20MCA241 DATA SCIENCE LAB

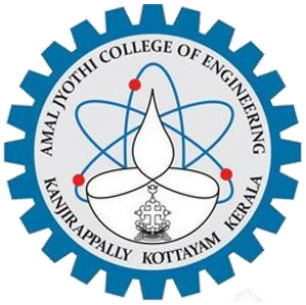
Lab Report Submitted By

ABHISHEK SCARIYA M B

Reg. No.: AJC20MCA-2001

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 **ABHISHEK SCARIYA M B (Reg.No:AJC20MCA-2001)** 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

Lab In-Charge

CONTENT

12	**Program to implement K-Means clustering technique using any standard dataset available in the public domain**	**05/01/2022**	**25**
13	**Programs on convolutional neural network to classify images from any standard dataset in the public domain**	**02/02/2022**	**29**
14	**Program to implement a simple web crawler using python**	**16/02/2022**	**34**
15	**Program to implement a simple web crawler using python**	**16/02/2022**	**36**
16	**Program to implement scrap of any website**	**16/02/2022**	**38**
17	**Program for Natural Language Processing which performs n-grams**	**16/02/2022**	**40**
18	**Program for Natural Language Processing which performs n-grams (Using in built functions)**	**16/02/2022**	**41**
19	**Program for Natural Language Processing which performs speech tagging**	**16/02/2022**	**42**
20	**Write a python program for natural program language processing with chunking**	**23/02/2022**	**44**
21	**Write a python program for natural program language processing with chunking.**	**23/02/2022**	**45**

PROGRAM NO : 01 **Date:24/11/2021**

AIM : Perform all matrix operation using python

Program Code :

```
import numpy

a1 = numpy.array([[1, 2], [4, 3]])
a2 = numpy.array([[7, 9], [11, 16]])

print("first array\n", a1)
print("second array\n", a2)

add = numpy.add(a1, a2)
sub = numpy.subtract(a1, a2)
mul = numpy.multiply(a1, a2)
div = numpy.divide(a1, a2)
dot = numpy.dot(a1, a2)
sum0 = numpy.sum(a1, axis=0)
sum1 = numpy.sum(a1, axis=1)
sum2 = numpy.sum(a1)
sqrt = numpy.sqrt(a2)
print("added arrays\n", add)
print("subtracted arrays\n", sub)
print("multiplied arrays\n", mul)
print("divided arrays\n", div)
print("product of arrays\n", dot)
print("square root of array a2\n", sqrt)
print("summation row wise\n", sum0)
print("summation column wise\n", sum1)
print("summation of a1\n", sum2)
print("transposition  of  array a1\n", a1.T)
```

Output:

```
first array
 [[1 2]
 [4 3]]
second array
 [[ 7  9]
 [11 16]]
added arrays
 [[ 8 11]
 [15 19]]
subtracted arrays
 [[ -6  -7]
 [ -7 -13]]
multiplied arrays
 [[ 7 18]
 [44 48]]
divided arrays
 [[0.14285714 0.22222222]
 [0.36363636 0.1875   ]]
product of arrays
 [[29 41]
 [61 84]]
square root of array a2
 [[2.64575131 3.        ]
 [3.31662479 4.        ]]
summation row wise
 [5 5]
summation column wise
 [3 7]
```

PROGRAM NO : 02 **Date:01/12/2021**

AIM : Program to perform SVD using python

Program Code :

```
from numpy import array
from scipy.linalg import svd
ar=array([[2,3,9],[5,6,7],[8,2,4]])
print(ar)
D, I, trans=svd(ar)
print('decompossed matrix:\n',D,'\n')
print('inverse matrix:\n',I,'\n')
print('transverse matrix:\n',trans,'\n')
```

Output:

```
[[2 3 9]
 [5 6 7]
 [8 2 4]]
decompossed matrix:
 [[-0.5642816  -0.63216723 -0.53099047]
 [-0.65124686 -0.05445576  0.75690957]
 [-0.50740891  0.77291602 -0.38096854]]

inverse matrix:
 [15.84862566  5.56106672  2.42808597]

transverse matrix:
 [[-0.53279502 -0.41739542 -0.73614573]
 [ 0.83557978 -0.12181192 -0.53569421]
 [-0.13392499  0.90052369 -0.41366796]]


Process finished with exit code 0
```

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 in build function

Program Code :

```
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
#loading data
iris = load_iris()
#create feature and target arrays
a = iris.data
b = iris.target
#split into training and test dataset
a_train, a_test, b_train, b_test = train_test_split(a, b, test_size=0.2, random_state=42)
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(a_train, b_train)
#predict on dataset which has not seen before
print(knn.predict(a_test))
#finding accuracy
x = knn.predict(a_test)
y = accuracy_score(b_test, x)
print(y)
```

Output:

```
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
1.0

Process finished with exit code 0
```

PROGRAM NO : 04 **Date:01/12/2021**

AIM : Program to implement k-NN Classification using any random dataset without using in-build functions

Program Code :

```
from math import sqrt



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



# locate most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        d = e_distance(test_row, train_row)
        distances.append((train_row, d))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append((distances[i][0]))
    return neighbors



# make a classification prediction with neighbors
def predict_class(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
```

```
    return prediction


# test distance function
dataset = [[2.7856, 3.6589, 0],
        [5.4268, 6.39698, 0],
        [6.9870, 2.39887, 0],
        [9.2014, 5.1478, 1],
        [4.8975, 2.59874, 1],
        [1.58547, 3.6542, 0]]
prediction = predict_class(dataset, dataset[0], 3)
print('expected %d, got %d.' % (dataset[0][-1], prediction))
```

Output:

```
expected 0, got 0.

Process finished with exit code 0
```

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 Code :

```
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:

```
[[ 1.92295008e+00  2.14601566e+00]
 [ 2.02016082e+00  3.78719297e-01]
 [-1.38221530e+00 -4.32498705e-01]
 [-1.18779381e+00 -1.01194013e+00]
 [ 1.92295008e+00 -9.25023920e-01]
 [ 3.67578135e-01  2.91803083e-01]
 [ 1.73156642e-01  1.46942725e-01]
 [ 2.02016082e+00  1.74040666e+00]
 [ 7.56421121e-01 -8.38107706e-01]
 [ 2.70367388e-01 -2.87638347e-01]
 [ 3.67578135e-01 -1.71750061e-01]
 [-1.18475597e-01  2.20395980e+00]
 [-1.47942605e+00 -6.35303205e-01]
 [-1.28500455e+00 -1.06988428e+00]
 [-1.38221530e+00  4.07691369e-01]
 [-1.09058306e+00  7.55356227e-01]
 [-1.47942605e+00 -2.00722133e-01]
 [ 9.50842613e-01 -1.06988428e+00]
 [ 9.50842613e-01  5.81523798e-01]
 [ 3.67578135e-01  9.87132798e-01]
 [ 5.61999628e-01 -8.96051849e-01]
 [-6.04529329e-01  1.45068594e+00]
 [-2.12648508e-02 -5.77359062e-01]
 [-6.04529329e-01  1.88526701e+00]
 [ 1.33968560e+00 -1.41754914e+00]
 [ 1.43689635e+00  9.87132798e-01]
 [ 7.59458956e-02 -8.09135634e-01]
 [-2.12648508e-02 -2.58666276e-01]
```

```
 [ 2.11737157e+00 -8.09135634e-01]
 [-1.86826903e+00  1.75914797e-01]
 [-2.15686344e-01  8.42272441e-01]
 [-1.86826903e+00 -1.27268878e+00]
 [ 2.11737157e+00  3.78719297e-01]
 [-1.38221530e+00  5.52551726e-01]
 [-1.09058306e+00 -3.45582490e-01]
 [ 1.73156642e-01 -6.64275277e-01]
 [ 3.67578135e-01  2.08236764e-03]
 [-6.04529329e-01  2.31984809e+00]
 [-3.12897090e-01  2.04886868e-01]
 [-1.57663679e+00 -2.00722133e-01]
 [ 6.59210374e-01 -1.38857706e+00]
 [-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
```

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 Code :

```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([5, 67, 44, 32, 12, 34]).reshape((-1, 1))
y = np.array([6, 76, 34, 23, 45, 23])
print(x)
print(y)
model = LinearRegression()
model.fit(x, y)
r = model.score(x, y)
print("coefficient of determination :", r)
print("intercept : ", model.intercept_)
print("slope : ", model.coef_)
y_pred = model.predict(x)
print("predicted response : ", y_pred)
```

Output:

```
[[ 5]
 [67]
 [44]
 [32]
 [12]
 [34]]
[ 6 76 34 23 45 23]
coefficient of determination : 0.5403255400751379
intercept :  8.8668710021322
slope :  [0.79277719]
predicted response :  [12.83075693 61.98294243 43.74906716 34.23574094 18.38019723 35.82129531]

Process finished with exit code 0
```

PROGRAM NO : 07 **Date:15/12/2021**

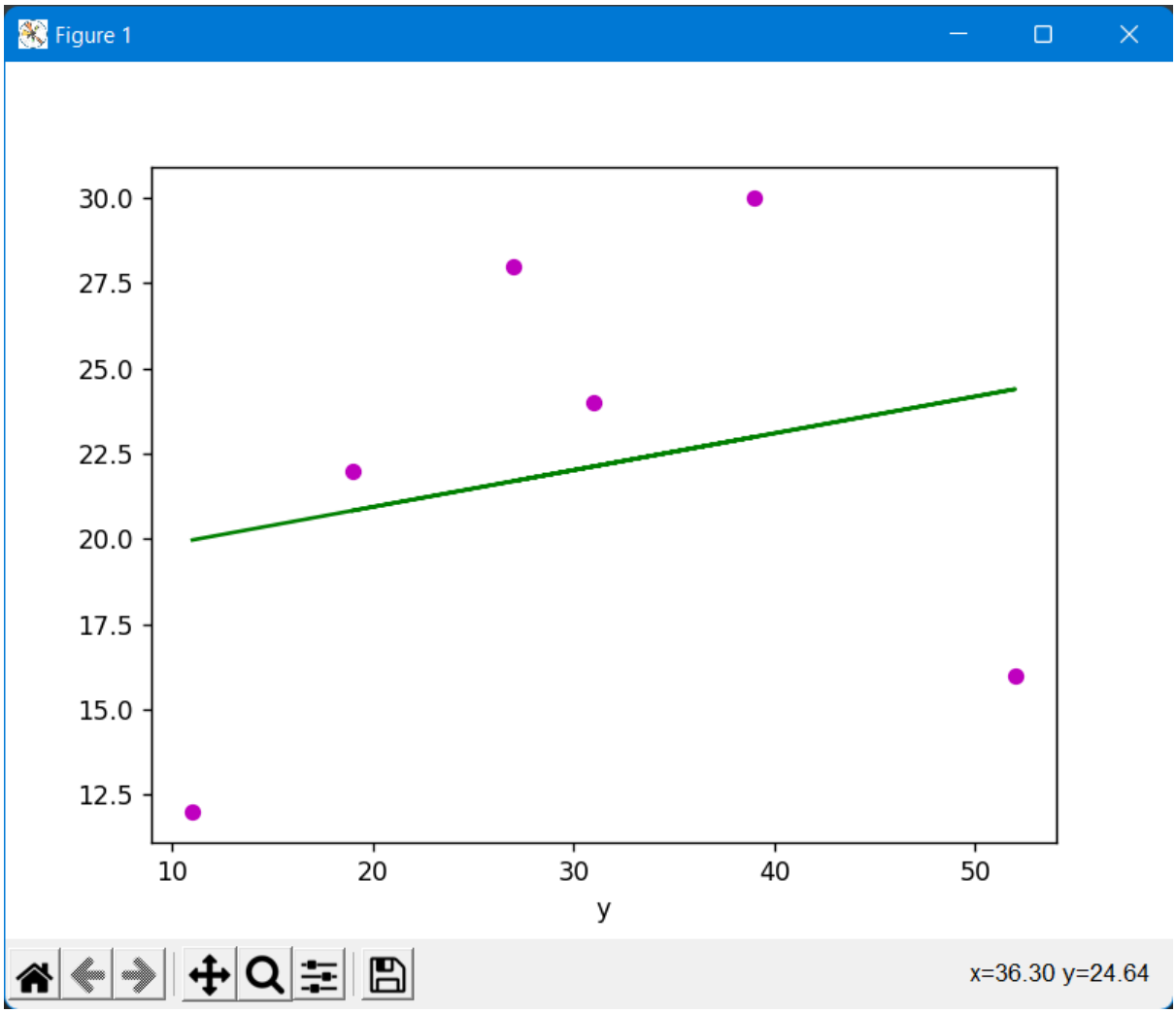
AIM : Program to implement Linear and Multiple regression techniques using any standard dataset available in public domain and evaluate its performance

Program Code :

```
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
 # no of observation
 n = np.size(x)
 # mean of x and y
 m_x = np.mean(x)
 m_y = np.mean(y)
 # cross deviation and deviation abt x
 SS_xy = np.sum(y * x) - n * m_y * m_x
 SS_xx = np.sum(x * x) - n * m_x * m_x
 # regression coefficient
 b_1 = SS_xy / SS_xx
 b_0 = m_y - b_1 * m_x
 return (b_0, b_1)
def ploting(x, y, b):
 plt.scatter(x, y, color="m", marker="o", s=30)
 # predicting response vector
 y_pred = b[0] + b[1] * x
 # plotting regression line
 plt.plot(x, y_pred, color="g")
 plt.xlabel('x')
 plt.xlabel('y')
 plt.show()
def main():
 x = np.array([19, 31, 52, 27, 39, 11])
 y = np.array([22, 24, 16, 28, 30, 12])
 b = estimate_coef(x, y)
 print("Estimated coefficients:b_0 = {} \
```

```
 b_1 = {}".format(b[0], b[1]))
 ploting(x, y, b)
if __name__ == "__main__":
 main()
```

Output:




```
Estimated coefficients:b_0 = 18.781895600063084  b_1 = 0.10786942122693575

Process finished with exit code 0
```

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 its performance

Program Code :

```
import pandas

data = pandas.read_csv("cars.csv")
x = data[['Weight', 'Volume']]
y = data['CO2']

from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(x, y)
# predict the co2 emission of a car the weight is 2300kg and the value is 1300
predictedco2 = regr.predict([[2300, 1300]])
print(predictedco2)
```

Output:

```
[107.30027195]

Process finished with exit code 0
```

PROGRAM NO : 09 **Date:15/12/2021**

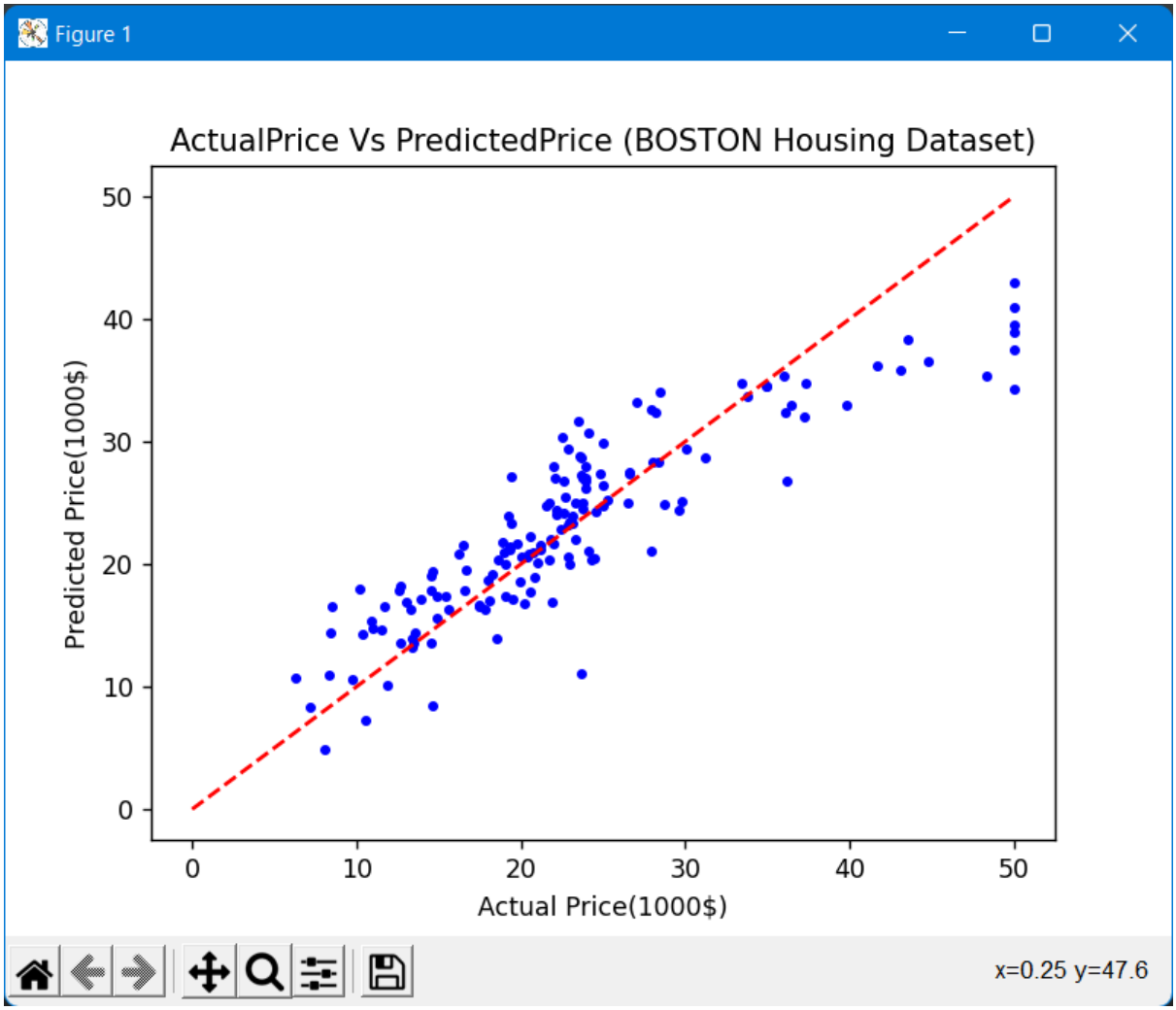
AIM : Program to implement multiple linear regression techniques using Boston dataset available in the public domain and evaluate its performance and plotting graph

Program Code :

```
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]

Intercept :  46.39649387182355
Variance score:  0.7836295385076291
Mean squared error: 19.83

Process finished with exit code 0
```



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 Code :

```
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()
```

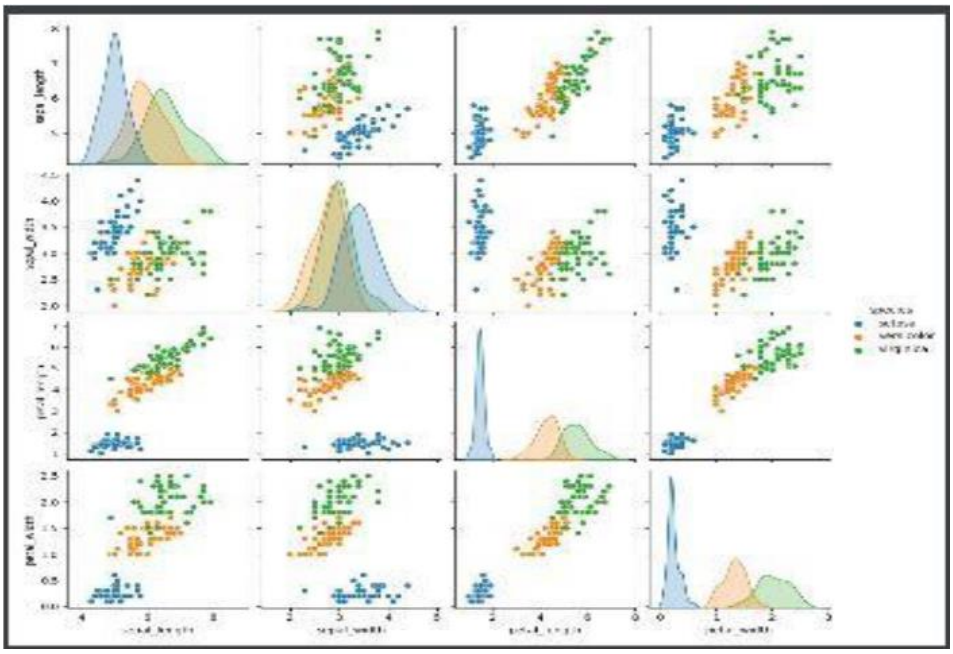
```
df1 = df1.drop('species', axis=1)
print(df1.shape)
print(df1.head())
# Defining the attributes
x = df1
print(target)
# label encoding
le = LabelEncoder()
target = le.fit_transform(target)
print(target)
y = target
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

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,
```

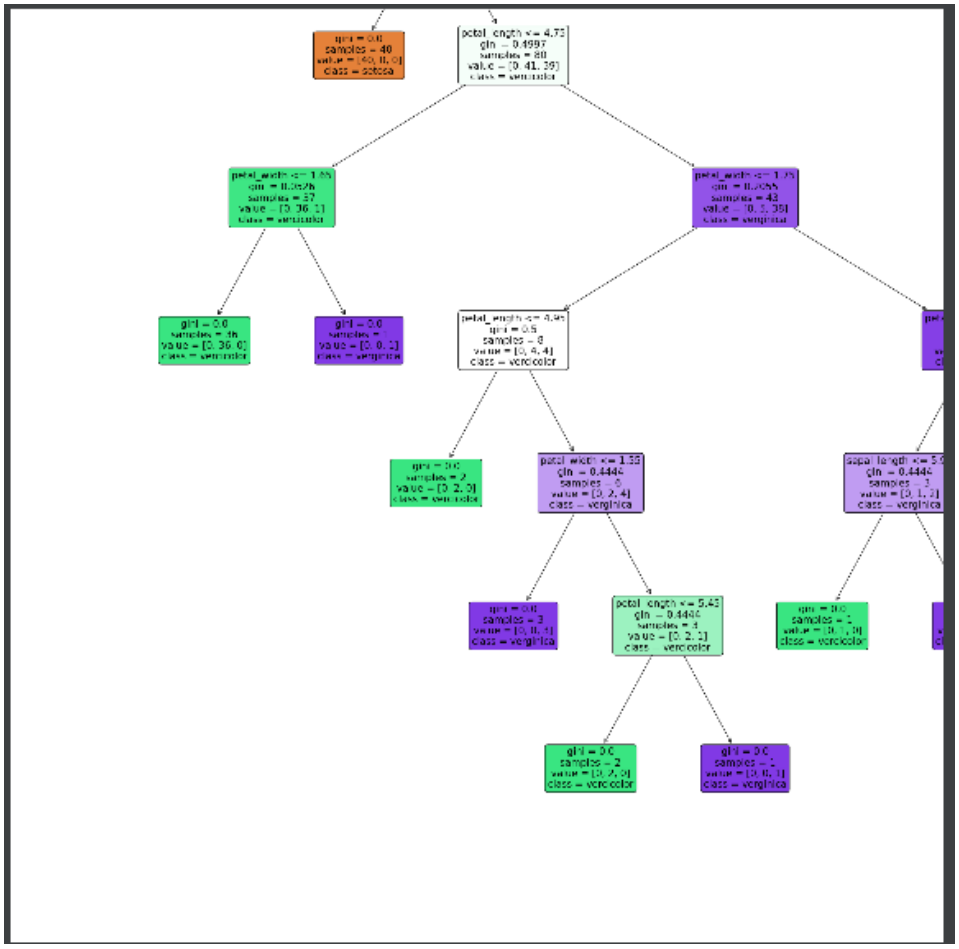
rounded=True)

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
```





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 Code :

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

# importing dataset
dataset = pd.read_csv("Mall_Customers.csv")
x = dataset.iloc[:, [3, 4]].values
print(x)
# finding optimal no of clusters using elbow
from sklearn.cluster import KMeans

wcss_list = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The elbow method graph')
mtp.xlabel('number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
# training the kmeans model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)
y_predict = kmeans.fit_predict(x)
print(y_predict)

# visualizing clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s=100, c='red', label='cluster1')
```

```
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s=100, c='green', label='cluster2')
mtp.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s=100, c='blue', label='cluster3')
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s=100, c='cyan', label='cluster4')
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s=100, c='yellow', label='cluster5')
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='orange',
label='center')
mtp.title('clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending score (1-100)')
mtp.legend()
mtp.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]
 [ 25   5]
 [ 25  73]
 [ 28  14]
 [ 28  82]
 [ 28  32]
 [ 28  61]
 [ 29  31]
```

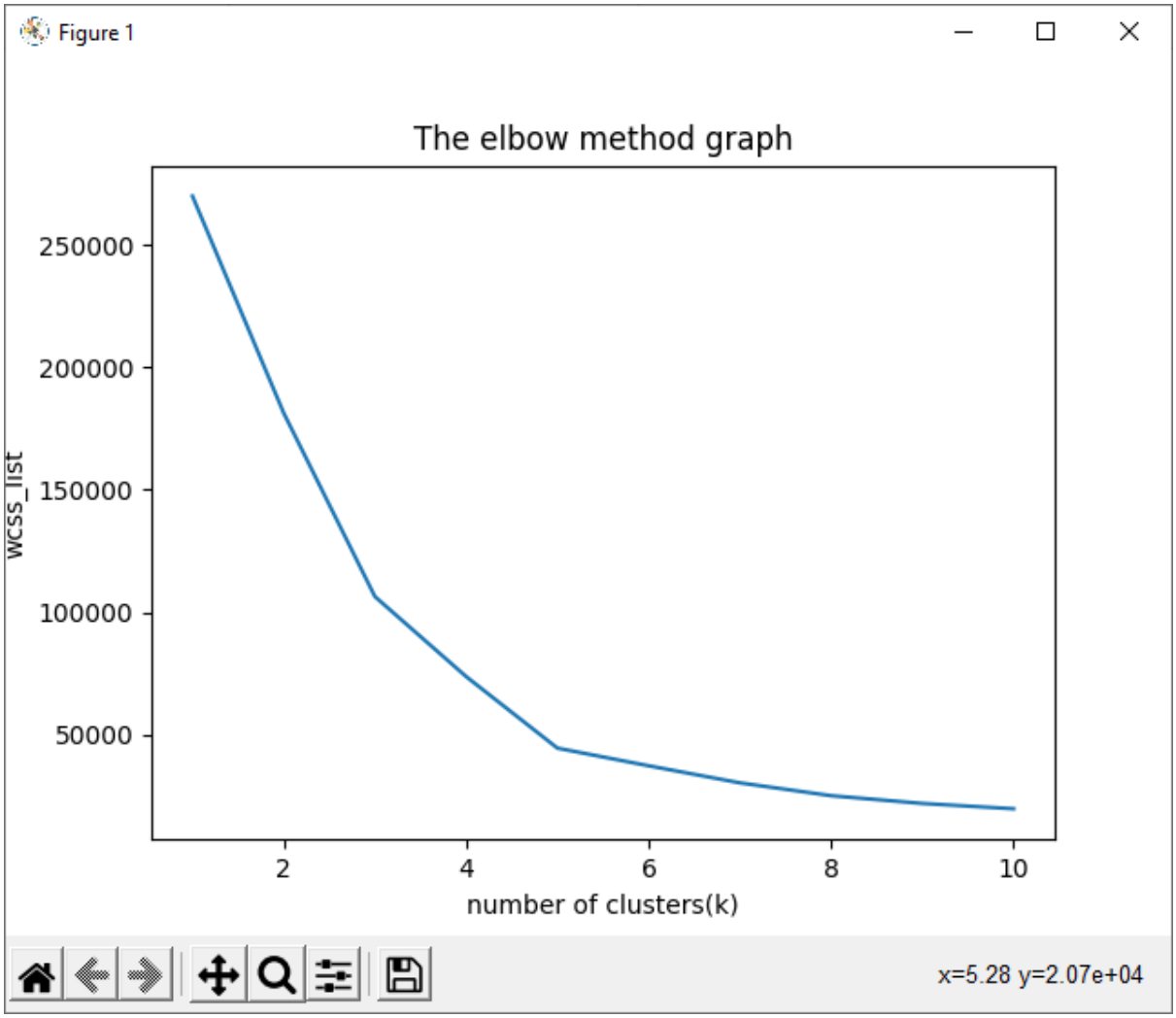


Figure 1
The elbow method graph
250000
200000
150000
100000
50000
wcss_list
2
4
6
8
10
number of clusters(k)
x=5.28 y=2.07e+04

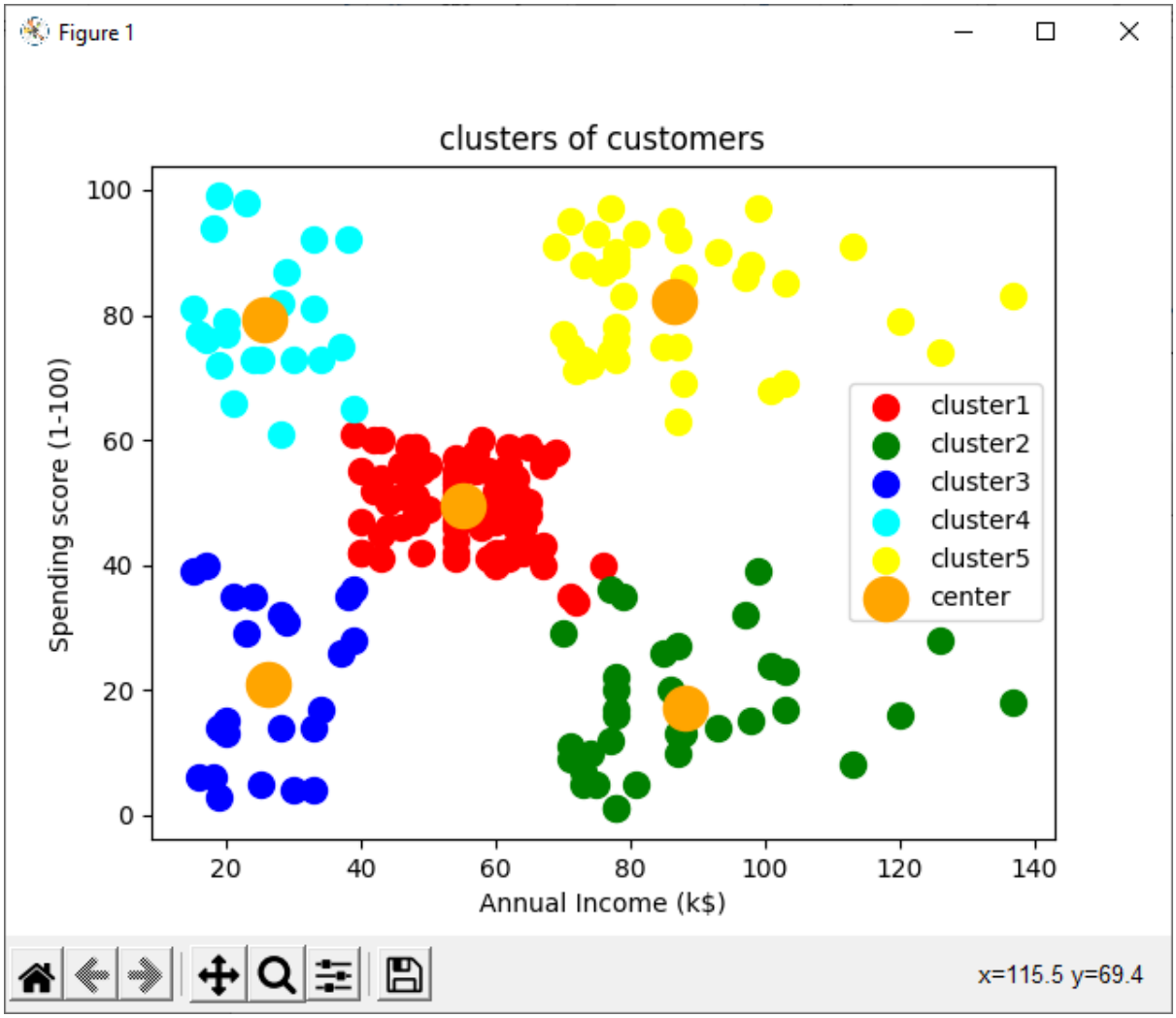


Figure 1
clusters of customers
Spending score (1-100)
Annual Income (k$)
100
80
60
40
20
0
20
40
60
80
100
120
140
cluster1
cluster2
cluster3
cluster4
cluster5
center
x=115.5 y=69.4

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 Code :

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

# importing dataset
dataset = pd.read_csv("world_country.csv")
x = dataset.iloc[:, [1, 2]].values
print(x)
# finding optimal no of clusters using elbow
from sklearn.cluster import KMeans

wcss_list = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The elbow method graph')
mtp.xlabel('number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
# training the kmeans model on a dataset
kmeans = KMeans(n_clusters=3, init='k-means++', random_state=42)
y_predict = kmeans.fit_predict(x)
print(y_predict)

# visualizing clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s=100, c='red', label='cluster1')
```

```
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s=100, c='green', label='cluster2')
mtp.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s=100, c='blue', label='cluster3')
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='orange',
label='centeroid')
mtp.title('Country clusters')
mtp.xlabel('latitude')
mtp.ylabel('longitude')
mtp.legend()
mtp.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]
 [ 4.75162310e+01  1.45500720e+01]
 [-2.52743980e+01  1.33775136e+02]
 [ 1.25211100e+01 -6.99683380e+01]
 [ 4.01431050e+01  4.75769270e+01]
 [ 4.39158860e+01  1.76790760e+01]
 [ 1.31938870e+01 -5.95431980e+01]
 [ 2.36849940e+01  9.03563310e+01]
 [ 5.05038870e+01  4.46993600e+00]
 [ 1.22383330e+01 -1.56159300e+00]
 [ 4.27338830e+01  2.54858300e+01]
 [ 2.59304140e+01  5.06377720e+01]
 [-3.37305600e+00  2.99188860e+01]
 [ 9.30769000e+00  2.31583400e+00]
 [ 3.23213840e+01 -6.47573700e+01]
 [ 4.53527700e+00  1.14727669e+02]
 [-1.62901540e+01 -6.35886530e+01]
 [-1.42350040e+01 -5.19252800e+01]
```

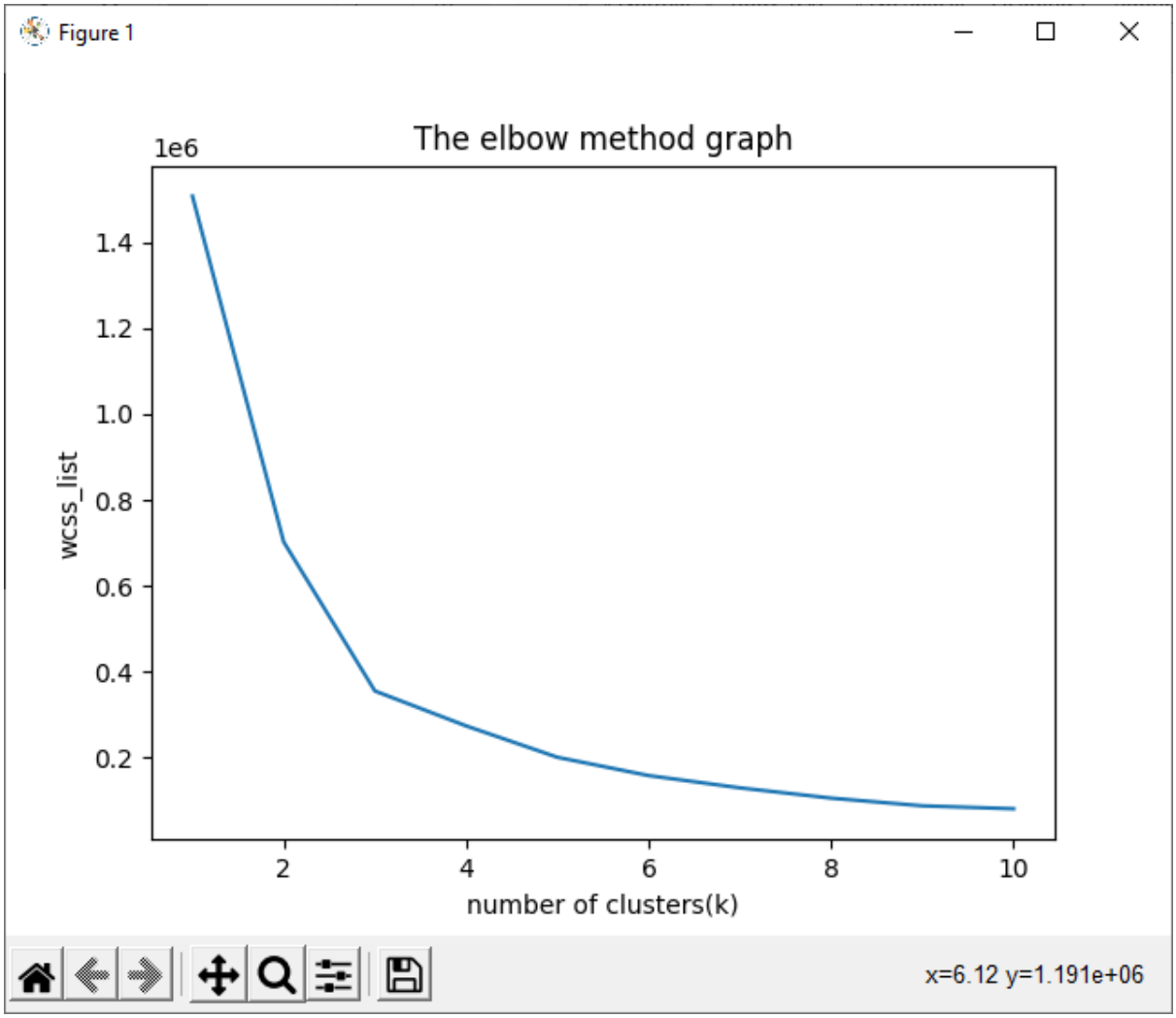


Figure 1
The elbow method graph
1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
wcss_list
2
4
6
8
10
number of clusters(k)
x=6.12 y=1.191e+06

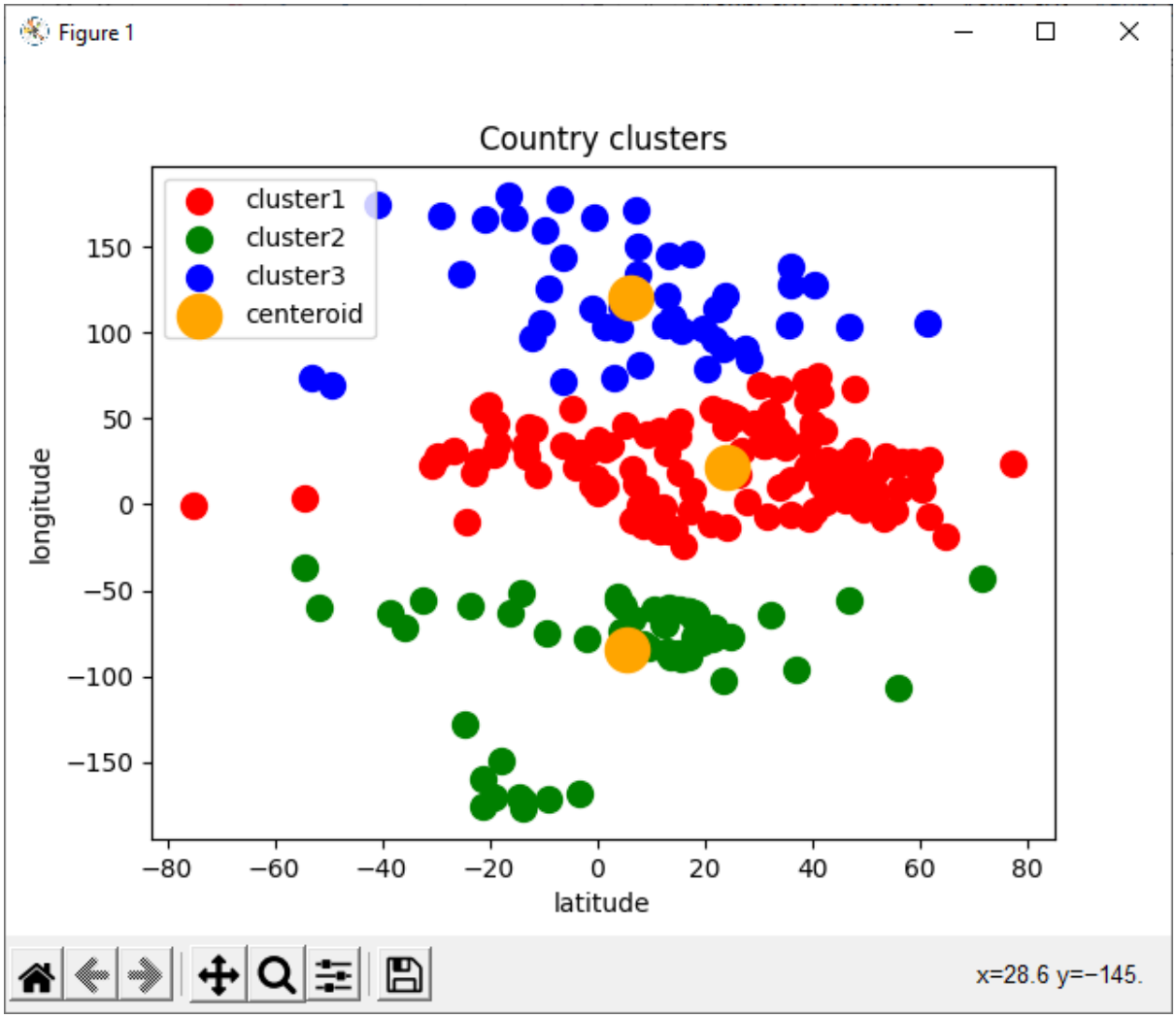


Figure 1
Country clusters
cluster1
cluster2
cluster3
centeroid
longitude
latitude
150
100
50
0
−50
−100
−150
−80
−60
−40
−20
0
20
40
60
80
x=28.6 y=−145.

PROGRAM NO : 13 **Date:02/02/2022**

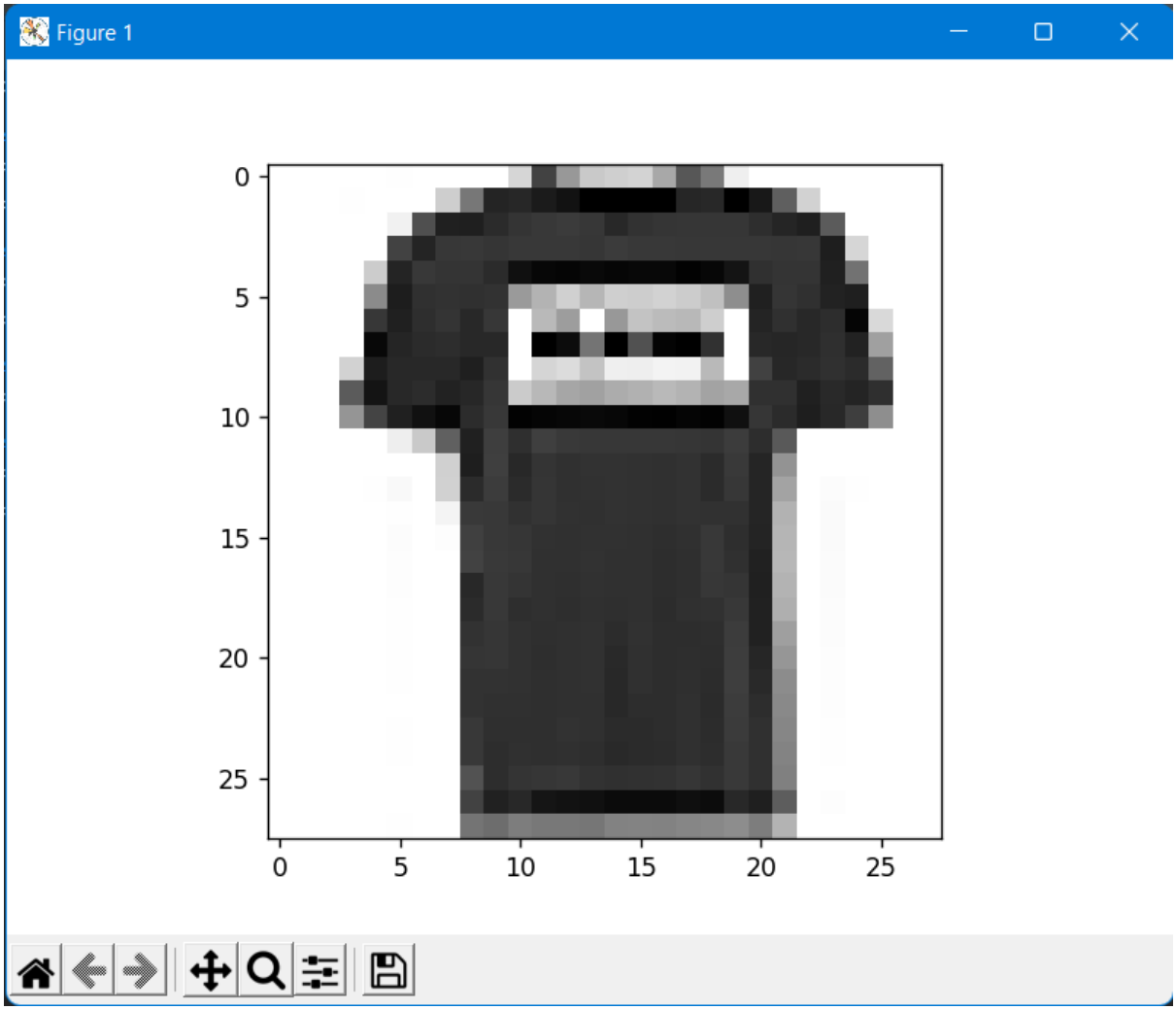
AIM : Programs on convolutional neural network to classify images from any standard dataset in the public domain

Program Code :

```
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:



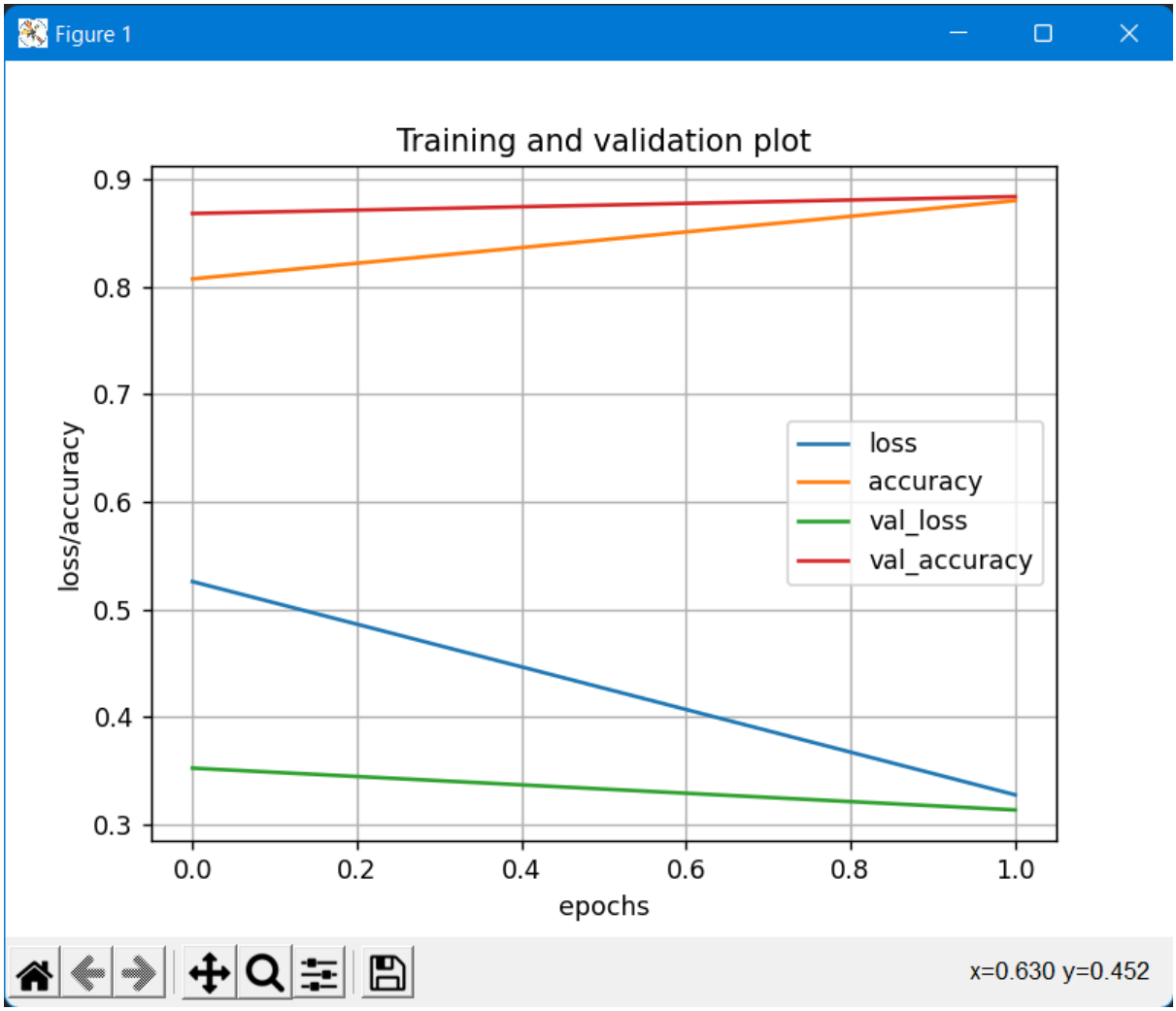


Figure 1
Training and validation plot
loss/accuracy
epochs
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.2
0.4
0.6
0.8
1.0
loss
accuracy
val_loss
val_accuracy
x=0.630 y=0.452

```
_________________________________________________________________
 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
```

```
 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 [==============================] - 51s 30ms/step - loss: 0.5258 - accuracy: 0.8072 - val_loss: 0.3524 - val_accuracy: 0.8680
Epoch 2/2
1688/1688 [==============================] - 46s 27ms/step - loss: 0.3276 - accuracy: 0.8800 - val_loss: 0.3135 - val_accuracy: 0.8837
313/313 [==============================] - 2s 7ms/step - loss: 0.3264 - accuracy: 0.8765
 Test Loss :0.3264442980289459, Test Accuracy : 0.8765000104904175

Process finished with exit code 0
```

PROGRAM NO : 14 **Date:16/02/2022**

AIM : Program to implement a simple web crawler using python

Program Code :

```
import requests
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, 'lxml')
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
movie_url = urls
movie_f = requests.get(movie_url, headers=headers)
movie_soup = BeautifulSoup(movie_f.content, 'lxml')
movie_content = movie_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:

```
[<a class="unstyled articleLink" href="/m/it_happened_one_night">
            It Happened One Night (1934)</a>, <a class="unstyled articleLink" href="/m/citizen_kane">
            Citizen Kane (1941)</a>, <a class="unstyled articleLink" href="/m/the_wizard_of_oz_1939">
            The Wizard of Oz (1939)</a>, <a class="unstyled articleLink" href="/m/modern_times">
            Modern Times (1936)</a>, <a class="unstyled articleLink" href="/m/black_panther_2018">
            Black Panther (2018)</a>, <a class="unstyled articleLink" href="/m/parasite_2019">
            Parasite (Gisaengchung) (2019)</a>, <a class="unstyled articleLink" href="/m/avengers_endgame">
            Avengers: Endgame (2019)</a>, <a class="unstyled articleLink" href="/m/1003707-casablanca">
            Casablanca (1942)</a>, <a class="unstyled articleLink" href="/m/knives_out">
            Knives Out (2019)</a>, <a class="unstyled articleLink" href="/m/us_2019">
            Us (2019)</a>, <a class="unstyled articleLink" href="/m/toy_story_4">
            Toy Story 4 (2019)</a>, <a class="unstyled articleLink" href="/m/lady_bird">
            Lady Bird (2017)</a>, <a class="unstyled articleLink" href="/m/mission_impossible_fallout">
            Mission: Impossible - Fallout (2018)</a>, <a class="unstyled articleLink" href="/m/blackkklansman">
            BlacKkKlansman (2018)</a>, <a class="unstyled articleLink" href="/m/get_out">
            Get Out (2017)</a>, <a class="unstyled articleLink" href="/m/the_irishman">
            The Irishman (2019)</a>, <a class="unstyled articleLink" href="/m/godfather">
            The Godfather (1972)</a>, <a class="unstyled articleLink" href="/m/mad_max_fury_road">
            Mad Max: Fury Road (2015)</a>, <a class="unstyled articleLink" href="/m/spider_man_into_the_spider_verse">
            Spider-Man: Into the Spider-Verse (2018)</a>, <a class="unstyled articleLink" href="/m/moonlight_2016">
            Moonlight (2016)</a>, <a class="unstyled articleLink" href="/m/sunset_boulevard">
            Sunset Boulevard (1950)</a>, <a class="unstyled articleLink" href="/m/1000626-all_about_eve">
            All About Eve (1950)</a>, <a class="unstyled articleLink" href="/m/the_cabinet_of_dr_caligari">
```

PROGRAM NO : 15 **Date:16/02/2022**

AIM : Program to implement a simple web crawler using python

Program Code :

```
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:

```
Wikipedia, the free encyclopedia;Main Page;/wiki/Wikipedia
Wikipedia, the free encyclopedia;Main Page;/wiki/Free_content
Wikipedia, the free encyclopedia;Main Page;/wiki/Encyclopedia
Wikipedia, the free encyclopedia;Main Page;/wiki/English_language
Wikipedia, the free encyclopedia;Main Page;/wiki/SS_Choctaw
Wikipedia, the free encyclopedia;Main Page;/wiki/Cargo_ship
Wikipedia, the free encyclopedia;Main Page;/wiki/Great_Lakes
Wikipedia, the free encyclopedia;Main Page;/wiki/Lake_freighter
Wikipedia, the free encyclopedia;Main Page;/wiki/Whaleback
Wikipedia, the free encyclopedia;Main Page;/wiki/Alexander_McDougall_(ship_designer)
Wikipedia, the free encyclopedia;Main Page;/wiki/American_Ship_Building_Company
Wikipedia, the free encyclopedia;Main Page;/wiki/Cleveland
Wikipedia, the free encyclopedia;Main Page;/wiki/Michigan
Wikipedia, the free encyclopedia;Main Page;/wiki/Detroit
Wikipedia, the free encyclopedia;Main Page;/wiki/Escanaba,_Michigan
Wikipedia, the free encyclopedia;Main Page;/wiki/Marquette,_Michigan
Wikipedia, the free encyclopedia;Main Page;/wiki/Glossary_of_nautical_terms#upbound
Wikipedia, the free encyclopedia;Main Page;/wiki/Iron_ore
Wikipedia, the free encyclopedia;Main Page;/wiki/Lake_Huron
Wikipedia, the free encyclopedia;Main Page;/wiki/New_Presque_Isle_Light
Wikipedia, the free encyclopedia;Main Page;/wiki/Glossary_of_nautical_terms#canaller
Wikipedia, the free encyclopedia;Main Page;/wiki/National_Register_of_Historic_Places
Wikipedia, the free encyclopedia;Main Page;/wiki/David_Berman_(musician)
Wikipedia, the free encyclopedia;Main Page;/wiki/The_Beautician_and_the_Beast
Wikipedia, the free encyclopedia;Main Page;/wiki/Great_Western_Railway_War_Memorial
```

PROGRAM NO : 16 **Date:16/02/2022**

AIM : Program to implement scrap of any website

Program Code :

```
# program to scrap websites and save quotes from website
import requests
from bs4 import BeautifulSoup
import csv
import lxml

URL = "http://www.values.com/inspirational-quotes"
r = requests.get(URL)
print(r.content)

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

# list to store quotes
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'}):

    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:

```
b'<!DOCTYPE html>\n<html class="no-js" dir="ltr" lang="en-US">\n    <head>\n        <title>Inspirational Quotes - Motivational Quotes -
 Leadership Quotes | PassItOn.com</title>\n        <meta charset="utf-8">\n        <meta http-equiv="content-type" content="text/html;
 charset=utf-8" />\n        <meta http-equiv="X-UA-Compatible" content="IE=edge" />\n        <meta name="viewport" content="width=device-width,
 initial-scale=1.0" />\n        <meta name="description" content="The Foundation for a Better Life | Pass It On.com">\n        <link
 rel="apple-touch-icon" sizes="180x180" href="/apple-touch-icon.png">\n        <link rel="icon" type="image/png" sizes="32x32"
 href="/favicon-32x32.png">\n        <link rel="icon" type="image/png" sizes="16x16" href="/favicon-16x16.png">\n        <link rel="manifest"
 href="/site.webmanifest">\n        <link rel="mask-icon" href="/safari-pinned-tab.svg" color="#c8102e">\n        <meta
 name="msapplication-TileColor" content="#c8102e">\n        <meta name="theme-color" content="#ffffff">\n        <link rel="stylesheet"
 href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
 integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">\n        <link rel="stylesheet"
 media="all" href="/assets/application-2a7a8e6a1c3f620bac9efa66420f5579.css" />\n        <meta name="csrf-param" content="authenticity_token"
 />\n<meta name="csrf-token" content="XJy5ztVQDsOgVqPETF64tk4vUqX3ZFCTLvcSHUnnj/BfEX6ofIjyEcVqn7tHD7Px9q4RXcxmnIucNzzdBNUJ3w==" />\n        <!--
 Global site tag (gtag.js) - Google Analytics -->\n        <script async src="https://www.googletagmanager
 .com/gtag/js?id=UA-1179606-29"></script>\n        <script>\n          window.dataLayer = window.dataLayer || [];\n          function gtag()
 {dataLayer.push(arguments);}\n          gtag(\'js\', new Date());\n          gtag(\'config\', \'UA-1179606-29\');\n        </script>\n\n
 <script>\n          window.fbAsyncInit = function() {\n            FB.init({\n              appId            : \'483774921971842\',\n
    autoLogAppEvents : true,\n              xfbml            : true,\n              version          : \'v6.0\'\n            });\n          };\n
       </script>\n        <script async defer src="https://connect.facebook.net/en_US/sdk.js"></script>\n\n        <meta property="og:site_name"
  content="passiton.com">\n        <meta property="og:title" content="Inspirational Quotes - Motivational Quotes - Leadership Quotes" />\n
  <meta property="og:type" content="website" />\n        <meta property="og:image" content="https://www.passiton.com/passiton.jpg" />\n
 <meta property="og:url" content="https://www.passiton.com/inspirational-quotes" />\n        <meta property="og:description" content="Find the
```

PROGRAM NO : 17 **Date:16/02/2022**

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

Program Code :

```
# creating a function to generate N-grams
def generate(text, WordsToCombine):
    words = text.split()
    output = []
    for i in range(len(words) - WordsToCombine + 1):
        output.append(words[i:i + WordsToCombine])
    return output


x=generate(text='this is a very good boook to study', WordsToCombine=3)
print(x)
```

Output:

```
[['this', 'is', 'a'], ['is', 'a', 'very'], ['a', 'very', 'good'], ['very', 'good', 'boook'], ['good', 'boook', 'to'], ['boook', 'to', 'study']]

Process finished with exit code 0
```

PROGRAM NO : 18 **Date:16/02/2022**

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

Program Code :

```
import nltk
from nltk.util import ngrams

sample = 'this is a very good book to study'
NGRAMS = ngrams(sequence=nltk.word_tokenize(sample), n=2)
for grams in NGRAMS:
    print(grams)
```

Output:

```
('this', 'is')
('is', 'a')
('a', 'very')
('very', 'good')
('good', 'book')
('book', 'to')
('to', 'study')

Process finished with exit code 0
```

PROGRAM NO : 19 **Date:16/02/2022**

AIM : Program for Natural Language Processing which performs speech tagging

Program Code :

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize


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

# Dummy text
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 exiting 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:
    # words tokenizers is used to find the words
    # and punctuation in a string
    wordslist = nltk.word_tokenize(i)

    # removing stop words from word list
    wordslist = [w for w in wordslist if not w in stop_words]

    # using  a Tagger . which is part of speech
```

```
# taggger or Pos-tagger
tagged = nltk.pos_tag(wordslist)

print(tagged)
```

Output:

```
[('Sukanya', 'NNP'), (',', ','), ('Rajib', 'NNP'), ('Naba', 'NNP'), ('good', 'JJ'), ('friends', 'NNS'), (',', ','), ('Sukanya', 'NNP'),
 ('getting', 'VBG'), ('married', 'VBD'), ('next', 'JJ'), ('year.Marriage', 'NN'), ('big', 'JJ'), ('step', 'NN'), ('one', 'CD'), ("'s", 'POS'),
 ('life.It', 'NN'), ('exiting', 'VBG'), ('frightening.But', 'JJ'), ('friendship', 'NN'), ('sacred', 'VBD'), ('bond', 'NN'), ('people.It', 'NN'),
 ('special', 'JJ'), ('kind', 'NN'), ('love', 'NN'), ('usMany', 'NN'), ('must', 'MD'), ('tried', 'VB'), ('searching', 'VBG'), ('friendbut', 'NN'),
  ('never', 'RB'), ('found', 'VBD'), ('right', 'JJ'), ('one', 'CD'), ('.', '.')]

Process finished with exit code 0
```

PROGRAM NO : 20 **Date:23/02/2022**

AIM : Program for Natural language processing which perform chunking

Program Code :

```
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=r"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'), ('the', 'DT'), ('fresh', 'JJ'), ('cheese', 'NN')]
(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))
```

PROGRAM NO : 21 **Date:23/02/2022**

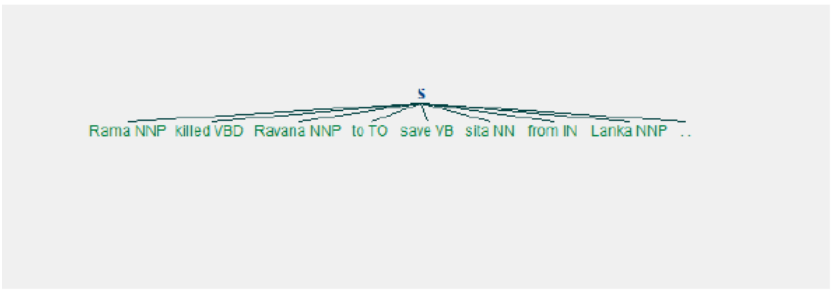
AIM : Write a python program for natural program language processing with chunking.

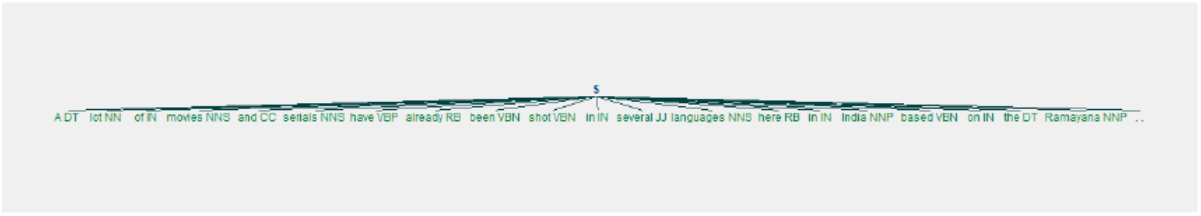
Program Code :

```
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)
tagged_words = nltk.pos_tag(words)
chunkGram = r"""VB: {}"""
chunkParser = nltk.RegexpParser(chunkGram)
chunked = chunkParser.parse(tagged_words)
print(chunked)
chunked.draw()
```

Output:

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\ajcemca\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
(S
  Rama/NNP
  killed/VBD
  Ravana/NNP
  to/TO
  save/VB
  sita/NN
  from/IN
  Lanka/NNP
  ./.)
```


S
Rama NNP killed VBD Ravana NNP to TO save VB sita NN from IN Lanka NNP ..


S
A DT lot NN of IN movies NNS and CC serials NNS have VBP already RB been VBN shot VBN in IN several JJ languages NNS here RB in IN India NNP based VBN on IN the DT Ramayana NNP ..