**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:**

****

**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:

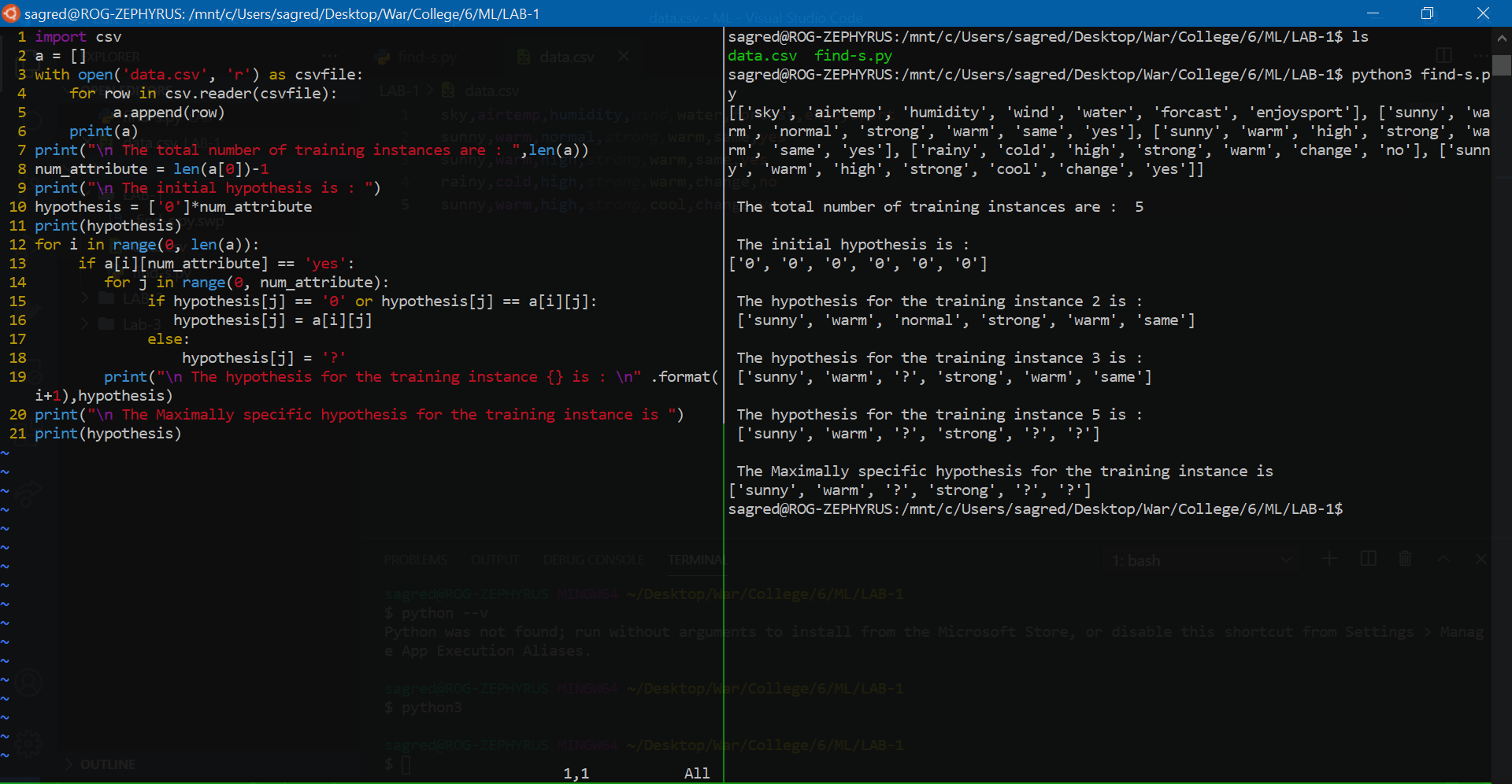
                 hypothesis[j] = '?'

        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)

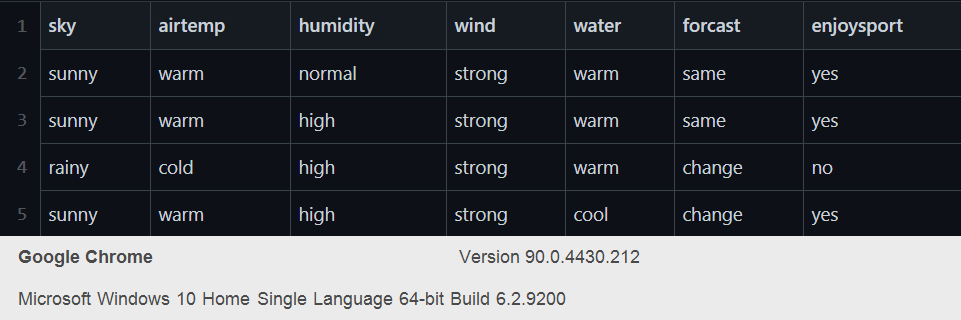
**Output:**



**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:**



**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(['?', '?', '?', '?', '?', '?'])

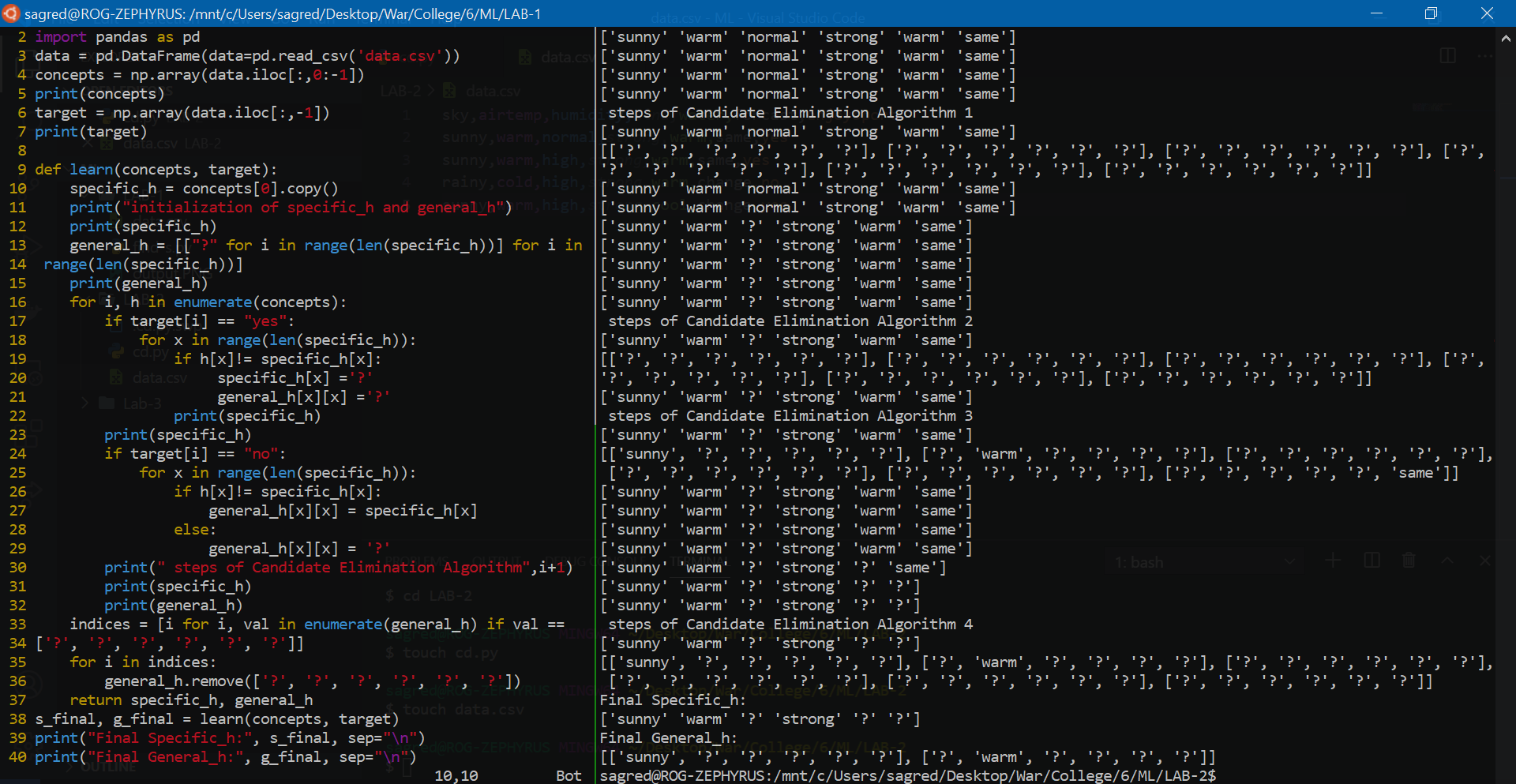
    return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

print("Final Specific\_h:", s\_final, sep="\n")

print("Final General\_h:", g\_final, sep="\n")

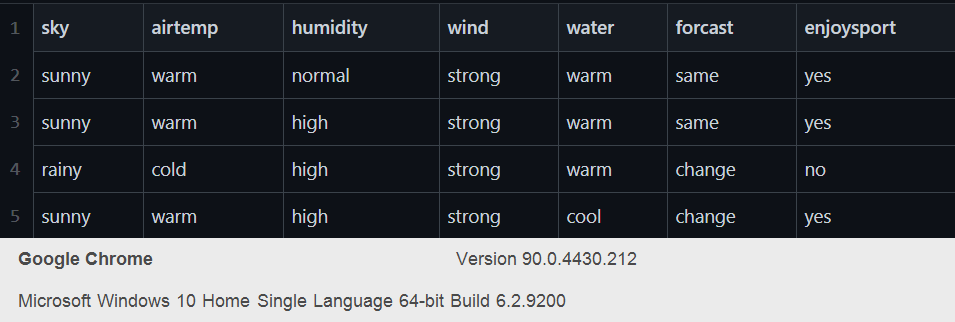
**Output:**

****

**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:**



**Code:**

import math

import csv

def load\_csv(filename):

    lines = csv.reader(open(filename, "r"))

    dataset = list(lines)

    headers = dataset.pop(0)

    return dataset, headers

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)

print("----------THE DECISION TREE----------")

print\_tree(model, 0)

testdata, features = load\_csv("test.csv")

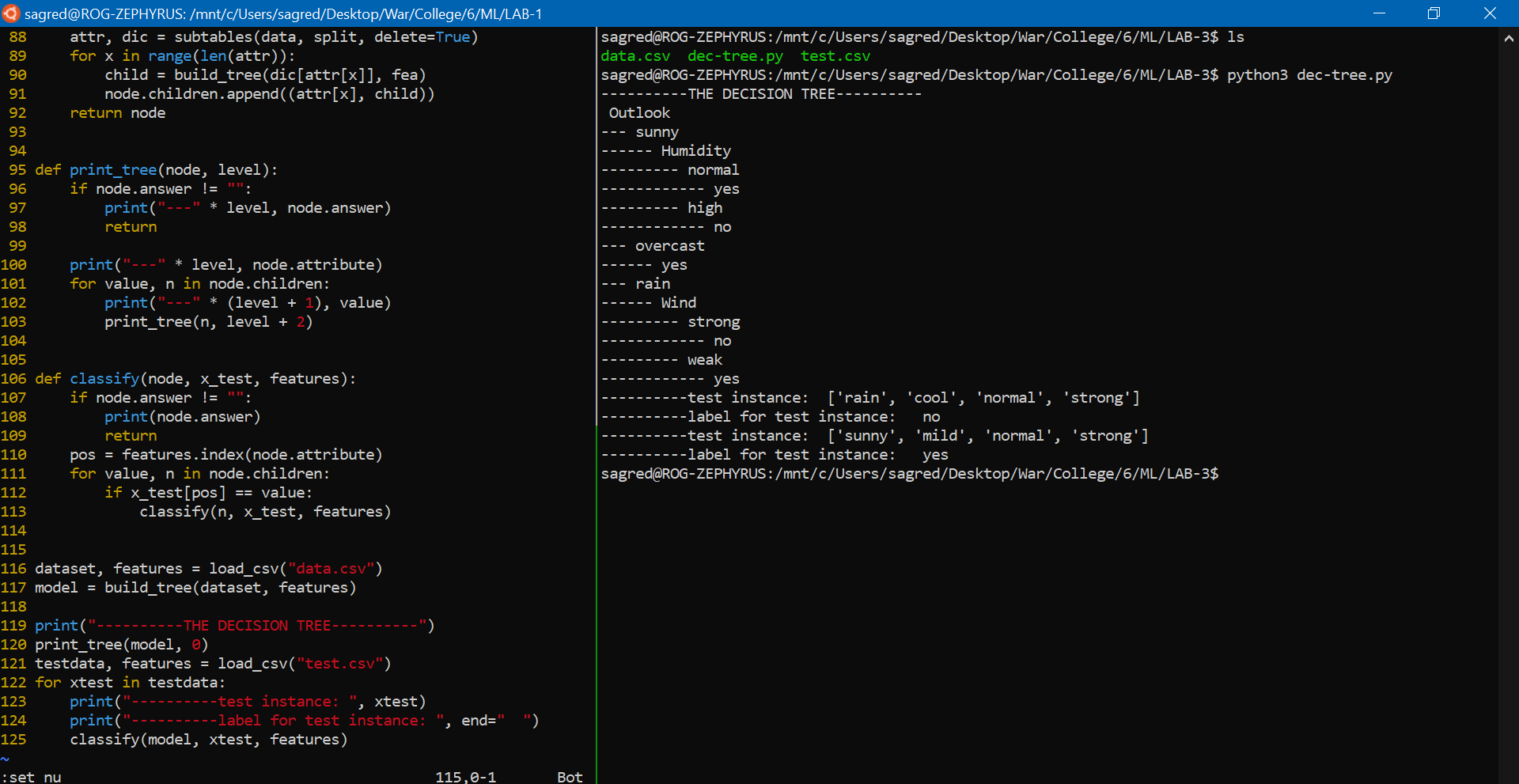
for xtest in testdata:

    print("----------test instance: ", xtest)

    print("----------label for test instance: ", end="  ")

    classify(model, xtest, features)

