ML LAB REPORT

NAME : YASHASWI R

USN : 1BM18CS154

Experiment 1: Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

Program:

```
import csv
a = []
with open('enjoysport.csv', 'r') as csvfile:
    next(csvfile)
    for row in csv.reader(csvfile):print("\n")
        a.append(row)

    print("\n",a)

print("\nThe total number of training instances are : ",len(a))

num_attribute = len(a[0])-1

print("\nThe initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)

for i in range(0, len(a)):
    if a[i][num_attribute] == 'yes':
        print ("\nInstance ", i+1, "is", a[i], " and is Positive Instance")
        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("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")

    if a[i][num_attribute] == 'no':
        print ("\nInstance ", i+1, "is", a[i], " and is Negative Instance Hence Ignored")
        print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")
```

print("\nThe Maximally specific hypothesis for the training instance is ", hypothesis)

Output:

```
PS C:\Users\BMSCE\Desktop\jagadeesh> python ml.py
[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes'],
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']]

The total number of training instances are :  4

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

Instance  1 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']  and is Positive Instance
The hypothesis for the training instance 1  is:  ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']


Instance  2 is ['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']  and is Positive Instance
The hypothesis for the training instance 2  is:  ['sunny', 'warm', '?', 'strong', 'warm', 'same']


Instance  3 is ['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']  and is Negative Instance Hence Ignored
The hypothesis for the training instance 3  is:  ['sunny', 'warm', '?', 'strong', 'warm', 'same']


Instance  4 is ['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']  and is Positive Instance
The hypothesis for the training instance 4  is:  ['sunny', 'warm', '?', 'strong', '?', '?']


The Maximally specific hypothesis for the training instance is  ['sunny', 'warm', '?', 'strong', '?', '?']
PS C:\Users\BMSCE\Desktop\jagadeesh>
```

Data Set:

enjoysport.csv

1	sky	airtemp	humidity	wind	water	forecast	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

Experiment 2: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Program:

```
import numpy as np
import pandas as pd

data = pd.read_csv('enjoysport.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, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '
?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '
?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?']]




For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '
?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?']]




For Loop Starts
If instance is Negative
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '
?', '?', '?', '?', 'same']]
```

```
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '
?', '?', '?', '?', '?']]




Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Output:

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Negative
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Dataset:

enjoysport.csv

1	sky	airtemp	humidity	wind	water	forecast	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

Experiment 3: 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.

Program:

```
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]
```

```python
    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("id3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3_test.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end="   ")
    classify(node1,xtest,features)
```

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   overcast
     yes
   rain
     Wind
       strong
         no
       weak
         yes
   sunny
     Humidity
       normal
         yes
       high
         no
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:   no
The test instance: ['sunny', 'mild', 'normal', 'strong']
```

```
The label for test instance:   yes
```

Output:

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   overcast
     yes
   rain
     Wind
       strong
         no
       weak
         yes
   sunny
     Humidity
       normal
         yes
       high
         no
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:   no
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance:   yes
```

Dataset:

id3.csv

1	Outlook	Temperature	Humidity	Wind	Answer
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no

id3_test.csv

1	Outlook	Temperature	Humidity	Wind
2	rain	cool	normal	strong
3	sunny	mild	normal	strong

Experiment 4: 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.

Program:

```
import pandas as pd

data = pd.read_csv('PlayTennis.csv')
data.head()
```

Out[1]:

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak

In [2]:

```
y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values

print(f'Target Values: {y}')
print(f'Features: \n{X}')
```

```
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes'
, 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
```

In [3]:

```
y_train = y[:8]
y_val = y[8:]

X_train = X[:8]
X_val = X[8:]

print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
```

```
Number of instances in training set: 8
Number of instances in testing set: 6
```

In [4]:

```
class NaiveBayesClassifier:


    def __init__(self, X, y):

        self.X, self.y = X, y

        self.N = len(self.X)

        self.dim = len(self.X[0])

        self.attrs = [[] for _ in range(self.dim)]

        self.output_dom = {}

        self.data = []

        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])

            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1

            else:
                self.output_dom[self.y[i]] += 1

            self.data.append([self.X[i], self.y[i]])
    def classify(self, entry):

        solve = None
        max_arg = -1

        for y in self.output_dom.keys():

            prob = self.output_dom[y]/self.N

            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N

            if prob > max_arg:
                max_arg = prob
```

```
                solve = y

        return solve
```

In [6]:

```
nbc = NaiveBayesClassifier(X_train, y_train)

total_cases = len(y_val)

good = 0
bad = 0
predictions = []

for i in range(total_cases):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)

    if y_val[i] == predict:
        good += 1
    else:
        bad += 1

print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)
```

```
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```

Output:

```
Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```

Dataset

PlayTennis.csv

1	PlayTennis	Outlook	Temperature	Humidity	Wind
2	No	Sunny	Hot	High	Weak
3	No	Sunny	Hot	High	Strong
4	Yes	Overcast	Hot	High	Weak
5	Yes	Rain	Mild	High	Weak
6	Yes	Rain	Cool	Normal	Weak
7	No	Rain	Cool	Normal	Strong
8	Yes	Overcast	Cool	Normal	Strong
9	No	Sunny	Mild	High	Weak
10	Yes	Sunny	Cool	Normal	Weak
11	Yes	Rain	Mild	Normal	Weak
12	Yes	Sunny	Mild	Normal	Strong
13	Yes	Overcast	Mild	High	Strong
14	Yes	Overcast	Hot	Normal	Weak
15	No	Rain	Mild	High	Strong

Experiment 4a: 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.

Program:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
```

In [4]:

```
df = pd.read_csv("data.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin'
, 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']
x = df[feature_col_names].values
y = df[predicted_class_names].values
print(df.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)
```

```
   num_preg  glucose_conc  diastolic_bp  thickness  insulin   bmi  diab_pred  \
0         6           148            72         35        0  33.6      0.627
1         1            85            66         29        0  26.6      0.351
2         8           183            64          0        0  23.3      0.672
3         1            89            66         23       94  28.1      0.167
4         0           137            40         35      168  43.1      2.288

   age  diabetes
0   50         1
1   31         0
2   32         1
3   21         0
4   33         1

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

In [5]:

```
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
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)
```

```
Confusion matrix
```

```
[[143  27]
 [ 37  47]]

Accuracy of the classifier: 0.7480314960629921
The value of Precision: 0.6351351351351351
The value of Recall: 0.5595238095238095
Predicted Value for individual Test Data: [1]
```

Output

```
Confusion matrix
[[143  27]
 [ 37  47]]

Accuracy of the classifier: 0.7480314960629921
The value of Precision: 0.6351351351351351
The value of Recall: 0.5595238095238095
Predicted Value for individual Test Data: [1]
```

Experiment 5: Write a program to construct a Bayesian network considering training data. Use this model to make predictions. With built in functions

Program

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
```

In [8]:

```
heartDisease = pd.read_csv('heart.csv')
```

In [9]:

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

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

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

In [10]:

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

```
 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
```

In [11]:

```
model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])
```

In [12]:

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

```
 Learning CPD using Maximum likelihood estimators
```

In [13]:

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

```
 Inferencing with Bayesian Network:
```

In [14]:

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

```
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 2505.56it/s]
Eliminating: age: 100%|██████████| 5/5 [00:00<00:00, 138.77it/s]
 1.Probability of HeartDisease given evidence=restecg :1
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1016 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2361 |
+-----------------+---------------------+
| heartdisease(3) |              0.2017 |
+-----------------+---------------------+
| heartdisease(4) |              0.4605 |
+-----------------+---------------------+
```

In [15]:

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

```
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 2524.56it/s]
Eliminating: restecg: 100%|██████████| 5/5 [00:00<00:00, 240.90it/s]
 2.Probability of HeartDisease given evidence= cp:2
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3742 |
+-----------------+---------------------+
| heartdisease(1) |              0.2018 |
+-----------------+---------------------+
```

```
| heartdisease(2) |              0.1375 |
+-----------------+---------------------+
| heartdisease(3) |              0.1541 |
+-----------------+---------------------+
| heartdisease(4) |              0.1323 |
+-----------------+---------------------+
```

Output

```
In [14]: print('\n 1.Probability of HeartDisease given evidence=restecg :1')
         q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
         print(q1)
```

```
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 2505.56it/s]
Eliminating: age: 100%|██████████| 5/5 [00:00<00:00, 138.77it/s]

 1.Probability of HeartDisease given evidence=restecg :1
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1016 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2361 |
+-----------------+---------------------+
| heartdisease(3) |              0.2017 |
+-----------------+---------------------+
| heartdisease(4) |              0.4605 |
+-----------------+---------------------+
```

```
In [15]: print('\n 2.Probability of HeartDisease given evidence= cp:2 ')
         q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
         print(q2)
```

```
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 2524.56it/s]
Eliminating: restecg: 100%|██████████| 5/5 [00:00<00:00, 240.90it/s]

 2.Probability of HeartDisease given evidence= cp:2
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3742 |
+-----------------+---------------------+
| heartdisease(1) |              0.2018 |
+-----------------+---------------------+
| heartdisease(2) |              0.1375 |
+-----------------+---------------------+
| heartdisease(3) |              0.1541 |
+-----------------+---------------------+
| heartdisease(4) |              0.1323 |
+-----------------+---------------------+
```

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

Experiment 5a: Write a program to construct a Bayesian network considering training data. Use this model to make predictions. Without built in functions

Program:

```
import bayespy as bp
import numpy as np
import pandas as pd
import csv
from colorama import init
from colorama import Fore, Back, Style
init()

# Define Parameter Enum values
# Age
ageEnum = {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1,
           'MiddleAged': 2, 'Youth': 3, 'Teen': 4}
# Gender
genderEnum = {'Male': 0, 'Female': 1}
# FamilyHistory
familyHistoryEnum = {'Yes': 0, 'No': 1}
# Diet(Calorie Intake)
dietEnum = {'High': 0, 'Medium': 1, 'Low': 2}
# LifeStyle
lifeStyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}
# Cholesterol
cholesterolEnum = {'High': 0, 'BorderLine': 1, 'Normal': 2}
# HeartDisease
heartDiseaseEnum = {'Yes': 0, 'No': 1}
```

In [18]:

```
data = pd.read_csv("heart.csv")
data =np.array(data, dtype='int8')
N = len(data)
```

In [19]:

```
# Input data column assignment
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:, 0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:, 1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:, 2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
```

```
diet.observe(data[:, 3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:, 4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:, 5])
```

In [20]:

```
# Prepare nodes and establish edges
# np.ones(2) -> HeartDisease has 2 options Yes/No
# plates(5, 2, 2, 3, 4, 3) -> corresponds to options present for domain values
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.MultiMixture(
    [age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Categorical, p
_heartdisease)
heartdisease.observe(data[:, 6])
p_heartdisease.update()
```

In [21]:

```
#print("Sample Probability")
#print("Probability(HeartDisease|Age=SuperSeniorCitizen, Gender=Female, FamilyHistory=Y
es, DietIntake=Medium, LifeStyle=Sedetary, Cholesterol=High)")
#print(bp.nodes.MultiMixture([ageEnum['SuperSeniorCitizen'], genderEnum['Female'], fami
lyHistoryEnum['Yes'], dietEnum['Medium'], lifeStyleEnum['Sedetary'], cholesterolEnum['H
igh']], bp.nodes.Categorical, p_heartdisease).get_moments()[0] [heartDiseaseEnum['Yes']
])

# Interactive Test
m = 0
while m == 0:
    print("\n")
    res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('E
nter Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' + str(familyHisto
ryEnum))), int(input('Enter dietEnum: ' + str(
        dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('E
nter Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categorical, p_heartdisease).ge
t_moments()[0][heartDiseaseEnum['Yes']]
    print("Probability(HeartDisease) = " + str(res))

# print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1 "))
```

```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, '
Teen': 4}0
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}1
```

```
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}2
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 0


Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, '
Teen': 4}4
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}1
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 1
```

Output:

```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}0
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}1
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}2
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 0

Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}4
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}1
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}2
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}2
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 1
```

Dataset

heart.csv

1	age	Gender	Family	diet	Lifestyle	cholestrol	heartdisease
2	0	0	1	1	3	0	1
3	0	1	1	1	3	0	1
4	1	1	0	0	2	1	1
5	4	0	1	1	3	2	0
6	3	1	1	0	0	2	0
7	2	0	1	1	1	0	1
8	4	0	1	0	2	0	1
9	0	0	1	1	3	0	1
10	3	1	1	0	0	2	0
11	1	1	0	0	0	2	1
12	4	1	0	1	2	0	1
13	4	0	1	1	3	2	0
14	2	1	0	0	0	0	0
15	2	0	1	1	1	0	1
16	3	1	1	0	0	1	0
17	0	0	1	0	0	2	1
18	1	1	0	1	2	1	1
19	3	1	1	1	0	1	0
20	4	0	1	1	3	2	0

Experiment 5b: Program for the illustration of Bayesian Belief networks using 5 nodes using Lung cancer data. (The Conditional probabilities are given)

Program

```
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
```

In [7]:

```
#DefineaStructure with nodes and edge
cancer_model=BayesianModel([('Pollution', 'Cancer'),
                ('Smoker', 'Cancer'),
                ('Cancer', 'Xray'),
                ('Cancer', 'Dyspnoea')])
```

In [8]:

```
print("Baysian network nodes are:")
print("\t",cancer_model.nodes())
print("Baysian network edges are:")
print('\t',cancer_model.edges())
```

```
Baysian network nodes are:
         ['Pollution', 'Cancer', 'Smoker', 'Xray', 'Dyspnoea']
Baysian network edges are:
         [('Pollution', 'Cancer'), ('Cancer', 'Xray'), ('Cancer', 'Dyspnoea'), ('Smoker
', 'Cancer')]
```

In [13]:

```
#Creation of Conditional Probability Table
cpd_poll = TabularCPD(variable='Pollution', variable_card=2,
             values=[[0.9],[0.1]])
cpd_smoke = TabularCPD(variable='Smoker', variable_card=2,
              values=[[0.3],[0.7]])
cpd_cancer = TabularCPD(variable='Cancer', variable_card=2,
               values=[[0.03, 0.05, 0.001, 0.02],
                       [0.97, 0.95, 0.999, 0.98]],
                        evidence=['Smoker','Pollution'],
                        evidence_card=[2, 2])
cpd_xray = TabularCPD(variable='Xray', variable_card=2,
             values=[[0.9, 0.2],[0.1, 0.8]],
             evidence=['Cancer'], evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2,
             values=[[0.65, 0.3],[0.35, 0.7]],
             evidence=['Cancer'], evidence_card=[2])
```

In [15]:

```
# Associating the parameters withthe model structure
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)
print('Model generated by adding conditional probability disttributions(cpds)')
```

```
Model generated by adding conditional probability disttributions(cpds)
```

In [21]:

```
# Checking if the cpds are valid for the model.
```

```python
print('Checking for Correctness of model :',end='' )
print(cancer_model.check_model())
'''print('All local idependencies are as follows')
cancer_model.get_independencies()
'''
print('Displaying CPDs')
print(cancer_model.get_cpds('Pollution'))
print(cancer_model.get_cpds('Smoker'))
print(cancer_model.get_cpds('Cancer'))
print(cancer_model.get_cpds('Xray'))
print(cancer_model.get_cpds('Dyspnoea'))
```

```
Checking for Correctness of model :True
Displaying CPDs
+--------------+-----+
| Pollution(0) | 0.9 |
+--------------+-----+
| Pollution(1) | 0.1 |
+--------------+-----+
+-----------+-----+
| Smoker(0) | 0.3 |
+-----------+-----+
| Smoker(1) | 0.7 |
+-----------+-----+
+-----------+--------------+--------------+--------------+--------------+
| Smoker    | Smoker(0)    | Smoker(0)    | Smoker(1)    | Smoker(1)    |
+-----------+--------------+--------------+--------------+--------------+
| Pollution | Pollution(0) | Pollution(1) | Pollution(0) | Pollution(1) |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(0) | 0.03         | 0.05         | 0.001        | 0.02         |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(1) | 0.97         | 0.95         | 0.999        | 0.98         |
+-----------+--------------+--------------+--------------+--------------+
+---------+-----------+-----------+
| Cancer  | Cancer(0) | Cancer(1) |
+---------+-----------+-----------+
| Xray(0) | 0.9       | 0.2       |
+---------+-----------+-----------+
| Xray(1) | 0.1       | 0.8       |
+---------+-----------+-----------+
+-------------+-----------+-----------+
| Cancer      | Cancer(0) | Cancer(1) |
+-------------+-----------+-----------+
| Dyspnoea(0) | 0.65      | 0.3       |
+-------------+-----------+-----------+
| Dyspnoea(1) | 0.35      | 0.7       |
+-------------+-----------+-----------+
```

In [22]:

```python
# #Inferencing with Bayesian Network
```

```python
# Computing the probability of Cancer given smoke.
cancer_infer = VariableElimination(cancer_model)

print('\ninferencing with Bayesian Network');
```

```
inferencing with Bayesian Network
```

In [29]:

```python
print('\n Probability of Cancer given Smoker')
q=cancer_infer.query(variables=['Cancer'],evidence={'Smoker': 1})
print(q)
```

```
Finding Elimination Order: : 100%|██████████| 3/3 [00:00<00:00, 1504.23it/s]
Eliminating: Xray: 100%|██████████| 3/3 [00:00<00:00, 752.07it/s]
 Probability of Cancer given Smoker
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 |
+-----------+---------------+
| Cancer(1) |        0.9971 |
+-----------+---------------+
```

In [31]:

```python
print('\nProbability of Cancer given Smoker,Pollution')
q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1,'Pollution': 1})
print(q)
```

```
Finding Elimination Order: : 100%|██████████| 2/2 [00:00<?, ?it/s]
Eliminating: Xray: 100%|██████████| 2/2 [00:00<00:00, 1003.06it/s]
Probability of Cancer given Smoker,Pollution
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 |
+-----------+---------------+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```

Output:

```
In [29]: print('\n Probability of Cancer given Smoker')
         q=cancer_infer.query(variables=['Cancer'],evidence={'Smoker': 1})
         print(q)

Finding Elimination Order: : 100%|██████████| 3/3 [00:00<00:00, 1504.23it/s]
Eliminating: Xray: 100%|██████████| 3/3 [00:00<00:00, 752.07it/s]

 Probability of Cancer given Smoker
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 |
+-----------+---------------+
| Cancer(1) |        0.9971 |
+-----------+---------------+
```

```
In [31]: print('\nProbability of Cancer given Smoker,Pollution')
         q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1,'Pollution': 1})
         print(q)

Finding Elimination Order: : 100%|██████████| 2/2 [00:00<?, ?it/s]
Eliminating: Xray: 100%|██████████| 2/2 [00:00<00:00, 1003.06it/s]

Probability of Cancer given Smoker,Pollution
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 |
+-----------+---------------+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```

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

K-Means Clustering

In [35]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

In [36]:
```
df = pd.read_csv('income.csv')
df.head(10)
```

Out[36]:

	Name	Age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000
5	Gautam	39	155000
6	David	41	160000
7	Andrea	38	162000
8	Brad	36	156000
9	Angelina	35	130000

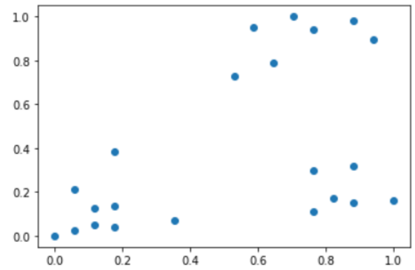
In [37]:
```
scaler = MinMaxScaler()
scaler.fit(df[['Age']])
df[['Age']] = scaler.transform(df[['Age']])

scaler.fit(df[['Income($)']])
df[['Income($)']] = scaler.transform(df[['Income($)']])
df.head(10)
```

Out[37]:

	Name	Age	Income($)
0	Rob	0.058824	0.213675
1	Michael	0.176471	0.384615
2	Mohan	0.176471	0.136752
3	Ismail	0.117647	0.128205
4	Kory	0.941176	0.897436
5	Gautam	0.764706	0.940171
6	David	0.882353	0.982906
7	Andrea	0.705882	1.000000
8	Brad	0.588235	0.948718
9	Angelina	0.529412	0.726496

```
In [38]: plt.scatter(df['Age'], df['Income($)'])
```

```
Out[38]: <matplotlib.collections.PathCollection at 0x7f23ce044f10>
```



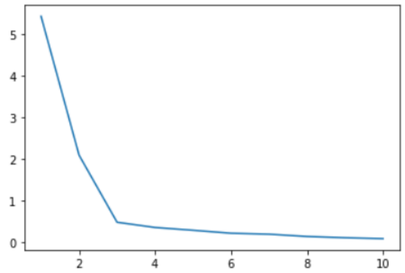
Finding Elbow Point

```
In [50]: k_range = range(1, 11)
         sse = []
         for k in k_range:
             kmc = KMeans(n_clusters=k)
             kmc.fit(df[['Age', 'Income($)']])
             sse.append(kmc.inertia_)
         sse
```

```
Out[50]: [5.434011511988179,
          2.091136388699078,
          0.4750783498553097,
          0.3491047094419566,
          0.2818479744366238,
          0.21055478995472496,
          0.18752738899206242,
          0.13265419827245162,
          0.10188787724979426,
          0.08026197041664467]
```

```
In [52]: plt.xlabel = 'Number of Clusters'
         plt.ylabel = 'Sum of Squared Errors'
         plt.plot(k_range, sse)
```

```
Out[52]: [<matplotlib.lines.Line2D at 0x7f23cb541c10>]
```



Therefore, the elbow point is 3

```
In [39]: km = KMeans(n_clusters=3)
         km
```

```
Out[39]: KMeans(n_clusters=3)
```

```
In [40]: y_predict = km.fit_predict(df[['Age', 'Income($)']])
         y_predict
```

```
Out[40]: array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2],
               dtype=int32)
```

In [41]:
```
df['cluster'] = y_predict
df.head()
```

Out[41]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
4	Kory	0.941176	0.897436	1

In [42]:
```
df0 = df[df.cluster == 0]
df0
```

Out[42]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
11	Tom	0.000000	0.000000	0
12	Arnold	0.058824	0.025641	0
13	Jared	0.117647	0.051282	0
14	Stark	0.176471	0.038462	0
15	Ranbir	0.352941	0.068376	0

In [44]:
```
df1 = df[df.cluster == 1]
df1
```

Out[44]:

	Name	Age	Income($)	cluster
4	Kory	0.941176	0.897436	1
5	Gautam	0.764706	0.940171	1
6	David	0.882353	0.982906	1
7	Andrea	0.705882	1.000000	1
8	Brad	0.588235	0.948718	1
9	Angelina	0.529412	0.726496	1
10	Donald	0.647059	0.786325	1

In [45]:
```
df2 = df[df.cluster == 2]
df2
```

Out[45]:

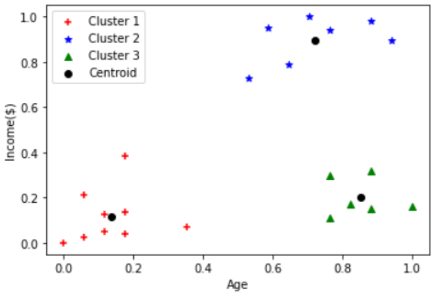
	Name	Age	Income($)	cluster
16	Dipika	0.823529	0.170940	2
17	Priyanka	0.882353	0.153846	2
18	Nick	1.000000	0.162393	2
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	2

In [47]:
```
km.cluster_centers_
```

Out[47]:
```
array([[0.1372549 , 0.11633428],
       [0.72268908, 0.8974359 ],
       [0.85294118, 0.2022792 ]])
```

In [49]:
```
p1 = plt.scatter(df0['Age'], df0['Income($)'], marker='+', color='red')
p2 = plt.scatter(df1['Age'], df1['Income($)'], marker='*', color='blue')
p3 = plt.scatter(df2['Age'], df2['Income($)'], marker='^', color='green')
c = plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color='black')
plt.xlabel('Age')
plt.ylabel('Income($)')
plt.legend((p1, p2, p3, c),
          ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))
```

Out[49]: <matplotlib.legend.Legend at 0x7f23cb75d910>

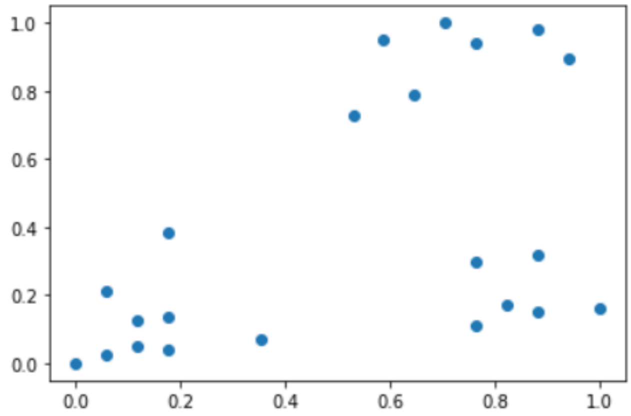


Data set

	Name	Age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000
5	Gautam	39	155000
6	David	41	160000
7	Andrea	38	162000
8	Brad	36	156000
9	Angelina	35	130000

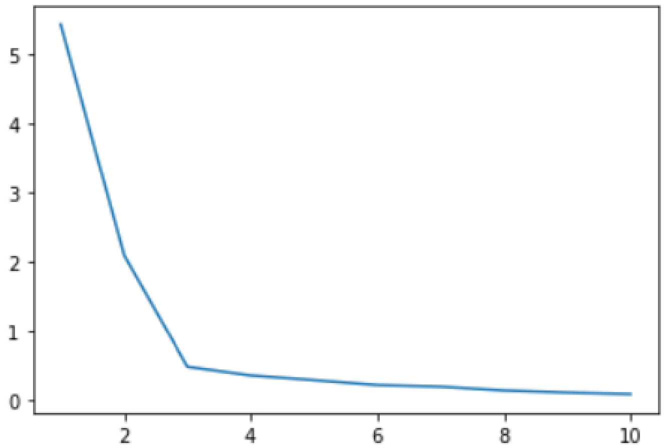
Outputs:

Out[38]: <matplotlib.collections.PathCollection at 0x7f23ce044f10>



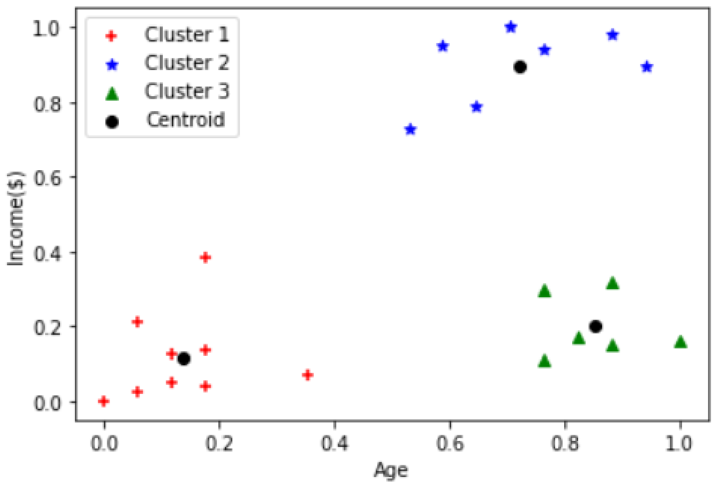
Finding Elbow Point

Out[52]: [<matplotlib.lines.Line2D at 0x7f23cb541c10>]



Therefore, the elbow point is 3

Out[49]: <matplotlib.legend.Legend at 0x7f23cb75d910>



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:

jupyter EM algorithm (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [1]:
```
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
```

In [2]:
```
iris = datasets.load_iris()
```

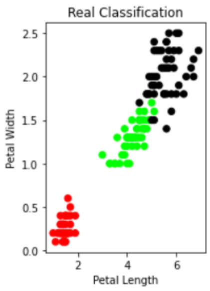
In [3]:
```
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

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

In [4]:
```
model = KMeans(n_clusters=3)
model.fit(X)


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

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

```
<Figure size 1008x504 with 0 Axes>
```

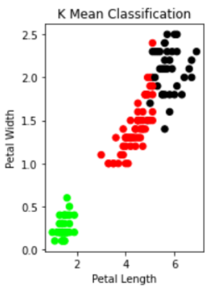
In [5]:
```
# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

Out[5]:
```
Text(0, 0.5, 'Petal Width')
```



In [6]:
```
# 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_))
```

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
```

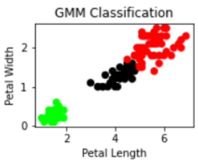


```
In [7]: from sklearn import preprocessing
        scaler = preprocessing.StandardScaler()
        scaler.fit(X)
        xsa = scaler.transform(X)
        xs = pd.DataFrame(xsa, columns = X.columns)
        #xs.sample(5)
```

```
In [8]: from sklearn.mixture import GaussianMixture
        gmm = GaussianMixture(n_components=3)
        gmm.fit(xs)

        y_gmm = gmm.predict(xs)
        #y_cluster_gmm
```

```
In [9]: 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')
```

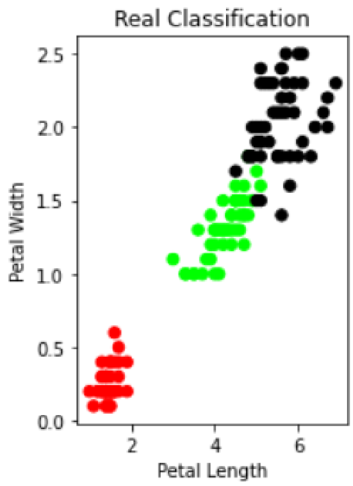
```
Out[9]: Text(0, 0.5, 'Petal Width')
```



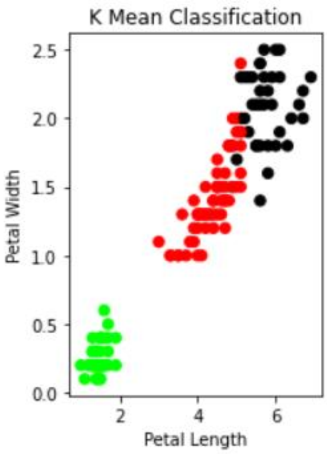

```
In [10]: print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
         print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))

         The accuracy score of EM:  0.0
         The Confusion matrix of EM:  [[ 0 50  0]
          [ 5  0 45]
          [50  0  0]]
```

OUTPUT:

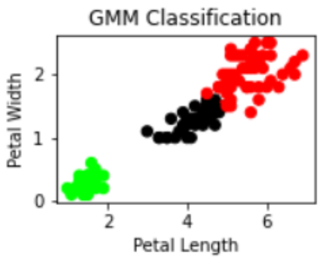
```
Out[5]: Text(0, 0.5, 'Petal Width')
```



```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
```



Out[9]: `Text(0, 0.5, 'Petal Width')`



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

CODE:

jupyter k-Nearest Neighbour algorithm Last Checkpoint: 10 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

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

In [2]:
```python
iris=datasets.load_iris()

x = iris.data
y = iris.target
```

In [3]:
```python
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)
```

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

In [4]:
```python
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
```

Out[4]:
```
KNeighborsClassifier()
```

In [5]:
```python
#To make predictions on our test data
y_pred=classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
Confusion Matrix
[[12  0  0]
 [ 0 17  0]
 [ 0  2 14]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.89      1.00      0.94        17
           2       1.00      0.88      0.93        16

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

OUTPUTS:

sepal-length sepal-width petal-length petal-width

[[5.1 3.5 1.4 0.2]

 [4.9 3. 1.4 0.2]

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

CODE:

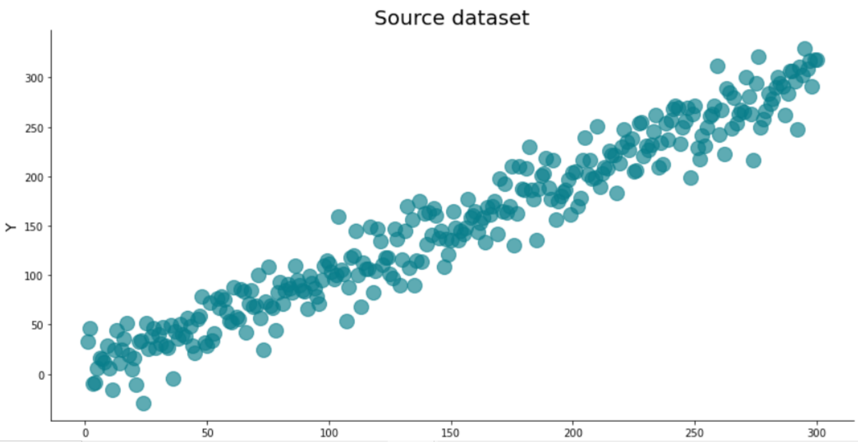
jupyter Linear Regression Last Checkpoint: 10 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['figure.figsize'] = (14, 7)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
```

In [2]:
```
class SimpleLinearRegression:
    '''
    A class which implements simple linear regression model.
    '''
    def __init__(self):
        self.b0 = None
        self.b1 = None

    def fit(self, X, y):
        '''
        Used to calculate slope and intercept coefficients.

        :param X: array, single feature
        :param y: array, true values
        :return: None
        '''
        numerator = np.sum((X - np.mean(X)) * (y - np.mean(y)))
        denominator = np.sum((X - np.mean(X)) ** 2)
        self.b1 = numerator / denominator
        self.b0 = np.mean(y) - self.b1 * np.mean(X)

    def predict(self, X):
        '''
        Makes predictions using the simple line equation.

        :param X: array, single feature
        :return: None
        '''
        if not self.b0 or not self.b1:
            raise Exception('Please call `SimpleLinearRegression.fit(X, y)` before making predictions.')
        return self.b0 + self.b1 * X
```

In [3]:
```
X = np.arange(start=1, stop=301)
y = np.random.normal(loc=X, scale=20)

plt.scatter(X, y, s=200, c='#087E8B', alpha=0.65)
plt.title('Source dataset', size=20)
plt.xlabel('X', size=14)
plt.ylabel('Y', size=14)
plt.show()
```



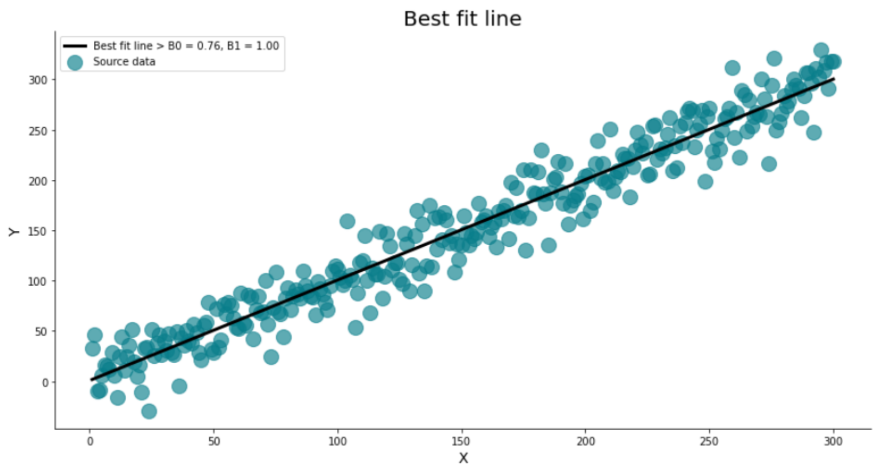
In [10]:
```python
y_test
```

Out[10]:
```
array([216.4830782 , 254.18392422, 135.2171287 ,   6.00611634,
       262.44023607, 254.08554128, 186.45136228, 119.57085249,
        16.24482097, 129.94351096, 253.77245264,  62.49205518,
       213.75211844,  55.76656866, 185.91462572, 235.13207428,
       306.47594529, 202.96628792, 121.29116532, 161.25110103,
        82.38078055, 113.15535798, 271.54726163, 228.59208578,
       105.43797403,  47.88866653, 278.86487912, 308.61599095,
       158.13683775, 237.16395612,  19.25675256, 168.52272352,
        27.0814794 ,  51.41943228, 221.83253837, 147.13198665,
        12.12192211,  65.76828439,  58.52979355,  73.08909188,
        87.18595842,  66.89190461, 261.92450218,  87.95566753,
        44.75632177,  85.34479371, 209.53581733, 231.0094898 ,
        99.86921444, 231.73827302, 208.22233627, 137.36009292,
       257.17245375,  69.50374256, 290.95119189, 266.48344161,
       109.10139697,  91.94878561, 156.50039422,  25.29295448])
```

In [11]:
```python
from sklearn.metrics import mean_squared_error

rmse = lambda y, y_pred: np.sqrt(mean_squared_error(y, y_pred))
rmse(y_test, preds)
```

Out[11]:
```
16.61571567651961
```

In [12]:
```python
model_all = SimpleLinearRegression()
model_all.fit(X, y)
preds_all = model_all.predict(X)

plt.scatter(X, y, s=200, c='#087E8B', alpha=0.65, label='Source data')
plt.plot(X, preds_all, color='#000000', lw=3, label=f'Best fit line > B0 = {model_all.b0:.2f}, B1 = {model_all.b1:.2f}')
plt.title('Best fit line', size=20)
plt.xlabel('X', size=14)
plt.ylabel('Y', size=14)
plt.legend()
plt.show()
```



In [5]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

In [6]:
```python
model = SimpleLinearRegression()
model.fit(X_train, y_train)
preds = model.predict(X_test)
```

In [7]:
```python
model.b0, model.b1
```

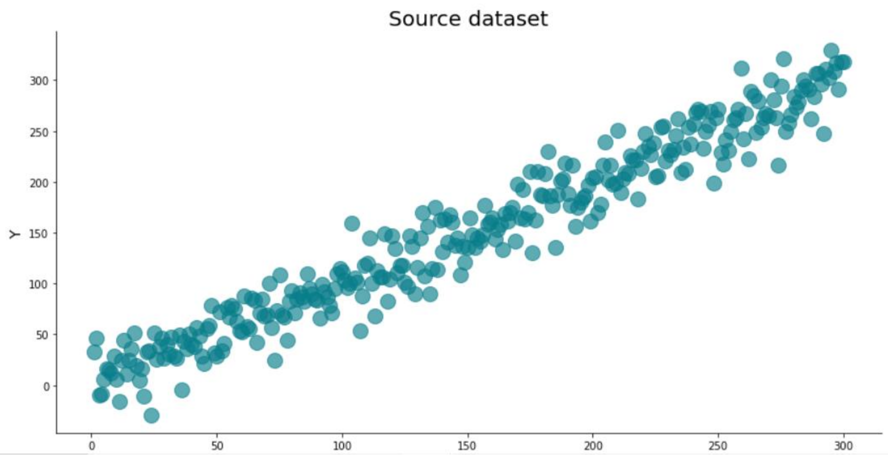
Out[7]:
```
(0.3453837247556635, 1.0022012857270859)
```

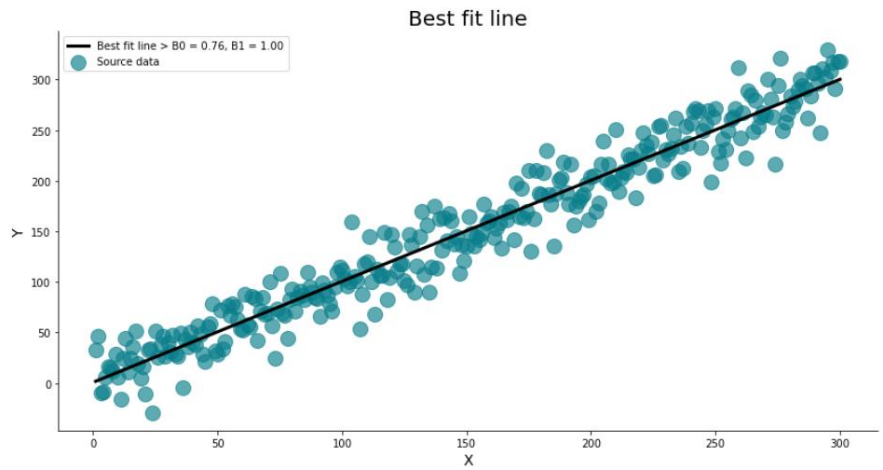
In [9]:
```python
preds
```

Out[9]:
```
array([204.79444601, 267.93312701, 153.68218044,  10.36739658,
       234.86048458, 227.84507558, 197.77903701, 110.58752515,
         6.35859144, 176.73281001, 238.86928973,  58.4730583 ,
       219.8274653 ,  46.44664287, 183.74821901, 222.83406916,
       290.98375659, 212.8120563 , 149.6733753 , 166.71079716,
        79.5192853 , 114.5963303 , 250.89570516, 251.89790644,
       105.57651873,  43.44003901, 282.9661463 , 296.9969643 ,
       158.69318687, 239.87149101,  18.38500687, 165.70859587,
        34.42022744,  25.40041587, 216.82086144, 120.60953801,
         8.36299401,  91.54570073,  47.44884415,  74.50827887,
        94.55230458,  77.51488273, 287.97715273,  61.47966215,
        78.51708401,  64.48626601, 235.86268587, 230.85167944,
       112.59192773, 232.85608201, 181.74381644, 145.66457016,
       240.8736923 ,  76.51268144, 299.00136687, 279.95954244,
        98.56110973,  93.5501033 , 193.77023187,  26.40261715])
```

```
In [13]: from sklearn.linear_model import LinearRegression

         sk_model = LinearRegression()
         sk_model.fit(np.array(X_train).reshape(-1, 1), y_train)
         sk_preds = sk_model.predict(np.array(X_test).reshape(-1, 1))

         sk_model.intercept_, sk_model.coef_

Out[13]: (0.3453837247556635, array([1.00220129]))

In [14]: rmse(y_test, sk_preds)

Out[14]: 16.61571567651961

In [ ]:
```

OUTPUTS:





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

CODE:

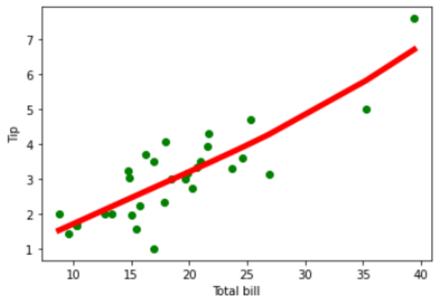
In [1]:
```
from numpy import *
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from numpy.linalg import *
```

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

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

In [4]:
```

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

In [6]:
```
data = pd.read_csv("tips.csv")
bill = array(data.total_bill)
tip = array(data.tip)
mbill = mat(bill)
mtip = mat(tip)
m = shape(mbill)[1]
one = mat(ones(m))
X = hstack((one.T, mbill.T))
# set k here
ypred = localWeightRegression(X, mtip, 10)
SortIndex = X[:, 1].argsort(0)
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()
```



DATA SET:

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	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	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3	Male	No	Sun	Dinner	4
14.83	3.02	Female	No	Sun	Dinner	2
21.58	3.92	Male	No	Sun	Dinner	2
10.33	1.67	Female	No	Sun	Dinner	3
16.29	3.71	Male	No	Sun	Dinner	3
16.97	3.5	Female	No	Sun	Dinner	3
20.65	3.35	Male	No	Sat	Dinner	3
17.92	4.08	Male	No	Sat	Dinner	2
20.29	2.75	Female	No	Sat	Dinner	2
15.77	2.23	Female	No	Sat	Dinner	2
39.42	7.58	Male	No	Sat	Dinner	4
19.82	3.18	Male	No	Sat	Dinner	2
17.81	2.34	Male	No	Sat	Dinner	4
13.37	2	Male	No	Sat	Dinner	2
12.69	2	Male	No	Sat	Dinner	2
21.7	4.3	Male	No	Sat	Dinner	2
19.65	3	Female	No	Sat	Dinner	2
9.55	1.45	Male	No	Sat	Dinner	2

Output: