20MCA241

Data Science Lab

Lab Record

Submitted by:

Akshay Murali

RMCA A

Roll no: 07

Program no: 01 Date: 24-11-2021

Aim: Perform all 8 matrix operations using Python using Numpy

Program:

```
import numpy as mato

print("Matrix Operations")
print("################")
arr1 = mato.array([[10, 15], [5, 20]])
arr2 = mato.array([[7, 5], [3, 2]])

print("Operations with Numpy")
print("Added = ", mato.add(arr1, arr2))
print("Subtract = ", mato.subtract(arr1, arr2))
print("Multiplied = ", mato.multiply(arr1, arr2))
print("Divided = ", mato.divide(arr1, arr2))

print("Dot = ", mato.dot(arr1, arr2))
print("Sum = ", mato.sum(arr1))
print("Sum = ", mato.sum(arr1))
print("Sum of rows= ", mato.sum(arr2, axis=1))
print("Sum of cols= ", mato.sum(arr2, axis=0))

print("Transpose of array1", arr1.T)
print("Transpose of array2", arr2.T)
print("Sqrt of array1", mato.sqrt(arr1))
```

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:/Users/ajcemca/PycharmProjects/Anilect/addmatrix.py
Matrix Operations
################
Operations with Numpy
Added =  [[17 20]
 [ 8 22]]
Subtract =  [[ 3 10]
 [ 2 18]]
Multiplied =  [[70 75]
 [15 40]]
Divided =  [[ 1.42857143  3.        ]
 [ 1.66666667 10.        ]]
Dot =  [[115  80]
 [ 95  65]]
Sum =  50
Sum =  50
Sum of rows=  [12  5]
Sum of cols=  [10  7]
Transpose of array1 [[10  5]
 [15 20]]
Transpose of array2 [[7 3]
 [5 2]]
Sqrt of array1 [[3.16227766 3.87298335]
 [2.23606798 4.47213595]]

Process finished with exit code 0
```

Result: The program has been executed and output verified

Program no: 02 Date: 01-12-2021

Aim: Perform SVD (Singular Value Decomposition) in Python

Program:

```
from numpy import array
from scipy.linalg import svd

Ar = array([[10, 20, 30, 40, 50], [15, 20, 25, 30, 35], [50, 40, 30, 20, 10]])
print(Ar)
i, j, k = svd(Ar)

print("\nDecomposition: ", i)
print("\nInverse Matrix: ", j)
print("\nTranspose of matrix", k)
```

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:/Users/ajcemca/PycharmProjects/Anilect/svd.py
[[10 20 30 40 50]
 [15 20 25 30 35]
 [50 40 30 20 10]]

Decomposition:  [[-0.63018567 -0.54861573 -0.54944226]
 [-0.51671457 -0.23186369  0.82416338]
 [-0.57954471  0.80328078 -0.13736056]]

Inverse Matrix:  [1.10469408e+02 4.65994629e+01 4.91043299e-15]

Transpose of matrix [[-0.38951789 -0.41748928 -0.44546066 -0.47343205 -0.50140344]
 [ 0.66953403  0.35454577  0.03955751 -0.27543074 -0.590419  ]
 [-0.38223409  0.33080407  0.58267801 -0.62883185  0.09758387]
 [-0.49419597  0.42632677  0.08789984  0.52200386 -0.54203451]
 [-0.09832317  0.63938576 -0.6728744  -0.17911579  0.3109276 ]]

Process finished with exit code 0
```

Result: The program has been executed and output verified

Program no: 03 Date: 01-12-2021

Aim: Program to implement K-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm.

Program:

```
from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris

from sklearn.metrics import accuracy_score


irisData = load_iris()

i = irisData.data

j = irisData.target


i_train, i_test, j_train, j_test = train_test_split(

    i, j, test_size=0.7, random_state=30

)


knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(i_train, j_train)

print(knn.predict(i_test))


# finding Accuracy of algorithm

k = knn.predict(i_test)

l = accuracy_score(j_test, k)

print("Accuracy is", l)
```

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:/Users/ajcemca/PycharmProjects/Anilect/knn.py
[0 0 0 2 1 1 2 2 1 2 0 2 1 1 0 1 0 0 0 2 2 0 0 0 2 2 2 2 0 1 2 1 2 2 2 2 1
 2 1 2 2 2 0 1 2 1 1 1 1 1 0 1 2 1 0 2 0 1 1 1 0 1 0 1 0 2 2 2 0 2 1 2 0 0
 1 0 2 2 2 1 0 1 0 1 1 1 2 0 1 0 1 2 1 0 0 0 2 2 0 1 1 1 0 0 0]

Process finished with exit code 0
```

With Accuracy

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:/Users/ajcemca/PycharmProjects/Anilect/knn.py
[0 0 0 2 1 1 2 2 1 2 0 2 1 1 0 1 0 0 0 2 2 0 0 0 2 2 2 2 0 1 2 1 2 2 2 2 1
 2 1 2 2 2 0 2 2 1 1 1 1 1 0 1 2 1 0 2 0 1 1 1 0 1 0 1 0 2 2 2 0 2 2 2 0 0
 1 0 2 2 2 1 0 1 0 1 1 1 2 0 1 0 1 2 1 0 0 0 2 2 0 1 1 1 0 0 0]
Accuracy is 0.9428571428571428

Process finished with exit code 0
```

Result: The program has been executed and output verified

Program no: 04 Date: 01-12-2021

Aim: Program to implement K-NN Classification using any random dataset without using in-built packages

Program:

```
from math import sqrt


def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1) - 1):
        distance += (row1[i] - row2[i]) ** 2
    return sqrt(distance)


# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))

    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors


def predict_classification(train, test_row, num_neighbors):
```

```
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    # print(set(output_values))
    prediction = max(set(output_values), key=output_values.count)
    return prediction

dataset = [[2.7810836, 2.550537003, 0],
        [1.465489372, 2.362125076, 0],
        [3.396561688, 4.400293529, 0],
        [1.38807019, 1.850220317, 0],
        [3.06407232, 3.005305973, 0],
        [7.627531214, 2.759262235, 1],
        [5.332441248, 2.088626775, 1],
        [6.922596716, 1.77106367, 1],
        [8.675418651, -0.242068655, 1],
        [7.673756466, 3.508563011, 1]]
prediction = predict_classification(dataset, dataset[0], 3)
print("Expected %d, Got %d." % (dataset[0][-1], prediction))
```

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\pyth
Expected 0, Got 0.

Process finished with exit code 0
```

Result: The program has been executed and output verified

Program no: 05 Date: 08-12-2021

Aim: Program to implement Naïve Bayes algorithm using any standard dataset available in public domain and find the accuracy of the algorithm.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# importing dataset
dataset = pd.read_csv("social_network_ads.csv")
a = dataset.iloc[:, [2, 3]].values
b = dataset.iloc[:, -1].values

# splitting into test and train dataset
from sklearn.model_selection import train_test_split
a_train, a_test, b_train, b_test = train_test_split(a, b, test_size=0.20, random_state=0)

# Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
a_train = sc.fit_transform(a_train)
a_test = sc.transform(a_test)
print(a_train)
print(a_test)

# training the naive bayes model on the training set
```

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(a_train, b_train)

# predicting the test set results
b_pred = classifier.predict(a_test)
print(b_pred)

# making confusion matrix
from sklearn.metrics import confusion_matrix, accuracy_score
ac = accuracy_score(b_test, b_pred)
co = confusion_matrix(b_test, b_pred)
print(ac)
print(co)
```

Output:

```
 [-1.09058306e+00  5.52551726e-01]
 [-1.96547978e+00  3.49747226e-01]
 [ 3.67578135e-01  2.62831011e-01]
 [ 1.73156642e-01 -2.87638347e-01]
 [ 1.43689635e+00 -1.04091221e+00]
 [ 8.53631867e-01  1.07404901e+00]]
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0
 0 0 0 0 1 1]
0.9125
[[55  3]
 [ 4 18]]

Process finished with exit code 0
```

Result: The program has been executed and output verified

Program no: 06 Date: 08-12-2021

Aim: Program to implement Linear and Multiple regression techniques using any standard dataset available in public

Program: (Build-in Func)

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([10,20,30,40,50,60]).reshape(-1,1)
y = np.array([5,10,15,20,25,30])
print("Linear Regression")
print("Array 1: ", x)
print("Array 2: ", y)
model = LinearRegression()
model.fit(x,y)
r_sq = model.score(x,y)
print("Coefficient of determination: ",r_sq)
print("Intercept: ",model.intercept_)
print("Slope: ",model.coef_)
print("Predicted response: ", y_pred,sep="\n")
plt.plot(x,y_pred, color = "g")
plt.title('Linear Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:\Users\ajcemc
Linear Regression
Array 1:  [[10]
 [20]
 [30]
 [40]
 [50]
 [60]]
Array 2:  [ 5 10 15 20 25 30]
Coefficient of determination:  1.0
Intercept:  -3.552713678800501e-15
Slope:  [0.5]
```

Result: The program has been executed and output verified

Program:

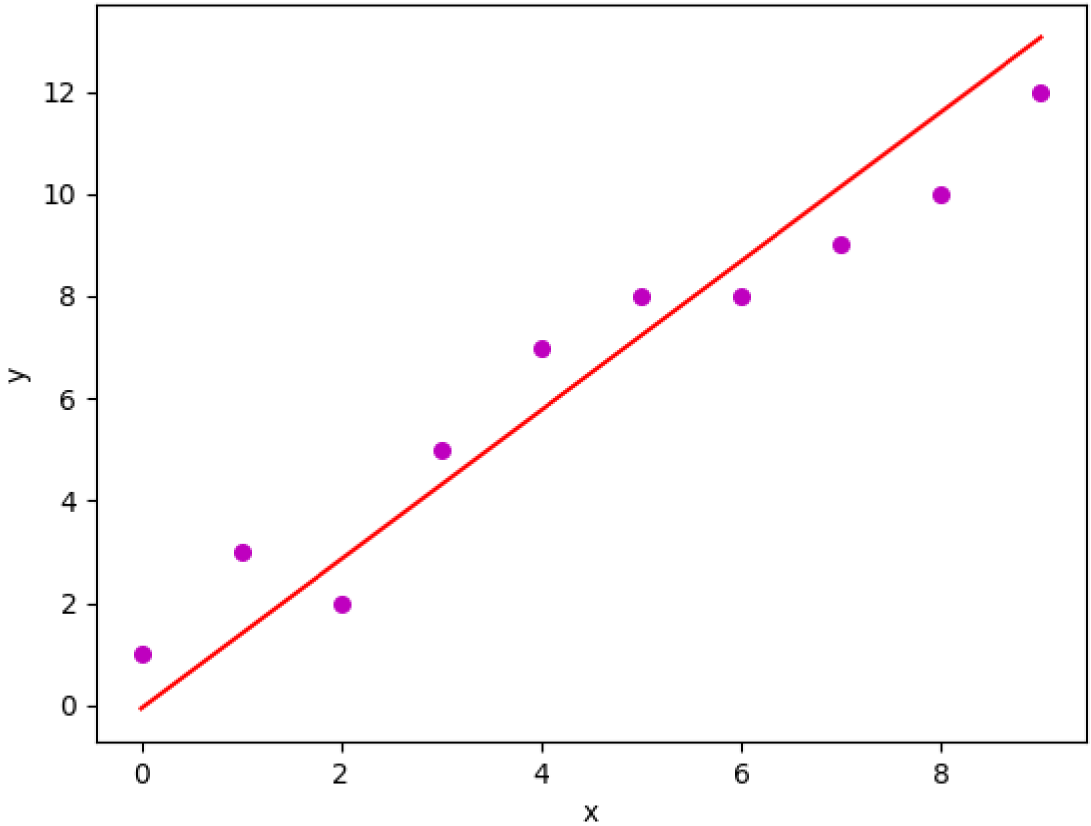
```
import numpy as np

import matplotlib.pyplot as plt


# A basic implementation of linear regression with one variable

# Part of Cosmos by OpenGenus Foundation

def estimate_coef(x, y):

    # number of observations/points

    n = np.size(x)


    # mean of x and y vector

    m_x, m_y = np.mean(x), np.mean(y)


    # calculating cross-deviation and deviation about x

    SS_xy = np.sum(y * x - n * m_y * m_x)

    SS_xx = np.sum(x * x - n * m_x * m_x)


    # calculating regression coefficients
```

```
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1 * m_x
    return b_0, b_1

def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color="m", marker="o", s=30)

    # predicted response vector
    y_pred = b[0] + b[1] * x

    # plotting the regression line
    plt.plot(x, y_pred, color="r")
    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
    # function to show plot
    plt.show()

def main():
    # observations
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients are:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))
```

```
    # plotting regression line
    plot_regression_line(x, y, b)


if __name__ == "__main__":
    main()
```

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:/Users/a
Estimated coefficients are:
b_0 = -0.05862068965517242
b_1 = 1.457471264367816

Process finished with exit code 0
```



Result: The program has been executed and output verified

Program no: 07 Date: 15-12-2021

Aim: Program to implement Linear and Multiple regression techniques using any standard dataset available in public domain and evaluate

Program:

```
import pandas

df = pandas.read_csv("cars.csv")

x = df[['Weight', 'Volume']]

y = df['CO2']

from sklearn import linear_model

regr = linear_model.LinearRegression()

regr.fit(x, y)

predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:/Users/ajcemca/P
C:\Users\ajcemca\PycharmProjects\Anilect\venv\lib\site-packages\sklearn\base.py:445
  warnings.warn(
[107.2087328]

Process finished with exit code 0
```

Result: The program has been executed and output verified

Program no: 08 Date: 15-12-2021

Aim: Program to implement Linear and Multiple regression techniques using cars dataset available in public domain and evaluate and find accuracy

Program:

```
import pandas as pd


df = pd.read_csv("cars.csv")
X = df[['Weight', 'Volume']]
y = df['CO2']
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(X, y)
predictedCO2 = regr.predict([[2300, 1300]])
print(predictedCO2)
```

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:/Users/ajcemca/P
C:\Users\ajcemca\PycharmProjects\Anilect\venv\lib\site-packages\sklearn\base.py:445
  warnings.warn(
[107.2087328]

Process finished with exit code 0
```

Result: The program has been executed and output verified

Program no: 09 Date: 15-12-2021

Aim: Program to implement multiple linear regression techniques using boston dataset available in the public domain and evaluate accuracy and plotting point.

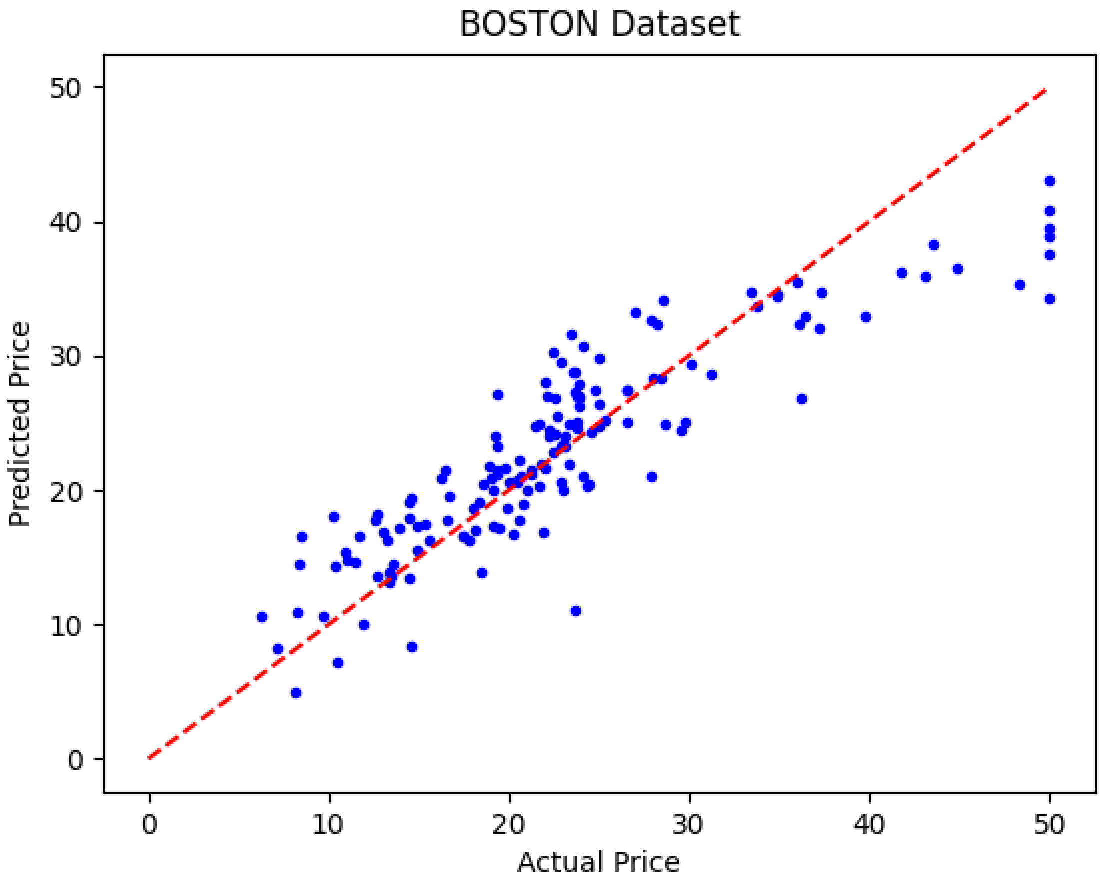
Program:

```
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

boston = datasets.load_boston(return_X_y=False)
X = boston.data
y = boston.target

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
reg = linear_model.LinearRegression()
reg.fit(X_train, y_train)
predicted = reg.predict(X_test)

# Regression coefficient
print('Coefficients are:\n', reg.coef_)

# Intecept
print('\nIntercept : ', reg.intercept_)
```

```
# variance score: 1 means perfect prediction
print('Variance score: ', reg.score(X_test, y_test))

# Mean Squared Error
print("Mean squared error: %.2f" % mean_squared_error(y_test, predicted))

# Original data of X_test
expected = y_test

# Plot a graph for expected and predicted values
plt.title('ActualPrice Vs PredictedPrice (BOSTON Housing Dataset)')
plt.scatter(expected, predicted, c='b', marker='.', s=36)
plt.plot([0, 50], [0, 50], '--r')
plt.xlabel('Actual Price(1000$)')
plt.ylabel('Predicted Price(1000$)')
plt.show()
```

Output:

```
Coefficients are:
 [-9.85424717e-02  6.07841138e-02  5.91715401e-02  2.43955988e+00
 -2.14699650e+01  2.79581385e+00  3.57459778e-03 -1.51627218e+00
  3.07541745e-01 -1.12800166e-02 -1.00546640e+00  6.45018446e-03
 -5.68834539e-01]
Variance score:  0.7836295385076291

Process finished with exit code 0
```



Result: The program has been executed and output verified

Program no: 10 Date: 22-12-2021

Aim: Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import plot_tree

df = sns.load_dataset('iris')
print(df.head())
print(df.info())
df.isnull().any()
print(df.shape)
# Let's plot pair plot to visualise the attributes all at once
sns.pairplot(data=df, hue="species")
plt.savefig('pne.png')

# Correction matrix
sns.heatmap(df.corr())
```

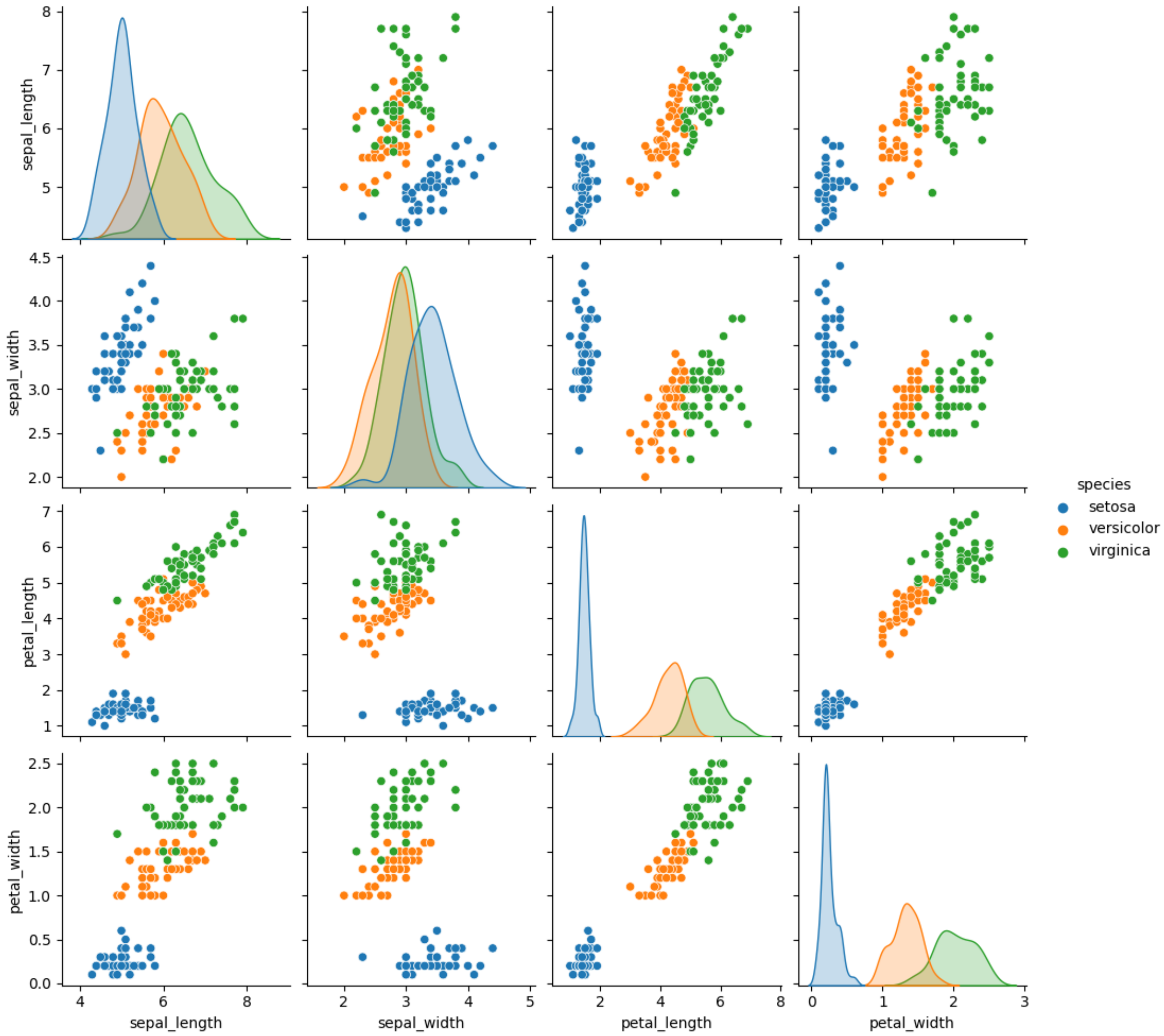
```
plt.savefig('one.png')

target = df['species']
df1 = df.copy()
df1 = df1.drop('species', axis=1)
print(df1.shape)
print(df1.head())
# Defining the attributes
x = df1
print(target)
# label encoding
le = LabelEncoder()
target = le.fit_transform(target)
print(target)

y = target
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

print("Training split input- ", X_train.shape)
print("Testing split input- ", X_test.shape)
# Defining the decision tree algorithm
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)

print('Decision Tree Classifier Created')
y_pred = dtree.predict(X_test)
print('Classification report - \n', classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
```

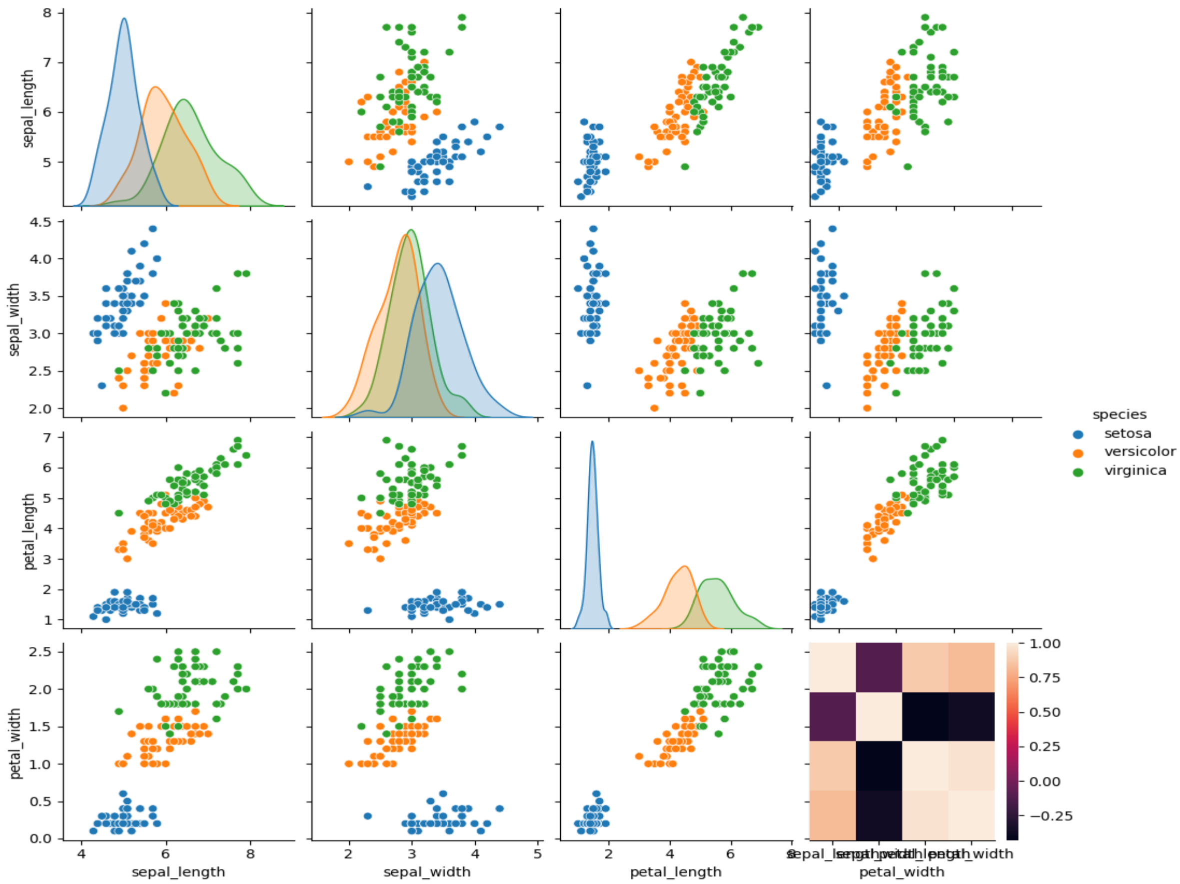
```
plt.figure(figsize=(5, 5))

sns.heatmap(data=cm, linewidth=.5, annot=True, square=True, cmap='Blues')

plt.ylabel('Actual label')

plt.xlabel('Predicted label')

all_sample_title = 'Accuracy score: {0}'.format(X_test, y_test)

plt.title(all_sample_title, size=15)

plt.savefig('two.png')


plt.figure(figsize=(20, 20))

dec_tree = plot_tree(decision_tree=dtree, feature_names=df1.columns,

            class_names=['setosa', 'vercicolor', 'verginica'], filled=True, precision=4,
rounded=True)

plt.savefig('tree.png')
```

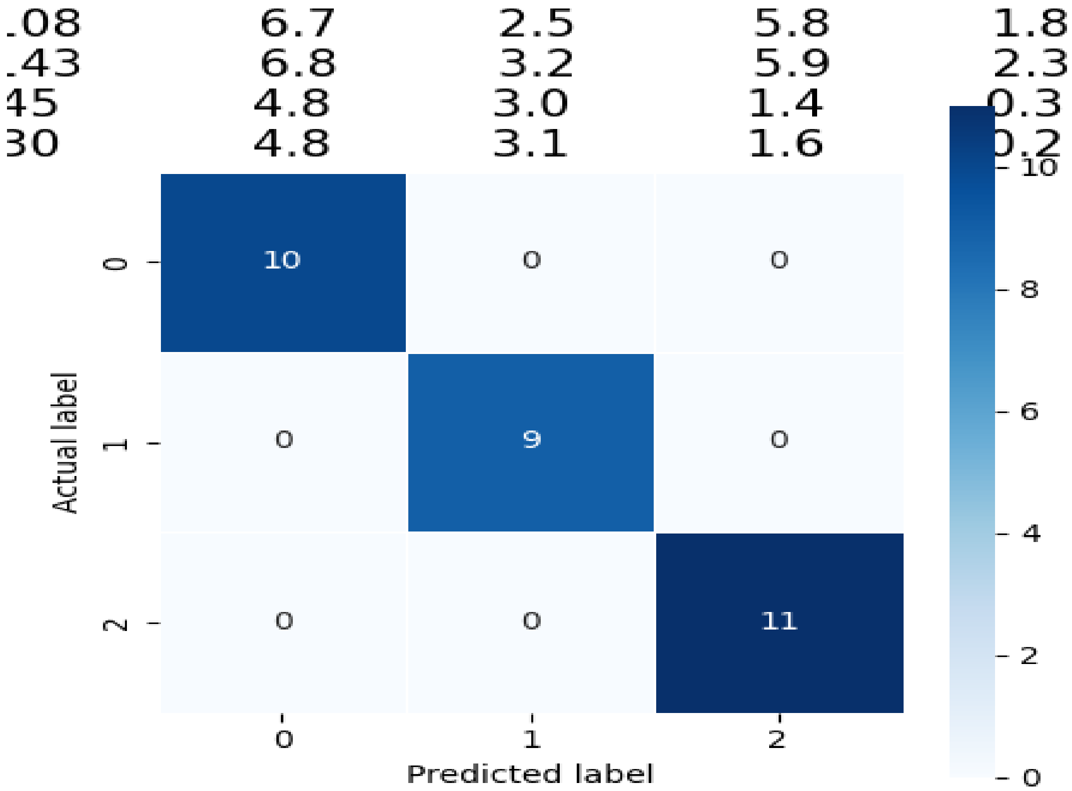
Output:

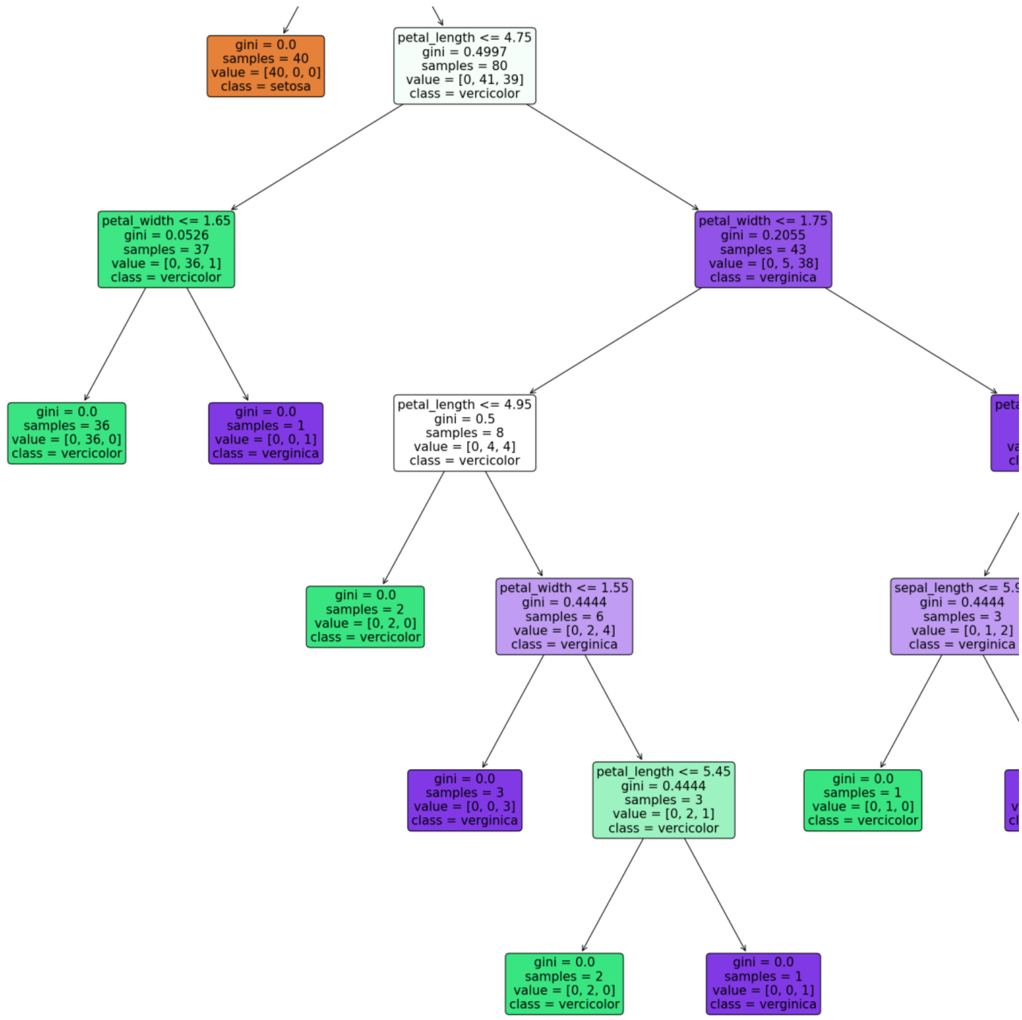
```
"C:\Anilect Jose\pythonProject\Scripts\python.exe" C:/Users/Admin/PycharmProjects/pythonProject/Anilect/dectree.py
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
(150, 5)
(150, 4)
   sepal_length  sepal_width  petal_length  petal_width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
0        setosa
1        setosa
```

```
2          setosa
3          setosa
4          setosa
         ...
145     virginica
146     virginica
147     virginica
148     virginica
149     virginica
Name: species, Length: 150, dtype: object
[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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Training split input-  (120, 4)
Testing split input-  (30, 4)
Decision Tree Classifier Created
Classification report -
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```




sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica


.08 6.7 2.5 5.8 1.8
.43 6.8 3.2 5.9 2.3
45 4.8 3.0 1.4 0.3
30 4.8 3.1 1.6 0.2
10
0
0
0
9
0
0
0
11
Actual label
Predicted label
0
1
2
0
1
2
10
8
6
4
2
0



Result: The program has been executed and output verified

Program no: 11 Date: 05-01-2022

Aim: Program to implement k-means clustering technique using any standard dataset available in the public domain.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd  # Importing the dataset


dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
print(X)


from sklearn.cluster import KMeans
wcss_list = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(X)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters=5, init="k-means++", random_state=42)
y_predict = kmeans.fit_predict(X)
print(y_predict)
```

plt.scatter(X[y_predict == 0, 0], X[y_predict == 0, 1], s=60, c='red', label='Cluster1')

plt.scatter(X[y_predict == 1, 0], X[y_predict == 1, 1], s=60, c='blue', label='Cluster2')

plt.scatter(X[y_predict == 2, 0], X[y_predict == 2, 1], s=60, c='green', label='Cluster3')

plt.scatter(X[y_predict == 3, 0], X[y_predict == 3, 1], s=60, c='violet', label='Cluster4')

plt.scatter(X[y_predict == 4, 0], X[y_predict == 4, 1], s=60, c='yellow', label='Cluster5')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=100, c='black', label='Centroids')

plt.xlabel('Annual Income (k$)')

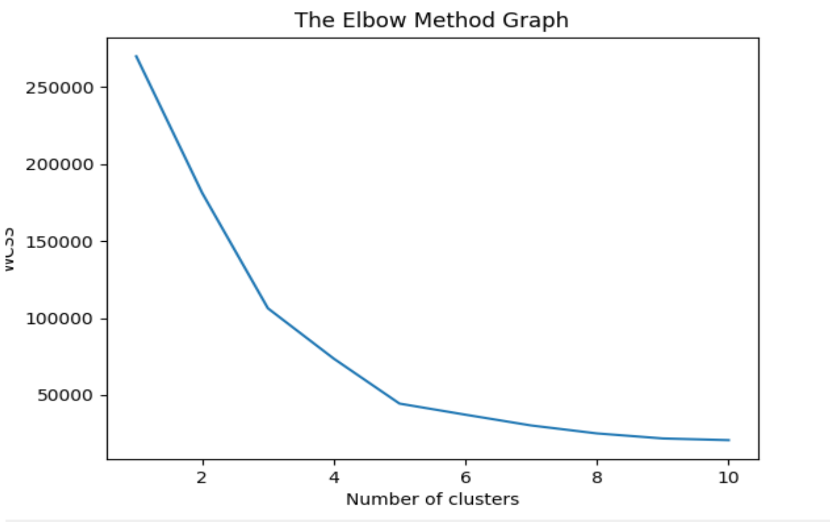
plt.ylabel('Spending Score (1-100)')

plt.legend()

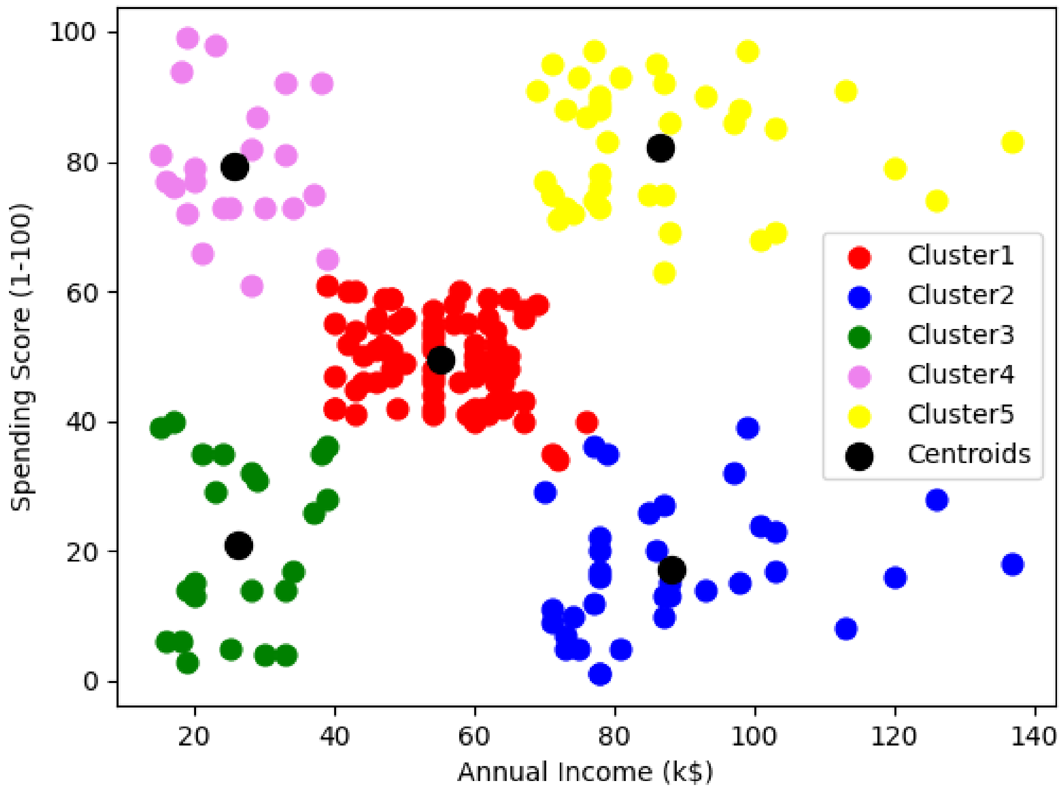
plt.show()

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:/Users/ajcemca/PycharmProjects/Anilect/k-mean/kmean.py
[[ 15  39]
 [ 15  81]
 [ 16   6]
 [ 16  77]
 [ 17  40]
 [ 17  76]
 [ 18   6]
 [ 18  94]
 [ 19   3]
 [ 19  72]
 [ 19  14]
 [ 19  99]
 [ 20  15]
 [ 20  77]
 [ 20  13]
 [ 20  79]
 [ 21  35]
 [ 21  66]
 [ 23  29]
 [ 23  98]
 [ 24  35]
 [ 24  73]
```



```
 [137  18]
 [137  83]]
[2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 0 2 3 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 4 1 4 0 4 1 4 1 4 0 4 1 4 1 4 1 4 1 4 0 4 1 4 1 4
 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1
 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4]
```



Result: The program has been executed and output verified

Program no: 12 Date: 05-01-2022

Aim: Program to implement k-means clustering technique using any standard dataset available in the public domain.

Program:

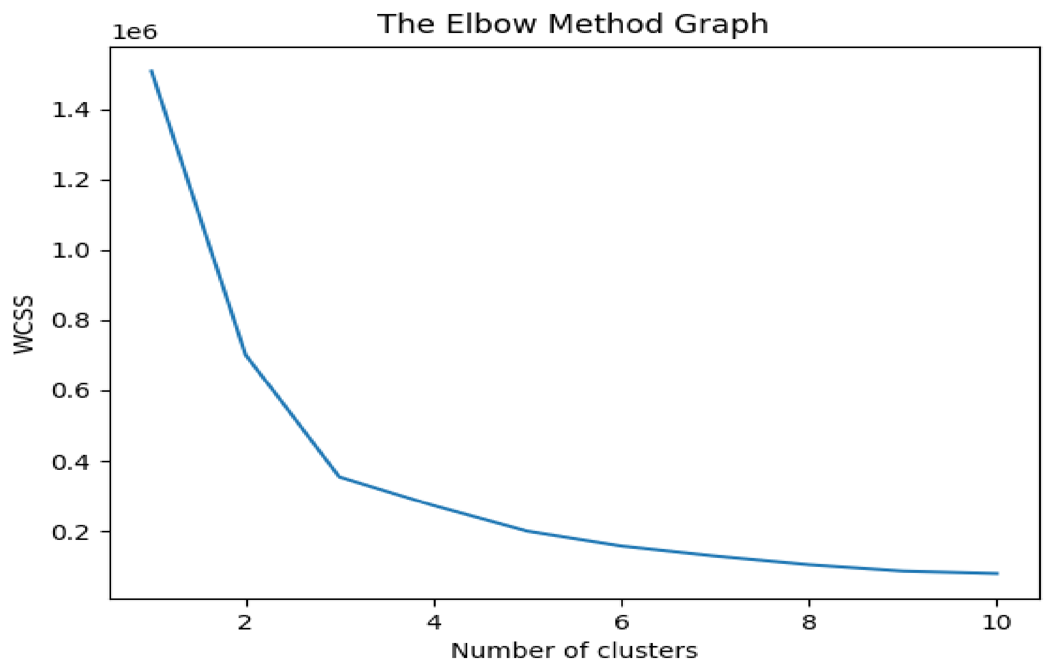
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('lati_log.csv')
X = dataset.iloc[:, [1, 2]].values
print(X)

from sklearn.cluster import KMeans

wcss_list = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++')
    kmeans.fit(X)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

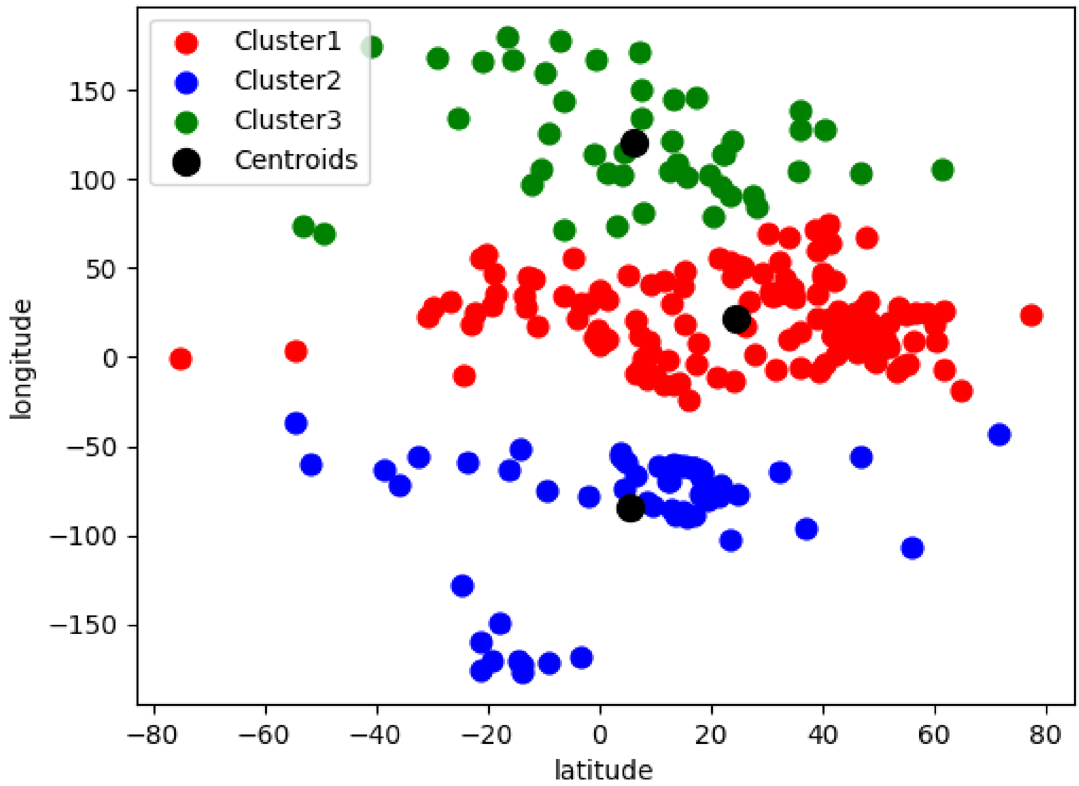
```
kmeans = KMeans(n_clusters=3, init="k-means++", random_state=42)

y_predict = kmeans.fit_predict(X)

print(y_predict)


plt.scatter(X[y_predict == 0, 0], X[y_predict == 0, 1], s=60, c='red', label='Cluster1')

plt.scatter(X[y_predict == 1, 0], X[y_predict == 1, 1], s=60, c='blue', label='Cluster2')

plt.scatter(X[y_predict == 2, 0], X[y_predict == 2, 1], s=60, c='green', label='Cluster3')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=100, c='black', label='Centroids')

plt.xlabel('latitude')

plt.ylabel('longitude')

plt.legend()


plt.show()
```

Output:

```
C:\Users\ajcemca\PycharmProjects\Anilect\venv\Scripts\python.exe C:/Users/ajcemca/PycharmProjects/Anilect/k-mean/kmean2.py
[[ 4.25462450e+01  1.60155400e+00]
 [ 2.34240760e+01  5.38478180e+01]
 [ 3.39391100e+01  6.77099530e+01]
 [ 1.70608160e+01 -6.17964280e+01]
 [ 1.82205540e+01 -6.30686150e+01]
 [ 4.11533320e+01  2.01683310e+01]
 [ 4.00690990e+01  4.50381890e+01]
 [ 1.22260790e+01 -6.90600870e+01]
 [-1.12026920e+01  1.78738870e+01]
 [-7.52509730e+01 -7.13890000e-02]
 [-3.84160970e+01 -6.36166720e+01]
 [-1.42709720e+01 -1.70132217e+02]
```

```
 [-1.90154380e+01  2.91548570e+01]]
[0 0 0 1 1 0 0 1 0 0 1 1 0 2 1 0 0 1 2 0 0 0 0 0 0 1 2 1 1 1 2 0 0 0 1 1 2
 0 0 0 0 0 1 1 0 2 1 1 1 0 2 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 2 1 2 0 0 0 0
 1 0 1 0 0 0 1 0 0 1 0 0 1 1 2 0 1 0 2 2 1 0 1 0 2 0 0 0 2 2 0 0 0 0 0 1 0
 2 0 0 2 1 0 1 2 2 0 1 0 2 0 1 0 2 0 0 0 0 0 0 0 0 0 0 0 2 0 0 2 2 2 2 1 0
 1 0 0 2 0 1 2 0 0 2 0 2 0 1 0 0 2 2 1 2 0 1 1 1 2 2 0 0 1 1 1 0 0 2 1 0 0
 0 0 2 0 0 2 0 0 0 2 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 2 0 2 0 1 2 0 0 1 0 1 2
 2 0 0 0 1 1 0 0 1 1 1 1 2 2 1 1 0 0 0 0 0 0]

Process finished with exit code 0
```





Result: The program has been executed and output verified