**Output:**

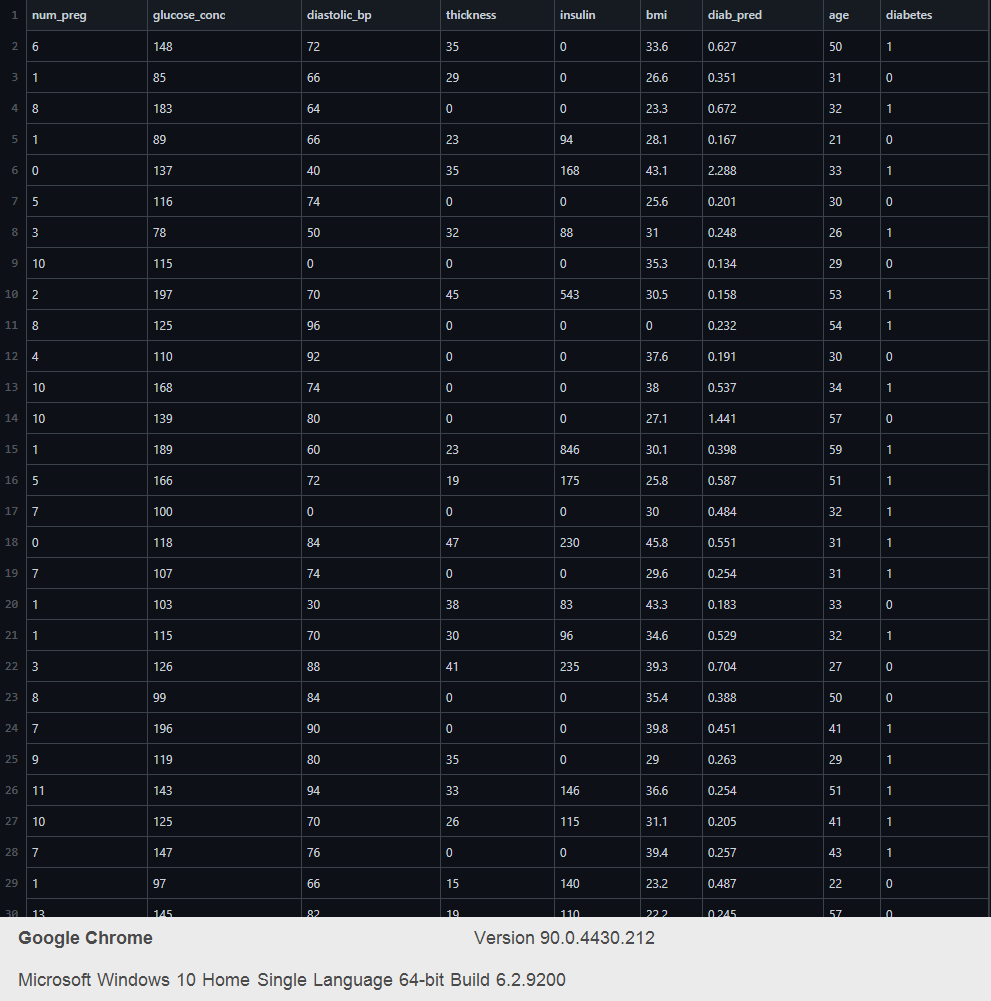


**Lab 4: Naïve Bayesian Classifier**

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:**



**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*

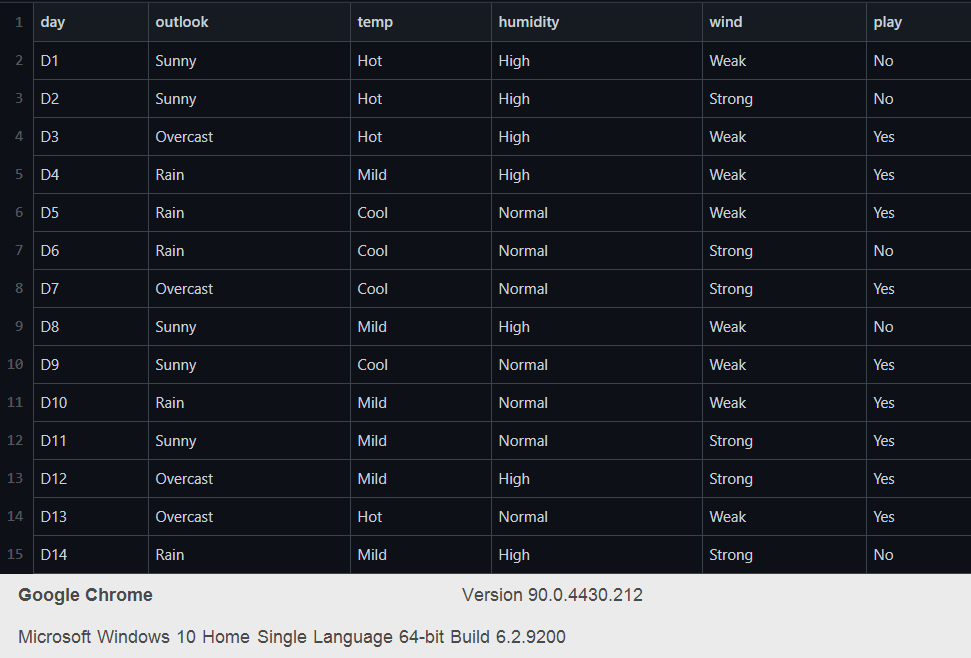
    accuracy = getaccuracy(testset, predictions)

    print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()

**Naïve Bayesian Classifier Example 2:**

**Dataset:**



**Code:**

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))

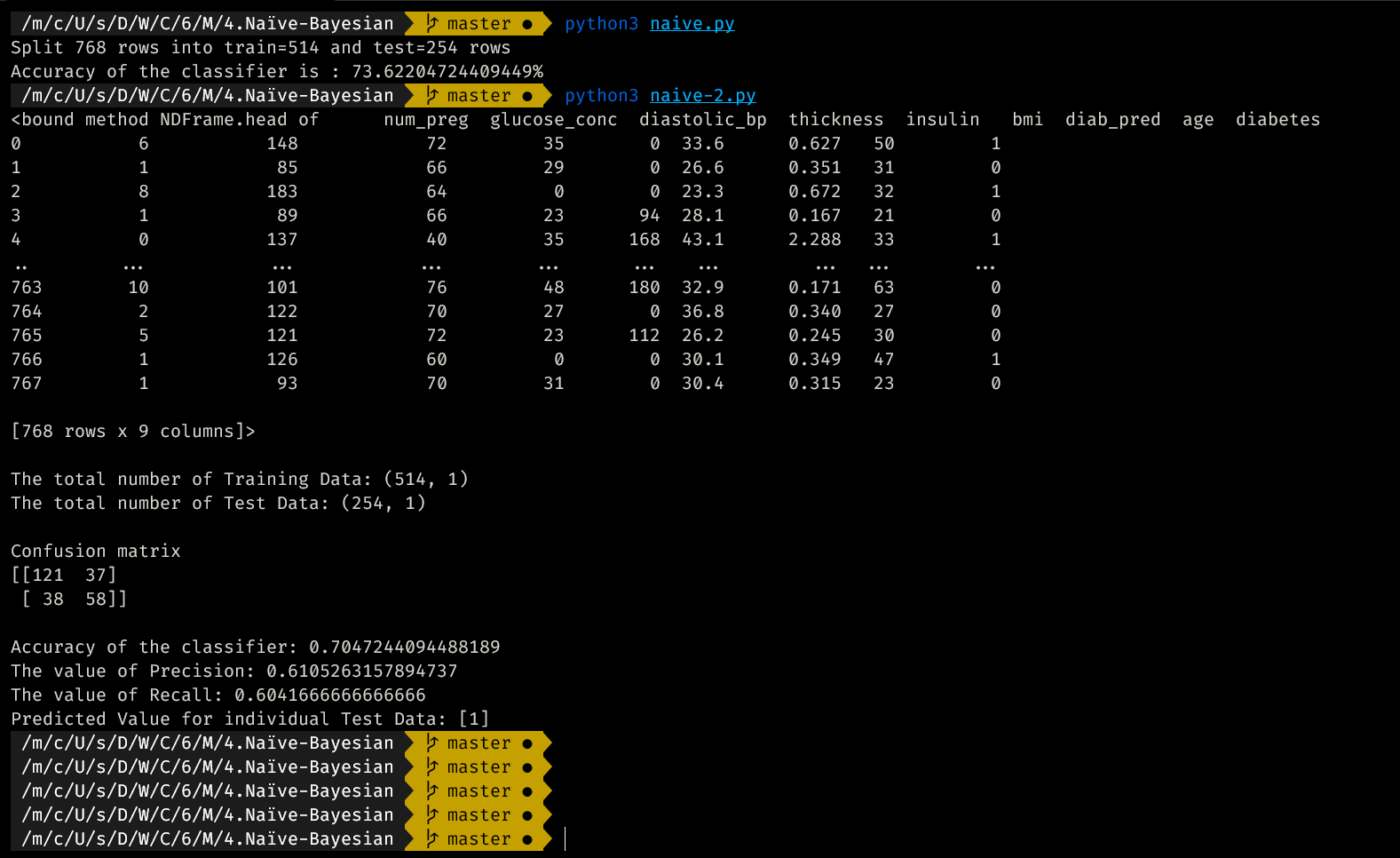
print('\nAccuracy of the classifier:',metrics.accuracy\_score(ytest,predicted))

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)

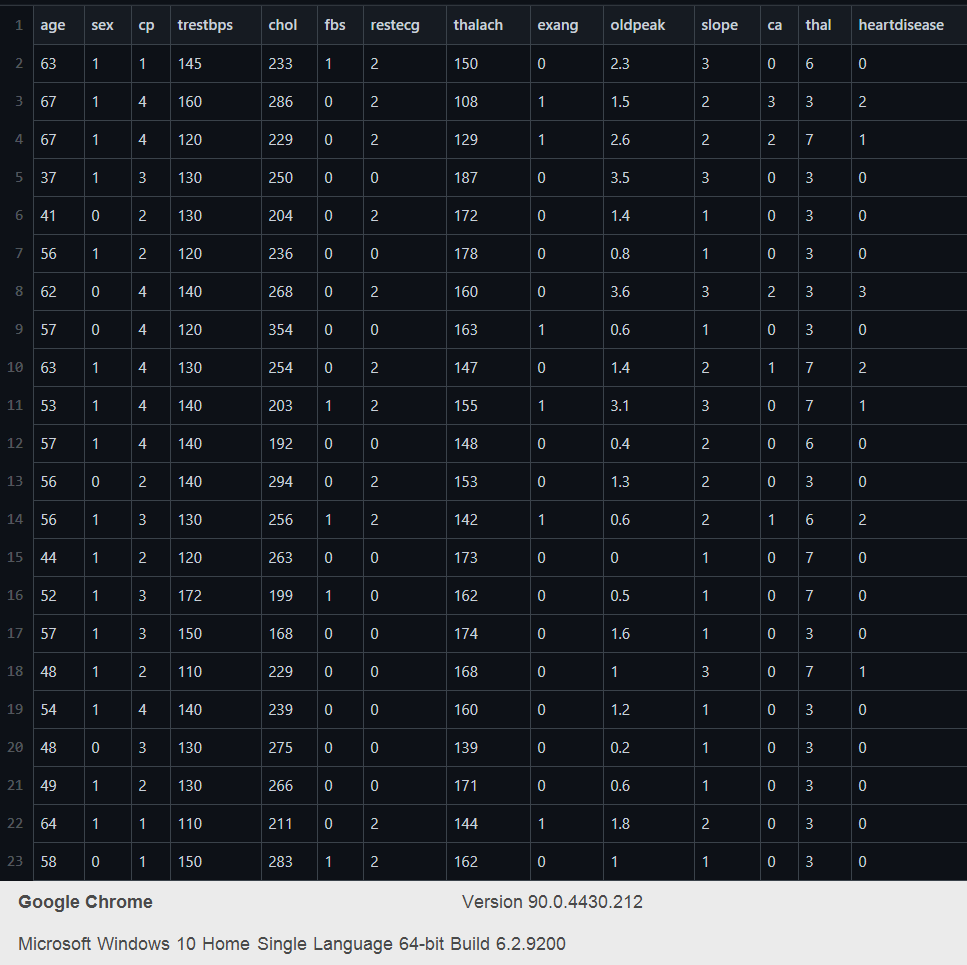
**Output:**



**Lab 5: Bayesian Network**

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

**Dataset:**



**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)

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%|███████████████████████████████████████████████████████████████████████████████████████████████████████| 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%|██████████████████████████████████████████████████████████████████████████████████████████| 5/5 [00:00<00:00, 3916.25it/s]**

**Eliminating: restecg: 100%|██████████████████████████████████████████████████████████████████████████████████████████████████| 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:**



**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')

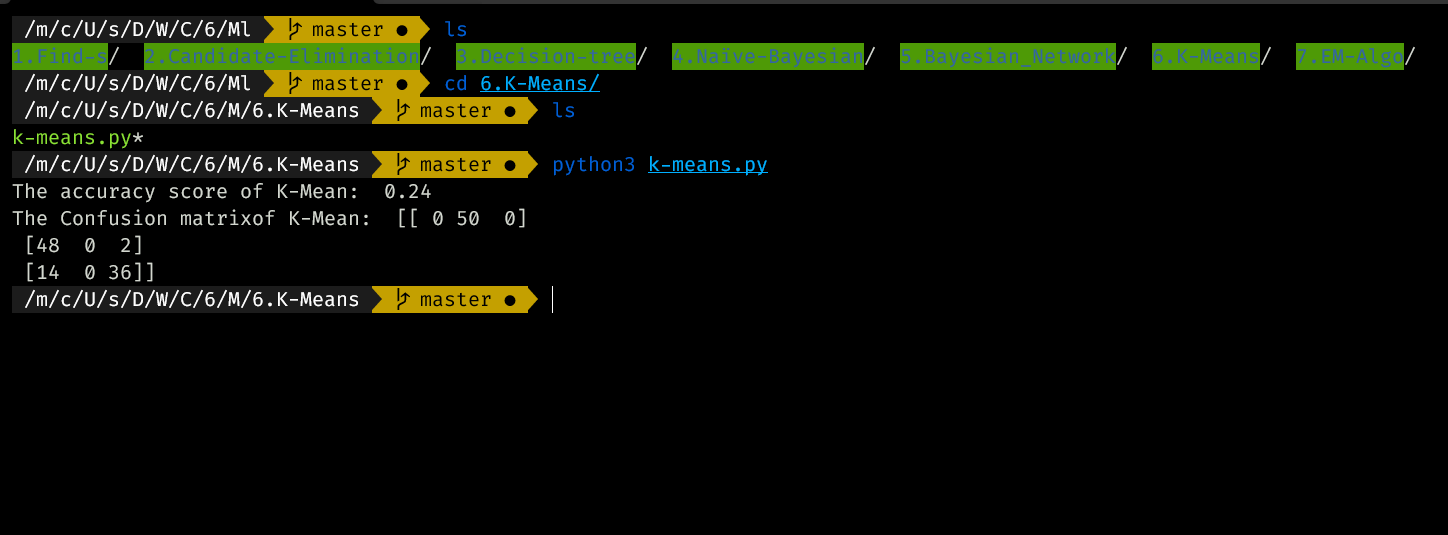
plt.xlabel('Petal Length')

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:**



**Lab 7: EM Algorithm**

Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results 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:**



**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')

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)

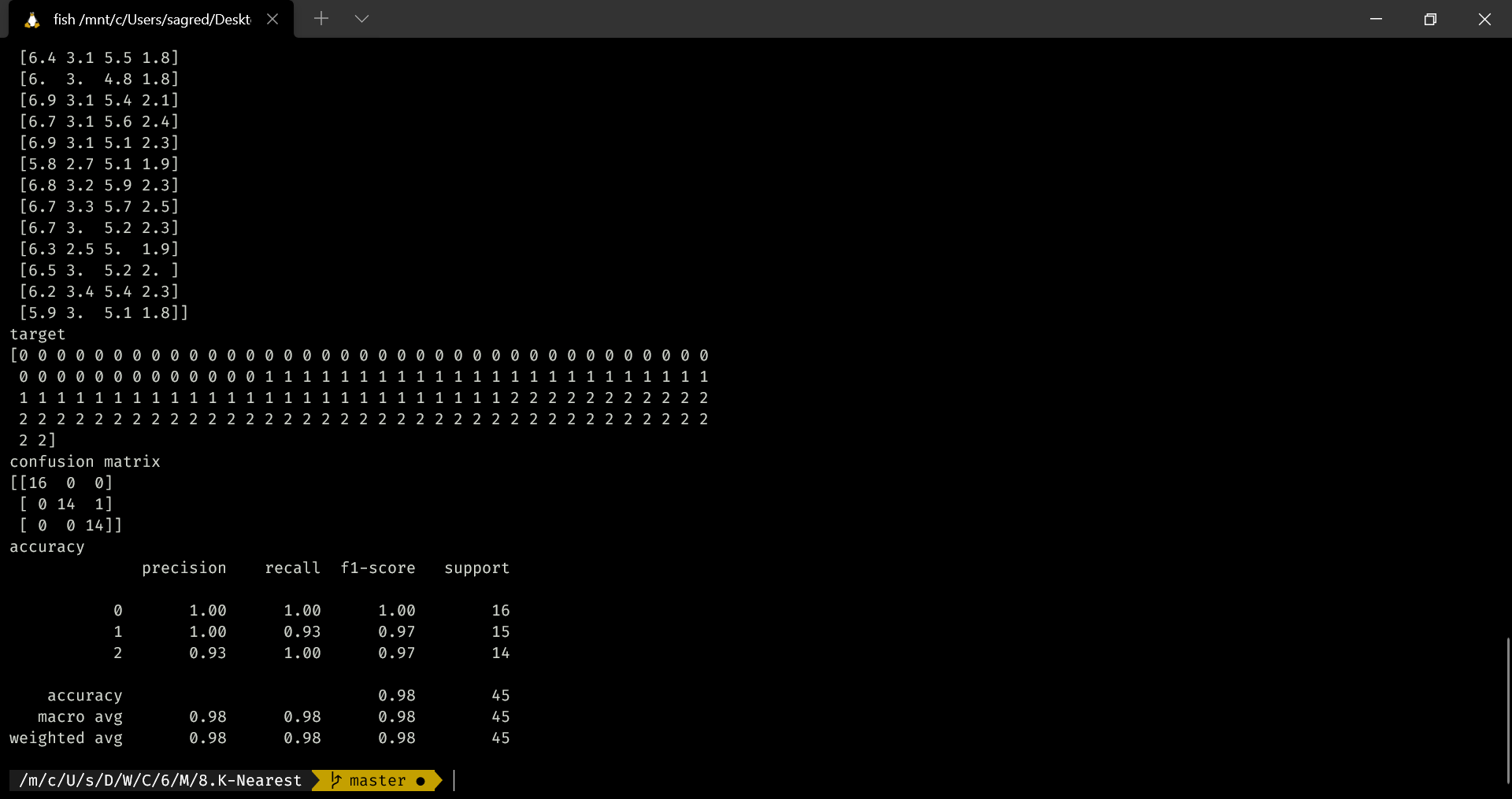
print('confusion matrix')

print(confusion\_matrix(y\_test,y\_pred))

print('accuracy')

print(classification\_report(y\_test,y\_pred))

**Output:**



**Lab 9: Linear Regression**

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

**Dataset:**



**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)')

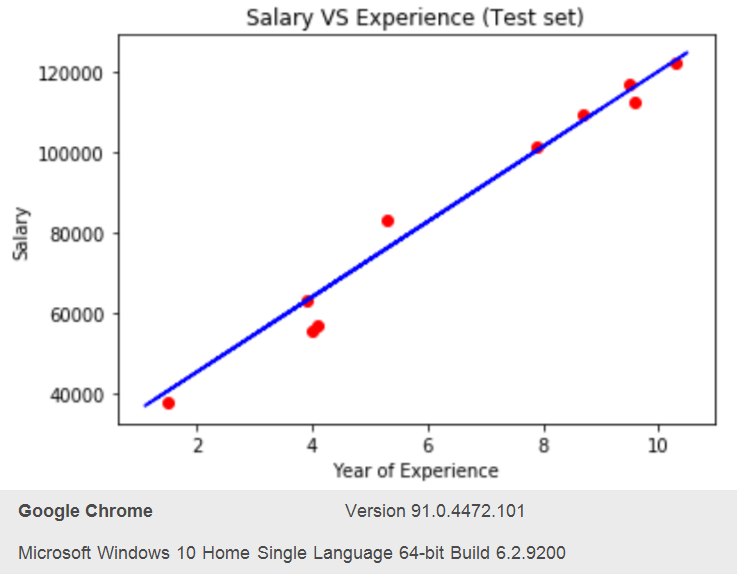
viz\_test.xlabel('Year of Experience')

viz\_test.ylabel('Salary')

viz\_test.show()

**Output:**

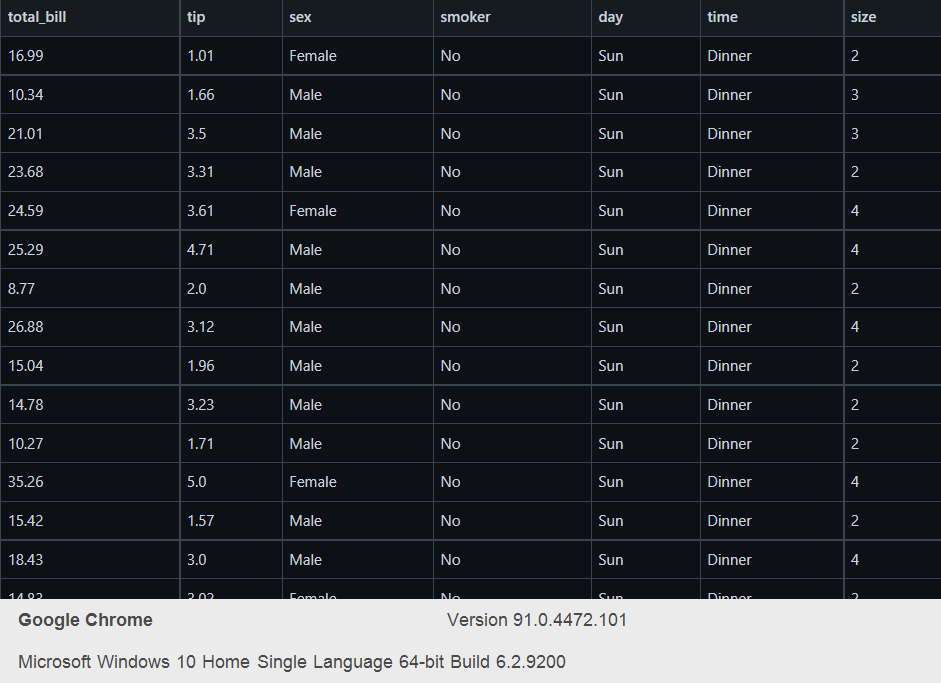




**Lab 9**: **Locally Weighted Regression**

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

**Dataset:**



**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:**

