VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning

Submitted by

Arpit Suman (1BM19CS026)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

Apr-2022 to Aug-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled “**Machine Learning**” carried out by **Arpit Suman (1BM19CS026),** who is a bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Prof. Saritha A.N.	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

Index Sheet

Sl. No.	Experiment Title	Page No.
1.	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.	4
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.	5
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	6
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	9
5.	Write a program to construct a Bayesian network considering training data. Use this model to make predictions.	11
6.	Apply k-Means algorithm to cluster a set of data stored in a .CSV file.	13
7.	Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.	18
8.	Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.	19
9.	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.	20
10.	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data	22

Course Outcomes:

CO1	Ability to **apply** the different learning algorithms.
CO2	Ability to **analyze** the learning techniques for given dataset.
CO3	Ability to **design** a model using machine learning to solve a problem.
CO4	Ability to **conduct** practical experiments to solve problems using appropriate machine learning techniques.

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

```
import pandas as pd
import numpy as np

#to read the data in the csv file
data = pd.read_csv("data.csv")
print(data,"n")

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

print("n The final hypothesis is:",train(d,target))
```

Output

```
    sky air temp humidity    wind water forecast enjoy sport
0  sunny     warm   normal  strong  warm     same         yes
1  sunny     warm     high  strong  warm     same         yes
2  rainy     cold     high  strong  warm   change          no
3  sunny     warm     high  strong  cool   change         yes n
n The attributes are:  [['sunny' 'warm' 'normal' 'strong' 'warm'
'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
n The target is:  ['yes' 'yes' 'no' 'yes']
n The final hypothesis is: ['sunny' 'warm' '?' 'strong' '?' '?']
```

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.

```
import numpy as np
import pandas as pd
data = pd.read_csv('data.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values: ",target)
def learn(concepts, target):
    sh = concepts[0].copy()
    print("\nInitialization of specific and genearal hypothesis")
    print("\nSpecific boundary: ", sh)
    gh = [["?" for i in range(len(sh))] for i in range(len(sh))]
    print("\nGeneric boundary: ",gh)
    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is positive ")
            for x in range(len(sh)):
                if h[x]!= sh[x]:
                    sh[x] ='?'
                    gh[x][x] ='?'
        if target[i] == "no":
            print("Instance is negative ")
            for x in range(len(sh)):
                if h[x]!= sh[x]:
                    gh[x][x] = sh[x]
                else:
                    gh[x][x] = '?'
        print("Specific boundary after ", i+1, "instance is ", sh)
        print("Generic boundary after ", i+1, "instance is ", gh)
        print("\n")
    indices = [i for i, val in enumerate(gh) if val == ['?', '?', '?', '?',
'?', '?']]
    for i in indices:
        gh.remove(['?', '?', '?', '?', '?', '?'])
    return sh, gh
sf, gf = learn(concepts, target)
```

```
print("Final specific hypothesis: ", sf, sep="\n")
print("Final general hypothesis: ", gf, sep="\n")
```

Output

```
Instances:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values:  ['yes' 'yes' 'no' 'yes']

Initialization of specific and genearal hypothesis

Specific boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 1 is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is positive
Specific boundary after  1 instance is  ['sunny' 'warm' 'normal' 'strong'
'warm' 'same']
Generic boundary after  1 instance is  [['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?']]


Instance 2 is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is positive
Specific boundary after  2 instance is  ['sunny' 'warm' '?' 'strong' 'warm'
'same']
Generic boundary after  2 instance is  [['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?']]


Instance 3 is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is negative
Specific boundary after  3 instance is  ['sunny' 'warm' '?' 'strong' 'warm'
'same']
Generic boundary after  3 instance is  [['sunny', '?', '?', '?', '?', '?'],
['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', 'same']]
```

```
Instance 4 is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is positive
Specific boundary after  4 instance is  ['sunny' 'warm' '?' 'strong' '?'
'?']
Generic boundary after  4 instance is  [['sunny', '?', '?', '?', '?', '?'],
['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?']]

Final specific hypothesis:
['sunny' 'warm' '?' 'strong' '?' '?']
Final general hypothesis:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

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

```
import pandas as pd
import math
import numpy as np
import pprint


data=pd.read_csv("tennis.csv")
print("\n Input Data Set is:\n", data)
features = [f for f in data]
features.remove("answer")


class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def find_entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = find_entropy(examples)
```

```
    for u in uniq:
        subdata = examples[examples[attr] == u]
        sub_e = find_entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) *
sub_e
    return gain


def id3(examples, attrs):
  root = Node()

  max_gain = 0
  max_feat = ""
  for feature in attrs:
      gain = info_gain(examples, feature)
      if gain > max_gain:
          max_gain = gain
          max_feat = feature
  root.value = max_feat
  uniq = np.unique(examples[max_feat])
  for u in uniq:
      subdata = examples[examples[max_feat] == u]
      if find_entropy(subdata) == 0.0:
          newNode = Node()
          newNode.isLeaf = True
          newNode.value = u
          newNode.pred = np.unique(subdata["answer"])
          root.children.append(newNode)
      else:
          tempNode = Node()
          tempNode.value = u
          new_attrs = attrs.copy()
          new_attrs.remove(max_feat)
          child = id3(subdata, new_attrs)
          tempNode.children.append(child)
          root.children.append(tempNode)
  return root


def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
```

```
        print(" : ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

root = id3(data, features)
print("Final decision tree:\n")
printTree(root)
```

Output

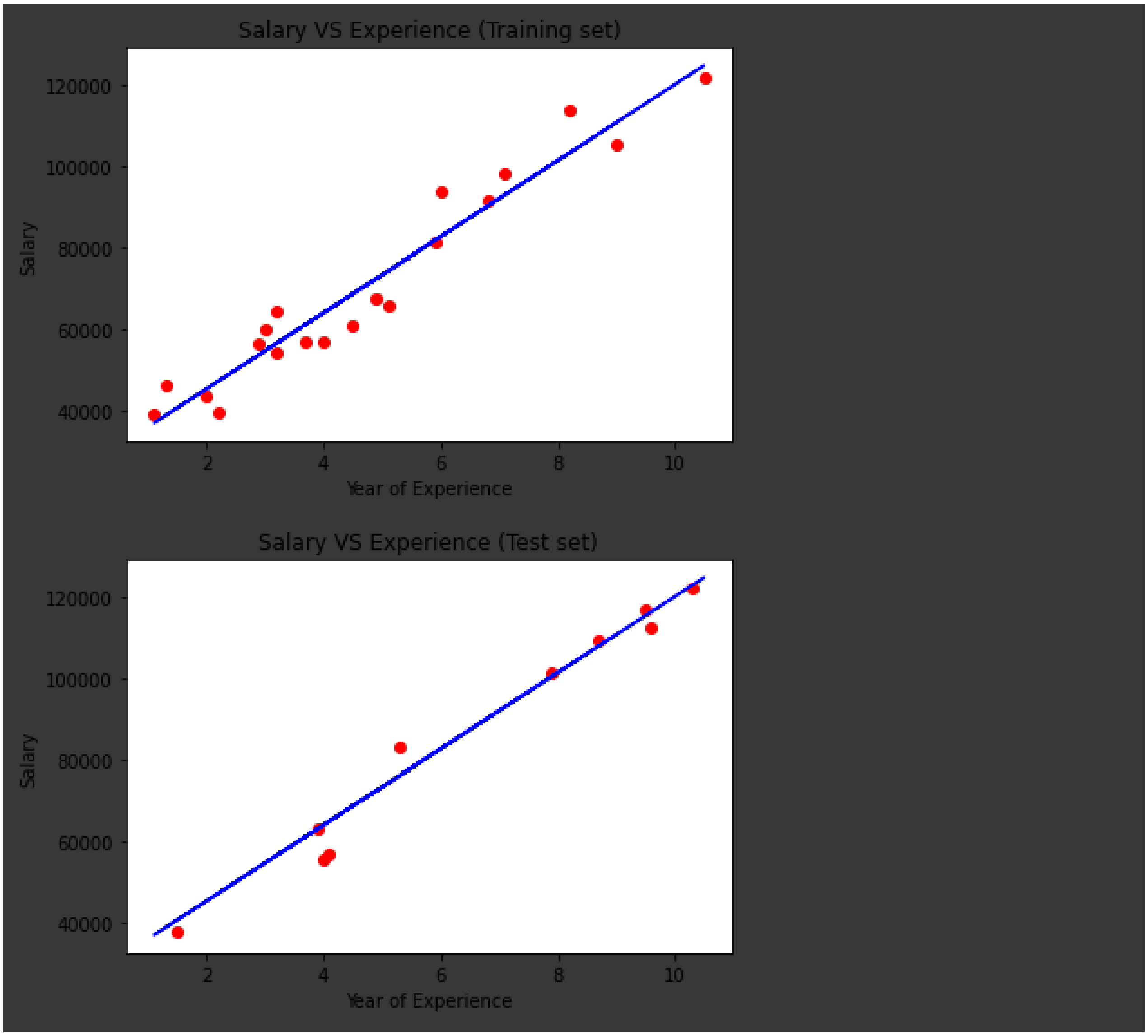
```
 Input Data Set is:
      outlook temperature humidity    wind answer
0       sunny         hot     high    weak     no
1       sunny         hot     high  strong     no
2    overcast         hot     high    weak    yes
3        rain        mild     high    weak    yes
4        rain        cool   normal    weak    yes
5        rain        cool   normal  strong     no
6    overcast        cool   normal  strong    yes
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
Final decision tree:

outlook
     overcast :  ['yes']

     rain
           wind
                 strong :  ['no']

                 weak :  ['yes']

     sunny
           humidity
                 high :  ['no']

                 normal :  ['yes']
```

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.

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

# Importing the dataset
#dataset = pd.read_csv('181105_missing-data.csv')
dataset = pd.read_csv('salary.csv')
X = dataset.iloc[:, :-1].values #get a copy of dataset exclude last column
y = dataset.iloc[:, 1].values #get array of dataset in column 1st

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3,
random_state=0)
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)

# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()

# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
```

```
viz_test.show()
```

Output



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

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

  return dataset

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

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

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

def stdev(numbers):
```

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

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

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

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

def calculateclassprobabilities(summaries, inputvector):
  probabilities = {} # probabilities contains the all prob of all class of
test data
  for classvalue, classsummaries in summaries.items():#class and attribute
information as mean and sd
    probabilities[classvalue] = 1
    for i in range(len(classsummaries)):
      mean, stdev = classsummaries[i] #take mean and sd of every attribute
for class 0 and 1 seperaely
      x = inputvector[i] #testvector's first attribute
      probabilities[classvalue] *= calculateprobability(x, mean,
stdev);#use normal dist
  return probabilities
def predict(summaries, inputvector): #training and test data is passed
  probabilities = calculateclassprobabilities(summaries, inputvector)
  bestLabel, bestProb = None, -1
  for classvalue, probability in probabilities.items():#assigns that class
which has he highest prob
    if bestLabel is None or probability > bestProb:
```

```
      bestProb = probability
      bestLabel = classvalue
  return bestLabel

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

def getaccuracy(testset, predictions):
  correct = 0
  for i in range(len(testset)):
    if testset[i][-1] == predictions[i]:
      correct += 1
  return (correct/float(len(testset))) * 100.0
def main():
  filename = 'naivedata.csv'
  splitratio = 0.67
  dataset = loadcsv(filename);

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

main()
```

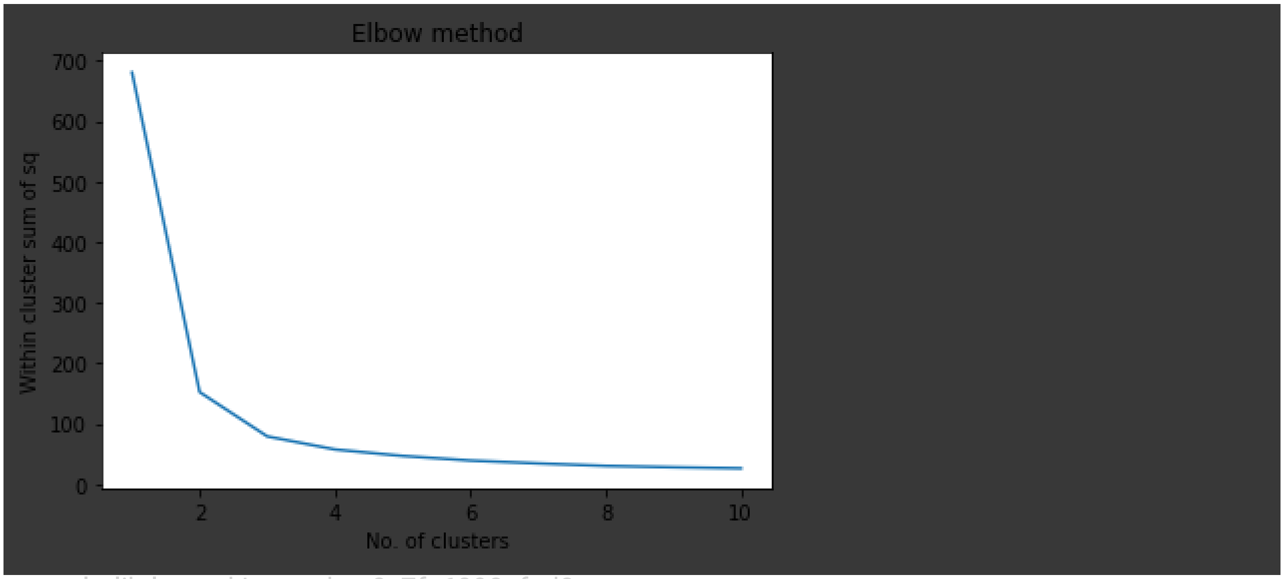
Output

```
Split 767 rows into train=513 and test=254 rows
Accuracy of the classifier is : 74.80314960629921%
```

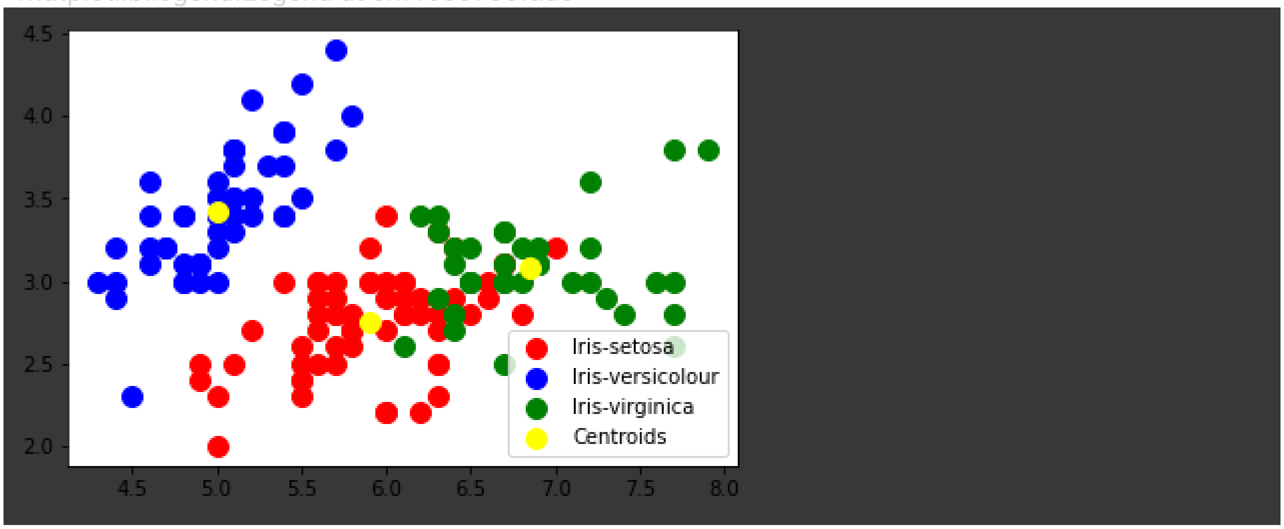
6. Apply k-Means algorithm to cluster a set of data stored in a .CSV file

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('iris.csv')
x = dataset.iloc[:, [1, 2, 3, 4]].values
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300,
n_init = 10, random_state = 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow method')
plt.xlabel('No. of clusters')
plt.ylabel('Within cluster sum of sq')
plt.show()
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init =
10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'red',
label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'blue',
label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green',
label = 'Iris-virginica')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s =
100, c = 'yellow', label = 'Centroids')
plt.legend()
```

Output



<matplotlib.legend.Legend at 0x7fc6098efad0>



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

```
!pip install pgmpy
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
cancer_model = BayesianModel([('Pollution', 'Cancer'),
                              ('Smoker', 'Cancer'),
                              ('Cancer', 'Xray'),
                              ('Cancer', 'Dyspnoea')])
print('Bayesian network nodes:')
print('\t', cancer_model.nodes())
print('Bayesian network edges:')
print('\t', cancer_model.edges())
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])
cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray,
cpd_dysp)
print('Model generated bt adding conditional probability
distribution(cpds)')
print('Checking for Correctness of model:', end='')
print(cancer_model.check_model())
'''print('All local dependencies are as follows')
cancer_model.get_independencies()
'''
```

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

Output

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pgmpy
  Downloading pgmpy-0.1.19-py3-none-any.whl (1.9 MB)
     |████████████████████████████████| 1.9 MB 5.0 MB/s
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.1.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.3.5)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.7/dist-packages (from pgmpy) (3.0.9)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.21.6)
Requirement already satisfied: torch in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.11.0+cu113)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from pgmpy) (4.64.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.7/dist-packages (from pgmpy) (2.6.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from pgmpy) (1.4.1)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages

```
(from pgmpy) (0.10.2)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas->pgmpy) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages
(from pandas->pgmpy) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages
(from python-dateutil>=2.7.3->pandas->pgmpy) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from scikit-learn->pgmpy) (3.1.0)
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages
(from statsmodels->pgmpy) (0.5.2)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from torch->pgmpy) (4.1.1)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.19
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at
pandas.testing instead.
  import pandas.util.testing as tm
Bayesian network nodes:
        ['Pollution', 'Cancer', 'Smoker', 'Xray', 'Dyspnoea']
Bayesian network edges:
        [('Pollution', 'Cancer'), ('Cancer', 'Xray'), ('Cancer', 'Dyspnoea'), ('Smoker', 'Cancer')]
Model generated bt adding conditional probability distribution(cpds)
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

Inferencing with Bayesian Network

Probability of Cancer given Smoker
/usr/local/lib/python3.7/dist-packages/pgmpy/models/BayesianModel.py:10:
FutureWarning: BayesianModel has been renamed to BayesianNetwork. Please use
BayesianNetwork class, BayesianModel will be removed in future.
  FutureWarning,

```
Finding Elimination Order: : 100%
1/1 [00:00<00:00, 8.95it/s]
Eliminating: Pollution: 100%
1/1 [00:00<00:00, 14.21it/s]
```

Cancer	phi(Cancer)
Cancer(0)	0.0029
Cancer(1)	0.9971

Probability of Cancer given Smoker, Pollution

```
Finding Elimination Order: :
0/0 [00:00<?, ?it/s]
0/0 [00:00<?, ?it/s]
```

Cancer	phi(Cancer)
Cancer(0)	0.0200
Cancer(1)	0.9800

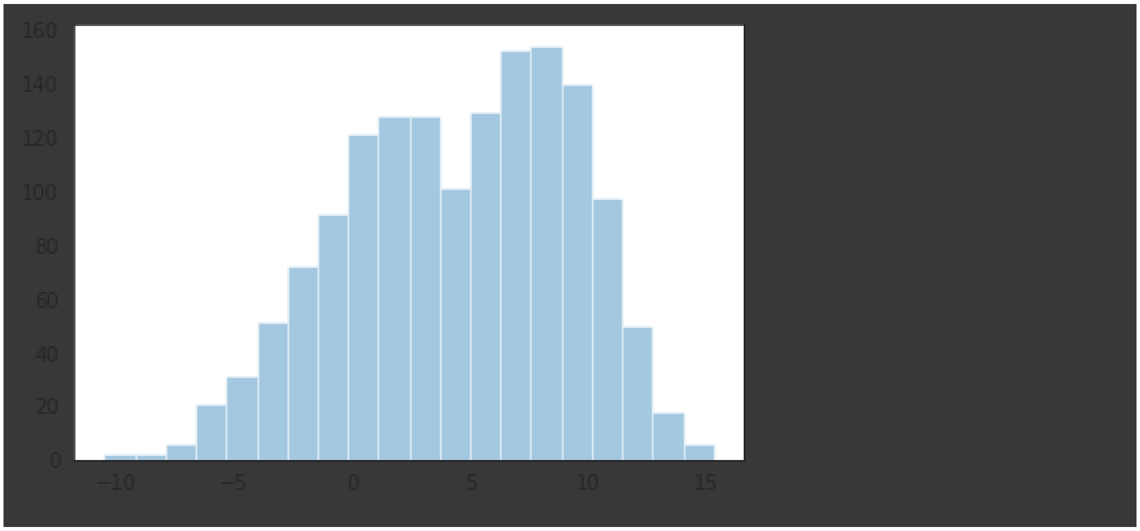
8. Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")
%matplotlib inline
#for matrix math
import numpy as np
#for normalization + probability density function computation
from scipy import stats
#for data preprocessing
import pandas as pd
from math import sqrt, log, exp, pi
from random import uniform
random_seed=36788765
np.random.seed(random_seed)

Mean1 = 2.0  # Input parameter, mean of first normal probability
distribution
Standard_dev1 = 4.0 #@param {type:"number"}
Mean2 = 9.0 # Input parameter, mean of second normal  probability
distribution
Standard_dev2 = 2.0 #@param {type:"number"}

# generate data
y1 = np.random.normal(Mean1, Standard_dev1, 1000)
y2 = np.random.normal(Mean2, Standard_dev2, 500)
data=np.append(y1,y2)

# For data visiualisation calculate left and right of the graph
Min_graph = min(data)
Max_graph = max(data)
x = np.linspace(Min_graph, Max_graph, 2000) # to plot the data
```

```
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1,
Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2,
Standard_dev2))
sns.distplot(data, bins=20, kde=False);

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components = 2, tol=0.000001)
gmm.fit(np.expand_dims(data, 1)) # Parameters: array-like, shape
(n_samples, n_features), 1 dimension dataset so 1 feature
Gaussian_nr = 1
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1,
Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2,
Standard_dev2))
for mu, sd, p in zip(gmm.means_.flatten(),
np.sqrt(gmm.covariances_.flatten()), gmm.weights_):
    print('Gaussian {:}: μ = {:.2}, σ = {:.2}, weight =
{:.2}'.format(Gaussian_nr, mu, sd, p))
    g_s = stats.norm(mu, sd).pdf(x) * p
    plt.plot(x, g_s, label='gaussian sklearn');
    Gaussian_nr += 1
sns.distplot(data, bins=20, kde=False, norm_hist=True)
gmm_sum = np.exp([gmm.score_samples(e.reshape(-1, 1)) for e in x])
#gmm gives log probability, hence the exp() function
plt.plot(x, gmm_sum, label='gaussian mixture');
plt.legend();
```

Output

Input Gaussian 1: μ = 2.0, σ = 4.0
Input Gaussian 2: μ = 9.0, σ = 2.0
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
`distplot` is a deprecated function and will be removed in a future version. Please adapt
your code to use either `displot` (a figure-level function with similar flexibility) or `histplot`
(an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)



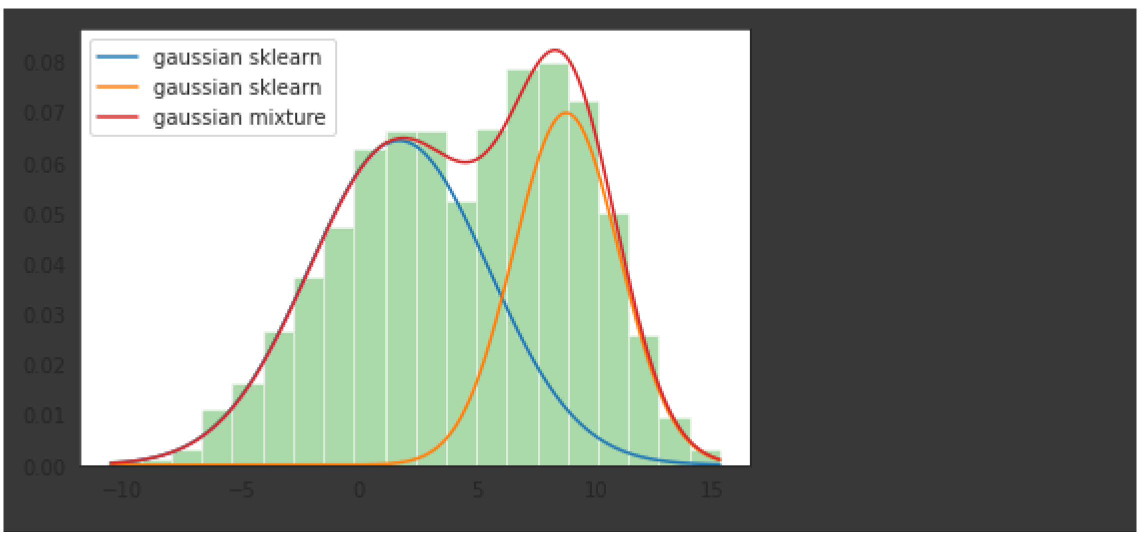
Input Gaussian 1: μ = 2.0, σ = 4.0

Input Gaussian 2: μ = 9.0, σ = 2.0

Gaussian 1: μ = 1.7, σ = 3.8, weight = 0.61

Gaussian 2: μ = 8.8, σ = 2.2, weight = 0.39

/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)



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

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

iris=datasets.load_iris()

x = iris.data
y = iris.target

# print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
# print(x)
# print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
# print(y)

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)

#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))
```

Output

```
Confusion Matrix
[[14  0  0]
 [ 0 10  1]
 [ 0  2 18]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       0.83      0.91      0.87        11
           2       0.95      0.90      0.92        20

    accuracy                           0.93        45
   macro avg       0.93      0.94      0.93        45
weighted avg       0.94      0.93      0.93        45
```

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

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr
def kernel(point,xmat, k):
 m,n = np1.shape(xmat)
 weights = np1.mat(np1.eye((m)))
 for j in range(m):
    diff = point - X[j]
    weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
 return weights
def localWeight(point,xmat,ymat,k):
  wei = kernel(point,xmat,k)
  W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
  return W
def localWeightRegression(xmat,ymat,k):
 m,n = np1.shape(xmat)
 ypred = np1.zeros(m)
 for i in range(m):
    ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
 return ypred
 # load data points
data = pd.read_csv('/content/tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2
dimensional array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
```

```
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='red')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'black', linewidth=1)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
```

Output