VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING

Submitted by

MOHAMMED IBRAHIM RAHIL S (1BM19CS090)

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 **Mohammed Ibrahim Rahil S (1BM19CS090),** 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 - (20CS6PCMAL)** work prescribed for the said degree.

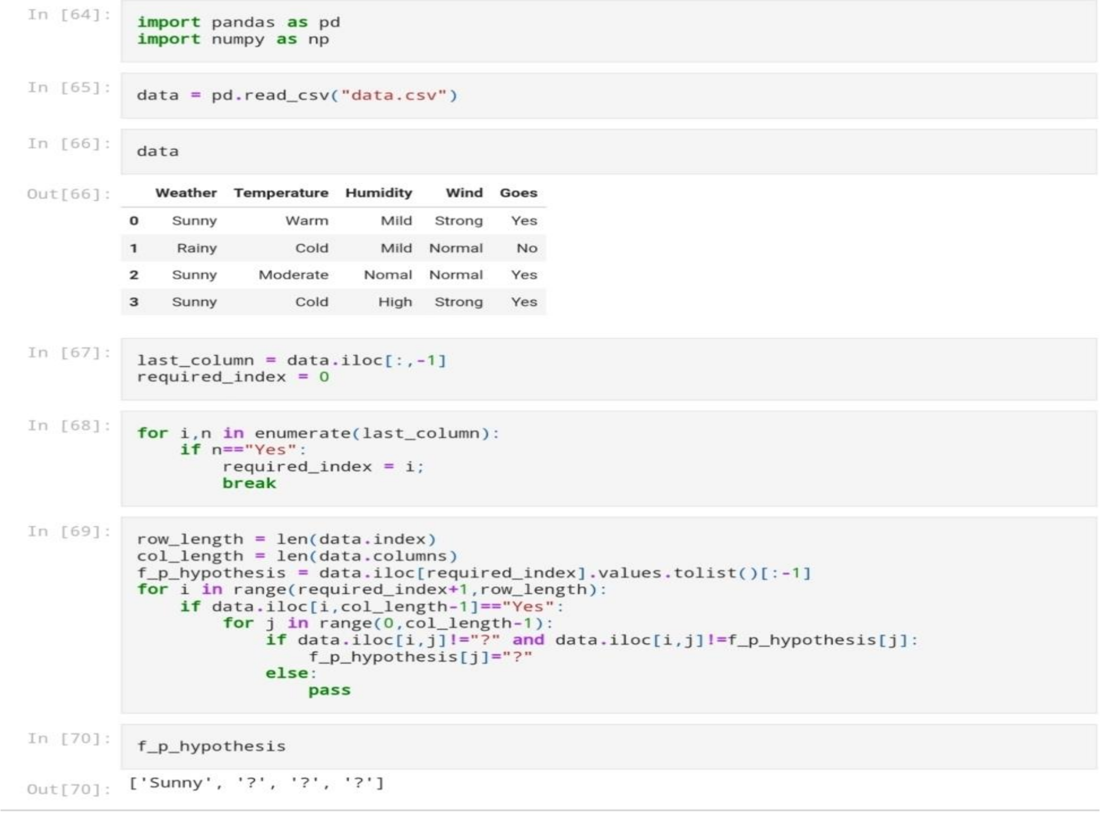
Dr. G.R.Asha	**Dr. Jyothi S Nayak**
Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

Index Sheet

Course Outcome

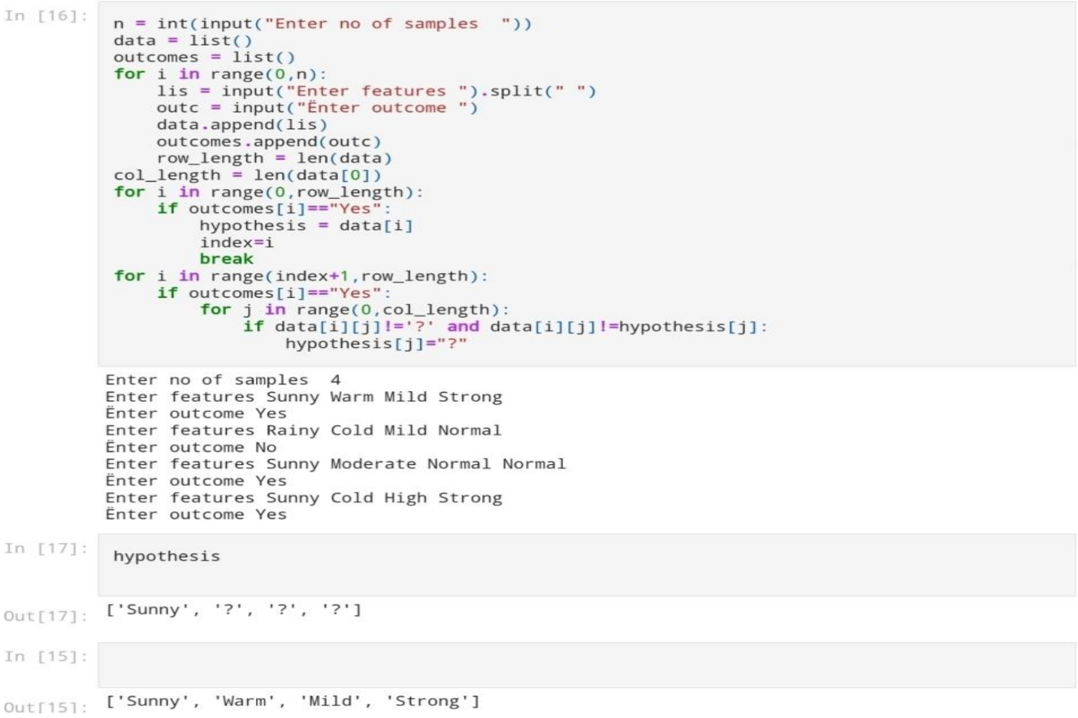
1. **Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.**



```
In [64]: import pandas as pd
         import numpy as np

In [65]: data = pd.read_csv("data.csv")

In [66]: data

Out[66]:
```

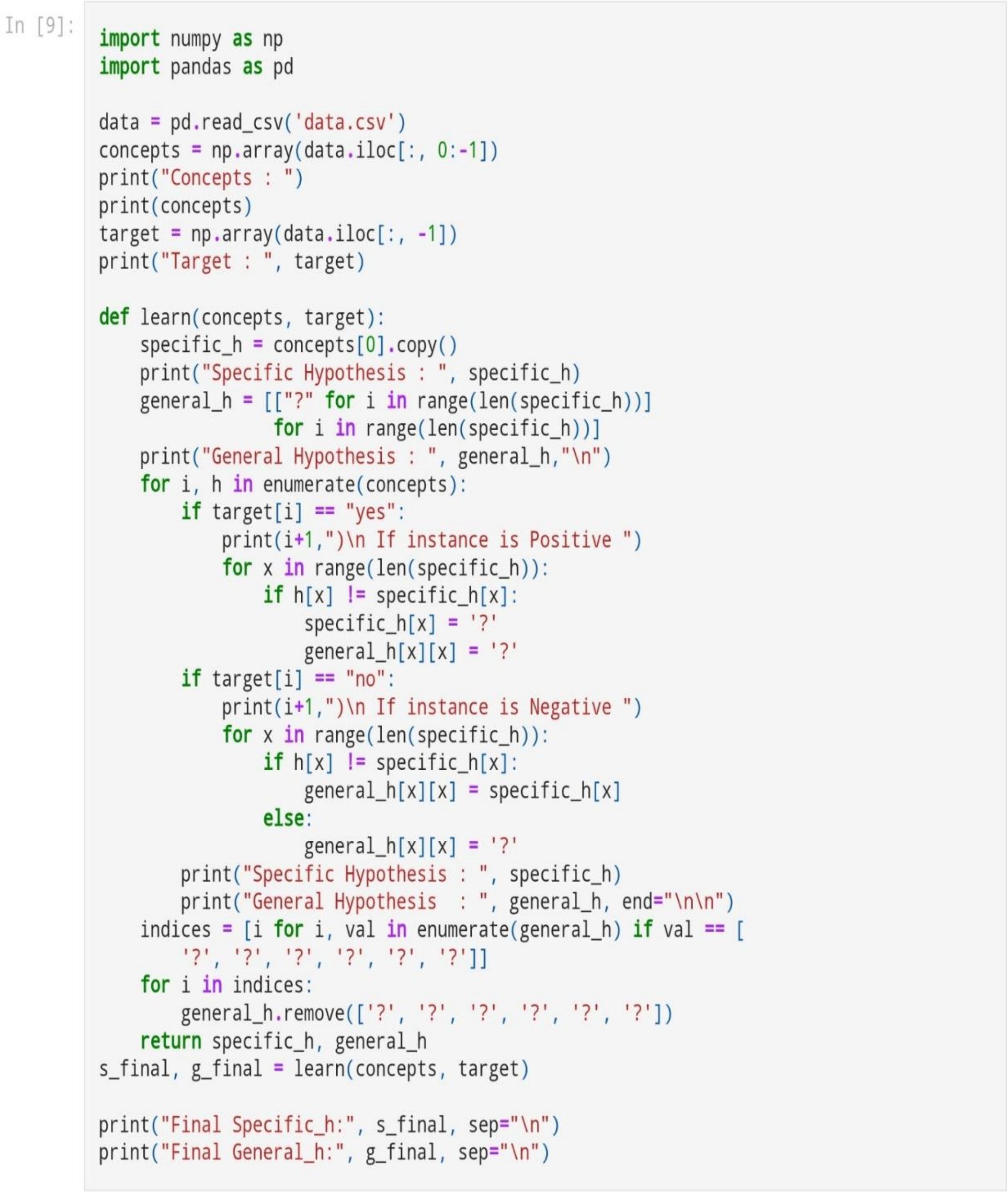
	Weather	Temperature	Humidity	Wind	Goes
0	Sunny	Warm	Mild	Strong	Yes
1	Rainy	Cold	Mild	Normal	No
2	Sunny	Moderate	Nomal	Normal	Yes
3	Sunny	Cold	High	Strong	Yes

```
In [67]: last_column = data.iloc[:,-1]
         required_index = 0

In [68]: for i,n in enumerate(last_column):
             if n=="Yes":
                 required_index = i;
                 break

In [69]: row_length = len(data.index)
         col_length = len(data.columns)
         f_p_hypothesis = data.iloc[required_index].values.tolist()[:-1]
         for i in range(required_index+1,row_length):
             if data.iloc[i,col_length-1]=="Yes":
                 for j in range(0,col_length-1):
                     if data.iloc[i,j]!="?" and data.iloc[i,j]!=f_p_hypothesis[j]:
                         f_p_hypothesis[j]="?"
                     else:
                         pass

In [70]: f_p_hypothesis

Out[70]: ['Sunny', '?', '?', '?']
```



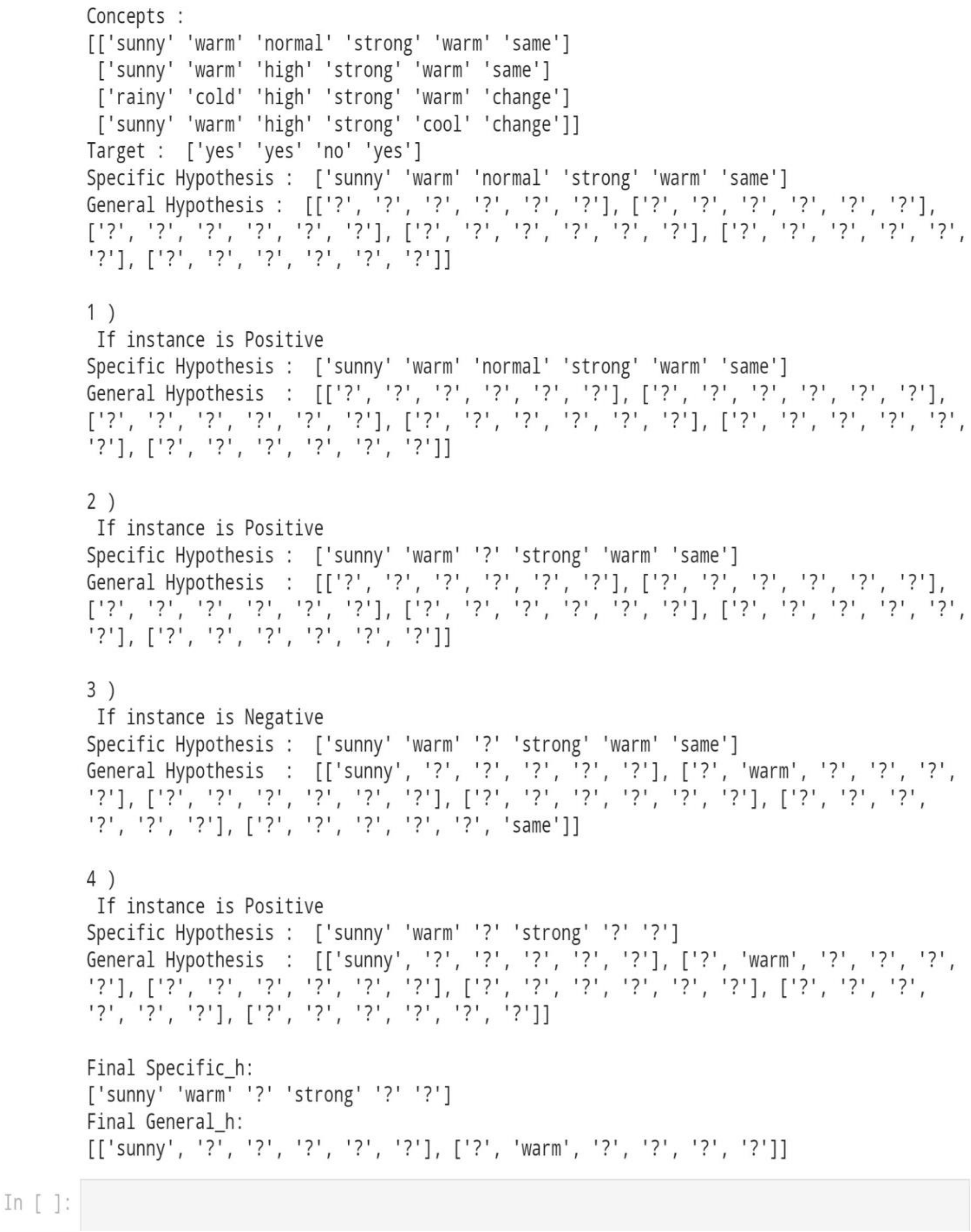
```
In [16]: n = int(input("Enter no of samples  "))
         data = list()
         outcomes = list()
         for i in range(0,n):
             lis = input("Enter features ").split(" ")
             outc = input("Ënter outcome ")
             data.append(lis)
             outcomes.append(outc)
             row_length = len(data)
         col_length = len(data[0])
         for i in range(0,row_length):
             if outcomes[i]=="Yes":
                 hypothesis = data[i]
                 index=i
                 break
         for i in range(index+1,row_length):
             if outcomes[i]=="Yes":
                 for j in range(0,col_length):
                     if data[i][j]!='?' and data[i][j]!=hypothesis[j]:
                         hypothesis[j]="?"

         Enter no of samples  4
         Enter features Sunny Warm Mild Strong
         Ënter outcome Yes
         Enter features Rainy Cold Mild Normal
         Ënter outcome No
         Enter features Sunny Moderate Normal Normal
         Ënter outcome Yes
         Enter features Sunny Cold High Strong
         Ënter outcome Yes

In [17]: hypothesis

Out[17]: ['Sunny', '?', '?', '?']

In [15]:

Out[15]: ['Sunny', 'Warm', 'Mild', 'Strong']
```

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.



In [9]:

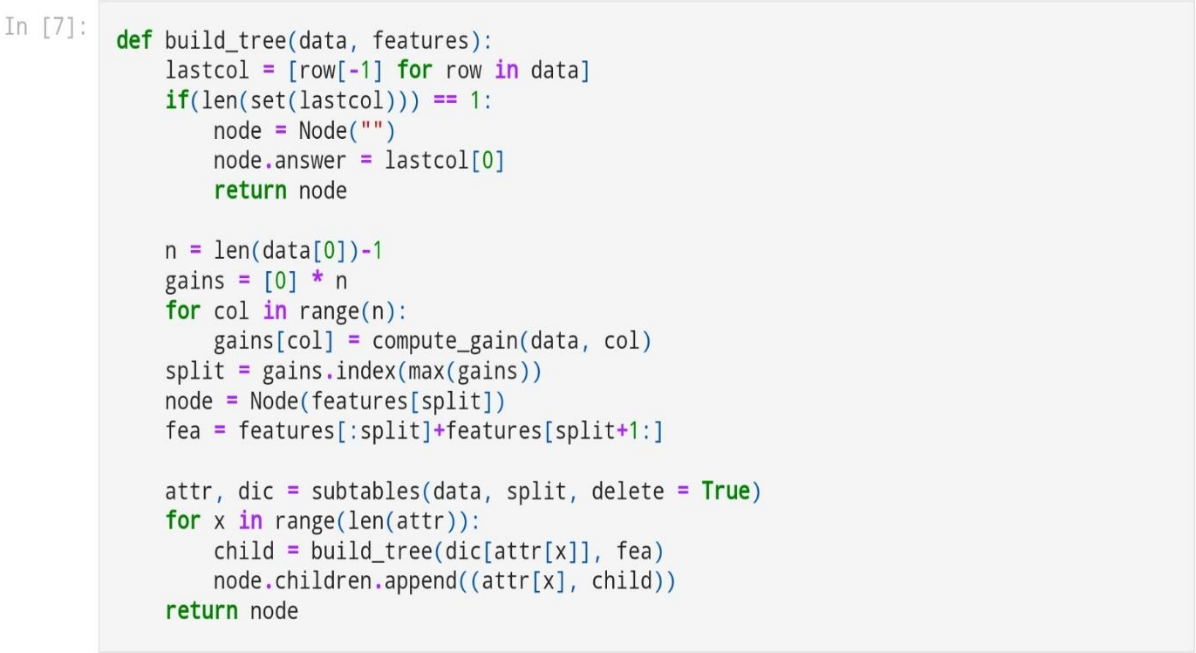
```python
import numpy as np
import pandas as pd

data = pd.read_csv('data.csv')
concepts = np.array(data.iloc[:, 0:-1])
print("Concepts : ")
print(concepts)
target = np.array(data.iloc[:, -1])
print("Target : ", target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("Specific Hypothesis : ", specific_h)
    general_h = [["?" for i in range(len(specific_h))]
                 for i in range(len(specific_h))]
    print("General Hypothesis : ", general_h,"\n")
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            print(i+1,")\n If 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(i+1,")\n If 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 Hypothesis : ", specific_h)
        print("General Hypothesis  : ", general_h, end="\n\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")
```



```
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']]
Target :  ['yes' 'yes' 'no' 'yes']
Specific Hypothesis :  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
General Hypothesis :  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?']]

1 )
 If instance is Positive
Specific Hypothesis :  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
General Hypothesis  :  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?']]

2 )
 If instance is Positive
Specific Hypothesis :  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
General Hypothesis  :  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?']]

3 )
 If instance is Negative
Specific Hypothesis :  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
General Hypothesis  :  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

4 )
 If instance is Positive
Specific Hypothesis :  ['sunny' 'warm' '?' 'strong' '?' '?']
General Hypothesis  :  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

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

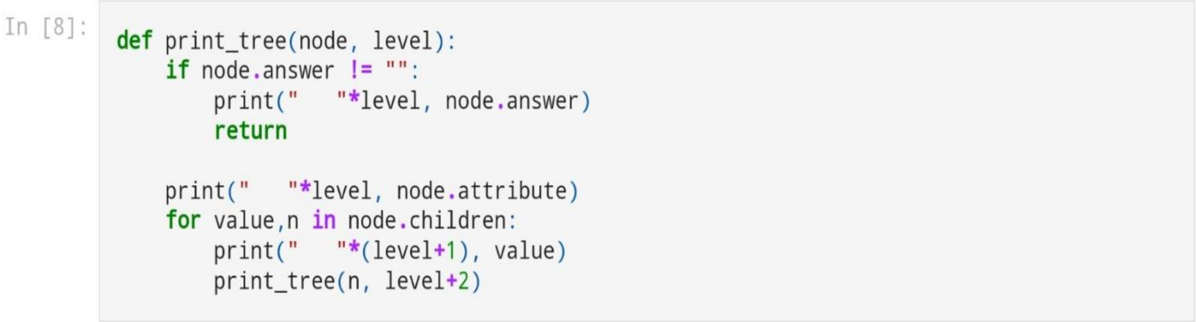
```
In [ ]:
```

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.



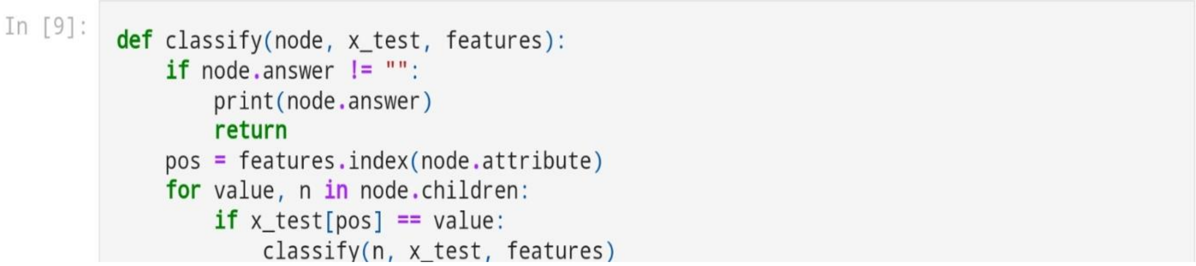
In [7]:

```
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
```



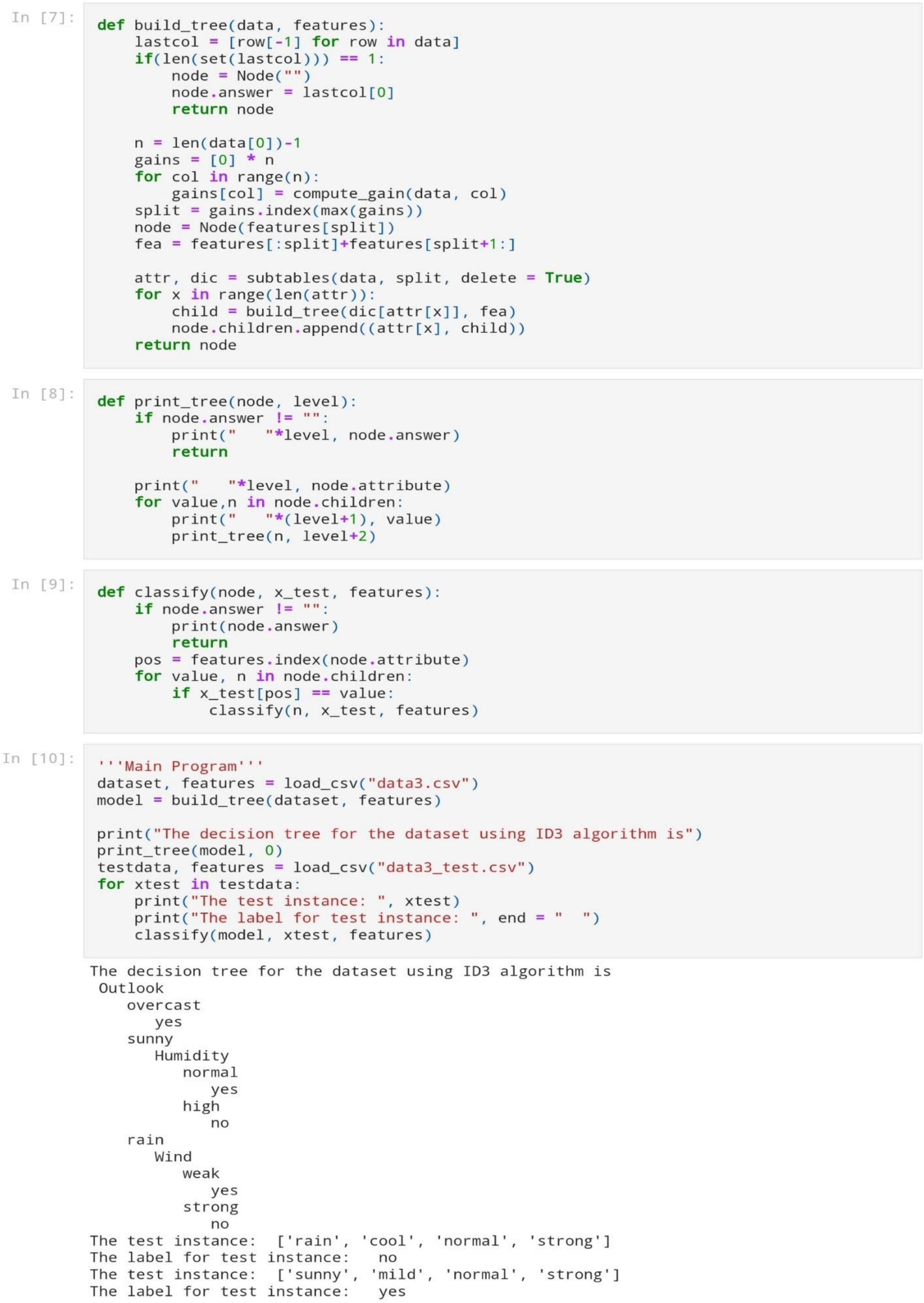
In [8]:

```
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 [9]:

```
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 [7]:
```
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
```

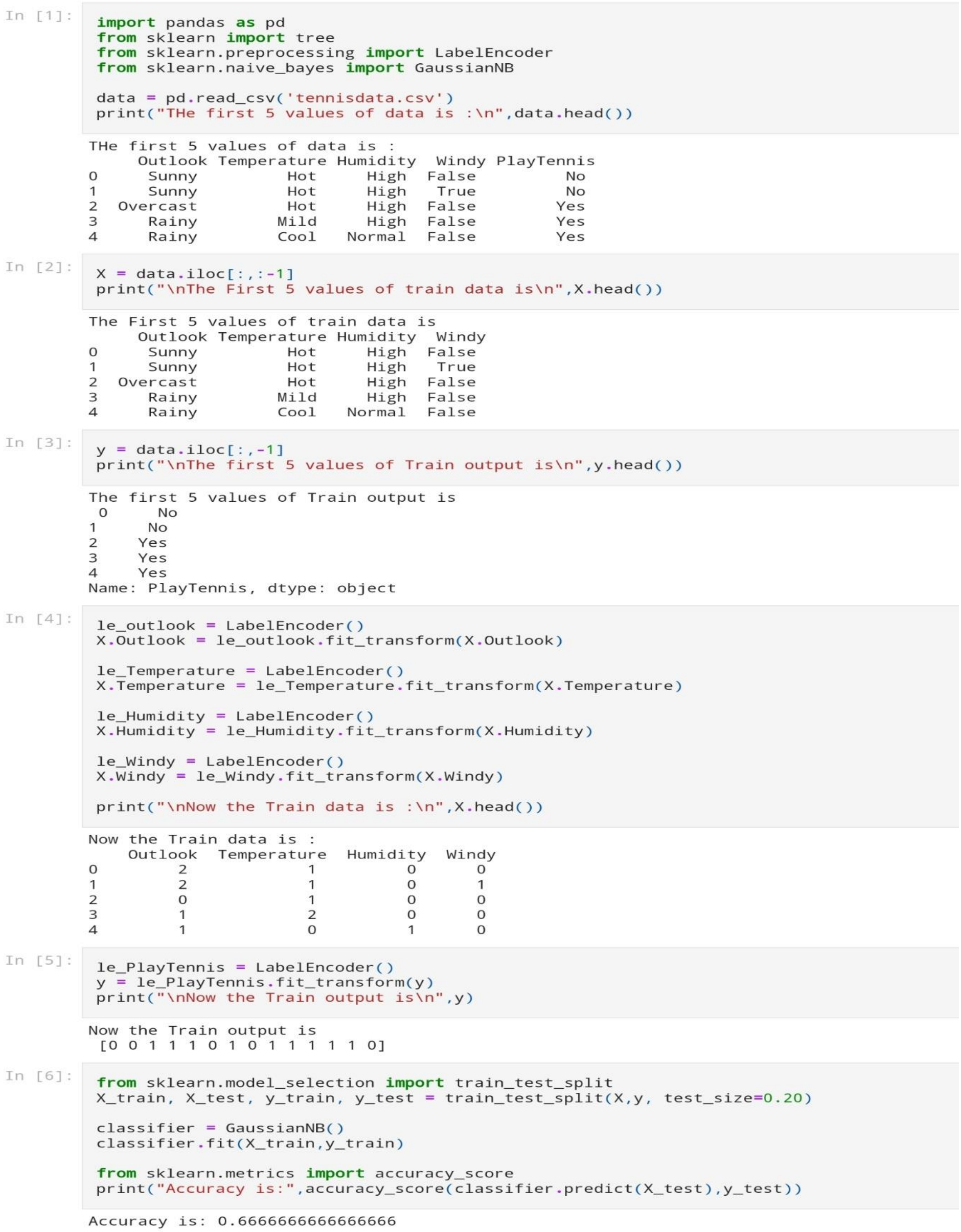
In [8]:
```
def print_tree(node, level):
    if node.answer != "":
        print("   "*level, node.answer)
        return

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

In [9]:
```
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 [10]:
```
'''Main Program'''
dataset, features = load_csv("data3.csv")
model = build_tree(dataset, features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(model, 0)
testdata, features = load_csv("data3_test.csv")
for xtest in testdata:
    print("The test instance: ", xtest)
    print("The label for test instance: ", end = "  ")
    classify(model, xtest, features)
```

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
    overcast
       yes
    sunny
       Humidity
          normal
             yes
          high
             no
    rain
       Wind
          weak
             yes
          strong
             no
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
```

4. 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.



```
In [1]:
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

data = pd.read_csv('tennisdata.csv')
print("THe first 5 values of data is :\n",data.head())
```

```
THe first 5 values of data is :
      Outlook Temperature Humidity  Windy PlayTennis
0      Sunny         Hot     High  False         No
1      Sunny         Hot     High   True         No
2   Overcast         Hot     High  False        Yes
3      Rainy        Mild     High  False        Yes
4      Rainy        Cool   Normal  False        Yes
```

```
In [2]:
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is\n",X.head())
```

```
The First 5 values of train data is
      Outlook Temperature Humidity  Windy
0      Sunny         Hot     High  False
1      Sunny         Hot     High   True
2   Overcast         Hot     High  False
3      Rainy        Mild     High  False
4      Rainy        Cool   Normal  False
```

```
In [3]:
y = data.iloc[:,-1]
print("\nThe first 5 values of Train output is\n",y.head())
```

```
The first 5 values of Train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object
```

```
In [4]:
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)

print("\nNow the Train data is :\n",X.head())
```

```
Now the Train data is :
    Outlook  Temperature  Humidity  Windy
0        2            1         0      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0
```

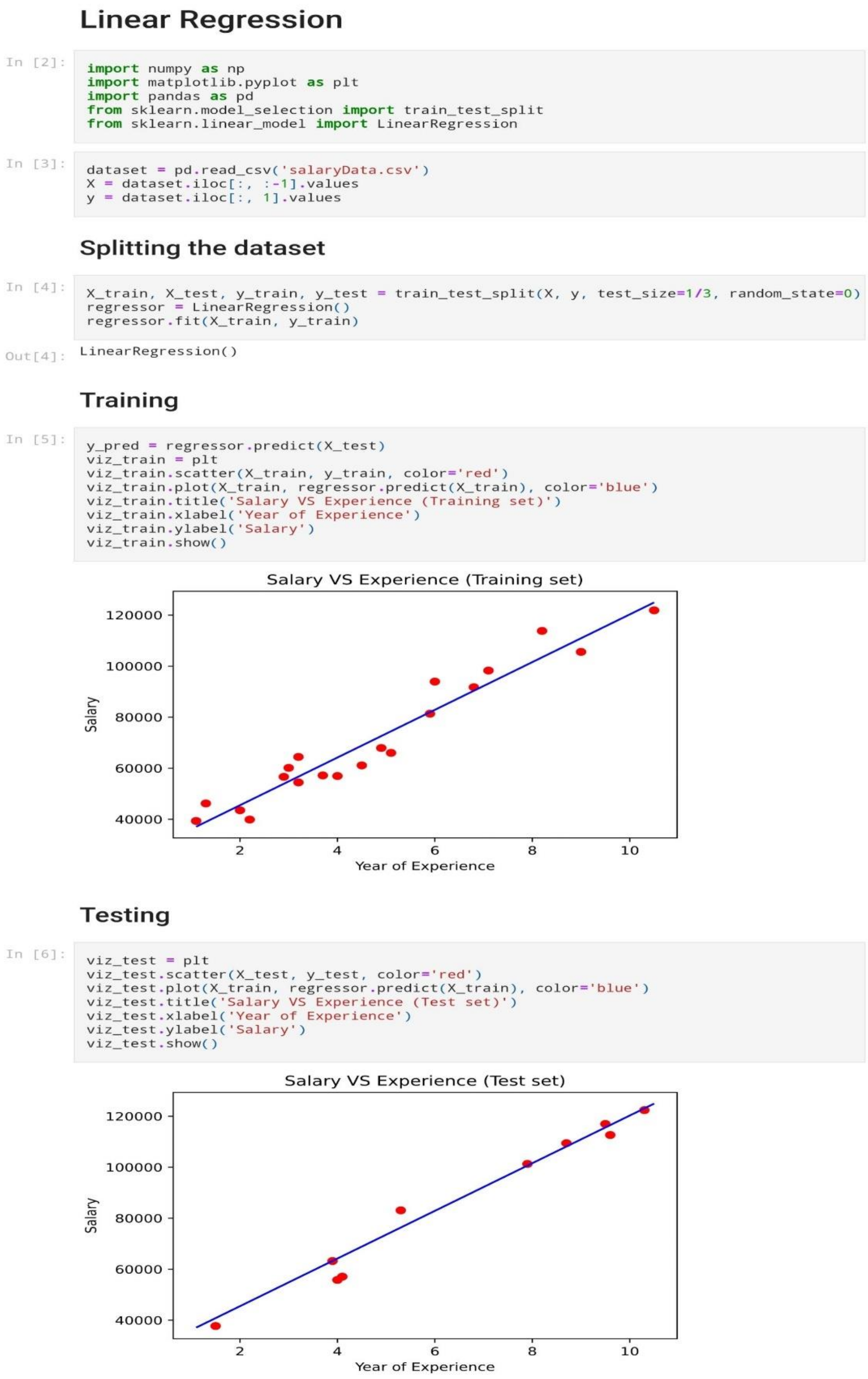
```
In [5]:
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
```

```
Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
In [6]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)

classifier = GaussianNB()
classifier.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

```
Accuracy is: 0.6666666666666666
```

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



Linear Regression

```
In [2]:  import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LinearRegression
```

```
In [3]:  dataset = pd.read_csv('salaryData.csv')
         X = dataset.iloc[:, :-1].values
         y = dataset.iloc[:, 1].values
```

Splitting the dataset

```
In [4]:  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
         regressor = LinearRegression()
         regressor.fit(X_train, y_train)

Out[4]:  LinearRegression()
```

Training

```
In [5]:  y_pred = regressor.predict(X_test)
         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()
```

Testing

```
In [6]:  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()
```