VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING

Submitted by

MITHILRAJ (1BM19CS086)

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 "LAB COURSE **MACHINE LEARNING**" carried out by **MITHILRAJ (1BM19CS086),** 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 **Course Title - (Course code)** work prescribed for the said degree.

SARITHA AN Lab-Incharge
Assistant prof
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

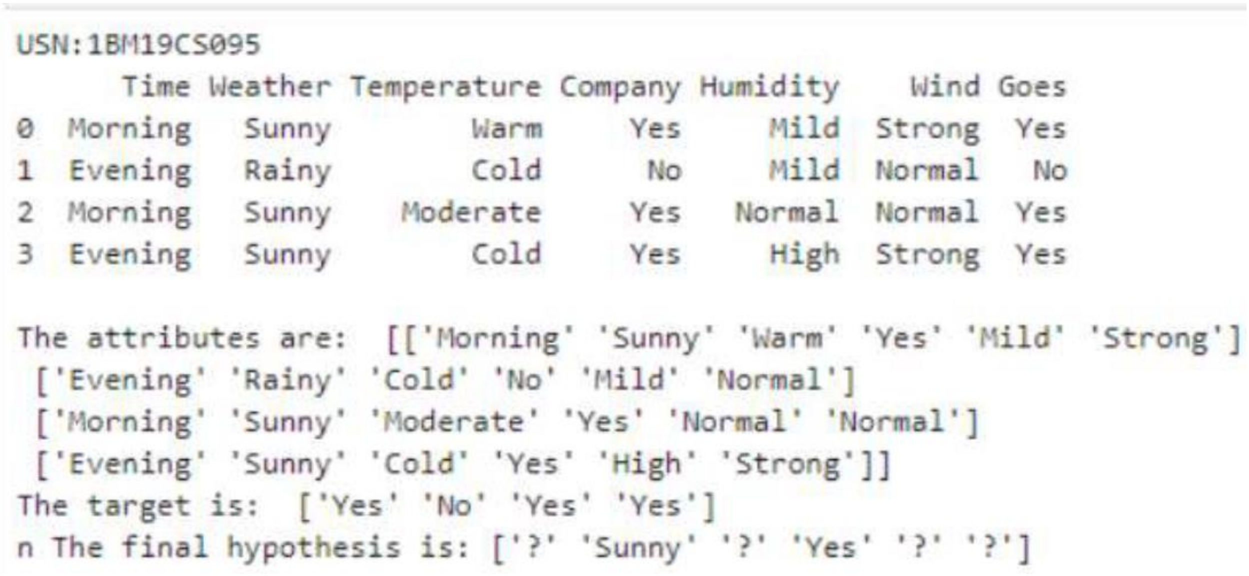
`

Index Sheet

LAB:- FIND-S ALGORITHM

```
import pandas as pd
import numpy as np

#to read the data in the csv file
print("USN:1BM19CS095")
data = pd.read_csv(r"C:\Users\admin\Downloads\data.csv")
print(data,"\n")

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("The attributes are: ",d)
```

```
#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
for i, val in enumerate(t):
if val == "Yes":
specific_hypothesis = c[i].copy()

break
for i, val in enumerate(c):
if t[i] == "Yes":
for x in range(len(specific_hypothesis)):
if val[x] != specific_hypothesis[x]:
specific_hypothesis[x] = '?'
else:
pass
return specific_hypothesis

#obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))
```

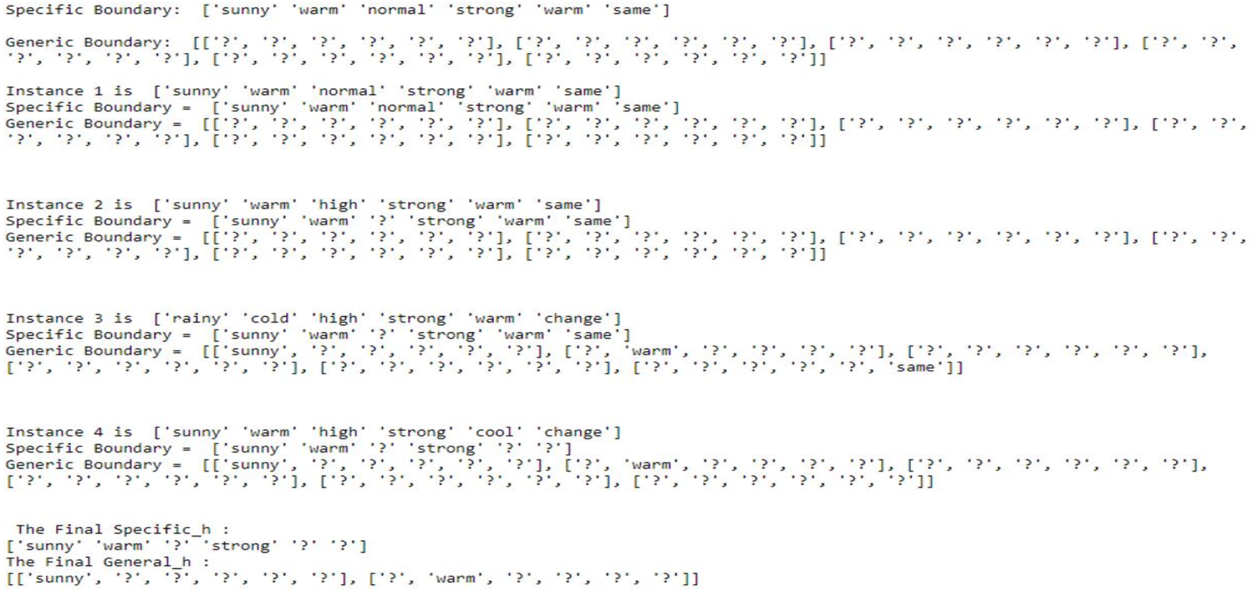


```
USN:1BM19CS095
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes

The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]
The target is:  ['Yes' 'No' 'Yes' 'Yes']
n The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

CANDIDATE ELIMINATION ALGORITHM:-

```
Import
numpy
as np
import pandas as pd
data = pd.read_csv(r'C:\Users\admin\Downloads\enjoysport.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("\nInitialization of specific_h and genearal_h")
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":
print("Instance is Positive ")
for x in range(len(specific_h)):
if h[x]!= specific_h[x]:
specific_h[x] ='?'
general_h[x][x] ='?'
if target[i] == "no":
print("Instance is Negative ")
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 Bundary after ", i+1, "Instance is ", specific_h)
print("Generic Boundary after ", i+1, "Instance is ", 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("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

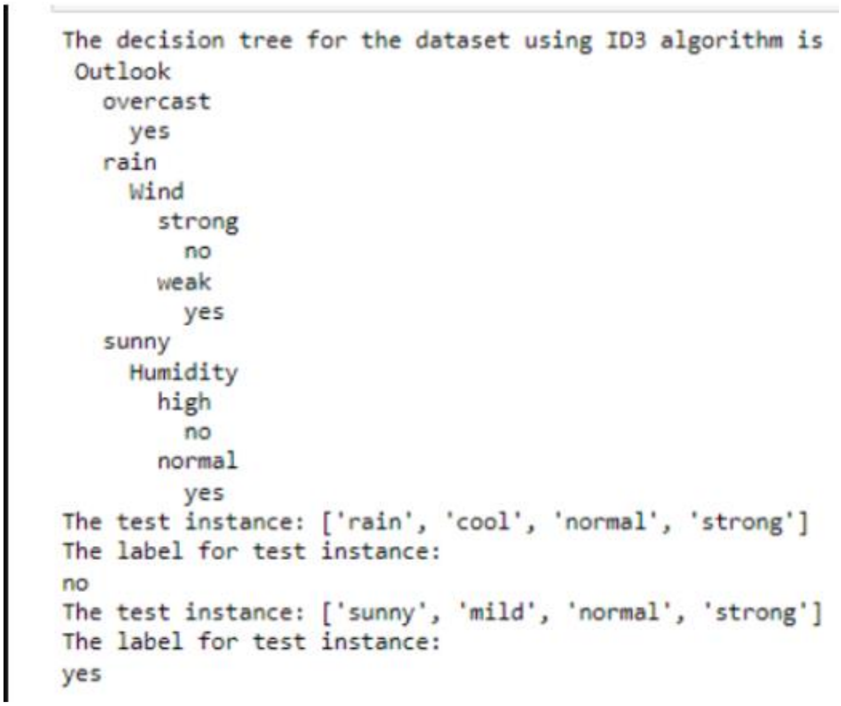


```
Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Specific Boundary =  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary =  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Specific Boundary =  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary =  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Specific Boundary =  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary =  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]


Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Specific Boundary =  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary =  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 The Final Specific_h :
['sunny' 'warm' '?' 'strong' '?' '?']
The Final General_h :
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

DECISION TREE ALGORITHM:-

```
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(r"C:\Users\admin\Downloads\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(r"C:\Users\admin\Downloads\id3_test.csv")

for xtest in testdata:
print("The test instance:",xtest)
print("The label for test instance:")

classify(node1,xtest,features)
```

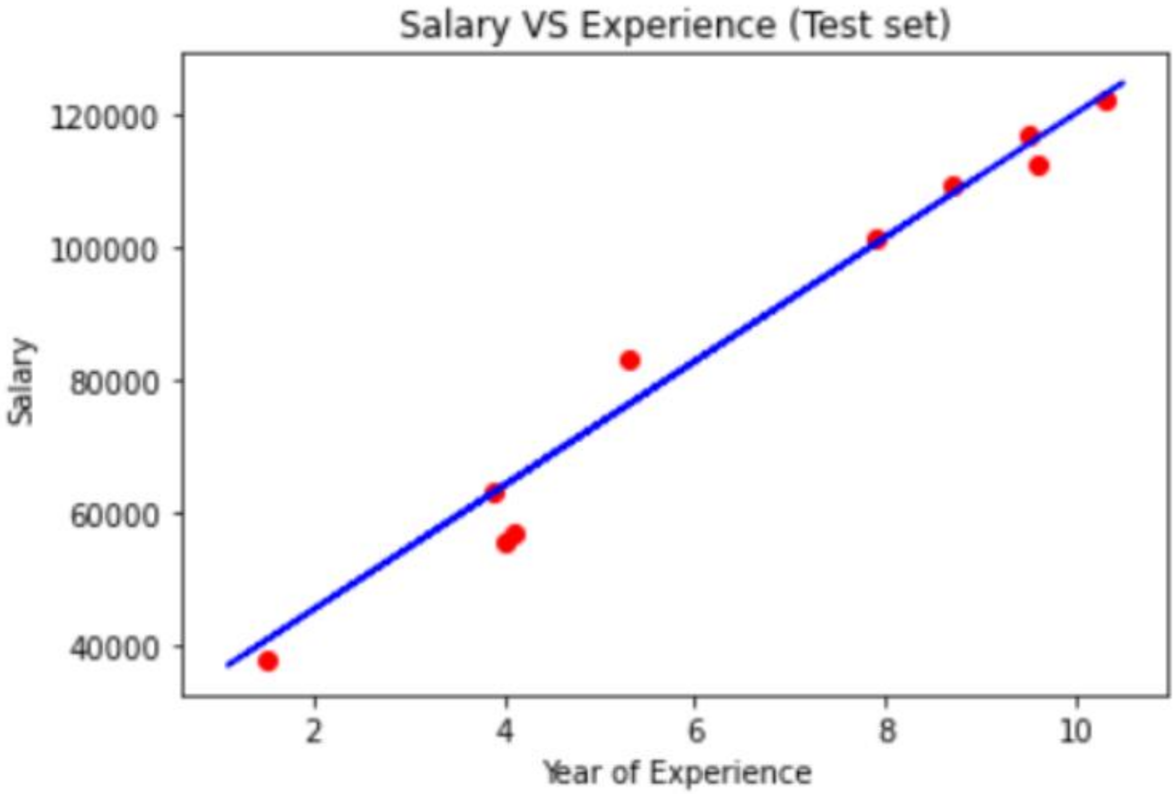


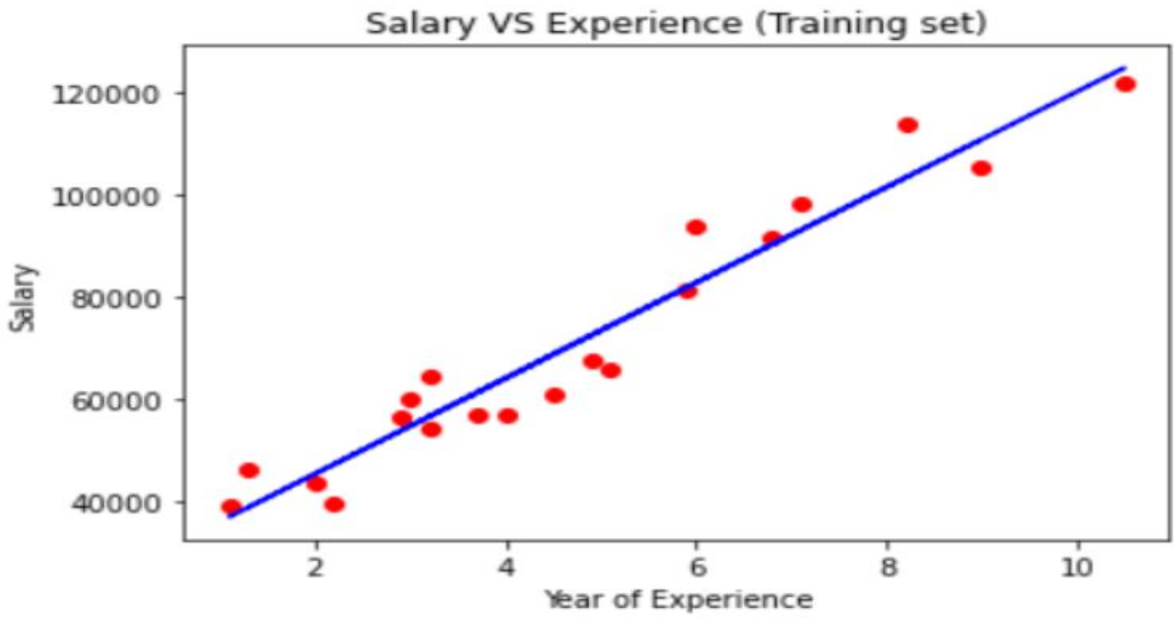
```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   overcast
     yes
   rain
     Wind
       strong
         no
       weak
         yes
   sunny
     Humidity
       high
         no
       normal
         yes
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for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

LINEAR REGRESSION:-

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

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

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3,
random_state=0)

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)

# Visualizing the Training set results
viz_train = plt
```

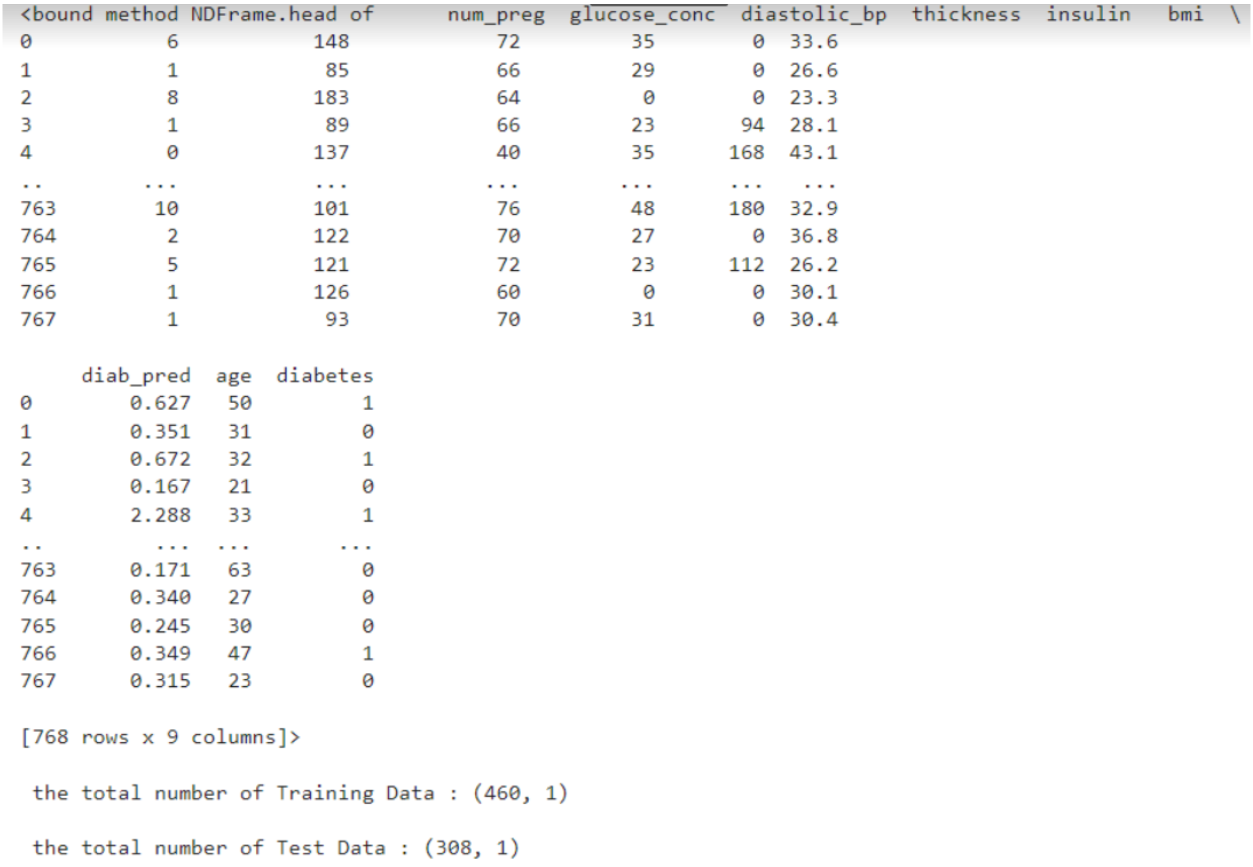
```
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()

# 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()
```



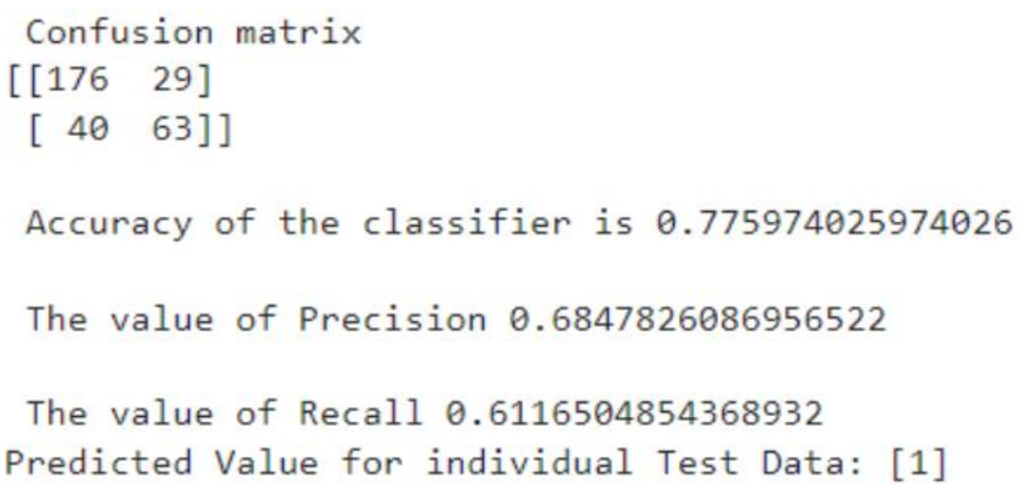


NAÏVE BAYES ALGORITHM:-

```
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(r"C:\Users\admin\Downloads\data5.csv")
col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness',
'insulin', 'bmi', 'diab_pred', 'age']
predicted_class = ['diabetes']
X = df[col_names].values
y = df[predicted_class].values

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)
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)

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

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)
```



```
<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)
```



```
 Confusion matrix
[[176  29]
 [ 40  63]]

 Accuracy of the classifier is 0.775974025974026

 The value of Precision 0.6847826086956522

 The value of Recall 0.6116504854368932
Predicted Value for individual Test Data: [1]
```

------------------------------------x------------------------------------