EXPT NO: 9B **A python program to implement**
DATE: 25.10.2024 **K-Means Model**

AIM:

To write a python program to implement the K-means Model.

PROCEDURE:

Implementing K - means Model using the mall_customer dataset involve the following steps:

Step 1: Import Necessary Libraries

First, import the libraries that are essential for data manipulation, visualization, and model building.

```
import numpy as np

import pandas as pd

from math import sqrt
```

Step 2 : load the Dataset

```
data = pd.read_csv('/content/Mall_Customers.csv')

data.head(5)
```

OUTPUT:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Step 3 : Preprocess the data

```
req_data = data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]



req_data.head(5)
```

OUTPUT :

	Age	Annual Income (k$)	Spending Score (1-100)
0	19	15	39
1	21	15	81
2	20	16	6
3	23	16	77
4	31	17	40

Step 4 : Assign the data points to clusters

```
shuffle_index = np.random.permutation(req_data.shape[0])  # Shuffle the
dataset rows


req_data = req_data.iloc[shuffle_index]

req_data.head(5)
```

OUTPUT :

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
14	Male	37	20	13
102	Male	67	62	59
89	Female	50	58	46
181	Female	32	97	86
183	Female	29	98	88

Step 5 : Update the clusters centers

```
train_size = int(req_data.shape[0]*0.7) # Set 70% of the data for
training

train_df = req_data.iloc[:train_size,:]

test_df = req_data.iloc[train_size:,:]

train = train_df.values # Convert train data to numpy array

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



def euclidean_distance(x_test, x_train):

    distance = 0

    for i in range(len(x_test)): # Loop through all features

        distance += (x_test[i]-x_train[i])**2

    return sqrt(distance)



def get_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() # Sort distances in ascending
order
     data = data[sort_indexes] # Sort the data based on sorted distances
    return data[:num_neighbors] # Return the closest 'num_neighbors'
neighbors
def prediction(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) # Return the most
frequent class (the majority vote)
    return predicted


def predict_classifier(x_test):
    classes = []
     neighbors = get_neighbors(x_test, req_data.values, 5) # Predict using
```

```
the top 5 neighbors

    for i in neighbors:

        classes.append(i[-1])

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

    print(predicted)

    return predicted

def accuracy(y_true, y_pred):

    num_correct = 0

    for i in range(len(y_true)):

        if y_true[i] == y_pred[i]: # Compare true values to predicted
values

            num_correct += 1
    accuracy = num_correct / len(y_true) # Calculate accuracy as the
ratio of correct predictions

    return accuracy

def accuracy(y_true, y_pred):

    num_correct = 0

    for i in range(len(y_true)):

        if y_true[i] == y_pred[i]:

            num_correct += 1

    return num_correct / len(y_true)

y_pred = []
```

```
for i in test:

    y_pred.append(prediction(i, train, 5)) # Make predictions for each
test instance



# Calculate and print the accuracy

acc = accuracy(y_true, y_pred)

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

OUTPUT :

```
Accuracy: 66.67%
```

RESULT :

Thus the python program implementing the k-means model is successful.