In [21]:

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {} #store distance from starting node
    parents = {}# parents contains an adjacency map of all nodes
    #ditance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node


    while len(open_set) > 0:
        n = None
            #node with lowest f() is found
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v

        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                #nodes 'm' not in first and last set are added to first   #n is set its par
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                        #for each node m,compare its distance from start i.e g(m) to the
                else:
                    if g[m] > g[n] + weight:
                        #update g(m)
                        #change parent of m to n
                        g[m] = g[n] + weight
                        parents[m] =n

                 #if m in closed set,remove and add to open
                    if m in closed_set:
                        closed_set.remove(m)
                        open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None

 # if the current node is the stop_node
 # then we begin reconstructin the path from it to the start_node
        if n == stop_node:
            path = []

            while parents[n] != n :
                path.append(n)
                n = parents[n]

            path.append(start_node)

            path.reverse()

            print('Path found: {}'.format(path))
            return path
```

```
 # remove n from the open_list, and add it to closed_list   # because all of his neighbors
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None
```

In [20]:

```
#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
#for simplicity we ll consider heuristic distances given  #and this function returns heuris
def heuristic(n):
    H_dist = {
     'A': 11,
     'B': 6,
     'C': 99,
     'D': 1,
     'E': 7,
     'G': 0,

     }
    return H_dist[n]

#Describe your graph here
Graph_nodes = {
 'A': [('B', 2), ('E', 3)],
 'B': [('C', 1),('G', 9)],
 'C': None,
 'E': [('D', 6)],
 'D': [('G', 1)],

}
aStarAlgo('A', 'G')
```

```
Path found: ['A', 'E', 'D', 'G']
```

Out[20]:

```
['A', 'E', 'D', 'G']
```

In [23]:

```
#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None
#for simplicity we ll consider heuristic distances given  #and this function returns heuris
def heuristic(n):
    H_dist = {  'S': 8,'A':8,'B': 4,  'C': 3,  'D': 500,  'E': 600,  'G': 0,

     }
    return H_dist[n]

#Describe your graph here
Graph_nodes = {
  'S': [('A', 1), ('B', 5),('C',8)],
'A': [('D', 3), ('E', 7),('G',9)],
 'B': [('G', 4)],
 'C': [('G', 4)],
 'E': None,
 'D': None,

}
aStarAlgo('S', 'G')
```

```
Path found: ['S', 'B', 'G']
```

Out[23]:

```
['S', 'B', 'G']
```

In [ ]:

In [4]:

```
import numpy as np
import pandas as pd

data = pd.read_csv('prg1.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):
        print("For Loop Starts")
        if target[i] == "yes":
            print("If instance is Positive ")
            for x in range(len(specific_h)):

                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("If instance is Negative ")
            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)
        print("\n")
        print("\n")

    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)
print("Final General_h:", g_final)
```

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

```
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

In [ ]:

In [1]:

```
import numpy as np
import pandas as pd
def entropy(target_col):
    val,counts = np.unique(target_col,return_counts = True)
    ent = sum( (-counts[i]/np.sum(counts)) * np.log2( counts[i]/np.sum(counts) )
       for i in range(len(val)))
    return ent

def infoGain(data,features,target):
    te = entropy(data[target])
    val,counts = np.unique(data[features],return_counts = True)
    eg = sum((counts[i]/sum(counts)) * entropy(data[data[features] == val[i]][target] )
             for i in range(len(val)))
    InfoGain = te-eg
    return InfoGain

def ID3(data,features,target,pnode):
    if len(np.unique(data[target])) == 1:
        return np.unique(data[target])[0]
    elif len(features) == 0:
        return pnode
    else:
        pnode = np.unique(data[target])[np.argmax(np.unique(data[target])[1])]
        IG = [infoGain(data,f,target) for f in features]
        index = np.argmax(IG)
        col = features[index]
        tree = {col:{}}
        features = [f for f in features if f!=col]
        for val in np.unique(data[col]):
            sub_data = data[data[col]==val].dropna()
            subtree = ID3(sub_data,features,target,pnode)
            tree[col][val] = subtree
        return tree


data = pd.read_csv('PlayTennis.csv')
testData = data.sample(frac = 0.1)
data.drop(testData.index,inplace = True)
print(data)
target = 'PlayTennis'
features = data.columns[data.columns!=target]
tree = ID3(data,features,target,None)
print (tree)
test = testData.to_dict('records')[0]
print(test,'=>', test['PlayTennis'])
```

```
     Outlook Temperature Humidity    Wind PlayTennis
0      Sunny         Hot     High    Weak         No
1      Sunny         Hot     High  Strong         No
2   Overcast         Hot     High    Weak        Yes
3       Rain        Mild     High    Weak        Yes
4       Rain        Cool   Normal    Weak        Yes
5       Rain        Cool   Normal  Strong         No
7      Sunny        Mild     High    Weak         No
8      Sunny        Cool   Normal    Weak        Yes
9       Rain        Mild   Normal    Weak        Yes
10     Sunny        Mild   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
```

```
12  Overcast         Hot   Normal    Weak        Yes
13      Rain        Mild     High  Strong         No
{'Outlook': {'Overcast': 'Yes', 'Rain': {'Wind': {'Strong': 'No', 'Weak':
'Yes'}}, 'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
{'Outlook': 'Overcast', 'Temperature': 'Cool', 'Humidity': 'Normal', 'Win
d': 'Strong', 'PlayTennis': 'Yes'} => Yes
```

In [ ]:

5. backpropagation

In [1]:

```
import numpy as np # numpy is commonly used to process number array
X = np.array([[2,9], [3,6], [4,8]]) # Features ( Hrs Slept, Hrs Studied)
y = np.array([[92], [86], [89]]) # Labels(Marks obtained)
X = X/np.amax(X,axis=0) # Normalize
y = y/100

def sigmoid(x):
    return 1/(1 + np.exp(-x))

def sigmoid_grad(x):
    return x * (1 - x)
# Variable initialization
epoch=1000 #Setting training iterations
eta =0.1 #Setting learning rate (eta)
input_neurons = 2 #number of features in data set
hidden_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
# Weight and bias - Random initialization
wh=np.random.uniform(size=(input_neurons,hidden_neurons)) # 2x3
bh=np.random.uniform(size=(1,hidden_neurons)) # 1x3
wout=np.random.uniform(size=(hidden_neurons,output_neurons)) # 1x1
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):#Forward Propogation
    h_ip=np.dot(X,wh) + bh # Dot product + bias
    h_act = sigmoid(h_ip) # Activation function
    o_ip=np.dot(h_act,wout) + bout
    output = sigmoid(o_ip)# Error at Output layer
    Eo = y-output # Error at o/p
    outgrad = sigmoid_grad(output)
    d_output = Eo* outgrad # Errj=Oj(1-Oj)(Tj-Oj)
# Error at Hidden later
    Eh = np.dot(d_output,wout.T) # .T means transpose
    hiddengrad = sigmoid_grad(h_act) # How much hidden layer wts contributed to error
    d_hidden = Eh * hiddengrad
    wout += np.dot(h_act.T,d_output) *eta # Dotproduct of nextlayererror and currentlayerop
    wh += np.dot(X.T,d_hidden) *eta
print("Normalized Input: \n" ,X)
print("Actual Output: \n" ,y)
print("Predicted Output: \n" ,output)
```

```
('Normalized Input: \n', array([[0, 1],
       [0, 0],
       [1, 0]]))
('Actual Output: \n', array([[0],
       [0],
       [0]]))
('Predicted Output: \n', array([[0.03188193],
       [0.05656392],
       [0.0272849 ]]))
```

In [ ]:

6. Naive bayes

In [2]:

```
import csv
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB

data = pd.read_csv('prima_indian_diabetes.csv')

x = np.array(data.iloc[:,0:-1])
y = np.array(data.iloc[:,-1])

print(data.head())

#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(x,y)

#Predict Output
predicted= model.predict([[6,149,78,35,0,34,0.625,54]])

print("Predicted Value:", predicted)
```

```
   pregnancies  Glucose  BloodPressure  SkinThickness  Insuline   BMI  \
0            6      148             72             35         0  33.6
1            1       85             66             29         0  26.6
2            8      183             64              0         0  23.3
3            1       89             66             23        94  28.1
4            0      137             40             35       168  43.1

   DiabeticPed  Age  Outcome
0        0.627   50        1
1        0.351   31        0
2        0.672   32        1
3        0.167   21        0
4        2.288   33        1
Predicted Value: [1]
```

7.

K Means Clustering

- It tries to minimize the distance of the points in a cluster

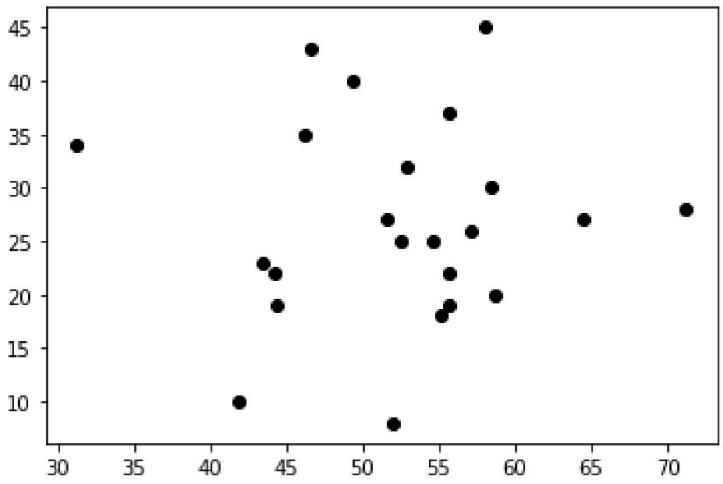
The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.

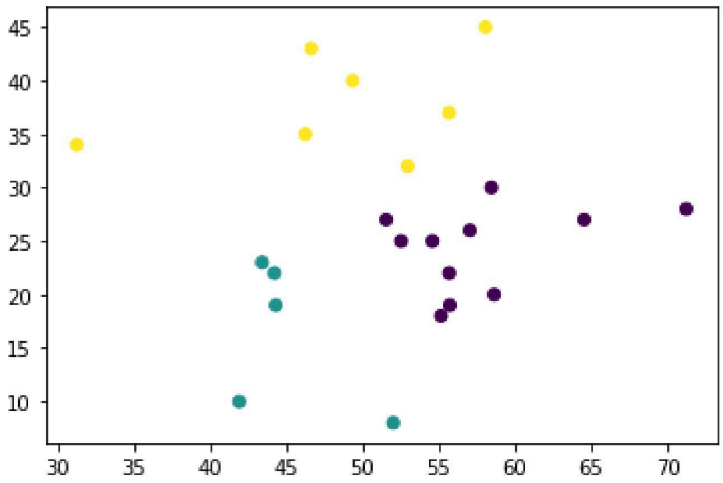
Steps in K means clustering

- Step 1 : Choose the number of clusters K
- Step 2 : Select k random points from data as centroids
- Step 3 : Assign all the points to the closest cluster centroid
- Step 4 : Recompute the centroids of newly formed clusters
- Step 5 : Repeat steps 3 and 4

In [2]:

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data = pd.read_csv('8-kmeansdata.csv')
f1 =data['Distance_Feature']
f2=data['Speeding_Feature']
X =np.array(list(zip(f1,f2)))
plt.scatter(f1,f2,color='black')
plt.show()
kmeans = KMeans(3).fit(X)
labels = kmeans.predict(X)
plt.scatter(f1,f2,c=labels)
plt.show()
gm = GaussianMixture(3).fit(X)
labels = gm.predict(X)
plt.scatter(f1,f2,c=labels)
plt.show()
```





8. KNN

In [1]:

```
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...")
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target)
classifier = KNeighborsClassifier(3).fit(x_train, y_train)
y_pred=classifier.predict(x_test)
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]), " Predictedlabel:",str
print("Classification Accuracy :" , classifier.score(x_test,y_test));
```

```
Iris Data set loaded...
Results of Classification using K-nn with K=1
 Sample: [6.1 2.6 5.6 1.4]  Actual-label: 2  Predictedlabel: 2
 Sample: [5.1 3.4 1.5 0.2]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.1 2.9 4.7 1.4]  Actual-label: 1  Predictedlabel: 1
 Sample: [4.9 3.1 1.5 0.2]  Actual-label: 0  Predictedlabel: 0
 Sample: [5.5 2.4 3.7 1. ]  Actual-label: 1  Predictedlabel: 1
 Sample: [4.8 3.  1.4 0.3]  Actual-label: 0  Predictedlabel: 0
 Sample: [5.7 3.8 1.7 0.3]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.5 3.  5.5 1.8]  Actual-label: 2  Predictedlabel: 2
 Sample: [5.4 3.4 1.5 0.4]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.4 3.1 5.5 1.8]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.  2.2 4.  1. ]  Actual-label: 1  Predictedlabel: 1
 Sample: [4.9 3.6 1.4 0.1]  Actual-label: 0  Predictedlabel: 0
 Sample: [7.9 3.8 6.4 2. ]  Actual-label: 2  Predictedlabel: 2
 Sample: [4.8 3.1 1.6 0.2]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.8 2.8 4.8 1.4]  Actual-label: 1  Predictedlabel: 1
 Sample: [5.6 2.9 3.6 1.3]  Actual-label: 1  Predictedlabel: 1
 Sample: [4.6 3.4 1.4 0.3]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.7 3.3 5.7 2.1]  Actual-label: 2  Predictedlabel: 2
 Sample: [5.6 2.7 4.2 1.3]  Actual-label: 1  Predictedlabel: 1
 Sample: [5.8 2.7 4.1 1. ]  Actual-label: 1  Predictedlabel: 1
 Sample: [5.8 2.7 5.1 1.9]  Actual-label: 2  Predictedlabel: 2
 Sample: [4.9 3.1 1.5 0.1]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.7 3.  5.  1.7]  Actual-label: 1  Predictedlabel: 1
 Sample: [6.7 3.3 5.7 2.5]  Actual-label: 2  Predictedlabel: 2
 Sample: [5.5 2.4 3.8 1.1]  Actual-label: 1  Predictedlabel: 1
 Sample: [5.8 2.8 5.1 2.4]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.8 3.  5.5 2.1]  Actual-label: 2  Predictedlabel: 2
 Sample: [7.2 3.  5.8 1.6]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.3 2.9 5.6 1.8]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.1 3.  4.9 1.8]  Actual-label: 2  Predictedlabel: 2
 Sample: [7.1 3.  5.9 2.1]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.7 3.  5.2 2.3]  Actual-label: 2  Predictedlabel: 2
 Sample: [4.7 3.2 1.3 0.2]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.  2.9 4.5 1.5]  Actual-label: 1  Predictedlabel: 1
 Sample: [6.3 2.3 4.4 1.3]  Actual-label: 1  Predictedlabel: 1
 Sample: [4.9 2.4 3.3 1. ]  Actual-label: 1  Predictedlabel: 1
 Sample: [6.5 2.8 4.6 1.5]  Actual-label: 1  Predictedlabel: 1
 Sample: [6.4 2.8 5.6 2.2]  Actual-label: 2  Predictedlabel: 2
Classification Accuracy : 1.0
```

Exploration of IRIS dataset

In [12]:

```
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 set loaded...
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
['setosa' 'versicolor' 'virginica']
```

In [13]:

```
# To print data dimensions or size
print(iris.data.shape)
```

```
(150, 4)
```

150 rows and 4 columns

In [14]:

```
# To print feature names
print(iris.feature_names)
#print(iris.data) #to view data
print(iris.target_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
['setosa' 'versicolor' 'virginica']
```

In [16]:

```
print(iris.data)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
```

In [17]:

```
print(iris.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 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 2]
```

KNN Steps

Computes the distance between the new data point with every training example.

For computing the distance measures such as Euclidean distance, Hamming distance or Manhattan distance will be used.

Model picks K entries in the database which are closest to the new data point.

Then it does the majority vote i.e the most common class/label among those K entries will be the class of the new data point.

Step 1 : Load Iris Data and explore features

Step 2 : Split the data and Train the Model

In [20]:

```
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target)
```

In [21]:

```
#Shape of Train and Test objects
print(x_train.shape)
print(x_test.shape)
```

```
(112, 4)
(38, 4)
```

In [22]:

```
# Shape of new y objects
print(y_train.shape)
print(y_test.shape)
```

```
(112,)
(38,)
```

Import the class 'KNeighborsClassifer' from 'neighbors' module

In [23]:

```
classifier = KNeighborsClassifier(3).fit(x_train, y_train)
```

fit' method is used to train the model on training data (X_train,y_train)

'predict' method to do the testing on testing data (X_test)

In [25]:

```
y_pred=classifier.predict(x_test)
```

In [26]:

```
print(len(x_test))
```

```
38
```

In [27]:

```
for r in range(0,len(x_test)):
 print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r]), " Predictedlabel:",str
print("Classification Accuracy :" , classifier.score(x_test,y_test));
```

```
 Sample: [6.5 3.  5.8 2.2]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.7 3.3 5.7 2.5]  Actual-label: 2  Predictedlabel: 2
 Sample: [5.5 2.6 4.4 1.2]  Actual-label: 1  Predictedlabel: 1
 Sample: [5.4 3.9 1.7 0.4]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.9 3.1 5.1 2.3]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.2 2.9 4.3 1.3]  Actual-label: 1  Predictedlabel: 1
 Sample: [4.9 3.  1.4 0.2]  Actual-label: 0  Predictedlabel: 0
 Sample: [5.6 2.8 4.9 2. ]  Actual-label: 2  Predictedlabel: 2
 Sample: [5.7 3.  4.2 1.2]  Actual-label: 1  Predictedlabel: 1
 Sample: [5.7 2.8 4.1 1.3]  Actual-label: 1  Predictedlabel: 1
 Sample: [7.4 2.8 6.1 1.9]  Actual-label: 2  Predictedlabel: 2
 Sample: [4.9 2.5 4.5 1.7]  Actual-label: 2  Predictedlabel: 1
 Sample: [6.  2.9 4.5 1.5]  Actual-label: 1  Predictedlabel: 1
 Sample: [6.3 2.5 4.9 1.5]  Actual-label: 1  Predictedlabel: 2
 Sample: [5.5 2.3 4.  1.3]  Actual-label: 1  Predictedlabel: 1
 Sample: [5.6 2.7 4.2 1.3]  Actual-label: 1  Predictedlabel: 1
 Sample: [6.3 2.7 4.9 1.8]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.2 3.4 5.4 2.3]  Actual-label: 2  Predictedlabel: 2
 Sample: [5.  3.  1.6 0.2]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.8 3.2 5.9 2.3]  Actual-label: 2  Predictedlabel: 2
 Sample: [5.4 3.9 1.3 0.4]  Actual-label: 0  Predictedlabel: 0
 Sample: [5.4 3.4 1.5 0.4]  Actual-label: 0  Predictedlabel: 0
 Sample: [5.  3.2 1.2 0.2]  Actual-label: 0  Predictedlabel: 0
 Sample: [4.4 2.9 1.4 0.2]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.4 2.9 4.3 1.3]  Actual-label: 1  Predictedlabel: 1
 Sample: [7.2 3.2 6.  1.8]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.2 2.8 4.8 1.8]  Actual-label: 2  Predictedlabel: 2
 Sample: [4.5 2.3 1.3 0.3]  Actual-label: 0  Predictedlabel: 0
 Sample: [4.9 2.4 3.3 1. ]  Actual-label: 1  Predictedlabel: 1
 Sample: [6.6 3.  4.4 1.4]  Actual-label: 1  Predictedlabel: 1
 Sample: [6.1 2.8 4.7 1.2]  Actual-label: 1  Predictedlabel: 1
 Sample: [4.8 3.  1.4 0.3]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.2 2.2 4.5 1.5]  Actual-label: 1  Predictedlabel: 1
 Sample: [5.8 4.  1.2 0.2]  Actual-label: 0  Predictedlabel: 0
 Sample: [4.9 3.1 1.5 0.1]  Actual-label: 0  Predictedlabel: 0
 Sample: [6.7 3.1 4.7 1.5]  Actual-label: 1  Predictedlabel: 1
 Sample: [7.1 3.  5.9 2.1]  Actual-label: 2  Predictedlabel: 2
 Sample: [6.  2.7 5.1 1.6]  Actual-label: 1  Predictedlabel: 2
Classification Accuracy : 0.9210526315789473
```

In [42]:

```
x_test = [[3,4,5,2],[5,4,5,2]]
y_pred=classifier.predict(x_test)
print(y_pred)
```

```
[1 1]
```

In [49]:

```
x_test = [[5,4,5,2]]
y_pred=classifier.predict(x_test)
print(y_pred)
```

```
[1]
```

In [ ]:

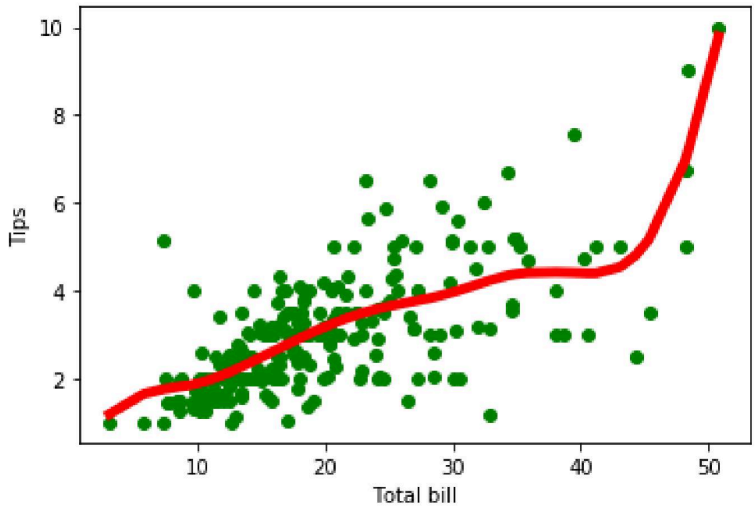
```

```

9. Locally Weighted Regression

In [3]:

```
# Importing Libraries

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


def localWeigh(point,X,ymat,k):
 m,n = np.shape(X)
#Initializing with identity matrix
 weights = np.mat(np.eye(m))

    #calculating weights for query points
 for i in range(m):
     diff = point - X[i]
     weights[i,i] = np.exp(diff*diff.T/(-2.0*k**2))
 W = (X.T *(weights*X)).I * (X.T*(weights*ymat.T))
 return W

# Function to make predictions
def localWeightReg(X,ymat,k):
 m,n = np.shape(X)
 ypred = np.zeros(m)
 for i in range(m):
     ypred[i] = X[i] * localWeigh(X[i],X,ymat,k)
 return ypred


def plott(X,pred):
 sortIndex = X[:,1].argsort(0)
 xsort = X[sortIndex][:,0][:,1]
 ysort = pred[sortIndex]
 plt.scatter(x,y,color='green')
 plt.plot(xsort,ysort,color="red",linewidth=5)
 plt.xlabel('Total bill')
 plt.ylabel('Tips')
 plt.show()

data = pd.read_csv('data10.csv')
x=data['total_bill']
y = data['tip']
xmat = np.mat(x)
ymat = np.mat(y)
size = np.shape(xmat)[1]
#print(xmat.shape)
ones = np.mat(np.ones(size))
X=np.hstack((ones.T,xmat.T))
pred = localWeightReg(X,ymat,3)
plott(X,pred)
```

```
(1, 244)
```



In [ ]: