VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

COURSE TITLE

Submitted by

Naman Singh (1BM19CS093)

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 **Naman Singh (1BM19CS093),** who is a 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 - (Course code)** work prescribed for the said degree.

Name of the Lab-Incharge
Designation
Department of CSE
BMSCE, Bengaluru

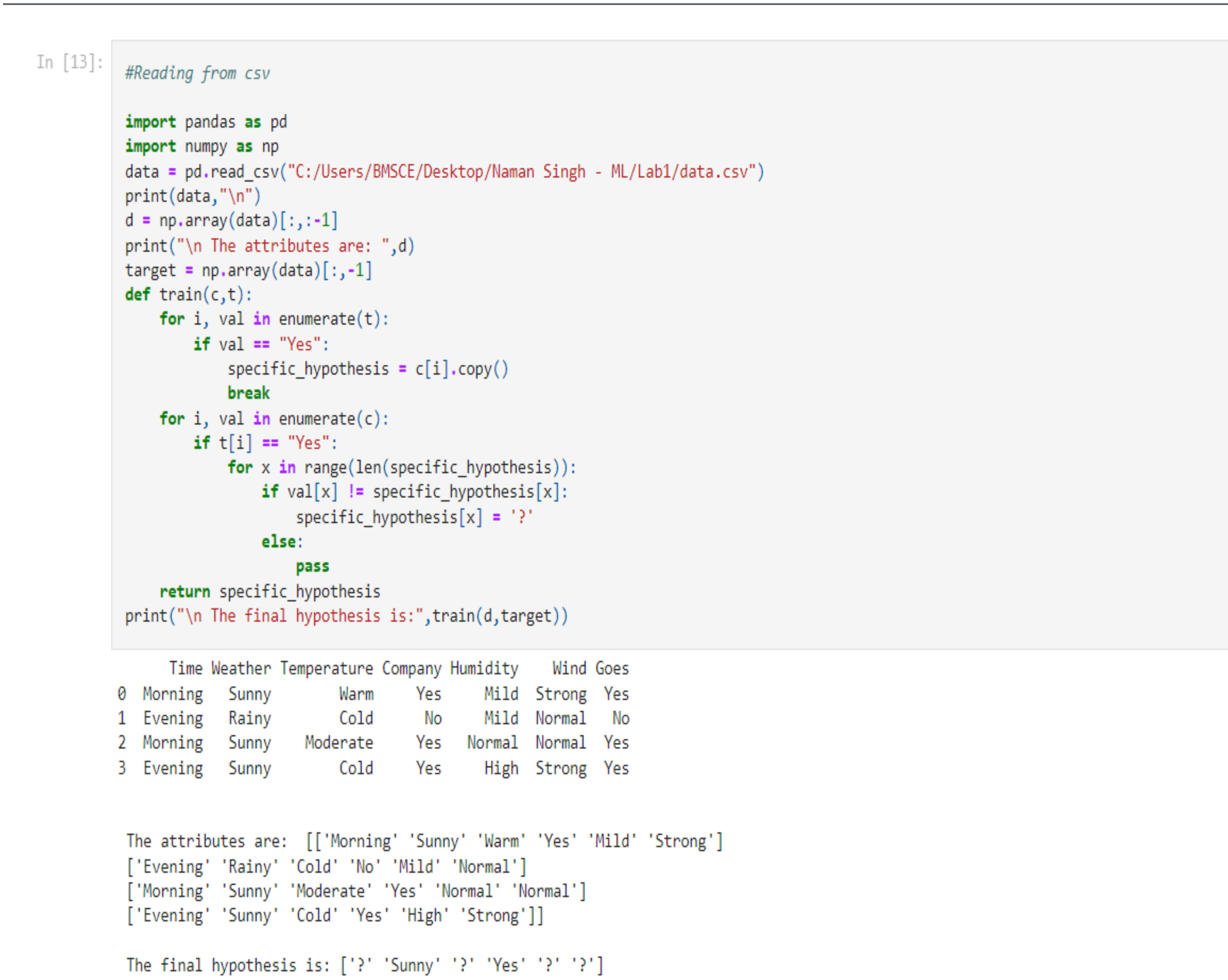
Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

`

Index Sheet

Course Outcome

Lab 1 : Find - S



```python
#Reading from csv

import pandas as pd
import numpy as np
data = pd.read_csv("C:/Users/BMSCE/Desktop/Naman Singh - ML/Lab1/data.csv")
print(data,"\n")
d = np.array(data)[:,:-1]
print("\n The attributes are: ",d)
target = np.array(data)[:,-1]
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
print("\n The final hypothesis is:",train(d,target))
```

```
      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 final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

In [10]:

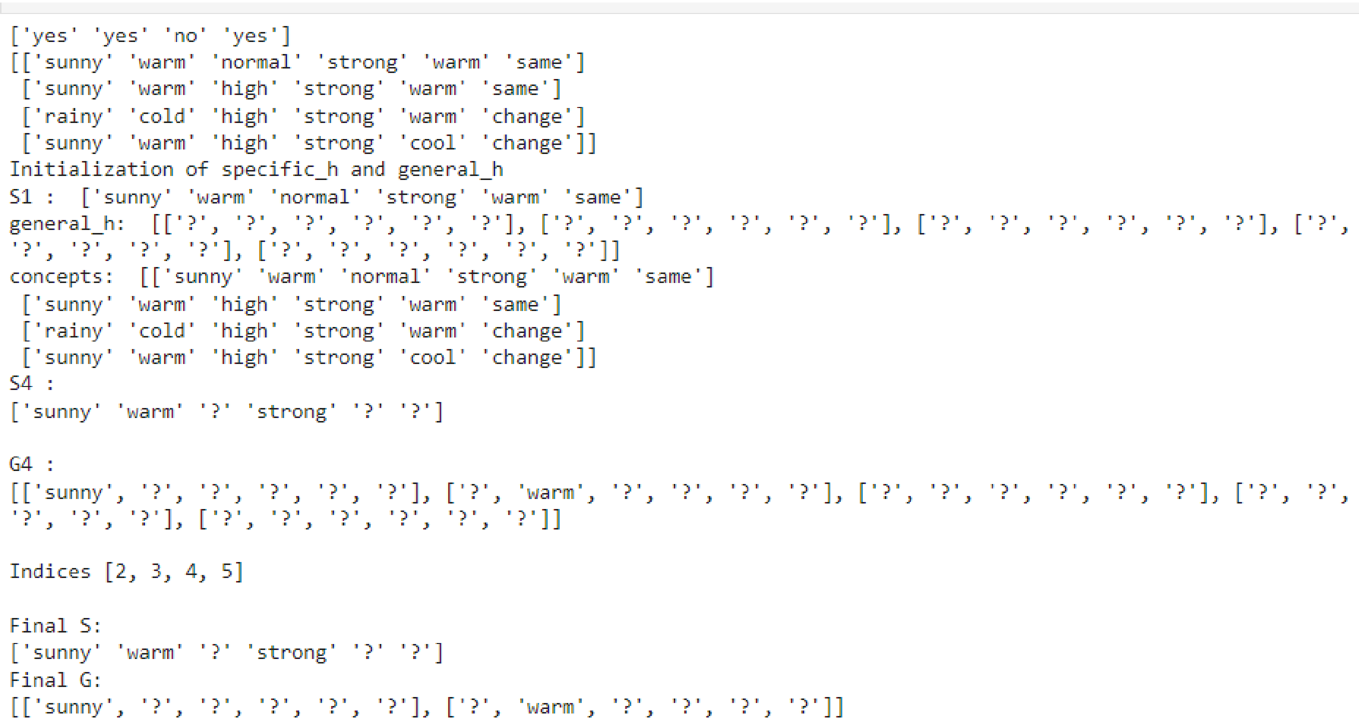
```
import pandas as pd
import numpy as np
data =  [['Morning','Sunny','Warm','Yes','Mild','Strong','Yes'],
 ['Evening','Rainy','Cold','No','Mild','Normal','No'],
 ['Morning','Sunny','Moderate','Yes','Normal','Normal','Yes'],
 ['Evening','Sunny','Cold','Yes','High','Strong','Yes']]
print(data,"\n")
d = np.array(data)[:,:-1]
print("\n The attributes are: ",d)
target = np.array(data)[:,-1]
print("\n The target is: ",target)
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
print("\n The final hypothesis is:",train(d,target))
```

```
[['Morning', 'Sunny', 'Warm', 'Yes', 'Mild', 'Strong', 'Yes'], ['Evening', 'Rainy', 'Cold', 'No', 'Mild', 'Normal', 'No
te', 'Yes', 'Normal', 'Normal', 'Yes'], ['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']

 The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

Lab 2 :

In [11]:

```
import numpy as np
import pandas as pd
data = pd.read_csv("data.csv")
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
print(target)
print(concepts)
def learn(concepts, target):
    for i,val in enumerate(target) :
        if val == "yes" :
            specific_h = concepts[i].copy()
            idx=i
            break
    print("Initialization of specific_h and general_h")
    print(f"S{idx+1} : ",specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("general_h: ",general_h)
    print("concepts: ",concepts)
    for i, h in enumerate(concepts):
        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(f"S{i+1} : ")
    print(specific_h,"\n")
    print(f"G{i+1} : ")
    print(general_h)
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    print("\nIndices",indices)
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final,g_final = learn(concepts, target)
print("\nFinal S:", s_final, sep="\n")
print("Final G:", g_final, sep="\n")
```



```
['yes' 'yes' 'no' 'yes']
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
Initialization of specific_h and general_h
S1 :  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
general_h:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
concepts:  [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
S4 :
['sunny' 'warm' '?' 'strong' '?' '?']

G4 :
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Indices [2, 3, 4, 5]

Final S:
['sunny' 'warm' '?' 'strong' '?' '?']
Final G:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Lab 3 :

In [61]:

```
import pandas as pd
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import matplotlib.image as img
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
data = pd.read_csv("car_evaluation.csv")
print(data)
```

```
     buying_price maintanence_cost number_of_doors number_of_persons lug_boot  \
0           vhigh            vhigh               2                 2    small   
1           vhigh            vhigh               2                 2    small   
2           vhigh            vhigh               2                 2    small   
3           vhigh            vhigh               2                 2      med   
4           vhigh            vhigh               2                 2      med   
...           ...              ...             ...               ...      ...   
1723          low              low           5more              more      med   
1724          low              low           5more              more      med   
1725          low              low           5more              more      big   
1726          low              low           5more              more      big   
1727          low              low           5more              more      big   

     safety decision  
0       low    unacc  
1       med    unacc  
2      high    unacc  
3       low    unacc  
4       med    unacc  
...     ...      ...  
1723    med     good  
1724   high    vgood  
1725    low    unacc  
1726    med     good  
1727   high    vgood  

[1728 rows x 7 columns]
```

In [ ]:

In [62]:

```
list(data.columns[:-1])
```

Out[62]:

```
['buying_price',
 'maintanence_cost',
 'number_of_doors',
 'number_of_persons',
 'lug_boot',
 'safety']
```

In [63]:

```
X=data.drop(data.columns[-1],axis=1)
Y=data[data.columns[-1]]
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.35,random_state=123)
print(X_train.shape)
```

```
(1123, 6)
```

In [65]:

```
le = preprocessing.LabelEncoder()
for column_name in X_train.columns:
    if X_train[column_name].dtype == object:
        X_train[column_name] = le.fit_transform(X_train[column_name])
    else:
        pass
```

In [66]:

```
dtree = DecisionTreeClassifier(criterion="entropy")
dtree = dtree.fit(X_train,Y_train)
```

In [68]:

```
le = preprocessing.LabelEncoder()
for column_name in X_test.columns:
    if X_test[column_name].dtype == object:
        X_test[column_name] = le.fit_transform(X_test[column_name])
    else:
        pass
```

In [69]:

```
y_pred = dtree.predict(X_test)
```

In [70]:

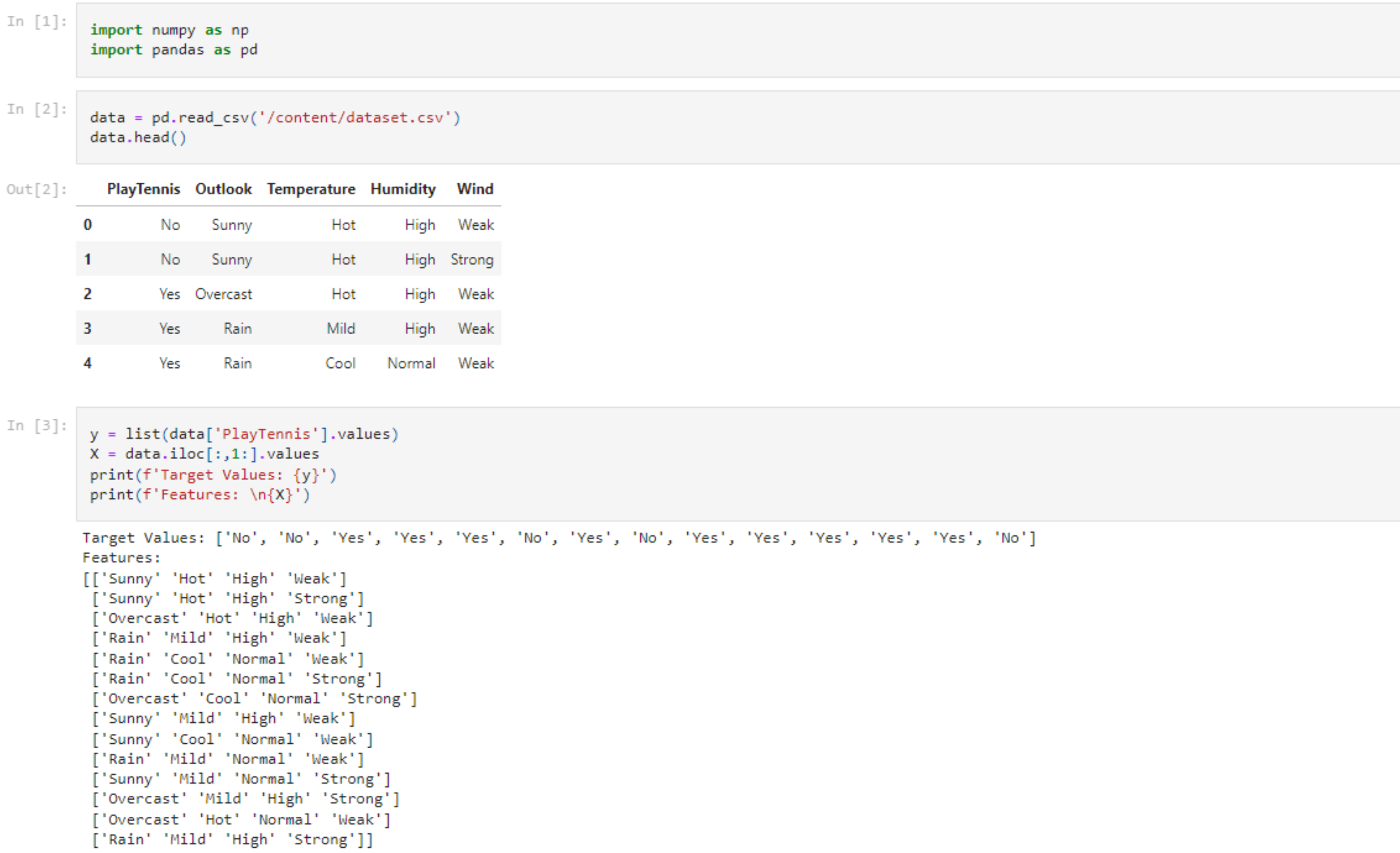
```
print(y_pred)
```

```
 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'unacc'
 'acc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'good' 'unacc' 'acc' 'acc'
 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'unacc' 'acc'
 'unacc' 'acc' 'acc' 'unacc' 'acc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc'
 'vgood' 'acc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc'
 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc'
 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'vgood' 'acc' 'acc' 'acc' 'unacc'
 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'acc' 'unacc' 'acc' 'acc' 'unacc'
 'acc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc'
 'unacc' 'unacc' 'good' 'unacc' 'acc' 'acc' 'acc' 'unacc' 'unacc' 'unacc'
 'unacc' 'unacc' 'acc' 'unacc' 'acc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc'
 'acc' 'good' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc'
 'acc' 'unacc' 'acc' 'acc' 'acc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc'
 'unacc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'vgood' 'acc'
 'unacc' 'unacc' 'unacc' 'good' 'acc' 'unacc' 'unacc' 'good' 'unacc'
 'unacc' 'unacc' 'unacc' 'acc' 'vgood' 'unacc' 'acc' 'unacc' 'unacc' 'acc'
 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'vgood'
 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'unacc' 'unacc' 'good'
 'acc' 'unacc' 'unacc' 'unacc' 'acc' 'acc' 'unacc' 'unacc' 'good' 'unacc'
 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'acc' 'unacc' 'unacc' 'unacc'
 'unacc' 'unacc' 'unacc' 'vgood' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc'
 'unacc' 'unacc' 'unacc' 'vgood' 'unacc' 'unacc' 'vgood' 'unacc' 'unacc'
 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'unacc' 'unacc'
 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'acc' 'acc' 'unacc' 'acc'
 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'unacc' 'unacc'
 'unacc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc'
 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc'
 'unacc' 'good' 'unacc' 'good' 'unacc' 'acc' 'unacc' 'unacc' 'acc' 'vgood'
 'acc' 'acc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc'
 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'acc' 'unacc'
 'unacc' 'acc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc' 'vgood' 'unacc'
 'unacc' 'acc' 'unacc' 'acc' 'unacc' 'acc' 'unacc' 'vgood' 'unacc' 'unacc'
 'acc' 'vgood' 'acc' 'unacc' 'unacc' 'vgood' 'unacc' 'unacc' 'unacc'
 'unacc' 'acc' 'unacc' 'unacc' 'vgood' 'good' 'vgood' 'unacc' 'acc' 'good'
 'unacc' 'acc' 'acc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc'
 'unacc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'acc'
 'unacc' 'unacc' 'acc' 'unacc' 'unacc' 'acc' 'unacc' 'unacc' 'unacc'
 'unacc' 'unacc' 'acc' 'unacc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc'
 'unacc' 'unacc' 'unacc' 'acc' 'unacc' 'unacc' 'unacc' 'unacc']
```

```
print("Accuracy : ",accuracy_score(Y_test,y_pred))
```

```
Accuracy :  0.9735537190082645
```

Lab 4 :



In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
data = pd.read_csv('/content/dataset.csv')
data.head()
```

Out[2]:

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak

In [3]:

```
y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values
print(f'Target Values: {y}')
print(f'Features: \n{X}')
```

```
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
```

In [4]:

```
y_train = y[:8]
y_val = y[8:]
X_train = X[:8]
X_val = X[8:]
print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
```

```
Number of instances in training set: 8
Number of instances in testing set: 6
```

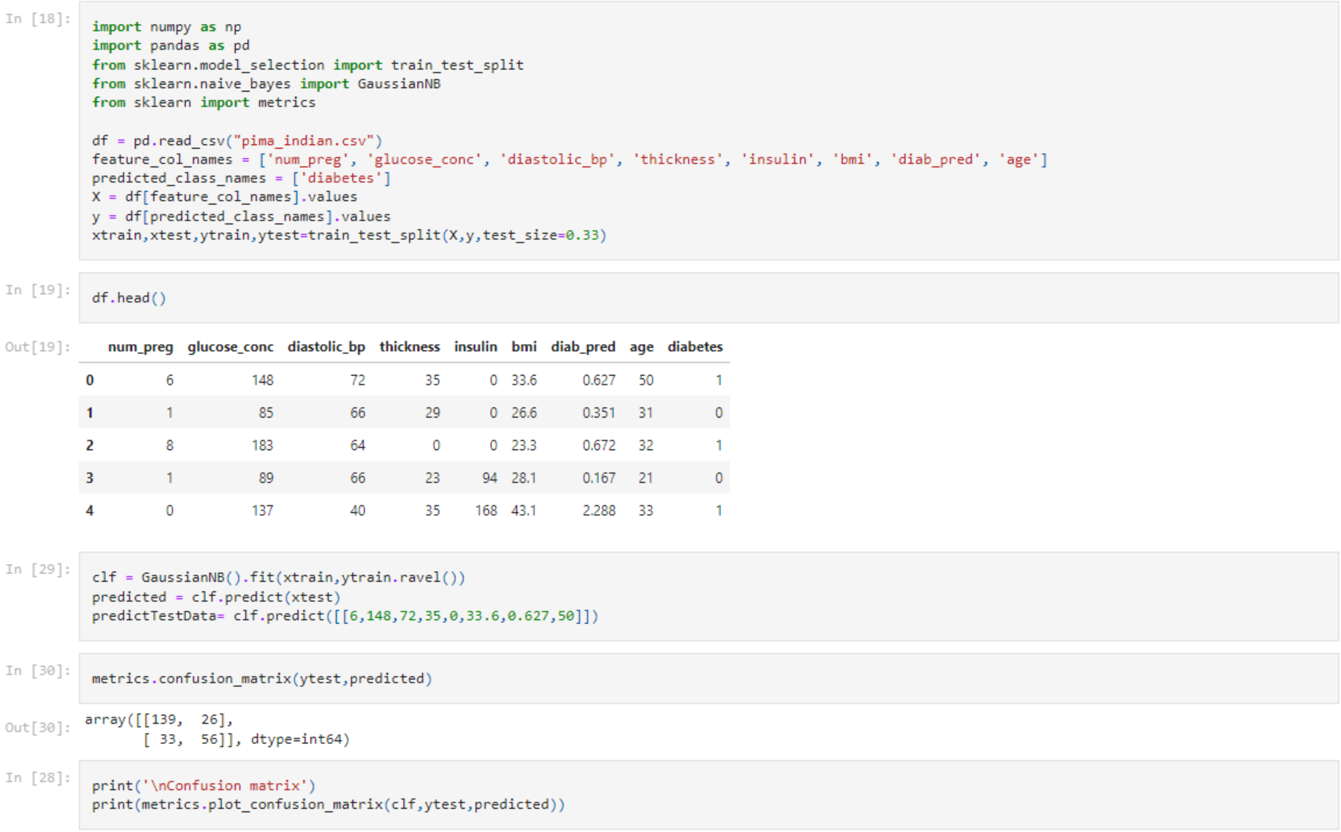
In [5]:

```
class NaiveBayesClassifier:
    def __init__(self, X, y):
        self.X, self.y = X, y
        self.N = len(self.X)
        self.dim = len(self.X[0])
        self.attrs = [[] for _ in range(self.dim)]
        self.output_dom = {}
        self.data = []
        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])
            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1
            else:
                self.output_dom[self.y[i]] += 1
            self.data.append([self.X[i], self.y[i]])
    def classify(self, entry):
        solve = None
        max_arg = -1
        for y in self.output_dom.keys():
            prob = self.output_dom[y]/self.N
            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N
            if prob > max_arg:
                max_arg = prob
                solve = y
        return solve
```

In [6]:

```
nbc = NaiveBayesClassifier(X_train, y_train)
total_cases = len(y_val)
good = 0
bad = 0
predictions = []
for i in range(total_cases):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)
    if y_val[i] == predict:
        good += 1
    else:
        bad += 1
print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)
```

```
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```



In [18]:
```
import numpy as np
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
y = df[predicted_class_names].values
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)
```

In [19]:
```
df.head()
```

Out[19]:

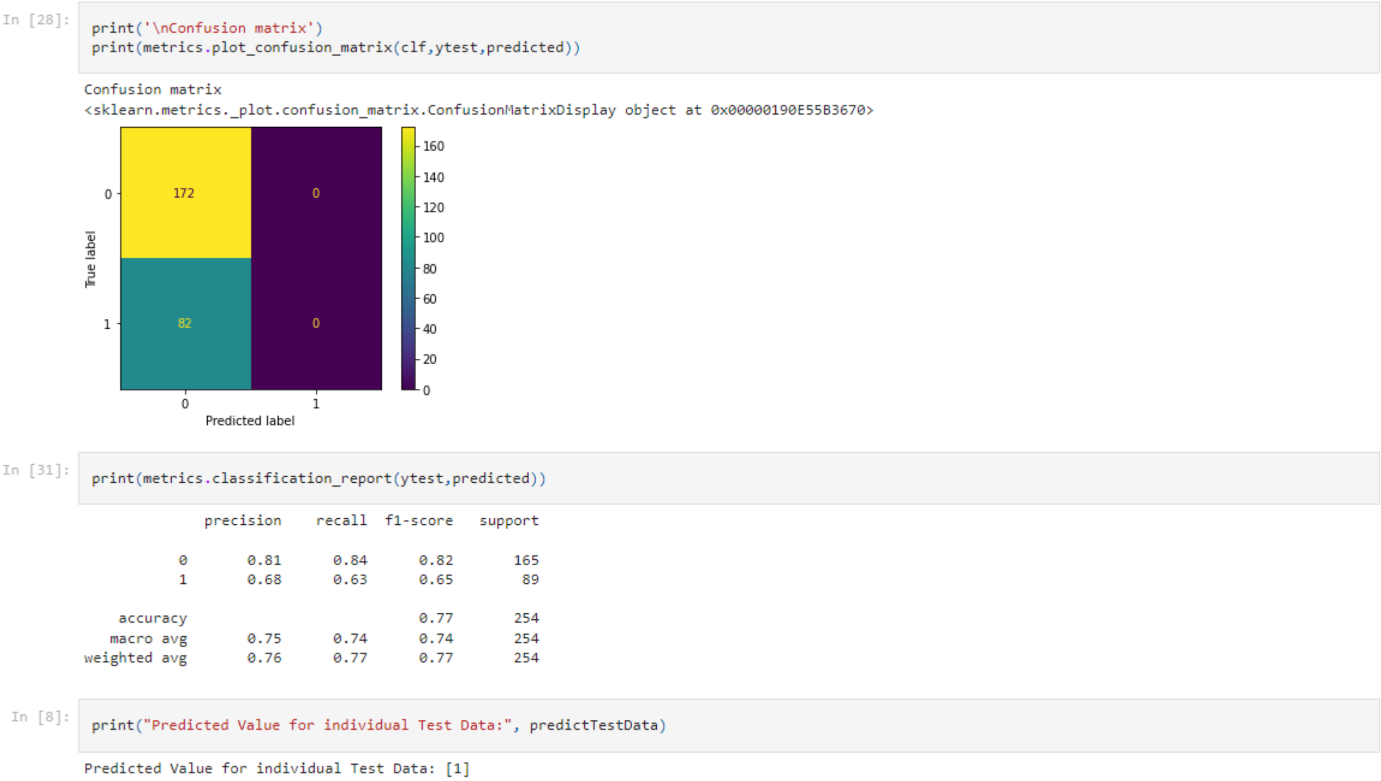
	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [29]:
```
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
```

In [30]:
```
metrics.confusion_matrix(ytest,predicted)
```

Out[30]:
```
array([[139,  26],
       [ 33,  56]], dtype=int64)
```

In [28]:
```
print('\nConfusion matrix')
print(metrics.plot_confusion_matrix(clf,ytest,predicted))
```



In [28]:
```
print('\nConfusion matrix')
print(metrics.plot_confusion_matrix(clf,ytest,predicted))
```

```
Confusion matrix
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x00000190E55B3670>
```

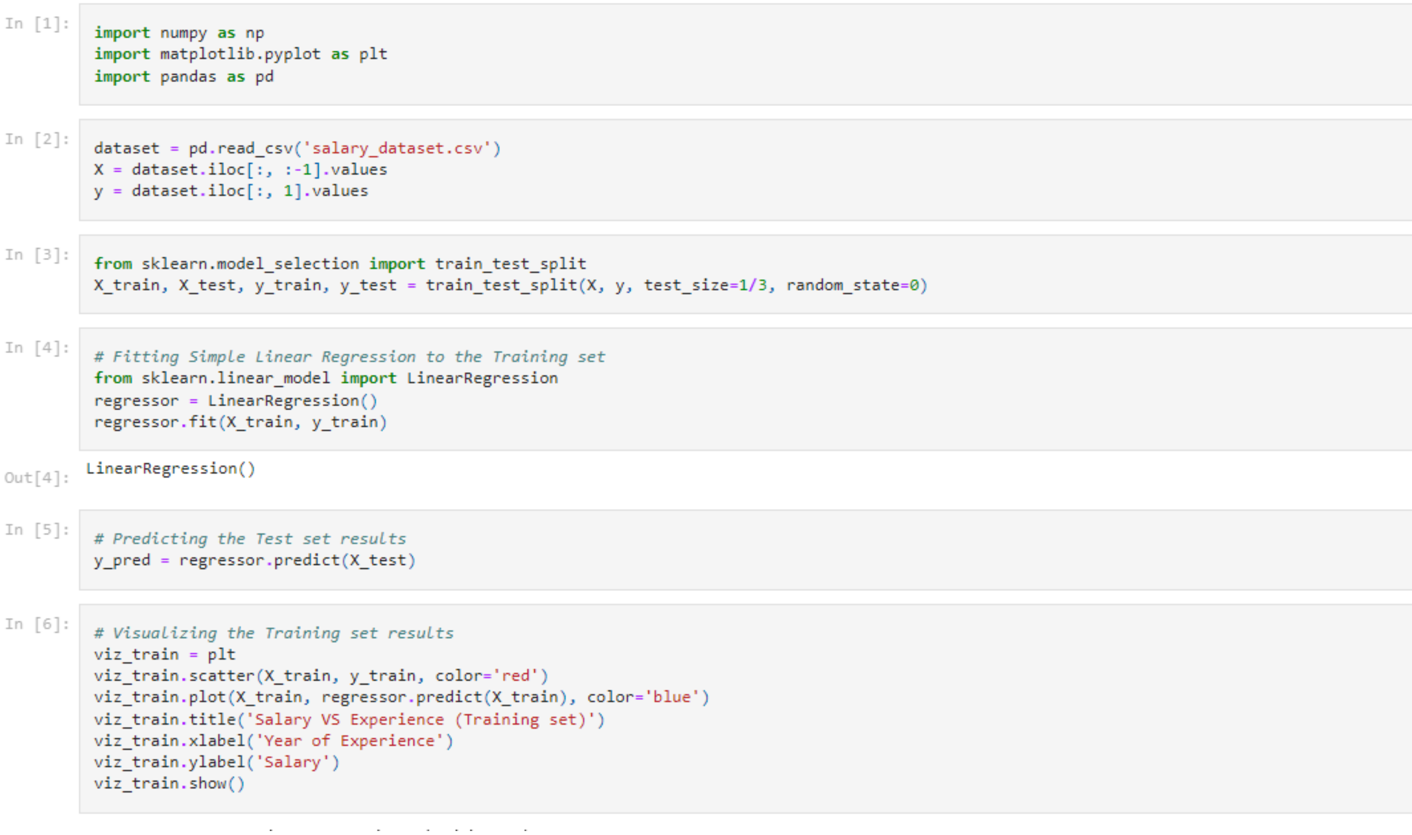
In [31]:
```
print(metrics.classification_report(ytest,predicted))
```

```
              precision    recall  f1-score   support

           0       0.81      0.84      0.82       165
           1       0.68      0.63      0.65        89

    accuracy                           0.77       254
   macro avg       0.75      0.74      0.74       254
weighted avg       0.76      0.77      0.77       254
```

In [8]:
```
print("Predicted Value for individual Test Data:", predictTestData)
```

```
Predicted Value for individual Test Data: [1]
```

Lab 5 :



In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
dataset = pd.read_csv('salary_dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

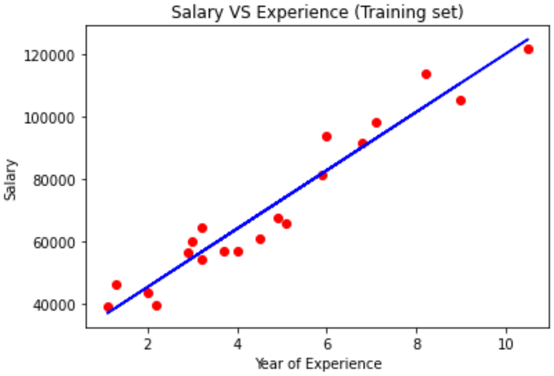
In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
```

In [4]:
```
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[4]:
```
LinearRegression()
```

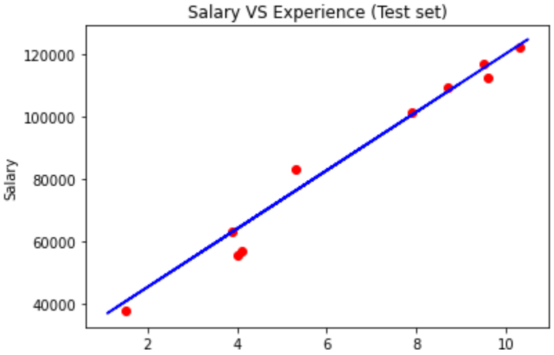
In [5]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
```

In [6]:
```
# 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 [7]:

```
# 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()
```


Salary VS Experience (Test set)
Salary