VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING (20CS6PCMAL)

Submitted by

Ruben Mathew (1BM19CS207)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2022 to July-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**MACHINE LEARNING**" carried out by **Ruben Mathew(1BM19CS207),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning- (20CS6PCMAL)** work prescribed for the said degree.

Name of the Lab-Incharge	**Prof.Saritha.A.N**
Designation	Assistant Professor
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

LAB PROGRAM 1:

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

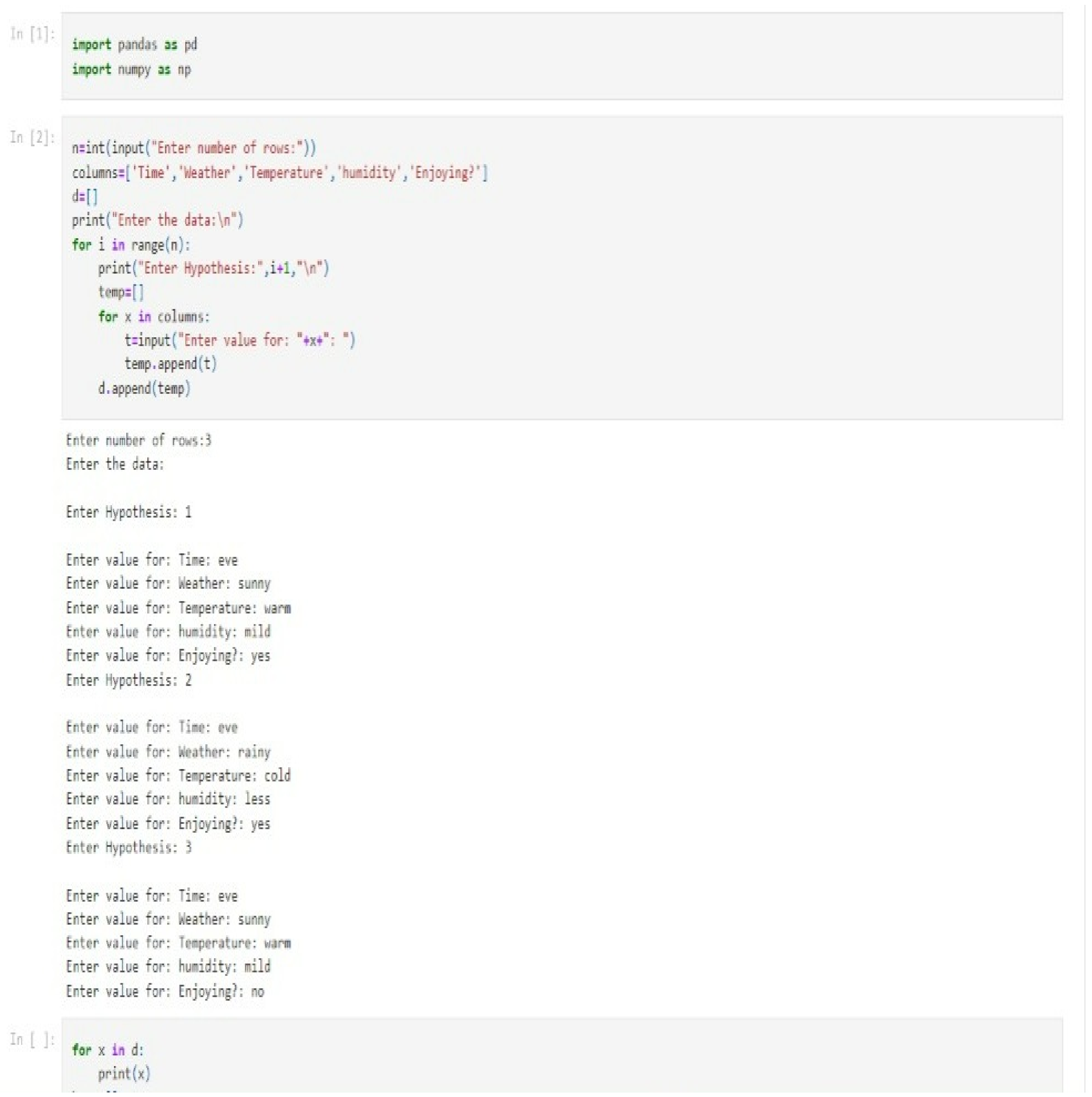
Code:

i.

```
import
csv


        def updateHypothesis(x,h):
            if h==[]:
                return x


            for i in range(0,len(h)):
                if x[i].upper()!=h[i].upper():
                    h[i] = '?'


            return h


        if __name__ == "__main__":
            data = []
            h = []


            # reading csv file
            with open('data.csv', 'r') as file:
                reader = csv.reader(file)
                print("Data: ")
                for row in reader:
                    data.append(row)
                    print(row)


            if data:
                for x in data:
                    if x[-1].upper()=="YES":
                        x.pop() # removing last field
                        h = updateHypothesis(x,h)
```

```
print("\nHypothesis: ",h)
```

ii.

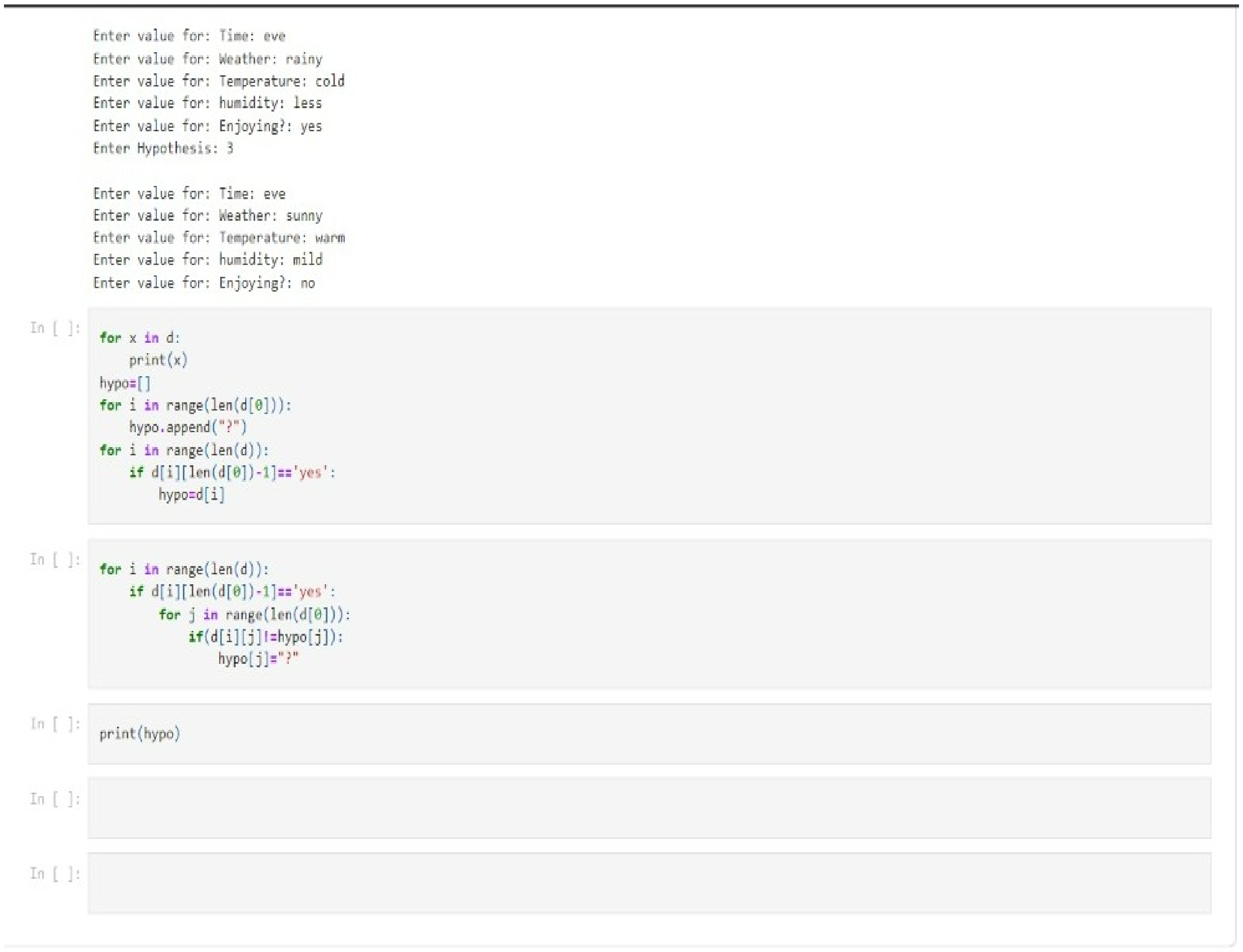


```
In [1]:
import pandas as pd
import numpy as np
```

```
In [2]:
n=int(input("Enter number of rows:"))
columns=['Time','Weather','Temperature','humidity','Enjoying?']
d=[]
print("Enter the data:\n")
for i in range(n):
    print("Enter Hypothesis:",i+1,"\n")
    temp=[]
    for x in columns:
        t=input("Enter value for: "+x+": ")
        temp.append(t)
    d.append(temp)
```

```
Enter number of rows:3
Enter the data:

Enter Hypothesis: 1

Enter value for: Time: eve
Enter value for: Weather: sunny
Enter value for: Temperature: warm
Enter value for: humidity: mild
Enter value for: Enjoying?: yes
Enter Hypothesis: 2

Enter value for: Time: eve
Enter value for: Weather: rainy
Enter value for: Temperature: cold
Enter value for: humidity: less
Enter value for: Enjoying?: yes
Enter Hypothesis: 3

Enter value for: Time: eve
Enter value for: Weather: sunny
Enter value for: Temperature: warm
Enter value for: humidity: mild
Enter value for: Enjoying?: no
```

```
In [ ]:
for x in d:
    print(x)
```



```
Enter value for: Time: eve
Enter value for: Weather: rainy
Enter value for: Temperature: cold
Enter value for: humidity: less
Enter value for: Enjoying?: yes
Enter Hypothesis: 3

Enter value for: Time: eve
Enter value for: Weather: sunny
Enter value for: Temperature: warm
Enter value for: humidity: mild
Enter value for: Enjoying?: no
```

In [ ]:

```
for x in d:
    print(x)
hypo=[]
for i in range(len(d[0])):
    hypo.append("?")
for i in range(len(d)):
    if d[i][len(d[0])-1]=='yes':
        hypo=d[i]
```

In [ ]:

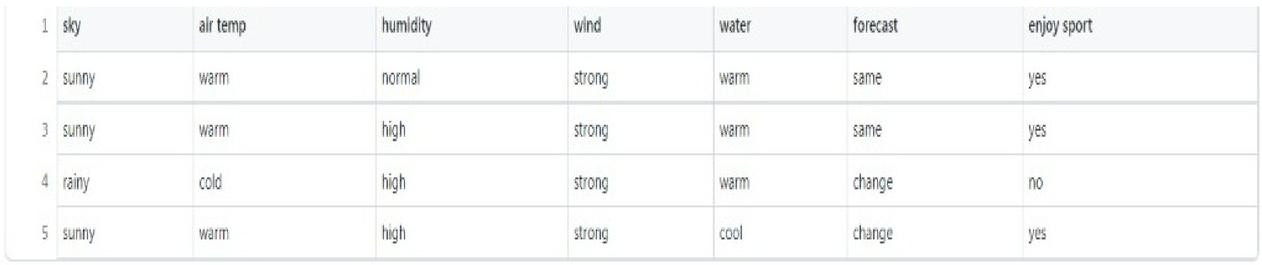
```
for i in range(len(d)):
    if d[i][len(d[0])-1]=='yes':
        for j in range(len(d[0])):
            if(d[i][j]!=hypo[j]):
                hypo[j]="?"
```

In [ ]:

```
print(hypo)
```

In [ ]:

In [ ]:



sky	air temp	humidity	wind	water	forecast	enjoy sport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

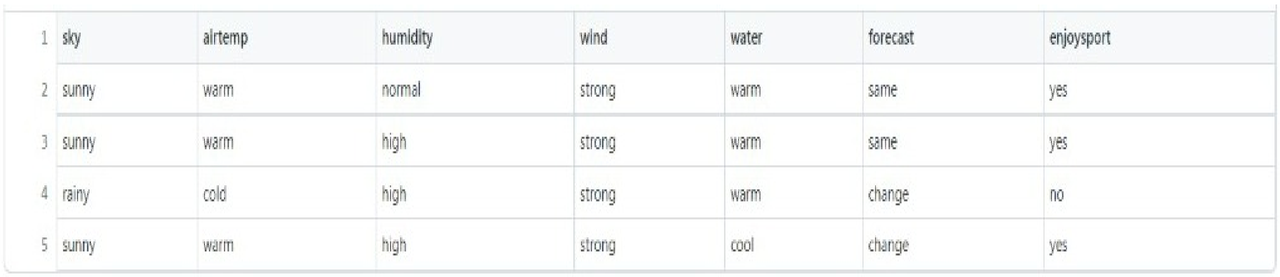
LAB PROGRAM 2:

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

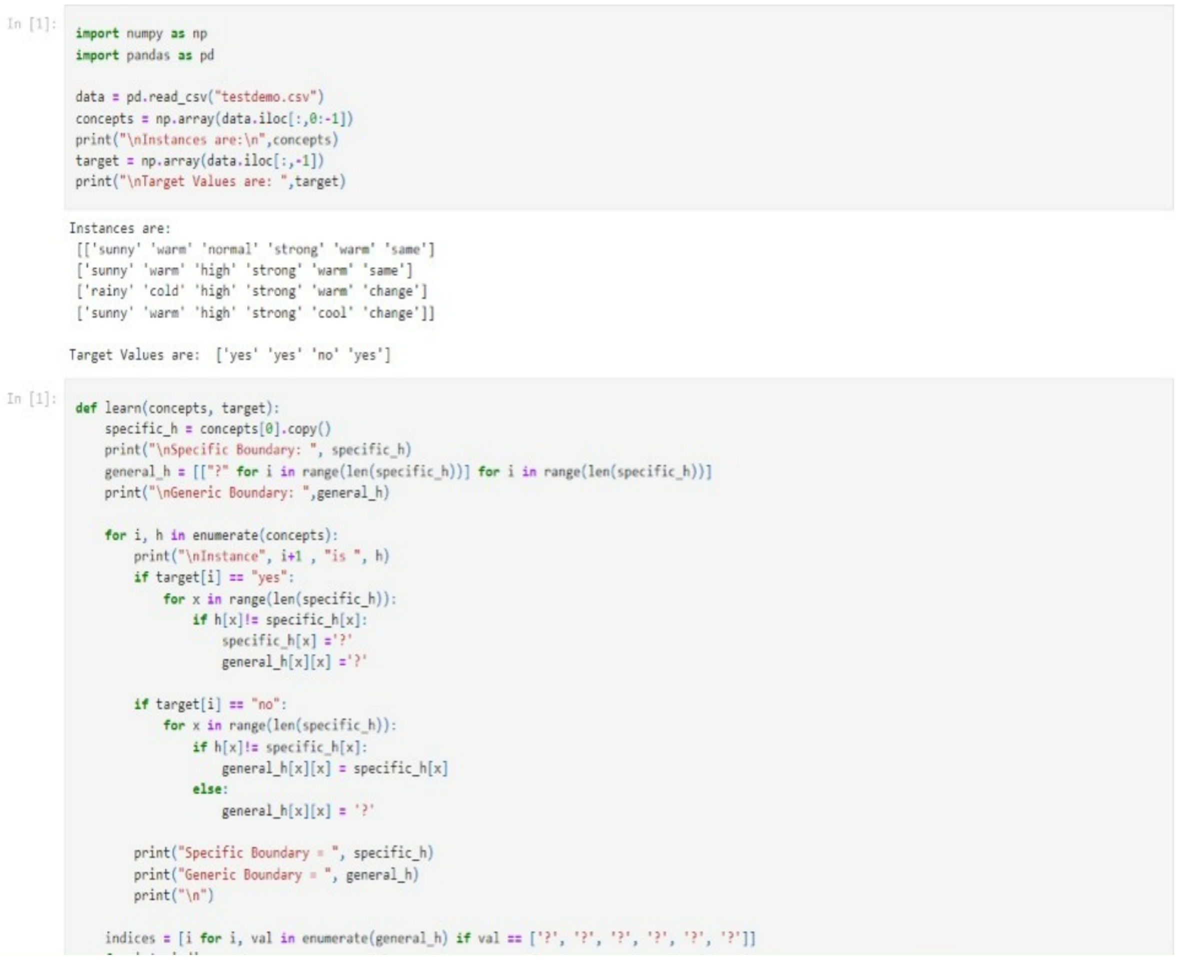
CODE:

```
import numpy as np
import pandas as pd

data = pd.read_csv("testdemo.csv")
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Boundary = ", specific_h)
        print("Generic Boundary = ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?',
'?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
```

```
s_final, g_final = learn(concepts, target)

print(" The Final Specific_h : ", s_final, sep="\n")
print("The Final General_h : ", g_final, sep="\n")
```



sky	airtemp	humidity	wind	water	forecast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes



```
In [1]:
import numpy as np
import pandas as pd

data = pd.read_csv("testdemo.csv")
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)
```

```
Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values are:  ['yes' 'yes' 'no' 'yes']
```

```
In [1]:
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Boundary = ", specific_h)
        print("Generic Boundary = ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
```

In [1]:

```python
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Boundary = ", specific_h)
        print("Generic Boundary = ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print(" The Final Specific_h : ", s_final, sep="\n")
print("The Final General_h : ", g_final, sep="\n")
```

LAB PROGRAM 3:

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Code:

```
import
math
         import csv
         def load_csv(filename):
             lines=csv.reader(open(filename,"r"));
             dataset = list(lines)
             headers = dataset.pop(0)
             return dataset,headers


         class Node:
             def __init__(self,attribute):
                 self.attribute=attribute
                 self.children=[]
                 self.answer=""

         def subtables(data,col,delete):
             dic={}
             coldata=[row[col] for row in data]
             attr=list(set(coldata))

             counts=[0]*len(attr)
             r=len(data)
             c=len(data[0])
             for x in range(len(attr)):
                 for y in range(r):
                     if data[y][col]==attr[x]:
                         counts[x]+=1

             for x in range(len(attr)):
                 dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
                 pos=0
                 for y in range(r):
                     if data[y][col]==attr[x]:
                         if delete:
                             del data[y][col]
                         dic[attr[x]][pos]=data[y]
                         pos+=1
             return attr,dic

         def entropy(S):
             attr=list(set(S))
             if len(attr)==1:
                 return 0
```

```
    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums


def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy


def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]



    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node


def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return
    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)



def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)

'''Main program'''
dataset,features=load_csv("id3.csv")
node1=build_tree(dataset,features)


print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3_test_1.csv")


for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:")
    classify(node1,xtest,features)
```

In [1]:

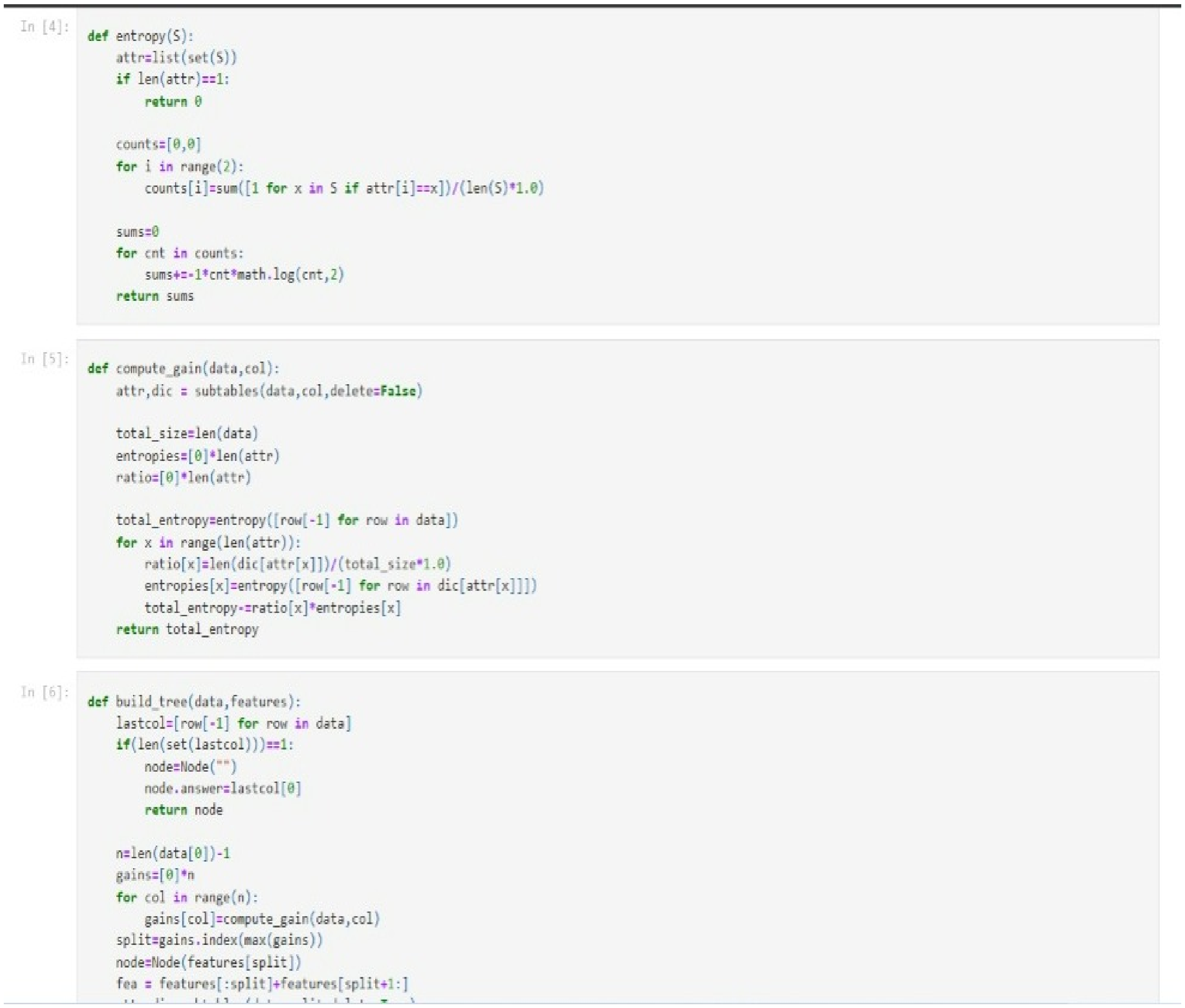
```python
import math
import csv
```

In [2]:

```python
def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

class Node:
    def __init__ (self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""
```

In [3]:

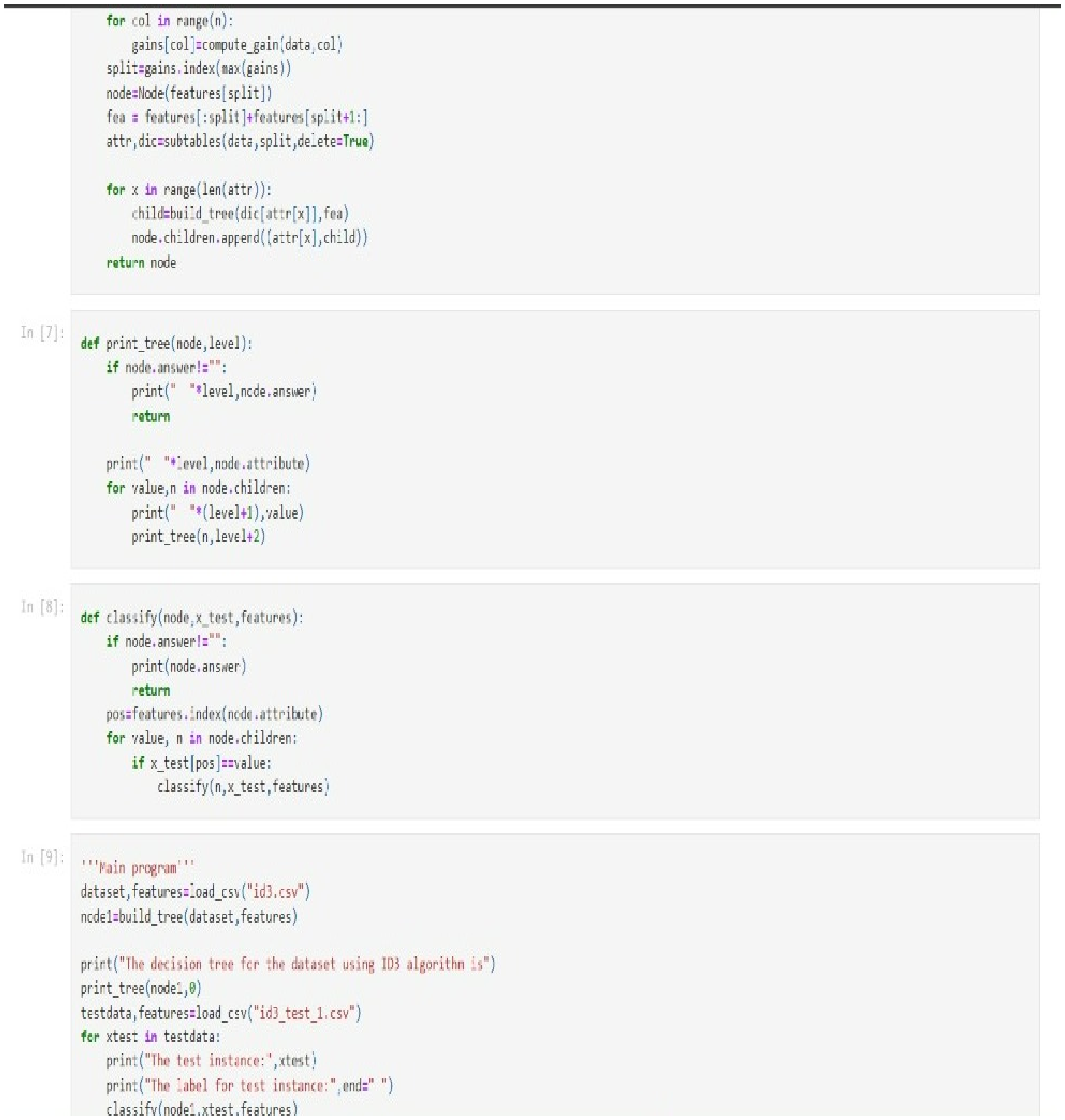
```python
def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1
    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic
```



```
In [4]:
def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums
```

```
In [5]:
def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy
```

```
In [6]:
def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]
```



```
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]
    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node
```

In [7]:

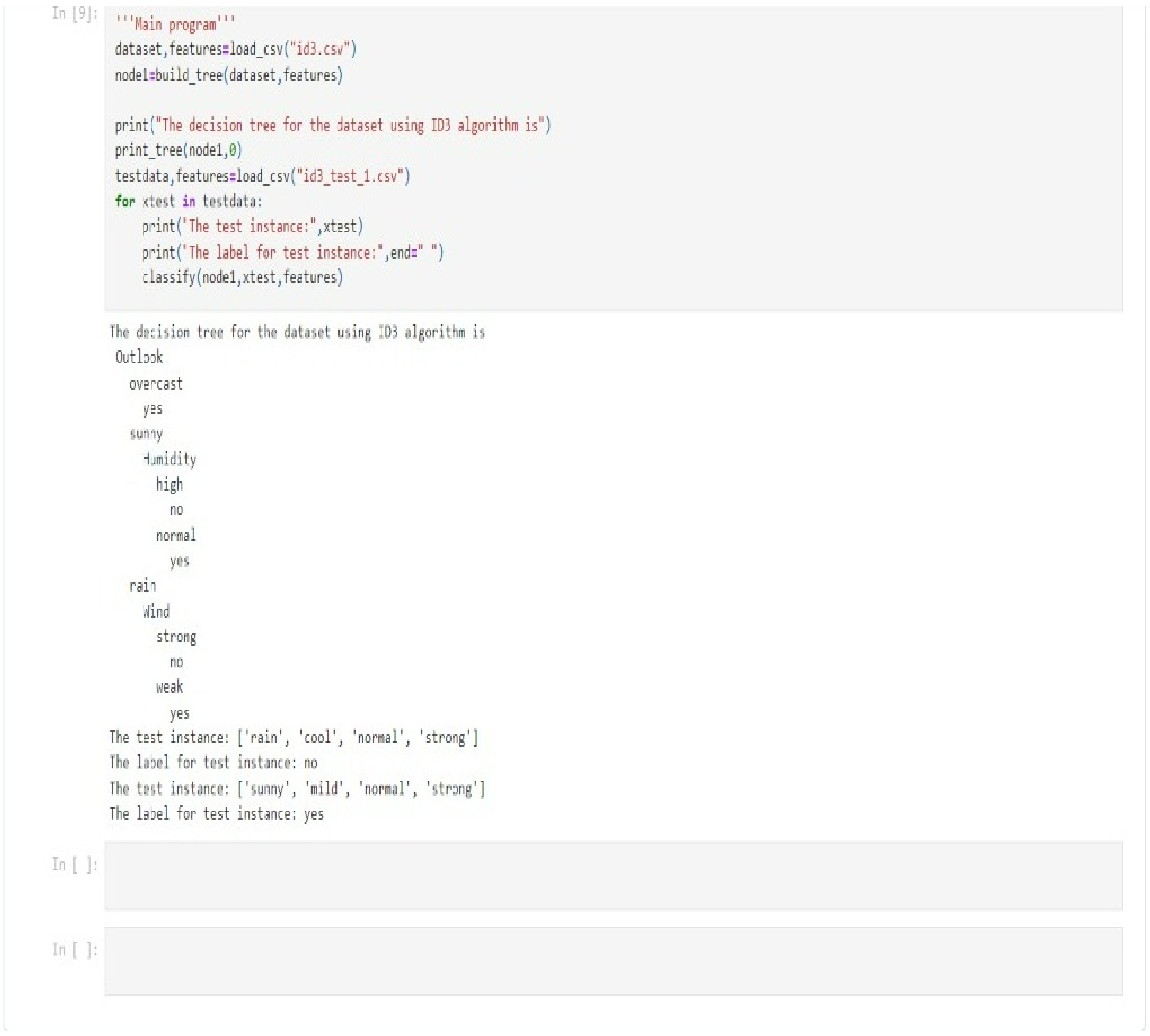
```
def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)
```

In [8]:

```
def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)
```

In [9]:

```
'''Main program'''
dataset,features=load_csv("id3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3_test_1.csv")
for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end=" ")
    classify(node1,xtest,features)
```

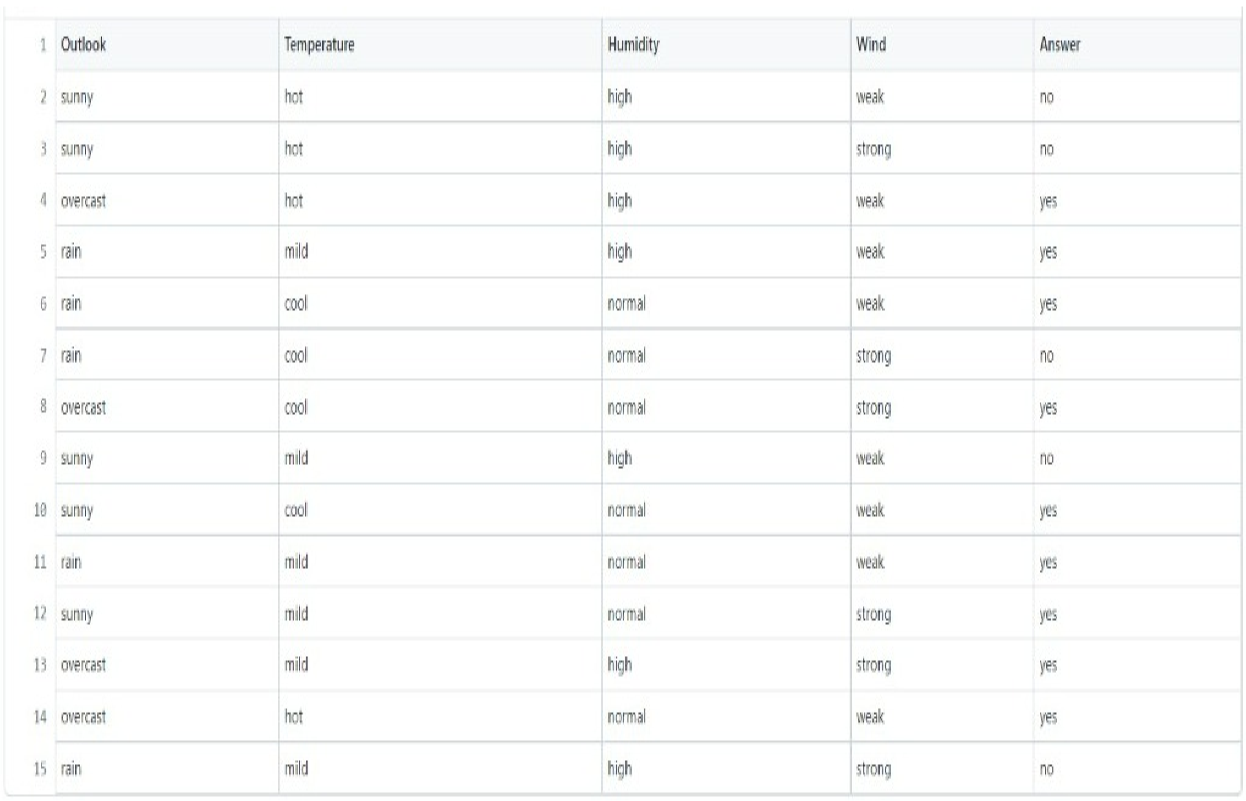


```
In [9]: '''Main program'''
        dataset,features=load_csv("id3.csv")
        node1=build_tree(dataset,features)

        print("The decision tree for the dataset using ID3 algorithm is")
        print_tree(node1,0)
        testdata,features=load_csv("id3_test_1.csv")
        for xtest in testdata:
            print("The test instance:",xtest)
            print("The label for test instance:",end=" ")
            classify(node1,xtest,features)
```

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   overcast
     yes
   sunny
     Humidity
       high
         no
       normal
         yes
   rain
     Wind
       strong
         no
       weak
         yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance: no
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance: yes
```

```
In [ ]:
```

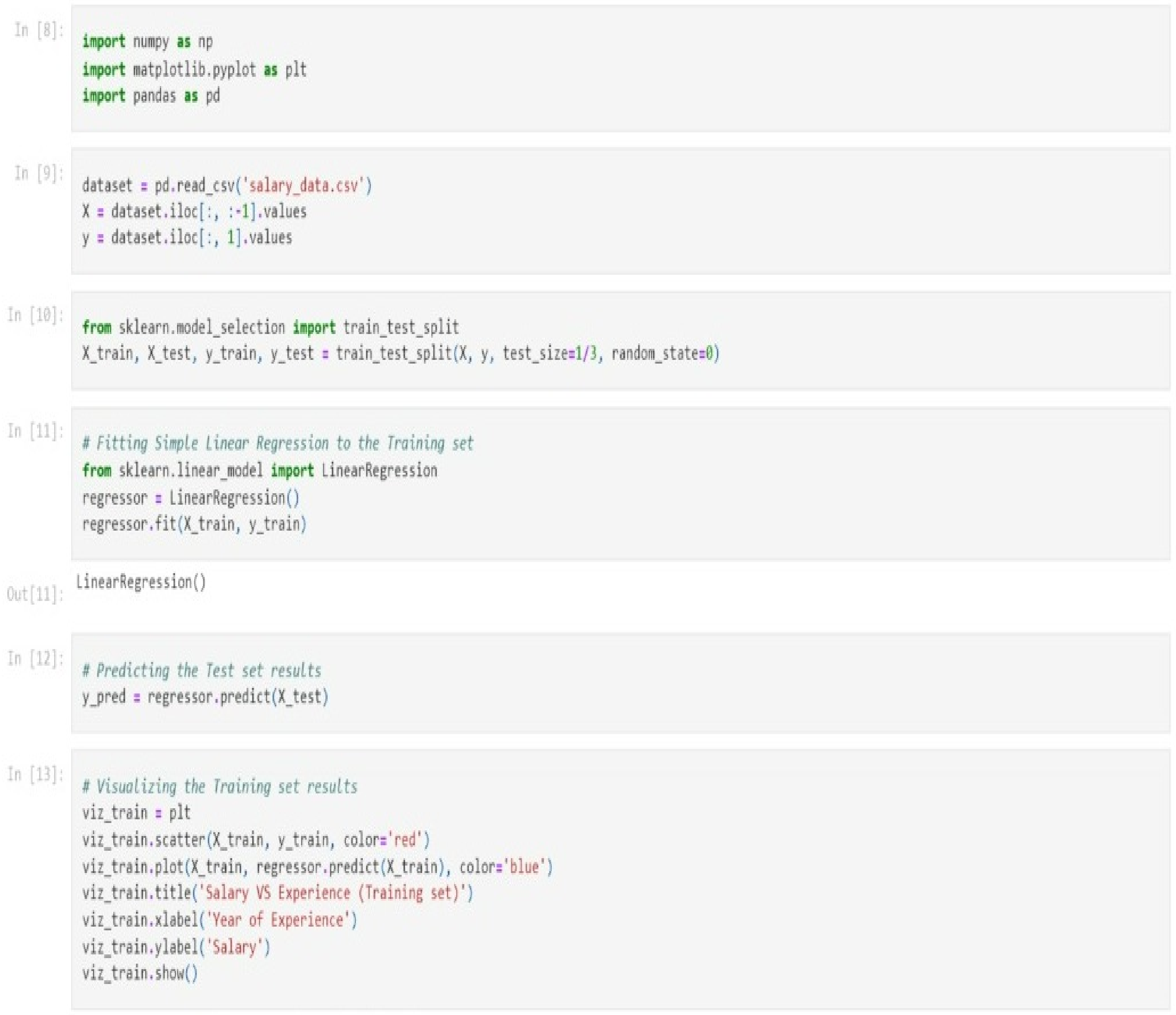
```
In [ ]:
```



Outlook	Temperature	Humidity	Wind	Answer
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes
rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no

LAB PROGRAM 4:

Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.



```
In [8]:
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

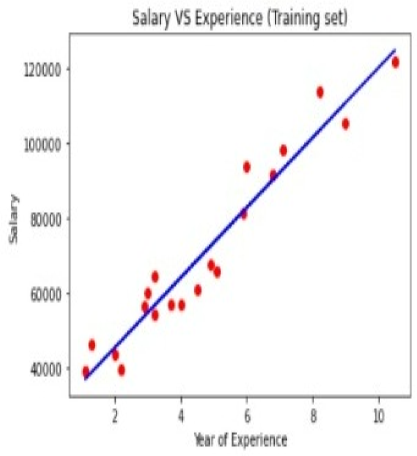
```
In [9]:
dataset = pd.read_csv('salary_data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

```
In [10]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
```

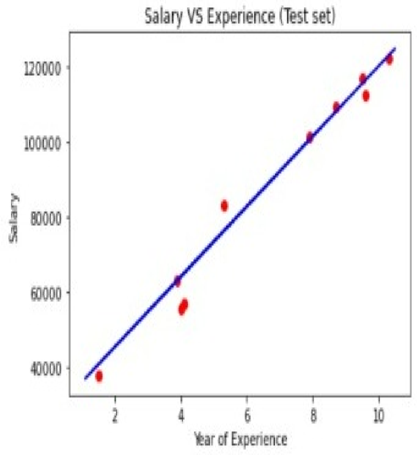
```
In [11]:
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

Out[11]:
LinearRegression()
```

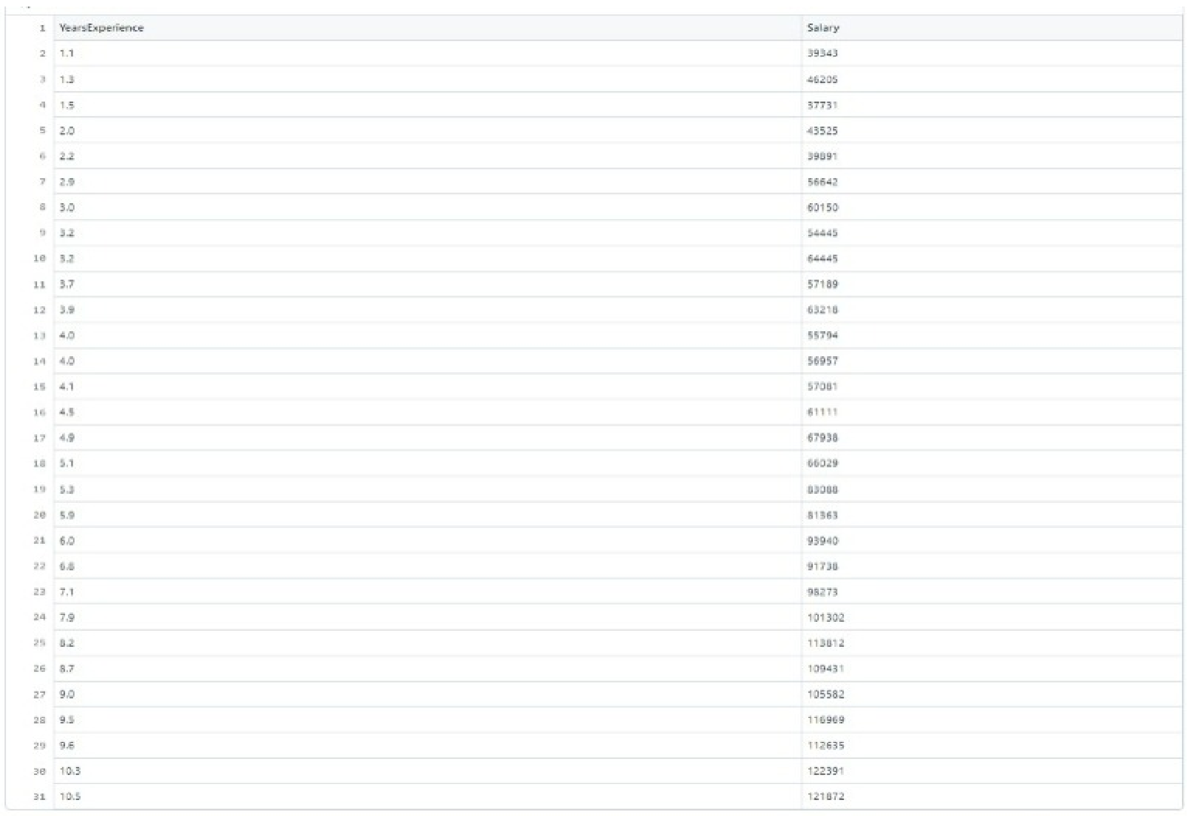
```
In [12]:
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

```
In [13]:
# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()
```



```
In [14]:
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```





YearsExperience	Salary
1.1	39343
1.3	46205
1.5	37731
2.0	43525
2.2	39891
2.9	56642
3.0	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4.0	55794
4.0	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029
5.3	83088
5.9	81363
6.0	93940
6.8	91738
7.1	98273
7.9	101302
8.2	113812
8.7	109431
9.0	105582
9.5	116969
9.6	112635
10.3	122391
10.5	121872

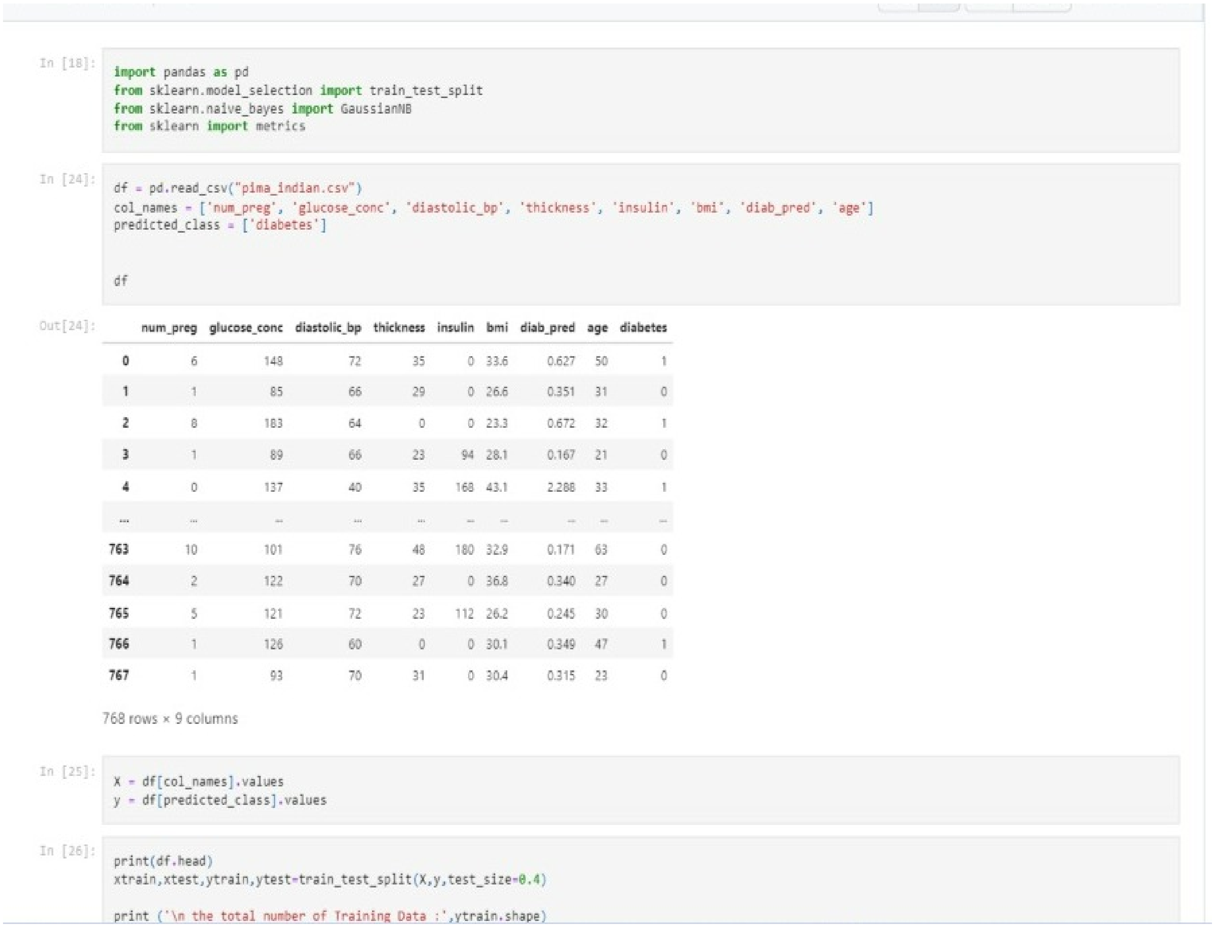
PROGRAM 5:

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn import metrics

df = pd.read_csv("pima_indian.csv")
```

```
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi',
'diab_pred', 'age']

predicted_class_names = ['diabetes']

X = df[feature_col_names].values # these are factors for the prediction

y = df[predicted_class_names].values # this is what we want to predict

#splitting the dataset into train and test data

xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)

print ('\n the total number of Training Data :',ytrain.shape)

print ('\n the total number of Test Data :',ytest.shape)

# Training Naive Bayes (NB) classifier on training data.clf = GaussianNB().fit(xtrain,ytrain.ravel())

predicted = clf.predict(xtest)

predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])

#printing Confusion matrix, accuracy, Precision and Recall

print('\n Confusion matrix')

print(metrics.confusion_matrix(ytest,predicted))

print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```



In [18]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

In [24]:

```
df = pd.read_csv("pima_indian.csv")
col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
predicted_class = ['diabetes']


