VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING (20CS6PCMA)

Submitted by

STHAVIR G SOROFF (1BM19CS161)

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

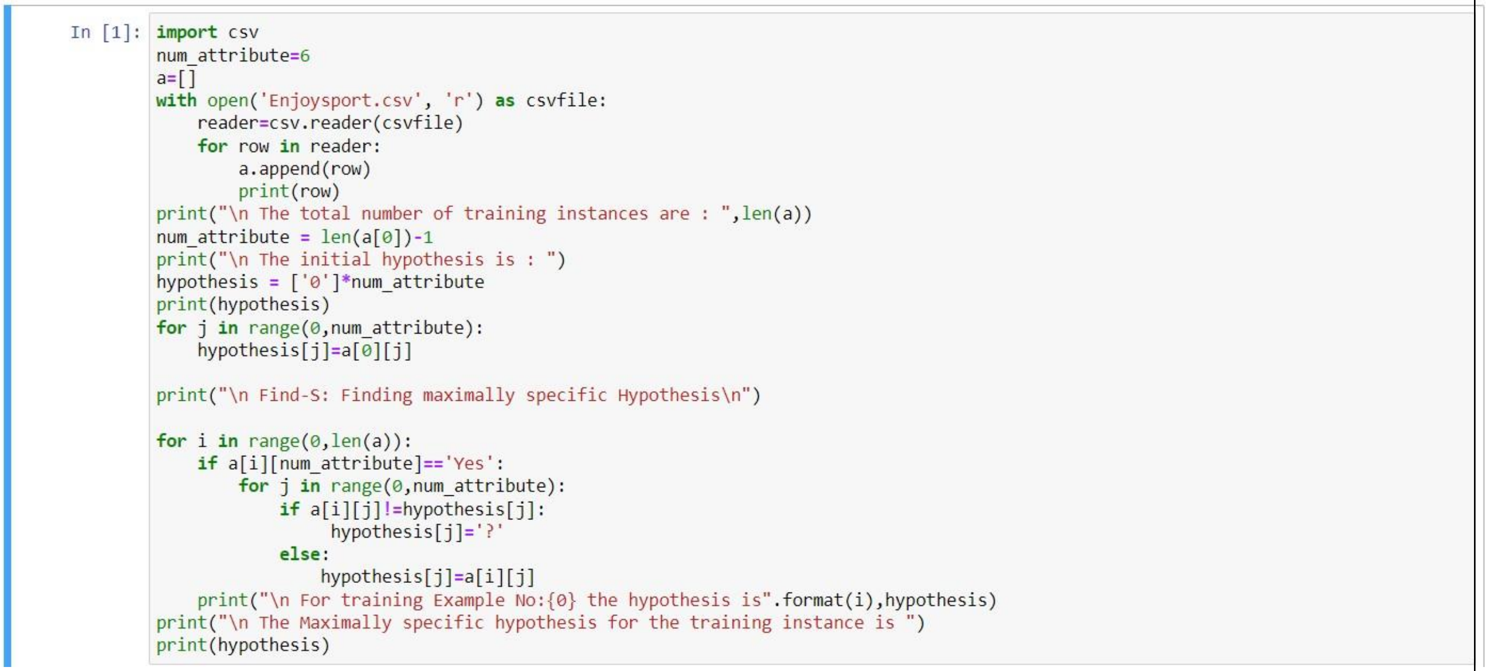
This is to certify that the Lab work entitled "**MACHINE LEARNING**" carried out by STHAVIR G SOROFF (1BM19CS161), who is bonafide student of B. M. S. **College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya 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
Department of CSE
BMSCE, Bengaluru

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

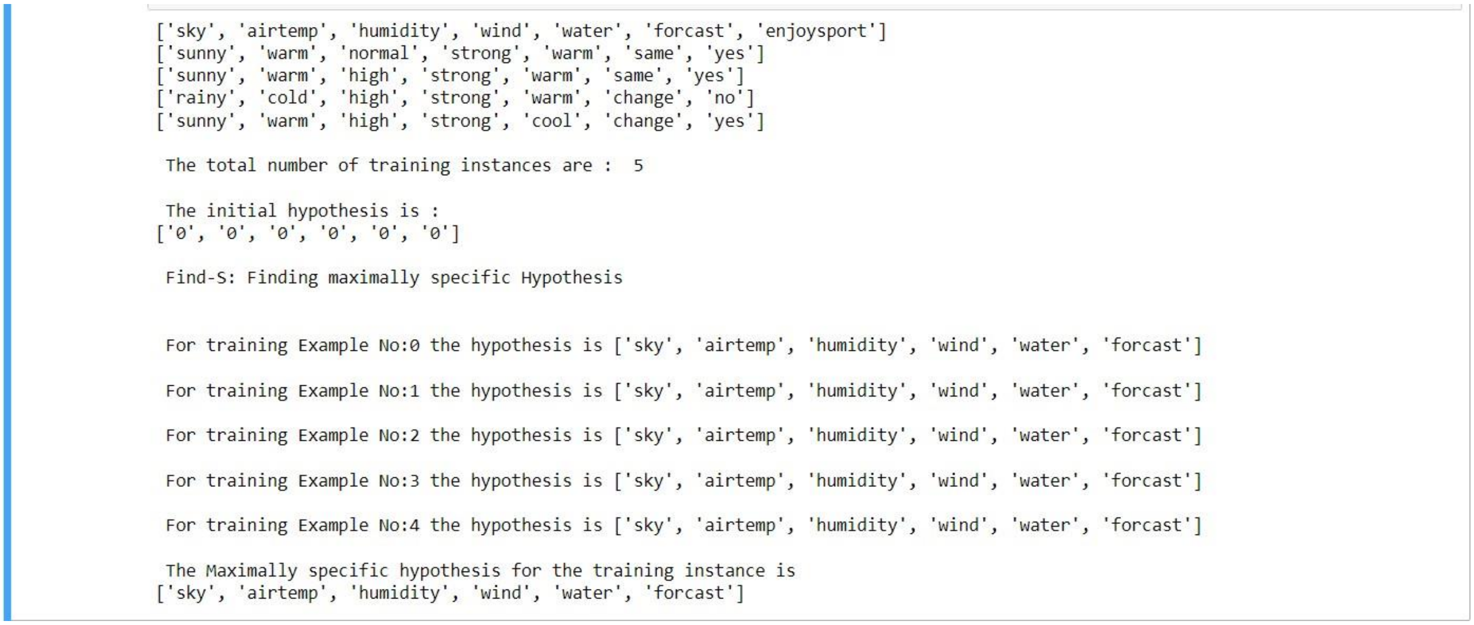
LAB 1

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



In [1]:

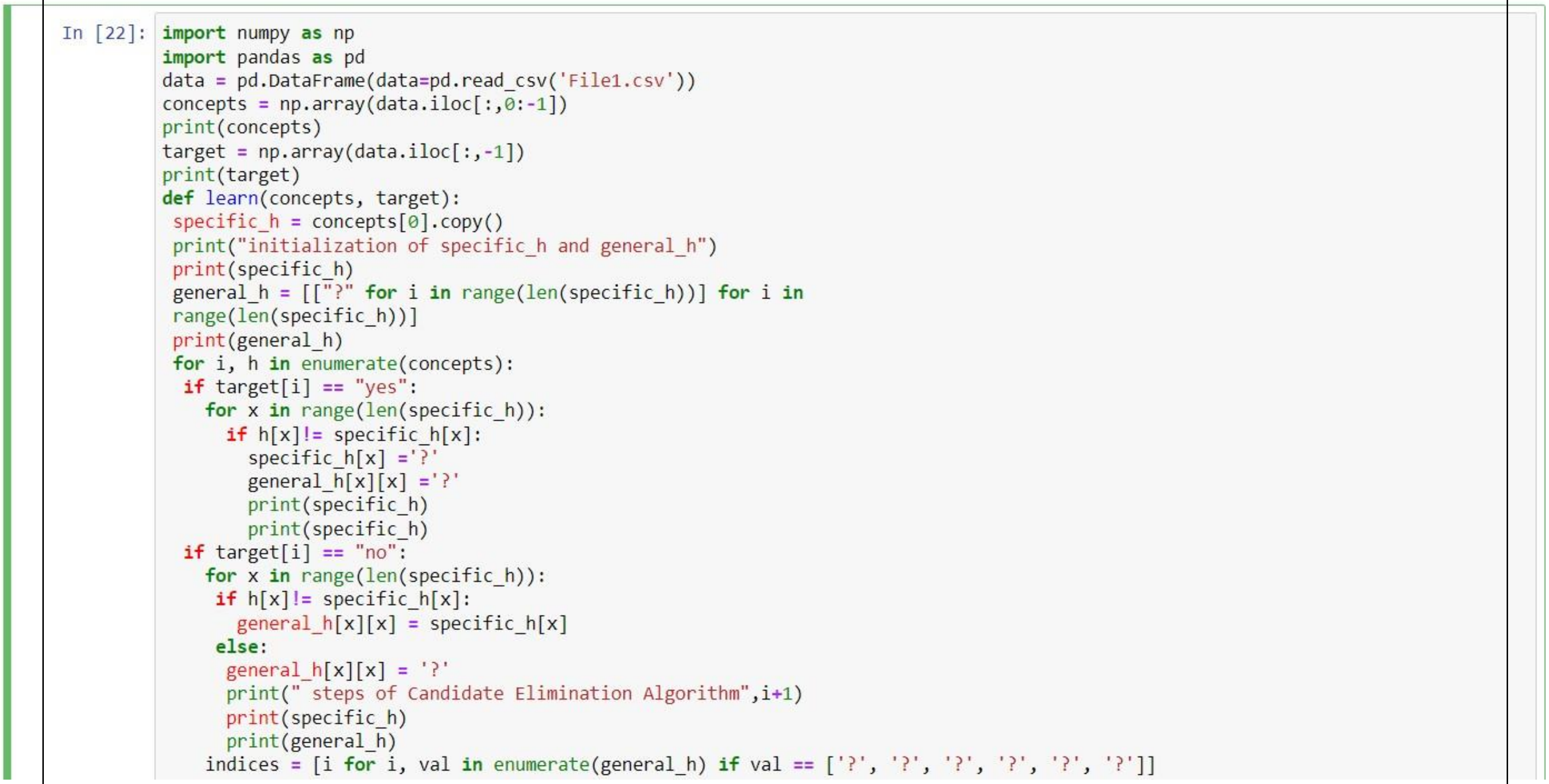
```
import csv
num_attribute=6
a=[]
with open('Enjoysport.csv', 'r') as csvfile:
    reader=csv.reader(csvfile)
    for row in reader:
        a.append(row)
        print(row)
print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis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



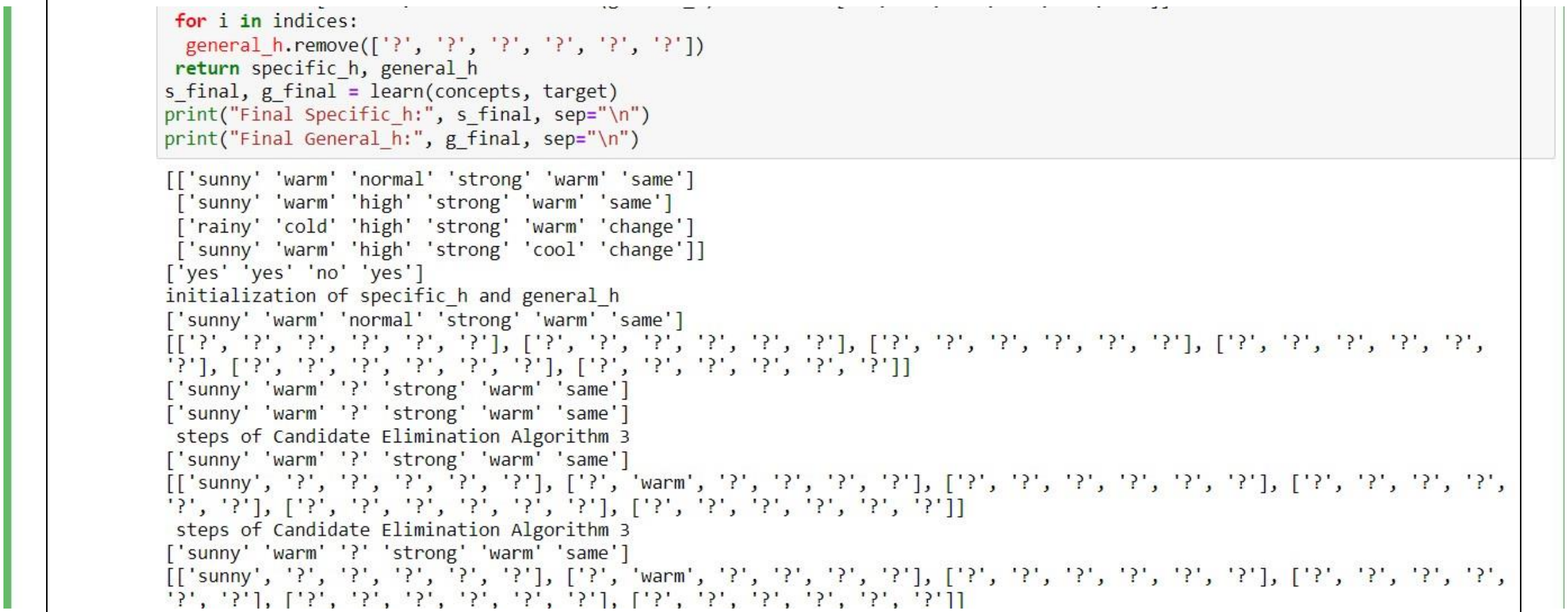
```
['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast', 'enjoysport']
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool',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

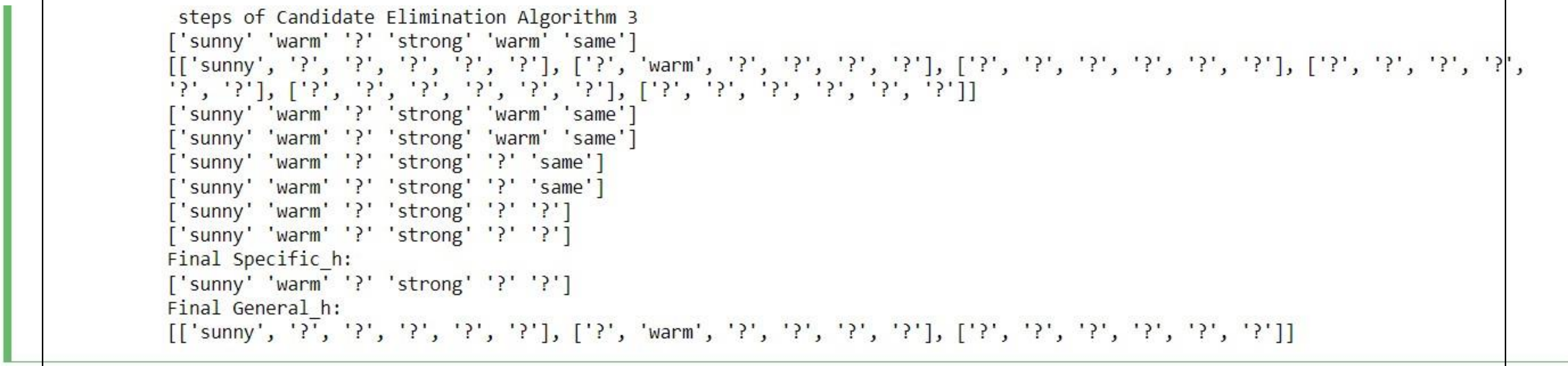


```
In [22]: import numpy as np
         import pandas as pd
         data = pd.DataFrame(data=pd.read_csv('File1.csv'))
         concepts = np.array(data.iloc[:,0:-1])
         print(concepts)
         target = np.array(data.iloc[:,-1])
         print(target)
         def learn(concepts, target):
          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")
```

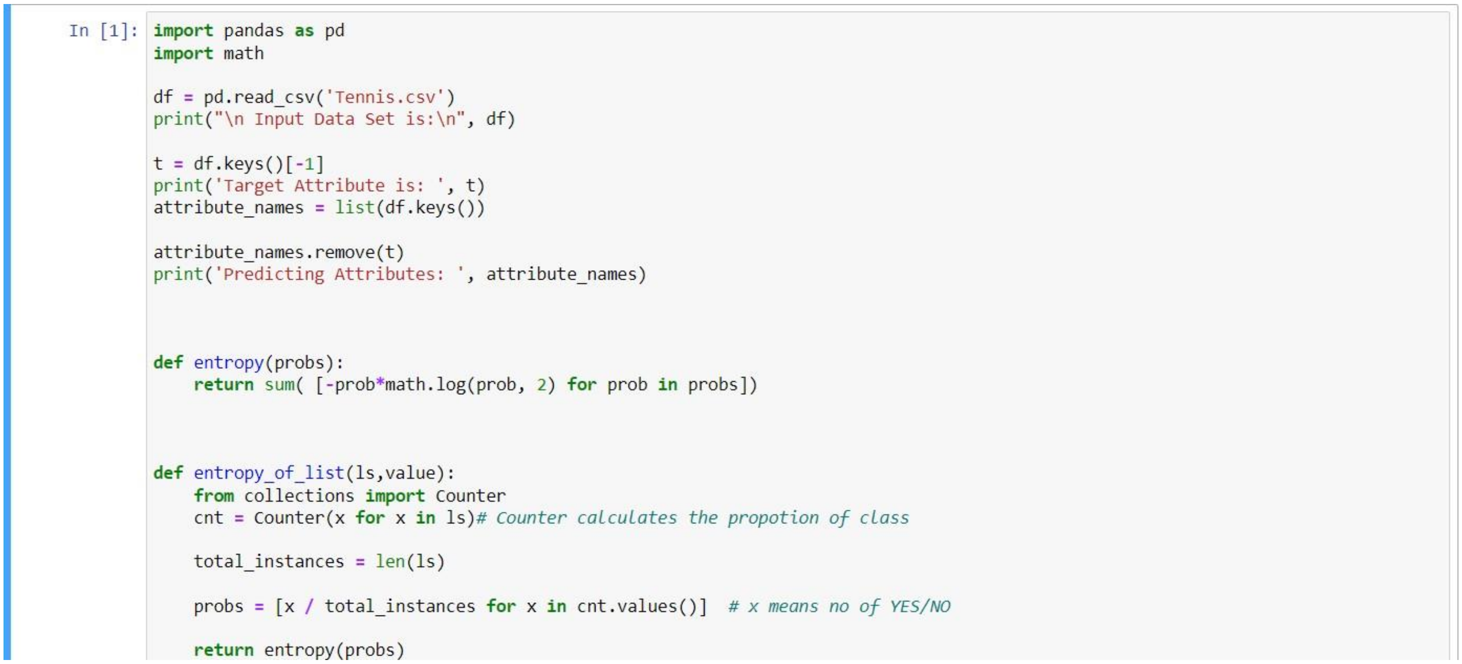
```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong'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



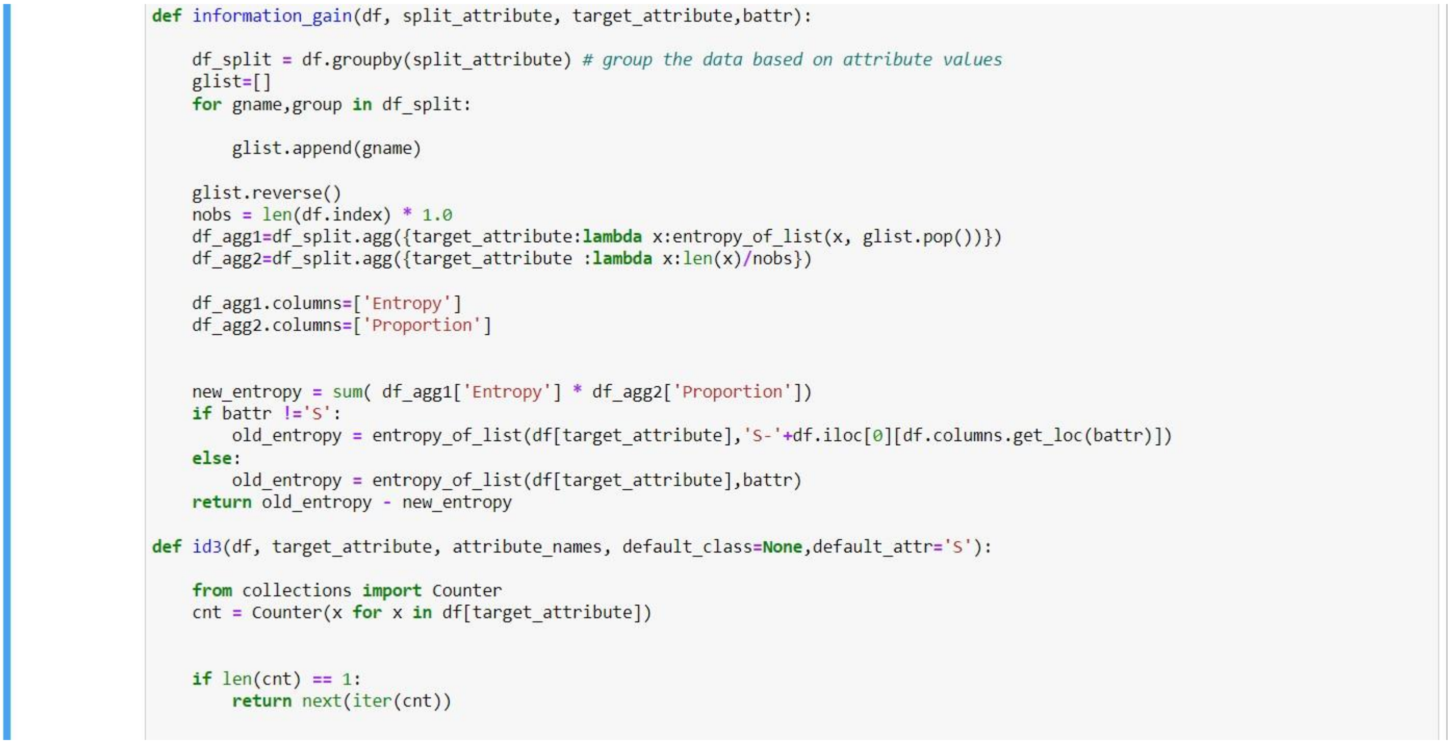
```
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny'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

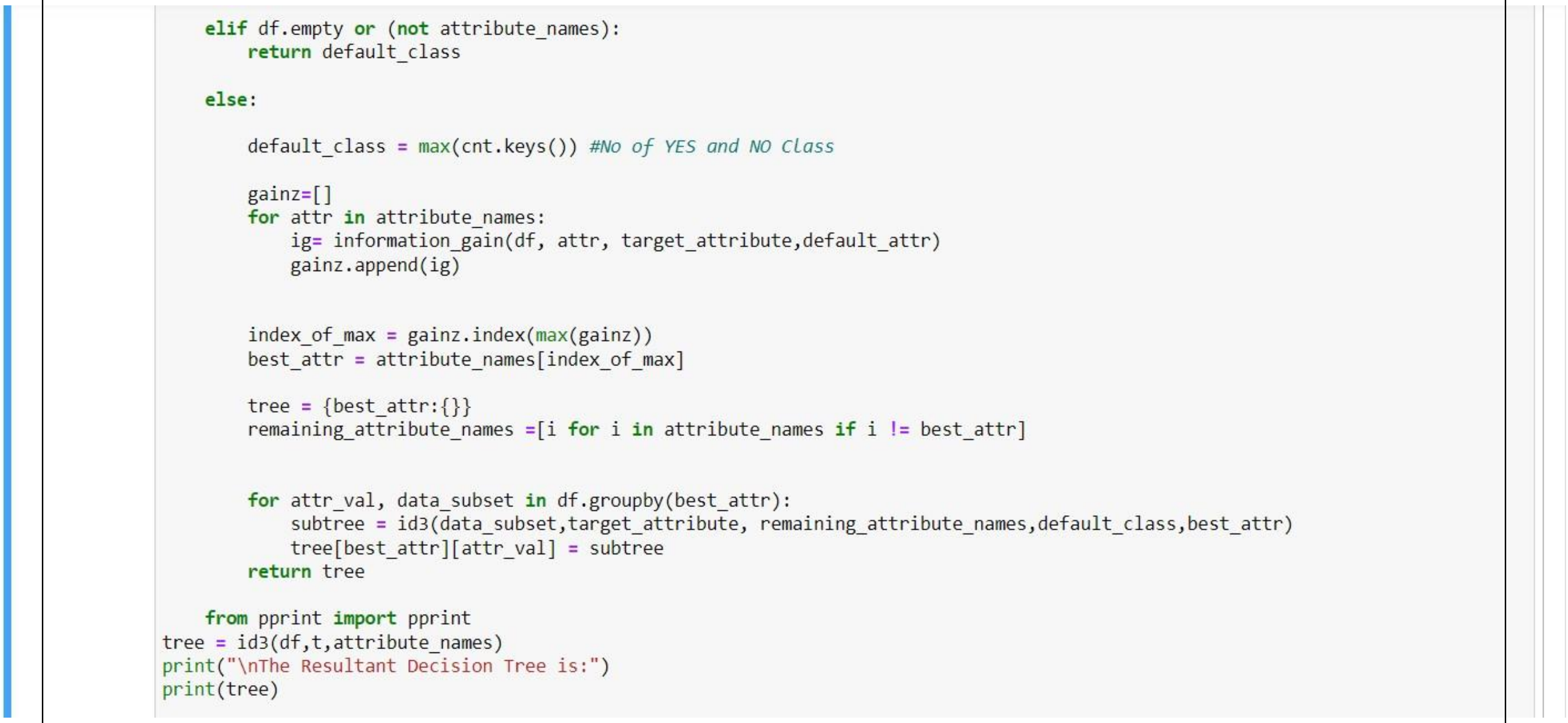
Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use anappropriate data set for building the decision tree and apply this knowledge to classify a new sample.



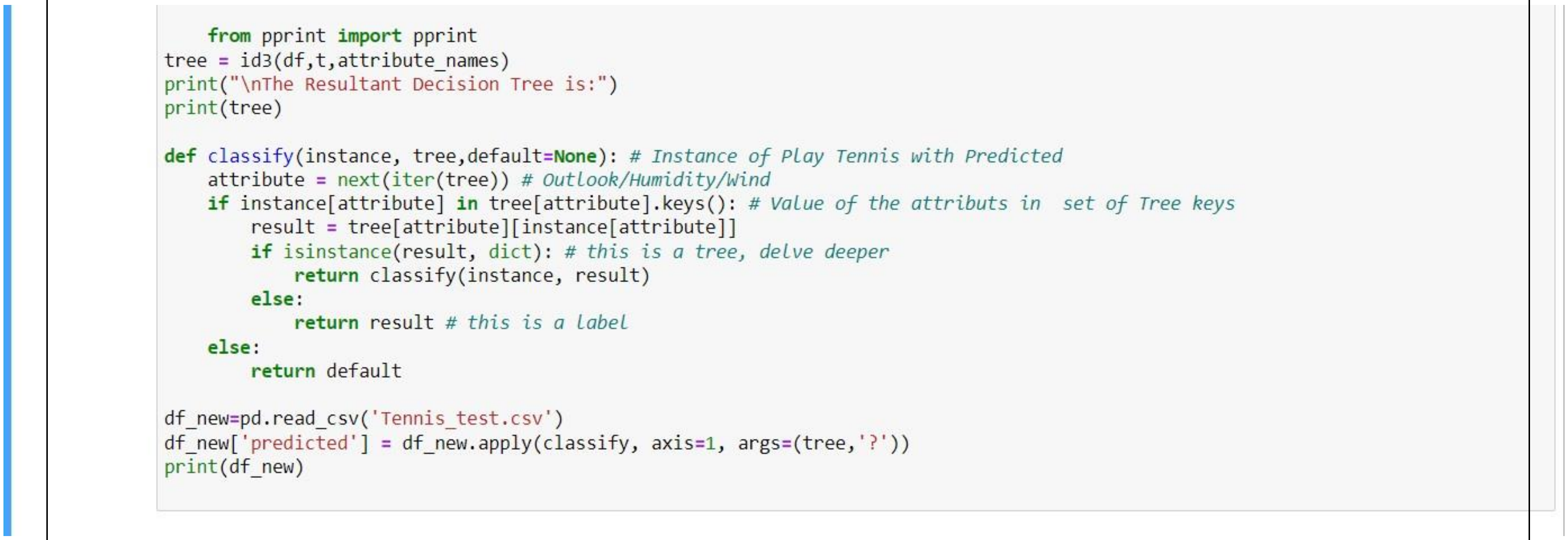
```
In [1]: import pandas as pd
        import math

        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)
```



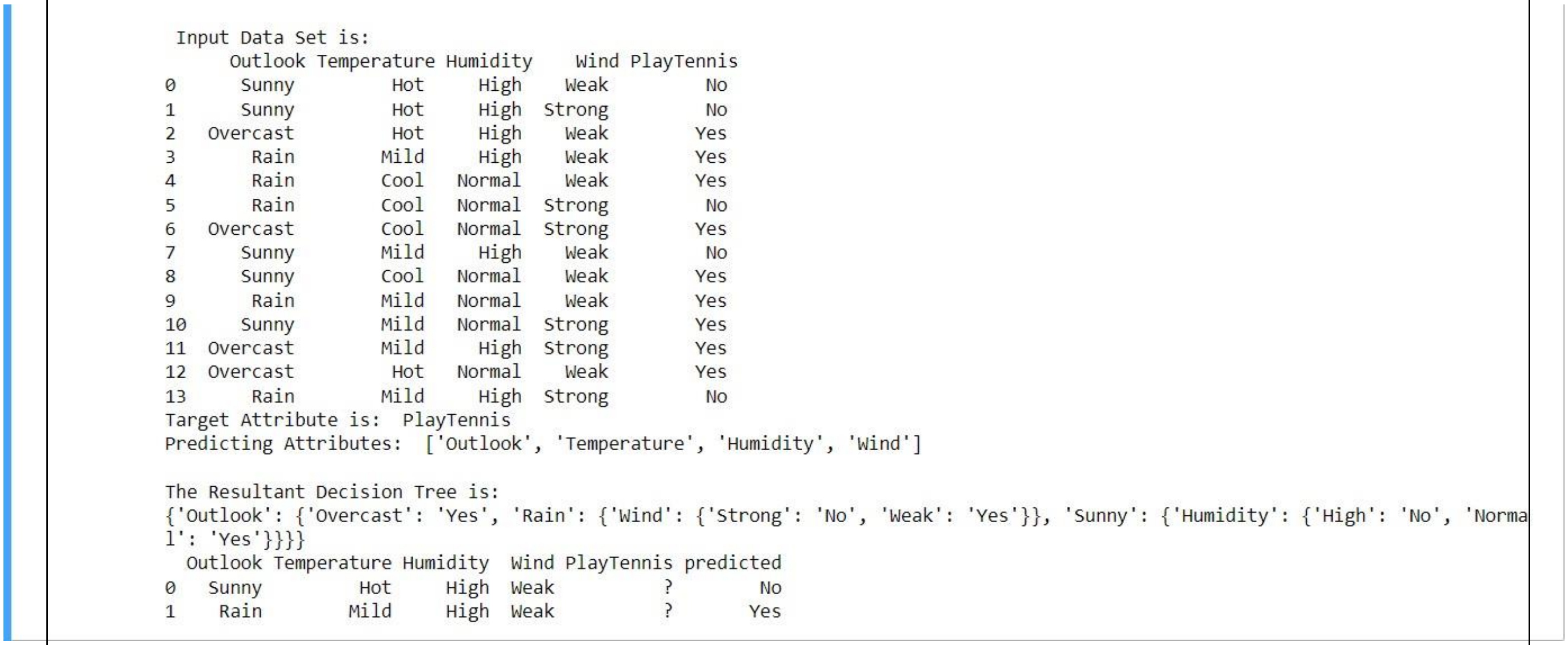
```
def information_gain(df, split_attribute, target_attribute,battr):

    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))
```



```
    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)
```

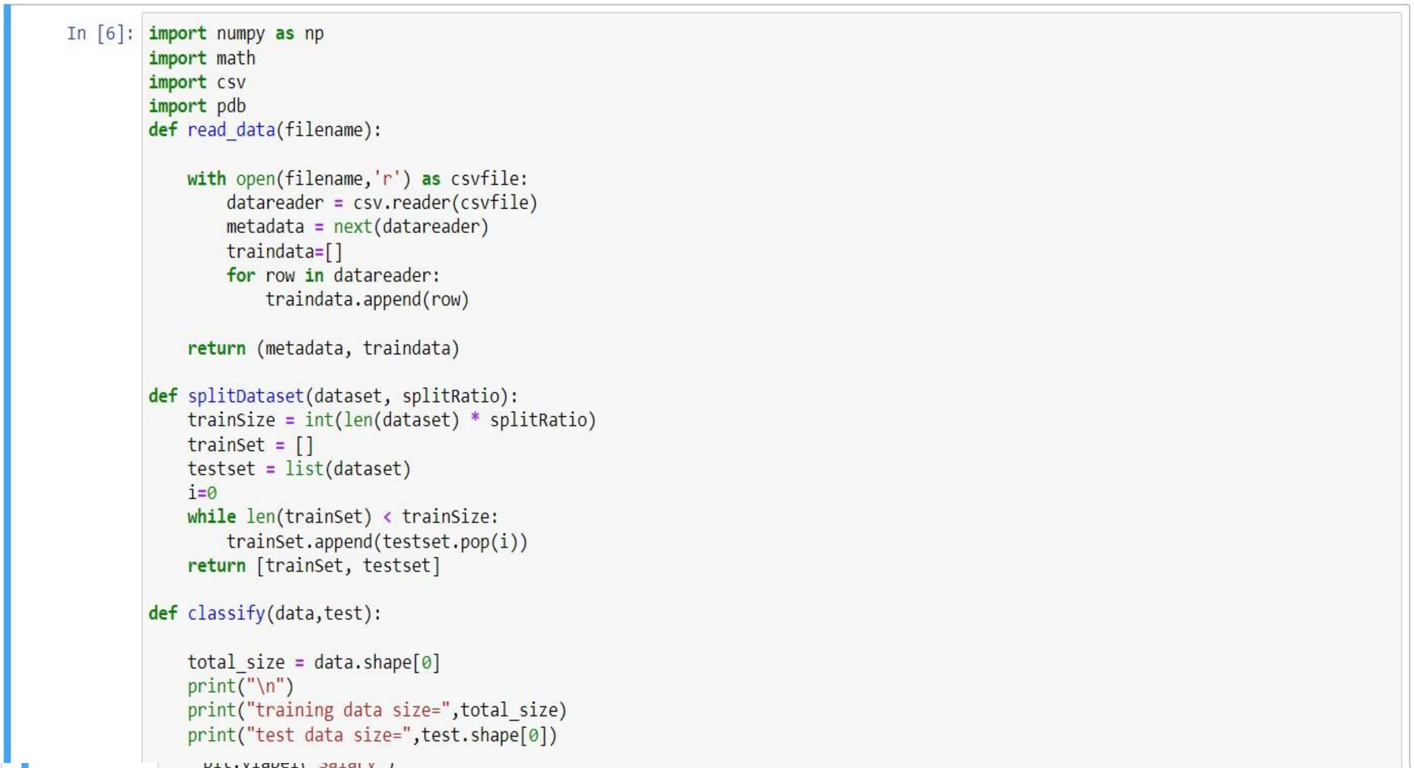


```
    from pprint import pprint
tree = id3(df,t,attribute_names)
print("\nThe Resultant Decision Tree is:")
print(tree)

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)
```



