	Exp.No. **Date:**

Aim:

Implement decision tree using python.

Program:

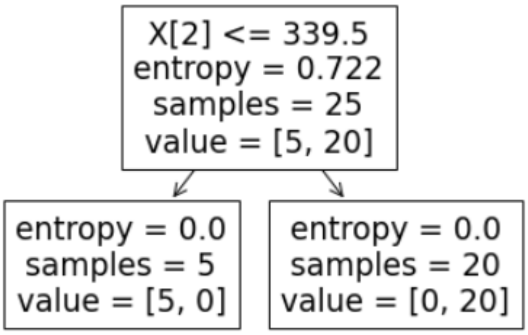
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
from sklearn import tree

# loading data file
balance_data=pd.read_csv('CREDITDATA.csv',sep=',',header=0)
print("dataset length::",len(balance_data))
print("dataset shape::",balance_data.shape)
print("dataset:")
balance_data.head()

#separating the target value
X=balance_data.values[:,1:5]
Y=balance_data.values[:,0]

#splitting dataset into Test and Train
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=100)

#Function to perform training with entrophy
clf_entropy=DecisionTreeClassifier(criterion="entropy",random_state=100,
max_depth=5,min_samples_leaf=5)
clf_entropy.fit(X_train,Y_train)

#Function to make predictions
Y_pred_en=clf_entropy.predict(X_test)
print(Y_pred_en)

#checking accuracy
print("Accuracy is", accuracy_score(Y_test,Y_pred_en)*100)

tree.plot_tree(clf_entropy)
```

Input:

Result	IP	LP	Creditscore	Housenumber
YES	230	234	555	100
YES	223	345	555	123
YES	11	234	666	1223
NO	33	222	111	4566
YES	22	345	555	3455
NO	554	677	112	234
YES	345	456	777	554
NO	344	455	123	4545
YES	444	477	555	654
YES	222	233	666	543
YES	555	777	567	567
YES	333	345	678	345
YES	666	888	567	367
YES	444	555	666	277
YES	344	675	567	388
YES	243	344	567	399
YES	566	677	776	455
YES	345	566	566	288
NO	222	455	122	299
NO	123	345	121	377
NO	134	234	111	177
NO	122	145	124	166
YES	222	456	678	199
YES	666	888	567	377
YES	333	555	788	200
YES	222	444	678	100
YES	566	677	675	377
YES	344	566	567	177
YES	344	455	578	233
YES	345	555	788	122
NO	234	455	123	544
NO	222	444	121	533
NO	122	222	111	522
NO	122	234	111	511
YES	455	567	678	194
YES	344	456	789	193

Output:

```
dataset length:: 36
dataset shape:: (36, 5)
dataset:
['YES' 'NO' 'NO' 'YES' 'NO' 'YES' 'YES' 'NO' 'NO' 'YES' 'NO']
Accuracy is 100.0

[Text(167.4, 163.07999999999998, 'X[2] <= 339.5\nentropy = 0.722\nsamples = 25\nvalue = [5, 20]'),
 Text(83.7, 54.360000000000014, 'entropy = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(251.10000000000002, 54.360000000000014, 'entropy = 0.0\nsamples = 20\nvalue = [0, 20]')]
```



Aim:

Implement Find-s algorithm using python

Program:

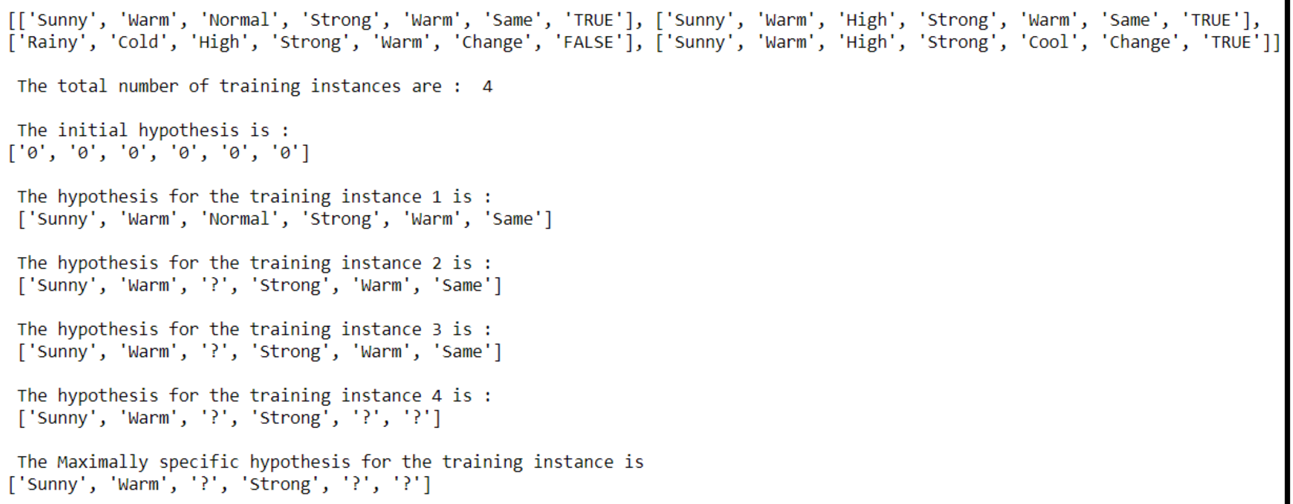
```
import csv
a = []
with open('enjoysport.csv', 'r') as csvfile:
   for row in csv.reader(csvfile):
      a.append(row)
   print(a)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for i in range(0, len(a)):
   if a[i][num_attribute] == 'TRUE':
       for j in range(0, num_attribute):
            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
            else:
                hypothesis[j] = '?'
   print("\n The hypothesis for the training instance {} is : \n".format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

Input:

Sunny	Warm	Normal	Strong	Warm	Same	TRUE
Sunny	Warm	High	Strong	Warm	Same	TRUE
Rainy	Cold	High	Strong	Warm	Change	FALSE
Sunny	Warm	High	Strong	Cool	Change	TRUE

Exp.No.
Date:

Output:



```
[['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'TRUE'], ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'TRUE'],
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'FALSE'], ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'TRUE']]

 The total number of training instances are :  4

 The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 1 is :
 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

 The hypothesis for the training instance 2 is :
 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

 The hypothesis for the training instance 3 is :
 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

 The hypothesis for the training instance 4 is :
 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 The Maximally specific hypothesis for the training instance is
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

	Exp.No. **Date:**

Aim:

Implement Candidate Elimination Algorithm using python.

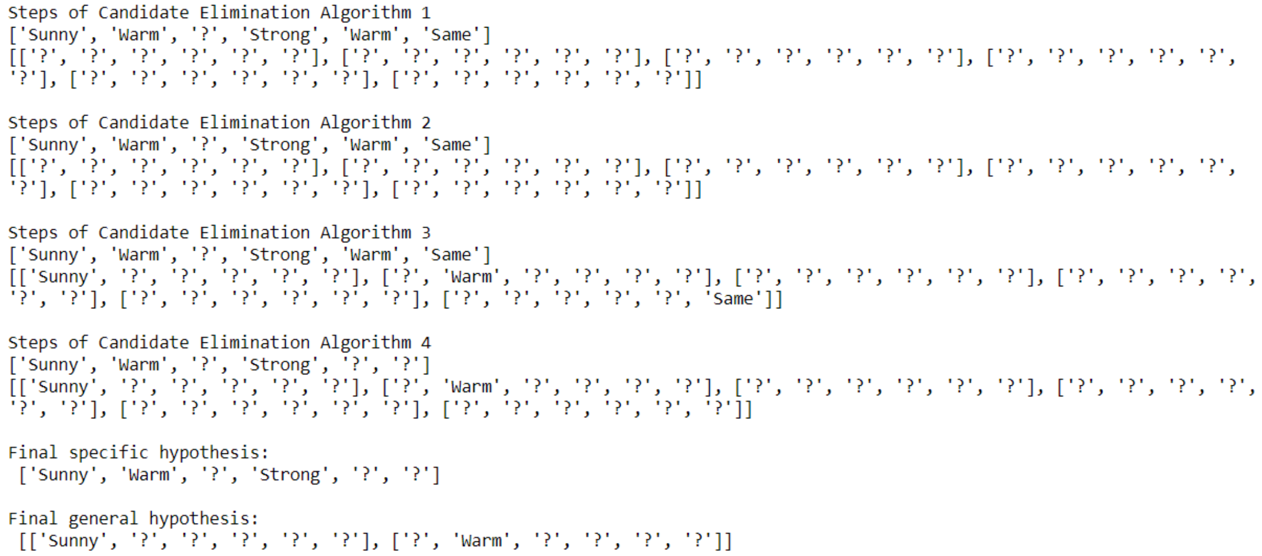
Program:

```
import csv
with open("enjoysport.csv") as f:
    csv_file=csv.reader(f)
    data=list(csv_file)

    s=data[1][:-1]
    g=[['?' for i in range(len(s))] for j in range(len(s))]

    for i in data:
        if i[-1]=="TRUE":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    s[j]='?'
                    g[j][j]='?'

        elif i[-1]=="FALSE":
            for j in range(len(s)):
                if i[j]!=s[j]:
                    g[j][j]=s[j]
                else:
                    g[j][j]="?"
        print("\nSteps of Candidate Elimination Algorithm",data.index(i)+1)
        print(s)
        print(g)
    gh=[]
    for i in g:
        for j in i:
            if j!='?':
                gh.append(i)
                break
    print("\nFinal specific hypothesis:\n",s)
    print("\nFinal general hypothesis:\n",gh)
```

Input:

Sunny	Warm	Normal	Strong	Warm	Same	TRUE
Sunny	Warm	High	Strong	Warm	Same	TRUE
Rainy	Cold	High	Strong	Warm	Change	FALSE
Sunny	Warm	High	Strong	Cool	Change	TRUE

Output:



```
Steps of Candidate Elimination Algorithm 1
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 3
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Steps of Candidate Elimination Algorithm 4
['Sunny', 'Warm', '?', 'Strong', '?', '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final specific hypothesis:
 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final general hypothesis:
 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

	Exp.No. Date:

Aim:

Implement Linear regression algorithm using python.

Program:

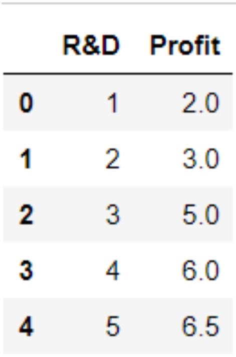
```
#Importing of libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import r2_score
import statsmodels.api as sm

#Reading data from file
data=pd.read_csv("profitdata.csv")
data.head()

#Plotting the graph and displaying
plt.scatter(data['R&D'],data['Profit'],c='black')
plt.show()

#Applying linear regression
X=data['R&D'].values.reshape(-1,1)
Y=data['Profit'].values.reshape(-1,1)
reg=linear_model.LinearRegression()
reg.fit(X,Y)
reg.intercept_
reg.coef_

#Best fit line
predictions=reg.predict(X)
plt.scatter(data['R&D'],data['Profit'],color='black')
plt.plot(data['R&D'],predictions,c='blue',linewidth=2)
plt.xlabel("R&D")
plt.ylabel("Profit")
plt.show()

X=data['R&D']
Y=data['Profit']
X2=sm.add_constant(X)
```

```
est=sm.OLS(Y,X2) # ordinary least squares
est2=est.fit()
print(est2.summary())
```

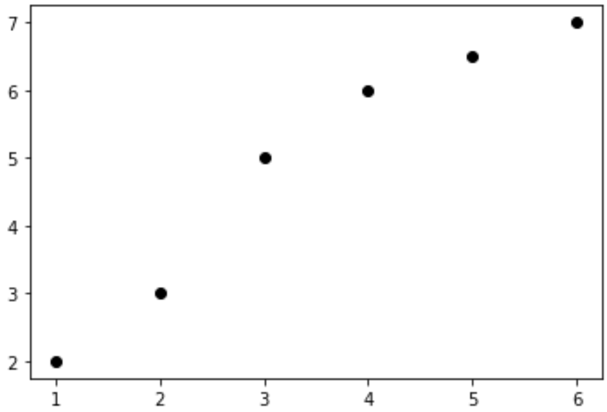
Input:

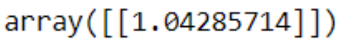
R&D	Profit
1	2
2	3
3	5
4	6
5	6.5
6	7

Output:

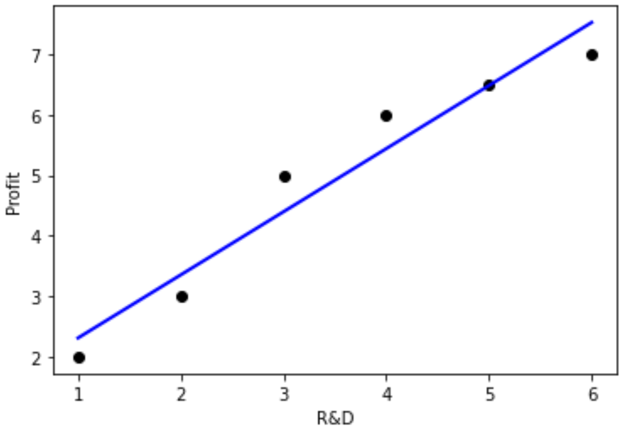


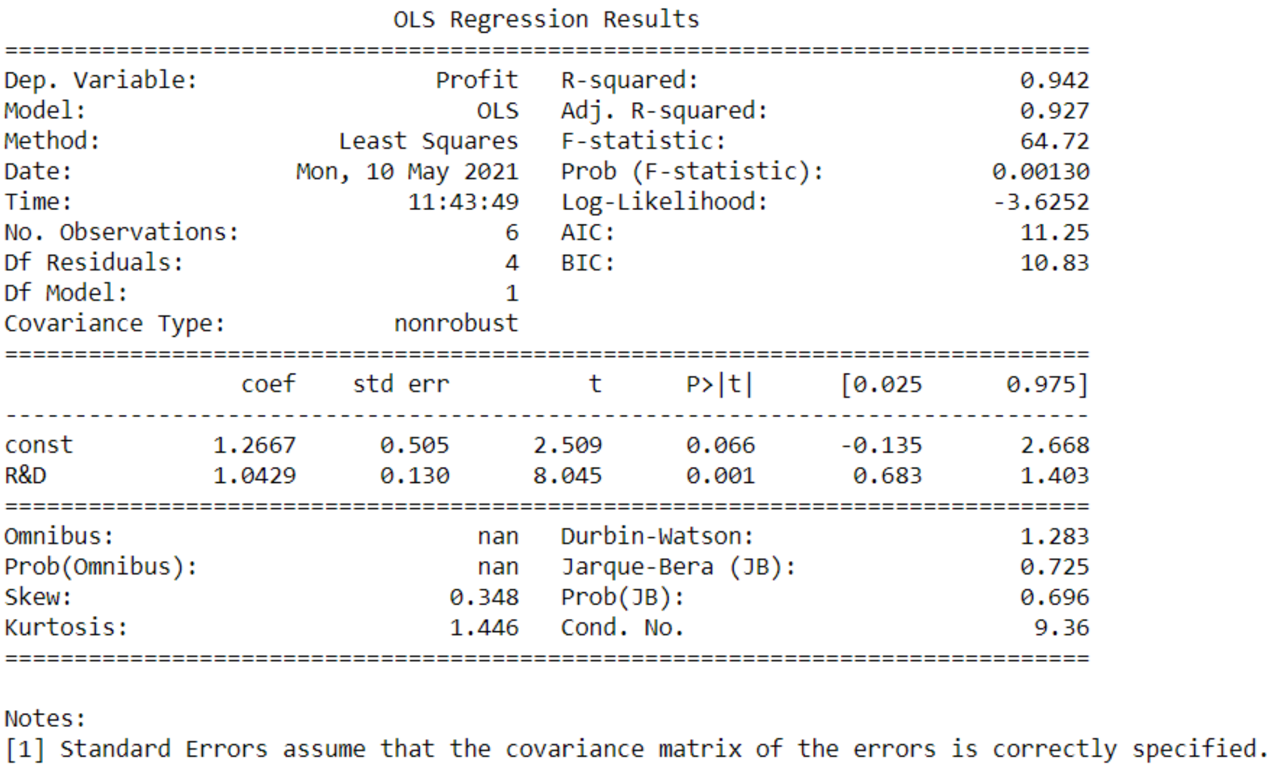
	R&D	Profit
0	1	2.0
1	2	3.0
2	3	5.0
3	4	6.0
4	5	6.5





```
array([[1.04285714]])
```





```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Profit   R-squared:                       0.942
Model:                            OLS   Adj. R-squared:                  0.927
Method:                 Least Squares   F-statistic:                     64.72
Date:                Mon, 10 May 2021   Prob (F-statistic):            0.00130
Time:                        11:43:49   Log-Likelihood:                -3.6252
No. Observations:                   6   AIC:                             11.25
Df Residuals:                       4   BIC:                             10.83
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.2667      0.505      2.509      0.066      -0.135       2.668
R&D            1.0429      0.130      8.045      0.001       0.683       1.403
==============================================================================
Omnibus:                          nan   Durbin-Watson:                   1.283
Prob(Omnibus):                    nan   Jarque-Bera (JB):                0.725
Skew:                           0.348   Prob(JB):                        0.696
Kurtosis:                       1.446   Cond. No.                         9.36
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Aim:

Implement Multi linear regression algorithm using python.

Program:

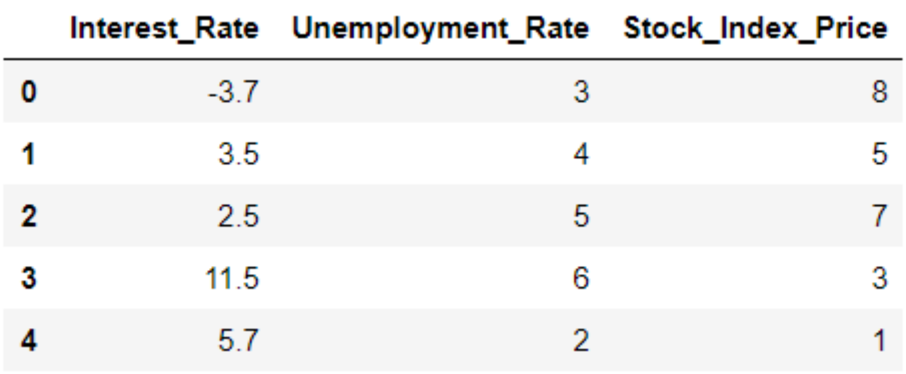
```
import pandas as pd
from sklearn import linear_model
import statsmodels.api as sm
Stock_Market = {'Interest_Rate': [-3.7,3.5,2.5,11.5,5.7],
            'Unemployment_Rate': [3,4,5,6,2],
            'Stock_Index_Price': [8,5,7,3,1]
            }
df =
pd.DataFrame(Stock_Market,columns=['Interest_Rate','Unemployment_Rate','Stock_Index_Price
'])
df.head()

X = df[['Stock_Index_Price','Unemployment_Rate']]
# here we have 2 variables for multiple regression. If you just want to use one variable for simple
linear regression, then use X = df['Interest_Rate'] for example.Alternatively, you may add
additional variables within the brackets

Y = df['Interest_Rate']

# with sklearn
regr = linear_model.LinearRegression()
regr.fit(X, Y)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)
```

Input:

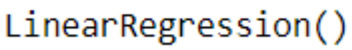
Interest_Rate: [-3.7,3.5,2.5,11.5,5.7],
Unemployment_Rate: [3,4,5,6,2],
Stock_Index_Price: [8,5,7,3,1]

	Exp.No. Date:

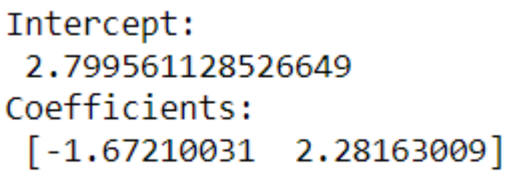
Output:



	Interest_Rate	Unemployment_Rate	Stock_Index_Price
0	-3.7	3	8
1	3.5	4	5
2	2.5	5	7
3	11.5	6	3
4	5.7	2	1



```
LinearRegression()
```



```
Intercept:
 2.799561128526649
Coefficients:
 [-1.67210031  2.28163009]
```

Aim:

Implement Naive bayes using python.

Program:

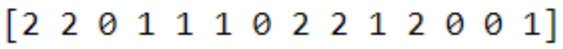
```
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

#Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()

# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)

# Converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
print("Temp:",temp_encoded)
print("Play:",label)

#Combinig weather and temp into single listof tuples
#features=tuple(zip(weather,temp))
features=[(weather_encoded[i],temp_encoded[i]) for i in range(0,len(weather_encoded))]
print(features)

#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB
#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(features,label)
#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print("Predicted Value:", predicted)
```

Exp.No.
Date:

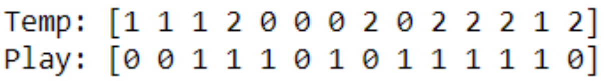
Input:

weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny', 'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

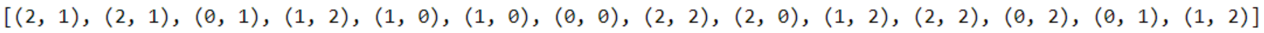
Output:



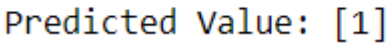
```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1]
```



```
Temp: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```



```
[(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0), (2, 2), (2, 0), (1, 2), (2, 2), (0, 2), (0, 1), (1, 2)]
```



```
Predicted Value: [1]
```

	Exp.No. **Date:**

Aim:

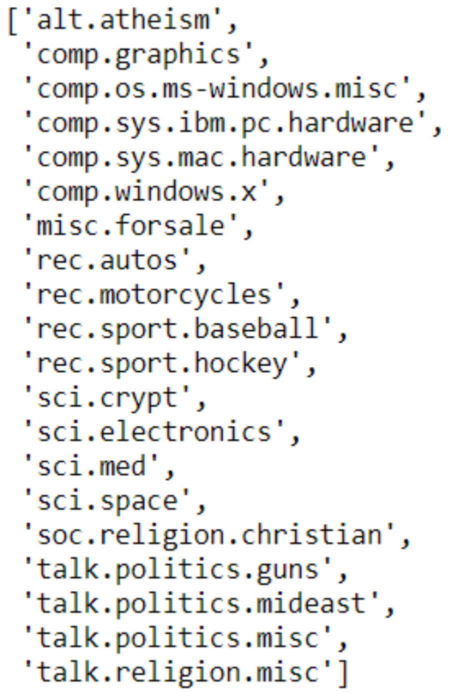
Implement Naive bayes classification with English text using python.

Program:

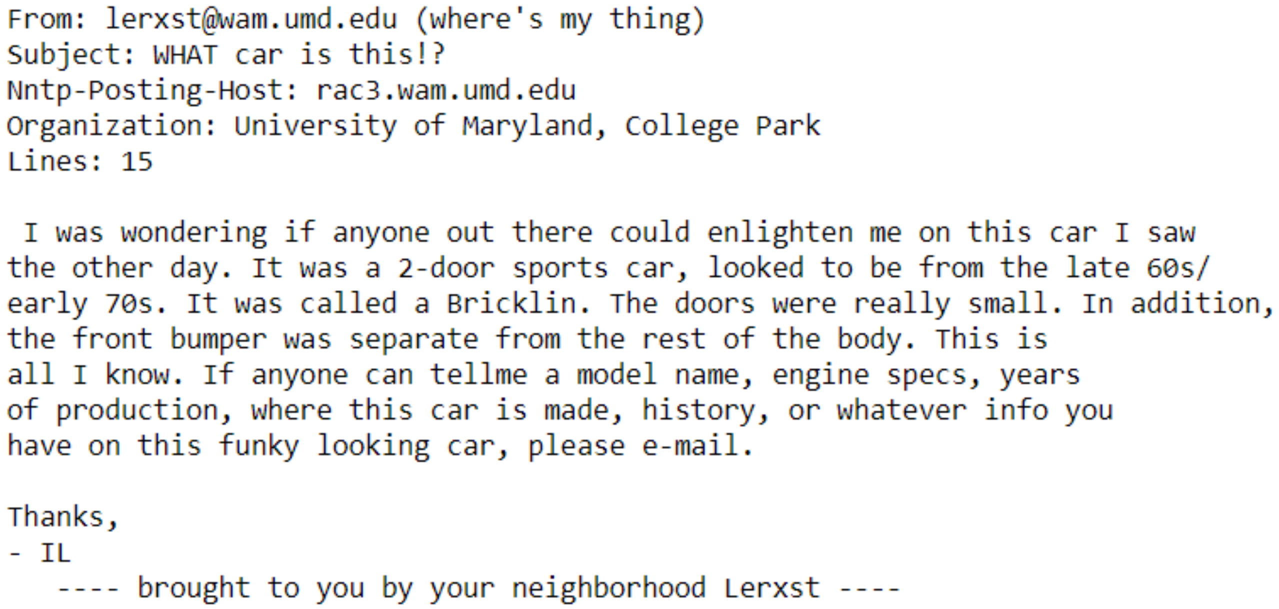
```
# %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.datasets import fetch_20newsgroups
data=fetch_20newsgroups()
data.target_names

categories=['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball',
'rec.sport.hockey',
 'sci.crypt', 'sci.electronics', 'sci.med',  'sci.space', 'soc.religion.christian', 'talk.politics.guns',
'talk.politics.mideast',
 'talk.politics.misc', 'talk.religion.misc']
train= fetch_20newsgroups(subset='train',categories=categories)
test= fetch_20newsgroups(subset='test',categories=categories)
print(train.data[0])

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline
model=make_pipeline(TfidfVectorizer(),MultinomialNB())
model.fit(train.data,train.target)
labels=model.predict(test.data)

#creation of confusion matrix and heat map
from sklearn.metrics import confusion_matrix
mat=confusion_matrix(test.target,labels)
sns.heatmap(mat.T,square=True,annot=True,fmt='d',cbar=False,xticklabels=train.target_names,y
ticklabels=train.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label')
```

```
def predict_category(s,train=train,model=model):
  pred=model.predict([s])
  return train.target_names[pred[0]]
predict_category('sending rocket to international space station')
```

Output:

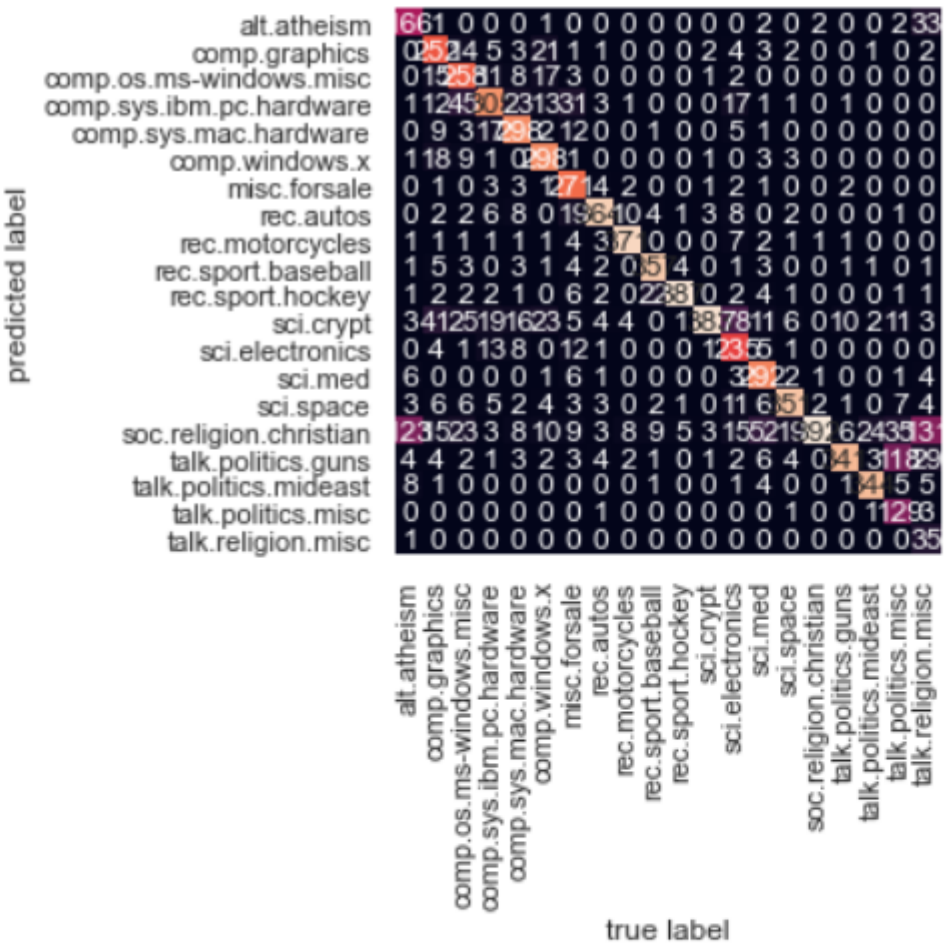


```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```



```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
```

```
Text(89.133125, 0.5, 'predicted label')
```



```
'sci.space'
```

Exp.No.
Date:

Aim:

Implement K-Means algorithm using python.

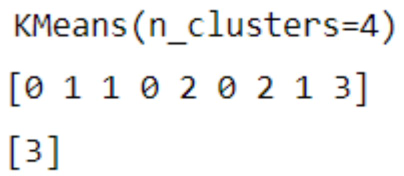
Program:

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

# read csv input file
input_data = pd.read_csv('kmeans.csv')

# initialize KMeans object specifying the number of desired clusters
kmeans = KMeans(n_clusters=4)

# learning the clustering from the input date
kmeans.fit(input_data.values)

# output the labels for the input data
print(kmeans.labels_)

# predict the classification for given data sample
predicted_class = kmeans.predict([[0.0906,0.606,1]])
print(predicted_class)
```

Input:

Var1	Var2	Class
1.713	1.586	0
0.18	1.786	1
0.353	1.24	1
0.94	1.566	0
1.486	0.759	1
1.266	1.106	0
1.54	0.419	1
0.459	1.799	1
0.773	0.186	1

Output:



```
KMeans(n_clusters=4)
[0 1 1 0 2 0 2 1 3]
[3]
```

Aim:

Implement KNN algorithm using python.

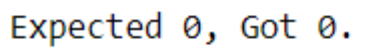
Program:

```
from math import sqrt

# calculate the Euclidean distance between two vectors
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

# Make a classification prediction with 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]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

# Test distance function
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], 7)

print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

Input:

```
[[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]]
```

Output:



```
Expected 0, Got 0.
```

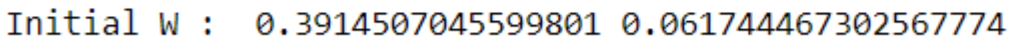
Aim:

Implement Backpropagation using python.

Program:

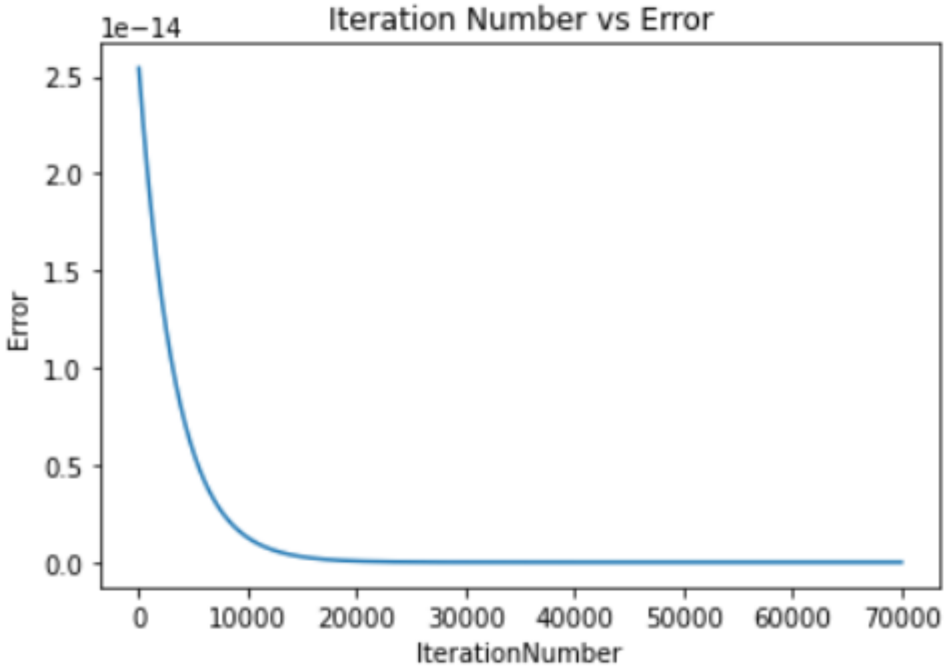
```
import numpy
import matplotlib.pyplot as plt

def sigmoid(sop):
     return 1.0/(1+numpy.exp(-1*sop))

def error(predicted, target):
     return numpy.power(predicted-target, 2)

def error_predicted_deriv(predicted, target):
    return 2*(predicted-target)

def sigmoid_sop_deriv(sop):
    return sigmoid(sop)*(1.0-sigmoid(sop))

def sop_w_deriv(x):
     return x

def update_w(w, grad, learning_rate):
    return w - learning_rate*grad

x1=0.1
x2=0.4
target = 0.7
learning_rate=0.01
w1=numpy.random.rand()
w2=numpy.random.rand()
print("Initial W : ", w1, w2)

predicted_output = []
network_error = []
old_err = 0
for k in range(70000):
       # Forward Pass
       y = w1*x1 + w2*x2
       predicted = sigmoid(y)
       err = error(predicted, target)
       predicted_output.append(predicted)
       network_error.append(err)
       # Backward Pass
       g1 = error_predicted_deriv(predicted, target)
       g2 = sigmoid_sop_deriv(y)
       g3w1 = sop_w_deriv(x1)
       g3w2 = sop_w_deriv(x2)
       gradw1 = g3w1*g2*g1
       gradw2 = g3w2*g2*g1
       w1 = update_w(w1, gradw1, learning_rate)
       w2 = update_w(w2, gradw2, learning_rate)
       #print(predicted)

plt.figure()
plt.plot(network_error)
plt.title("Iteration Number vs Error")
plt.xlabel("IterationNumber")
plt.ylabel("Error")
plt.show()

plt.figure()
plt.plot(predicted_output)
plt.title("Iteration Number vs Prediction")
plt.xlabel("Iteration Number")
plt.ylabel("Prediction")
plt.show()
```

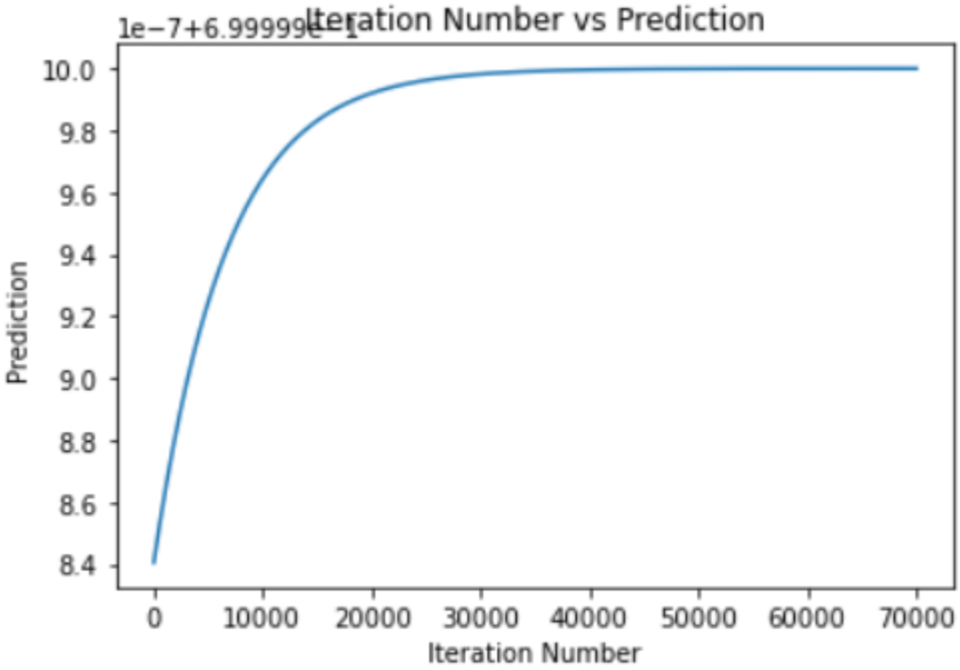
Output:



```
Initial W :  0.3914507045599801 0.061744467302567774
```

Exp.No.
Date:


Iteration Number vs Error
1e−14
2.5
2.0
1.5
1.0
0.5
0.0
Error
0
10000
20000
30000
40000
50000
60000
70000
IterationNumber


Iteration Number vs Prediction
1e−7+6.99999e−1
10.0
9.8
9.6
9.4
9.2
9.0
8.8
8.6
8.4
Prediction
0
10000
20000
30000
40000
50000
60000
70000
Iteration Number

Aim:

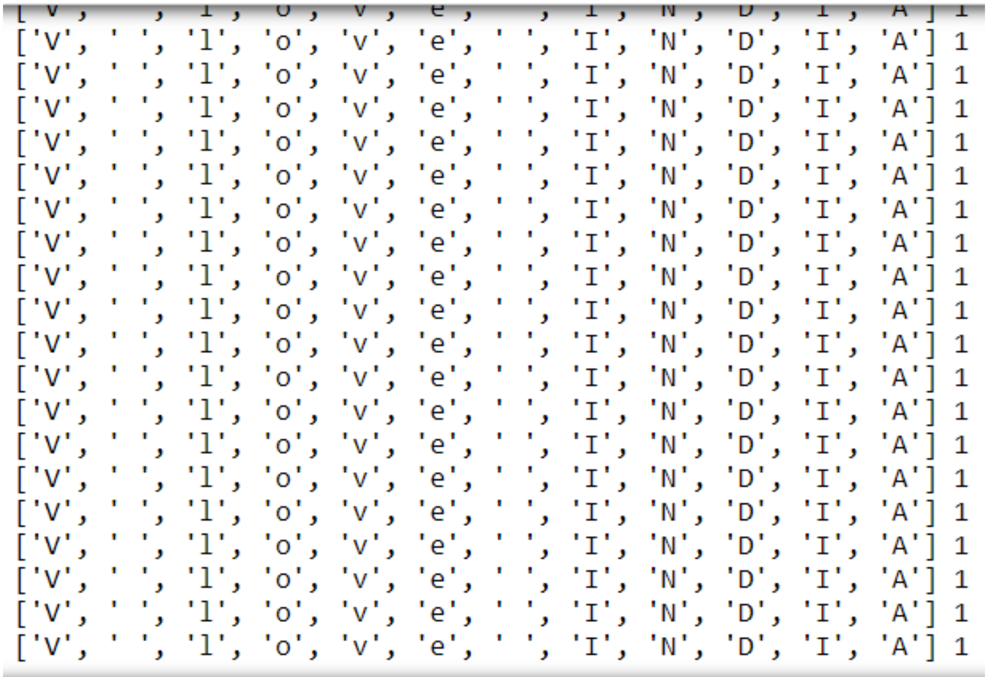
Implement Genetic algorithm using python.

Program:

```
import random
# Number of individuals in each generation
POPULATION_SIZE = 50
# Valid genes
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ 1234567890, .-
;:_!"#%&/()=?@${[]}'''
# Target string to be generated
TARGET = "I love INDIA"
class Individual(object):
    ''' Class representing individual in population '''

    def __init__(self, chromosome):
        self.chromosome = chromosome
        self.fitness = self.cal_fitness()

    @classmethod
    def mutated_genes(self):
        ''' create random genes for mutation '''
        global GENES
        gene = random.choice(GENES)
        return gene

    @classmethod
    def create_gnome(self):
        ''' create chromosome or string of genes '''
        global TARGET
        gnome_len = len(TARGET)
        return [self.mutated_genes() for _ in range(gnome_len)]

    def mate(self, par2):
        ''' Perform mating and produce new offspring '''
       # chromosome for offspring
        child_chromosome = []
        for gp1, gp2 in zip(self.chromosome, par2.chromosome):
        # random probability
        prob = random.random()
    # if prob is less than 0.45, insert gene from parent 1
        if prob < 0.45:
          child_chromosome.append(gp1)
        elif prob < 0.90:
          child_chromosome.append(gp2)
    # otherwise insert random gene(mutate), for maintaining diversity
        else:
         child_chromosome.append(self.mutated_genes())
    # create new Individual(offspring) using  generated chromosome for offspring
    return Individual(child_chromosome)

  def cal_fitness(self):
    global TARGET
    fitness = 0
    for gs, gt in zip(self.chromosome, TARGET):
      if gs != gt:
        fitness+= 1
    return fitness
 # Driver code
def main():
  global POPULATION_SIZE
  #current generation
  generation = 1
  found = False
  population = []
  # create initial population
  for _ in range(POPULATION_SIZE):
    gnome = Individual.create_gnome()
    population.append(Individual(gnome))
  while not found:
    # sort the population in increasing order of fitness score
    population = sorted(population, key = lambda x:x.fitness)
    # if the individual having lowest fitness score ie. 0 then we know that we have reached to
the targetand break the loop
    if population[0].fitness <= 0:
      found = True
      break
     # Otherwise generate new offsprings for new generation
```

```
        new_generation = []
        # Perform Elitism, that mean 10% of fittest population goes to the next generation
        s = int((10*POPULATION_SIZE)/100)
        new_generation.extend(population[:s])
        # From 50% of fittest population, Individuals  will mate to produce offspring
        s = int((90*POPULATION_SIZE)/100)
        for _ in range(s):
            parent1 = random.choice(population[:50])
            parent2 = random.choice(population[:50])
            child = parent1.mate(parent2)
            new_generation.append(child)
        population = new_generation
        print(population[0].chromosome,population[0].fitness)
       # print("Generation: {}\tString: {}\tFitness: {}".\
format(generation,"".join(population[0].chromosome), population[0].fitness))
        generation += 1
        print(population[0].chromosome,population[0].fitness)
       # print("Generation: {}\tString: {}\tFitness: {}".\
format(generation,"".join(population[0].chromosome), population[0].fitness))
if __name__ == '__main__':
    main()
```

Output:



```
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
['V', ' ', 'l', 'o', 'v', 'e', ' ', 'I', 'N', 'D', 'I', 'A'] 1
```

Exp.No.
Date:

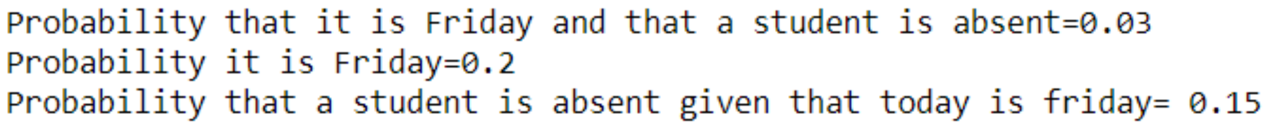
Aim:

The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is theprobability that a student is absent given that today is Friday? Apply Baye’s rule in python to get the result.

Program:

```
x=float(input("Probability that it is Friday and that a student is absent="))
y=float(input("Probability it is Friday="))
z=x/y
print("Probability that a student is absent given that today is friday=",z)
```

Output:



```
Probability that it is Friday and that a student is absent=0.03
Probability it is Friday=0.2
Probability that a student is absent given that today is friday= 0.15
```

Exp.No.
Date: