VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING
(20CS6PCMAL)

Submitted by

PUNEETH K (1BM19CS125)

in partial fulfilment 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

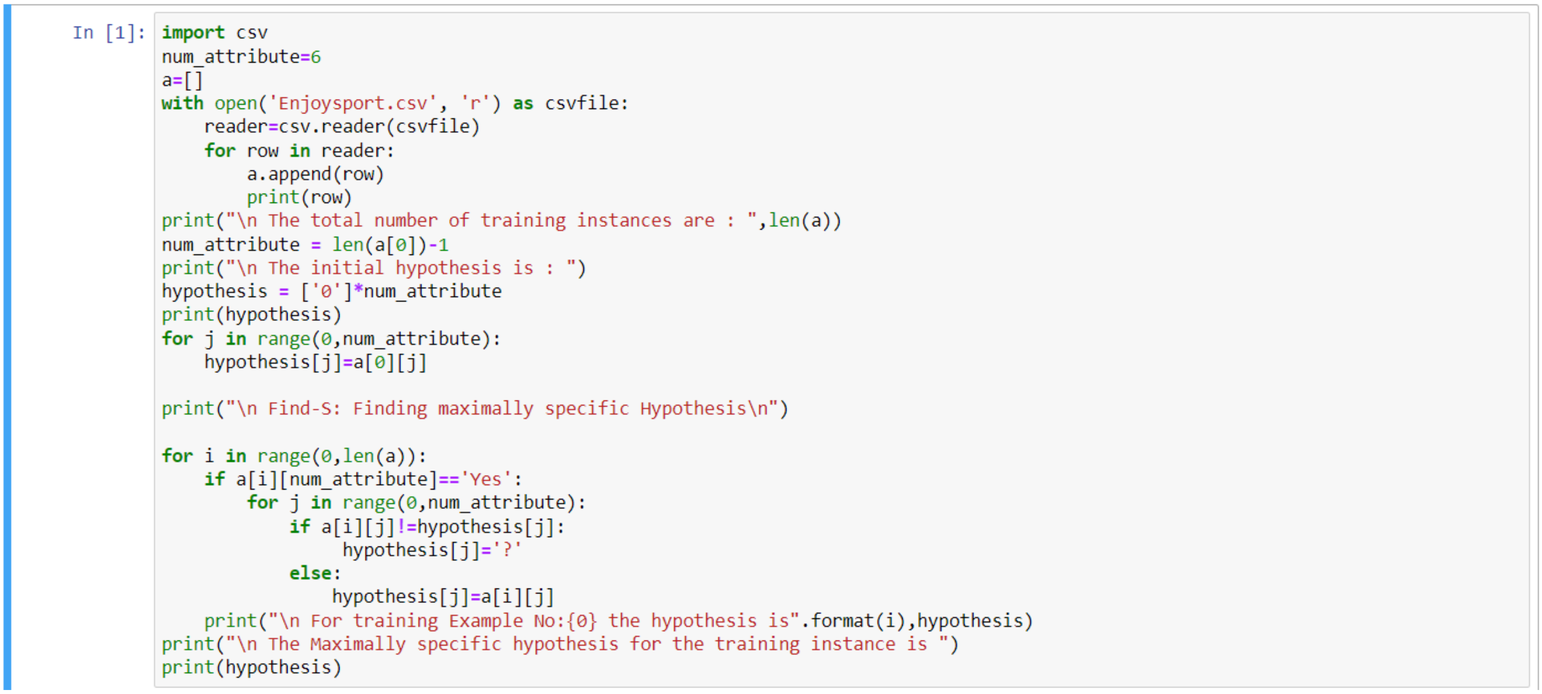
CERTIFICATE

This is to certify that the Lab work entitled "**MACHINE LEARNING**" carried out by **PUNEETH K(1BM19CS125),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment 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.

Saritha A N	
Assistant Professor	
Name of the Lab-In charge	**Dr. Jyothi S Nayak**
Designation	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

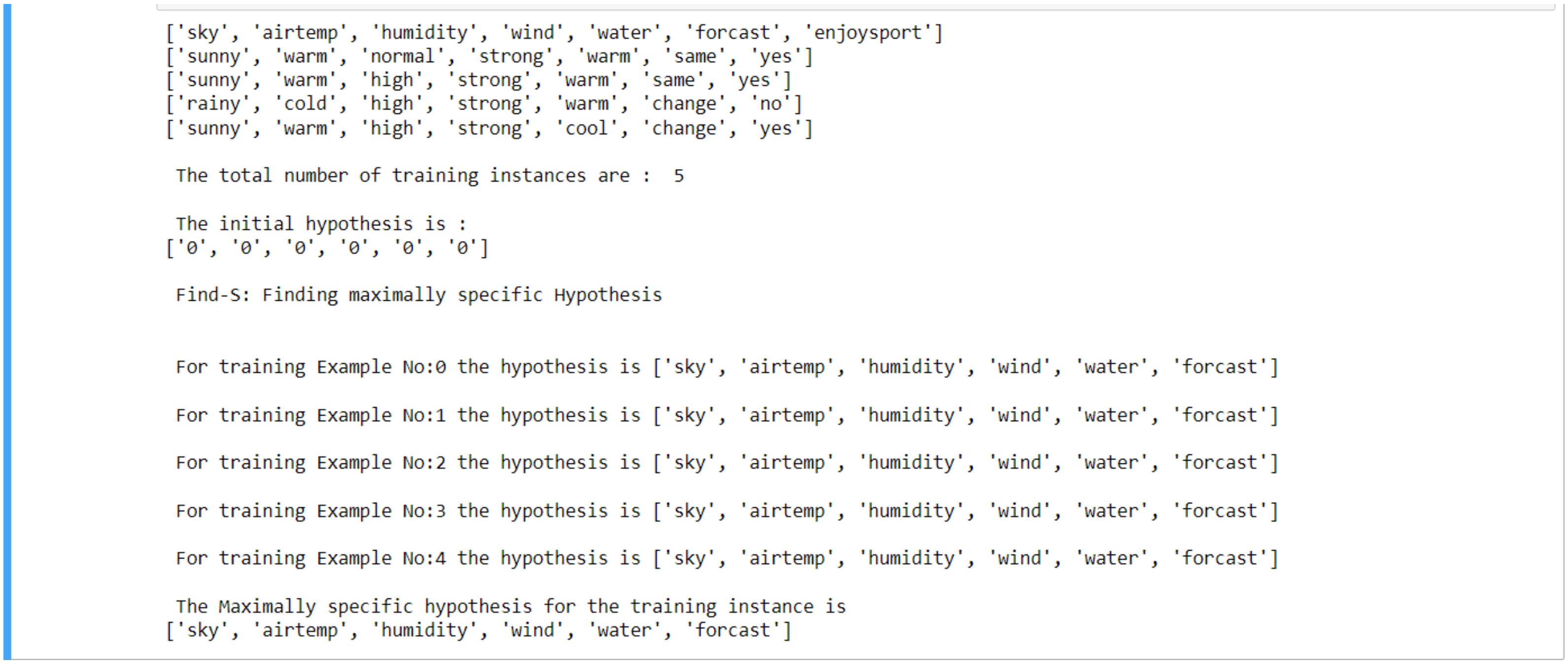
LAB 1

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



In [1]:

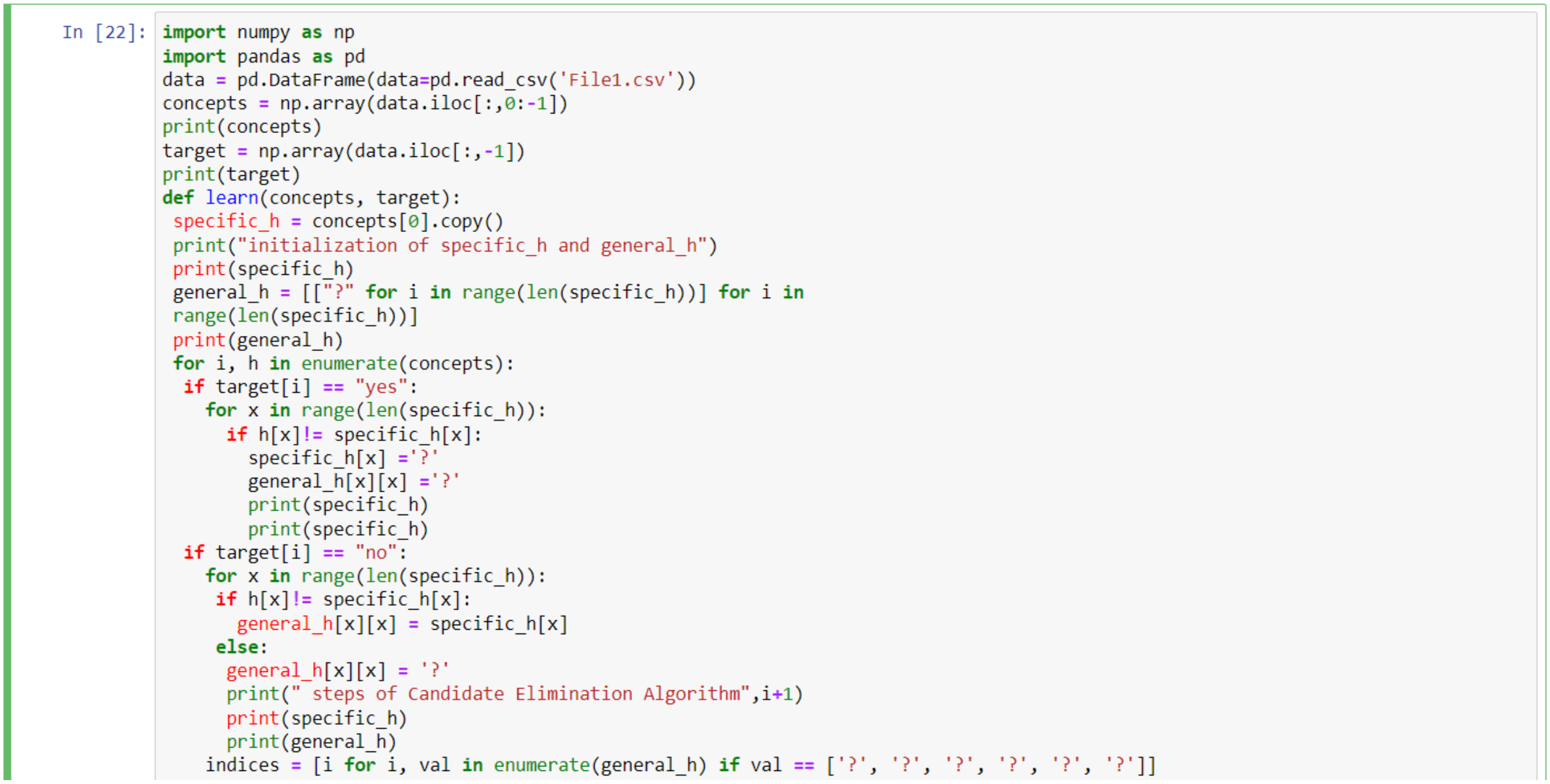
```python
import csv
num_attribute=6
a=[]
with open('Enjoysport.csv', 'r') as csvfile:
    reader=csv.reader(csvfile)
    for row in reader:
        a.append(row)
        print(row)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for j in range(0,num_attribute):
    hypothesis[j]=a[0][j]

print("\n Find-S: Finding maximally specific Hypothesis\n")

for i in range(0,len(a)):
    if a[i][num_attribute]=='Yes':
        for j in range(0,num_attribute):
            if a[i][j]!=hypothesis[j]:
                hypothesis[j]='?'
            else:
               hypothesis[j]=a[i][j]
    print("\n For training Example No:{0} the hypothesis is".format(i),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```



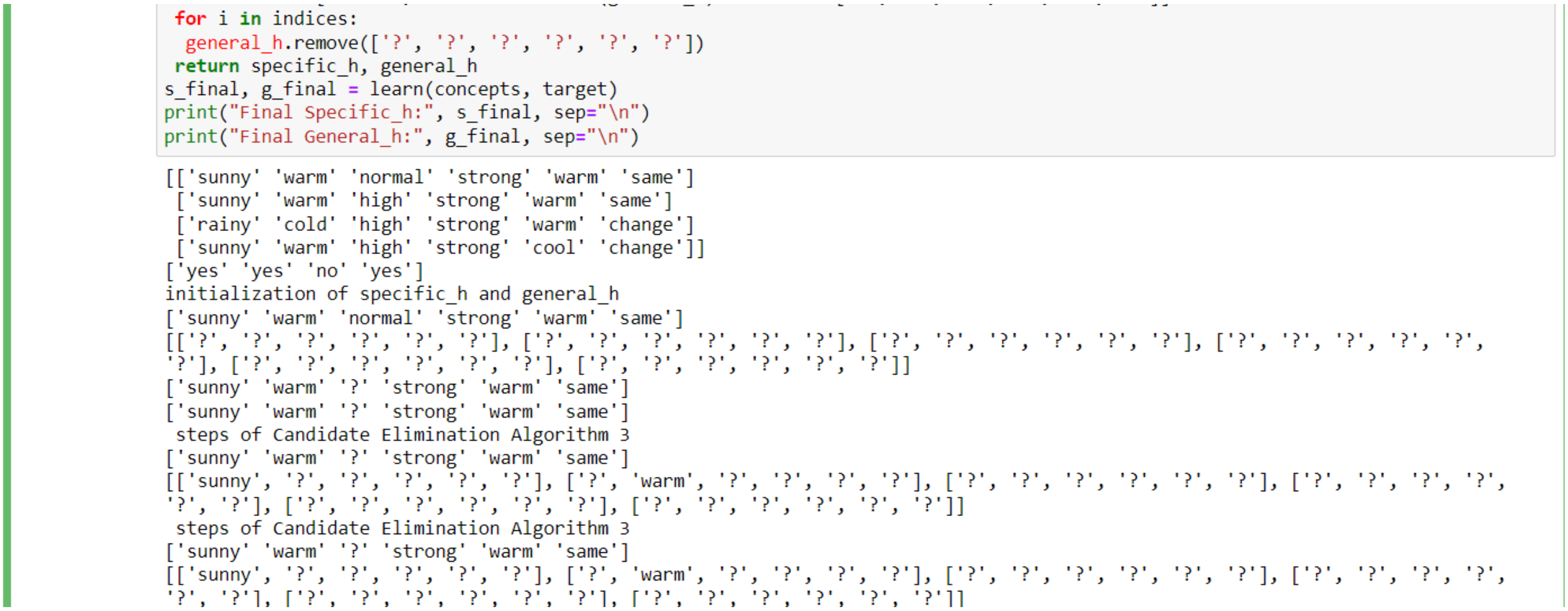
```
['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast', 'enjoysport']
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']

 The total number of training instances are :  5

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

 Find-S: Finding maximally specific Hypothesis


 For training Example No:0 the hypothesis is ['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast']

 For training Example No:1 the hypothesis is ['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast']

 For training Example No:2 the hypothesis is ['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast']

 For training Example No:3 the hypothesis is ['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast']

 For training Example No:4 the hypothesis is ['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast']

 The Maximally specific hypothesis for the training instance is 
['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast']
```

LAB 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.

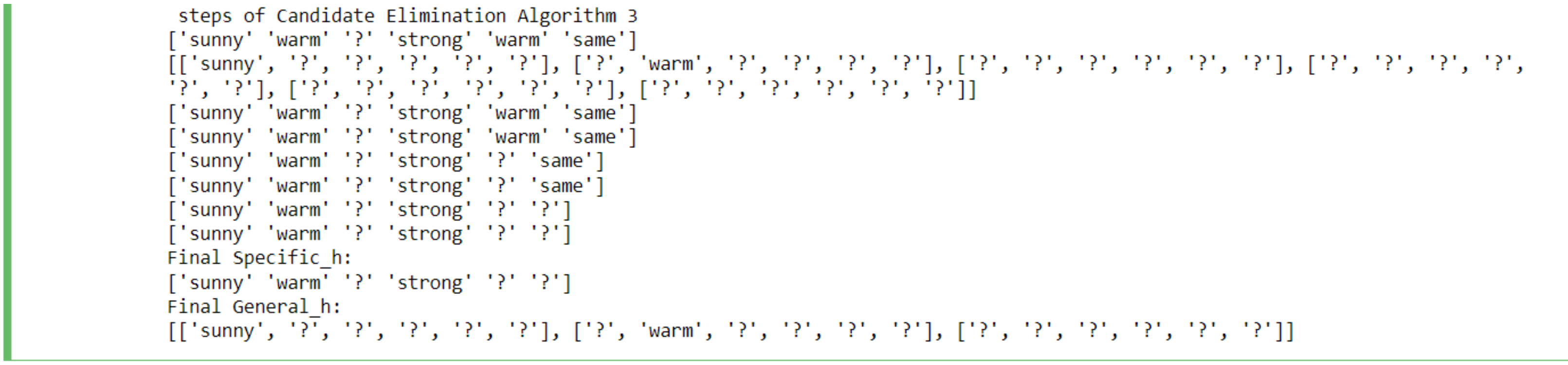


```
In [22]: import numpy as np
         import pandas as pd
         data = pd.DataFrame(data=pd.read_csv('File1.csv'))
         concepts = np.array(data.iloc[:,0:-1])
         print(concepts)
         target = np.array(data.iloc[:,-1])
         print(target)
         def learn(concepts, target):
          specific_h = concepts[0].copy()
          print("initialization of specific_h and general_h")
          print(specific_h)
          general_h = [["?" for i in range(len(specific_h))] for i in
          range(len(specific_h))]
          print(general_h)
          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] ='?'
                 print(specific_h)
                 print(specific_h)
           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(" steps of Candidate Elimination Algorithm",i+1)
               print(specific_h)
               print(general_h)
             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")
```

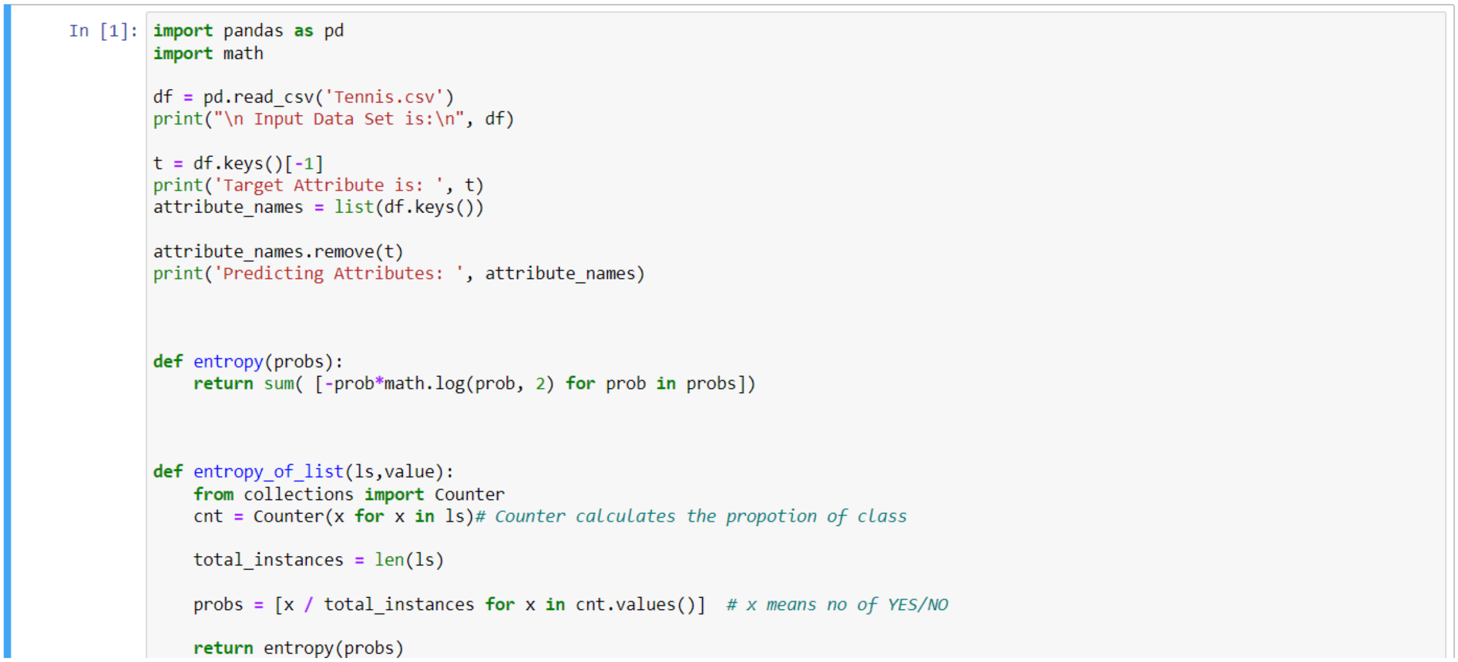
```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```



```
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' 'same']
['sunny' 'warm' '?' 'strong' '?' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
['sunny' 'warm' '?' 'strong' '?' '?']
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

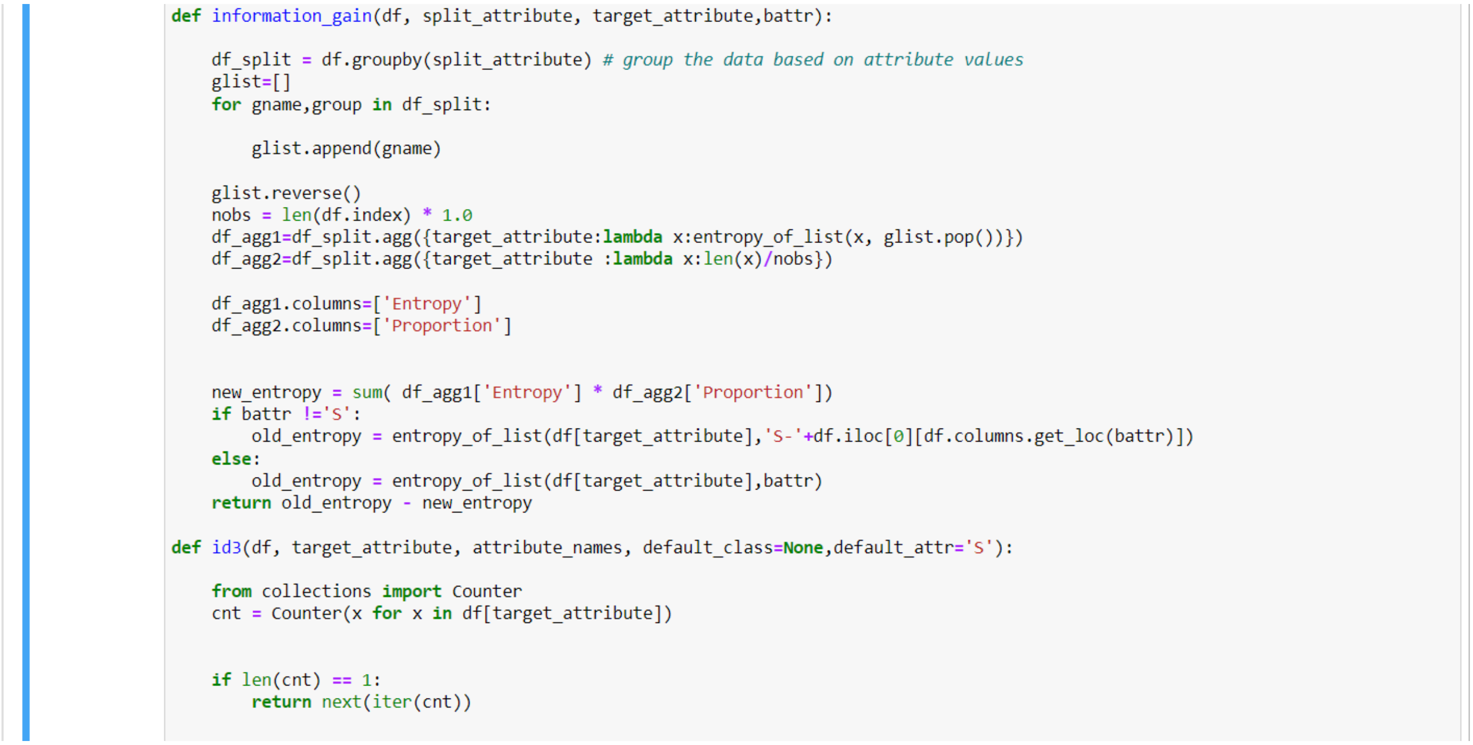
LAB 3

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 [1]:

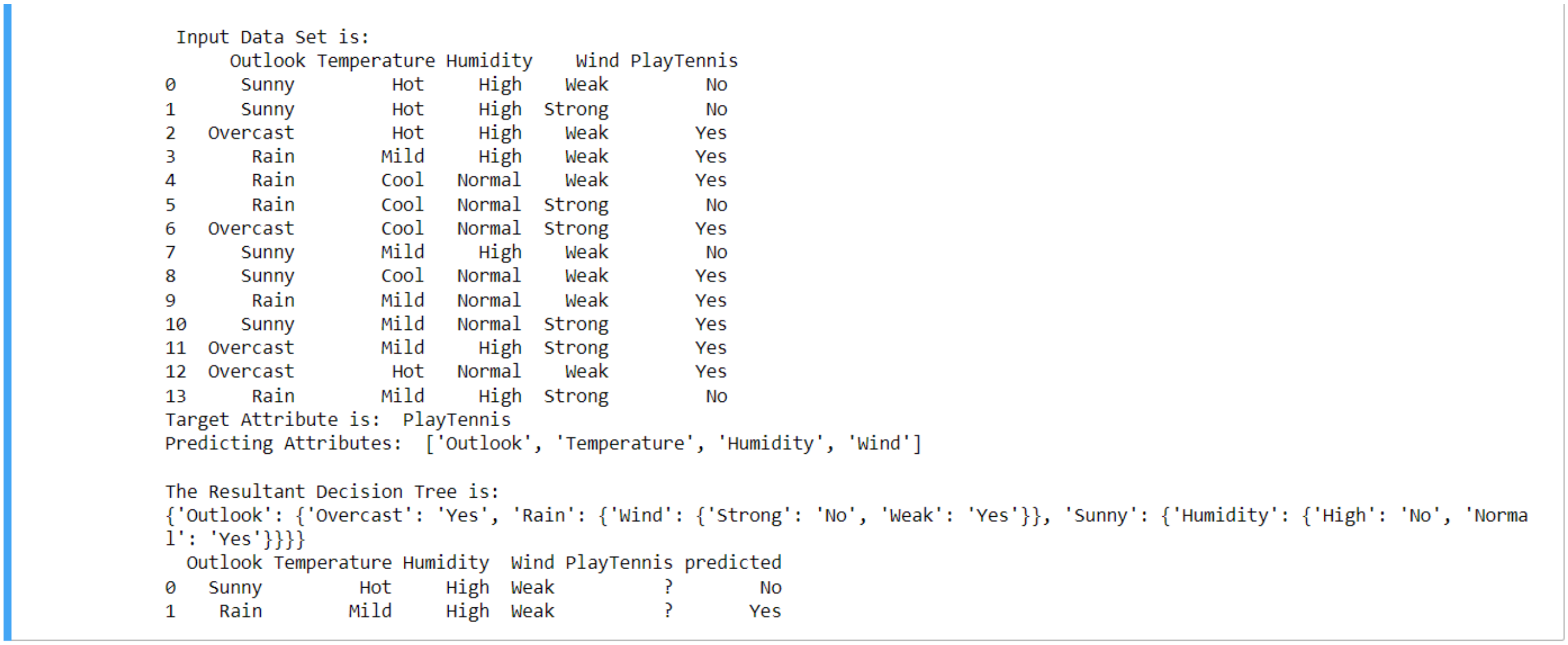
```
import pandas as pd
import math

df = pd.read_csv('Tennis.csv')
print("\n Input Data Set is:\n", df)

t = df.keys()[-1]
print('Target Attribute is: ', t)
attribute_names = list(df.keys())

attribute_names.remove(t)
print('Predicting Attributes: ', attribute_names)



def entropy(probs):
    return sum( [-prob*math.log(prob, 2) for prob in probs])



def entropy_of_list(ls,value):
    from collections import Counter
    cnt = Counter(x for x in ls)# Counter calculates the propotion of class

    total_instances = len(ls)

    probs = [x / total_instances for x in cnt.values()]  # x means no of YES/NO

    return entropy(probs)
```



```
def information_gain(df, split_attribute, target_attribute,battr):

    df_split = df.groupby(split_attribute) # group the data based on attribute values
    glist=[]
    for gname,group in df_split:

        glist.append(gname)

    glist.reverse()
    nobs = len(df.index) * 1.0
    df_agg1=df_split.agg({target_attribute:lambda x:entropy_of_list(x, glist.pop())})
    df_agg2=df_split.agg({target_attribute :lambda x:len(x)/nobs})

    df_agg1.columns=['Entropy']
    df_agg2.columns=['Proportion']


    new_entropy = sum( df_agg1['Entropy'] * df_agg2['Proportion'])
    if battr !='S':
        old_entropy = entropy_of_list(df[target_attribute],'S-'+df.iloc[0][df.columns.get_loc(battr)])
    else:
        old_entropy = entropy_of_list(df[target_attribute],battr)
    return old_entropy - new_entropy

def id3(df, target_attribute, attribute_names, default_class=None,default_attr='S'):

    from collections import Counter
    cnt = Counter(x for x in df[target_attribute])


    if len(cnt) == 1:
        return next(iter(cnt))
```

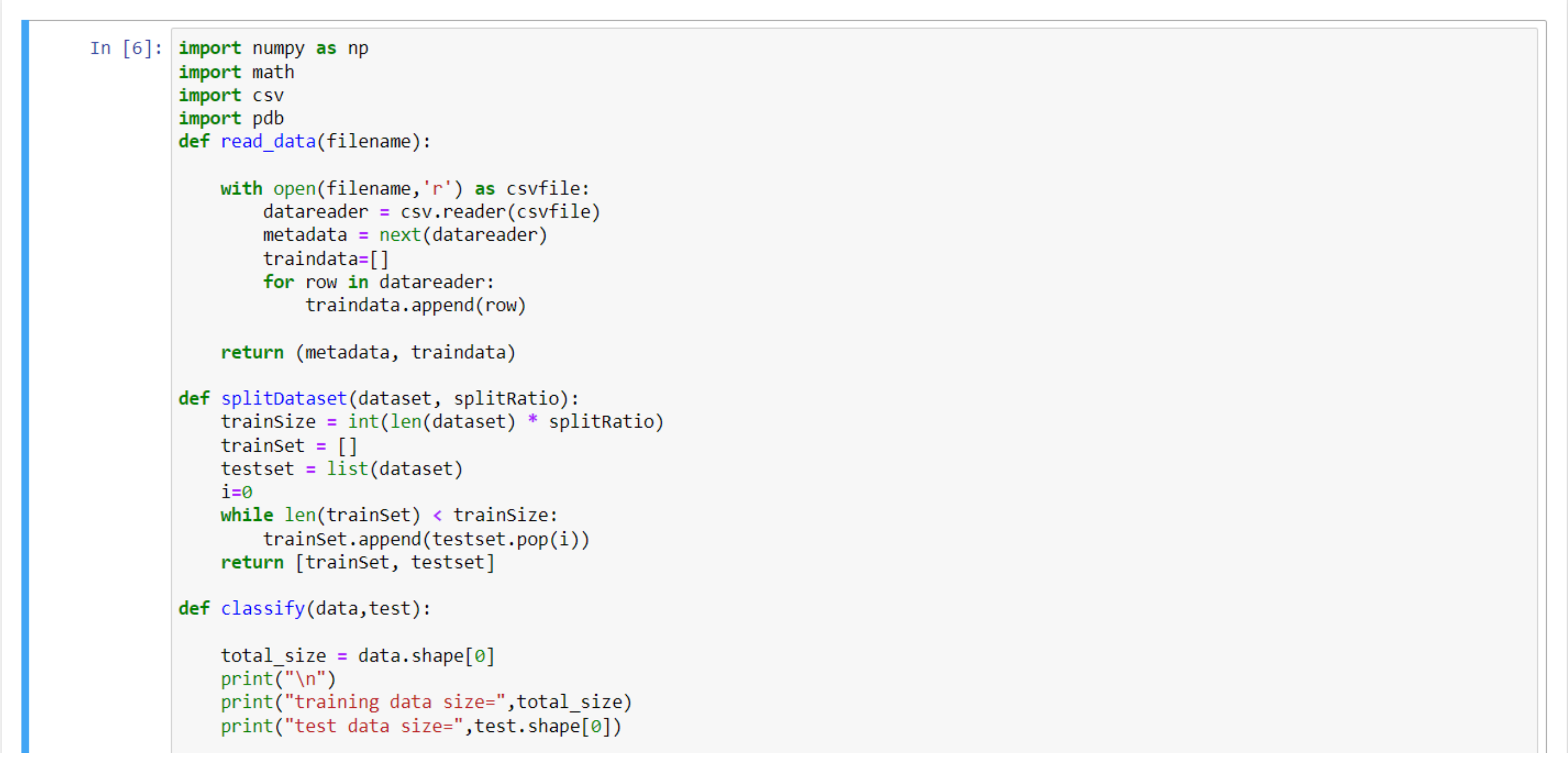
```python
    elif df.empty or (not attribute_names):
        return default_class

    else:

        default_class = max(cnt.keys()) #No of YES and NO Class

        gainz=[]
        for attr in attribute_names:
            ig= information_gain(df, attr, target_attribute,default_attr)
            gainz.append(ig)


        index_of_max = gainz.index(max(gainz))
        best_attr = attribute_names[index_of_max]

        tree = {best_attr:{}}
        remaining_attribute_names =[i for i in attribute_names if i != best_attr]


        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,target_attribute, remaining_attribute_names,default_class,best_attr)
            tree[best_attr][attr_val] = subtree
        return tree

    from pprint import pprint
tree = id3(df,t,attribute_names)
print("\nThe Resultant Decision Tree is:")
print(tree)
```

```python
    from pprint import pprint
tree = id3(df,t,attribute_names)
print("\nThe Resultant Decision Tree is:")
print(tree)

def classify(instance, tree,default=None): # Instance of Play Tennis with Predicted
    attribute = next(iter(tree)) # Outlook/Humidity/Wind
    if instance[attribute] in tree[attribute].keys(): # Value of the attributs in  set of Tree keys
        result = tree[attribute][instance[attribute]]
        if isinstance(result, dict): # this is a tree, delve deeper
            return classify(instance, result)
        else:
            return result # this is a label
    else:
        return default

df_new=pd.read_csv('Tennis_test.csv')
df_new['predicted'] = df_new.apply(classify, axis=1, args=(tree,'?'))
print(df_new)
```

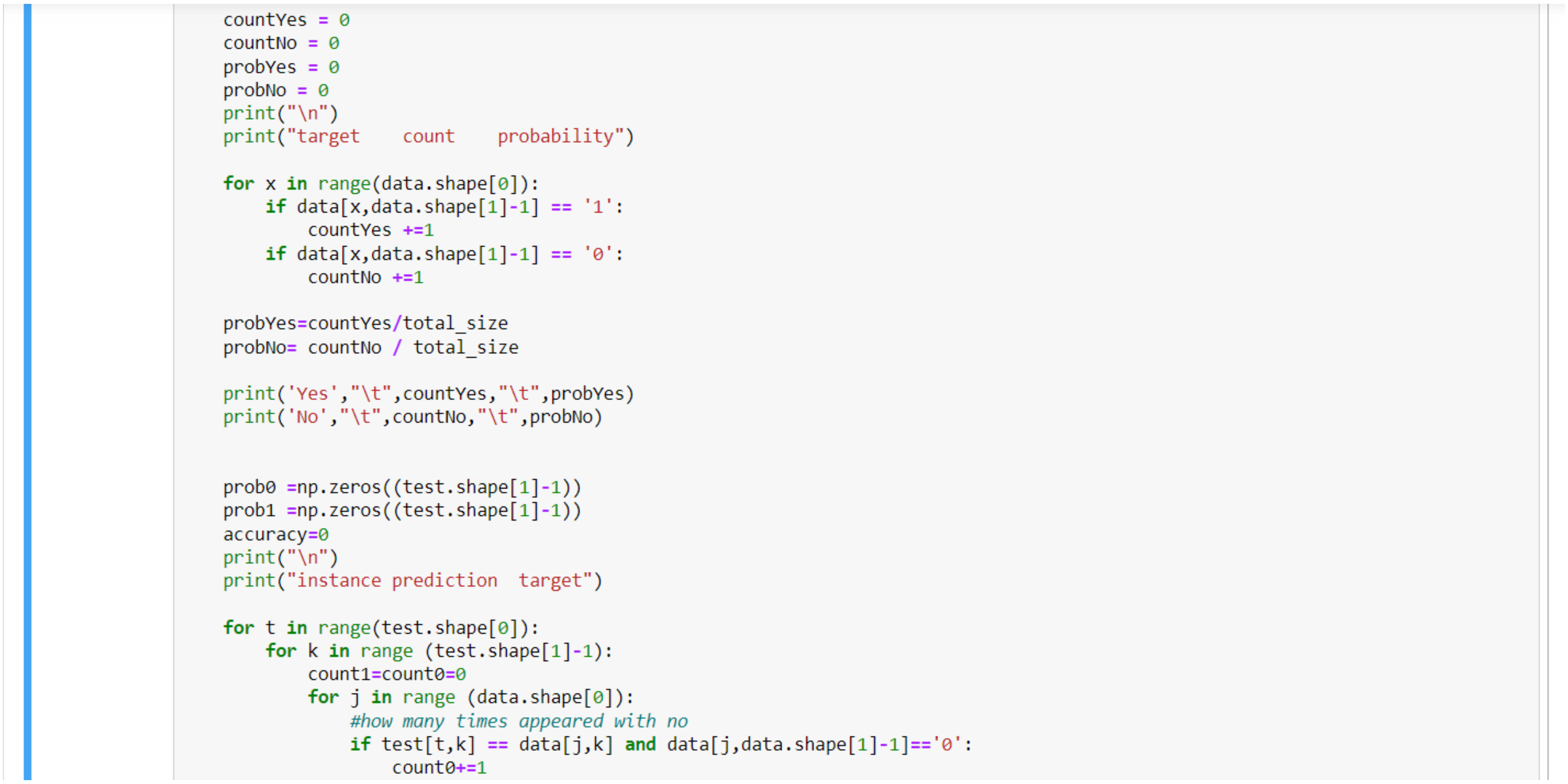


```
 Input Data Set is:
      Outlook Temperature Humidity    Wind PlayTennis
0       Sunny         Hot     High    Weak         No
1       Sunny         Hot     High  Strong         No
2    Overcast         Hot     High    Weak        Yes
3        Rain        Mild     High    Weak        Yes
4        Rain        Cool   Normal    Weak        Yes
5        Rain        Cool   Normal  Strong         No
6    Overcast        Cool   Normal  Strong        Yes
7       Sunny        Mild     High    Weak         No
8       Sunny        Cool   Normal    Weak        Yes
9        Rain        Mild   Normal    Weak        Yes
10      Sunny        Mild   Normal  Strong        Yes
11   Overcast        Mild     High  Strong        Yes
12   Overcast         Hot   Normal    Weak        Yes
13       Rain        Mild     High  Strong         No
Target Attribute is:  PlayTennis
Predicting Attributes:  ['Outlook', 'Temperature', 'Humidity', 'Wind']

The Resultant Decision Tree is:
{'Outlook': {'Overcast': 'Yes', 'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}}, 'Sunny': {'Humidity': {'High': 'No', 'Norma
l': 'Yes'}}}}
  Outlook Temperature Humidity  Wind PlayTennis predicted
0   Sunny         Hot     High  Weak          ?        No
1    Rain        Mild     High  Weak          ?       Yes
```

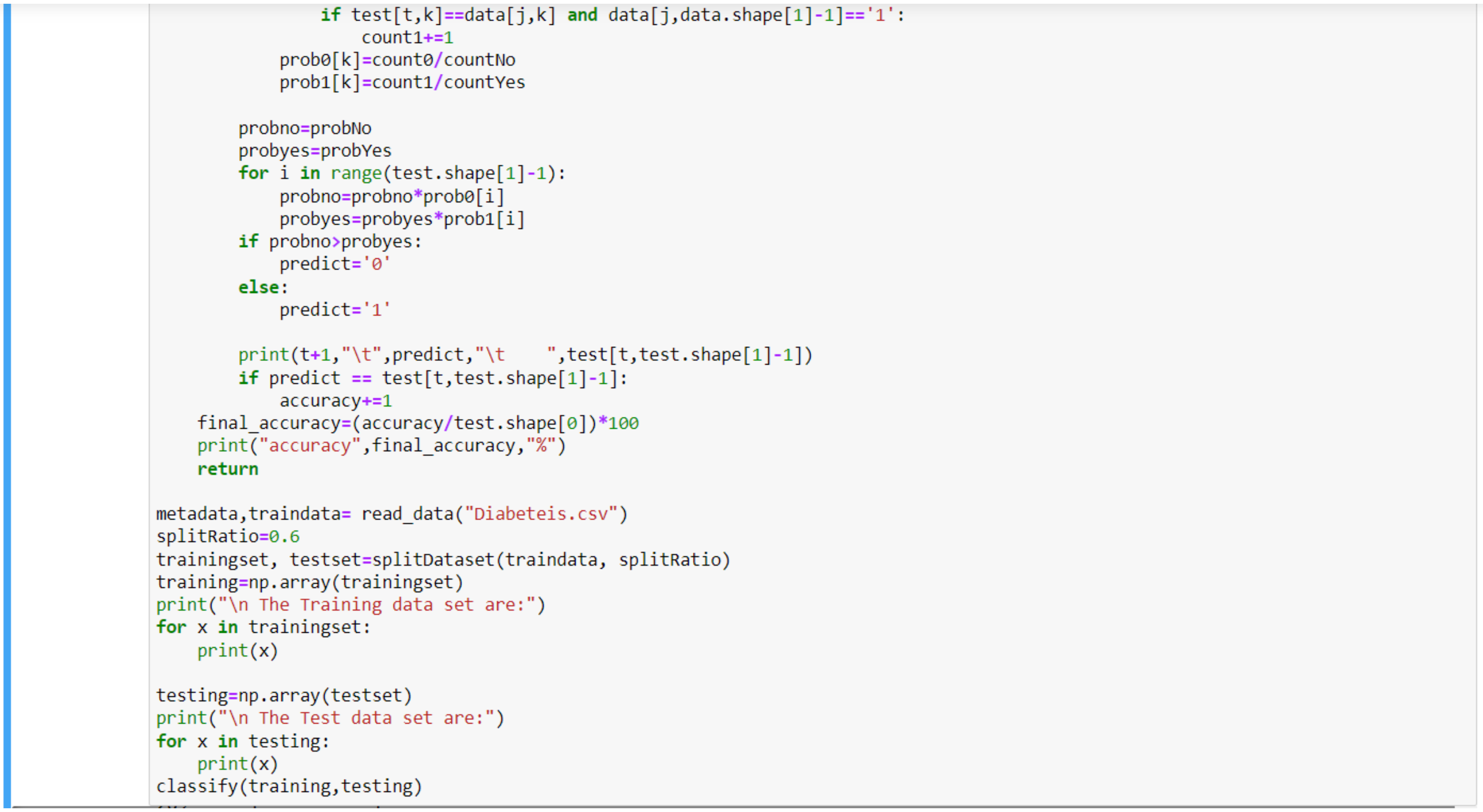
LAB 4



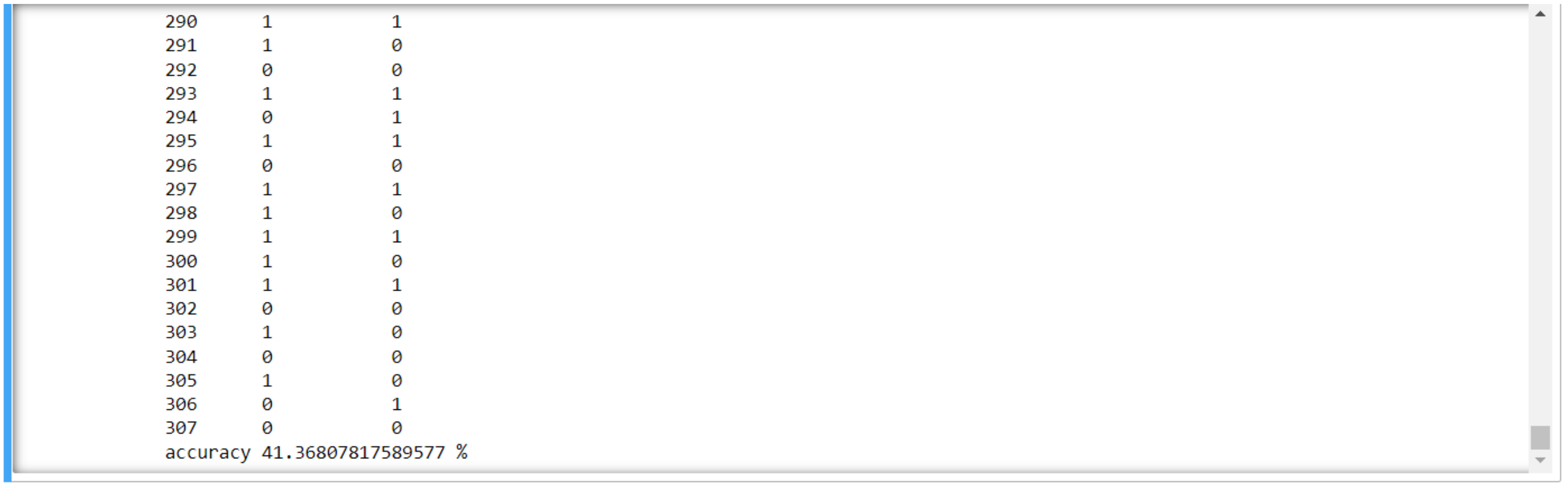
```
In [6]: import numpy as np
        import math
        import csv
        import pdb
        def read_data(filename):

            with open(filename,'r') as csvfile:
                datareader = csv.reader(csvfile)
                metadata = next(datareader)
                traindata=[]
                for row in datareader:
                    traindata.append(row)

            return (metadata, traindata)

        def splitDataset(dataset, splitRatio):
            trainSize = int(len(dataset) * splitRatio)
            trainSet = []
            testset = list(dataset)
            i=0
            while len(trainSet) < trainSize:
                trainSet.append(testset.pop(i))
            return [trainSet, testset]

        def classify(data,test):

            total_size = data.shape[0]
            print("\n")
            print("training data size=",total_size)
            print("test data size=",test.shape[0])
```



```
            countYes = 0
            countNo = 0
            probYes = 0
            probNo = 0
            print("\n")
            print("target     count     probability")

            for x in range(data.shape[0]):
                if data[x,data.shape[1]-1] == '1':
                    countYes +=1
                if data[x,data.shape[1]-1] == '0':
                    countNo +=1

            probYes=countYes/total_size
            probNo= countNo / total_size

            print('Yes',"\t",countYes,"\t",probYes)
            print('No',"\t",countNo,"\t",probNo)


            prob0 =np.zeros((test.shape[1]-1))
            prob1 =np.zeros((test.shape[1]-1))
            accuracy=0
            print("\n")
            print("instance prediction  target")

            for t in range(test.shape[0]):
                for k in range (test.shape[1]-1):
                    count1=count0=0
                    for j in range (data.shape[0]):
                        #how many times appeared with no
                        if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='0':
                            count0+=1
```



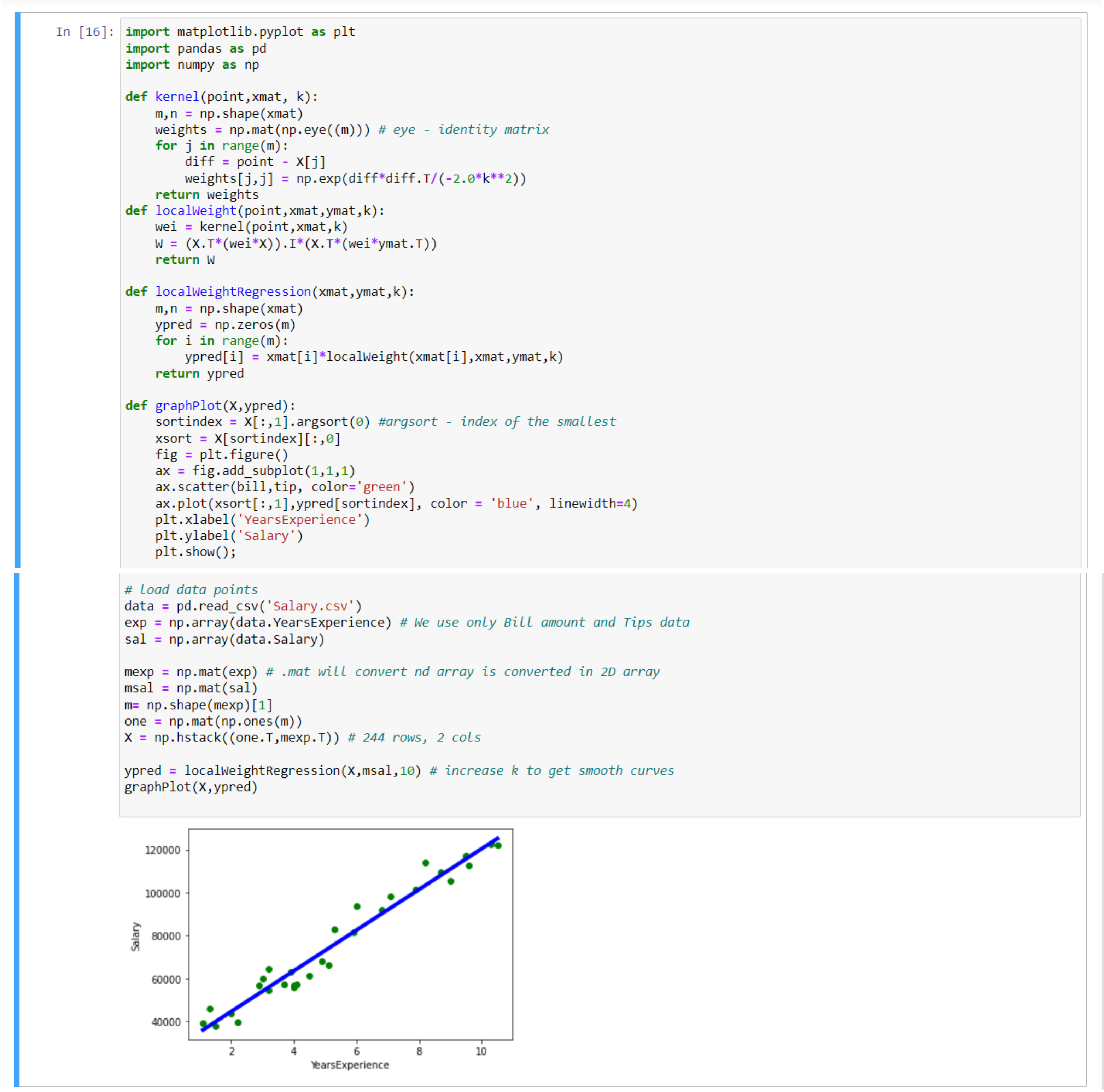
```
                if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='1':
                    count1+=1
            prob0[k]=count0/countNo
            prob1[k]=count1/countYes

        probno=probNo
        probyes=probYes
        for i in range(test.shape[1]-1):
            probno=probno*prob0[i]
            probyes=probyes*prob1[i]
        if probno>probyes:
            predict='0'
        else:
            predict='1'

        print(t+1,"\t",predict,"\t    ",test[t,test.shape[1]-1])
        if predict == test[t,test.shape[1]-1]:
            accuracy+=1
    final_accuracy=(accuracy/test.shape[0])*100
    print("accuracy",final_accuracy,"%")
    return

metadata,traindata= read_data("Diabeteis.csv")
splitRatio=0.6
trainingset, testset=splitDataset(traindata, splitRatio)
training=np.array(trainingset)
print("\n The Training data set are:")
for x in trainingset:
    print(x)

testing=np.array(testset)
print("\n The Test data set are:")
for x in testing:
    print(x)
classify(training,testing)
```



```
290      1           1
291      1           0
292      0           0
293      1           1
294      0           1
295      1           1
296      0           0
297      1           1
298      1           0
299      1           1
300      1           0
301      1           1
302      0           0
303      1           0
304      0           0
305      1           0
306      0           1
307      0           0
accuracy 41.36807817589577 %
```

LAB 5

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



In [16]:

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

def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights
def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'blue', linewidth=4)
    plt.xlabel('YearsExperience')
    plt.ylabel('Salary')
    plt.show();
```

```
# load data points
data = pd.read_csv('Salary.csv')
exp = np.array(data.YearsExperience) # We use only Bill amount and Tips data
sal = np.array(data.Salary)

mexp = np.mat(exp) # .mat will convert nd array is converted in 2D array
msal = np.mat(sal)
m= np.shape(mexp)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mexp.T)) # 244 rows, 2 cols

ypred = localWeightRegression(X,msal,10) # increase k to get smooth curves
graphPlot(X,ypred)
```