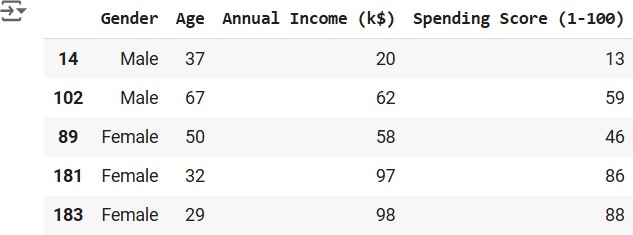
**Step4:Assignthedatapointstoclusters**

shuffle\_index = np.random.permutation(req\_data.shape[0]) #Shufflethe dataset rows

req\_data = req\_data.iloc[shuffle\_index]

req\_data.head(5)

# OUTPUT:



**Step5:Updatetheclusterscenters**

train\_size = int(req\_data.shape[0]\*0.7) #Set70%ofthedatafortraining train\_df = req\_data.iloc[:train\_size,:]

test\_df = req\_data.iloc[train\_size:,:]

train = train\_df.values #Converttraindatatonumpyarray test = test\_df.values # Convert test data to numpy array y\_true = test[:,-1] # The target values for the test set

print('Train\_Shape:',train\_df.shape) print('Test\_Shape: ', test\_df.shape)

from math import sqrt

defeuclidean\_distance(x\_test,x\_train): distance = 0

for i in range(len(x\_test)): #Loopthroughallfeatures distance += (x\_test[i]-x\_train[i])\*\*2

return sqrt(distance)

defget\_neighbors(x\_test,x\_train,num\_neighbors): distances = []

data = []

for i in x\_train: distances.append(euclidean\_distance(x\_test,i)) data.append(i)

distances = np.array(distances)

data = np.array(data)

sort\_indexes = distances.argsort() #Sortdistancesinascendingorder data = data[sort\_indexes] # Sort the data based on sorted distances

return data[:num\_neighbors] #Returntheclosest'num\_neighbors'neighbors

defprediction(x\_test,x\_train,num\_neighbors): classes = []

neighbors=get\_neighbors(x\_test,x\_train,num\_neighbors) for i in neighbors:

classes.append(i[-1]) #The target value is the last column

predicted = max(classes, key=classes.count) #Returnthemostfrequent class (the majority vote)

return predicted

defpredict\_classifier(x\_test): classes = []

neighbors = get\_neighbors(x\_test, req\_data.values, 5) #Predictusing the top 5 neighbors

for i in neighbors: classes.append(i[-1])

predicted = max(classes, key=classes.count) #Returnthemajorityvote print(predicted)

return predicted

defaccuracy(y\_true,y\_pred): num\_correct = 0

for i in range(len(y\_true)):

if y\_true[i] == y\_pred[i]: #Comparetruevaluestopredictedvalues num\_correct += 1

accuracy = num\_correct / len(y\_true) #Calculateaccuracyastheratio of correct predictions

return accuracy

defaccuracy(y\_true,y\_pred): num\_correct = 0

for i in range(len(y\_true)):

ify\_true[i]==y\_pred[i]: num\_correct += 1

returnnum\_correct/len(y\_true) y\_pred = []

for i in test:

y\_pred.append(prediction(i, train, 5)) #Makepredictionsforeachtest instance

#Calculateandprinttheaccuracy acc = accuracy(y\_true, y\_pred)

print(f"Accuracy: {acc \* 1000:.2f}%")

# OUTPUT:



**RESULT:**

Thusthepythonprogramimplementingthek-meansmodelissuccessful.