Ex. No.: 9 a.

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, KNeighborsClassifier from sklearn.neighbors, and classification_report and confusion_matrix from sklearn.metrics.

2. Load and Explore the Dataset

● Load the dataset using pandas.

● Display the first few rows of the dataset using df.head().

● Display the dimensions of the dataset using df.shape().

● Display the descriptive statistics of the dataset using df.describe().

3. Preprocess the Data

● Separate the features (X) and the target variable (y).

● Split the data into training and testing sets using train_test_split.

● Standardize the features using StandardScaler.

4. Train the KNN Model

● Create an instance of KNeighborsClassifier with a specified number of neighbors (k).

● For each data point, calculate the Euclidean distance to all other data points.

● Select the K nearest neighbors based on the calculated Euclidean distances.

● Among the K nearest neighbors, count the number of data points in each category.

● Assign the new data point to the category for which the number of neighbors is maximum.

5. Make Predictions

● Use the trained model to make predictions on the test data.

● Evaluate the Model

● Generate the confusion matrix and classification report using the actual and predicted values.

● Print the confusion matrix and classification report.

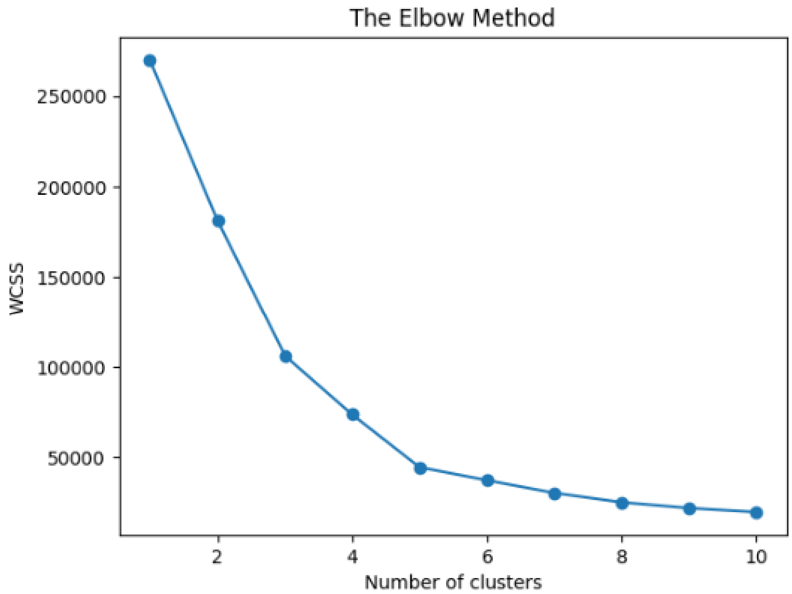
Program:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans

# Load dataset (make sure the path is correct in your environment)
dataset = pd.read_csv('/content/Mall_Customers - Mall_Customers.csv')
print(dataset.head())  # Display first few rows

# Select features for clustering (Annual Income and Spending Score)
X = dataset.iloc[:, [3, 4]].values

# Elbow Method to find the optimal number of clusters
wcss = []
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)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss, marker='o')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
```

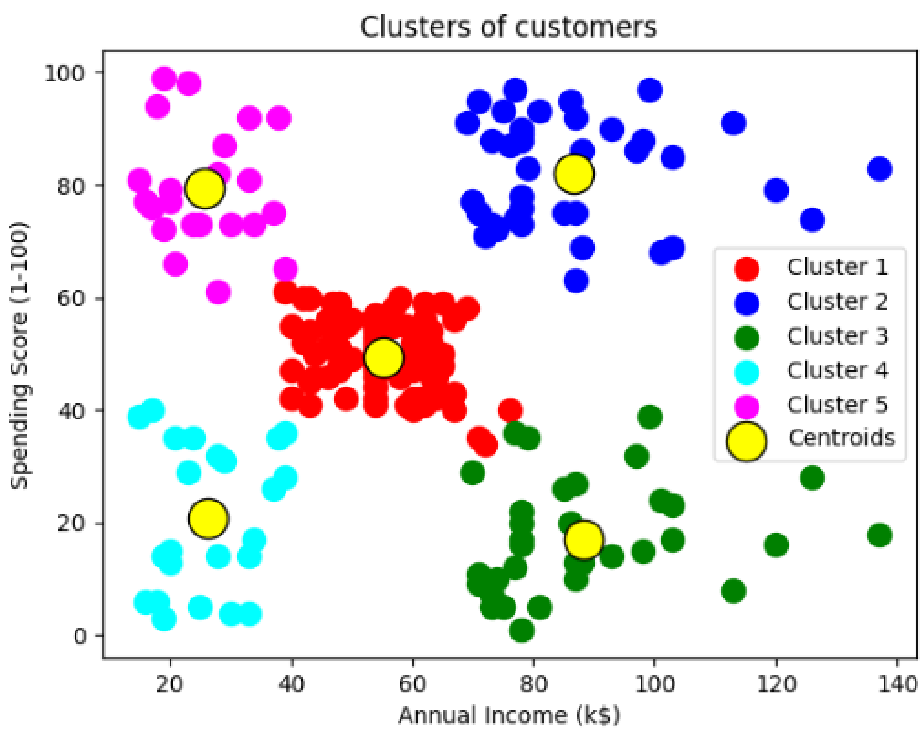
```python
plt.show()

# Fit KMeans with 5 clusters
kmeans = KMeans(
    n_clusters=5,
    init='k-means++',
    max_iter=300,
    n_init=10,
    random_state=0
)
y_kmeans = kmeans.fit_predict(X)

print("Cluster labels:", y_kmeans)
print("Type of labels:", type(y_kmeans))

# Visualizing the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s=100, c='red',
label='Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s=100, c='blue',
label='Cluster 2')
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='Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s=100, c='magenta',
label='Cluster 5')

# Plot centroids
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=300, c='yellow', label='Centroids', edgecolors='black')

plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

OUTPUT:

```
   CustomerID   Genre  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
```




```
Cluster labels: [3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3
 4 3 4 3 4 3 0 3 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 2 1 0 1 2 1 2 1 0 1 2 1 2 1 2 1 2 1 0 1 2 1 2 1
 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1]
Type of labels: <class 'numpy.ndarray'>
```



RESULT:-

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

Ex. No.: 9 b.

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.

2. Load and Preprocess Data:

Load the dataset.

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

3. Initialize Cluster Centers:

Choose the number of clusters (K).

Initialize K cluster centersrandomly.

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 data points in each cluster.

Update the cluster centers to the calculated means.

6. 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).

7. Plot the Clusters:

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

PROGRAM:

```python
import pandas as pd
import numpy as np
from math import sqrt

data = pd.read_csv('/content/IRIS - IRIS.csv')
print("First 5 rows of the dataset:")
print(data.head())

shuffle_index = np.random.permutation(data.shape[0])
req_data = data.iloc[shuffle_index].reset_index(drop=True)
print("\nAfter shuffling, first 5 rows:")
print(req_data.head())

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("\nTrain shape:", train_df.shape)
print("Test shape:", test_df.shape)

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()
    data = data[sort_indexes]
    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)
    return predicted

def accuracy_func(y_true, y_pred):
    num_correct = sum(y_true[i] == y_pred[i] for i in range(len(y_true)))
    accuracy = num_correct / len(y_true)
    return accuracy

y_pred = []
for i in test:
    y_pred.append(prediction(i, train, 5))

print("\nPredicted classes for test samples:")
print(y_pred)

acc = accuracy_func(y_true, y_pred)
print(f"\nAccuracy on test set: {acc*100:.2f}%")

print("\n5 random samples from test data:")
print(test_df.sample(5))
```

OUTPUT:

```
First 5 rows of the dataset:
   sepal_length  sepal_width  petal_length  petal_width      species
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa

After shuffling, first 5 rows:
   sepal_length  sepal_width  petal_length  petal_width          species
0           6.1          2.8           4.0          1.3  Iris-versicolor
1           5.8          2.7           5.1          1.9   Iris-virginica
2           5.4          3.7           1.5          0.2      Iris-setosa
3           6.0          2.9           4.5          1.5  Iris-versicolor
4           6.1          3.0           4.6          1.4  Iris-versicolor

Train shape: (105, 5)
Test shape: (45, 5)

Predicted classes for test samples:
['Iris-setosa', 'Iris-versicolor', 'Iris-setosa', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa'

Accuracy on test set: 97.78%

5 random samples from test data:
     sepal_length  sepal_width  petal_length  petal_width          species
119           5.6          3.0           4.5          1.5  Iris-versicolor
118           5.1          3.5           1.4          0.3      Iris-setosa
111           6.3          2.9           5.6          1.8   Iris-virginica
126           6.5          3.0           5.2          2.0   Iris-virginica
110           7.4          2.8           6.1          1.9   Iris-virginica
```

RESULT:-

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