# A python program to implement KNN model

Aim:

To implement a python program using a KNN Algorithm in a model.

Algorithm.

1. Import necessary libraries.

* Import necessary libraries: pandas, numpy, train_test_split from sklearn.model_selection, StandardScaler from sklearn.preprocessing.

2. Load and Explore the Dataset.

* Load the dataset using pandas.
* Display the ~~displa~~ dimention of the dataset using df.shape().
* Display the descriptive statistics of the dataset using df.describe().

3. Preprocess the Data:

* Seperate the features(x) and target variable(y).
* Standardize the features using StandardScaler.

4. Train the KNN model:

- Create an instance of KNeighbors classifier with a specified number of neighbors (k).

- Select the k nearest neighbors based on the calculation euclidean distance.

5. Make predictions:

- Use the trained model to make predictions on the test data.
- Evaluate the model.
- Print the confusion matrix and classification report.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('../input/mall-customer/mall_customers.csv')

X = dataset.iloc[:, [3,4]].values
print(dataset)

from sklearn.cluster import KMeans
```

```
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300)
wcss.append(kmeans.inertia_)
plt.plot(range(1,11), wcss)
plt.title('The elbow method')
plt.ylabel('wcss')
plt.show()

kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300,
                n_init=10, random_state=0)

Y_kmeans = kmeans.fit_predict(x)
y_means
type(y_means)
numpy.ndarray
Y_kmeans
plt.scatter(x[Y_kmeans == 0,0]x [Y_kmeans == 0,1],s=100, c='red',
            label='cluster1')

plt.scatter(x[Y_kmeans == 1,0]x [Y_kmeans == 1,1], s=100,
            c='blue', label='cluster2')
```

```
plt.scatter(x[y_kmeans==2,0],x[y_kmeans==2,1],s=100,
            c='green', label='cluster 3')

plt.scatter(x[y_kmeans==3,0]x[y_kmeans==3,1],s=100,c='cyan',
            label='cluster4')
plt.title('cluster of customers')
plt.ylabel('Annual Income(k$)')
plt.legend()
plt.show()

plt.scatter(x[y_kmeans==4,0]x[y_kmeans==4,1],s=100,
            c='magenta', label='cluster5')

plt.scatter(kmeans.cluster_centers_[:,0],kmeans_centers_[:,1],
            s=300,c='yellow',label='centroids')
plt.title('clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score(k-100)')
plt.legend()
plt.show()
```

Result:

Thus the python program to implement KNN ~~model~~ has successfully implemented and the results have been verified and analyzed.

Ex.p.no:8

# A python program to implement K-Means model

Aim:

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

Algorithm:

1. Import Necessary libraries.

2. Load and preprocess data:
 - Load and dataset.
 - Preprocess the data if needed (e.g. Scaling)

3. Initialize Cluster centers:

 Choose the number of clusters (k)
 Initialize K cluster centers randomly.

4. 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.

5. Update cluster Centers:
 - Calculate the mean of the point in each cluster.
 - Update the cluster to the calculated means.

6. Repeat steps 4 and 5:

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

7. Plot the clusters:

* plot the data points and the cluster 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])
req_data=req_data.iloc[shuffle_index] req_data.head(5)
train_size =int(req_data.shape[0]*0.7)
test_df=req_data.iloc[train_size:,:]
test=test_df.values
Y_test =test[:,-1]
Print('Train_shape:',train_df.shape)
Print('Test_shape:',test_df.shape)
```

Train_Shape: (105, 5)

Test_shape: (45, 5)

from math import Sqrt

def ~~en~~ euclidean-distance (x-test, x-train)

distance = 0

for i in range (len (x-test) -1):

return sqrt (distance)

distances = []

data = []

for i in x_train:

distances. append (euclidean_distance (x_test, i))

data. append (i)

data = np. array (distances)

~~So~~

distances data in ascending order

can get the nearest neighbors

return data [: num_~~m~~ neighbors]

classes = []

for i in neighbors:

classes. append (i[-1])

return predicted

classes = []

```
neighbors = get_neighbors (x_test, train_data.values, 5)
for i in neighbors:
class.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

for i in test:
y_pred.append (prediction (i, train, 5))
y_pred
  ['Iris-virginica',
   'Iris-versicolor',
   'Iris-versicolor',
   'Iris-setosa',
   'Iris-setosa',
   Iris-'ser virginica',
   'Iris-viginica',
   'Iris-setosa',
   'Iris-virginica',
   'Iris_virginica',
   'Iris-setosa',
```

```
'Iris-setosa',]

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.