NAVEENA K N

USN: 1BM20CS411

MACHINE LEARNING

Lab Programs

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

```
import csv

def updatehypothesis(x,h):

if h == []:

for i in range(0,len(x)):

h.append("$")

print("Initial State : ", h)

return x

for i in range(0,len(x)):

if x[i].upper() != h[i].upper() :

h[i] = "?"

print("Most Specific Hypothesis : ", h)

return h

if __name__ == "__main__":

data = []

h = []

with open('datasheet1.csv','r') as file :

reader = csv.reader(file)

print("Data Set : ")

for row in reader:

data.append(row)

print(row)

if data:

for x in data:

if x[-1].upper() == "YES":

x.pop()
```

```
    h = updatehypothesis(x,h)
print("Maximally Specific Hypothesis : " , h)
```

Output:

```
['sky', 'airtemp', 'humidity', 'wind', 'water', 'forecast', 'enjoysport']
['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']
Initial State :  ['$', '$', '$', '$', '$', '$']
Most Specific Hypothesis :  ['sunny', 'warm', '?', 'strong', 'warm', 'same']
Most Specific Hypothesis :  ['sunny', 'warm', '?', 'strong', '?', '?']
Maximally Specific Hypothesis :  ['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 csv
a=[]
with open('datasheet1.csv','r') as csvfile:
for row in csv.reader(csvfile):
a.append(row)
print(a)
print("")
num_attributes = len(a[0])-1
s=['0']*num_attributes
g=[["?" for i in range(len(s))] for i in range(len(s))]
for i in range(0,len(a)):
if a[i][num_attributes]=='yes':
for j in range(0,num_attributes):
if s[j]=='0' or s[j]==a[i][j]:
s[j]=a[i][j]
else:
s[j]='?'
else:
for j in range(0,num_attributes):
if(s[j] == a[i][j] or s[j] =='?'):
g[j][j]='?'
continue
else:
g[j][j] = s[j]
for j in range(0,num_attributes):
if s[j]!=g[j][j] or s[j]=='?':
g[j][j]='?'
```

indices = [i for i, val in enumerate(g) if val == ['?', '?', '?', '?', '?', '?']]

for i in indices:

g.remove(['?', '?', '?', '?', '?', '?'])

print("Specific hypothesis:",s)

print("General hypothesis:",g)

Output:

```
[['sky', 'airtemp', 'humidity', 'wind', 'water', 'forecast', 'enjoysport'],
['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']]

Specific hypothesis: ['sunny', 'warm', '?', 'strong', '?', '?']
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 math
import csv
def load_csv(id3):
lines=csv.reader(open('../input/mlid3/id3.csv',"r"));
dataset = list(lines)
headers = dataset.pop(0)
return dataset,headers
class Node:
def __init__(self,attribute):
self.attribute=attribute
self.children=[]
self.answer=""
def subtables(data,col,delete):
dic={}
coldata=[row[col] for row in data]
attr=list(set(coldata))
counts=[0]*len(attr)
r=len(data)
c=len(data[0])
for x in range(len(attr)):
for y in range(r):
if data[y][col]==attr[x]:
counts[x]+=1
for x in range(len(attr)):
dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
pos=0
for y in range(r):
if data[y][col]==attr[x]:
```

```
if delete:
del data[y][col]
dic[attr[x]][pos]=data[y]
pos+=1
return attr,dic
def entropy(S):
attr=list(set(S))
if len(attr)==1:
return 0
counts=[0,0]
for i in range(2):
counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)
sums=0
for cnt in counts:
sums+=-1*cnt*math.log(cnt,2)
return sums
def compute_gain(data,col):
attr,dic = subtables(data,col,delete=False)
total_size=len(data)
entropies=[0]*len(attr)
ratio=[0]*len(attr)

total_entropy=entropy([row[-1] for row in data])
for x in range(len(attr)):
ratio[x]=len(dic[attr[x]])/(total_size*1.0)
entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
total_entropy-=ratio[x]*entropies[x]
return total_entropy
def build_tree(data,features):
lastcol=[row[-1] for row in data]
```

```
if(len(set(lastcol)))==1:
node=Node("")
node.answer=lastcol[0]
return node
n=len(data[0])-1
gains=[0]*n
for col in range(n):
gains[col]=compute_gain(data,col)
split=gains.index(max(gains))
node=Node(features[split])
fea = features[:split]+features[split+1:]
attr,dic=subtables(data,split,delete=True)
for x in range(len(attr)):
child=build_tree(dic[attr[x]],fea)
node.children.append((attr[x],child))
return node
def print_tree(node,level):
if node.answer!="":
print("  "*level,node.answer)
return
print("  "*level,node.attribute)
for value,n in node.children:
print("  "*(level+1),value)
print_tree(n,level+2)
def classify(node,x_test,features):
if node.answer!="":
print(node.answer)
return
pos=features.index(node.attribute)
for value, n in node.children:
```

```
 if x_test[pos]==value:
classify(n,x_test,features)
dataset,features=load_csv("../input/mlid3/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("../input/playtennis/ID3.csv")
# print(features,"\n\n",testdata)
# for xtest in testdata:
#     print("The test instance:",xtest)
#     print("The label for test instance:",end="   ")
#     classify(node1,xtest,features)
```

Dataset:

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

Output:

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   sunny
     Humidity
       normal
         yes
       high
         no
   overcast
     yes
   rain
     Wind
       strong
         no
       weak
         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 pandas as pd
data = pd.read_csv('../input/playtennisnb/PlayTennis.csv')
data.head()
y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values
print(f'Target Values: {y}')
print(f'Features: \n{X}')
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)}")
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
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)
```

Dataset:

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

Output:

```
Enter some code at the bottom of this console and press [Enter].
  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
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']]
Number of instances in training set: 8
Number of instances in testing set: 6
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
```

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

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv('../input/salarydata/salaryData.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 1].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)

# 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='green')

viz_train.plot(X_train, regressor.predict(X_train), color='black')

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='green')

viz_test.plot(X_train, regressor.predict(X_train), color='black')

viz_test.title('Salary VS Experience (Test set)')

viz_test.xlabel('Year of Experience')

viz_test.ylabel('Salary')

viz_test.show()
```

Dataset:

1	YearsExperience	Salary
2	1.1	39343
3	1.3	46205
4	1.5	37731
5	2.0	43525
6	2.2	39891
7	2.9	56642
8	3.0	60150
9	3.2	54445
10	3.2	64445
11	3.7	57189
12	3.9	63218
13	4.0	55794
14	4.0	56957
15	4.1	57081
16	4.5	61111
17	4.9	67938
18	5.1	66029
19	5.3	83088
20	5.9	81363
21	6.0	93940
22	6.8	91738
23	7.1	98273
24	7.9	101302
25	8.2	113812
26	8.7	109431
27	9.0	105582
28	9.5	116969
29	9.6	112635
30	10.3	122391
31	10.5	121872

Output:


Salary VS Experience (Training set)
Salary
120000
100000
80000
60000
40000
2
4
6
8
10
Year of Experience

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

```
!pip install pgmpy

# Starting with defining the network structure

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination


#Define a Structure with nodes and edges

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])
```

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

# Checking if the cpds are valid for the model.
print('Checking for Correctness of model:', end='')
print(cancer_model.check_model())
'''print('All local dependencies 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'))
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)

{
 "cells": [
  {
   "cell_type": "code",
```

"execution_count": 1,

"id": "144c0515",

"metadata": {},

"outputs": [

{

"name": "stdout",

"output_type": "stream",

"text": [

"Requirement already satisfied: pgmpy in c:\\programdata\\anaconda3\\lib\\site-packages (0.1.18)\n",

"Requirement already satisfied: statsmodels in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (0.12.2)\n",

"Requirement already satisfied: pandas in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (1.3.4)\n",

"Requirement already satisfied: numpy in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (1.20.3)\n",

"Requirement already satisfied: tqdm in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (4.62.3)\n",

"Requirement already satisfied: networkx in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (2.6.3)\n",

"Requirement already satisfied: pyparsing in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (3.0.4)\n",

"Requirement already satisfied: scikit-learn in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (0.24.2)\n",

"Requirement already satisfied: joblib in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (1.1.0)\n",

"Requirement already satisfied: torch in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (1.11.0)\n",

"Requirement already satisfied: scipy in c:\\programdata\\anaconda3\\lib\\site-packages (from pgmpy) (1.7.1)\n",

"Requirement already satisfied: python-dateutil>=2.7.3 in c:\\programdata\\anaconda3\\lib\\site-packages (from pandas->pgmpy) (2.8.2)\n",

"Requirement already satisfied: pytz>=2017.3 in c:\\programdata\\anaconda3\\lib\\site-packages (from pandas->pgmpy) (2021.3)\n",

"Requirement already satisfied: six>=1.5 in c:\\programdata\\anaconda3\\lib\\site-packages (from python-dateutil>=2.7.3->pandas->pgmpy) (1.16.0)\n", "Requirement already satisfied: threadpoolctl>=2.0.0 in c:\\programdata\\anaconda3\\lib\\site-packages (from scikit-learn->pgmpy) (2.2.0)\n",

"Requirement already satisfied: patsy>=0.5 in c:\\programdata\\anaconda3\\lib\\site-packages (from statsmodels->pgmpy) (0.5.2)\n",

"Requirement already satisfied: typing-extensions in c:\\programdata\\anaconda3\\lib\\site-packages (from torch->pgmpy) (3.10.0.2)\n",

"Requirement already satisfied: colorama in c:\\programdata\\anaconda3\\lib\\site-packages (from tqdm->pgmpy) (0.4.4)\n"

]

}

],

"source": [

"!pip install pgmpy"

]

},

{

"cell_type": "code",

"execution_count": 2,

"id": "66f84fa3",

"metadata": {},

"outputs": [

{

"name": "stdout",

"output_type": "stream",

"text": [

"Bayesian network nodes:\n",

"\t ['Pollution', 'Cancer', 'Smoker', 'Xray', 'Dyspnoea']\n",

"Bayesian network edges:\n",

"\t [('Pollution', 'Cancer'), ('Cancer', 'Xray'), ('Cancer', 'Dyspnoea'), ('Smoker', 'Cancer')]\n"

]

},

{

"name": "stderr",

```
    "output_type": "stream",
    "text": [
     "C:\\ProgramData\\Anaconda3\\lib\\site-packages\\pgmpy\\models\\BayesianModel.py:8: FutureWarning: BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.\n",
     "  warnings.warn(\n"
    ]
   }
  ],
  "source": [
   "# Starting with defining the network structure\n",
   "from pgmpy.models import BayesianModel\n",
   "from pgmpy.factors.discrete import TabularCPD\n",
   "from pgmpy.inference import VariableElimination\n",
   "\n",
   "#Define a Structure with nodes and edges\n",
   "cancer_model = BayesianModel([('Pollution', 'Cancer'), \n",
   "                              ('Smoker', 'Cancer'),\n",
   "                              ('Cancer', 'Xray'),\n",
   "                              ('Cancer', 'Dyspnoea')])\n",
   "print('Bayesian network nodes:')\n",
   "print('\\t', cancer_model.nodes())\n",
   "print('Bayesian network edges:')\n",
   "print('\\t', cancer_model.edges())"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 3,
  "id": "fa388cca",
  "metadata": {},
  "outputs": [],
  "source": [
```

```
 "cpd_poll = TabularCPD(variable='Pollution', variable_card=2,\n",
 "                    values=[[0.9], [0.1]])\n",
 "cpd_smoke = TabularCPD(variable='Smoker', variable_card=2,\n",
 "                     values=[[0.3], [0.7]])\n",
 "cpd_cancer = TabularCPD(variable='Cancer', variable_card=2,\n",
 "                      values=[[0.03, 0.05, 0.001, 0.02],\n",
 "                              [0.97, 0.95, 0.999, 0.98]],\n",
 "                      evidence=['Smoker', 'Pollution'],\n",
 "                      evidence_card=[2, 2])\n",
 "cpd_xray = TabularCPD(variable='Xray', variable_card=2,\n",
 "                    values=[[0.9, 0.2], [0.1, 0.8]],\n",
 "                    evidence=['Cancer'], evidence_card=[2])\n",
 "cpd_dysp = TabularCPD(variable='Dyspnoea', variable_card=2,\n",
 "                    values=[[0.65, 0.3], [0.35, 0.7]],\n",
 "                    evidence=['Cancer'], evidence_card=[2])"
]
},
{
"cell_type": "code",
"execution_count": 4,
"id": "93abdbbe",
"metadata": {},
"outputs": [
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "Model generated bt adding conditional probability distribution(cpds)\n",
   "Checking for Correctness of model:True\n"
  ]
 }
],
```

```
"source": [
 "# Associating the parameters with the model structure.\n",
 "cancer_model.add_cpds(cpd_poll, cpd_smoke, cpd_cancer, cpd_xray, cpd_dysp)\n",
 "print('Model generated bt adding conditional probability distribution(cpds)')\n",
 "\n",
 "# Checking if the cpds are valid for the model.\n",
 "print('Checking for Correctness of model:', end='')\n",
 "print(cancer_model.check_model())"
]
},
{
"cell_type": "code",
"execution_count": 5,
"id": "b080e129",
"metadata": {},
"outputs": [
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "Displaying CPDs\n",
   "+--------------+-----+\n",
   "| Pollution(0) | 0.9 |\n",
   "+--------------+-----+\n",
   "| Pollution(1) | 0.1 |\n",
   "+--------------+-----+\n",
   "+-----------+-----+\n",
   "| Smoker(0) | 0.3 |\n",
   "+-----------+-----+\n",
   "| Smoker(1) | 0.7 |\n",
   "+-----------+-----+\n",
   "+-----------+--------------+--------------+--------------+--------------+\n",
```

```
    "| Smoker    | Smoker(0)    | Smoker(0)    | Smoker(1)    | Smoker(1)    |\n",
    "+-----------+--------------+--------------+--------------+--------------+\n",
    "| Pollution | Pollution(0) | Pollution(1) | Pollution(0) | Pollution(1) |\n",
    "+-----------+--------------+--------------+--------------+--------------+\n",
    "| Cancer(0) | 0.03         | 0.05         | 0.001        | 0.02         |\n",
    "+-----------+--------------+--------------+--------------+--------------+\n",
    "| Cancer(1) | 0.97         | 0.95         | 0.999        | 0.98         |\n",
    "+-----------+--------------+--------------+--------------+--------------+\n",
    "+---------+-----------+-----------+\n",
    "| Cancer  | Cancer(0) | Cancer(1) |\n",
    "+---------+-----------+-----------+\n",
    "| Xray(0) | 0.9       | 0.2       |\n",
    "+---------+-----------+-----------+\n",
    "| Xray(1) | 0.1       | 0.8       |\n",
    "+---------+-----------+-----------+\n",
    "+-------------+-----------+-----------+\n",
    "| Cancer      | Cancer(0) | Cancer(1) |\n",
    "+-------------+-----------+-----------+\n",
    "| Dyspnoea(0) | 0.65      | 0.3       |\n",
    "+-------------+-----------+-----------+\n",
    "| Dyspnoea(1) | 0.35      | 0.7       |\n",
    "+-------------+-----------+-----------+\n"
   ]
  }
 ],
 "source": [
  """print('All local dependencies are as follows')\n",
  "cancer_model.get_independencies()\n",
  """\n",
  "\n",
  "print('Displaying CPDs')\n",
  "print(cancer_model.get_cpds('Pollution'))\n",
```

```
   "print(cancer_model.get_cpds('Smoker'))\n",
   "print(cancer_model.get_cpds('Cancer'))\n",
   "print(cancer_model.get_cpds('Xray'))\n",
   "print(cancer_model.get_cpds('Dyspnoea'))"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 6,
  "id": "8f31415d",
  "metadata": {},
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "\n",
     "Inferencing with Bayesian Network\n",
     "\n",
     "Probability of Cancer given Smoker\n"
    ]
   },
   {
    "data": {
     "application/vnd.jupyter.widget-view+json": {
      "model_id": "36579805e76f4cb382138d7fc144b2e0",
      "version_major": 2,
      "version_minor": 0
     },
     "text/plain": [
      "  0%|          | 0/1 [00:00<?, ?it/s]"
     ]
```

```
  },
  "metadata": {},
  "output_type": "display_data"
 },
 {
  "data": {
   "application/vnd.jupyter.widget-view+json": {
    "model_id": "127dc5ef57d24fa2a8ec5ff86fd500b4",
    "version_major": 2,
    "version_minor": 0
   },
   "text/plain": [
    "  0%|          | 0/1 [00:00<?, ?it/s]"
   ]
  },
  "metadata": {},
  "output_type": "display_data"
 },
 {
  "name": "stdout",
  "output_type": "stream",
  "text": [
   "+-----------+---------------+\n",
   "| Cancer    |   phi(Cancer) |\n",
   "+===========+===============+\n",
   "| Cancer(0) |        0.0029 |\n",
   "+-----------+---------------+\n",
   "| Cancer(1) |        0.9971 |\n",
   "+-----------+---------------+\n",
   "\n",
   "Probability of Cancer given Smoker, Pollution\n"
  ]
```

},
{
"data": {
"application/vnd.jupyter.widget-view+json": {
"model_id": "cb7a2f59cfc244c0bde73db7f9e8c1c7",
"version_major": 2,
"version_minor": 0
},
"text/plain": [
"0it [00:00, ?it/s]"
]
},
"metadata": {},
"output_type": "display_data"
},
{
"data": {
"application/vnd.jupyter.widget-view+json": {
"model_id": "3482095f6cb441099263051c952a49c7",
"version_major": 2,
"version_minor": 0
},
"text/plain": [
"0it [00:00, ?it/s]"
]
},
"metadata": {},
"output_type": "display_data"
},
{
"name": "stdout",
"output_type": "stream",

```
    "text": [
     "+-----------+---------------+\n",
     "| Cancer    |   phi(Cancer) |\n",
     "+===========+===============+\n",
     "| Cancer(0) |        0.0200 |\n",
     "+-----------+---------------+\n",
     "| Cancer(1) |        0.9800 |\n",
     "+-----------+---------------+\n"
    ]
   }
  ],
  "source": [
   "cancer_infer = VariableElimination(cancer_model)\n",
   "print('\\nInferencing with Bayesian Network')\n",
   "\n",
   "print('\\nProbability of Cancer given Smoker')\n",
   "q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1})\n",
   "print(q)\n",
   "\n",
   "print('\\nProbability of Cancer given Smoker, Pollution')\n",
   "q = cancer_infer.query(variables=['Cancer'], evidence={'Smoker': 1,'Pollution': 1})\n",
   "print(q)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "id": "84bf83e4",
  "metadata": {},
  "outputs": [],
  "source": []
 }
```

```
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.7"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

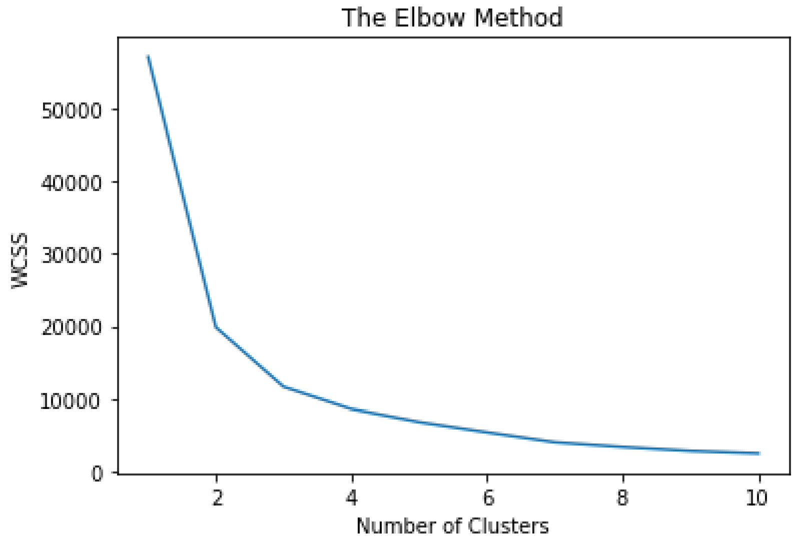
7) Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

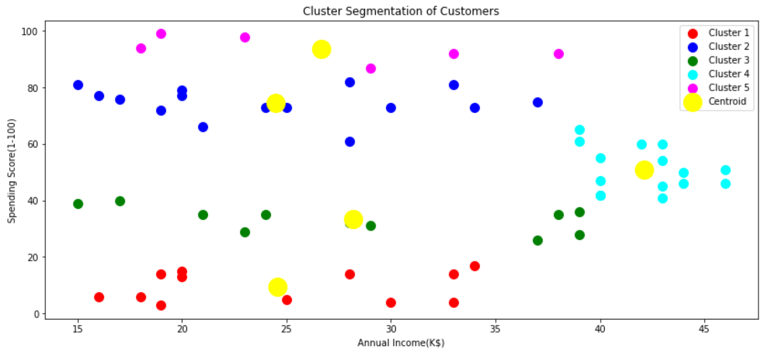
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv('Desktop/Mall_Customers.csv')
df.head()
df.describe()
df.isnull().sum() # to check for missing data
df.shape
sns.countplot(df['Gender'])
sns.distplot(df['Age'])
sns.distplot(df['Annual Income (k$)'])
sns.distplot(df['Spending Score (1-100)'])
# Elbow method to find the optimal number of Clusters
data=df.iloc[:,[3,4]].values
from sklearn.cluster import KMeans
wcss=[] # within cluster sum of square
for i in range(1,11):
    kmeans=KMeans(n_clusters=i, init='k-means++',random_state=0)
    kmeans.fit(data)
    wcss.append(kmeans.inertia_) #inertia_ = to find the wcss value

plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
kmeans=KMeans(n_clusters=5,init='k-means++',random_state=0)
y_kmeans=kmeans.fit_predict(data)
#plotting the the clusters
fig,ax = plt.subplots(figsize=(14,6))
```

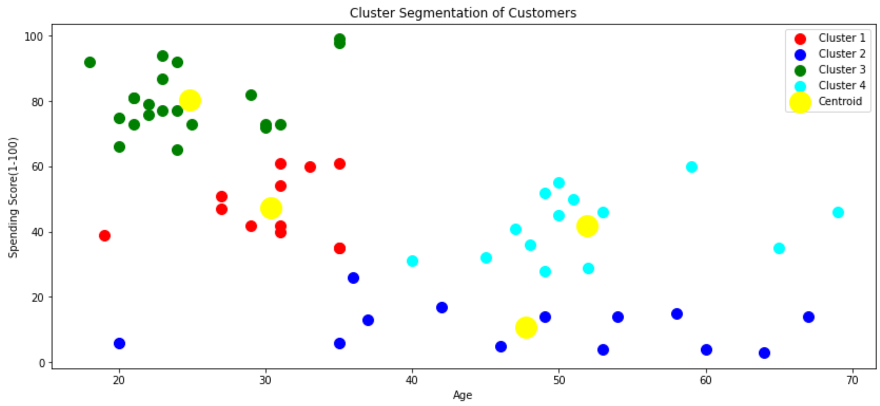
```
ax.scatter(data[y_kmeans==0,0],data[y_kmeans==0,1],s=100,c='red',label='Cluster 1')
ax.scatter(data[y_kmeans==1,0],data[y_kmeans==1,1],s=100,c='blue',label='Cluster 2')
ax.scatter(data[y_kmeans==2,0],data[y_kmeans==2,1],s=100,c='green',label='Cluster 3')
ax.scatter(data[y_kmeans==3,0],data[y_kmeans==3,1],s=100,c='cyan',label='Cluster 4')
ax.scatter(data[y_kmeans==4,0],data[y_kmeans==4,1],s=100,c='magenta',label='Cluster 5')

ax.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=400,c='yellow',label='
Centroid')
plt.title('Cluster Segmentation of Customers')
plt.xlabel('Annual Income(K$)')
plt.ylabel('Spending Score(1-100)')
plt.legend()
plt.show()
data = df.iloc[:,[2,4]].values
from sklearn.cluster import KMeans
wcss=[]  # within cluster sum of square
for i in range(1,11):
    kmeans=KMeans(n_clusters=i, init='k-means++',random_state=0)
    kmeans.fit(data)
    wcss.append(kmeans.inertia_)  # inertia_ = to find the wcss value

plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
kmeans=KMeans(n_clusters=4,init='k-means++',random_state=0)
y_kmeans=kmeans.fit_predict(data)

#Plotting the clusters
fig,ax = plt.subplots(figsize=(14,6))
ax.scatter(data[y_kmeans==0,0],data[y_kmeans==0,1],s=100,c='red',label='Cluster 1')
```

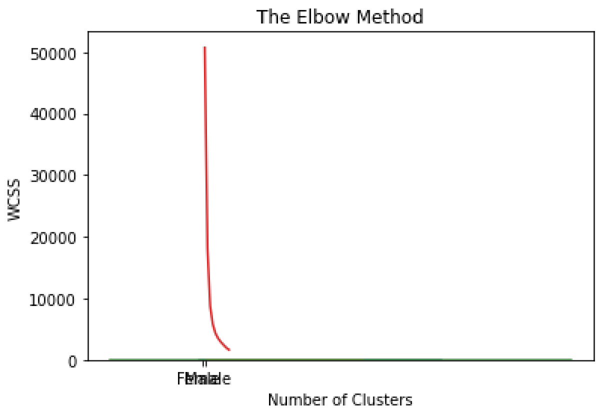
```
ax.scatter(data[y_kmeans==1,0],data[y_kmeans==1,1],s=100,c='blue',label='Cluster 2')
ax.scatter(data[y_kmeans==2,0],data[y_kmeans==2,1],s=100,c='green',label='Cluster 3')
ax.scatter(data[y_kmeans==3,0],data[y_kmeans==3,1],s=100,c='cyan',label='Cluster 4')

ax.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=400,c='yellow',label='
Centroid')
plt.title('Cluster Segmentation of Customers')
plt.xlabel('Age')
plt.ylabel('Spending Score(1-100)')
plt.legend()
plt.show()
```

CustomerI	Gender	Age	Annual Inc	Spending S
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14
12	Female	35	19	99
13	Female	58	20	15
14	Female	24	20	77
15	Male	37	20	13
16	Male	22	20	79
17	Female	35	21	35
18	Male	20	21	66
19	Male	52	23	29
20	Female	35	23	98
21	Male	35	24	35
22	Male	25	24	73
23	Female	46	25	5
24	Male	31	25	73
25	Female	54	28	14
26	Male	29	28	82
27	Female	45	28	32
28	Male	35	28	61
29	Female	40	29	31
30	Female	23	29	87
31	Male	60	30	4
32	Female	21	30	73
33	Male	53	33	4
34	Male	18	33	92
35	Female	49	33	14
36	Female	21	33	81
37	Female	42	34	17
38	Female	30	34	73
39	Female	36	37	26
40	Female	20	37	75
41	Female	65	38	35
42	Male	24	38	92
43	Male	48	39	36
44	Female	31	39	61
45	Female	49	39	28
46	Female	24	39	65
47	Female	50	40	55
48	Female	27	40	47
49	Female	29	40	42
50	Female	31	40	42
51	Female	49	42	52
52	Male	33	42	60
53	Female	31	43	54
54	Male	59	43	60
55	Female	50	43	45
56	Male	47	43	41
57	Female	51	44	50
58	Male	69	44	46
59	Female	27	46	51
60	Male	53	46	46


The Elbow Method
WCSS
0
10000
20000
30000
40000
50000
2
4
6
8
10
Number of Clusters


Cluster Segmentation of Customers
Spending Score(1-100)
Annual Income(K$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroid


Cluster Segmentation of Customers
Spending Score(1-100)
Age
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Centroid


The Elbow Method
WCSS
0
10000
20000
30000
40000
50000
Number of Clusters

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

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

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

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

model = KMeans(n_clusters=3)
model.fit(X)



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

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

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

```
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))


from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

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

OUTPUT:

The accuracy score of K-Mean: 0.24

The Confusion matrixof K-Mean: [[0 50 0]

 [48 0 2]

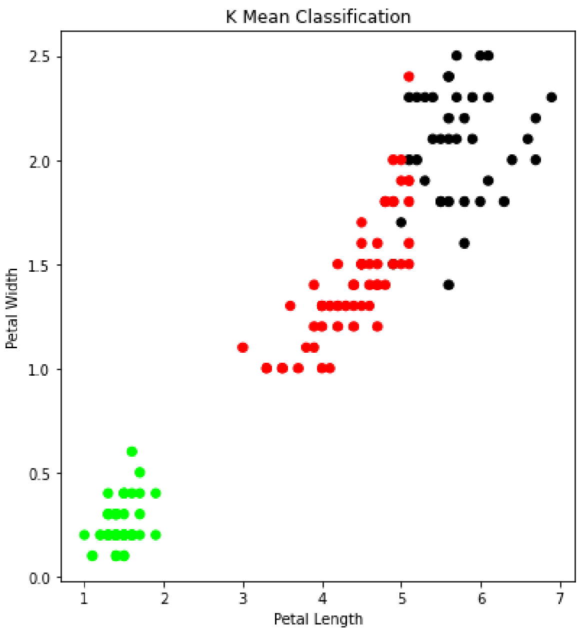
 [14 0 36]]

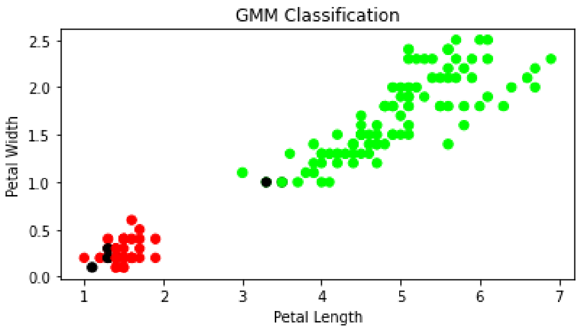
The accuracy score of EM: 0.62

The Confusion matrix of EM: [[45 0 5]

 [0 48 2]

 [0 50 0]]





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:

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]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
```

[4.8 3.1 1.6 0.2]

[5.4 3.4 1.5 0.4]

[5.2 4.1 1.5 0.1]

[5.5 4.2 1.4 0.2]

[4.9 3.1 1.5 0.2]

[5. 3.2 1.2 0.2]

[5.5 3.5 1.3 0.2]

[4.9 3.6 1.4 0.1]

[4.4 3. 1.3 0.2]

[5.1 3.4 1.5 0.2]

[5. 3.5 1.3 0.3]

[4.5 2.3 1.3 0.3]

[4.4 3.2 1.3 0.2]

[5. 3.5 1.6 0.6]

[5.1 3.8 1.9 0.4]

[4.8 3. 1.4 0.3]

[5.1 3.8 1.6 0.2]

[4.6 3.2 1.4 0.2]

[5.3 3.7 1.5 0.2]

[5. 3.3 1.4 0.2]

[7. 3.2 4.7 1.4]

[6.4 3.2 4.5 1.5]

[6.9 3.1 4.9 1.5]

[5.5 2.3 4. 1.3]

[6.5 2.8 4.6 1.5]

[5.7 2.8 4.5 1.3]

[6.3 3.3 4.7 1.6]

[4.9 2.4 3.3 1.]

[6.6 2.9 4.6 1.3]

[5.2 2.7 3.9 1.4]

[5. 2. 3.5 1.]

[5.9 3. 4.2 1.5]

```
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
```

[5.6 2.7 4.2 1.3]

[5.7 3. 4.2 1.2]

[5.7 2.9 4.2 1.3]

[6.2 2.9 4.3 1.3]

[5.1 2.5 3. 1.1]

[5.7 2.8 4.1 1.3]

[6.3 3.3 6. 2.5]

[5.8 2.7 5.1 1.9]

[7.1 3. 5.9 2.1]

[6.3 2.9 5.6 1.8]

[6.5 3. 5.8 2.2]

[7.6 3. 6.6 2.1]

[4.9 2.5 4.5 1.7]

[7.3 2.9 6.3 1.8]

[6.7 2.5 5.8 1.8]

[7.2 3.6 6.1 2.5]

[6.5 3.2 5.1 2.]

[6.4 2.7 5.3 1.9]

[6.8 3. 5.5 2.1]

[5.7 2.5 5. 2.]

[5.8 2.8 5.1 2.4]

[6.4 3.2 5.3 2.3]

[6.5 3. 5.5 1.8]

[7.7 3.8 6.7 2.2]

[7.7 2.6 6.9 2.3]

[6. 2.2 5. 1.5]

[6.9 3.2 5.7 2.3]

[5.6 2.8 4.9 2.]

[7.7 2.8 6.7 2.]

[6.3 2.7 4.9 1.8]

[6.7 3.3 5.7 2.1]

[7.2 3.2 6. 1.8]

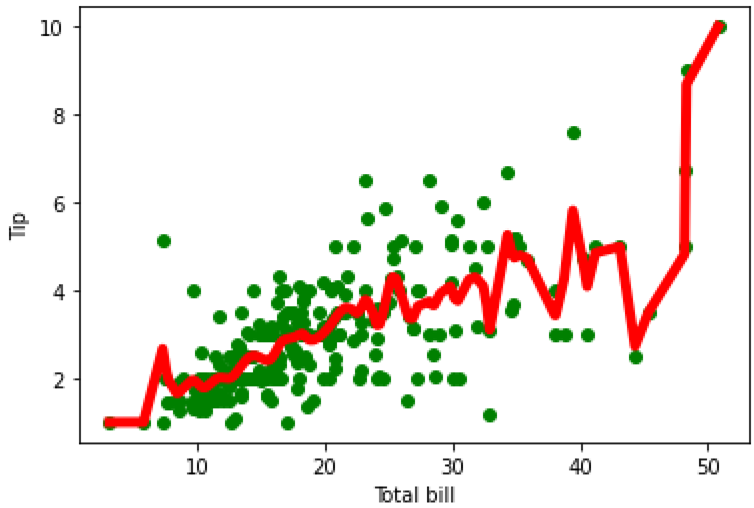
```
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Confusion Matrix
[[16  0  0]
```

```
 [ 0 14  2]
 [ 0  0 13]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.88      0.93        16
           2       0.87      1.00      0.93        13

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.95        45
weighted avg       0.96      0.96      0.96        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

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

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

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

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

# load data points
data = pd.read_csv('Desktop/dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)
```

```
#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
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();
```



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
18.35	2.5	Male	No	Sat	Dinner	4
15.06	3	Female	No	Sat	Dinner	2
20.69	2.45	Female	No	Sat	Dinner	4
17.78	3.27	Male	No	Sat	Dinner	2
24.06	3.6	Male	No	Sat	Dinner	3
16.31	2	Male	No	Sat	Dinner	3
16.93	3.07	Female	No	Sat	Dinner	3
18.69	2.31	Male	No	Sat	Dinner	3
31.27	5	Male	No	Sat	Dinner	3
16.04	2.24	Male	No	Sat	Dinner	3
17.46	2.54	Male	No	Sun	Dinner	2
13.94	3.06	Male	No	Sun	Dinner	2
9.68	1.32	Male	No	Sun	Dinner	2
30.4	5.6	Male	No	Sun	Dinner	4
18.29	3	Male	No	Sun	Dinner	2
22.23	5	Male	No	Sun	Dinner	2
32.4	6	Male	No	Sun	Dinner	4
28.55	2.05	Male	No	Sun	Dinner	3
18.04	3	Male	No	Sun	Dinner	2
12.54	2.5	Male	No	Sun	Dinner	2
10.29	2.6	Female	No	Sun	Dinner	2
34.81	5.2	Female	No	Sun	Dinner	4
9.94	1.56	Male	No	Sun	Dinner	2
25.56	4.34	Male	No	Sun	Dinner	4
19.49	3.51	Male	No	Sun	Dinner	2
38.01	3	Male	Yes	Sat	Dinner	4
26.41	1.5	Female	No	Sat	Dinner	2
11.24	1.76	Male	Yes	Sat	Dinner	2
48.27	6.73	Male	No	Sat	Dinner	4
20.29	3.21	Male	Yes	Sat	Dinner	2
13.81	2	Male	Yes	Sat	Dinner	2
11.02	1.98	Male	Yes	Sat	Dinner	2
18.29	3.76	Male	Yes	Sat	Dinner	4
17.59	2.64	Male	No	Sat	Dinner	3
20.08	3.15	Male	No	Sat	Dinner	3
16.45	2.47	Female	No	Sat	Dinner	2
3.07	1	Female	Yes	Sat	Dinner	1
20.23	2.01	Male	No	Sat	Dinner	2
15.01	2.09	Male	Yes	Sat	Dinner	2
12.02	1.97	Male	No	Sat	Dinner	2
17.07	3	Female	No	Sat	Dinner	3
26.86	3.14	Female	Yes	Sat	Dinner	2
25.28	5	Female	Yes	Sat	Dinner	2
14.73	2.2	Female	No	Sat	Dinner	2
10.51	1.25	Male	No	Sat	Dinner	2
17.92	3.08	Male	Yes	Sat	Dinner	2
27.2	4	Male	No	Thur	Lunch	4
22.76	3	Male	No	Thur	Lunch	2
17.29	2.71	Male	No	Thur	Lunch	2
19.44	3	Male	Yes	Thur	Lunch	2
16.66	3.4	Male	No	Thur	Lunch	2
10.07	1.83	Female	No	Thur	Lunch	1
32.68	5	Male	Yes	Thur	Lunch	2
15.98	2.03	Male	No	Thur	Lunch	2
34.83	5.17	Female	No	Thur	Lunch	4
13.03	2	Male	No	Thur	Lunch	2
18.28	4	Male	No	Thur	Lunch	2
24.71	5.85	Male	No	Thur	Lunch	2
21.16	3	Male	No	Thur	Lunch	2
28.97	3	Male	Yes	Fri	Dinner	2
22.49	3.5	Male	No	Fri	Dinner	2
5.75	1	Female	Yes	Fri	Dinner	2
16.32	4.3	Female	Yes	Fri	Dinner	2
22.75	3.25	Female	No	Fri	Dinner	2
40.17	4.73	Male	Yes	Fri	Dinner	4
27.28	4	Male	Yes	Fri	Dinner	2
12.03	1.5	Male	Yes	Fri	Dinner	2
21.01	3	Male	Yes	Fri	Dinner	2
12.46	1.5	Male	No	Fri	Dinner	2
11.35	2.5	Female	Yes	Fri	Dinner	2
15.38	3	Female	Yes	Fri	Dinner	2
44.3	2.5	Female	Yes	Sat	Dinner	3
22.42	3.48	Female	Yes	Sat	Dinner	2
20.92	4.08	Female	No	Sat	Dinner	2
15.36	1.64	Male	Yes	Sat	Dinner	2
20.49	4.06	Male	Yes	Sat	Dinner	2
25.21	4.29	Male	Yes	Sat	Dinner	2
18.24	3.76	Male	No	Sat	Dinner	2
14.31	4	Female	Yes	Sat	Dinner	2
14	3	Male	No	Sat	Dinner	2
7.25	1	Female	No	Sat	Dinner	1
38.07	4	Male	No	Sun	Dinner	3
23.95	2.55	Male	No	Sun	Dinner	2
25.71	4	Female	No	Sun	Dinner	3
17.31	3.5	Female	No	Sun	Dinner	2
29.93	5.07	Male	No	Sun	Dinner	4
10.65	1.5	Female	No	Thur	Lunch	2
12.43	1.8	Female	No	Thur	Lunch	2
24.08	2.92	Female	No	Thur	Lunch	4
11.69	2.31	Male	No	Thur	Lunch	2
13.42	1.68	Female	No	Thur	Lunch	2
14.26	2.5	Male	No	Thur	Lunch	2
15.95	2	Male	No	Thur	Lunch	2
12.48	2.52	Female	No	Thur	Lunch	2
29.8	4.2	Female	No	Thur	Lunch	6
8.52	1.48	Male	No	Thur	Lunch	2
14.52	2	Female	No	Thur	Lunch	2
11.38	2	Female	No	Thur	Lunch	2
22.82	2.18	Male	No	Thur	Lunch	3
19.08	1.5	Male	No	Thur	Lunch	2
20.27	2.83	Female	No	Thur	Lunch	2
11.17	1.5	Female	No	Thur	Lunch	2
12.26	2	Female	No	Thur	Lunch	2
18.26	3.25	Female	No	Thur	Lunch	2
8.51	1.25	Female	No	Thur	Lunch	2
10.33	2	Female	No	Thur	Lunch	2
14.15	2	Female	No	Thur	Lunch	2
16	2	Male	Yes	Thur	Lunch	2
13.16	2.75	Female	No	Thur	Lunch	2
17.47	3.5	Female	No	Thur	Lunch	2
34.3	6.7	Male	No	Thur	Lunch	6
41.19	5	Male	No	Thur	Lunch	5
27.05	5	Female	No	Thur	Lunch	6
16.43	2.3	Female	No	Thur	Lunch	2