df
```

Out[24]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

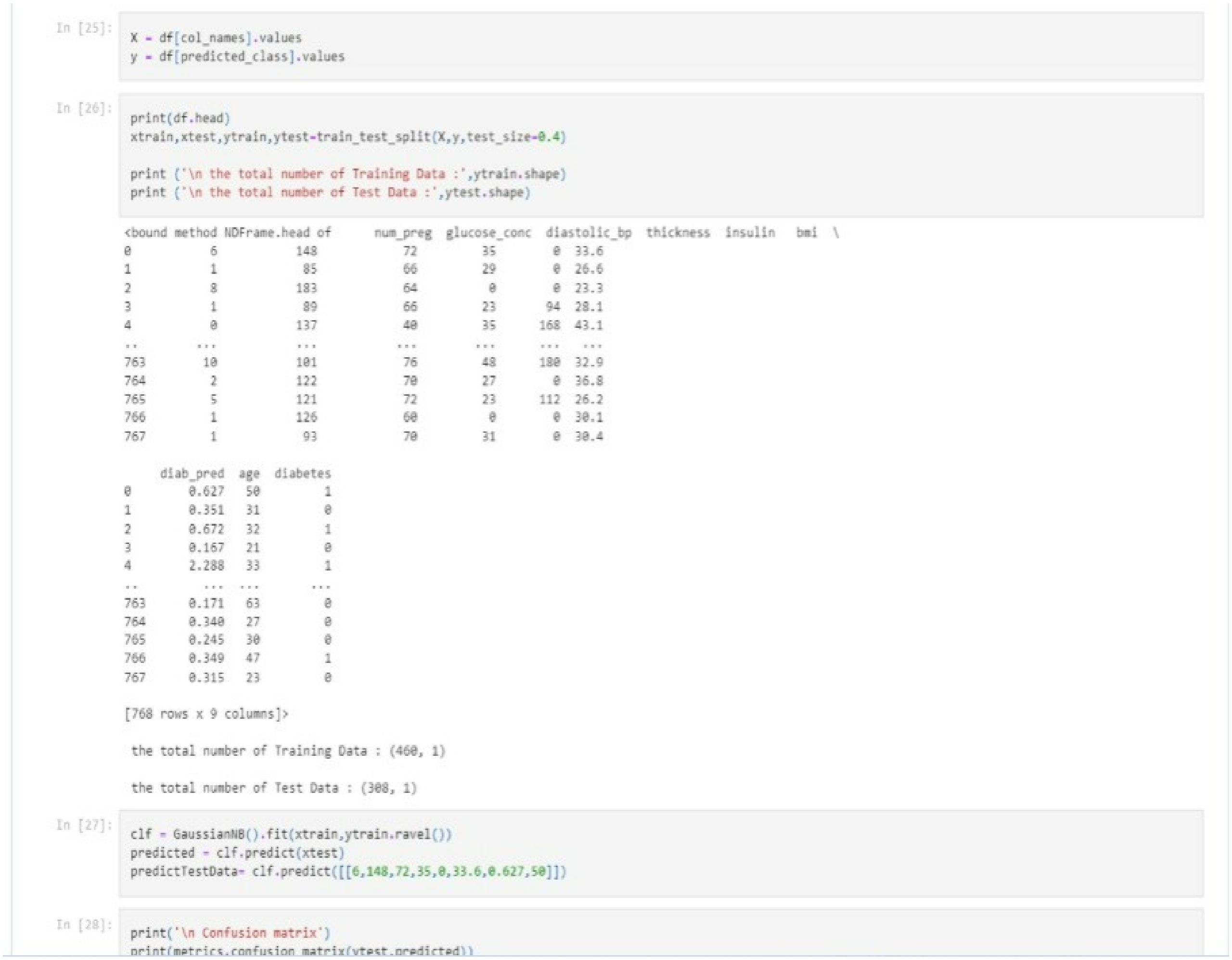
768 rows × 9 columns

In [25]:

```
X = df[col_names].values
y = df[predicted_class].values
```

In [26]:

```
print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.4)

print ('\n the total number of Training Data :',ytrain.shape)
```



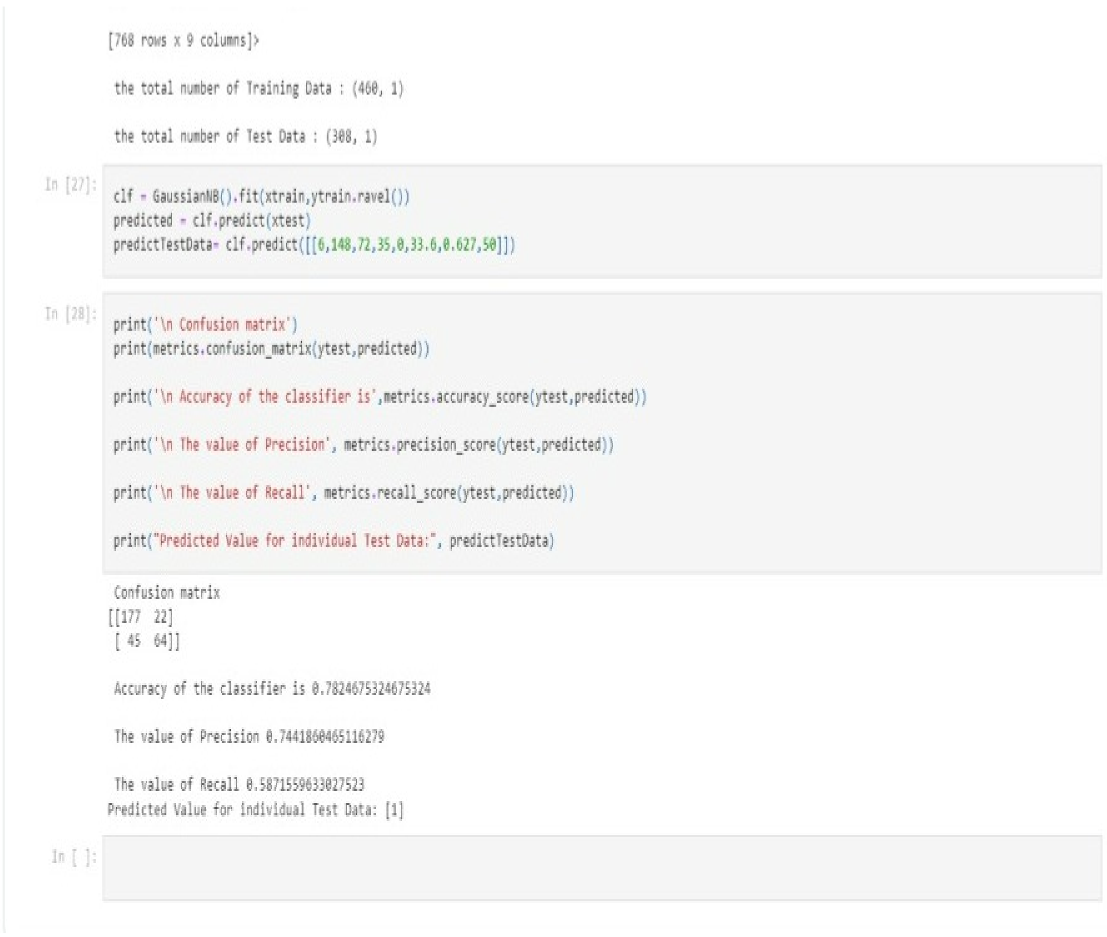
```
In [25]:
X = df[col_names].values
y = df[predicted_class].values
```

```
In [26]:
print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.4)

print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)
```

```
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  thickness  insulin   bmi  \
0           6           148            72         35        0  33.6
1           1            85            66         29        0  26.6
2           8           183            64          0        0  23.3
3           1            89            66         23       94  28.1
4           0           137            40         35      168  43.1
..        ...           ...           ...        ...      ...   ...
763        10           101            76         48      180  32.9
764         2           122            70         27        0  36.8
765         5           121            72         23      112  26.2
766         1           126            60          0        0  30.1
767         1            93            70         31        0  30.4

     diab_pred  age  diabetes
0        0.627   50         1
1        0.351   31         0
2        0.672   32         1
3        0.167   21         0
4        2.288   33         1
..         ...  ...       ...
763      0.171   63         0
764      0.340   27         0
765      0.245   30         0
766      0.349   47         1
767      0.315   23         0

[768 rows x 9 columns]>

 the total number of Training Data : (460, 1)

 the total number of Test Data : (308, 1)
```

```
In [27]:
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
```

```
In [28]:
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```



```
[768 rows x 9 columns]>

 the total number of Training Data : (460, 1)

 the total number of Test Data : (308, 1)
```

In [27]:
```
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
```

In [28]:
```
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

```
 Confusion matrix
[[177  22]
 [ 45  64]]

 Accuracy of the classifier is 0.7824675324675324

 The value of Precision 0.7441860465116279

 The value of Recall 0.5871559633027523
Predicted Value for individual Test Data: [1]
```

In [ ]: