In [1]:

```
#prog1
import csv
with open('P1_data.csv', 'r') as f:
    reader = csv.reader(f)
    headers = next(reader)
    your_list = list(reader)
h = [['0', '0', '0', '0', '0', '0']]

for i in your_list:
    print(i)
    if i[-1] == "TRUE":
        j = 0
        for x in i:
            if x != "TRUE":
                if x != h[0][j] and h[0][j] == '0':
                    h[0][j] = x
                elif x != h[0][j] and h[0][j] != '0':
                    h[0][j] = '?'
                else:
                    pass
            j = j + 1
print("The maximally specific hypothesis for a given training example is: ")
print(h)
```

```
["'Sunny'", "'Warm'", "'Normal'", "'Strong'", "'Warm'", "'Same'", 'TRUE']
["'Sunny'", "'Warm'", "'High'", "'Strong'", "'Warm'", "'Same'", 'TRUE']
["'Rainy'", "'Cold'", "'High'", "'Strong'", "'Warm'", "'Change'", 'FALSE']
["'Sunny'", "'Warm'", "'High'", "'Strong'", "'Cool'", "'Change'", 'TRUE']
The maximally specific hypothesis for a given training example is:
[["'Sunny'", "'Warm'", '?', "'Strong'", '?', '?']]
```

In [2]:

```
#prog2
import csv

def get_domains(examples):
    d = [set() for i in examples[0]]
    for x in examples:
        for i, xi in enumerate(x):
            d[i].add(xi)
    return [list(sorted(x)) for x in d]

def more_general(h1, h2):
    more_general_parts = []
    for x, y in zip(h1, h2):
        mg = x == "?" or (x != "0" and (x == y or y == "0"))
        more_general_parts.append(mg)
    return all(more_general_parts)

def fulfills(example, hypothesis):
# the implementation is the same as for hypotheses:
    return more_general(hypothesis, example)

def min_generalizations(h, x):
    h_new = list(h)
    for i in range(len(h)):
        if not fulfills(x[i:i+1], h[i:i+1]):
            h_new[i] = '?' if h[i] != '0' else x[i]
    return [tuple(h_new)]

def min_specializations(h, domains, x):
    results = []
```

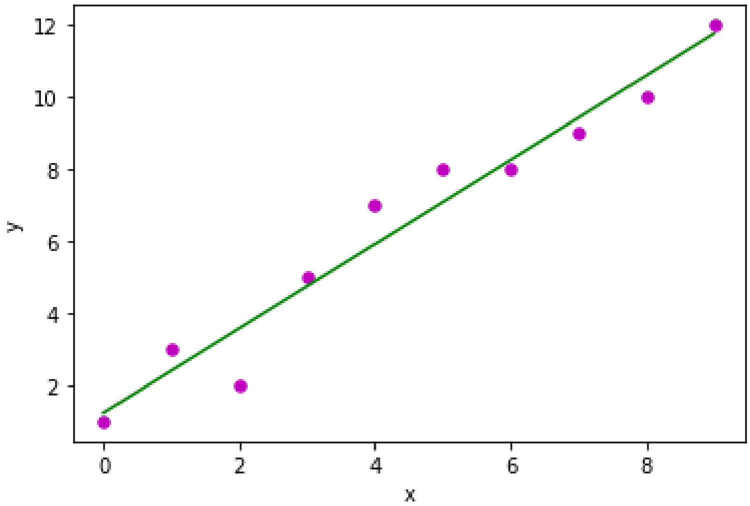
```
    for i in range(len(h)):
        if h[i] == "?":
            for val in domains[i]:
                if x[i] != val:
                    h_new = h[:i] + (val,) + h[i+1:]
                    results.append(h_new)
        elif h[i] != "0":
            h_new = h[:i] + ('0',) + h[i+1:]
            results.append(h_new)
    return results

def generalize_S(x, G, S):
    S_prev = list(S)
    for s in S_prev:
        if s not in S:
            continue
        if not fulfills(x, s):
            S.remove(s)
            Splus = min_generalizations(s, x)
            ## keep only generalizations that have a counterpart in G
            S.update([h for h in Splus if any([more_general(g,h) for g in G])])
            ## remove hypotheses less specific than any other in S
            S.difference_update([h for h in S if any([more_general(h, h1) for h1 in
    return S

def specialize_G(x, domains, G, S):
    G_prev = list(G)
    for g in G_prev:
        if g not in G:
            continue
        if fulfills(x, g):
            G.remove(g)
            Gminus = min_specializations(g, domains, x)
            ## keep only specializations that have a conuterpart in S
            G.update([h for h in Gminus if any([more_general(h, s) for s in S])])
            ## remove hypotheses less general than any other in G
            G.difference_update([h for h in G if any([more_general(g1, h) for g1 in
    return G

def candidate_elimination(examples):
    domains = get_domains(examples)[:-1]
    n = len(domains)
    G = set([("?",)*n])
    S = set([("0",)*n])
    print("Maximally specific hypotheses - S ")
    print("Maximally general hypotheses - G ")
    i=0
    print("\nS[0]:",str(S),"\nG[0]:",str(G))
    for xcx in examples:
        i=i+1
        x, cx = xcx[:-1], xcx[-1] # Splitting data into attributes and decisions
        if cx=='Y': # x is positive example
            G = {g for g in G if fulfills(x, g)}
            S = generalize_S(x, G, S)
        else: # x is negative example
            S = {s for s in S if not fulfills(x, s)}
            G = specialize_G(x, domains, G, S)
        print("\nS[{0}]:".format(i),S)
        print("G[{0}]:".format(i),G)
    return

with open('P2_dataset1.txt') as csvFile:
    examples = [tuple(line) for line in csv.reader(csvFile)]
candidate_elimination(examples)
```

```
Maximally specific hypotheses - S
Maximally general hypotheses - G

S[0]: {('0', '0', '0', '0', '0', '0')}
G[0]: {('?', '?', '?', '?', '?', '?')}

S[1]: {('sunny', 'warm', 'normal', 'strong', 'warm', 'same')}
G[1]: {('?', '?', '?', '?', '?', '?')}

S[2]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}
G[2]: {('?', '?', '?', '?', '?', '?')}

S[3]: {('sunny', 'warm', '?', 'strong', 'warm', 'same')}
G[3]: {('?', '?', '?', '?', '?', 'same'), ('?', 'warm', '?', '?', '?', '?'), ('sunn
y', '?', '?', '?', '?', '?')}

S[4]: {('sunny', 'warm', '?', 'strong', '?', '?')}
G[4]: {('?', 'warm', '?', '?', '?', '?'), ('sunny', '?', '?', '?', '?', '?')}
```

In [3]:

```python
#prog3
import numpy as np
import matplotlib.pyplot as plt
def estimate_coef(x, y):
    # number of observations/points
    n =np.size(x)
    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)
    # calculating cross-deviation and deviation aboutx
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) -n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    return(b_0, b_1)
def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m", marker = "o", s = 30)
    # predicted response vector
    y_pred = b[0] + b[1]*x
    # plotting the regression line
    plt.plot(x, y_pred, color = "g")
    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
    # function to show plot
    plt.show()
def main():
    # observations
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])
    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \nb_1 = {}".format(b[0], b[1]))
    # plotting regression line
    plot_regression_line(x, y, b)
if __name__ == "__main__":
    main()
```

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



In [4]:

```
#prog4
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 = ""
# NULL indicates children exists.
# Not Null indicates this is a Leaf Node
def subtables(data, col, delete):
    dic = {}
    coldata = [ row[col] for row in data]
    attr = list(set(coldata)) # All values of attribute retrived
    for k in attr:
        dic[k] = []
    for y in range(len(data)):
        key = data[y][col]
        if delete:
            del data[y][col]
        dic[key].append(data[y])
    return attr, dic
def entropy(S):
    attr = list(set(S))
    if len(attr) == 1: #if all are +ve/-ve then entropy = 0
        return 0
    counts = [0,0] # Only two values possible 'yes' or 'no'
    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):
    attValues, dic = subtables(data, col, delete=False)
    total_entropy = entropy([row[-1] for row in data])
    for x in range(len(attValues)):
        ratio = len(dic[attValues[x]]) / ( len(data) * 1.0)
        entro = entropy([row[-1] for row in dic[attValues[x]]])
```

```
        total_entropy -= ratio*entro
    return total_entropy

def build_tree(data, features):
    lastcol = [row[-1] for row in data]
    if (len(set(lastcol))) == 1: # If all samples have same labels return that label
        node=Node("")
        node.answer = lastcol[0]
        return node
    n = len(data[0])-1
    gains = [compute_gain(data, col) for col in range(n) ]
    split = gains.index(max(gains)) # Find max gains and returns index
    node = Node(features[split]) # 'node' stores attribute selected
    #del (features[split])
    fea = features[:split]+features[split+1:]
    attr, dic = subtables(data, split, delete=True) # Data will be spilt in subtable
    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) # Displays leaf node yes/no
        return
    print(" "*level, node.attribute) # Displays attribute Name
    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("P3_data3.csv") # Read Tennis data
node = build_tree(dataset, features) # Build decision tree
print("The decision tree for the dataset using ID3 algorithm is ")
print_tree(node, 0)

testdata, features = load_csv("P3_data3_test.csv")
for xtest in testdata:
    print("The test instance : ",xtest)
    print("The predicted label : ", end="")
    classify(node,xtest,features)
```

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

```
The test instance :  ['rain', 'cool', 'normal', 'strong']
The predicted label : no
The test instance :  ['sunny', 'mild', 'normal', 'strong']
The predicted label : yes
```

In [5]:

```
#prog5
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100
#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
#Variable initialization
epoch=5000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
#Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)
#Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contribut
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror and current
    # bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    #bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89415347]
```

```
[0.8819451 ]
[0.89366227]]
```

In [6]:

```
#prog6
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 = {}
#creates a dictionary of classes 1 and 0 where the values are the instacnes belongin
    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):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries

def summarizeByClass(dataset):
    separated = separateByClass(dataset)
#print(separated)
    summaries = {}
    for classValue, instances in separated.items():
#summaries is a dic of tuples(mean,std) for each class value
        summaries[classValue] = summarize(instances)
    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 = {}
    for classValue, classSummaries in summaries.items():#class and attribute informa
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i] #take mean and sd of every attribute for
            x = inputVector[i] #testvector's first attribute
            probabilities[classValue] *= calculateProbability(x, mean, stdev);#use n
    return probabilities

def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():#assigns that class which h
        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 = 'P5_naivedata.csv'
    splitRatio = 0.67
    dataset = loadCsv(filename);
    print('Pima Indian Diabetes Dataset loaded...')
    print('Total instances available :',len(dataset))
    print('Total attributes present :',len(dataset[0])-1)
    print("First Five instances of dataset:")
    for i in range(5):
        print(i+1 , ':' , dataset[i])
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print('\nDataset is split into training and testing set.')
    print('Training examples = {0} \nTesting examples = {1}'.format(len(trainingSet)
    # prepare model
    summaries = summarizeByClass(trainingSet);
    #print(summaries)
    # test model
    predictions = getPredictions(summaries, testSet)
    #print(predictions)
    accuracy = getAccuracy(testSet, predictions)
    print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()
```

```
Pima Indian Diabetes Dataset loaded...
Total instances available : 744
Total attributes present : 8
First Five instances of dataset:
1 : [6.0, 134.0, 34.0, 35.0, 0.0, 33.5, 0.456, 51.0, 1.0]
2 : [5.0, 45.0, 45.0, 56.0, 0.0, 32.6, 0.87, 46.0, 1.0]
3 : [7.0, 34.0, 23.0, 34.0, 0.0, 34.9, 0.65, 65.0, 1.0]
```

```
4 : [9.0, 56.0, 65.0, 23.0, 76.0, 34.2, 0.765, 34.0, 0.0]
5 : [8.0, 78.0, 89.0, 23.0, 123.0, 12.45, 0.34, 65.0, 0.0]

Dataset is split into training and testing set.
Training examples = 498
Testing examples = 246
Accuracy of the classifier is : 100.0%
```

