Musaliar College of Engineering & Technology

MUSALIAR COLLEGE P.O., PATHANAMTHITTA - 689 653

..... -.....

LABORATORY RECORD

Certified that this is the Bonafide Record of the work done by

Sri/Smt.....

of.....Semester Class of .....(Roll No .....)

of ..... Branch

in the ..... Laboratory

during the academic year 2022 - 2024

Name of Examination

Reg. No. External Examiner Staff in- charge

DEPARTMENT OF COMPUTER APPLICATIONS

VISION

"To produce competent and dynamic professionals in the field of Computer Applications to thriveand cater the changing needs of the society through research and education".

MISSION

To impart high quality technical education and knowledge in Computer Applications.

To introduce moral, ethical and social values to Computer Application students.

To establish industry institute interaction to enhance the skills of Computer Application students.

To promote research aimed towards betterment of society.

INDEX

SL.N O	DATE	NAME OF EXPERIMENT	PAGE NO.	COURSE OUTCOME	REMARKS

EXPERIMENT NO: 1

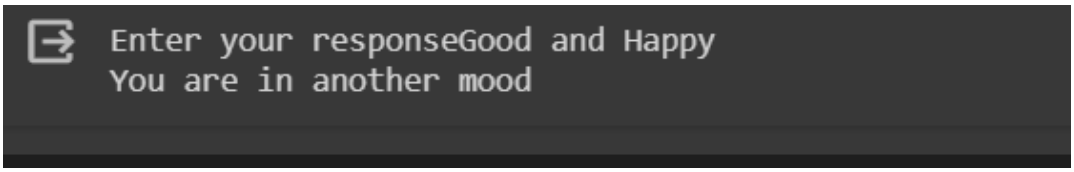
AIM:

Python program to analyse social media review.

PROGRAM:

```
l1=['Good','Fine','Nice','Happy','Positive']
l2=['Sad','Tired','Bad','Frustrated','Not','Negative']
str1=input("Enter your response")
flag=0
ncount=0
t=str1.split()
for i in range(len(t)):
  for j in range(len(l1)):
    if t[1]==l1[j]:
      flag=1
  for k in range(len(l2)):
    if t[i]==l2[k]:
      ncount=ncount+1
      flag=1
if flag==0:
   print("You are in another mood")
elif ncount%2==0:
  print("Positive")
else:
  print('Negative')
```

OUTPUT:



EXPERIMENT NO: 2

AIM:

Python program to print the QR Code of a website.

PROGRAM:

```
import pyqrcode
import png
from pyqrcode import QRCode

# String which represents the QR code
s = input("enter your url")
myqr=input("enter your file name")
# Generate QR code
url = pyqrcode.create(s)

# Create and save the svg file naming "myqr.svg"
url.svg(myqr+".svg", scale = 8)

# Create and save the png file naming "myqr.png"
url.png(myqr+'.png', scale = 6)
```

OUTPUT:

EXPERIMENT NO: 3

AIM:

Program using matplotlib to analyse the data on graph and pie chart.

PROGRAM:

For Graph

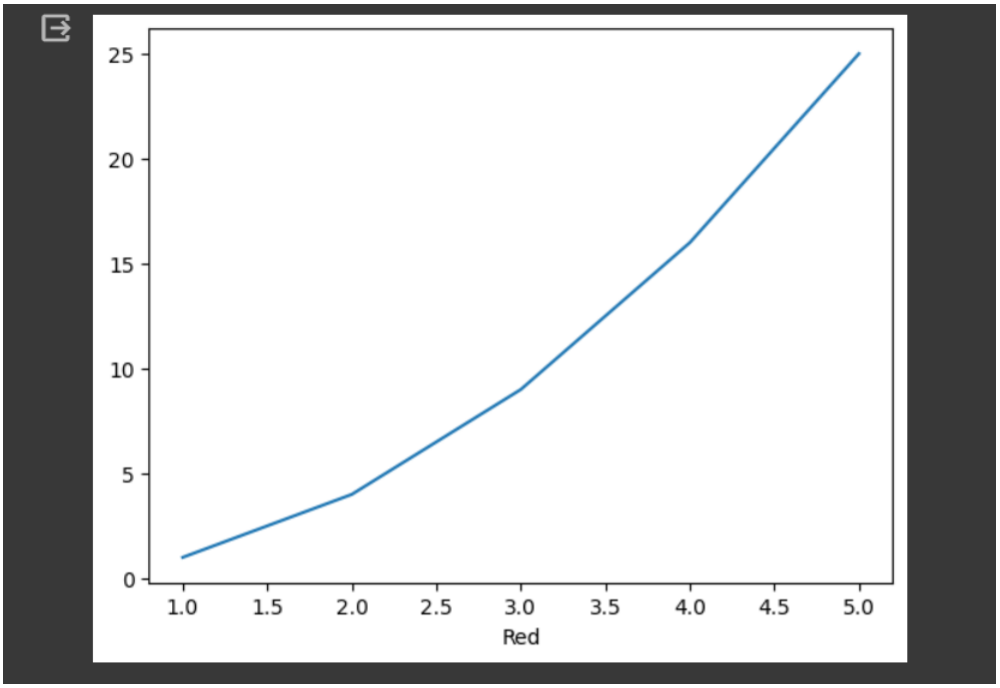
```
import matplotlib.pyplot as plt
x = [1,2,3,4,5]
y = [1,4,9,16,25]
plt.xlabel("Red")
plt.plot(x,y)
plt.show()
```

