20MCA241 DATA SCIENCE LAB

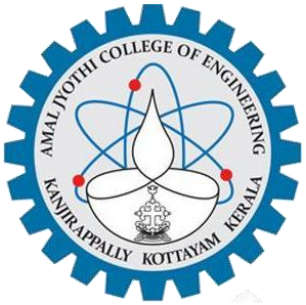
Lab Report Submitted By

ANAND K ANIL

Reg. No.: AJC20MCA-2015

In Partial fulfillment for the Award of the Degree Of

MASTER OF COMPUTER APPLICATIONS (2 Year)
(MCA)
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY



AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY

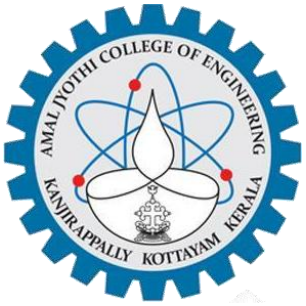
[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC with 'A' grade. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2020-2022

DEPARTMENT OF COMPUTER APPLICATIONS

AMAL JYOTHI COLLEGE OF ENGINEERING

KANJIRAPPALLY



CERTIFICATE

This is to certify that the Lab report, "**20MCA241 DATA SCIENCE LAB"** is the bonafide work of **ANAND K ANIL (Reg.No:AJC20MCA-2015)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2021-22.

Ms. Meera Rose Mathew
Staff In-Charge

Rev.Fr.Dr.Rubin Thottupurathu Jose
Head of the Department

Internal Examiner

External Examiner

CONTENT

11	**Program to implement K-Means Clustering technique using any standard dataset available in the public domain.**	**05/01/2022**	**24**	
12	**Program to implement K-Means Clustering technique using any standard dataset available in the public domain.**	**05/01/2022**	**27**	
13	**Programs on Convolutional Neural Network(CNN) to classify images from any standard dataset in the public domain.**	**02/02/2022**	**30**	
14	**Program to implement a simple Web Crawler using python.**	**16/02/2022**	**36**	
15	**Program to implement a simple Web Crawler using python.**	**16/02/2022**	**38**	
16	**Program to implement scraping of any webpage of any popular website.**	**16/02/2022**	**40**	
17	**Program for Natural Language Processing which performs n-grams.**	**16/02/2022**	**43**	
18	**Program for Natural Language Processing which performs n-grams(Using built-in functions).**	**23/02/2022**	**44**	
19	**Program for Natural Language Processing which performs speech tagging.**	**23/02/2022**	**45**	
20	**Write a python program for natural program language processing with chunking.**	**23/02/2022**	**47**	
21	**Write a python program for natural program language processing with chunking.**	**23/02/2022**	**48**	

Program no: 01 **Date:** 24-11-2021

Aim: Perform all matrix operation using python.

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:

```
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: Program to perform SVD using 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:

```
[[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 using built-in function

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:

```
[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 built-in functions.

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:

```
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

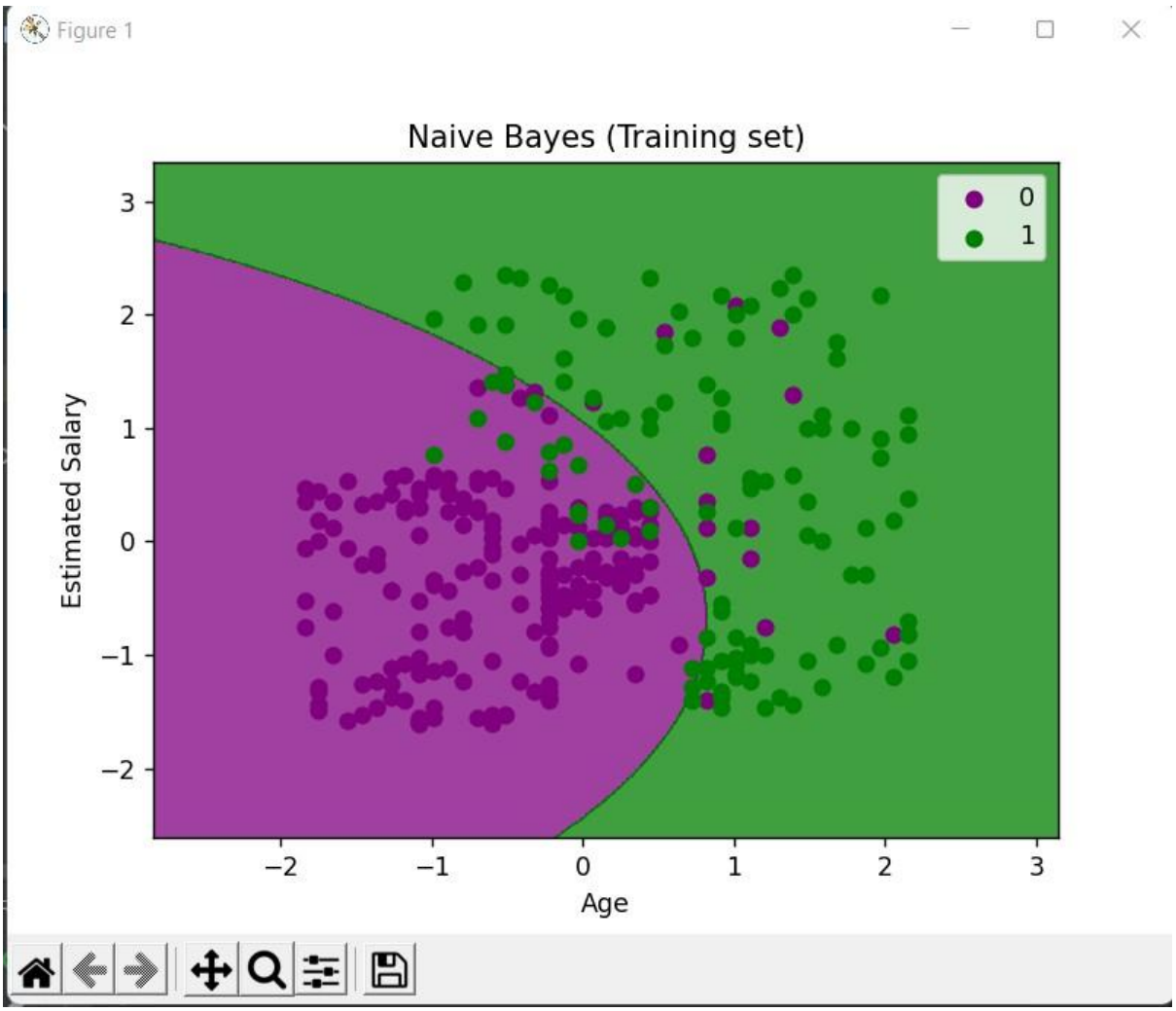
Aim: Program to implement Naïve Bayes Algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm.

Program:

```
import pandas as pd

dataset = pd.read_csv('Social_Network_Ads.csv')
x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=10)
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
gnb.fit(x_train, y_train)
y_pred = gnb.predict(x_test)
print(y_pred)
from sklearn import metrics

print("Accuracy", metrics.accuracy_score(y_test, y_pred) * 100)
import numpy as nm
import matplotlib.pyplot as mtp
from matplotlib.colors import ListedColormap

x_set, y_set = x_train, y_train
```

```
X1, X2 = nm.meshgrid(nm.arange(start=x_set[:, 0].min() - 1, stop=x_set[:, 0].max() + 1,
                               step=0.01),
                     nm.arange(start=x_set[:, 1].min() - 1, stop=x_set[:, 1].max() + 1, step=0.01))
mtp.contourf(X1, X2, gnb.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha=0.75, cmap=ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c=ListedColormap(('purple',
                                                            'green'))(i), label=j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
x_set, y_set = x_test, y_test
X1, X2 = nm.meshgrid(nm.arange(start=x_set[:, 0].min() - 1, stop=x_set[:, 0].max() + 1,
                               step=0.01),
                     nm.arange(start=x_set[:, 1].min() - 1, stop=x_set[:, 1].max() + 1, step=0.01))
mtp.contourf(X1, X2, gnb.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha=0.75, cmap=ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c=ListedColormap(('purple',
                                                            'green'))(i), label=j)
mtp.title('Naive Bayes (test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:




```
[0 0 1 1 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 1
 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1 1 0 1 0 0 0 0 0 0
 0 0 0 0 1 1]
Accuracy 91.25
```

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 the public domain

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:

```
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]

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 its performance

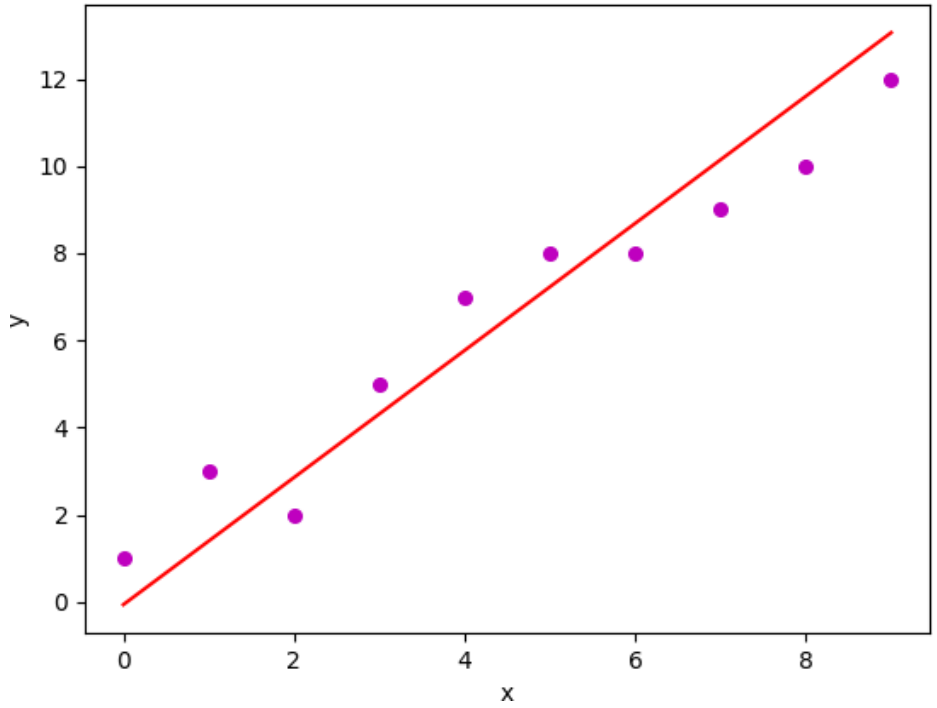
Program:

```
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:

```
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: 08 **Date:** 15-12-2021

Aim: Program to implement Linear and Multiple regression techniques using car dataset available in public domain and evaluate its performance

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:

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 its performance and plotting graph

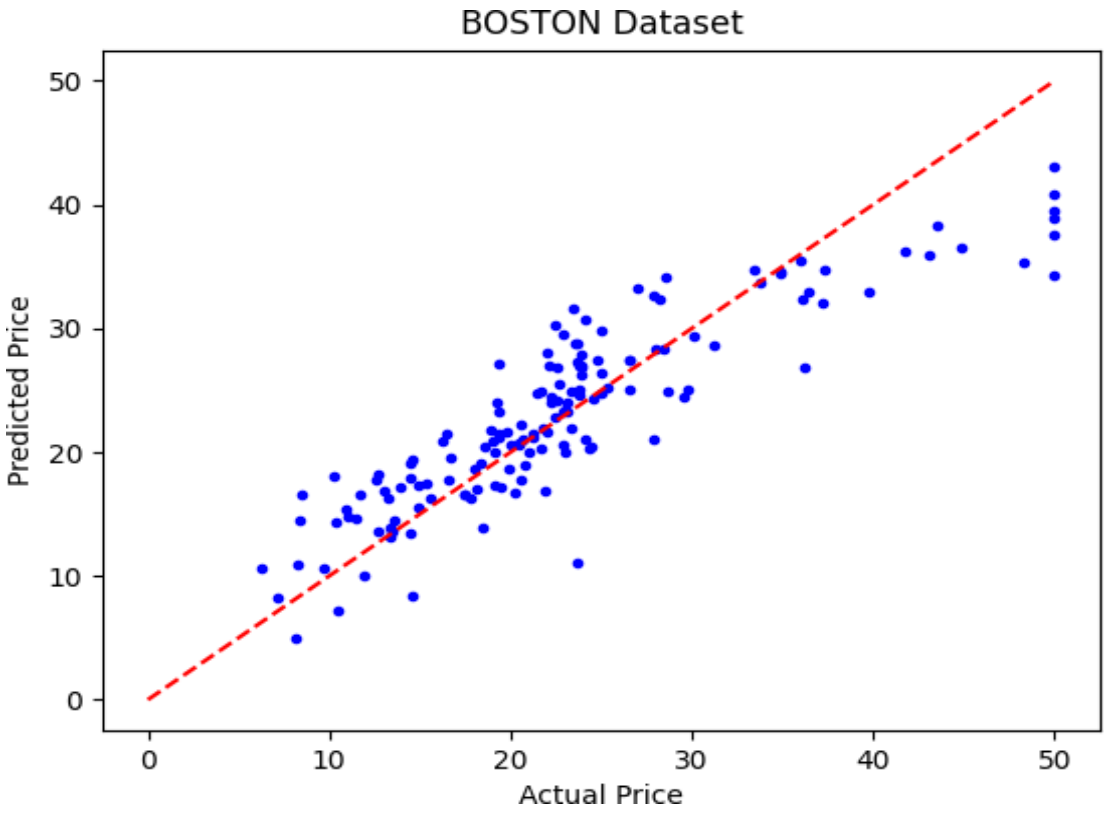
Program:

```
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 tree 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,
```

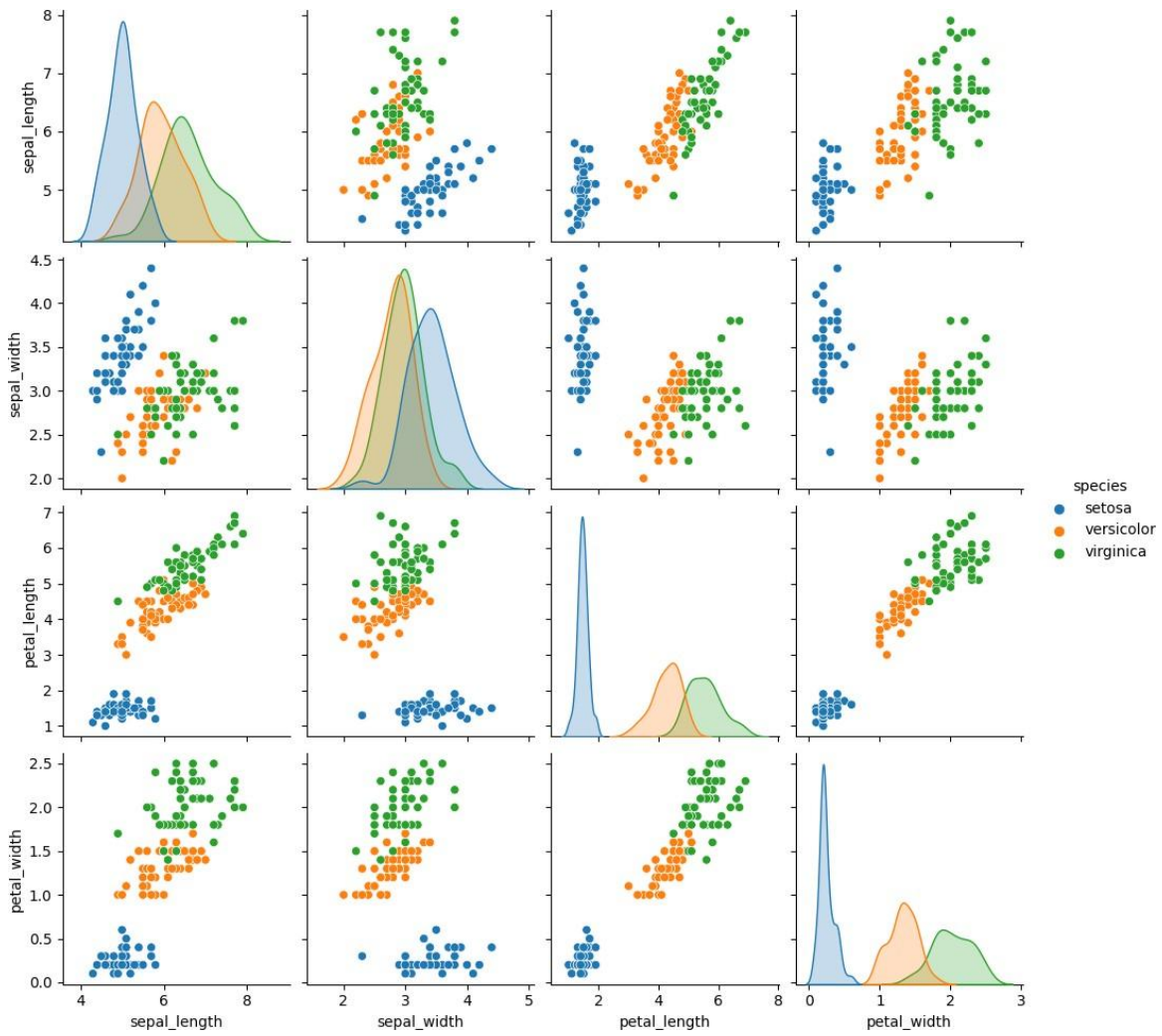
class_names=['setosa', 'vercicolor', 'verginica'], filled=True, precision=4, rounded=True)

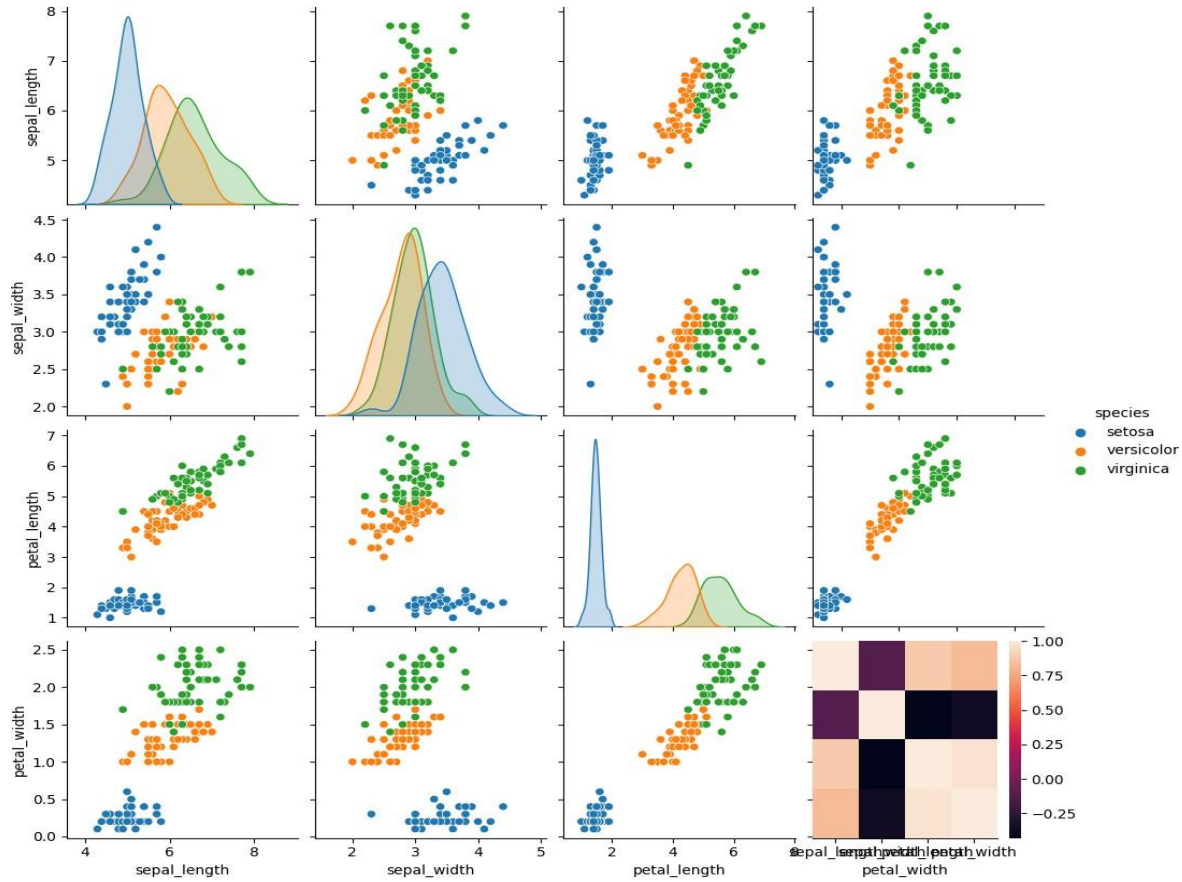
plt.savefig('tree.png')

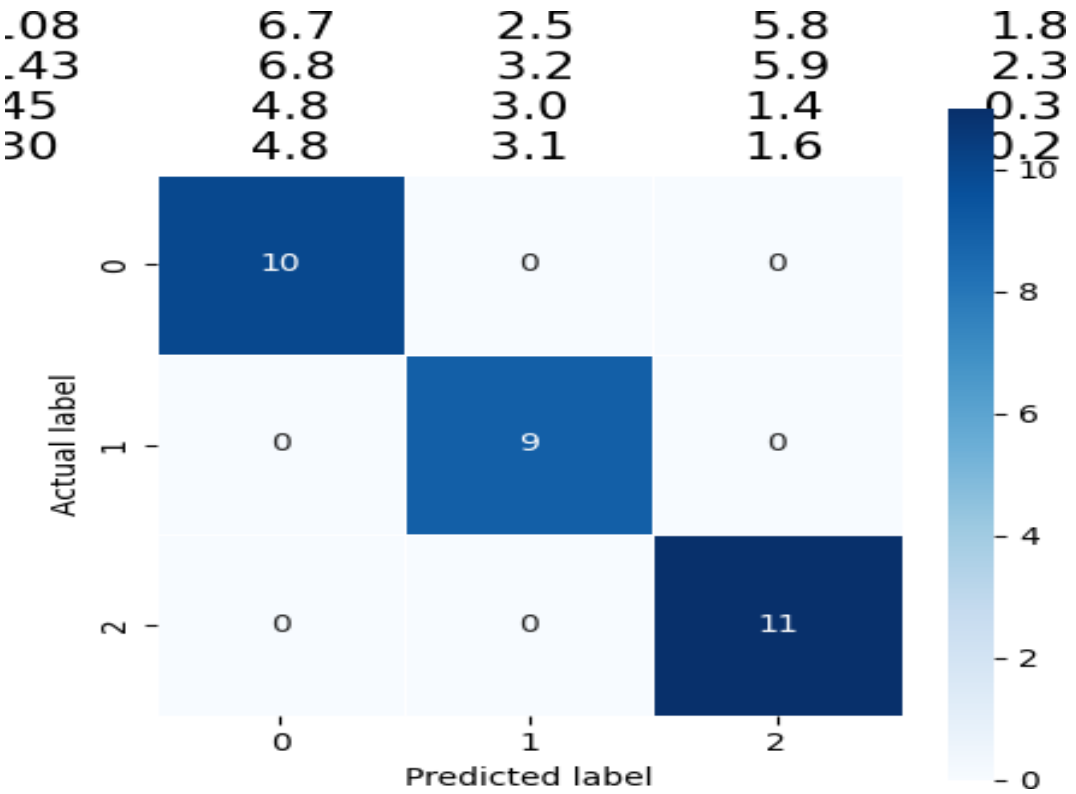
Output:

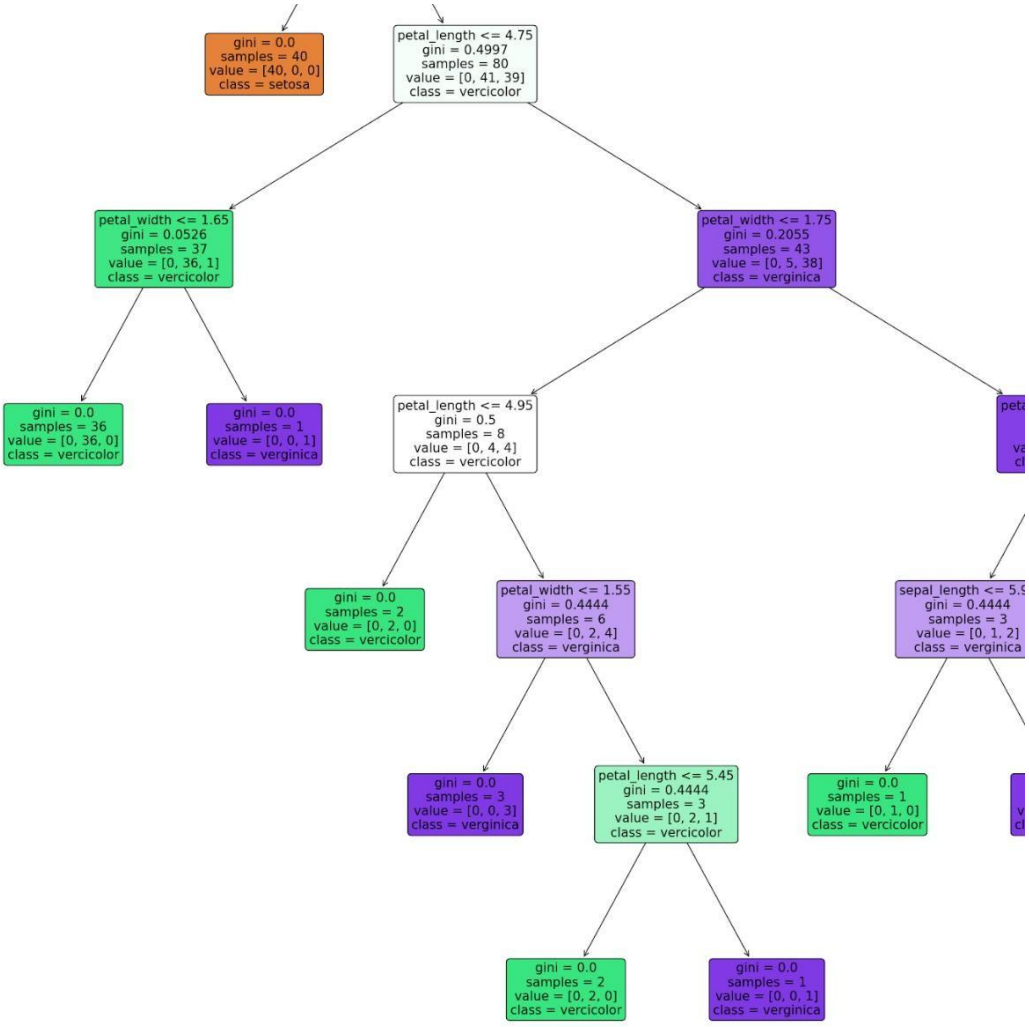
```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          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 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 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


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



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.

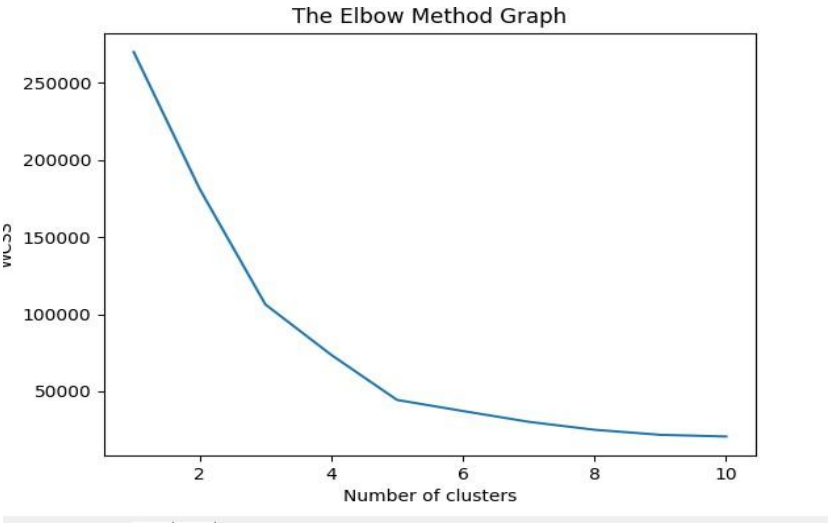
Program:

```
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')
```

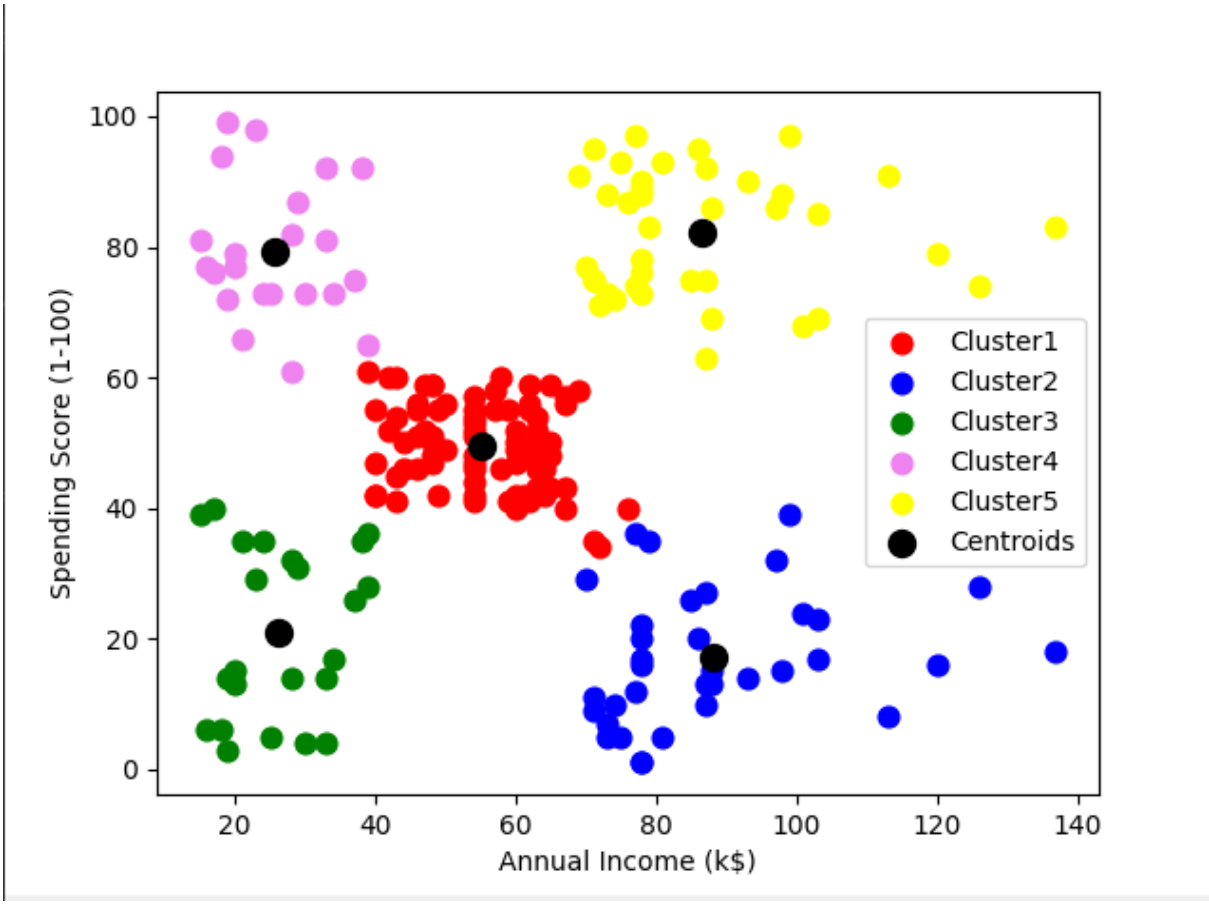
```
plt.scatter(X[y_predict == 3, 0], X[y_predict == 3, 1], s=60, c='violet', label='Cluster4')
plt.scatter(X[y_predict == 4, 0], X[y_predict == 4, 1], s=60, c='yellow', label='Cluster5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=100, c='black', label='Centroids')

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

Output:

```
[[ 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 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 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
 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:

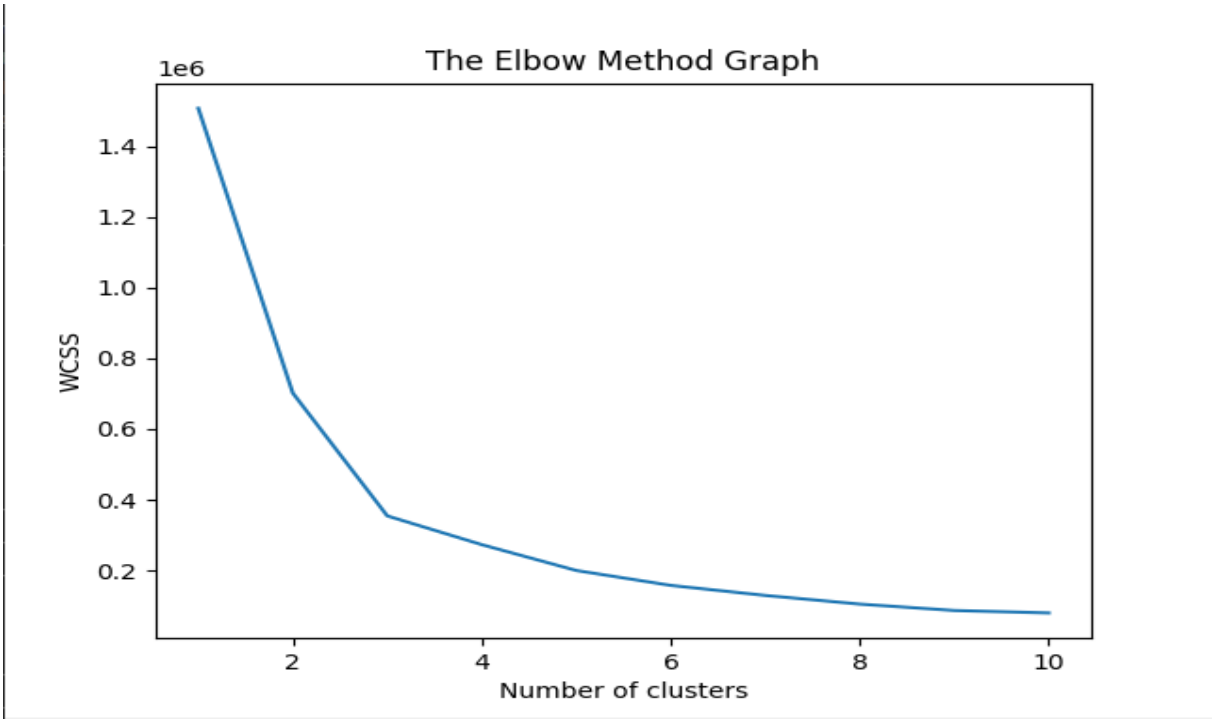
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

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')
```

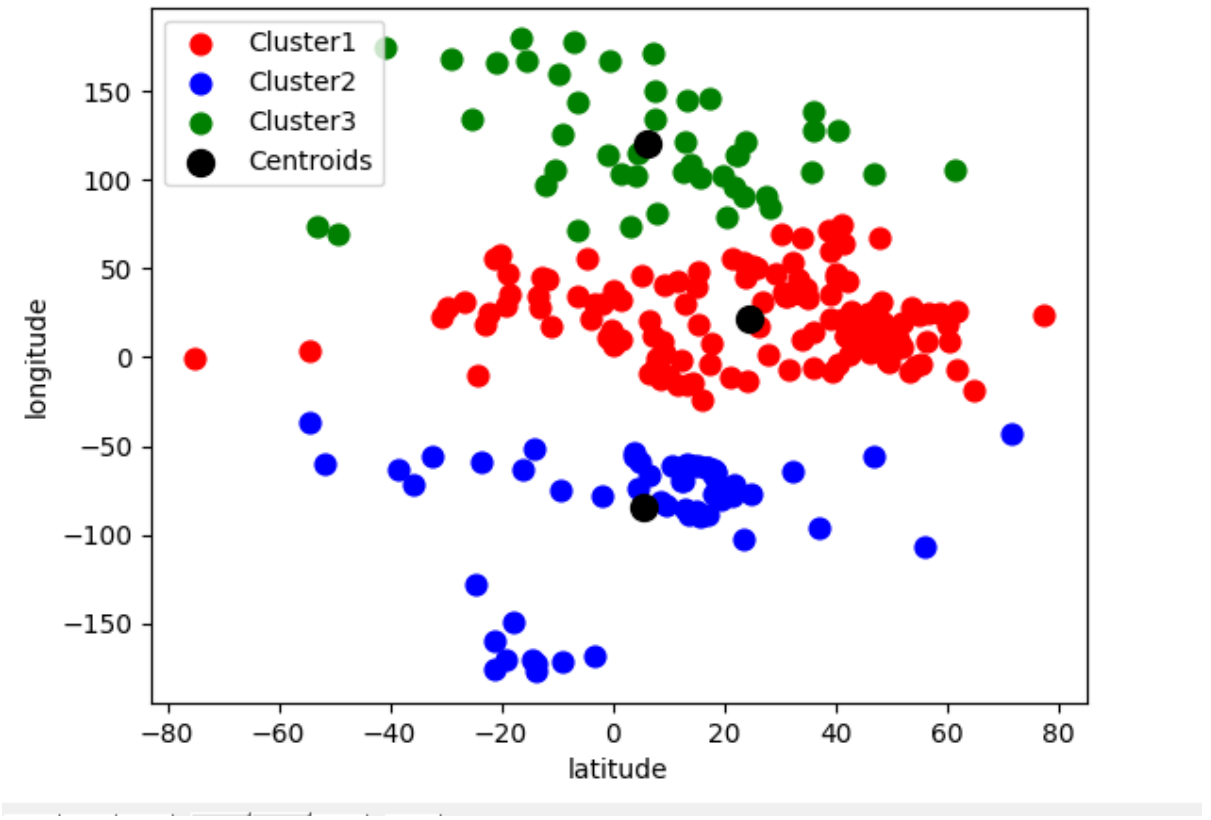
```
plt.scatter(X[y_predict == 2, 0], X[y_predict == 2, 1], s=60, c='green', label='Cluster3')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=100, c='black', label='Centroids')


plt.xlabel('latitude')
plt.ylabel('longitude')
plt.legend()

plt.show()
```

Output:

```
[[ 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

Program no: 13 **Date:** 02-02-2022

Aim: Program on convolutional neural network to classify images from any standard dataset in the public domain.

Program:

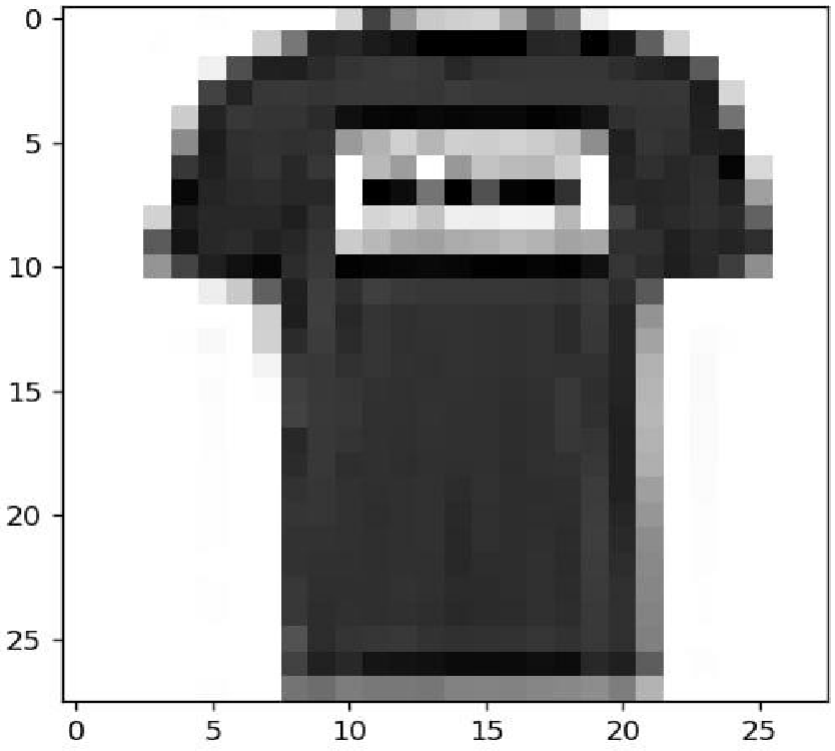
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras

np.random.seed(42)
# tf.set.random. seed(42)
fashion_mnist = keras.datasets.fashion_mnist
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
print(X_train.shape, X_test.shape)


X_train = X_train / 255.0
X_test = X_test / 255.0
plt.imshow(X_train[1], cmap='binary')
plt.show()
np.unique(y_test)

class_names = ['T-Shirt/Top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', '8ag', 'Ankle Boot']
n_rows = 5
n_cols = 10
plt.figure(figsize=(n_cols * 1.4, n_rows * 1.6))


for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
```

```
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap='binary', interpolation='nearest')
        plt.axis('off')
        plt.title(class_names[y_train[index]])
        plt.show()

model_CNN = keras.models.Sequential()
model_CNN.add(keras.layers.Conv2D(filters=32, kernel_size=7, padding='same',
activation='relu', input_shape=[28, 28, 1]))
model_CNN.add(keras.layers.MaxPooling2D(pool_size=2))
model_CNN.add(keras.layers.Conv2D(filters=64, kernel_size=3, padding='same',
activation='relu'))
model_CNN.add(keras.layers.MaxPooling2D(pool_size=2))
model_CNN.add(keras.layers.Conv2D(filters=32, kernel_size=3, padding='same',
activation='relu'))
model_CNN.add(keras.layers.MaxPooling2D(pool_size=2))
model_CNN.summary()

model_CNN.add(keras.layers.Flatten())
model_CNN.add(keras.layers.Dense(units=128, activation='relu'))
model_CNN.add(keras.layers.Dense(units=64, activation='relu'))
model_CNN.add(keras.layers.Dense(units=10, activation='softmax'))
model_CNN.summary()

model_CNN.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]

history_CNN = model_CNN.fit(X_train, y_train, epochs=2, validation_split=0.1)
pd.DataFrame(history_CNN.history).plot()


plt.grid(True)
plt.xlabel('epochs')
```

```
plt.ylabel('loss/accuracy')
plt.title('Training and validation plot')
plt.show()

test_loss, test_accuracy = model_CNN.evaluate(X_test, y_test)
print(' Test Loss :{}, Test Accuracy : {}'.format(test_loss, test_accuracy))
```

Output:

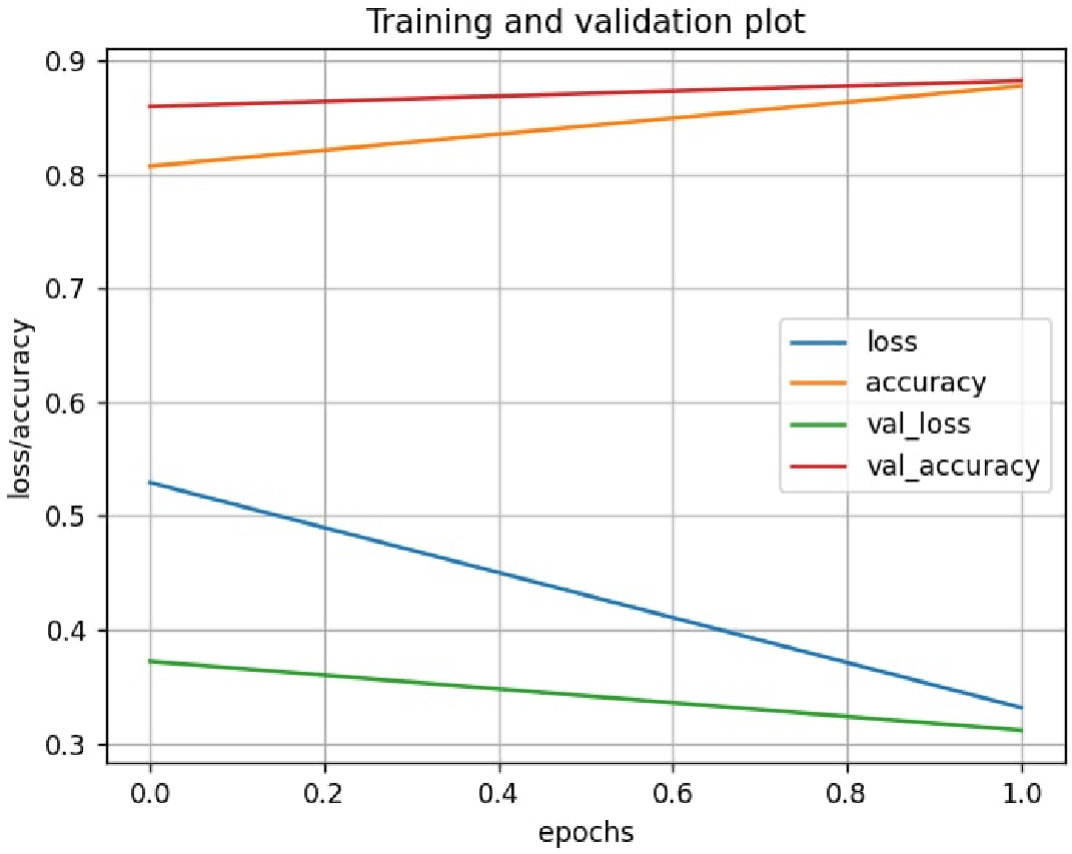
```
(60000, 28, 28) (10000, 28, 28)
```




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        1600

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 32)          18464

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 32)         0
 2D)

=================================================================
Total params: 38,560
Trainable params: 38,560
Non-trainable params: 0
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        1600
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        1600

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 32)          18464

 max_pooling2d_2 (MaxPooling  (None, 3, 3, 32)         0
 2D)

 flatten (Flatten)           (None, 288)               0

 dense (Dense)               (None, 128)               36992

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
```

```
=================================================================
Total params: 84,458
Trainable params: 84,458
Non-trainable params: 0
_________________________________________________________________
Epoch 1/2
1688/1688 [==============================] - 53s 31ms/step - loss: 0.5294 - accuracy: 0.8071 - val_loss: 0.3724 - val_accuracy: 0.8595
Epoch 2/2
1688/1688 [==============================] - 52s 31ms/step - loss: 0.3318 - accuracy: 0.8775 - val_loss: 0.3120 - val_accuracy: 0.8822
313/313 [==============================] - 3s 10ms/step - loss: 0.3342 - accuracy: 0.8752
 Test Loss :0.3342011570930481, Test Accuracy : 0.8751999735832214

Process finished with exit code 0
```



Result: The program has been executed and output verified

Program no: 14 **Date:** 16-02-2022

Aim: Program to implement a simple web crawler using python

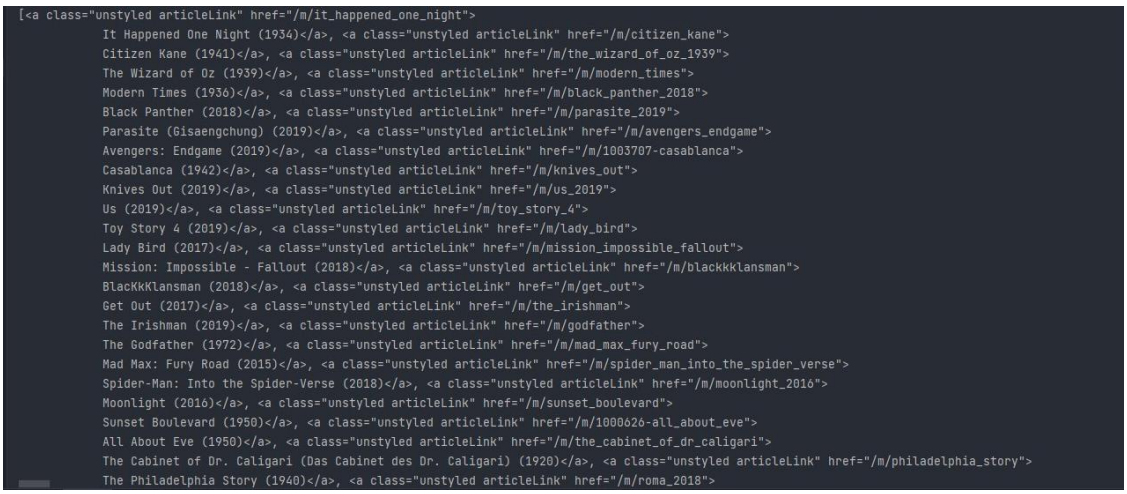
Program:

```
import requests
import lxml

from bs4 import BeautifulSoup

url = "https://www.rottentomatoes.com/top/bestofrt/"

headers = {
   'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36 QIHU 360SE'
}
f = requests.get(url, headers=headers)
movies_lst = []
soup = BeautifulSoup(f.content, 'html.parser')

movies = soup.find('table', {
   'class': 'table'
}).find_all('a')
print(movies)
num = 0

for anchor in movies:
   urls = 'https://www.rottentomatoes.com' + anchor['href']
   movies_lst.append(urls)
print(movies_lst)
num += 1
movies_Url = urls

movie_f = requests.get(movies_Url, headers=headers)
movies_soup = BeautifulSoup(movie_f.content, 'lxml')
```

```
movie_content = movies_soup.find('div', {
    'class': 'movie_synopsis clamp clamp-6 js-clamp'
})

print(num, urls, '\n', 'Movie:' + anchor.string.strip())
print('Movie info: ' + movie_content.string.strip())
```

Output:

Result: The program has been executed and output verified



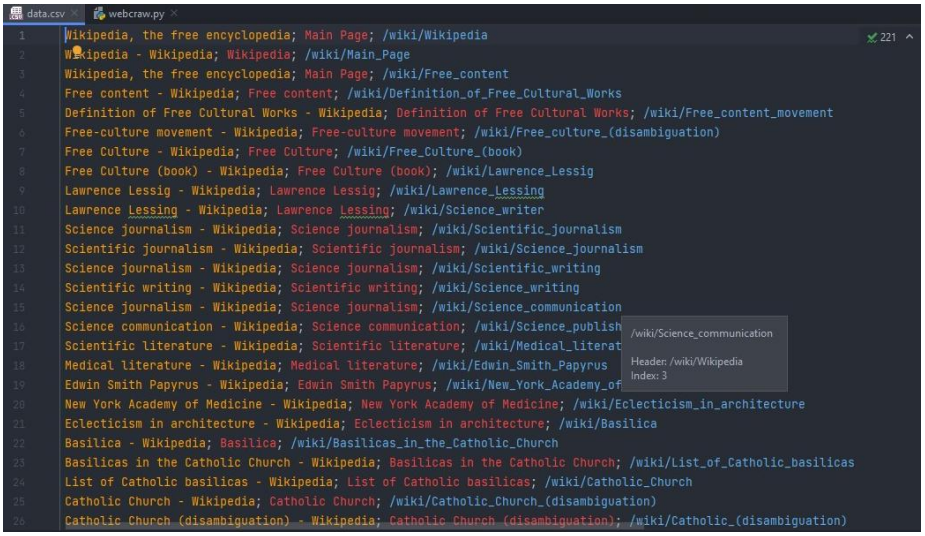
Program no: 15 **Date:** 16-02-2022

Aim: Program to implement a simple web crawler using python

Program:

```
from bs4 import BeautifulSoup
import requests

pages_crawled = [];

def crawler(url):
    page = requests.get(url)
    soup = BeautifulSoup(page.text, 'html.parser')
    links = soup.find_all('a')

    for link in links:

        if 'href' in link.attrs:

            if link['href'].startswith('/wiki') and ":" not in link['href']:

                if link['href'] not in pages_crawled:
                    new_link = f"https://en.wikipedia.org{link['href']}"

                    pages_crawled.append(link['href'])

                    try:
                        with open('data.csv', 'a') as file:
                            file.write(f'{soup.title.text}; {soup.h1.text}; {link["href"]}\n')
                        crawler(new_link)

                    except:
                        continue

crawler("https://en.wikipedia.org")
```

Output:



Result: The program has been executed and output verified

Program no: 16 **Date:** 16-02-2022

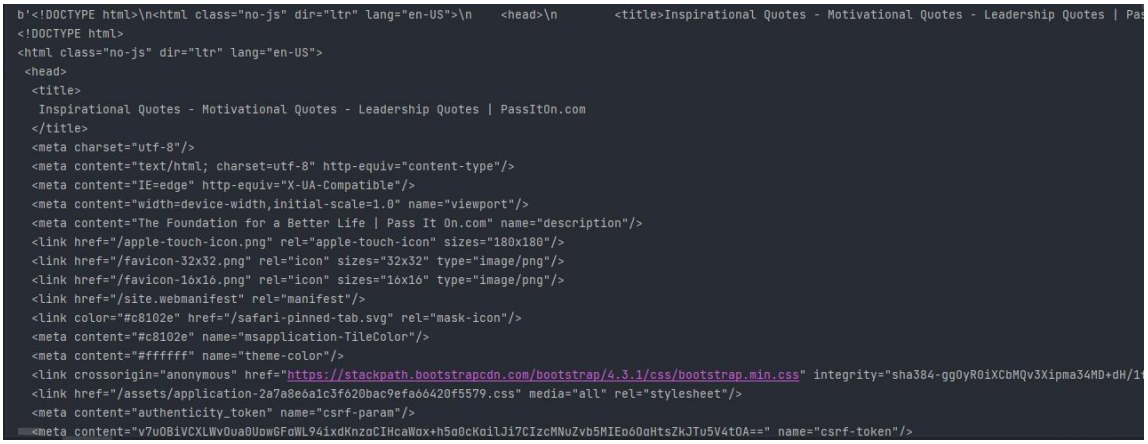
Aim: Program to implement scrap of any webpage

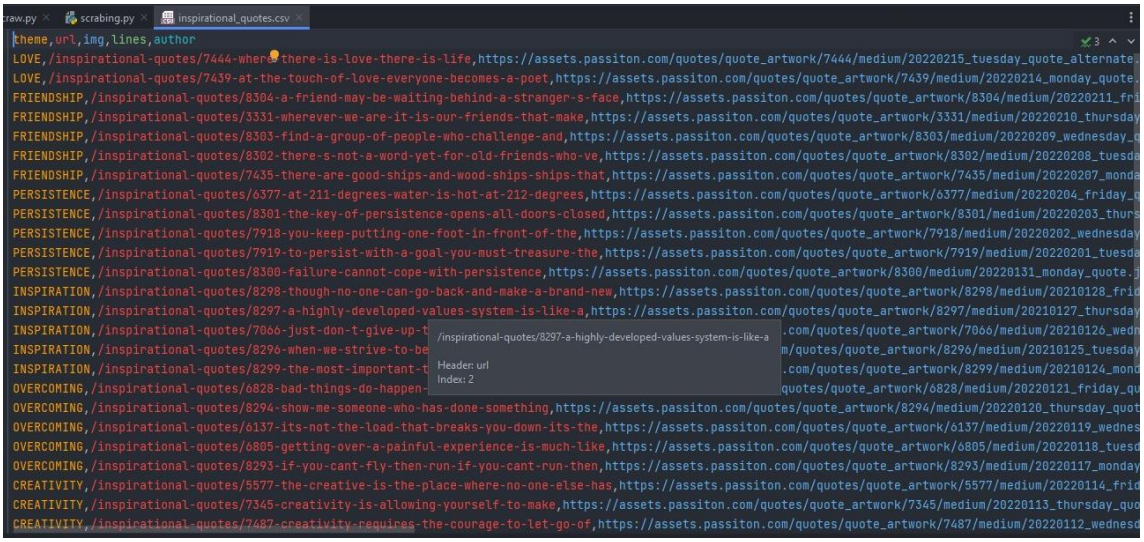
Program:

```
import requests
from bs4 import BeautifulSoup
import csv

URL = "http://www.values.com/inspirational-quotes"

r = requests.get(URL)
print(r.content)

soup = BeautifulSoup(r.content, 'lxml')
print(soup.prettify())

quotes = []
table = soup.find('div', attrs={'id': 'all_quotes'})

for row in table.findAll('div',
                         attrs={'class': 'col-6 col-lg-3 text-center margin-30px-bottom sm-margin-30px-top'}):

    quote = {}

    quote['theme'] = row.h5.text
    quote['url'] = row.a['href']
    quote['img'] = row.img['src']
    quote['lines'] = row.img['alt'].split(" #")[0]
    quote['author'] = row.img['alt'].split(" #")[1]
    quotes.append(quote)

filename = 'inspirational_quotes.csv'

with open(filename, 'w', newline='') as f:
```

```
w = csv.DictWriter(f, ['theme', 'url', 'img', 'lines', 'author'])
w.writeheader()

for quote in quotes:
    w.writerow(quote)
```

Output:





Result: The program has been executed and output verified

Program no: 17 **Date:** 16-02-2022

Aim: Program for Natural Language Processing which performs n-grams

Program:

```
def generate_ngrams(text, WordsToCombine):
    words = text.split()
    output = []
    for i in range(len(words) - WordsToCombine + 1):
        output.append(words[i:i + WordsToCombine])
    return output

x = generate_ngrams(text='Hello there, Welcome to DS Lab Record', WordsToCombine=3)
print(x)
```

Output:

```
[['Hello', 'there,', 'Welcome'], ['there,', 'Welcome', 'to'], ['Welcome', 'to', 'DS'], ['to', 'DS', 'Lab'], ['DS', 'Lab', 'Record']]

Process finished with exit code 0
```

Result: The program has been executed and output verified

Program no: 18 **Date:** 16-02-2022

Aim: Program for Natural Language Processing which performs n-grams (Using in built functions)

Program:

```
import nltk

nltk.download()

from nltk.util import ngrams

samplText = 'This is a very good book to study'

NGRAMS = ngrams(sequence=nltk.word_tokenize(samplText), n=2)

for grams in NGRAMS:

    print(grams)
```

Output:

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
('This', 'is')
('is', 'a')
('a', 'very')
('very', 'good')
('good', 'book')
('book', 'to')
('to', 'study')

Process finished with exit code 0
```

Result: The program has been executed and output verified

Program no: 19 **Date:** 16-02-2022

Aim: Program for Natural Language Processing which performs speech tagging

Program:

```
import nltk
from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize, sent_tokenize

stop_words = set(stopwords.words('english'))

txt = "Sukanya, Rajib and Naba are my good friends. " \
      "Sukanya is getting married next year. " \
      "Marriage is a big step in one’s life." \
      "It is both exciting and frightening. " \
      "But friendship is a sacred bond between people." \
      "It is a special kind of love between us. " \
      "Many of you must have tried searching for a friend " \
      "but never found the right one."

# sent_tokenize is one of instances of
# PunktSentenceTokenizer from the nltk.tokenize.punkt module

tokenized = sent_tokenize(txt)

for i in tokenized:
    # Word tokenizers is used to find the words
    # and punctuation in a string
    wordsList = nltk.word_tokenize(i)

    # removing stop words from wordList
    wordsList = [w for w in wordsList if not w in stop_words]

    # Using a Tagger. Which is part-of-speech
```

```
# tagger or POS-tagger.
tagged = nltk.pos_tag(wordsList)

print(tagged)
```

Output:

Result: The program has been executed and output verified

Program no: 20 **Date:** 23-02-2022

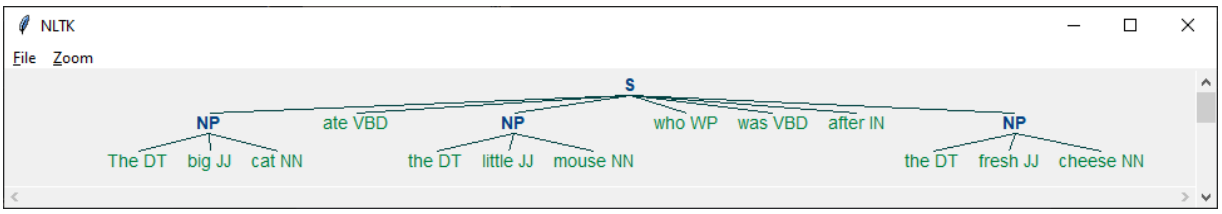
Aim: Write a python program for natural program language processing with chunking

Program:

```
import nltk
new = "The big cat ate the little mouse who was after the fresh cheese"
new_tokens = nltk.word_tokenize(new)
print(new_tokens)
new_tag = nltk.pos_tag(new_tokens)
print(new_tag)
grammer = "NP: {<DT>?<JJ>*<NN>}"
chunkParser = nltk.RegexpParser(grammer)
chunked = chunkParser.parse(new_tag)
print(chunked)
chunked.draw(
```

Output:

```
['The', 'big', 'cat', 'ate', 'the', 'little', 'mouse', 'who', 'was', 'after', 'the', 'fresh', 'cheese']
[('The', 'DT'), ('big', 'JJ'), ('cat', 'NN'), ('ate', 'VBD'), ('the', 'DT'), ('little', 'JJ'), ('mouse', 'NN'), ('who', 'WP'), ('was', 'VBD'), ('after', 'IN'),
(S
  (NP The/DT big/JJ cat/NN)
  ate/VBD
  (NP the/DT little/JJ mouse/NN)
  who/WP
  was/VBD
  after/IN
  (NP the/DT fresh/JJ cheese/NN))
```



Result: The program has been executed and output verified

Program no: 21 **Date:** 23-02-2022

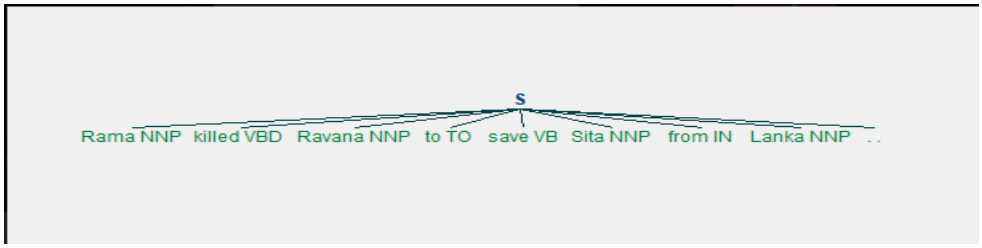
Aim: Write a python program for natural program language processing with chunking

Program:

```
import nltk
nltk.download('averaged_perceptron_tagger')
sample_text = """Rama killed Ravana to save Sita from Lanka. The legend of the Ramayan is the most popular Indian epic. A lot of movies and serials have already been shot in several languages here in India based on the Ramayana. """
tokenized = nltk.sent_tokenize(sample_text)
for i in tokenized:
    words = nltk.word_tokenize(i)
    # print(words)
    tagged_words = nltk.pos_tag(words)
    # print(tagged_words)
    chunkGram = r"""VB: {}"""
    chunkParser = nltk.RegexpParser(chunkGram)
    chunked = chunkParser.parse(tagged_words)
    print(chunked)
    chunked.draw()
```

Output:

```
(S
  Rama/NNP
  killed/VBD
  Ravana/NNP
  to/TO
  save/VB
  Sita/NNP
  from/IN
  Lanka/NNP
  ./.)
(S
  The/DT
  legend/NN
  of/IN
  the/DT
  Ramayan/NNP
  is/VBZ
  the/DT
  most/RBS
  popular/JJ
  Indian/JJ
  epic/NN
  ./.)
(S
```



Result: The program has been executed and output verified