In [9]:

```
#prog7
import pandas as pd
msg=pd.read_csv('naivetext1.csv',names=['message','label'])
print('Total instances in the dataset:',msg.shape[0])
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
Y=msg.labelnum

print('\nThe message and its label of first 5 instances are listed below')
X5, Y5 = X[0:5], msg.label[0:5]
for x, y in zip(X5,Y5):
    print(x,',',y)

#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,Y)
print('\nDataset is split into Training and Testing samples')
print('Total training instances :', xtrain.shape[0])
print('Total testing instances :', xtest.shape[0])

#output of count vectoriser is a sparse matrix
# CountVectorizer - stands for 'feature extraction'
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain) #Sparse matrix
xtest_dtm=count_vect.transform(xtest)
print(count_vect.get_feature_names())
print('\nTotal features extracted using CountVectorizer:',xtrain_dtm.shape[1])

print('\nFeatures for first 5 training instances are listed below')
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df[0:5])#tabular representation
#print(xtrain_dtm) #Same as above but sparse matrix representation

# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

print('\nClassstification results of testing samples are given below')
for doc, p in zip(xtest, predicted):
    pred = 'pos' if p==1 else 'neg'
    print('%s -> %s ' % (doc, pred))

#printing accuracy metrics
from sklearn import metrics
print('\nAccuracy metrics')
print('\nAccuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('\nConfusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\nRecall')
print(metrics.recall_score(ytest,predicted))
print('\nPrecison ')
print(metrics.precision_score(ytest,predicted))
```

```
Total instances in the dataset: 18

The message and its label of first 5 instances are listed below
I love this sandwich , pos
This is an amazing place , pos
I feel very good about these beers , pos
This is my best work , pos
What an awesome view , pos

Dataset is split into Training and Testing samples
Total training instances : 13
Total testing instances : 5
['about', 'an', 'awesome', 'beers', 'best', 'boss', 'can', 'dance', 'deal', 'do', 'e
nemy', 'feel', 'fun', 'good', 'great', 'have', 'he', 'holiday', 'horrible', 'house',
'is', 'juice', 'like', 'love', 'my', 'not', 'of', 'place', 'sandwich', 'sworn', 'tas
te', 'the', 'these', 'this', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'wen
t', 'what', 'will', 'with', 'work']

Total features extracted using CountVectorizer: 45

Features for first 5 training instances are listed below
   about  an  awesome  beers  best  boss  can  dance  deal  do  ...  today  \
0      0   0        0      0     0     0    0      0     0   1  ...      0
1      0   0        0      0     0     0    0      0     0   0  ...      0
2      0   0        0      0     0     0    0      1     0   0  ...      0
3      0   1        1      0     0     0    0      0     0   0  ...      0
4      0   0        0      0     0     0    0      0     0   0  ...      0

   tomorrow  very  view  we  went  what  will  with  work
0         0     0     0   0     0     0     0     0     0
1         1     0     0   1     0     0     1     0     0
2         0     0     0   0     0     0     0     0     0
3         0     0     0   0     0     0     0     0     0
4         0     0     0   0     0     0     0     0     0

[5 rows x 45 columns]

Classstification results of testing samples are given below
This is an amazing place -> pos
That is a bad locality to stay -> pos
I do not like this restaurant -> neg
I am sick and tired of this place -> pos
I am tired of this stuff -> neg

Accuracy metrics

Accuracy of the classifer is 0.6

Confusion matrix
[[2 2]
 [0 1]]

Recall
1.0

Precison
0.3333333333333333
```

In [10]:

```python
#prog8
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
```

```python
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination #Read the attributes
lines = list(csv.reader(open('P7_data7_names.csv', 'r')));
attributes = lines[0]
#attributes = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
#Read Cleveland Heart dicease data
heartDisease = pd.read_csv('P7_data7_heart.csv', names = attributes)
heartDisease = heartDisease.replace('?', np.nan)
# Display the data
print('Few examples from the dataset are given below')
print(heartDisease.head())
print('\nAttributes and datatypes')
print(heartDisease.dtypes)
# Model Baysian Network
model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'),
('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),
('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])
# Learning CPDs using Maximum Likelihood Estimators
print('\nLearning CPDs using Maximum Likelihood Estimators...');
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)
# Inferencing with Bayesian Network print('\nInferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model) # Computing the probability of bronc
print('\n1.Probability of HeartDisease given Age=29')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 29}, joint
print(q['heartdisease'])
print('\n2. Probability of HeartDisease given chol (Cholestoral) =126')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'chol': 126}, joi
print(q['heartdisease'])
```

```
Few examples 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               object
thal             object
heartdisease      int64
dtype: object

Learning CPDs using Maximum Likelihood Estimators...
D:\anaconda_installation_folder\lib\site-packages\pgmpy\models\BayesianModel.py:8: F
utureWarning: BayesianModel has been renamed to BayesianNetwork. Please use Bayesian
```

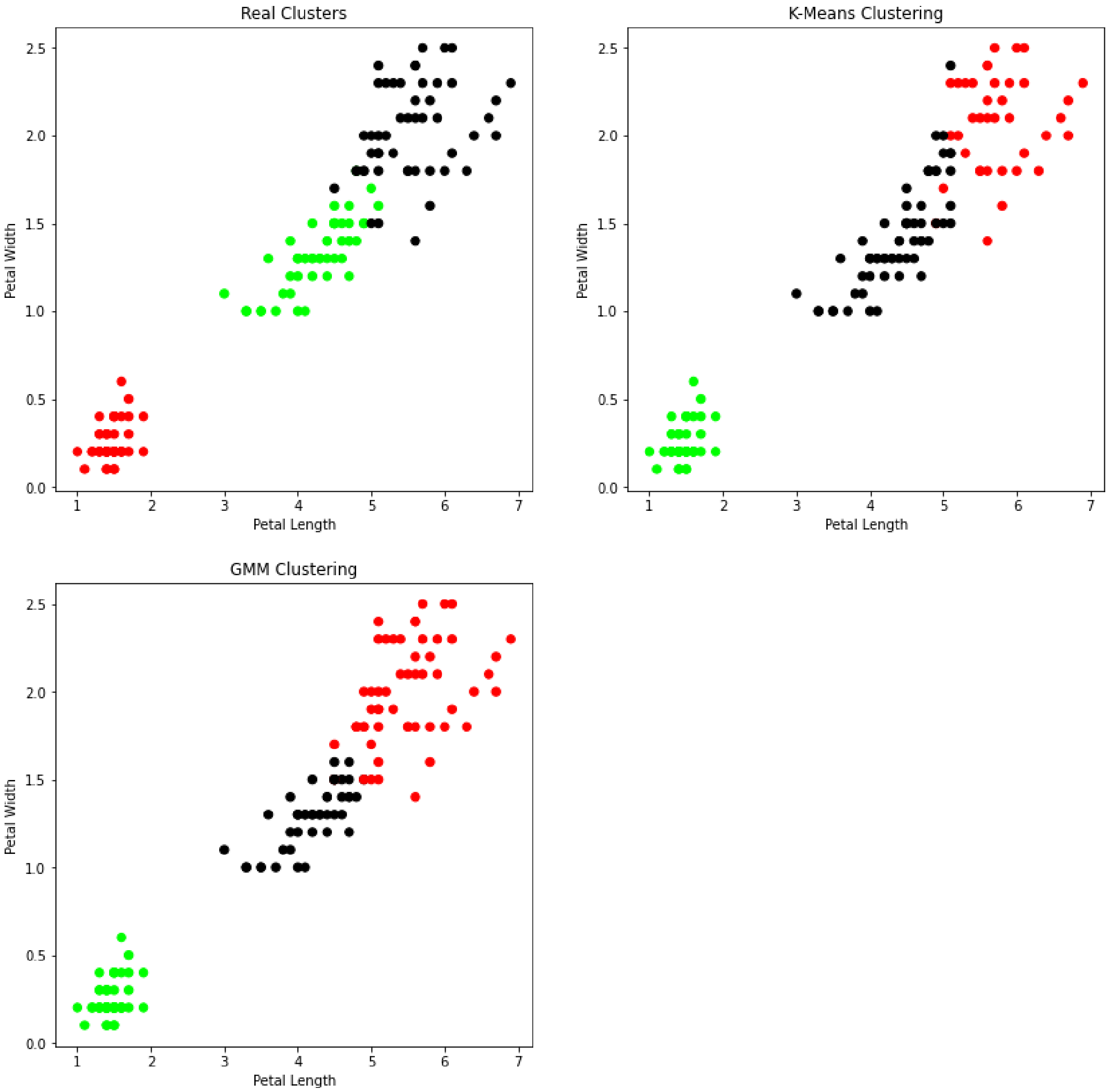
```
Network class, BayesianModel will be removed in future.
  warnings.warn(
1.Probability of HeartDisease given Age=29
+----------------+---------------------+
| heartdisease   |   phi(heartdisease) |
+================+=====================+
| heartdisease(0) |              0.5846 |
+----------------+---------------------+
| heartdisease(1) |              0.1820 |
+----------------+---------------------+
| heartdisease(2) |              0.1006 |
+----------------+---------------------+
| heartdisease(3) |              0.0799 |
+----------------+---------------------+
| heartdisease(4) |              0.0530 |
+----------------+---------------------+

2. Probability of HeartDisease given chol (Cholestoral) =126
+----------------+---------------------+
| heartdisease   |   phi(heartdisease) |
+================+=====================+
| heartdisease(0) |              1.0000 |
+----------------+---------------------+
| heartdisease(1) |              0.0000 |
+----------------+---------------------+
| heartdisease(2) |              0.0000 |
+----------------+---------------------+
| heartdisease(3) |              0.0000 |
+----------------+---------------------+
| heartdisease(4) |              0.0000 |
+----------------+---------------------+
```

In [11]:

```python
#prog9
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
# import some data to play with
iris = datasets.load_iris()
X = pd.DataFrame(iris.data) #print(X)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
# Build the K Means Model
model = KMeans(n_clusters=3)
model.fit(X) # model.labels_ : Gives cluster no for which samples belongs to # # Vis
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications using Petal features
plt.subplot(2, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
# Plot the Models Classifications
plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width') # General EM for GMM

from sklearn import preprocessing
```

```
# transform your data such that its distribution will have a # mean value 0 and stan
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('Observation: The GMM using EM algorithm based clustering matched the true lab
```

```
Observation: The GMM using EM algorithm based clustering matched the true labels mor
e closely than the Kmeans.
```



In [13]:

```
#prog10
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

iris=datasets.load_iris()
print("Iris Data set loaded...")
iris_data=iris.data
```

```
iris_labels=iris.target #print(iris_data) #print(iris_labels)
x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_labels,test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)

# Prints Label no. and their names
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))

classifier=KNeighborsClassifier(n_neighbors=1)
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)

# Display the results
print("Results of Classification using K-nn with K=1 ")
for r in range(0,len(x_test)):
    print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r]), " Predicted-
print("Classification Accuracy :" , classifier.score(x_test,y_test))
```

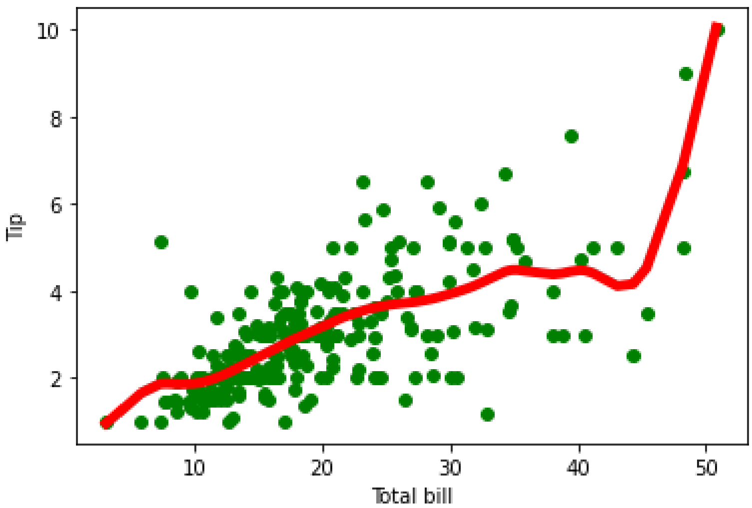
```
Iris Data set loaded...
Dataset is split into training and testing...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
 Sample: [7.  3.2 4.7 1.4]  Actual-label: 1  Predicted-label: 1
 Sample: [6.3 2.5 5.  1.9]  Actual-label: 2  Predicted-label: 2
 Sample: [7.9 3.8 6.4 2. ]  Actual-label: 2  Predicted-label: 2
 Sample: [6.4 3.2 5.3 2.3]  Actual-label: 2  Predicted-label: 2
 Sample: [6.  2.2 5.  1.5]  Actual-label: 2  Predicted-label: 1
 Sample: [6.8 3.  5.5 2.1]  Actual-label: 2  Predicted-label: 2
 Sample: [5.6 3.  4.5 1.5]  Actual-label: 1  Predicted-label: 1
 Sample: [4.9 3.  1.4 0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [5.9 3.  4.2 1.5]  Actual-label: 1  Predicted-label: 1
 Sample: [5.7 3.8 1.7 0.3]  Actual-label: 0  Predicted-label: 0
 Sample: [5.  3.4 1.5 0.2]  Actual-label: 0  Predicted-label: 0
 Sample: [6.5 2.8 4.6 1.5]  Actual-label: 1  Predicted-label: 1
 Sample: [5.2 2.7 3.9 1.4]  Actual-label: 1  Predicted-label: 1
 Sample: [5.7 4.4 1.5 0.4]  Actual-label: 0  Predicted-label: 0
 Sample: [4.9 3.6 1.4 0.1]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.9333333333333333
```

In [14]:

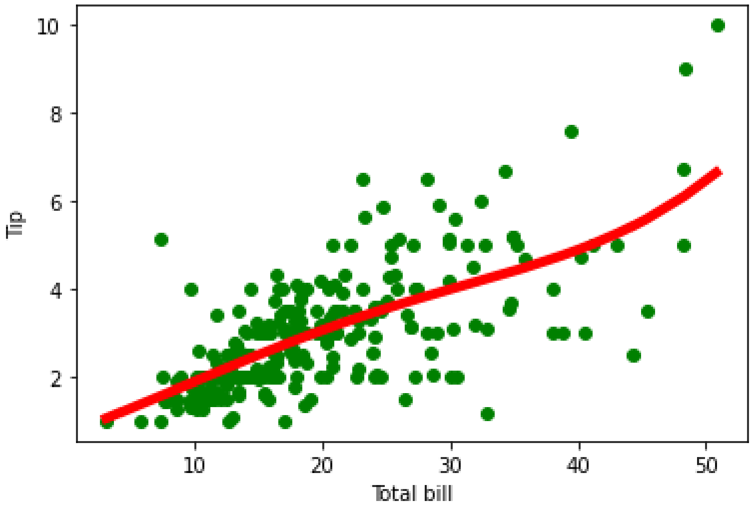
```
#prog11
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    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 = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();
# load data points
data = pd.read_csv('P10_data10_tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)
mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

print('Regression with parameter k = 2')
ypred = localWeightRegression(X,mtip,2) # increase k to get smooth curves
graphPlot(X,ypred)

print('Regression with parameter k = 10')
ypred = localWeightRegression(X,mtip,10) # increase k to get smooth curves
graphPlot(X,ypred)
```

```
Regression with parameter k = 2
```




```
Regression with parameter k = 10
```


10
8
6
4
2
Tip
10
20
30
40
50
Total bill