A PYTHON PROGRAM TO IMPLEMENT K-MEANS MODEL

**Ex.No.: 9B**

**Date of Experiment: 24/10/2024**

**AIM:-**

To implement a python program using a K-Means Algorithm in a model.

**ALGORITHM:-**

Step1: Import all the other necessary libraries(numpy as np, matplotlib.pyplot as plt and sklearn.tree,pandas as pd and seaborn as sns).

Step2: Select the number K to decide the number of clusters.

Step3: Select random K points or centroids. (It can be different from the input dataset). Step4:

Assign each data point to their closest centroid, which will form the predefined K clusters. Step5: Calculate the variance and place a new centroid of each cluster.

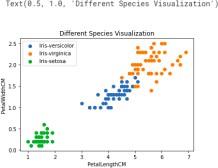
Step6: Repeat the fourth steps, which means assign each datapoint to the new closest centroid of each cluster.

Step7: If any reassignment occurs, then go to step-5 else go to FINISH.

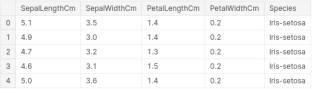
Step8: Train the model and plot the graph using scatterplot() function.

**IMPLEMENTATION:-**

data = pd.read\_csv('../input/k-means-clustering/KNN (3).csv') data.head(5)

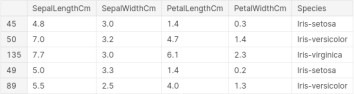


req\_data = data.iloc[:,1:] req\_data.head(5)



shuffle\_index = np.random.permutation(req\_data.shape[0]) #shuffling the row index of our dataset

req\_data = req\_data.iloc[shuffle\_index] req\_data.head(5)



train\_size = int(req\_data.shape[0]\*0.7)

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

test = test\_df.values y\_true = test[:,-1]

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)-1):

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() #argsort() function returns indices by sorting distances data in ascending order

data = data[sort\_indexes] #modifying our data based on sorted indices, so that we can get the nearest neighbors

return data[:num\_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])

predicted = max(classes, key=classes.count) #taking the most repeated class return predicted

def predict\_classifier(x\_test): classes = []

neighbors = get\_neighbors(x\_test, req\_data.values, 5) for i in neighbors:

classes.append(i[-1])

predicted = max(classes, key=classes.count) 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]:

num\_correct+=1

accuracy = num\_correct/len(y\_true) return accuracy

y\_pred = [] for i in test:

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

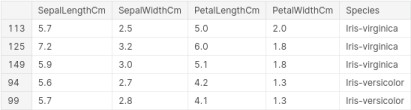


accuracy = accuracy(y\_true, y\_pred)

accuracy



test\_df.sample(5)



**RESULT:-**

Thus the python program to implement the K-Means model has been successfully implemented and the results have been verified and analyzed.