For Pie Chart

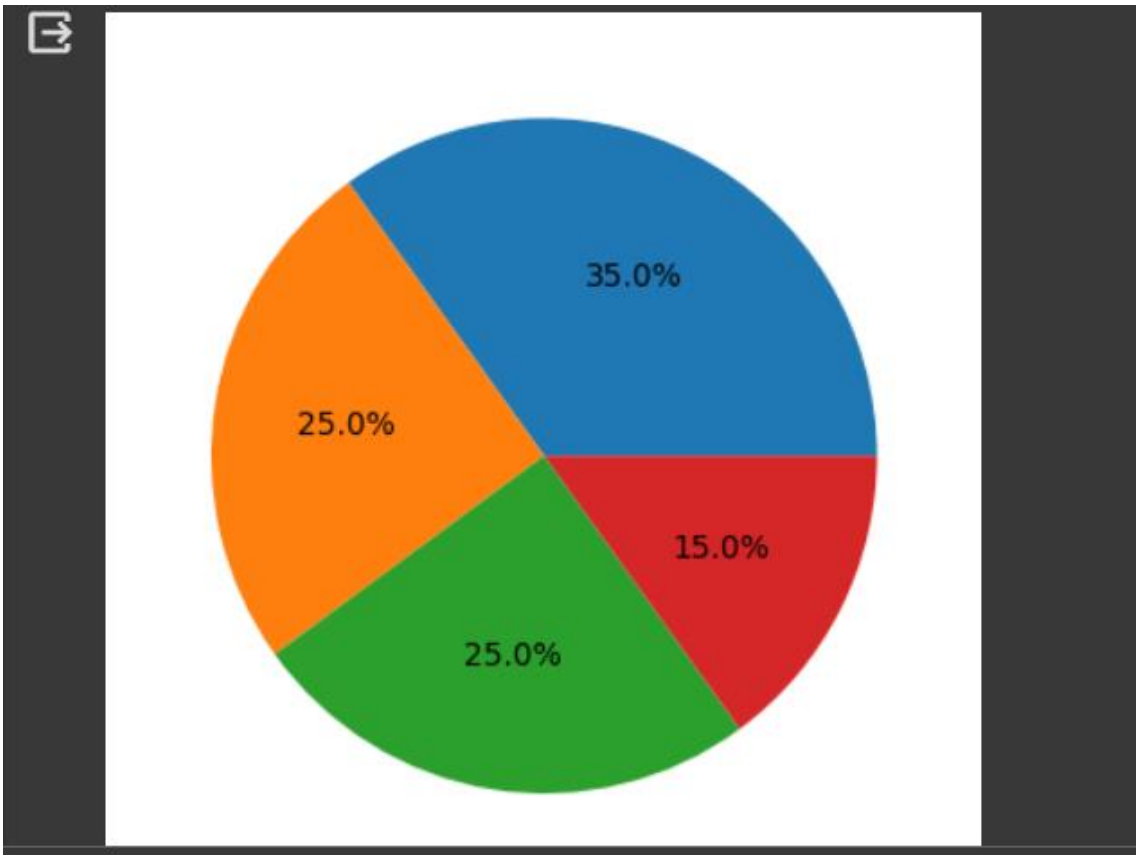
```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
plt.pie(y,autopct='%1.1f%%')
plt.show()
```

OUTPUT:

For Graph



For Pie Chart



EXPERIMENT NO: 4

AIM:

Perform the following Matrix Operations:

1.Dot Product of Matrix

2.Transpose of Matrix

3.Determinant of Matrix

4.Trace of Matrix

5.Inverse of Matrix

6.Rank of Matrix

7.Eigen Values and Eigen Vector

PROGRAM:

1. Dot Product of Matrix

```
import numpy as np
def create_matrix(mc):
   print("Array" +str(mc)+ "Element")
   array_1 = map(int,input().split())
   array_1 = np.array(list(array_1))
   print("\n Array" + str(mc) +"Rowcolumn :")
   row,column = map(int,input().split())
   if(len(array_1) != (row*column)):
      print("Row_column size do not match")
      return create_matrix(mc)
   array_1 = array_1.reshape(row,column)
   print("\n Array" + str(mc))
   print(array_1)
   return array_1
arr1 = create_matrix(1)
arr2 = create_matrix(2)
if(arr1.shape == (arr2.shape)):
   print("Dot product")
   print(np.dot(arr1,arr2))
else:
   print("Dimension do not match")
```

OUTPUT:

```
Array1Element
1 2 3 4

 Array1Rowcolumn :
2 2

 Array1
[[1 2]
 [3 4]]
Array2Element
2 3 4 5

 Array2Rowcolumn :
2 2

 Array2
[[2 3]
 [4 5]]
Dot product
[[10 13]
 [22 29]]
```

2. Transpose of Matrix

```
import numpy as np
def create_matrix(mc):
    print("\nARRAY "+str(mc)+" Elements : ")
    array_1 = map(int, input().split())
    array_1 = np.array(list(array_1))
    #print(arr)
    print("\nARRAY "+str(mc)+" ,  ROW COLUMN : ")
    row,column = map(int, input().split())
    if(len(array_1)!= (row*column)):
        print("\nRow and Column size not match with total elements !! retry")
        return create_matrix(mc)
    array_1 = array_1.reshape(row,column)
    print("\nARRAY "+str(mc))
    print(array_1)
    print("\nTranspose : ")
    return array_1
print(create_matrix(1).transpose())
```

OUTPUT:

```
ARRAY 1 Elements :
1 2 3 4

ARRAY 1 ,  ROW COLUMN :
2 2

ARRAY 1
[[1 2]
 [3 4]]

Transpose :
[[1 3]
 [2 4]]
```

3. Determinant of Matrix

```
import numpy as np

def create_matrix(mc):
   print("\nARRAY "+str(mc)+" Elements : ")
   array_1 = map(int, input().split())
   array_1 = np.array(list(array_1))
   #print(arr)
   print("\nARRAY "+str(mc)+" ,  ROW COLUMN : ")
   row,column = map(int, input().split())
   if(len(array_1)!= (row*column)):
      print("\nRow and Column size not match with total elements !! retry")
      return create_matrix(mc)
   array_1 = array_1.reshape(row,column)
   print("\nARRAY "+str(mc))
   print(array_1)
   print("\nDeterminant : ")
   return array_1

print(np.linalg.det(create_matrix(1)))
```

OUTPUT:

```
ARRAY 1 Elements :
2 3 4 5

ARRAY 1 ,  ROW COLUMN :
2 2

ARRAY 1
[[2 3]
 [4 5]]

Determinant :
-2.0
```

4. Trace of Matrix

```
import numpy as np

def create_matrix(mc):
   print("\nARRAY "+str(mc)+" Elements : ")
   array_1 = map(int, input().split())
   array_1 = np.array(list(array_1))
   #print(arr)
   print("\nARRAY "+str(mc)+" ,  ROW COLUMN : ")
   row,column = map(int, input().split())
   if(len(array_1)!= (row*column)):
      print("\nRow and Column size not match with total elements !! retry")
      return create_matrix(mc)
   array_1 = array_1.reshape(row,column)
   print("\nARRAY "+str(mc))
   print(array_1)
   print("\nTrace : ")
   return array_1

print(create_matrix(1).trace())
```

OUTPUT:

```
ARRAY 1 Elements :
1 2 3 4

ARRAY 1 ,  ROW COLUMN :
2 2

ARRAY 1
[[1 2]
 [3 4]]

Trace :
5
```

5. Inverse of Matrix

```
import numpy as np
def create_matrix(mc):
   print("\nARRAY "+str(mc)+" Elements : ")
   array_1 = map(int, input().split())
   array_1 = np.array(list(array_1))
   #print(arr)
   print("\nARRAY "+str(mc)+" ,  ROW COLUMN : ")
   row,column = map(int, input().split())
   if(len(array_1)!= (row*column)):
      print("\nRow and Column size not match with total elements !! retry")
      return create_matrix(mc)
   array_1 = array_1.reshape(row,column)
```

```
   print("\nARRAY "+str(mc))
   print(array_1)
   print("\nInverse : ")
   return array_1
print(np.linalg.inv(create_matrix(1)))
```

OUTPUT:

```
ARRAY 1 Elements :
1 2 3 4

ARRAY 1 ,  ROW COLUMN :
2 2

ARRAY 1
[[1 2]
 [3 4]]

Inverse :
[[-2.   1. ]
 [ 1.5 -0.5]]
```

6. Rank of Matrix

```
import numpy as np
def create_matrix(mc):
   print("\nARRAY "+str(mc)+" Elements : ")
   array_1 = map(int, input().split())
   array_1 = np.array(list(array_1))
   #print(arr)
   print("\nARRAY "+str(mc)+" ,  ROW COLUMN : ")
   row,column = map(int, input().split())
   if(len(array_1)!= (row*column)):
      print("\nRow and Column size not match with total elements !! retry")
      return create_matrix(mc)
   array_1 = array_1.reshape(row,column)
   print("\nARRAY "+str(mc))
   print(array_1)
   print("\nRank : ")
   return array_1
print(np.linalg.matrix_rank(create_matrix(1)))
```

OUTPUT:

```
ARRAY 1 Elements :
1 2 3 4

ARRAY 1 ,  ROW COLUMN :
2 2
```

```
ARRAY 1
[[1 2]
 [3 4]]

Rank :
2
```

7. Eigen Values and Eigen Vectors

```
import numpy as np
def create_matrix(mc):
   print("\nARRAY "+str(mc)+" Elements : ")
   array_1 = map(int, input().split())
   array_1 = np.array(list(array_1))
   #print(arr)
   print("\nARRAY "+str(mc)+" ,  ROW COLUMN : ")
   row,column = map(int, input().split())
   if(len(array_1)!= (row*column)):
      print("\nRow and Column size not match with total elements !! retry")
      return create_matrix(mc)
   array_1 = array_1.reshape(row,column)
   print("\nARRAY "+str(mc))
   print(array_1)
   return array_1

x,y = np.linalg.eig(create_matrix(1))

print("\nE-value : ")
print(x)
print("\nE-vector : ")
print(y)
```

OUTPUT :

```
ARRAY 1 Elements :
1 2 3 4

ARRAY 1 ,  ROW COLUMN :
2 2

ARRAY 1
[[1 2]
 [3 4]]

E-value :
[-0.37228132  5.37228132]

E-vector :
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

EXPERIMENT NO: 5

AIM:

Program to implement Naive Bayes classification using any dataset Input:

PROGRAM:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
X,y=load_iris(return_X_y=True)
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.5,random_state=0)
gnb=GaussianNB()
y_pred=gnb.fit(X_train,y_train).predict(X_test)
print(y_pred)
x_new=[[5,5,4,4]]
y_new=gnb.fit(X_train,y_train).predict(x_new)
print("predicted output for [[5,5,4,4]]:",y_new)
print("Naive Bayes score:",gnb.score(X_test,y_test))
```

OUTPUT:

```
[2 1 0 2 0 2 0 1 1 1 1 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1 1 1 2 0 2 0 0 1 2 2 1 2 1 2 1 1 2 1 1 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0
 1]
predicted output for [[5,5,4,4]]: [2]
Naive Bayes score: 0.9466666666666667
```

EXPERIMENT NO: 6

AIM:

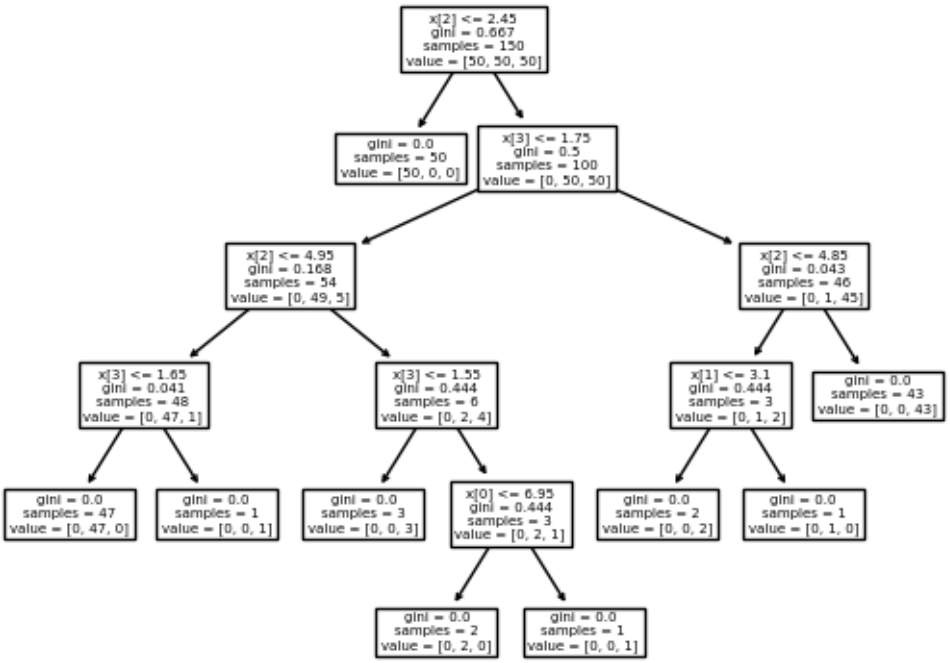
Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.

PROGRAM:

```
from sklearn.datasets import load_iris
from sklearn import tree
iris=load_iris()
X,y=iris.data, iris.target
clf=tree.DecisionTreeClassifier()
clf=clf.fit(X,y)
tree.plot_tree(clf)
```

OUTPUT:

```
[Text(0.5, 0.9166666666666666, 'x[2] <= 2.45\ngini = 0.667\nsamples = 150\nvalue = [50,
50, 50]'),
 Text(0.4230769230769231, 0.75, 'gini = 0.0\nsamples = 50\nvalue = [50, 0, 0]'),
 Text(0.5769230769230769, 0.75, 'x[3] <= 1.75\ngini = 0.5\nsamples = 100\nvalue = [0,
50, 50]'),
 Text(0.3076923076923077, 0.5833333333333334, 'x[2] <= 4.95\ngini = 0.168\nsamples =
54\nvalue = [0, 49, 5]'),
 Text(0.15384615384615385, 0.4166666666666667, 'x[3] <= 1.65\ngini = 0.041\nsamples =
48\nvalue = [0, 47, 1]'),
 Text(0.07692307692307693, 0.25, 'gini = 0.0\nsamples = 47\nvalue = [0, 47, 0]'),
 Text(0.23076923076923078, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.46153846153846156, 0.4166666666666667, 'x[3] <= 1.55\ngini = 0.444\nsamples =
6\nvalue = [0, 2, 4]'),
 Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.5384615384615384, 0.25, 'x[0] <= 6.95\ngini = 0.444\nsamples = 3\nvalue = [0,
2, 1]'),
 Text(0.46153846153846156, 0.08333333333333333, 'gini = 0.0\nsamples = 2\nvalue = [0,
2, 0]'),
 Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
1]'),
 Text(0.8461538461538461, 0.5833333333333334, 'x[2] <= 4.85\ngini = 0.043\nsamples =
46\nvalue = [0, 1, 45]'),
 Text(0.7692307692307693, 0.4166666666666667, 'x[1] <= 3.1\ngini = 0.444\nsamples =
3\nvalue = [0, 1, 2]'),
 Text(0.6923076923076923, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.8461538461538461, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9230769230769231, 0.4166666666666667, 'gini = 0.0\nsamples = 43\nvalue = [0, 0,
43]')]
```


x[2] <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
gini = 0.0
samples = 50
value = [50, 0, 0]
x[3] <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
x[2] <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
x[2] <= 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
x[3] <= 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
x[3] <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
x[1] <= 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 43
value = [0, 0, 43]
gini = 0.0
samples = 47
value = [0, 47, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
gini = 0.0
samples = 3
value = [0, 0, 3]
x[0] <= 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]

EXPERIMENT NO: 7

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
irisdata=load_iris()
print(irisdata.data)
x=irisdata.data
y=irisdata.target
x_train ,x_test, y_train, y_test=train_test_split(x, y )
Knn=KNeighborsClassifier(n_neighbors=1)
Knn.fit(x_train,y_train)
print(x_test)
print(Knn.predict(x_test))
print(y_test)
result=Knn.predict([[2,4,6,2]])
print(irisdata.target_names[result])
```

OUTPUT:

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
[5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
[5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
```

```
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]
 [7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.
 [5.2 2.7 3.9 1.4]
 [5.4 3.  4.5 1.5]
 [6.8 2.8 4.8 1.4]
 [6.7 3.1 5.6 2.4]
 [6.3 3.4 5.6 2.4]
 [5.8 2.6 4.  1.2]
 [6.5 3.2 5.1 2. ]
 [5.  2.3 3.3 1. ]
 [6.6 2.9 4.6 1.3]
 [6.3 2.8 5.1 1.5]]
[2 1 0 0 0 1 1 2 1 2 0 2 0 1 0 0 0 0 0 1 2 1 0 1 0 1 0 2 1 1 1 2 2 1 2 1 1
 1]
[2 1 0 0 0 1 1 2 1 2 0 1 0 1 0 0 0 0 0 2 2 1 0 1 0 1 0 2 1 1 1 2 2 1 2 1 1
 2]
['virginica']
```

EXPERIMENT NO: 8

AIM:

Program to implement Linear Regression.

PROGRAM:

```
from google.colab import files
files.upload()
```

nahar1.csv

opening	Closing
137	137.2
133	134.05
135.95	136
132.6	134.8
122.6	133.5
125	125.15
127	128.2
127.5	129.55
126.5	129.8
127.35	128
129.4	129.4
127.5	129
126.95	126.95
126.45	129.6
127.65	127.65
124	127.7
129.7	134.75
128.5	133.3
125	129.05
124.5	131.6
117	124.8
114.95	117.6
115.1	117.55
109.45	118.2
119.45	119.45
117	118.5
128.4	128.5
129.5	129.6
125.9	129.45
131	131.1
131.4	135.35
139.9	139.9
138.05	138.9
139	140

```
   142.9     142.9
  144.45    151.95
   136.5     142.9
  142.45    142.45
     140    144.85
     139    143.95
     129     138.9
  132.45    133.95
   141.1    143.95
     152       152
   158.5     158.5
  155.05    157.95
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import datasets, linear_model
from sklearn import metrics
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv('nahar1.csv')
X = df[['opening']]
y = df[['closing']]
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.3)
print(x_train)
print(y_train)
clf = LinearRegression()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
print(y_pred)
newvalue=float(input("enter today's opening"))
y_pred = clf.intercept_ + clf.coef_ *newvalue
print(y_pred)
```

OUTPUT:

```
opening
27   129.50
10   129.40
16   129.70
13   126.45
17   128.50
33   139.00
1    133.00
19   124.50
36   136.50
29   131.00
    closing
27   129.60
10   129.40
```

```
6    128.20
44   158.50
21   117.60
26   128.50
18   129.05
2    136.00
0    137.20
12   126.95
15   127.70
29   131.10
[[128.53678975]
[153.25495148]
 [125.41150493]
 [133.74559777]
 [140.04352021]
 [120.1079913 ]
 [129.91002095]
 [144.21056663]]
enter today's opening128
[[130.52560736]]
```

EXPERIMENT NO: 9

AIM:

Program to implement Multiple Regression.

PROGRAM:

```
from google.colab import files
files.upload()
```

data.csv

Car	Model	Volume	Weight	CO2	
2	TOYOTA	AYGO	1000	790	99
3	MITSUBISHI	SPACE STAR	1200	1160	95
4	SKODA	CITIGO	900	929	95
5	FIAT	FIAT 500	1500	865	90
6	MINI	COOPER	1000	1140	105
7	VW	UP!	1400	929	105
8	SKODA	FABIA	1500	1109	90
9	MERCEDES	A-CLASS	1500	1365	92
10	FORD	FIESTA	1500	1112	98
11	AUDI	A1	1600	1150	99

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import datasets, linear_model

from sklearn import metrics
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv('data.csv')
X=df[['Weight','Volume']]
y=df[['CO2']]
X=X.values
```

```
y=y.values
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.3)
print(x_train)
print(y_train)
regr=linear_model.LinearRegression()
regr.fit(x_train,y_train)
predictedCO2=regr.predict([[2300,1300]])
print(predictedCO2)
```

OUTPUT:

```
[[1140 1500]
 [1109 1400]
 [ 929 1000]
 [1160 1200]
 [1365 1500]
 [ 865  900]
 [1112 1500]]
[[105]
 [ 90]
 [ 95]
 [ 95]
 [ 92]
 [ 90]
 [ 98]]
[[71.51564377]]
```

EXPERIMENT NO:10

AIM:

Program to implement text classification using Support Vector Machine

PROGRAM:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import svm
cancer=datasets.load_breast_cancer()
x_train,x_test,y_train,y_test=train_test_split(cancer.data,cancer.target,test_size=0.3,random_state=109)
clf=svm.SVC(kernel='linear')
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
print("Actual values",y_test)
print("Predicted values",y_pred)
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
print("Precision:",metrics.precision_score(y_test,y_pred))
print("Recall:",metrics.recall_score(y_test,y_pred))
```

OUTPUT:

```
Actual values [1 1 0 0 1 0 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 1 1 1 0 1 1
1 1 1
 0 1 1 0 1 0 1 1 1 0 1 0 0 1 1 0 1 1 0 1 1 1 0 0 1 0 1 0 0 1 1 1 1 0 1 1 1
 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 1 0 1 1
 0 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 1 1 1 0 1 1 0 0 1 0
 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1]
Predicted values [1 1 0 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 1 1 0 1 1 1 1 0 1
1 1 1 1
 0 1 1 0 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 1 1 1 0 0 1 0 1 0 0 1 1 1 1 0 1 1 1
 0 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 1 0 1 0 1 1
 0 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 1 0 1 1 1 0 1 1 0 0 1 0
 0 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1]
Accuracy: 0.9649122807017544
Precision: 0.9811320754716981
Recall: 0.9629629629629629
```

EXPERIMENT NO: 11

AIM:

Program on Decision Tree.

PROGRAM:

```
from google.colab import files
uploaded=files.upload()
```

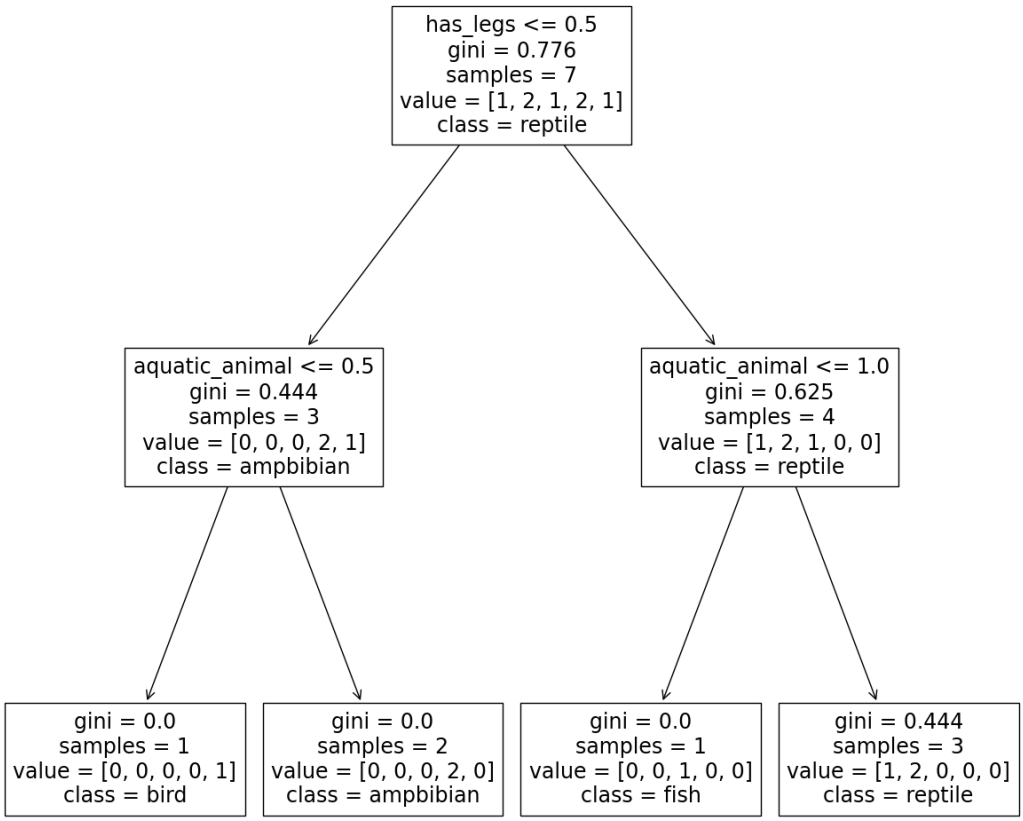
data1.csv

	A	B	C	D	E	F
1	name	gives_birth	aquatic_ar	aerial_anir	has_legs	class_label
2	human	1	0	0	1	mammal
3	python	0	0	0	0	reptile
4	salmon	0	1	0	0	fish
5	frog	0	2	0	1	ampbibian
6	bat	1	0	1	1	bird
7	pigeon	0	0	1	1	bird
8	cat	1	0	0	1	mammal
9	shark	1	1	0	0	fish
10	turtle	0	2	0	1	amphibian
11	salamande	0	2	0	1	amphibian

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn import metrics
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data1.csv')
X = df[['gives_birth','aquatic_animal','aerial_animal','has_legs']]
y = df[['class_label']]
target_names = ['mammal','reptile','fish','ampbibian','bird']
feature_names = ['gives_birth','aquatic_animal','aerial_animal','has_legs']
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.3)
print(x_train)
print(y_train)
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
print(y_pred)
print(clf.predict(x_train))
print(y_train.values.ravel())
```

OUTPUT:

```
gives_birth  aquatic_animal  aerial_animal  has_legs
1            0               0              0         0
3            0               2              0         1
5            0               0              1         1
8            0               2              0         1
2            0               1              0         0
9            0               2              0         1
7            1               1              0         0
  class_label
1     reptile
3   ampbibian
5        bird
8   amphibian
2        fish
9   amphibian
7        fish
['bird' 'bird' 'bird']
['reptile' 'amphibian' 'bird' 'amphibian' 'fish' 'amphibian' 'fish']
['reptile' 'ampbibian' 'bird' 'amphibian' 'fish' 'amphibian' 'fish']
```



EXPERIMENT NO: 12

AIM:

Program to implement k-means clustering technique.

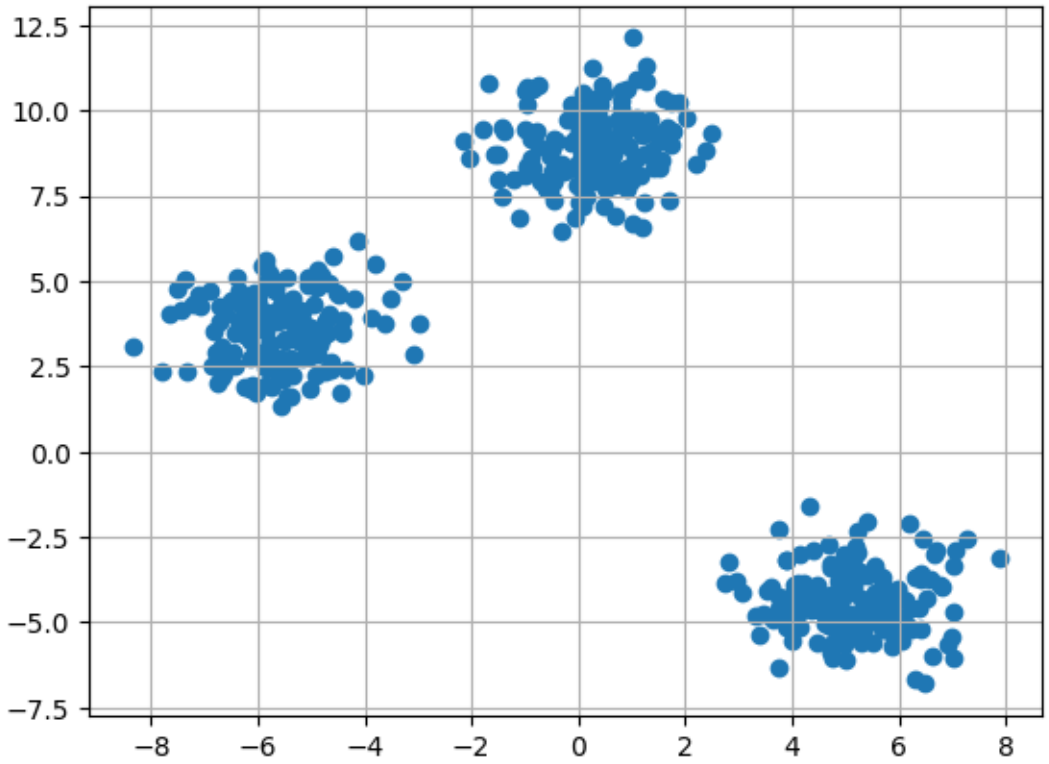
PROGRAM:

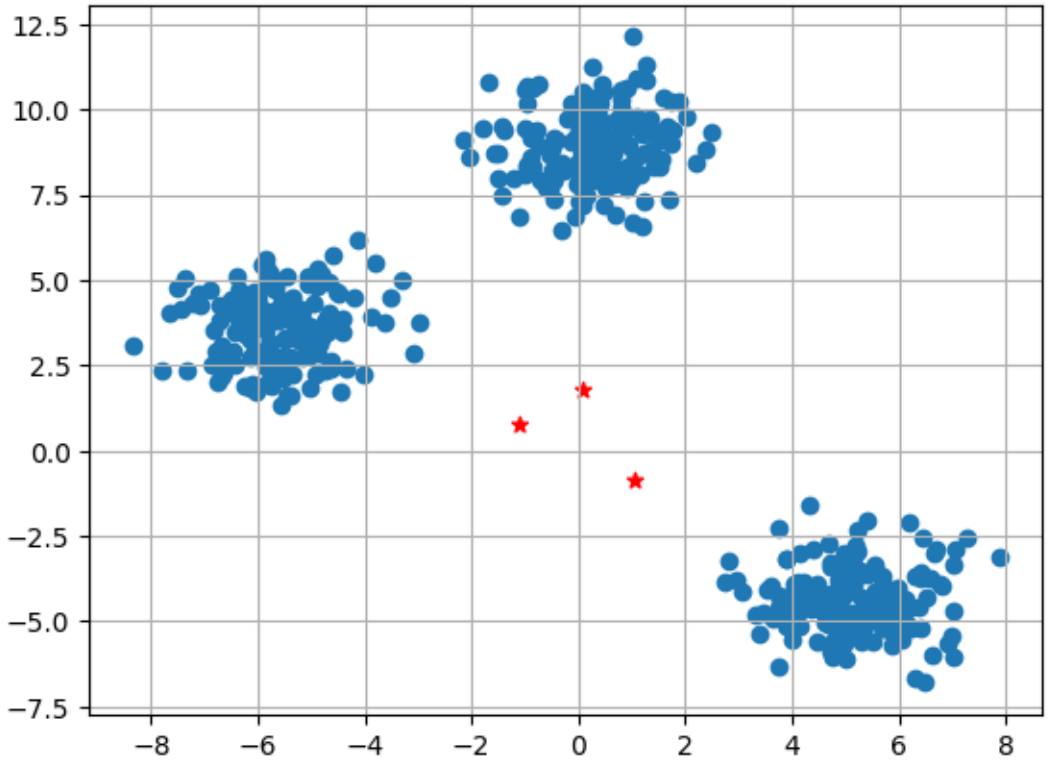
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
X,y = make_blobs(n_samples = 500,n_features = 2,centers =
3,random_state = 23)
fig = plt.figure(0)
plt.grid(True)
plt.scatter(X[:,0],X[:,1])
plt.show()
k = 3
clusters = {}
np.random.seed(23)
for idx in range(k):
    center = 2*(2*np.random.random((X.shape[1],))-1)
    points = []
    cluster = {
        'center' : center,
        'points' : []
    }

    clusters[idx] = cluster

clusters
plt.scatter(X[:,0],X[:,1])
plt.grid(True)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '*',c = 'red')
plt.show()
```

OUTPUT:


12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−8
−6
−4
−2
0
2
4
6
8


12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−8
−6
−4
−2
0
2
4
6
8

EXPERIMENT NO: 13

AIM:

Implement a program for natural language processing steps:

(a) Ngram

(b) Tokenization

(c) Parts of speech tagging

(d) Stemming/ lemmatization

PROGRAM:

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk import ngrams
sentence= input("Enter the sentence")
n=int(input("Enter the value of n:"))
n_grams=ngrams(sentence.split(),n)
print("ngrams.printing")
for grams in n_grams:
  print(grams)

from nltk import word_tokenize,sent_tokenize
print("tokens printing")
print(word_tokenize(sentence))
print(sent_tokenize(sentence))

from nltk import pos_tag
tokenized_text=word_tokenize(sentence)
tags=tokens_tag=pos_tag(tokenized_text)
print(tags)
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
ps= PorterStemmer()
for w in tokenized_text:
  print(w,":",ps.stem(w))
```

OUTPUT:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
Enter the sentencelife is so beautiful
Enter the value of n:3
ngrams.printing
('life', 'is', 'so')
('is', 'so', 'beautiful')
tokens printing
['life', 'is', 'so', 'beautiful']
['life is so beautiful']
[('life', 'NN'), ('is', 'VBZ'), ('so', 'RB'), ('beautiful', 'JJ')]
life : life
is : is
so : so
beautiful : beauti
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

EXPERIMENT NO: 14

AIM:

Program on convolutional neural network to classify images from any standard dataset in the public domain using Keras framework.

PROGRAM:

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D, Flatten
from tensorflow.keras import utils
# to calculate accuracy
from sklearn.metrics import accuracy_score
# loading the dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# building the input vector from the 28x28 pixels
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# normalizing the data to help with the training
X_train /= 255
X_test /= 255
# one-hot encoding using keras' numpy-related utilities
n_classes = 10
print("Shape before one-hot encoding: ", y_train.shape)
Y_train = utils.to_categorical(y_train, n_classes)
Y_test = utils.to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)
# building a linear stack of layers with the sequential model
model = Sequential()
# convolutional layer
model.add(Conv2D(25, kernel_size=(3,3), strides=(1,1), padding='valid', activation='relu',
input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(1,1)))
# flatten output of conv
model.add(Flatten())
# hidden layer
model.add(Dense(100, activation='relu'))
# output layer
model.add(Dense(10, activation='softmax'))
# compiling the sequential model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
# training the model for 10 epochs
model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_test, Y_test))
```

OUTPUT:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Shape before one-hot encoding:  (60000,)
Shape after one-hot encoding:  (60000, 10)
Epoch 1/10
469/469 [==============================] - 49s 100ms/step - loss: 0.2007 - accuracy: 0.9421 - val_loss: 0.0756 -
val_accuracy: 0.9771
Epoch 2/10
469/469 [==============================] - 41s 88ms/step - loss: 0.0583 - accuracy: 0.9829 - val_loss: 0.0560 -
val_accuracy: 0.9823
Epoch 3/10
469/469 [==============================] - 42s 89ms/step - loss: 0.0351 - accuracy: 0.9898 - val_loss: 0.0581 -
val_accuracy: 0.9794
Epoch 4/10
469/469 [==============================] - 41s 87ms/step - loss: 0.0214 - accuracy: 0.9934 - val_loss: 0.0550 -
val_accuracy: 0.9831
Epoch 5/10
469/469 [==============================] - 40s 84ms/step - loss: 0.0133 - accuracy: 0.9963 - val_loss: 0.0567 -
val_accuracy: 0.9822
Epoch 6/10
469/469 [==============================] - 44s 95ms/step - loss: 0.0094 - accuracy: 0.9974 - val_loss: 0.0570 -
val_accuracy: 0.9838
Epoch 7/10
469/469 [==============================] - 43s 92ms/step - loss: 0.0075 - accuracy: 0.9977 - val_loss: 0.0586 -
val_accuracy: 0.9843
Epoch 8/10
469/469 [==============================] - 42s 90ms/step - loss: 0.0050 - accuracy: 0.9986 - val_loss: 0.0633 -
val_accuracy: 0.9834
Epoch 9/10
469/469 [==============================] - 41s 88ms/step - loss: 0.0044 - accuracy: 0.9987 - val_loss: 0.0663 -
val_accuracy: 0.9824
Epoch 10/10
469/469 [==============================] - 41s 87ms/step - loss: 0.0042 - accuracy: 0.9988 - val_loss: 0.0707 -
val_accuracy: 0.9827
<keras.src.callbacks.History at 0x7dc8f538b6d0>
```

EXPERIMENT NO: 15

AIM:

Program to implement a simple web crawler and scrapping web pages.

PROGRAM:

```
import requests
from bs4 import BeautifulSoup

URL = "https://realpython.github.io/fake-jobs/"
page = requests.get(URL)
print(page.text)
soup = BeautifulSoup(page.content, "html.parser")
results = soup.find(id="ResultsContainer")
job_elements = results.find_all("div", class_="card-content")
for job_element in job_elements:
   title_element = job_element.find("h2", class_="title")
   company_element = job_element.find("h3", class_="company")
   location_element = job_element.find("p", class_="location")
   print(title_element.text.strip())
   print(company_element.text.strip())
   print(location_element.text.strip())
   print()
```

OUTPUT:

```
  ..... <div class="content">
      <p class="location">
        Jamesville, AA
      </p>
      <p class="is-small has-text-grey">
        <time datetime="2021-04-08">2021-04-08</time>
      </p>
    </div>
    <footer class="card-footer">
        <a href="https://www.realpython.com" target="_blank" class="card-footer-
item">Learn</a>
        <a href="https://realpython.github.io/fake-jobs/jobs/dispensing-optician-67.html" target="_blank"
class="card-footer-item">Apply</a>
    </footer>
  </div>
</div>
</div>
<div class="column is-half">
<div class="card">
  <div class="card-content">
    <div class="media">
      <div class="media-left">
```

```
        <figure class="image is-48x48">
          <img src="https://files.realpython.com/media/real-python-logo-
thumbnail.7f0db70c2ed2.jpg?__no_cf_polish=1" alt="Real Python Logo">
        </figure>
      </div>
      <div class="media-content">
        <h2 class="title is-5">Designer, fashion/clothing</h2>
        <h3 class="subtitle is-6 company">Vasquez Ltd</h3>
      </div>
    </div> …….


…….. Historic buildings inspector/conservation officer

Smith LLC
North Brandonville, AP

Data scientist
Thomas Group
Port Robertfurt, AA

Psychiatrist
Silva-King
Burnettbury, AE

Structural engineer
Pierce-Long
Herbertside, AA

Immigration officer
Walker-Simpson
Christopherport, AP

Python Programmer (Entry-Level)
Cooper and Sons
West Victor, AE

Neurosurgeon
Donovan, Gonzalez and Figueroa
Port Aaron, AP

Broadcast engineer
Morgan, Butler and Bennett
Loribury, AA

Make
Snyder-Lee
Angelastad, AP

Nurse, adult
Harris PLC
Larrytown, AE
…….
```