**Ex. No.: 9 b.**

**Date:**  8/11/23

**A PYTHON PROGRAM TO IMPLEMENT K-MEANS MODEL**

**Aim:**

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

**Algorithm:**

1. Import Necessary Libraries:

Import required libraries like numpy, matplotlib.pyplot, and sklearn.cluster.

1. Load and Preprocess Data:

Load the dataset.

Preprocess the data if needed (e.g., scaling).

1. Initialize Cluster Centers:

Choose the number of clusters (K).

Initialize K cluster centers randomly.

1. Assign Data Points to Clusters:

For each data point, calculate the distance to each cluster center. Assign the data point to the cluster with the nearest center.

1. Update Cluster Centers:

Calculate the mean of the data points in each cluster. Update the cluster centers to the calculated means.

1. Repeat Steps 4 and 5:

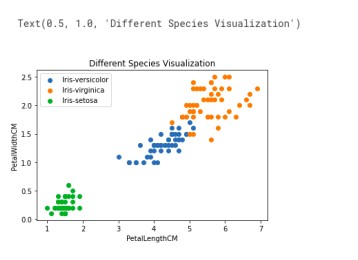
Repeat the assignment of data points to clusters and updating of cluster centers until convergence (i.e., when the cluster assignments do not change much between iterations).

1. Plot the Clusters:

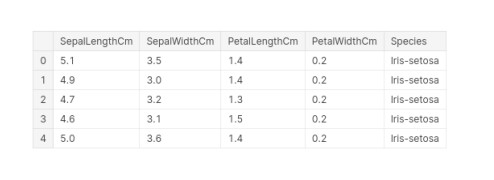
Plot the data points and the cluster centers to visualize the clustering result.

**PROGRAM:**

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

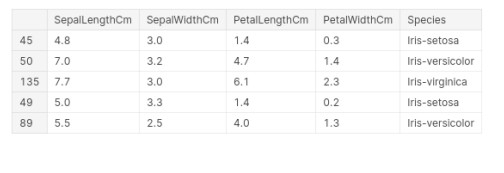


req\_data = data.iloc[:,1:] req\_data.head(5)

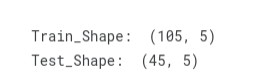


shuffle\_index = np.random.permutation(req\_data.shape[0])

#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):

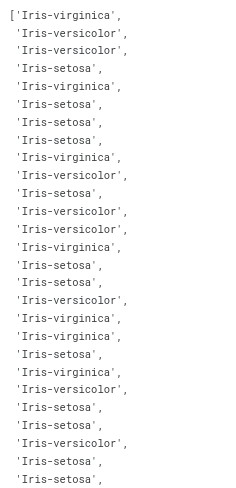
classes = [] neighbors = get\_neighbors(x\_test, x\_train, num\_neighbors) for i in neighbors: classes.append(i[-1]) predicted = max(classes, key=classes.count) #taking the most repeated class return predicted def predict\_classifier(x\_test):

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):

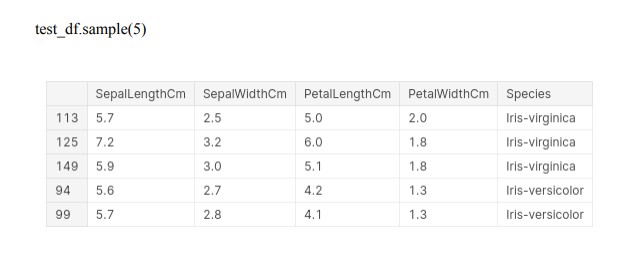
num\_correct = 0 for i in range(len(y\_true)): if y\_true[i]==y\_pred[i]: num\_correct+=1 accuracy = num\_correct/len(y\_true) return accuracy y\_pred = [] for i in test:

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



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





**RESULT:-**

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