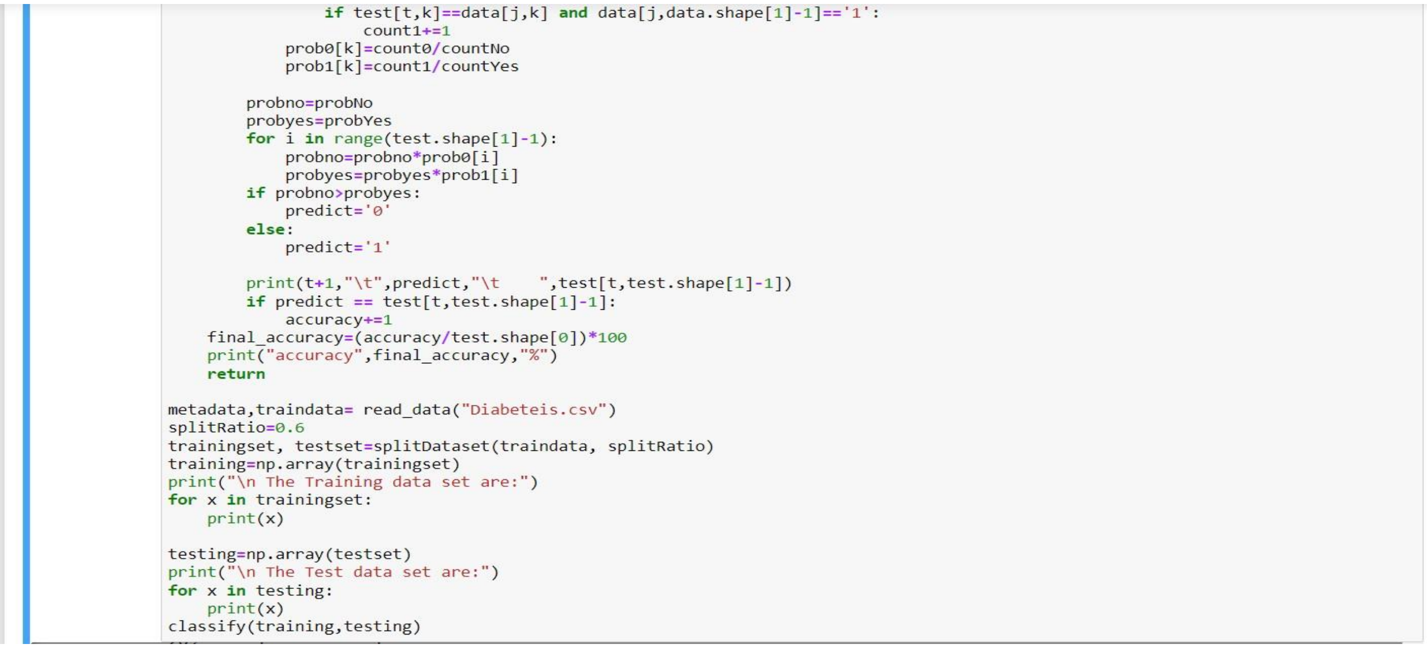
```
 Input Data Set is:
      Outlook Temperature Humidity    Wind PlayTennis
0      Sunny         Hot     High    Weak         No
1      Sunny         Hot     High  Strong         No
2   Overcast         Hot     High    Weak        Yes
3       Rain        Mild     High    Weak        Yes
4       Rain        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 Implement the Naïve Bayesian Algorithm

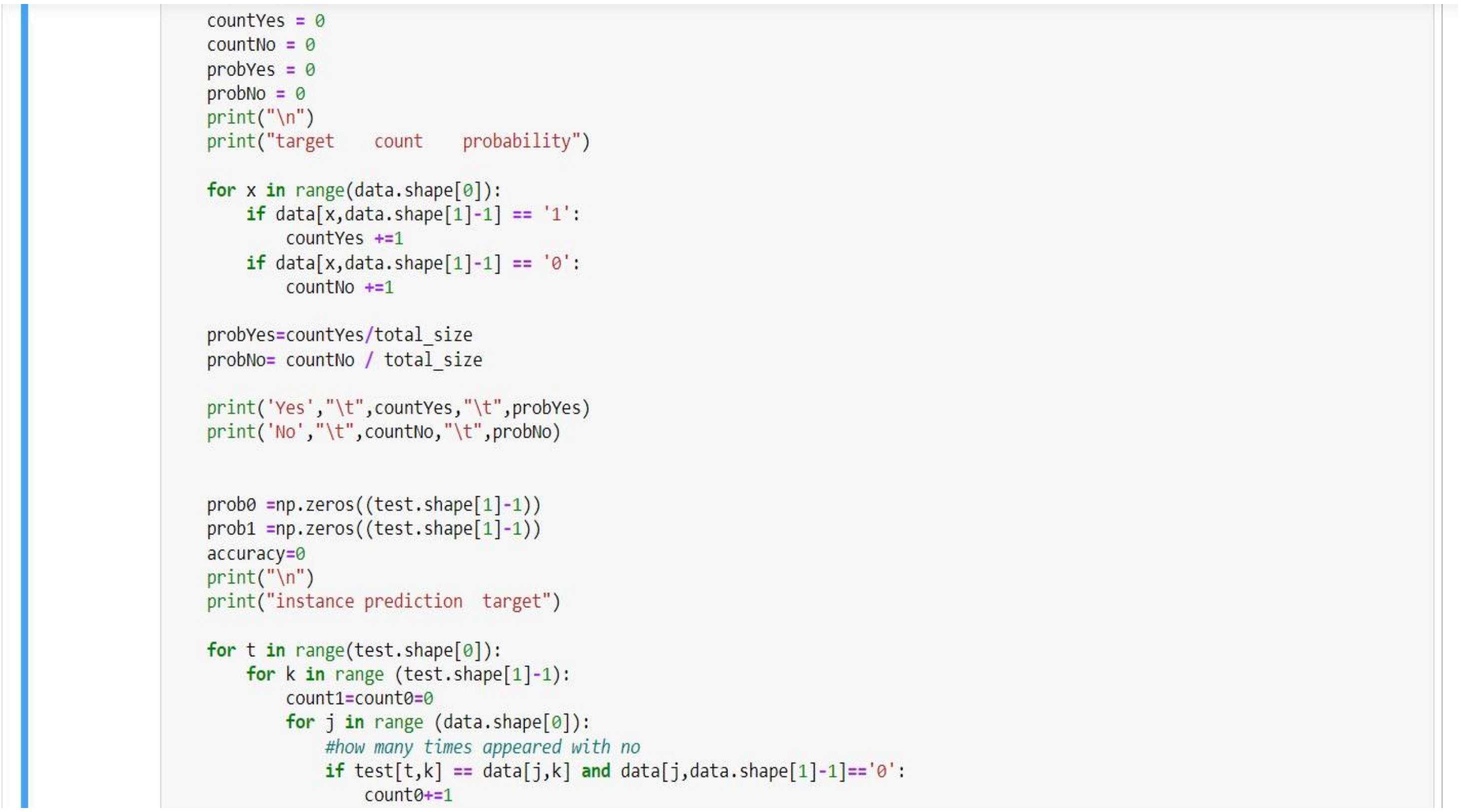


```
In [6]: import numpy as np
        import math
        import csv
        import pdb
        def read_data(filename):

            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])
```

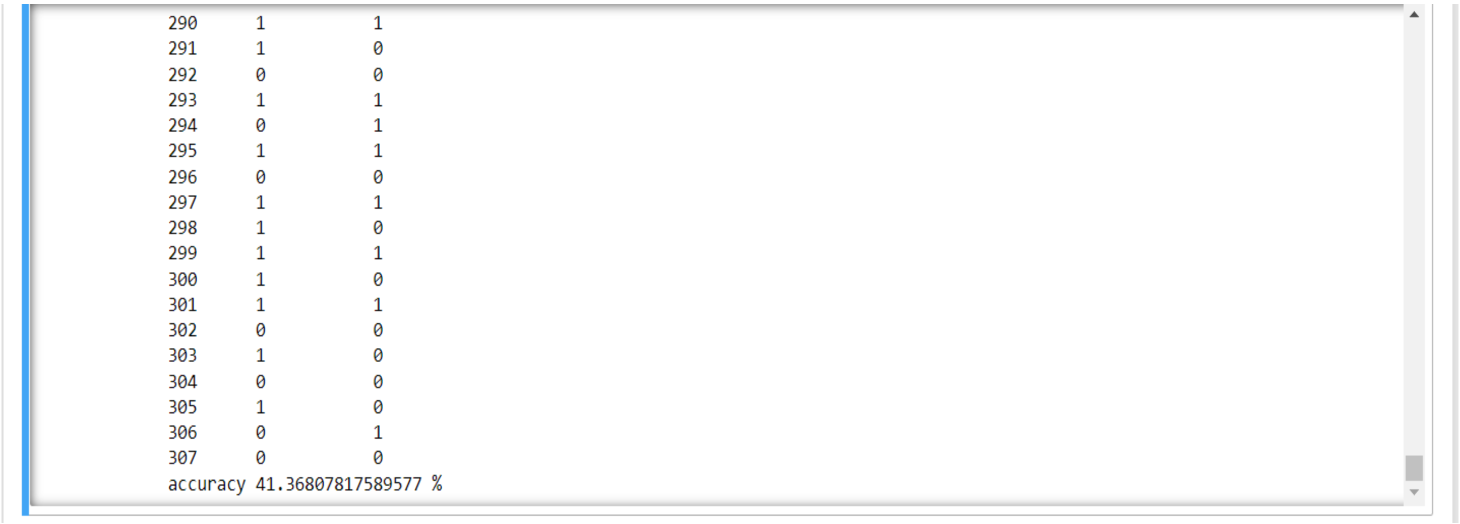
```
plt.ylabel('Salary')
plt.show();
```



```
                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)
```



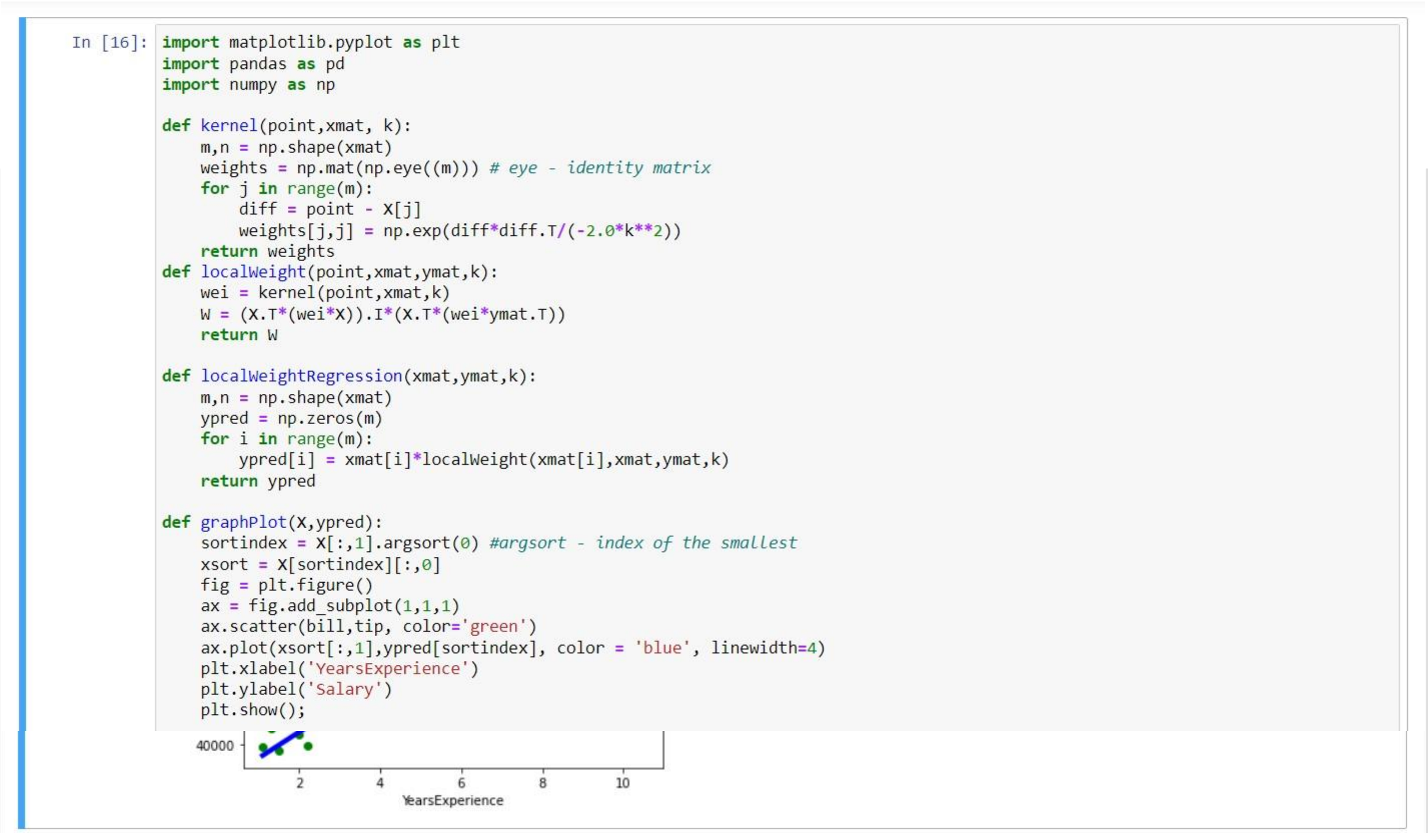
```
countYes = 0
countNo = 0
probYes = 0
probNo = 0
print("\n")
print("target    count    probability")

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
```



```
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

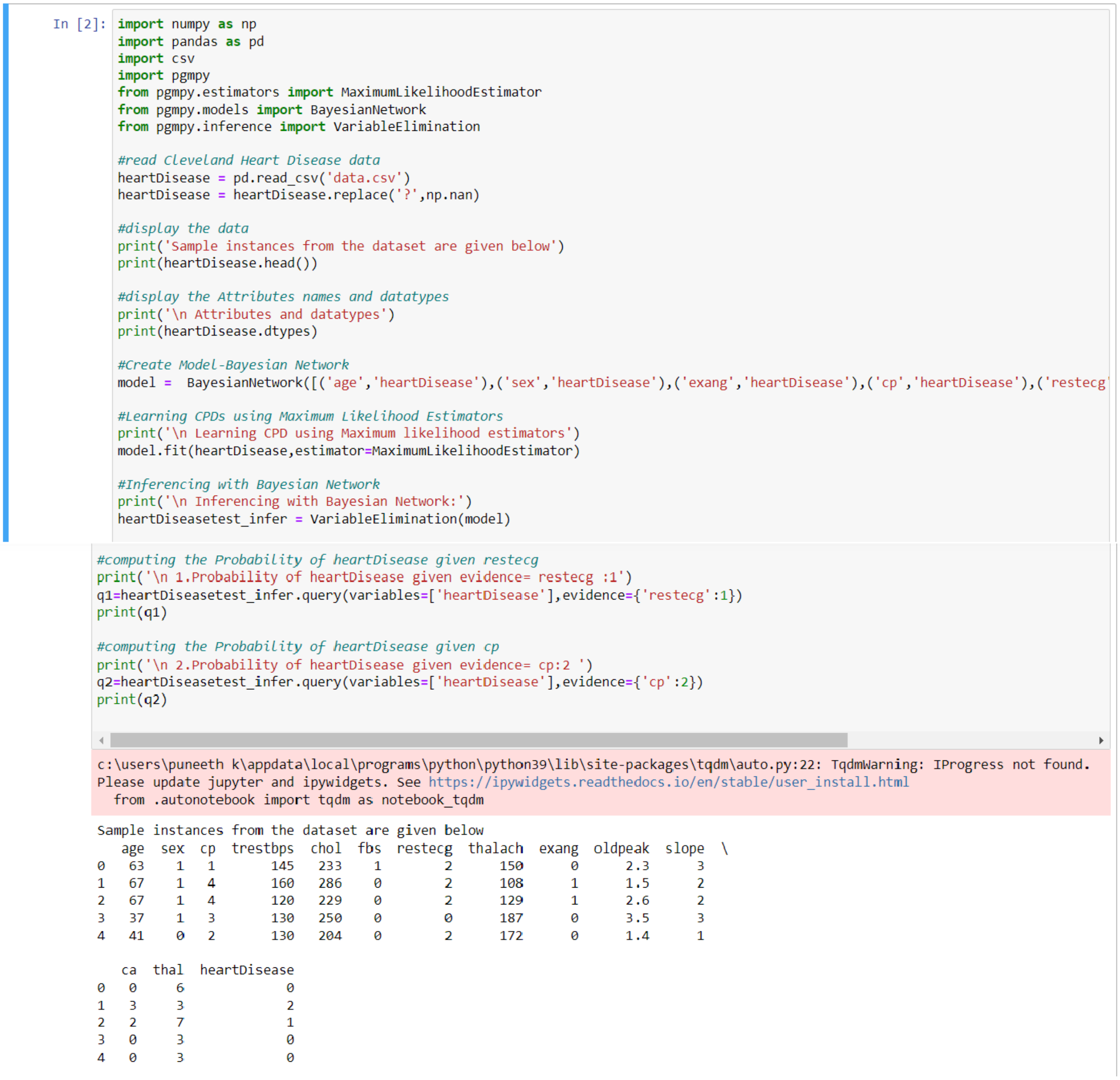
Implement the Linear Regression algorithm in order to fit data points. Select appropriate data setfor your experiment and d



```
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();
```

LAB 6

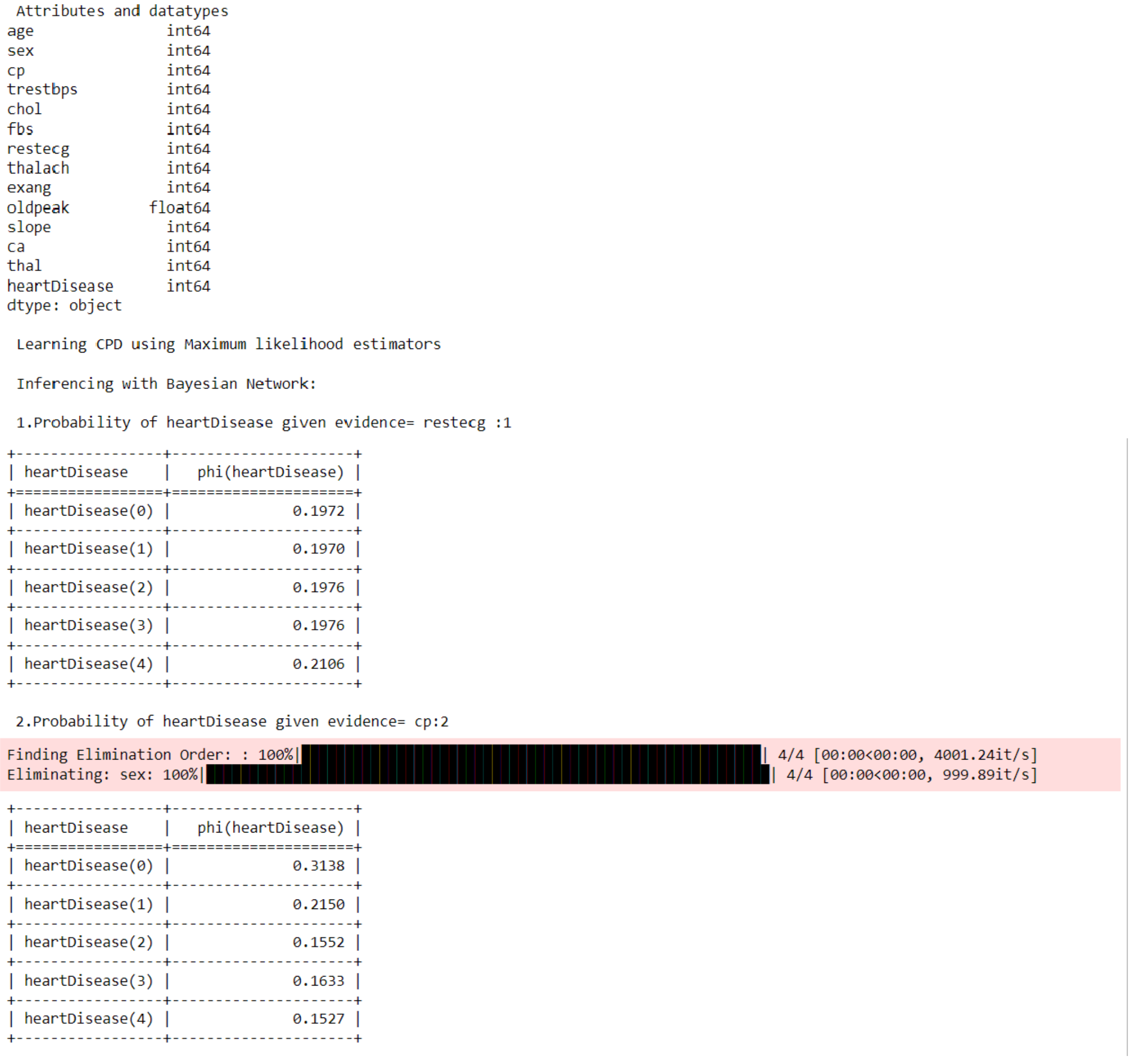
Apply Bayesian network over the given set of data



```
In [2]: import numpy as np
        import pandas as pd
        import csv
        import pgmpy
        from pgmpy.estimators import MaximumLikelihoodEstimator
        from pgmpy.models import BayesianNetwork
        from pgmpy.inference import VariableElimination

        #read Cleveland Heart Disease data
        heartDisease = pd.read_csv('data.csv')
        heartDisease = heartDisease.replace('?',np.nan)

        #display the data
        print('Sample instances from the dataset are given below')
        print(heartDisease.head())

        #display the Attributes names and datatypes
        print('\n Attributes and datatypes')
        print(heartDisease.dtypes)

        #Create Model-Bayesian Network
        model =  BayesianNetwork([('age','heartDisease'),('sex','heartDisease'),('exang','heartDisease'),('cp','heartDisease'),('restecg'

        #Learning CPDs using Maximum Likelihood Estimators
        print('\n Learning CPD using Maximum likelihood estimators')
        model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

        #Inferencing with Bayesian Network
        print('\n Inferencing with Bayesian Network:')
        heartDiseasetest_infer = VariableElimination(model)

        #computing the Probability of heartDisease given restecg
        print('\n 1.Probability of heartDisease given evidence= restecg :1')
        q1=heartDiseasetest_infer.query(variables=['heartDisease'],evidence={'restecg':1})
        print(q1)

        #computing the Probability of heartDisease given cp
        print('\n 2.Probability of heartDisease given evidence= cp:2 ')
        q2=heartDiseasetest_infer.query(variables=['heartDisease'],evidence={'cp':2})
        print(q2)
```

```
c:\users\puneeth k\appdata\local\programs\python\python39\lib\site-packages\tqdm\auto.py:22: TqdmWarning: IProgress not found.
Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

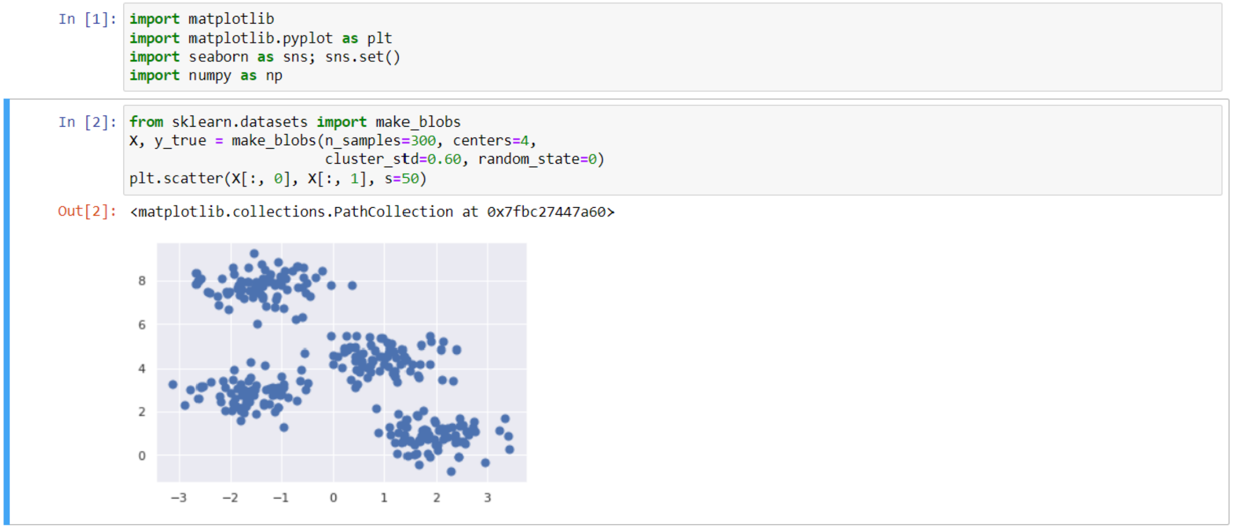
```
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   63    1   1       145   233    1        2      150      0      2.3      3
1   67    1   4       160   286    0        2      108      1      1.5      2
2   67    1   4       120   229    0        2      129      1      2.6      2
3   37    1   3       130   250    0        0      187      0      3.5      3
4   41    0   2       130   204    0        2      172      0      1.4      1

   ca  thal  heartDisease
0   0     6             0
1   3     3             2
2   2     7             1
3   0     3             0
4   0     3             0
```



```
 Attributes and datatypes
age                int64
sex                int64
cp                 int64
trestbps           int64
chol               int64
fbs                int64
restecg            int64
thalach            int64
exang              int64
oldpeak          float64
slope              int64
ca                 int64
thal               int64
heartDisease       int64
dtype: object

 Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1.Probability of heartDisease given evidence= restecg :1
+-----------------+---------------------+
| heartDisease    |   phi(heartDisease) |
+=================+=====================+
| heartDisease(0) |              0.1972 |
+-----------------+---------------------+
| heartDisease(1) |              0.1970 |
+-----------------+---------------------+
| heartDisease(2) |              0.1976 |
+-----------------+---------------------+
| heartDisease(3) |              0.1976 |
+-----------------+---------------------+
| heartDisease(4) |              0.2106 |
+-----------------+---------------------+

 2.Probability of heartDisease given evidence= cp:2
Finding Elimination Order: : 100%|████████████████████████████████████████████████████| 4/4 [00:00<00:00, 4001.24it/s]
Eliminating: sex: 100%|████████████████████████████████████████████████████████████████| 4/4 [00:00<00:00, 999.89it/s]
+-----------------+---------------------+
| heartDisease    |   phi(heartDisease) |
+=================+=====================+
| heartDisease(0) |              0.3138 |
+-----------------+---------------------+
| heartDisease(1) |              0.2150 |
+-----------------+---------------------+
| heartDisease(2) |              0.1552 |
+-----------------+---------------------+
| heartDisease(3) |              0.1633 |
+-----------------+---------------------+
| heartDisease(4) |              0.1527 |
+-----------------+---------------------+
```

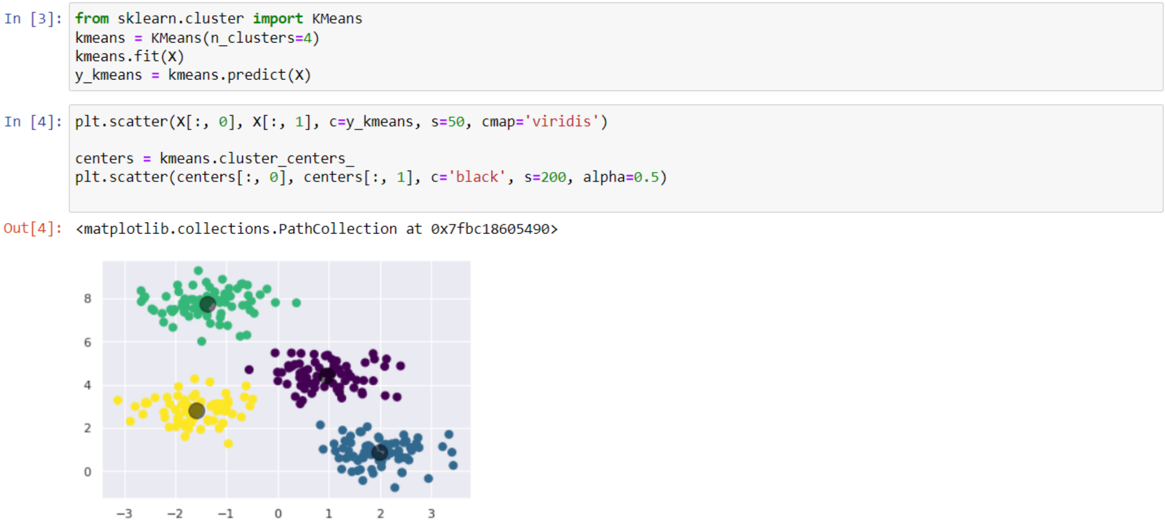
LAB 7

Apply k-Means algorithm to cluster a set of data stored in a .CSV file.



```
In [1]: import matplotlib
        import matplotlib.pyplot as plt
        import seaborn as sns; sns.set()
        import numpy as np
```

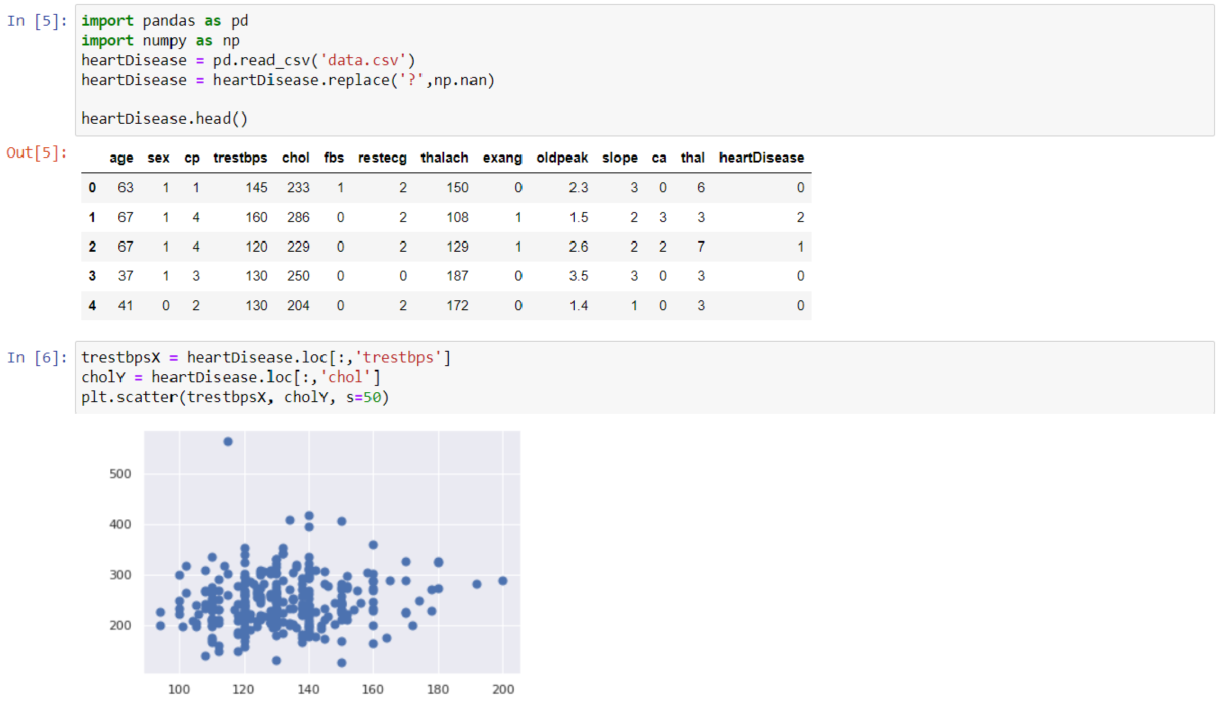
```
In [2]: from sklearn.datasets import make_blobs
        X, y_true = make_blobs(n_samples=300, centers=4,
                               cluster_std=0.60, random_state=0)
        plt.scatter(X[:, 0], X[:, 1], s=50)

Out[2]: <matplotlib.collections.PathCollection at 0x7fbc27447a60>
```



```
In [3]: from sklearn.cluster import KMeans
        kmeans = KMeans(n_clusters=4)
        kmeans.fit(X)
        y_kmeans = kmeans.predict(X)
```

```
In [4]: plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

        centers = kmeans.cluster_centers_
        plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)

Out[4]: <matplotlib.collections.PathCollection at 0x7fbc18605490>
```

K-Means on Heart Disease Dataset

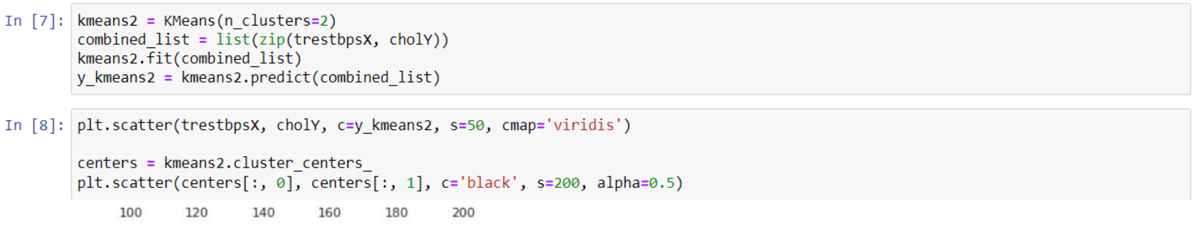


In [5]:
```
import pandas as pd
import numpy as np
heartDisease = pd.read_csv('data.csv')
heartDisease = heartDisease.replace('?',np.nan)

heartDisease.head()
```

Out[5]:

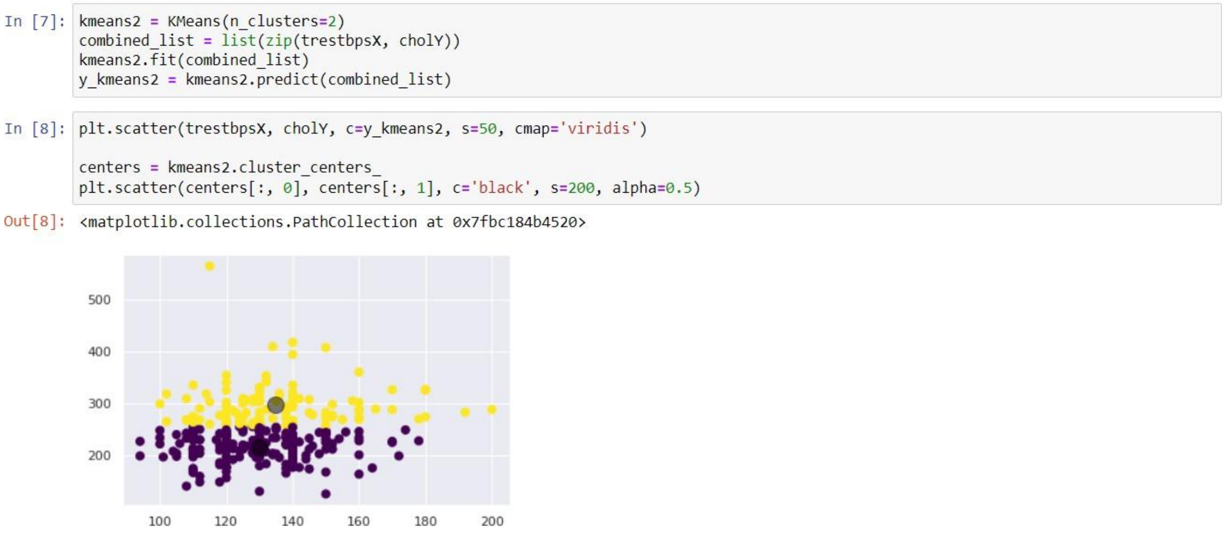
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartDisease
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

In [6]:
```
trestbpsX = heartDisease.loc[:,'trestbps']
cholY = heartDisease.loc[:,'chol']
plt.scatter(trestbpsX, cholY, s=50)
```



In [7]:
```
kmeans2 = KMeans(n_clusters=2)
combined_list = list(zip(trestbpsX, cholY))
kmeans2.fit(combined_list)
y_kmeans2 = kmeans2.predict(combined_list)
```

In [8]:
```
plt.scatter(trestbpsX, cholY, c=y_kmeans2, s=50, cmap='viridis')

centers = kmeans2.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
```



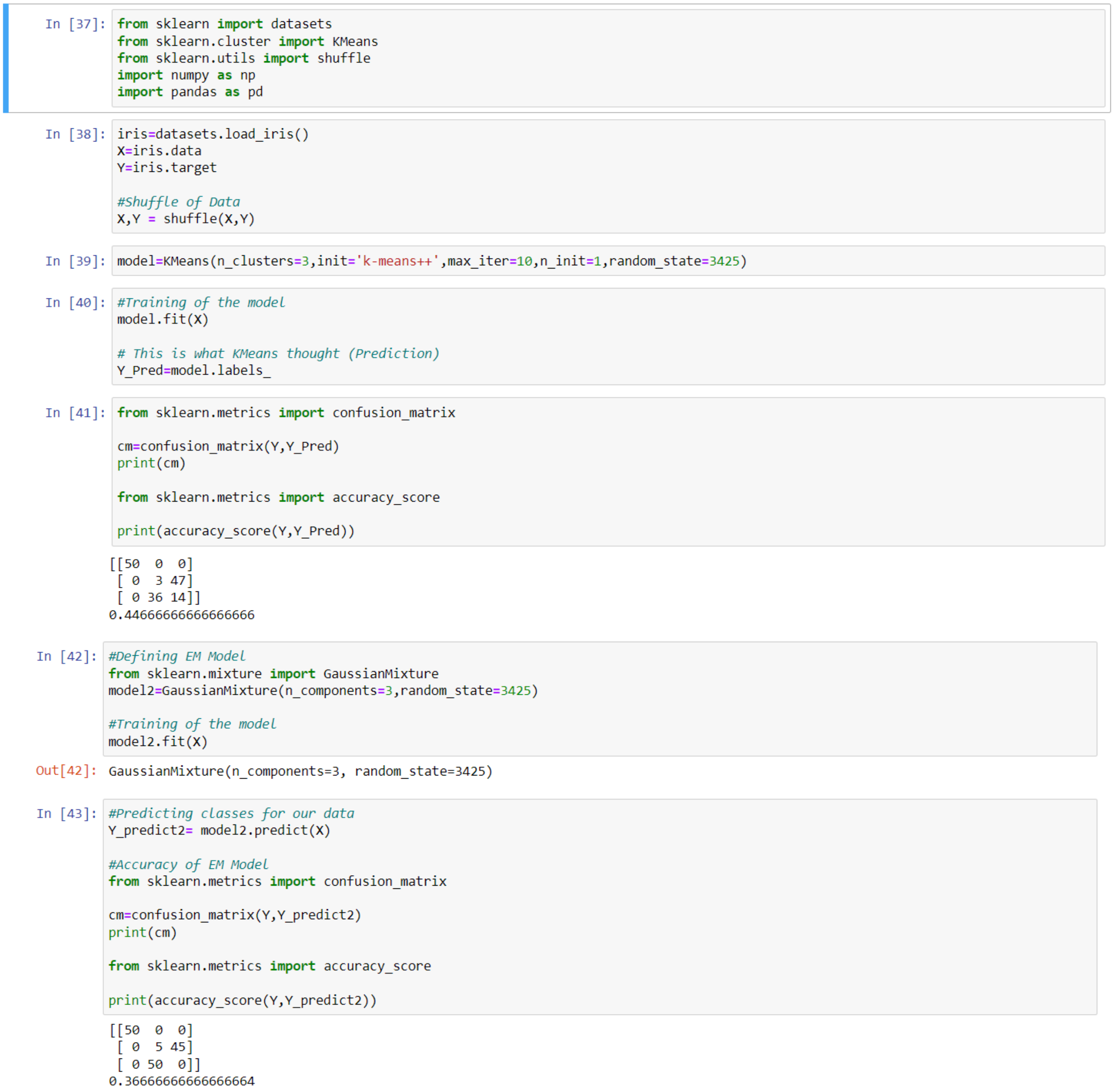
In [7]:
```
kmeans2 = KMeans(n_clusters=2)
combined_list = list(zip(trestbpsX, cholY))
kmeans2.fit(combined_list)
y_kmeans2 = kmeans2.predict(combined_list)
```

In [8]:
```
plt.scatter(trestbpsX, cholY, c=y_kmeans2, s=50, cmap='viridis')

centers = kmeans2.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
```

Out[8]:
```
<matplotlib.collections.PathCollection at 0x7fbc184b4520>
```

LAB 8

Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Meansalgorithm and EM algorithm



```
In [37]: from sklearn import datasets
         from sklearn.cluster import KMeans
         from sklearn.utils import shuffle
         import numpy as np
         import pandas as pd
```

```
In [38]: iris=datasets.load_iris()
         X=iris.data
         Y=iris.target

         #Shuffle of Data
         X,Y = shuffle(X,Y)
```

```
In [39]: model=KMeans(n_clusters=3,init='k-means++',max_iter=10,n_init=1,random_state=3425)
```

```
In [40]: #Training of the model
         model.fit(X)

         # This is what KMeans thought (Prediction)
         Y_Pred=model.labels_
```

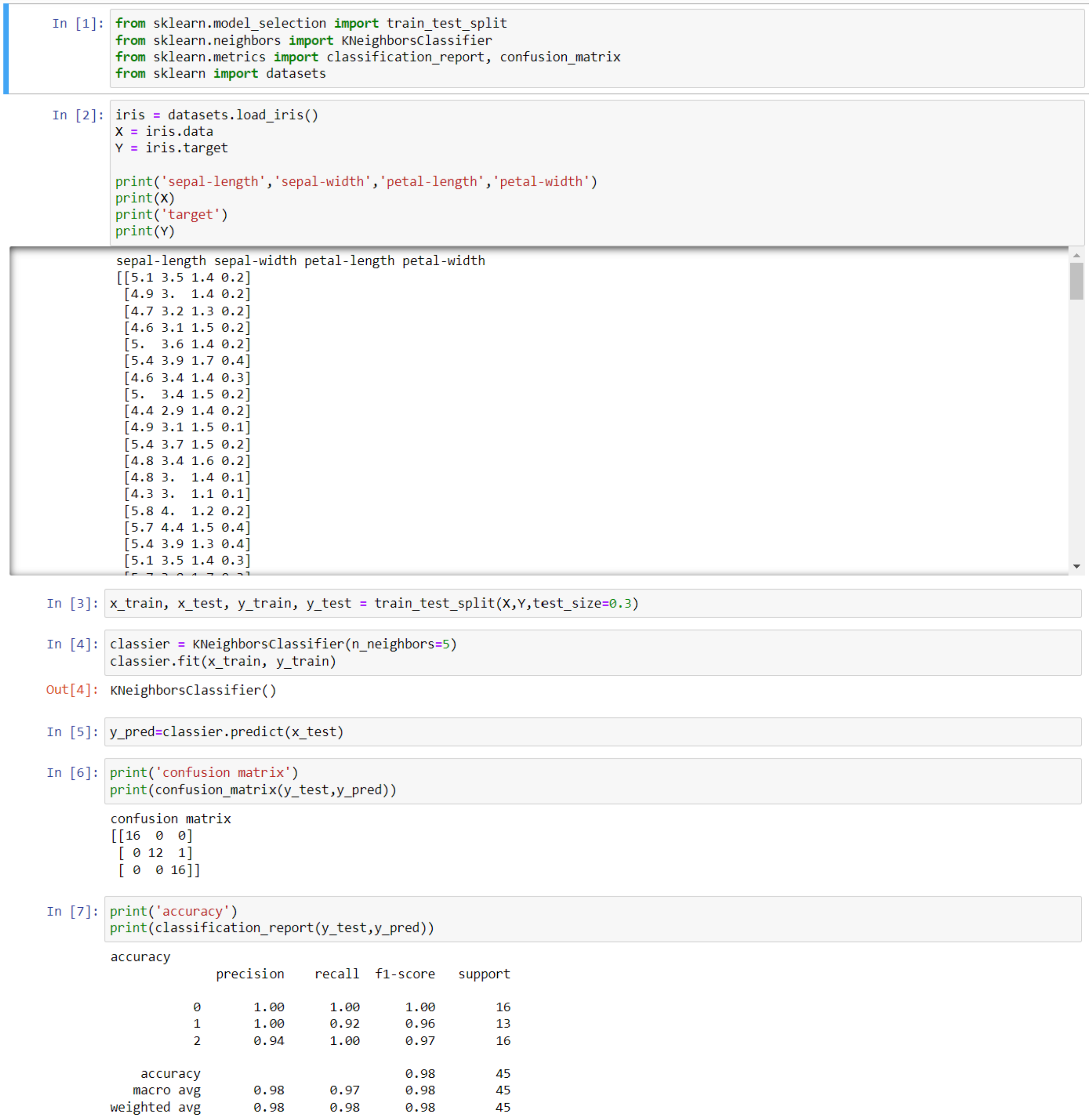
```
In [41]: from sklearn.metrics import confusion_matrix

         cm=confusion_matrix(Y,Y_Pred)
         print(cm)

         from sklearn.metrics import accuracy_score

         print(accuracy_score(Y,Y_Pred))

         [[50  0  0]
          [ 0  3 47]
          [ 0 36 14]]
         0.44666666666666666
```

```
In [42]: #Defining EM Model
         from sklearn.mixture import GaussianMixture
         model2=GaussianMixture(n_components=3,random_state=3425)

         #Training of the model
         model2.fit(X)

Out[42]: GaussianMixture(n_components=3, random_state=3425)
```

```
In [43]: #Predicting classes for our data
         Y_predict2= model2.predict(X)

         #Accuracy of EM Model
         from sklearn.metrics import confusion_matrix

         cm=confusion_matrix(Y,Y_predict2)
         print(cm)

         from sklearn.metrics import accuracy_score

         print(accuracy_score(Y,Y_predict2))

         [[50  0  0]
          [ 0  5 45]
          [ 0 50  0]]
         0.36666666666666664
```

LAB 9

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print bothcorrect and wrong predictions.



```
In [1]: from sklearn.model_selection import train_test_split
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import classification_report, confusion_matrix
        from sklearn import datasets
```

```
In [2]: iris = datasets.load_iris()
        X = iris.data
        Y = iris.target

        print('sepal-length','sepal-width','petal-length','petal-width')
        print(X)
        print('target')
        print(Y)
```

```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
```

```
In [3]: x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.3)
```

```
In [4]: classier = KNeighborsClassifier(n_neighbors=5)
        classier.fit(x_train, y_train)

Out[4]: KNeighborsClassifier()
```

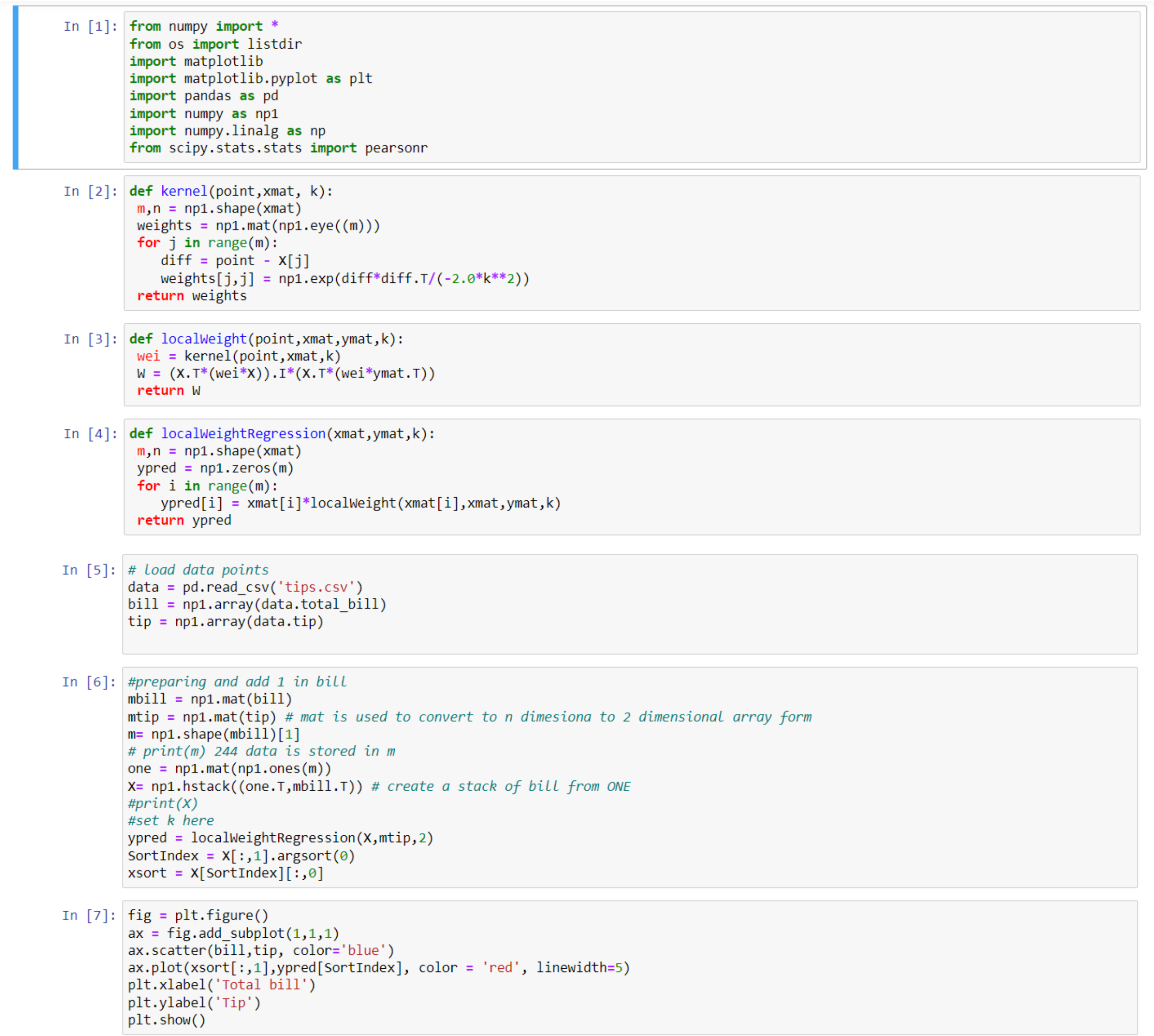
```
In [5]: y_pred=classier.predict(x_test)
```

```
In [6]: print('confusion matrix')
        print(confusion_matrix(y_test,y_pred))

        confusion matrix
        [[16  0  0]
         [ 0 12  1]
         [ 0  0 16]]
```

```
In [7]: print('accuracy')
        print(classification_report(y_test,y_pred))

        accuracy
                      precision    recall  f1-score   support

                   0       1.00      1.00      1.00        16
                   1       1.00      0.92      0.96        13
                   2       0.94      1.00      0.97        16

            accuracy                           0.98        45
           macro avg       0.98      0.97      0.98        45
        weighted avg       0.98      0.98      0.98        45
```

LAB 10

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Selectappropriate data set for your experiment and draw graphs.



```
In [1]: from numpy import *
        from os import listdir
        import matplotlib
        import matplotlib.pyplot as plt
        import pandas as pd
        import numpy as np1
        import numpy.linalg as np
        from scipy.stats.stats import pearsonr
```

```
In [2]: def kernel(point,xmat, k):
         m,n = np1.shape(xmat)
         weights = np1.mat(np1.eye((m)))
         for j in range(m):
            diff = point - X[j]
            weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
         return weights
```

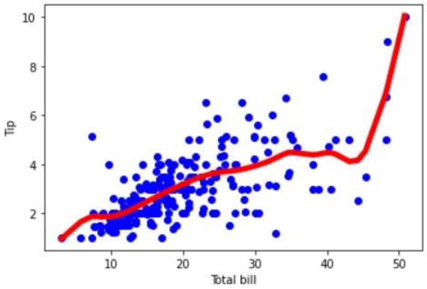
```
In [3]: def localWeight(point,xmat,ymat,k):
         wei = kernel(point,xmat,k)
         W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
         return W
```

```
In [4]: def localWeightRegression(xmat,ymat,k):
         m,n = np1.shape(xmat)
         ypred = np1.zeros(m)
         for i in range(m):
            ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
         return ypred
```

```
In [5]: # load data points
        data = pd.read_csv('tips.csv')
        bill = np1.array(data.total_bill)
        tip = np1.array(data.tip)
```

```
In [6]: #preparing and add 1 in bill
        mbill = np1.mat(bill)
        mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
        m= np1.shape(mbill)[1]
        # print(m) 244 data is stored in m
        one = np1.mat(np1.ones(m))
        X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
        #print(X)
        #set k here
        ypred = localWeightRegression(X,mtip,2)
        SortIndex = X[:,1].argsort(0)
        xsort = X[SortIndex][:,0]
```

```
In [7]: fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        ax.scatter(bill,tip, color='blue')
        ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
        plt.xlabel('Total bill')
        plt.ylabel('Tip')
        plt.show()
```



```
In [8]: import numpy as np
        from bokeh.plotting import figure, show, output_notebook
        from bokeh.layouts import gridplot
        from bokeh.io import push_notebook
```

```
In [9]: def local_regression(x0, X, Y, tau):# add bias term
            x0 = np.r_[1, x0] # Add one to avoid the loss in information
            X = np.c_[np.ones(len(X)), X]
            # fit model: normal equations with kernel
            xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W
            beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product
            # predict value
            return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
```

```
In [10]: def radial_kernel(x0, X, tau):
             return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
         # Weight or Radial Kernal Bias Function
```

```
In [11]: n = 1000
         # generate dataset
         X = np.linspace(-3, 3, num=n)
         print("The Data Set ( 10 Samples) X :\n",X[1:10])
         Y = np.log(np.abs(X ** 2 - 1) + .5)
         print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
         # jitter X
         X += np.random.normal(scale=.1, size=n)
         print("Normalised (10 Samples) X :\n",X[1:10])
         domain = np.linspace(-3, 3, num=300)
         print(" Xo Domain Space(10 Samples) :\n",domain[1:10])

         The Data Set ( 10 Samples) X :
          [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
          -2.95795796 -2.95195195 -2.94594595]
         The Fitting Curve Data Set (10 Samples) Y :
          [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
          2.11015444 2.10584249 2.10152068]
         Normalised (10 Samples) X :
          [-2.95983905 -2.77699311 -3.06439147 -3.15903005 -3.19868861 -3.00406048
          -2.9445708  -2.87933746 -2.94253902]
          Xo Domain Space(10 Samples) :
          [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
          -2.85953177 -2.83946488 -2.81939799]
```

```
In [12]: def plot_lwr(tau):
         # prediction through regression
             prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
             plot = figure(plot_width=400, plot_height=400)
             plot.title.text='tau=%g' % tau
             plot.scatter(X, Y, alpha=.3)
             plot.line(domain, prediction, line_width=2, color='red')
             return plot
```

```
In [13]: show(gridplot([
         [plot_lwr(10.), plot_lwr(1.)],
         [plot_lwr(0.1), plot_lwr(0.01)]]))
```

```
In [ ]:
```