ML Lab Record

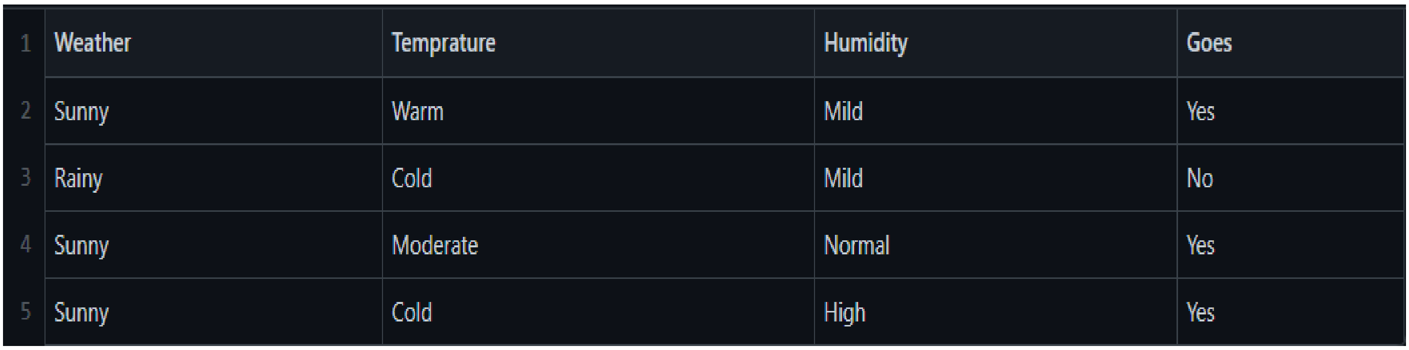
Sagar Reddy S N

1BM18CS156

Lab 1: Find S Algorithm

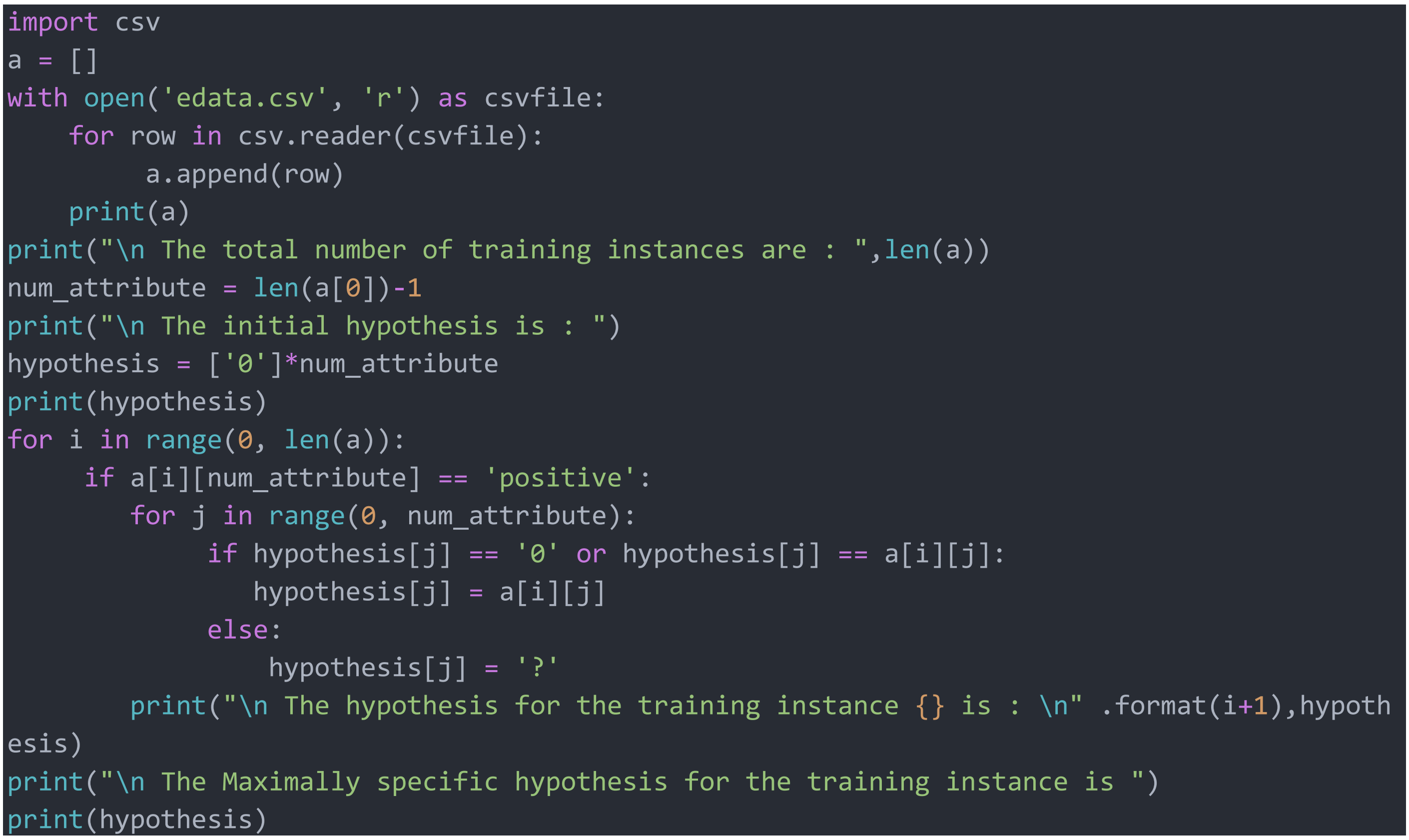
Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

Dataset:



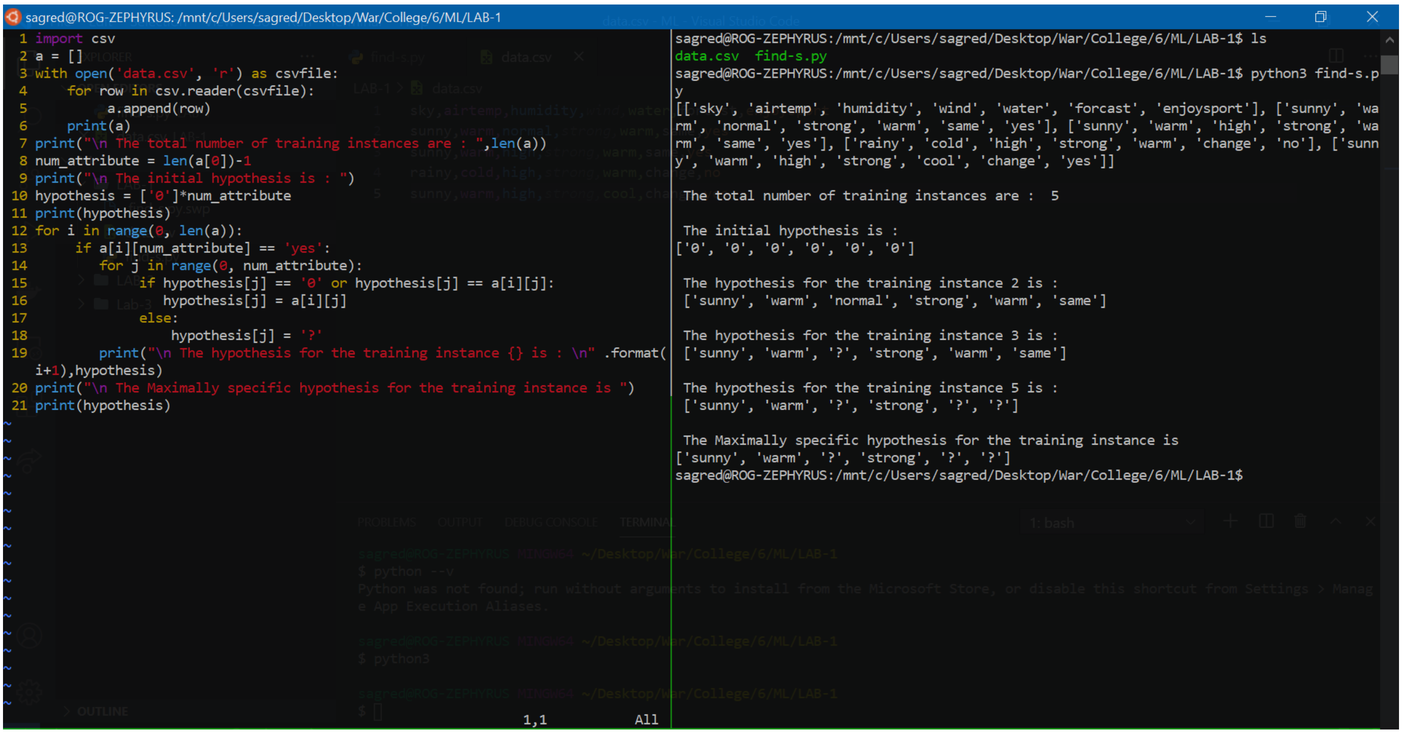
Weather	Temprature	Humidity	Goes
Sunny	Warm	Mild	Yes
Rainy	Cold	Mild	No
Sunny	Moderate	Normal	Yes
Sunny	Cold	High	Yes

Code:



```
import csv
a = []
with open('edata.csv', 'r') as csvfile:
    for row in csv.reader(csvfile):
         a.append(row)
    print(a)
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 i in range(0, len(a)):
     if a[i][num_attribute] == 'positive':
        for j in range(0, num_attribute):
             if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
             else:
                 hypothesis[j] = '?'
        print("\n The hypothesis for the training instance {} is : \n" .format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

Output:



sagred@ROG-ZEPHYRUS: /mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-1

```python
import csv
a = []
with open('data.csv', 'r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)
    print(a)
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 i in range(0, len(a)):
     if a[i][num_attribute] == 'yes':
         for j in range(0, num_attribute):
             if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                 hypothesis[j] = a[i][j]
             else:
                 hypothesis[j] = '?'
         print("\n The hypothesis for the training instance {} is : \n" .format(
i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

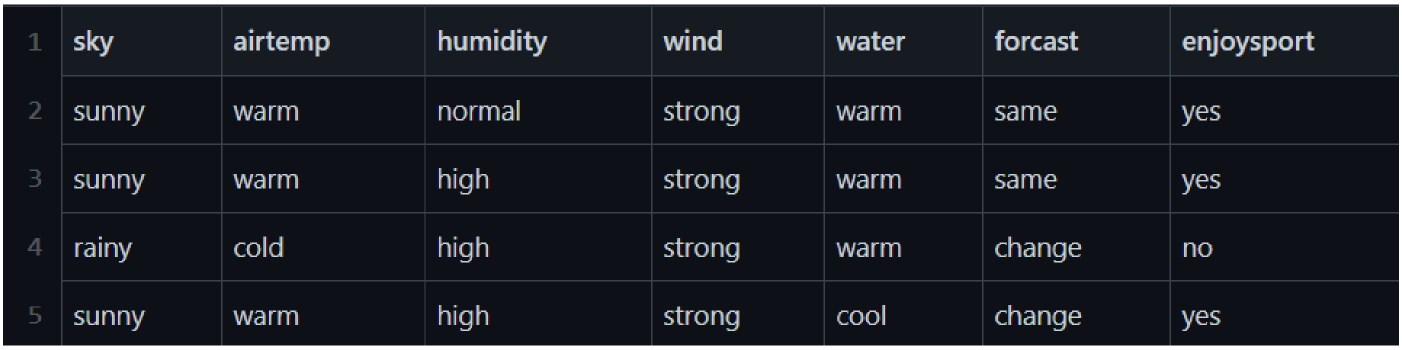
```
1,1           All
```

```
sagred@ROG-ZEPHYRUS:/mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-1$ ls
data.csv  find-s.py
sagred@ROG-ZEPHYRUS:/mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-1$ python3 find-s.p
y
[['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast', 'enjoysport'], ['sunny', 'wa
rm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong', 'wa
rm', 'same', 'yes'], ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'], ['sunn
y', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

 The total number of training instances are :  5

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

 The hypothesis for the training instance 2 is :
 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

 The hypothesis for the training instance 3 is :
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

 The hypothesis for the training instance 5 is :
 ['sunny', 'warm', '?', 'strong', '?', '?']

 The Maximally specific hypothesis for the training instance is
['sunny', 'warm', '?', 'strong', '?', '?']
sagred@ROG-ZEPHYRUS:/mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-1$
```

Lab 2: Candidate Elimination Algorithm

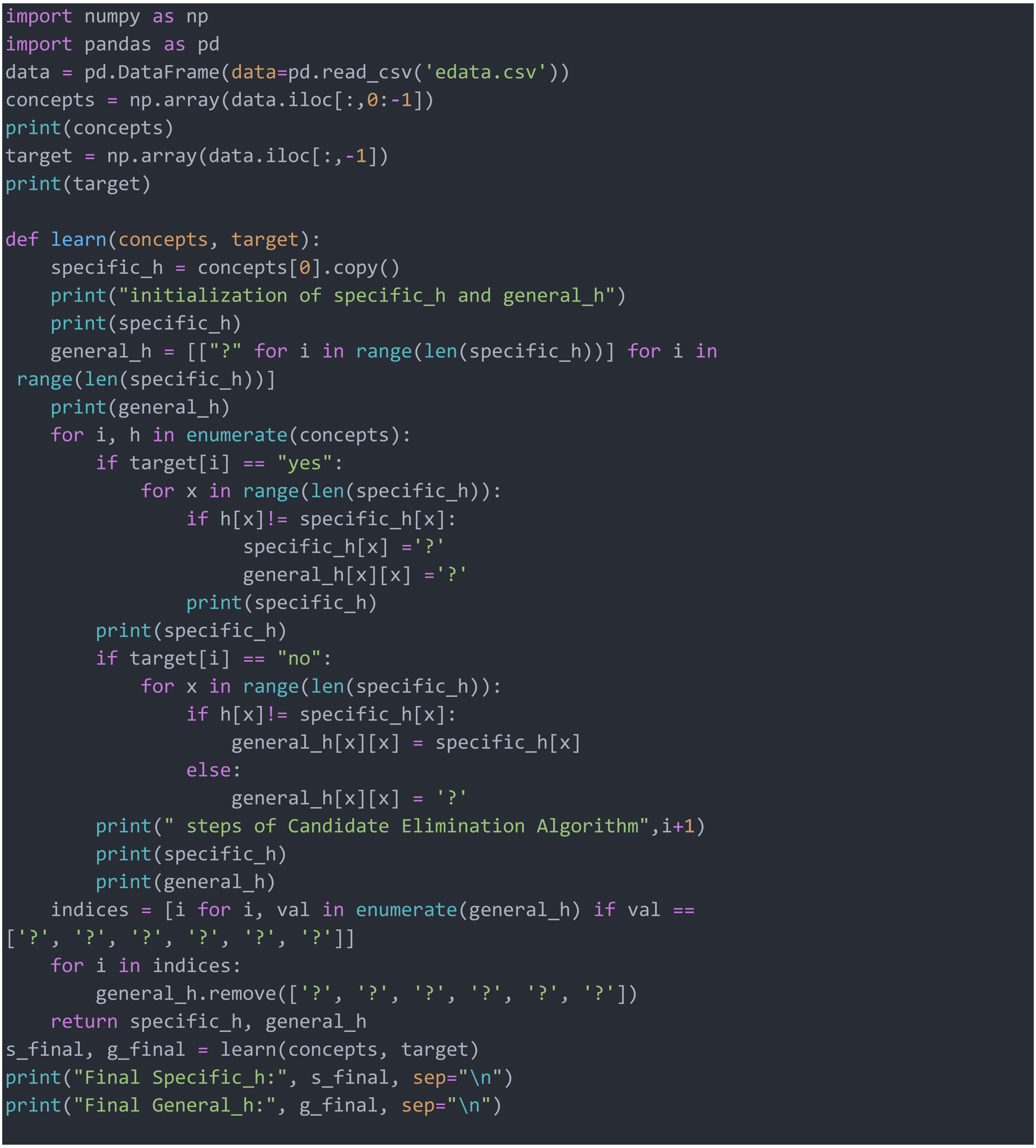
For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Dataset:



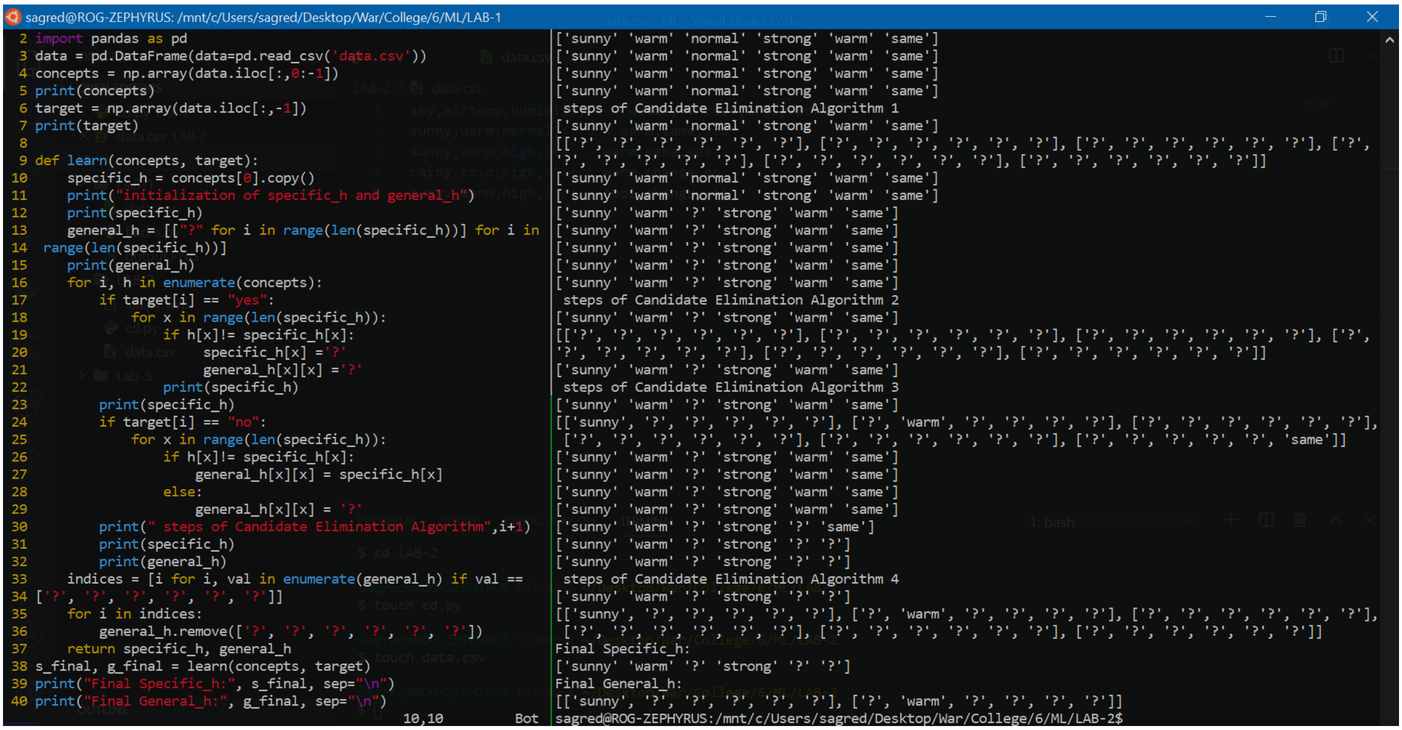
1	sky	airtemp	humidity	wind	water	forcast	enjoysport
2	sunny	warm	normal	strong	warm	same	yes
3	sunny	warm	high	strong	warm	same	yes
4	rainy	cold	high	strong	warm	change	no
5	sunny	warm	high	strong	cool	change	yes

Code:



```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('edata.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")
```

Output:



sagred@ROG-ZEPHYRUS: /mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-1

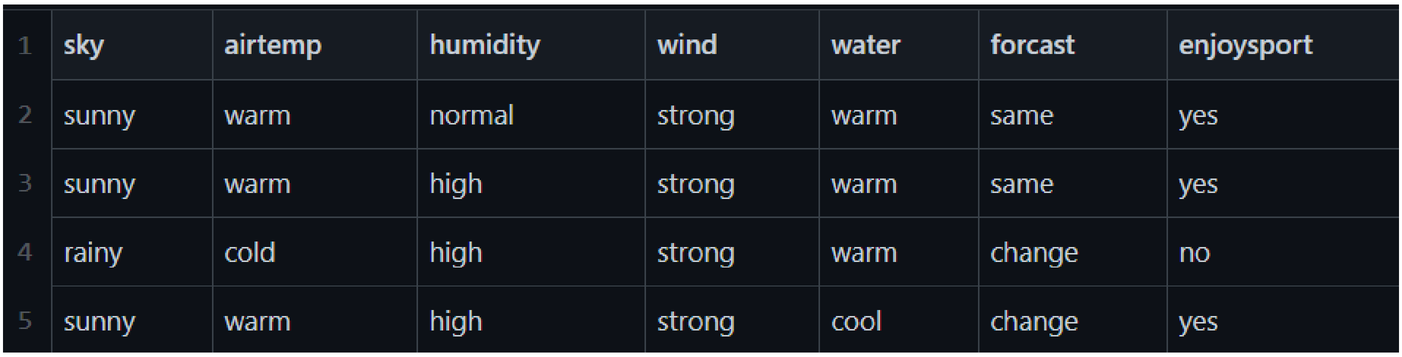
```python
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('data.csv'))
concepts = np.array(data.iloc[:,0:-1])
print(concepts)
target = np.array(data.iloc[:,-1])
print(target)

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")
                                                10,10          Bot
```

```
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
['sunny' 'warm' '?' 'strong' '?' '?']
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
 ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
sagred@ROG-ZEPHYRUS:/mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-2$
```

Lab 3: Decision Tree

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.

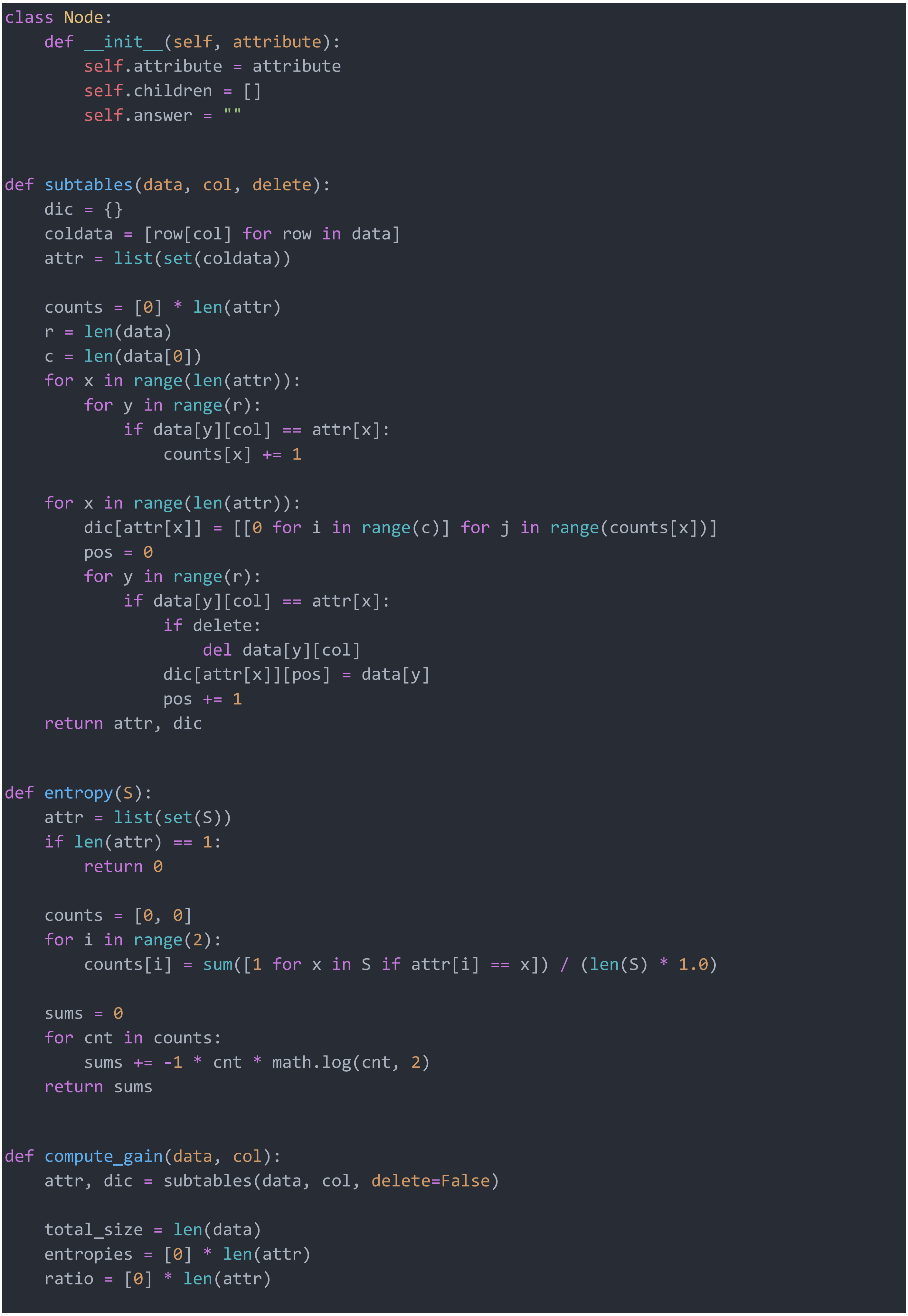
Dataset:



1	sky	airtemp	humidity	wind	water	forcast	enjoysport
2	sunny	warm	normal	strong	warm	same	yes
3	sunny	warm	high	strong	warm	same	yes
4	rainy	cold	high	strong	warm	change	no
5	sunny	warm	high	strong	cool	change	yes

Code:

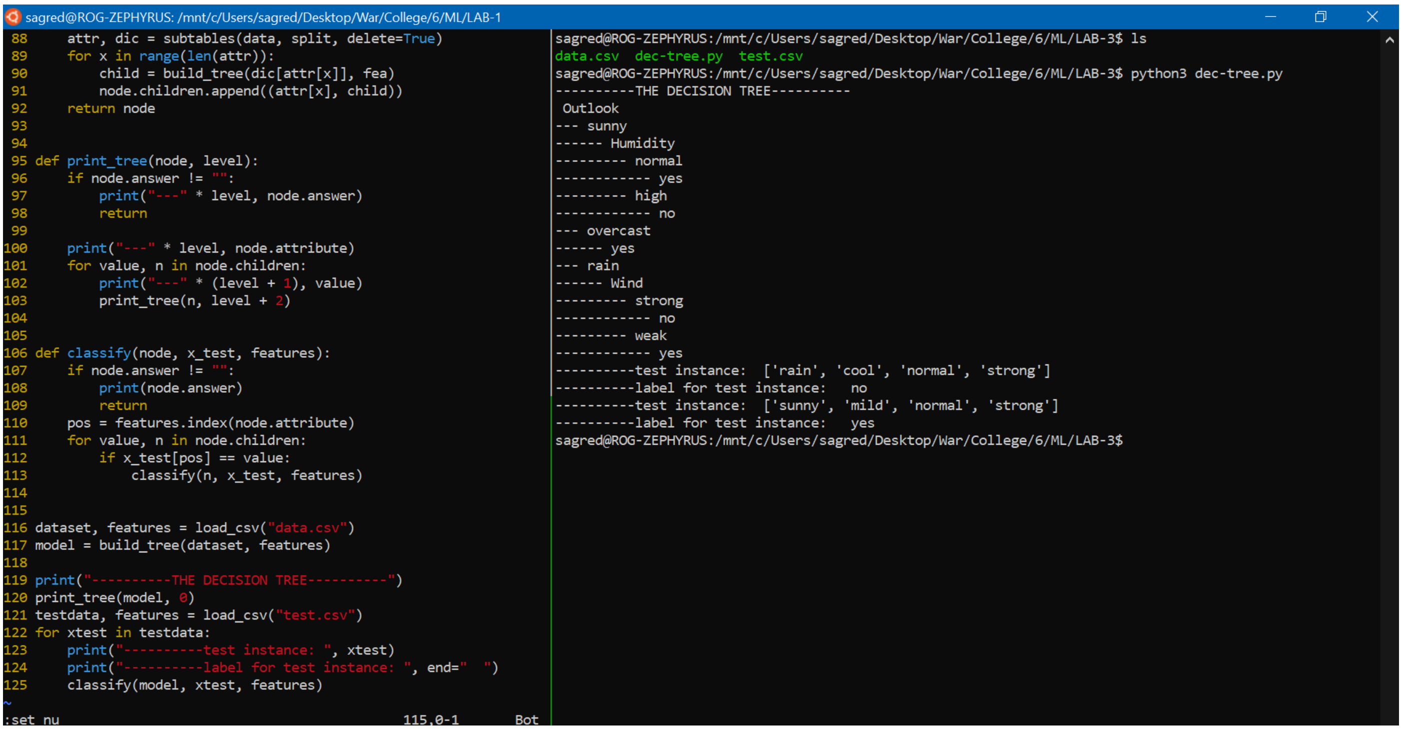
```python
import math
import csv

def load_csv(filename):
    lines = csv.reader(open(filename, "r"))
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset, headers
```



```python
class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = ""


def subtables(data, col, delete):
    dic = {}
    coldata = [row[col] for row in data]
    attr = list(set(coldata))

    counts = [0] * len(attr)
    r = len(data)
    c = len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col] == attr[x]:
                counts[x] += 1

    for x in range(len(attr)):
        dic[attr[x]] = [[0 for i in range(c)] for j in range(counts[x])]
        pos = 0
        for y in range(r):
            if data[y][col] == attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos] = data[y]
                pos += 1
    return attr, dic


def entropy(S):
    attr = list(set(S))
    if len(attr) == 1:
        return 0

    counts = [0, 0]
    for i in range(2):
        counts[i] = sum([1 for x in S if attr[i] == x]) / (len(S) * 1.0)

    sums = 0
    for cnt in counts:
        sums += -1 * cnt * math.log(cnt, 2)
    return sums


def compute_gain(data, col):
    attr, dic = subtables(data, col, delete=False)

    total_size = len(data)
    entropies = [0] * len(attr)
    ratio = [0] * len(attr)
    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x] = len(dic[attr[x]]) / (total_size * 1.0)
        entropies[x] = entropy([row[-1] for row in dic[attr[x]]])
        total_entropy -= ratio[x] * entropies[x]
    return total_entropy


def build_tree(data, features):
    lastcol = [row[-1] for row in data]
    if (len(set(lastcol))) == 1:
        node = Node("")
        node.answer = lastcol[0]
        return node

    n = len(data[0]) - 1
    gains = [0] * n
    for col in range(n):
        gains[col] = compute_gain(data, col)
    split = gains.index(max(gains))
    node = Node(features[split])
    fea = features[:split] + features[split + 1:]

    attr, dic = subtables(data, split, delete=True)
    for x in range(len(attr)):
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))
    return node


def print_tree(node, level):
    if node.answer != "":
        print("---" * level, node.answer)
        return

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


def classify(node, x_test, features):
    if node.answer != "":
        print(node.answer)
        return
    pos = features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos] == value:
            classify(n, x_test, features)


'''Main Program'''
dataset, features = load_csv("edata.csv")
model = build_tree(dataset, features)
```

```python
print("----------THE DECISION TREE----------")
print_tree(model, 0)
testdata, features = load_csv("test.csv")
for xtest in testdata:
    print("----------test instance: ", xtest)
    print("----------label for test instance: ", end="  ")
    classify(model, xtest, features)
```

Output:


sagred@ROG-ZEPHYRUS: /mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-1

```python
    attr, dic = subtables(data, split, delete=True)
    for x in range(len(attr)):
        child = build_tree(dic[attr[x]], fea)
        node.children.append((attr[x], child))
    return node


def print_tree(node, level):
    if node.answer != "":
        print("---" * level, node.answer)
        return

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


def classify(node, x_test, features):
    if node.answer != "":
        print(node.answer)
        return
    pos = features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos] == value:
            classify(n, x_test, features)


dataset, features = load_csv("data.csv")
model = build_tree(dataset, features)

print("----------THE DECISION TREE----------")
print_tree(model, 0)
testdata, features = load_csv("test.csv")
for xtest in testdata:
    print("----------test instance: ", xtest)
    print("----------label for test instance: ", end="  ")
    classify(model, xtest, features)
~
:set nu                                                115,0-1       Bot
```

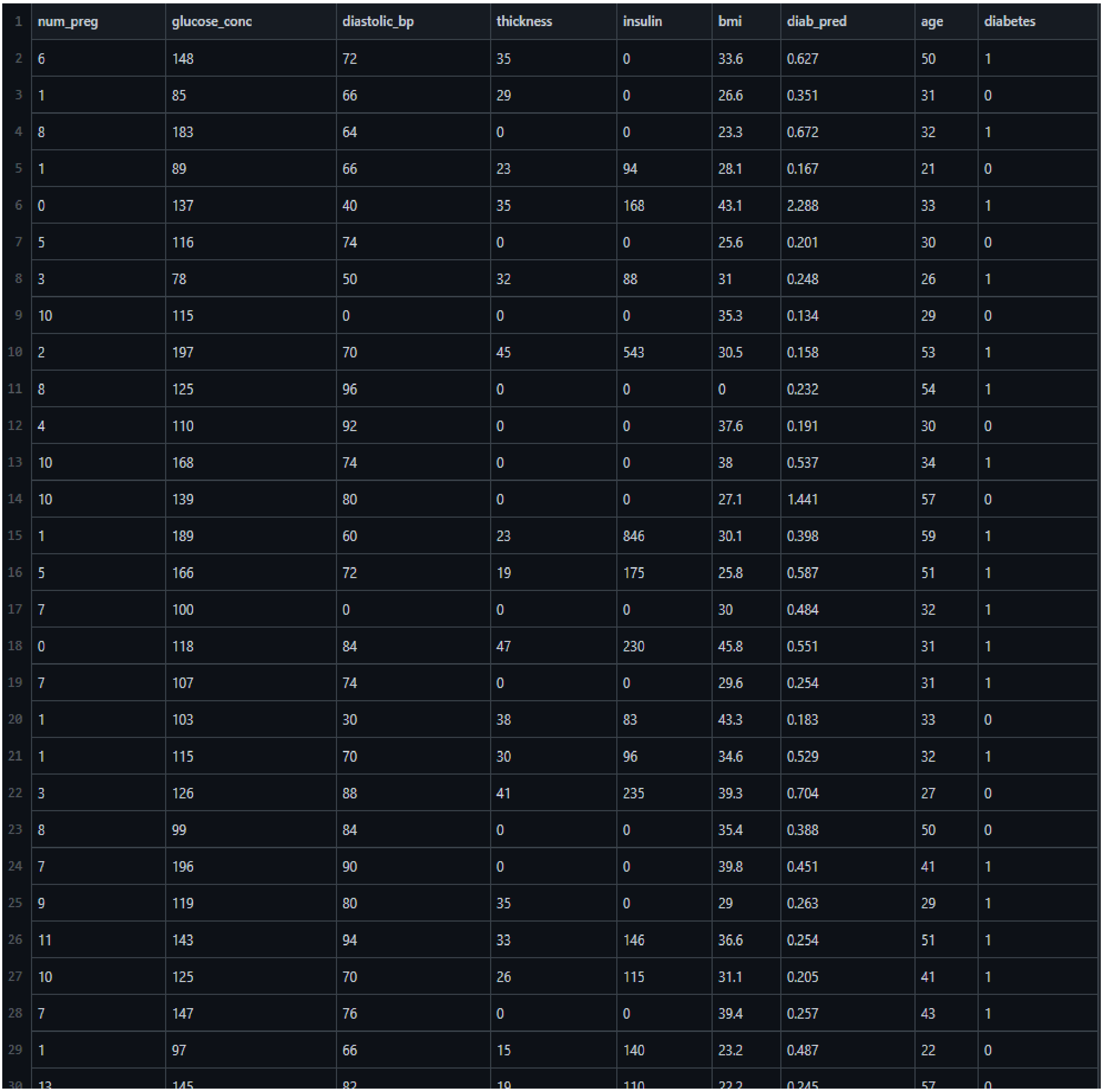
```
sagred@ROG-ZEPHYRUS:/mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-3$ ls
data.csv  dec-tree.py  test.csv
sagred@ROG-ZEPHYRUS:/mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-3$ python3 dec-tree.py
----------THE DECISION TREE----------
 Outlook
--- sunny
------ Humidity
--------- normal
------------ yes
--------- high
------------ no
--- overcast
------ yes
--- rain
------ Wind
--------- strong
------------ no
--------- weak
------------ yes
----------test instance:  ['rain', 'cool', 'normal', 'strong']
----------label for test instance:   no
----------test instance:  ['sunny', 'mild', 'normal', 'strong']
----------label for test instance:   yes
sagred@ROG-ZEPHYRUS:/mnt/c/Users/sagred/Desktop/War/College/6/ML/LAB-3$
```

Lab 4: Naïve Bayesian Classifier

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

Naïve Bayesian Classifier Example 1:

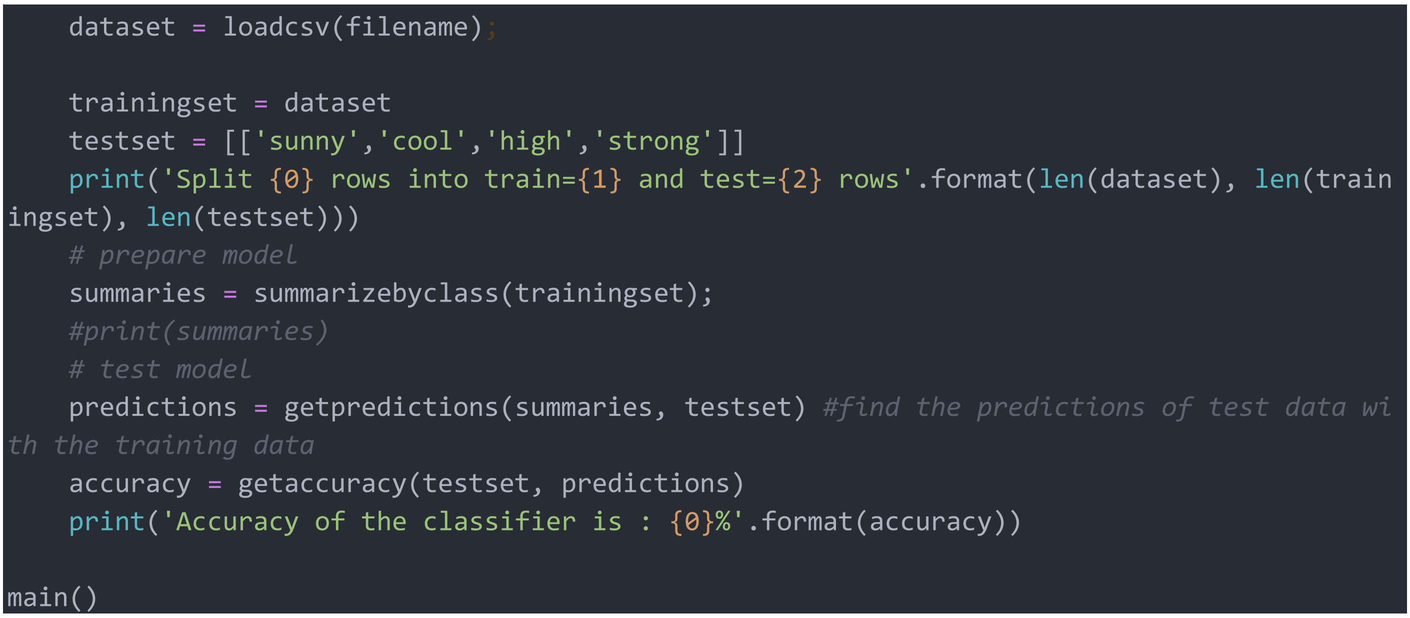
Dataset:



num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
11	143	94	33	146	36.6	0.254	51	1
10	125	70	26	115	31.1	0.205	41	1
7	147	76	0	0	39.4	0.257	43	1
1	97	66	15	140	23.2	0.487	22	0
13	145	82	19	110	22.2	0.245	57	0

Code:

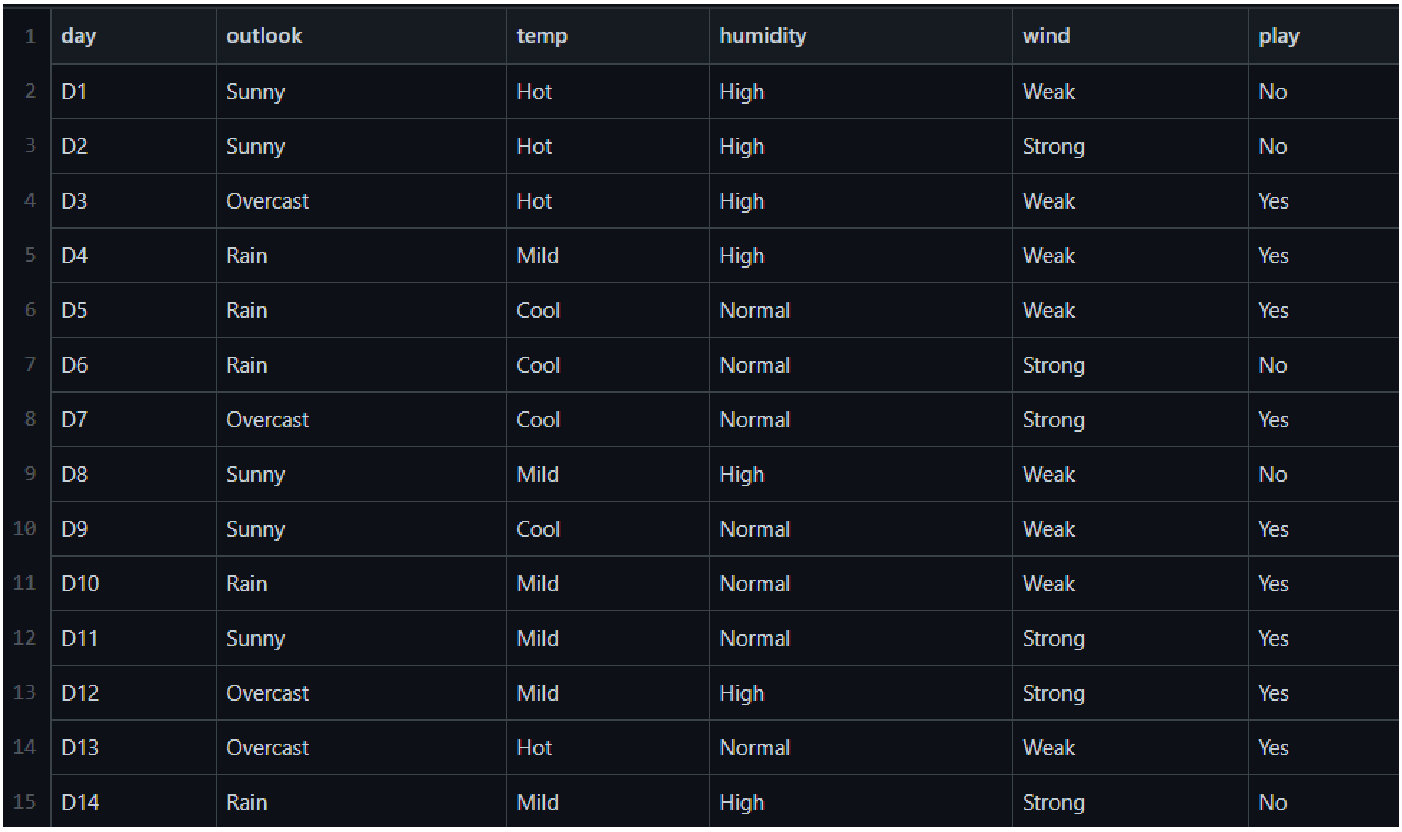
```
import csv
import random
import math

def loadcsv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    for i in range(len(dataset)):
       #converting strings into numbers for processing
        dataset[i] = [float(x) for x in dataset[i]]

    return dataset

def splitdataset(dataset, splitratio):
    #67% training size
    trainsize = int(len(dataset) * splitratio);
    trainset = []
    copy = list(dataset);
    while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for training data
        index = random.randrange(len(copy));
        trainset.append(copy.pop(index))
    return [trainset, copy]

def separatebyclass(dataset):
    separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

def summarize(dataset): #creates a dictionary of classes
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
    del summaries[-1] #excluding labels +ve or -ve
    return summaries

def summarizebyclass(dataset):
    separated = separatebyclass(dataset);
    #print(separated)
    summaries = {}
    for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
        summaries[classvalue] = summarize(instances) #summarize is used to cal to mean and std
    return summaries

def calculateprobability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateclassprobabilities(summaries, inputvector):
    probabilities = {} # probabilities contains the all prob of all class of test data
    for classvalue, classsummaries in summaries.items():#class and attribute information as mean and sd
        probabilities[classvalue] = 1
        for i in range(len(classsummaries)):
            mean, stdev = classsummaries[i] #take mean and sd of every attribute for class 0 and 1 seperaely
            x = inputvector[i] #testvector's first attribute
            probabilities[classvalue] *= calculateprobability(x, mean, stdev);#use normal dist
    return probabilities

def predict(summaries, inputvector): #training and test data is passed
    probabilities = calculateclassprobabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():#assigns that class which has he highest prob
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel

def getpredictions(summaries, testset):
    predictions = []
    for i in range(len(testset)):
        result = predict(summaries, testset[i])
        predictions.append(result)
    return predictions

def getaccuracy(testset, predictions):
    correct = 0
    for i in range(len(testset)):
        if testset[i][-1] == predictions[i]:
            correct += 1
    return (correct/float(len(testset))) * 100.0

def main():
    filename = 'edata.csv'
    splitratio = 0.67
```



```
    dataset = loadcsv(filename);

    trainingset = dataset
    testset = [['sunny','cool','high','strong']]
    print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset), len(testset)))
    # prepare model
    summaries = summarizebyclass(trainingset);
    #print(summaries)
    # test model
    predictions = getpredictions(summaries, testset) #find the predictions of test data with the training data
    accuracy = getaccuracy(testset, predictions)
    print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

Naïve Bayesian Classifier Example 2:

Dataset:

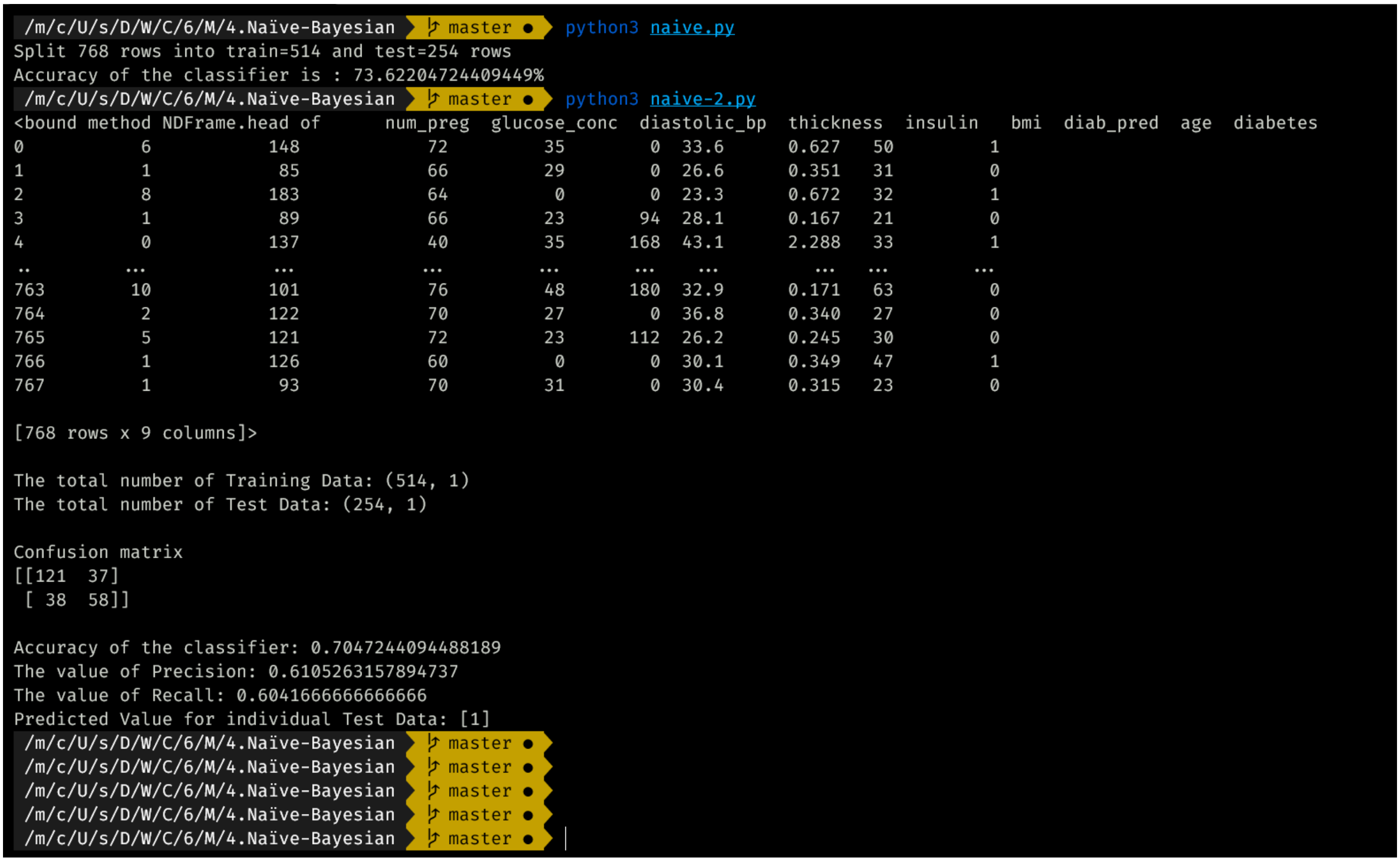


day	outlook	temp	humidity	wind	play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Code:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn import preprocessing

dataf = pd.read_csv("./edata.csv")
feature_col_names = ['outlook','temp','humidity','wind']
predicted_class_names = ['play']

def MultiLabelEncoder(columnlist,dataframe):
    for i in columnlist:
        labelencoder_X=preprocessing.LabelEncoder()
        dataframe[i]=labelencoder_X.fit_transform(dataframe[i])
    return dataframe
le = preprocessing.LabelEncoder()
feature_col = ['outlook','temp','humidity','wind','play']

Xdata = MultiLabelEncoder(feature_col,dataf)
X = Xdata[feature_col_names]

yy = dataf[predicted_class_names]


y = Xdata[predicted_class_names]
print(dataf.head)

xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)
print ('\nThe total number of Training Data:',ytrain.shape)
print ('The total number of Test Data:',ytest.shape)

print(xtrain,ytrain)
classif = GaussianNB().fit(xtrain,ytrain)
print(classif)
predicted = classif.predict(xtest)
pri_enc = le.fit_transform(['sunny','cool','high','strong'])

predictTestData= classif.predict([pri_enc])

print('\nConfusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\nAccuracy of the classifier:',metrics.accuracy_score(ytest,predicted))

print('The value of Precision:', metrics.precision_score(ytest,predicted))

print('The value of Recall:', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

Output:

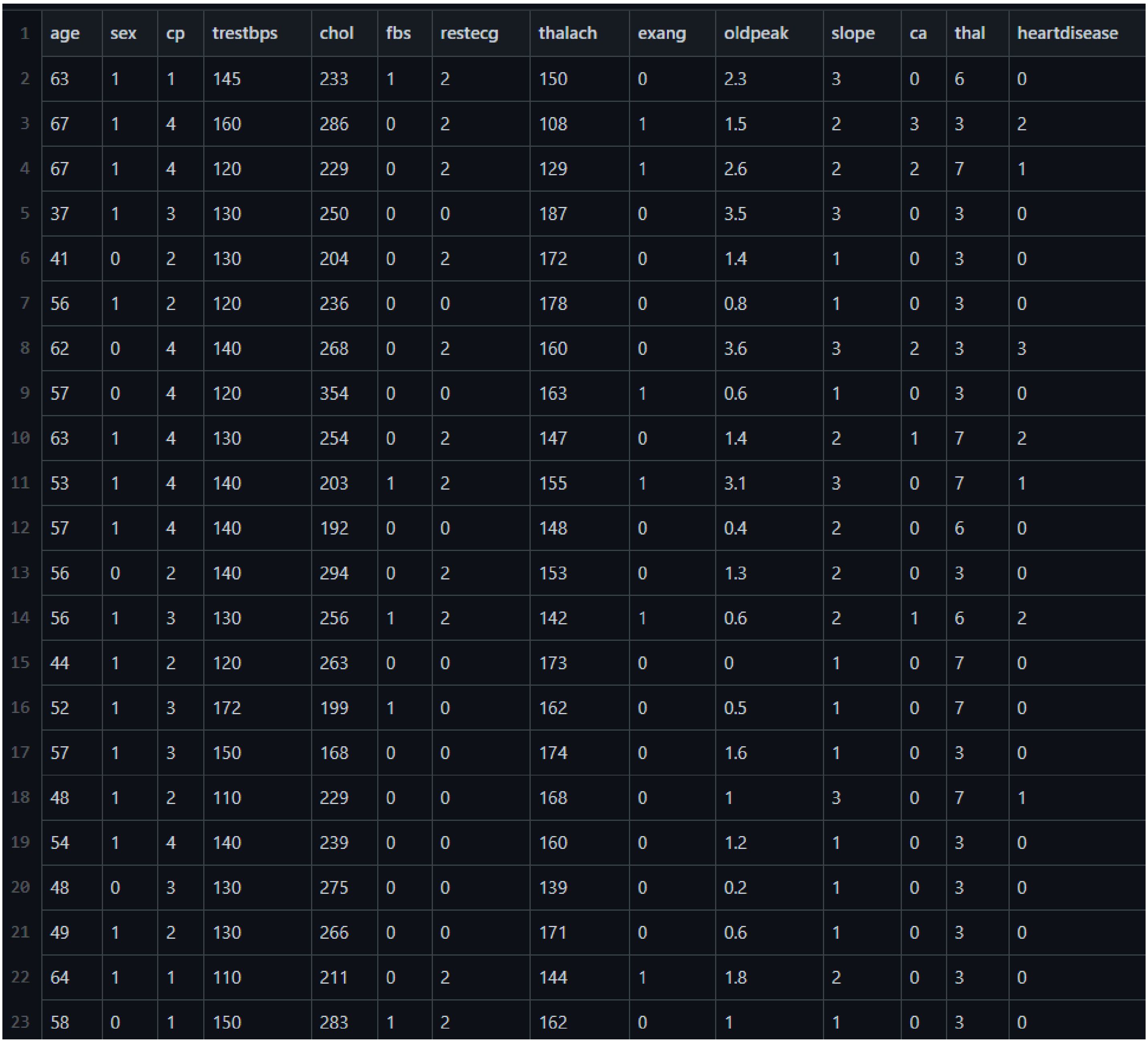


```
 /m/c/U/s/D/W/C/6/M/4.Naïve-Bayesian  master ●  python3 naive.py
Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is : 73.62204724409449%
 /m/c/U/s/D/W/C/6/M/4.Naïve-Bayesian  master ●  python3 naive-2.py
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  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
..        ...           ...          ...        ...      ...   ...        ...  ...        ...
763        10           101           76         48      180  32.9      0.171    63          0
764         2           122           70         27        0  36.8      0.340    27          0
765         5           121           72         23      112  26.2      0.245    30          0
766         1           126           60          0        0  30.1      0.349    47          1
767         1            93           70         31        0  30.4      0.315    23          0

[768 rows x 9 columns]>

The total number of Training Data: (514, 1)
The total number of Test Data: (254, 1)

Confusion matrix
[[121  37]
 [ 38  58]]

Accuracy of the classifier: 0.7047244094488189
The value of Precision: 0.6105263157894737
The value of Recall: 0.6041666666666666
Predicted Value for individual Test Data: [1]
 /m/c/U/s/D/W/C/6/M/4.Naïve-Bayesian  master ●
 /m/c/U/s/D/W/C/6/M/4.Naïve-Bayesian  master ●
 /m/c/U/s/D/W/C/6/M/4.Naïve-Bayesian  master ●
 /m/c/U/s/D/W/C/6/M/4.Naïve-Bayesian  master ●
 /m/c/U/s/D/W/C/6/M/4.Naïve-Bayesian  master ●
```

Lab 5: Bayesian Network

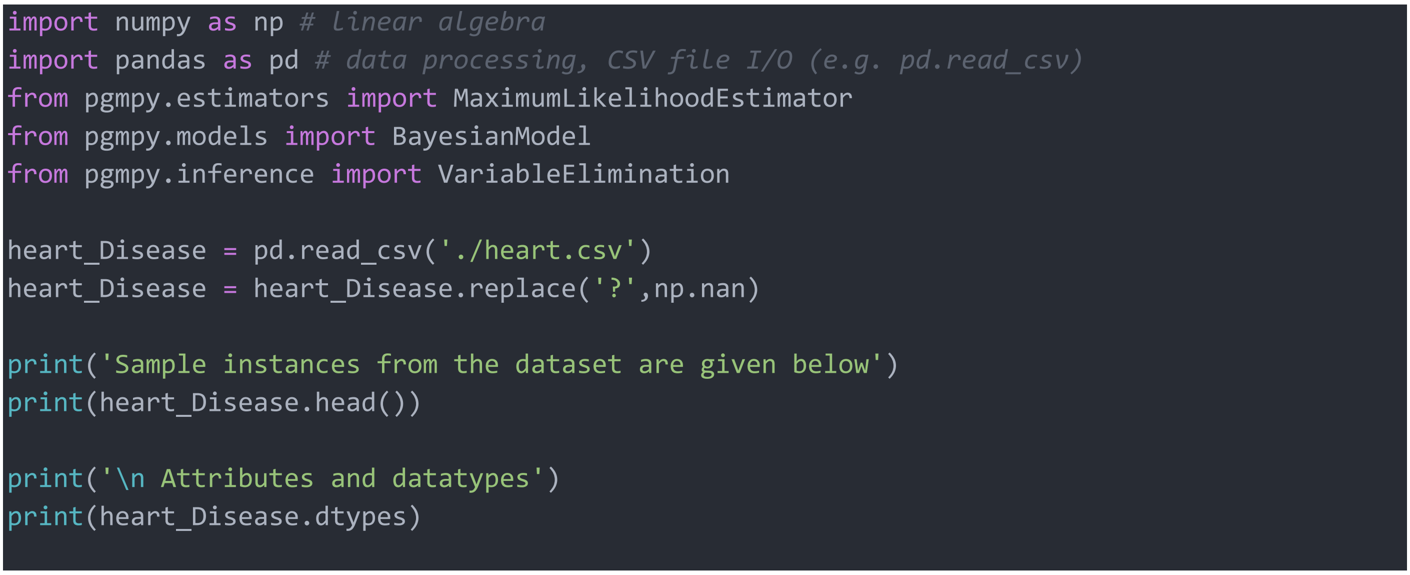
Write a program to construct a Bayesian network considering training data. Use this model to make predictions

Dataset:

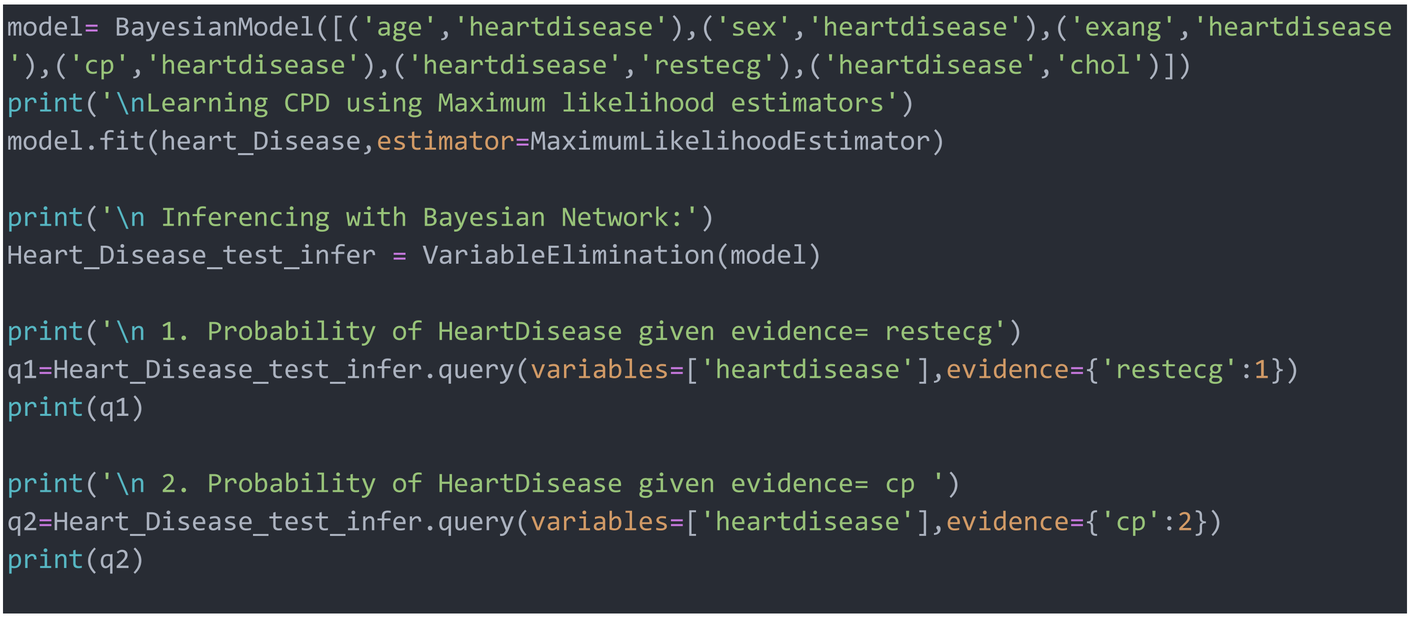


age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
53	1	4	140	203	1	2	155	1	3.1	3	0	7	1
57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
56	1	3	130	256	1	2	142	1	0.6	2	1	6	2
44	1	2	120	263	0	0	173	0	0	1	0	7	0
52	1	3	172	199	1	0	162	0	0.5	1	0	7	0
57	1	3	150	168	0	0	174	0	1.6	1	0	3	0
48	1	2	110	229	0	0	168	0	1	3	0	7	1
54	1	4	140	239	0	0	160	0	1.2	1	0	3	0
48	0	3	130	275	0	0	139	0	0.2	1	0	3	0
49	1	2	130	266	0	0	171	0	0.6	1	0	3	0
64	1	1	110	211	0	2	144	1	1.8	2	0	3	0
58	0	1	150	283	1	2	162	0	1	1	0	3	0

Code:



```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

heart_Disease = pd.read_csv('./heart.csv')
heart_Disease = heart_Disease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heart_Disease.head())

print('\n Attributes and datatypes')
print(heart_Disease.dtypes)
```



```python
model= BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease
'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heart_Disease,estimator=MaximumLikelihoodEstimator)

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

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=Heart_Disease_test_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=Heart_Disease_test_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

Output:

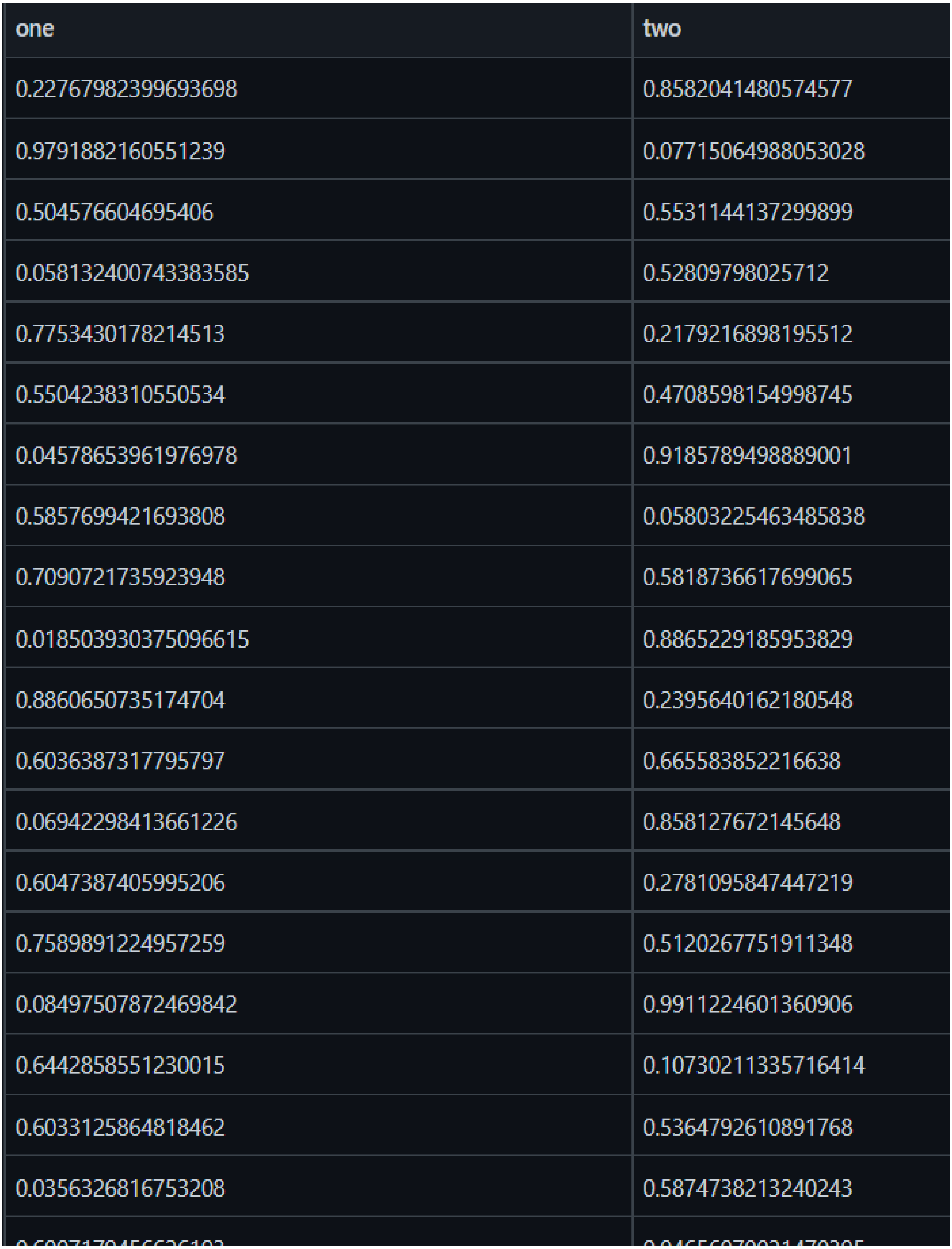
```
/m/c/U/s/D/W/C/6/M/5.Bayesian_Network ⍰ ⍰ master ● ⍰ python3 Bayesian_Network.py
Sample instances from the dataset are given below
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope ca thal  heartdisease
0   63    1   1       145   233    1        2      150      0      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

 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              object
thal            object
heartdisease     int64
dtype: object

Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg
Finding Elimination Order: : 100%|██████████████████████████████████████████████████
████████████████████████████████████| 5/5 [00:00<00:00, 2702.86it/s]
```

```
Eliminating: cp: 100%|[redacted]| 5/5 [00:00<00:00, 251.50it/s]
+-----------------+--------------------+                                                    | 0/5 [00:00<?, ?it/s]
| heartdisease    |   phi(heartdisease) |
+=================+====================+
| heartdisease(0) |             0.1012 |
+-----------------+--------------------+
| heartdisease(1) |             0.0000 |
+-----------------+--------------------+
| heartdisease(2) |             0.2392 |
+-----------------+--------------------+
| heartdisease(3) |             0.2015 |
+-----------------+--------------------+
| heartdisease(4) |             0.4581 |
+-----------------+--------------------+

 2. Probability of HeartDisease given evidence= cp
Finding Elimination Order: : 100%|[redacted]| 5/5 [00:00<00:00, 3916.25it/s]
Eliminating: restecg: 100%|[redacted]| 5/5 [00:00<00:00, 487.52it/s]
+-----------------+--------------------+                                                    | 0/5 [00:00<?, ?it/s]
| heartdisease    |   phi(heartdisease) |
+=================+====================+
| heartdisease(0) |             0.3610 |
+-----------------+--------------------+
| heartdisease(1) |             0.2159 |
+-----------------+--------------------+
| heartdisease(2) |             0.1373 |
+-----------------+--------------------+
| heartdisease(3) |             0.1537 |
+-----------------+--------------------+
| heartdisease(4) |             0.1321 |
+-----------------+--------------------+
```

Lab 6: K Means

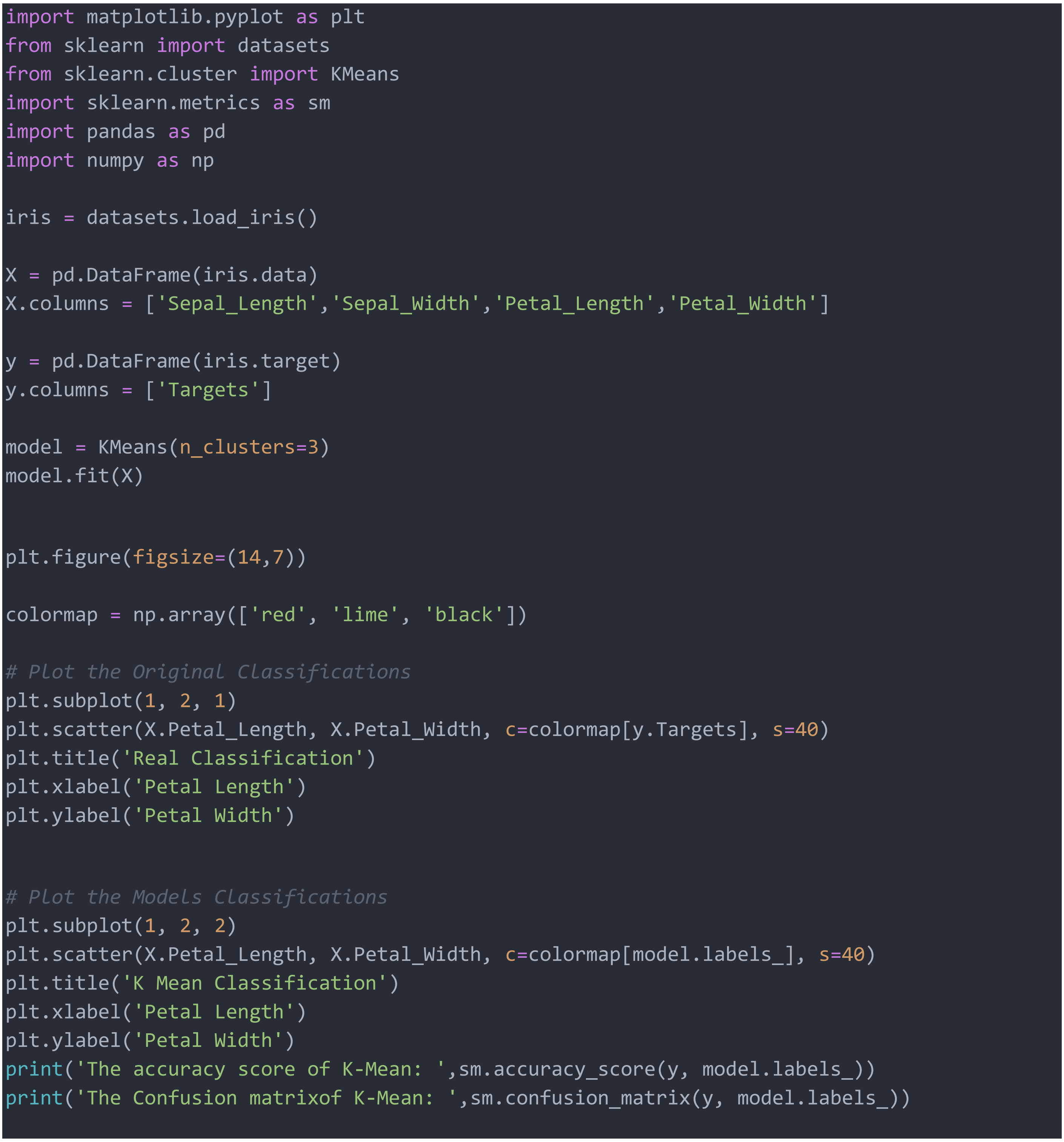
Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

Dataset:



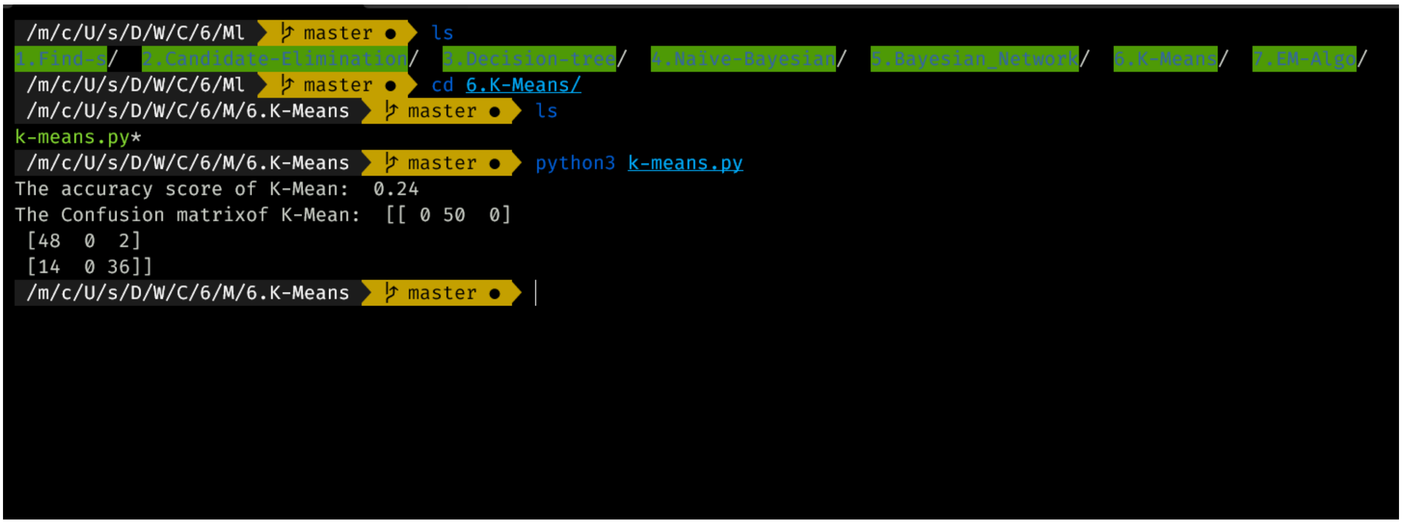
one	two
0.22767982399693698	0.8582041480574577
0.9791882160551239	0.07715064988053028
0.504576604695406	0.5531144137299899
0.058132400743383585	0.52809798025712
0.7753430178214513	0.2179216898195512
0.5504238310550534	0.4708598154998745
0.04578653961976978	0.9185789498889001
0.5857699421693808	0.05803225463485838
0.7090721735923948	0.5818736617699065
0.018503930375096615	0.8865229185953829
0.8860650735174704	0.2395640162180548
0.6036387317795797	0.665583852216638
0.06942298413661226	0.858127672145648
0.6047387405995206	0.2781095847447219
0.7589891224957259	0.5120267751911348
0.08497507872469842	0.9911224601360906
0.6442858551230015	0.10730211335716414
0.6033125864818462	0.5364792610891768
0.0356326816753208	0.5874738213240243
0.6007170456626103	0.04656070021470305

Code:



```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)


plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))
```

Output:



```
 /m/c/U/s/D/W/C/6/Ml   master ●  ls
1.Find-s/  2.Candidate-Elimination/  3.Decision-tree/  4.Naïve-Bayesian/  5.Bayesian_Network/  6.K-Means/  7.EM-Algo/
 /m/c/U/s/D/W/C/6/Ml   master ●  cd 6.K-Means/
 /m/c/U/s/D/W/C/6/M/6.K-Means   master ●  ls
k-means.py*
 /m/c/U/s/D/W/C/6/M/6.K-Means   master ●  python3 k-means.py
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
 /m/c/U/s/D/W/C/6/M/6.K-Means   master ●  |
```

Lab 7: EM Algorithm

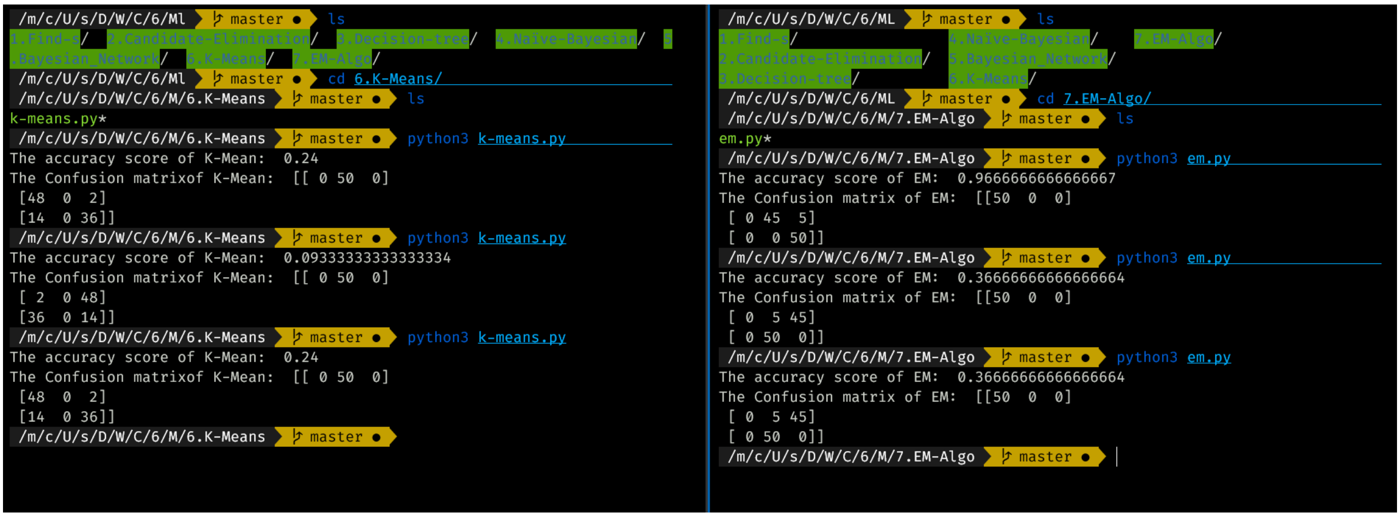
Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the res ults of k-Means algorithm and EM algorithm.

Code:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)


plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
```

```
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

Output:



```
 /m/c/U/s/D/W/C/6/Ml  master ●  ls
1.Find-s/  2.Candidate-Elimination/  3.Decision-tree/   4.Naïve-Bayesian/   5
.Bayesian_Network/  6.K-Means/  7.EM-Algo/
 /m/c/U/s/D/W/C/6/Ml  master ●  cd 6.K-Means/
 /m/c/U/s/D/W/C/6/M/6.K-Means  master ●  ls
k-means.py*
 /m/c/U/s/D/W/C/6/M/6.K-Means  master ●  python3 k-means.py
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
 /m/c/U/s/D/W/C/6/M/6.K-Means  master ●  python3 k-means.py
The accuracy score of K-Mean:  0.09333333333333334
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [ 2  0 48]
 [36  0 14]]
 /m/c/U/s/D/W/C/6/M/6.K-Means  master ●  python3 k-means.py
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
 /m/c/U/s/D/W/C/6/M/6.K-Means  master ●
```

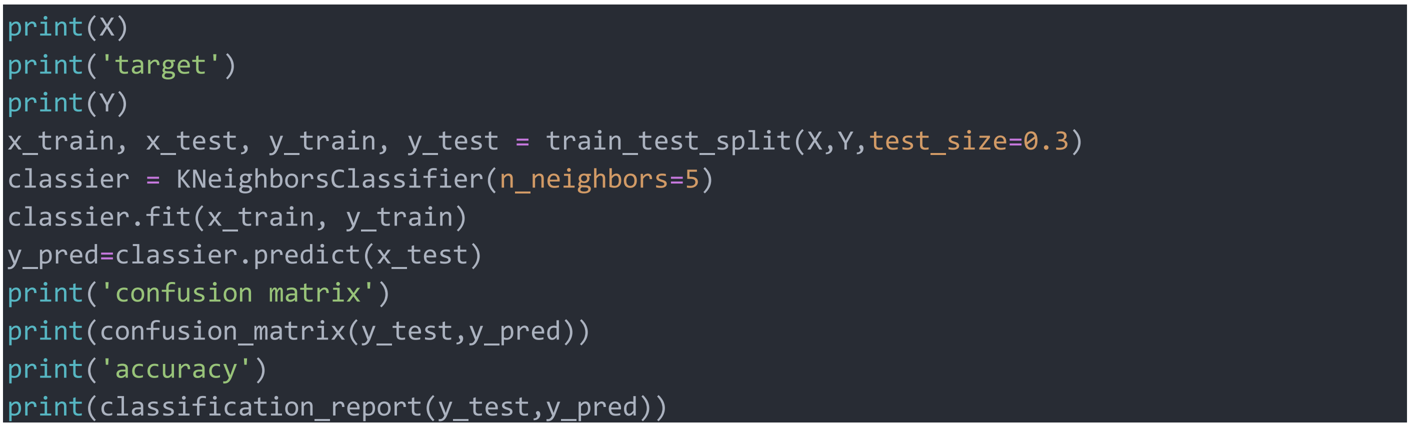
```
 /m/c/U/s/D/W/C/6/ML  master ●  ls
1.Find-s/                  4.Naïve-Bayesian/    7.EM-Algo/
2.Candidate-Elimination/   5.Bayesian_Network/
3.Decision-tree/           6.K-Means/
 /m/c/U/s/D/W/C/6/ML  master ●  cd 7.EM-Algo/
 /m/c/U/s/D/W/C/6/M/7.EM-Algo  master ●  ls
em.py*
 /m/c/U/s/D/W/C/6/M/7.EM-Algo  master ●  python3 em.py
The accuracy score of EM:  0.9666666666666667
The Confusion matrix of EM:  [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
 /m/c/U/s/D/W/C/6/M/7.EM-Algo  master ●  python3 em.py
The accuracy score of EM:  0.36666666666666664
The Confusion matrix of EM:  [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
 /m/c/U/s/D/W/C/6/M/7.EM-Algo  master ●  python3 em.py
The accuracy score of EM:  0.36666666666666664
The Confusion matrix of EM:  [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
 /m/c/U/s/D/W/C/6/M/7.EM-Algo  master ●  |
```

Lab 8: K Nearest

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

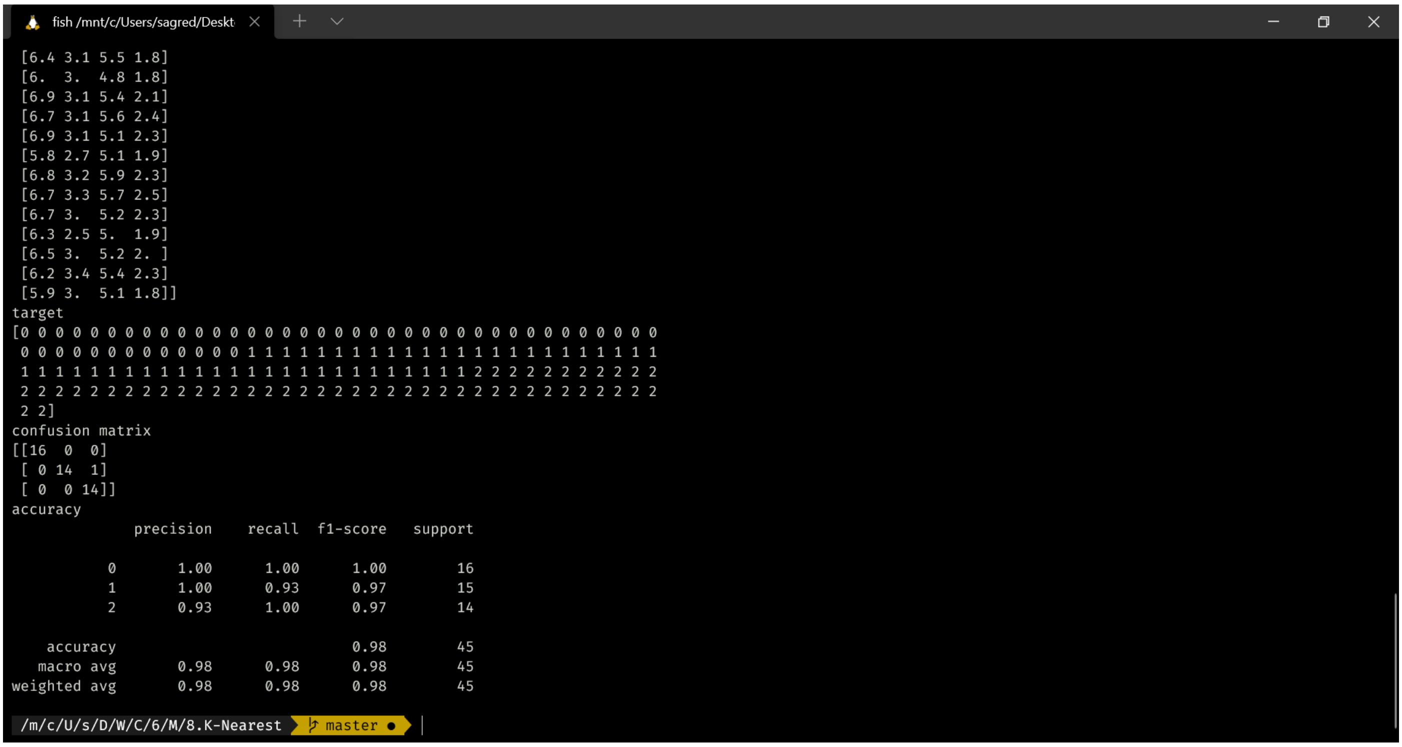
Code:

```
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
iris = datasets.load_iris()
X = iris.data
Y = iris.target
print('sepal-length','sepal-width','petal-length','petal-width')
```



```python
print(X)
print('target')
print(Y)
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.3)
classier = KNeighborsClassifier(n_neighbors=5)
classier.fit(x_train, y_train)
y_pred=classier.predict(x_test)
print('confusion matrix')
print(confusion_matrix(y_test,y_pred))
print('accuracy')
print(classification_report(y_test,y_pred))
```

Output:

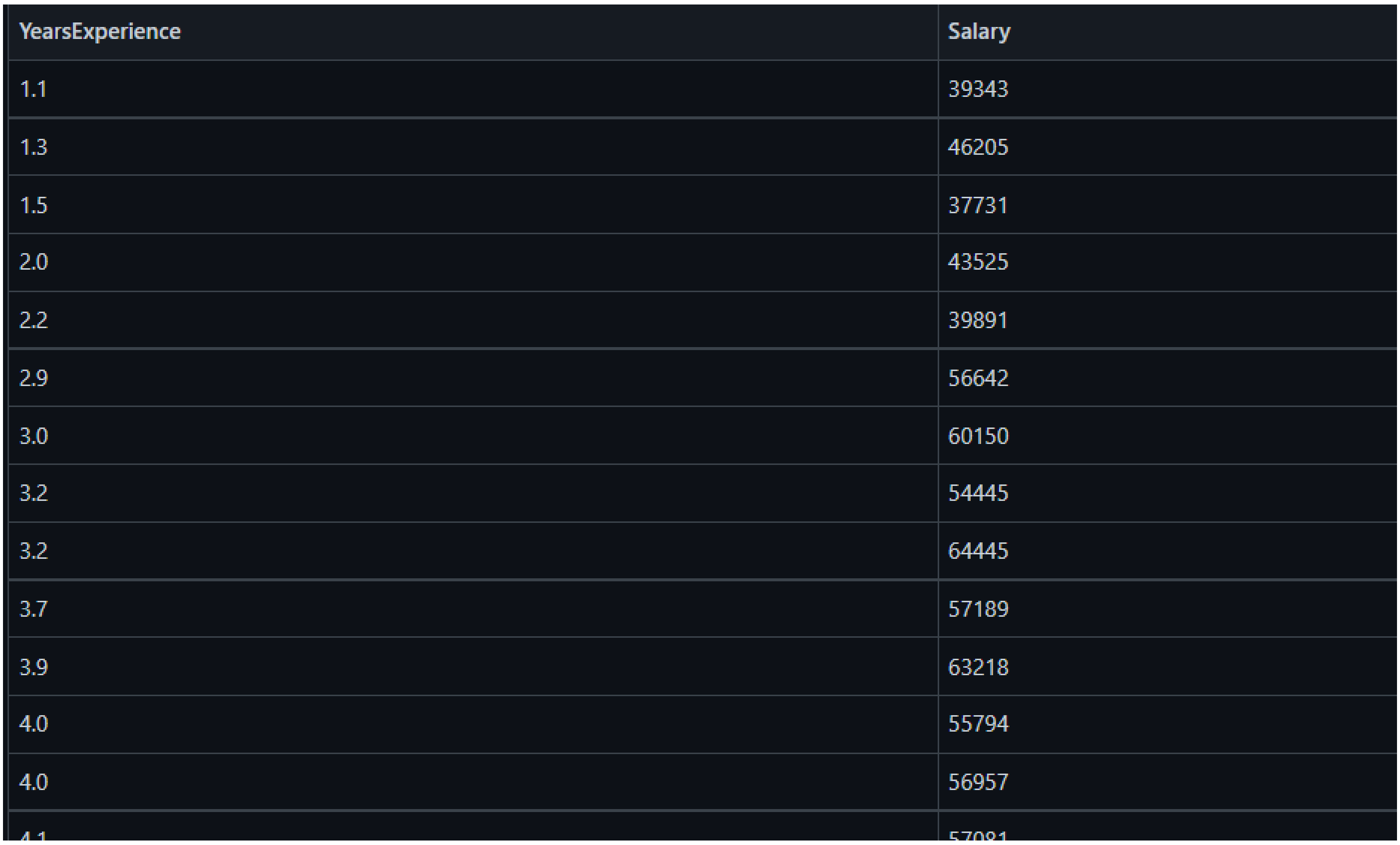


```
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
target
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
confusion matrix
[[16  0  0]
 [ 0 14  1]
 [ 0  0 14]]
accuracy
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.93      0.97        15
           2       0.93      1.00      0.97        14

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45

 /m/c/U/s/D/W/C/6/M/8.K-Nearest  master ●
```

Lab 9: Linear Regression

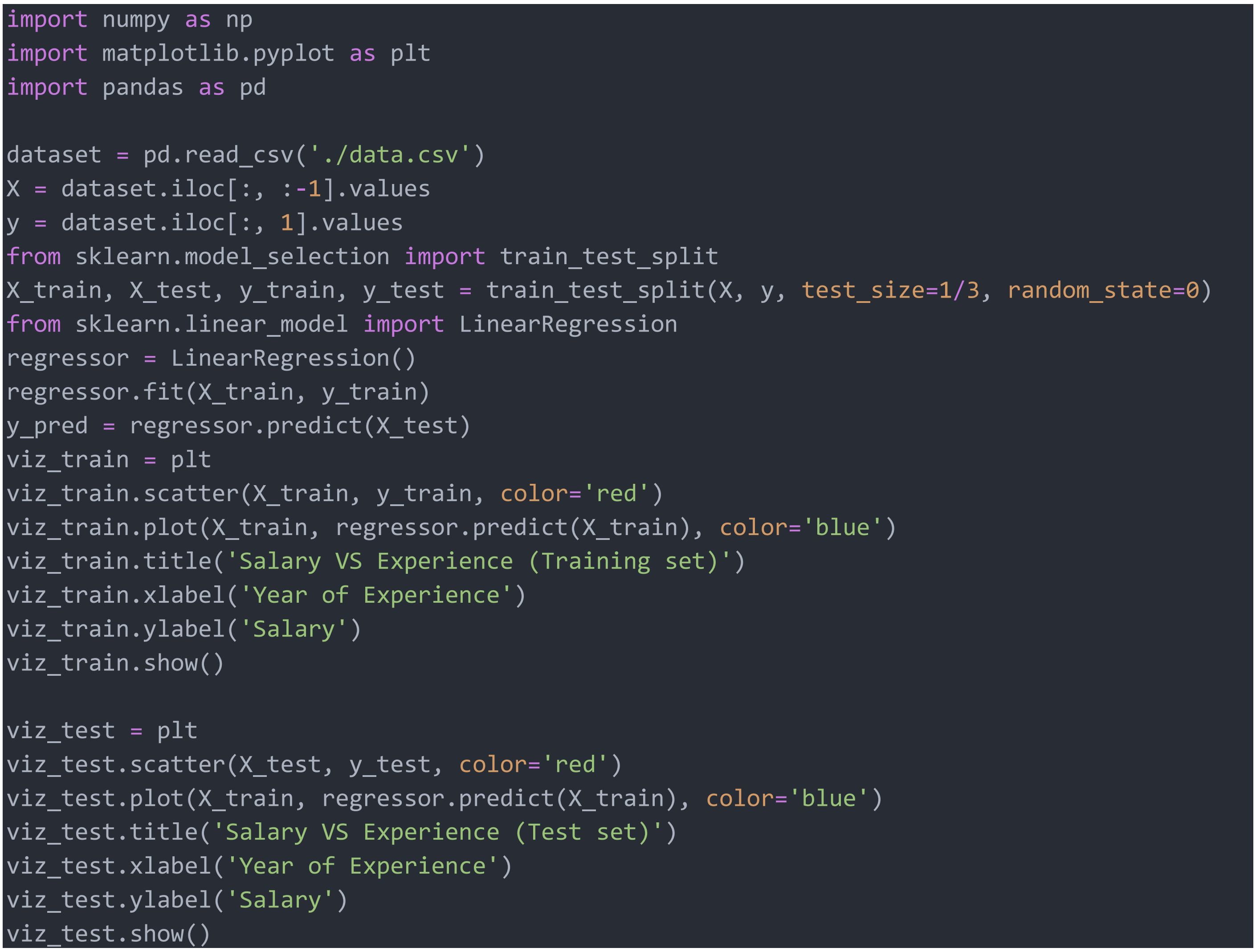
Implement the Linear Regression algorithm in order to fit data points. Select ap propriate data set for your experiment and draw graphs.

Dataset:



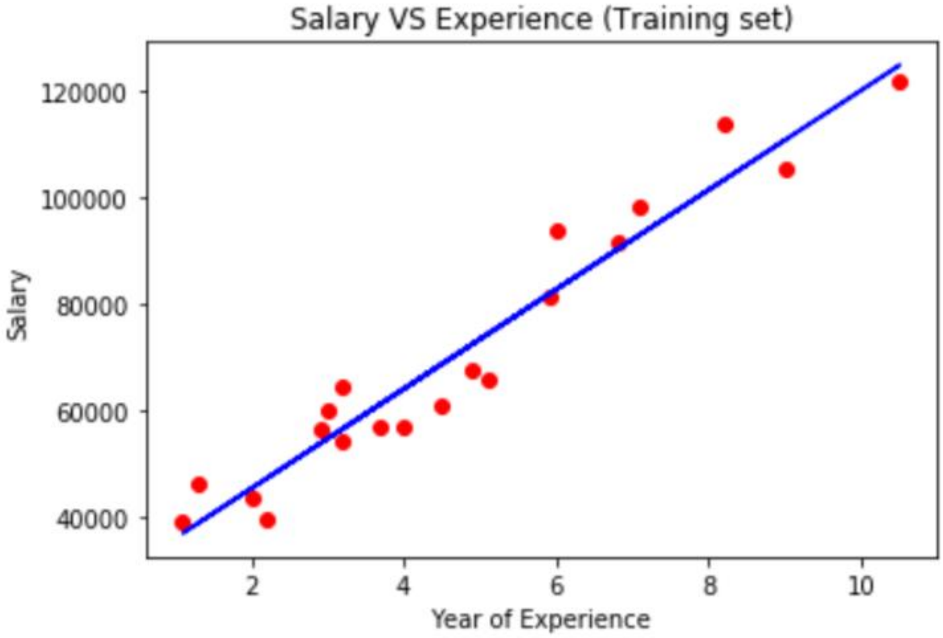
YearsExperience	Salary
1.1	39343
1.3	46205
1.5	37731
2.0	43525
2.2	39891
2.9	56642
3.0	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4.0	55794
4.0	56957
4.1	57081

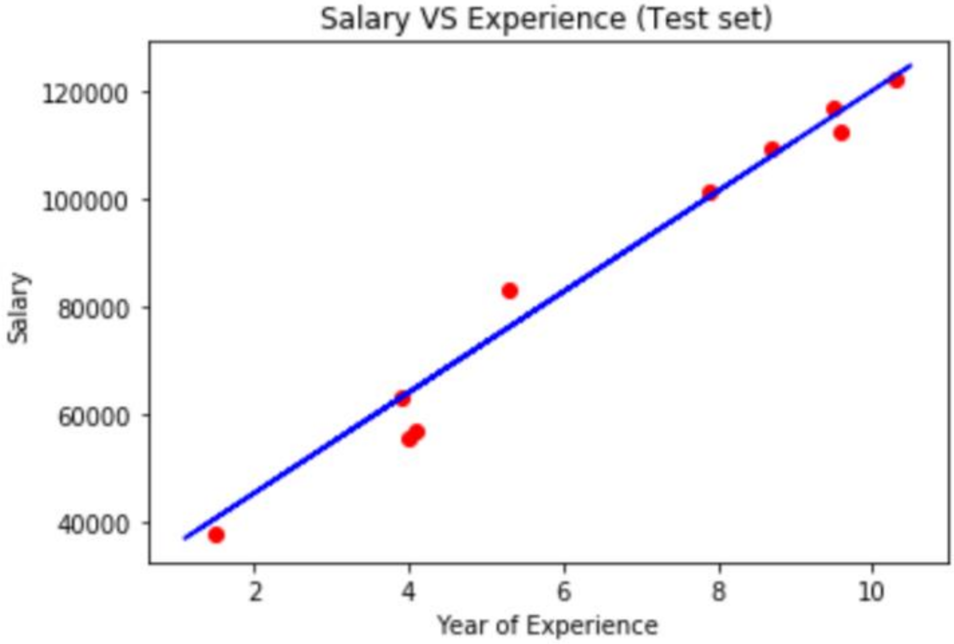
Code:



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

dataset = pd.read_csv('./data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
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()

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()
```

Output:

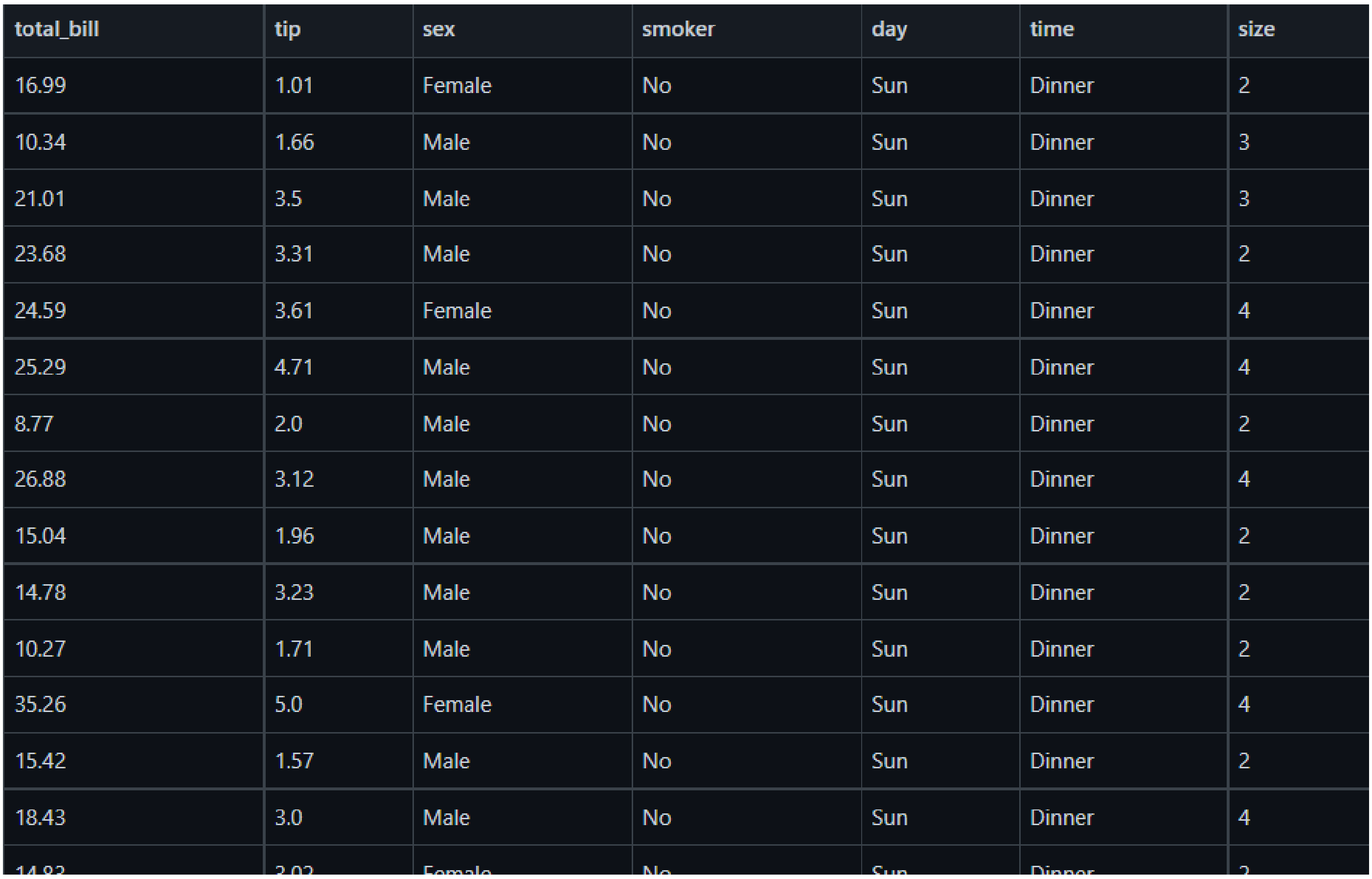




Lab 9: Locally Weighted Regression

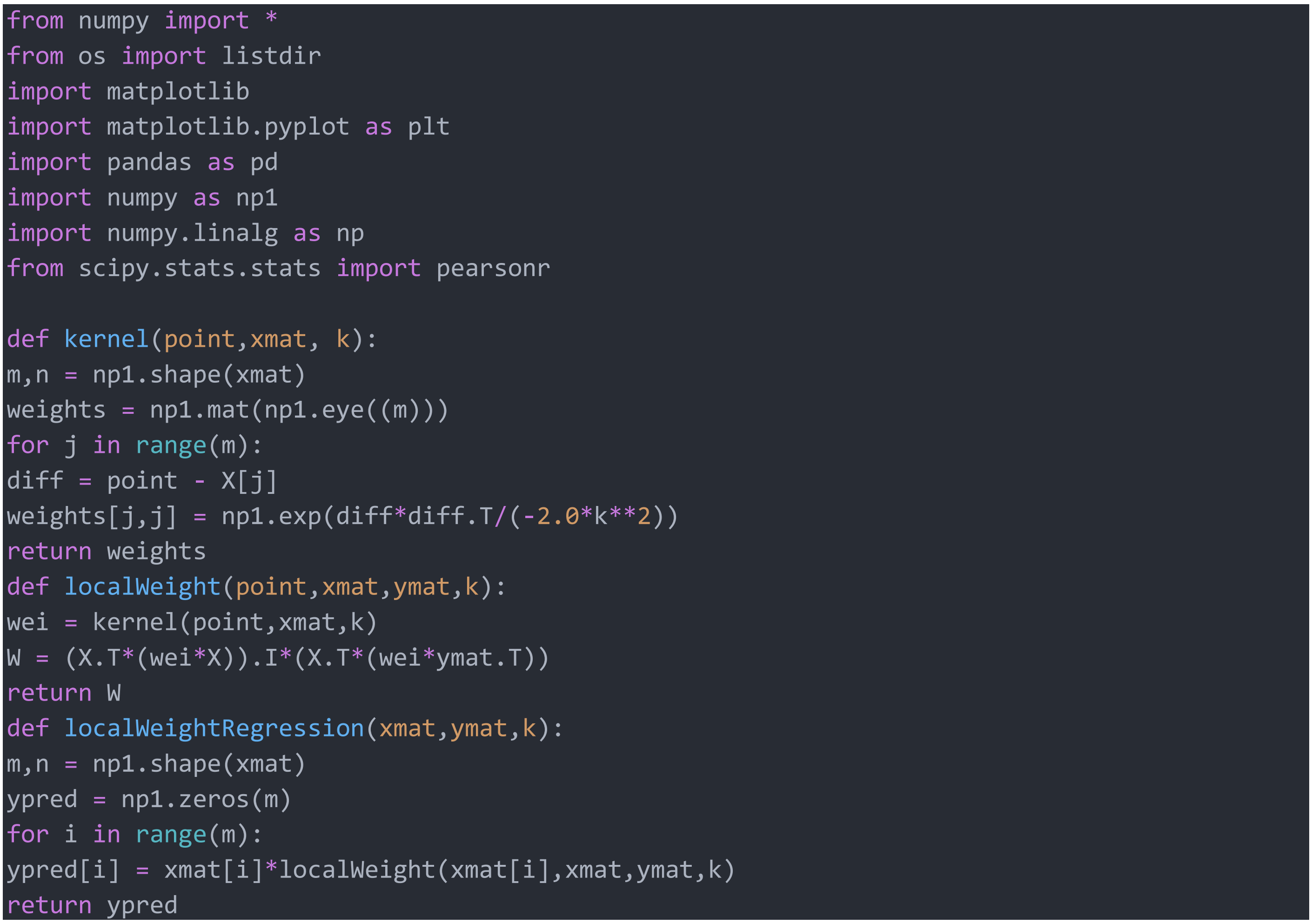
Implement the non-parametric Locally Weighted Regression algorithm in order of data points. Select appropriate data set for your experiment and draw graphs

Dataset:

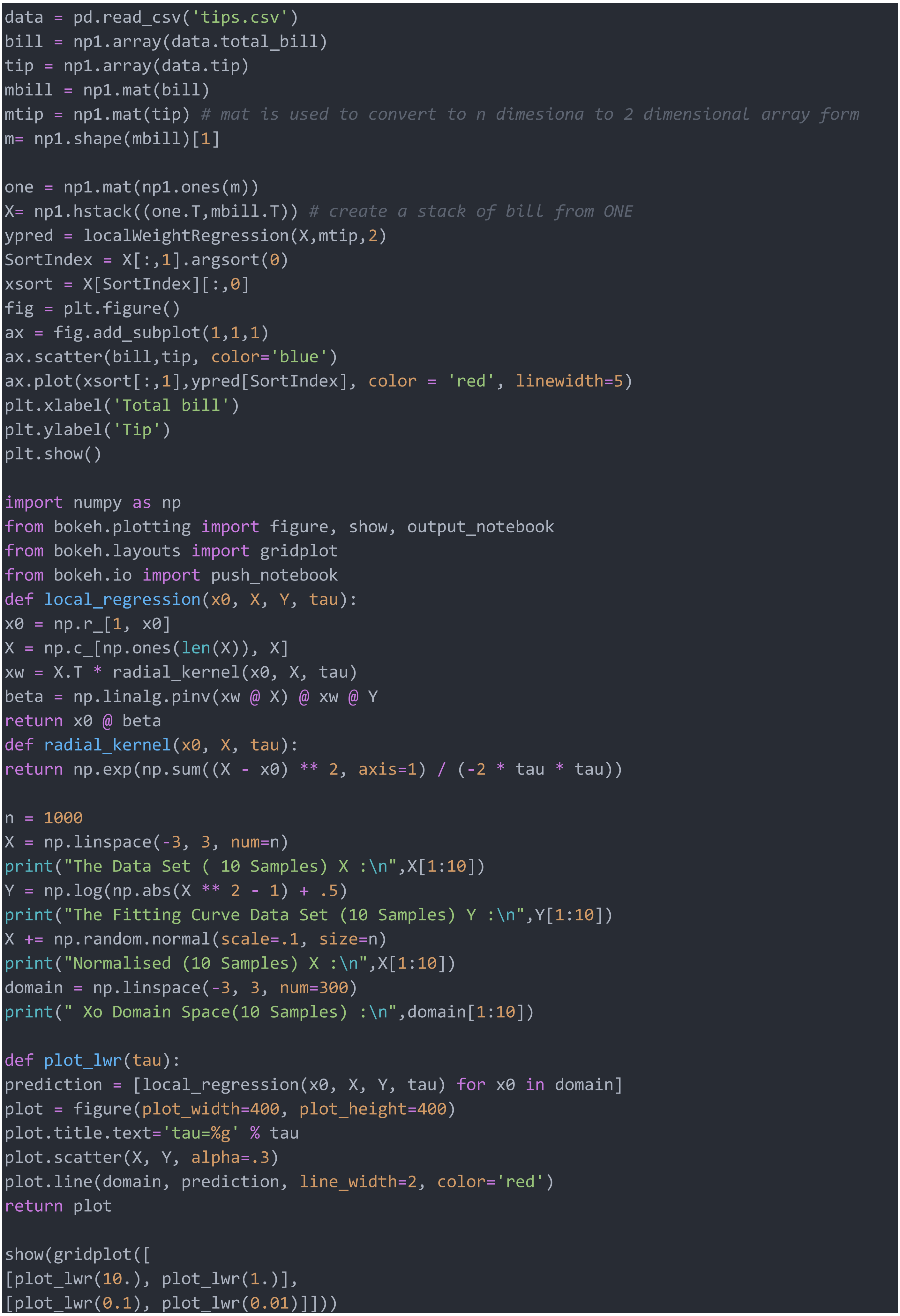


total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2.0	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2
14.78	3.23	Male	No	Sun	Dinner	2
10.27	1.71	Male	No	Sun	Dinner	2
35.26	5.0	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3.0	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2

Code:

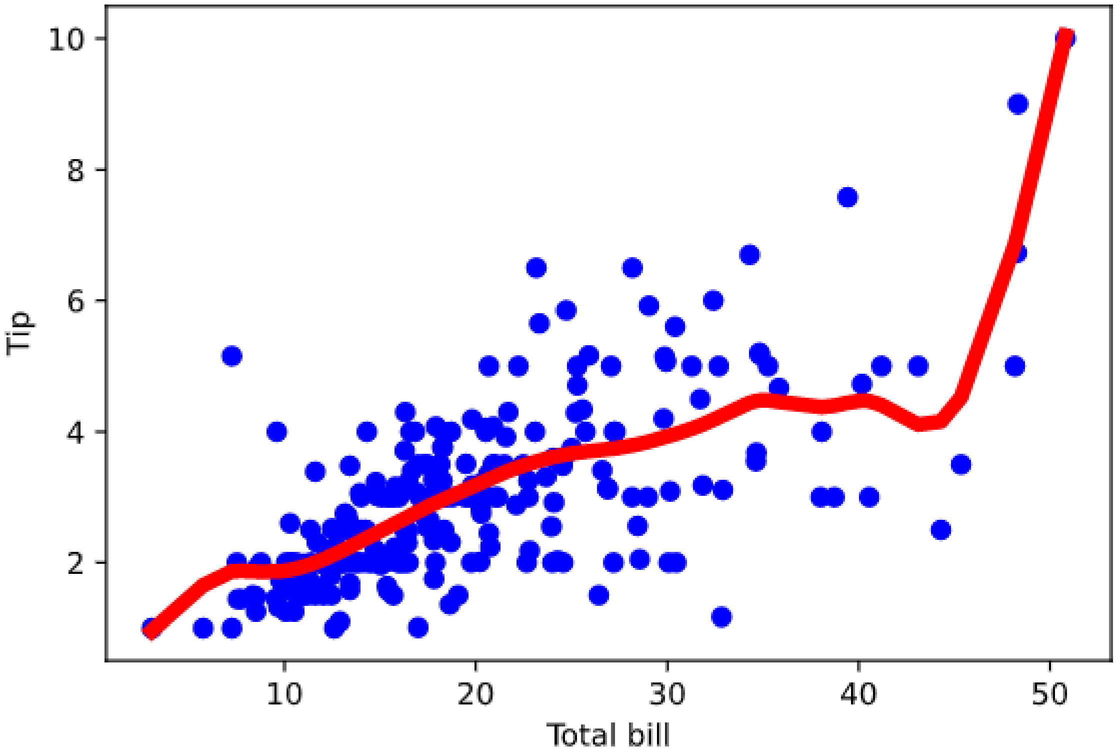


```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr

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



```
data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
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]

one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
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()

import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook
def local_regression(x0, X, Y, tau):
x0 = np.r_[1, x0]
X = np.c_[np.ones(len(X)), X]
xw = X.T * radial_kernel(x0, X, tau)
beta = np.linalg.pinv(xw @ X) @ xw @ Y
return x0 @ beta
def radial_kernel(x0, X, tau):
return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))

n = 1000
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])
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])

def plot_lwr(tau):
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

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

Output:


Tip
Total bill