VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING

Submitted by

RAMYA RAMESH (1BM19CS227)

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
April-2022 to July-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 **RAMYA RAMESH (1BM19CS227),** 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 academic year 2021-2022. The Lab report has been approved as it satisfies the academic requirements in respect of an **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Dr. K Panimozhi	**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
1	**<u>FIND-S :-</u> Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.**
2	**<u>CANDIDATE ELIMINATION :-</u> 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**
3	**<u>ID3 :-</u> 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**
4	**<u>NAÏVE BAYESIAN CLASSIFIER</u> :- 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**
5	**<u>LINEAR REGRESSION :-</u> Implement the Linear Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.**
6	**<u>BAYESIAN NETWORK :-</u>Write a program to construct a Bayesian network considering training data. Use this model to make predictions**
7	**<u>K-MEANS :-</u>Apply k-Means algorithm to cluster a set of data stored in a .CSV file.**
8	**<u>EM :-</u>Apply EM algorithm to cluster a set of data stored in a .CSV file.Compare the results of k-Means algorithm and EM algorithm.**
9	**<u>K NEAREST NEIGHBOUR :-</u>Write a program to implement k- NearestNeighbor algorithm to classify the iris data set. Print both correct and wrong predictions.**
10	**<u>LOCALLY WEIGHTED REGRESSION :-</u>Implement the non- parametric Locally Weighted Regression algorithm in order to fit data points. Select the appropriate data set for your experiment and draw graphs.**

<u>Course Outcome:-</u>

At the end of the course the student will be able to

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

NAME: RAMYA RAMESH
USN: 1BM19CS227
COURSE NAME: MACHINE LEARNING
COURSE CODE: 20CS6PCMAL

ML LAB REPORT

PROGRAM-1

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

TRAINING DATA USED: enjoysport.csv

1	sky	air_temp	humidity	wind	water	forecast	enjoy_sport
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

CODE:

```
import csv
a=[]
with open(r'C:\Users\bmsce\Desktop\enjoysport.csv','r') as csvfile:
    next(csvfile)
    for r in csv.reader(csvfile):
        a.append(r)
    print(a)
num_attr = len(a[0]) - 1
hyp = ['0']*num_attr
print("\n The initial hypothesis is:",hyp)

for i in range(0,len(a)):
    if a[i][num_attr]=='yes':
        print("\n Instance",i+1,"is",a[i],"and it is a Positive Instance")
        for j in range(0,num_attr):
            if hyp[j]=='0' or hyp[j]==a[i][j]:
                hyp[j]=a[i][j]
            else:
                hyp[j]='?'
        print("The hypothesis for the training instance",i+1,"is:",hyp,"\n")
    if a[i][num_attr]=='no':
        print("\n Instance",i+1,"is",a[i],"and it is a Negative Instance. Hence, it is IGNORED.")
        print("The hypothesis for the training instance",i+1,"is:",hyp,"\n")
print("\n The Maximally specific hypothesis for the training instance is:",hyp)
```

OUTPUT:

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

 Instance 1 is ['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'] and it is a 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 it is a 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 it is a Negative Instance. Hence, it is 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 it is a 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', '?', '?']
```

PROGRAM-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.

TRAINING DATA USED: library.csv

1	citations	size	inlibrary	price	editions	buy
2	many	big	no	expensive	many	yes
3	some	small	no	affordable	one	no
4	many	small	yes	affordable	many	yes
5	many	small	no	expensive	many	yes

CODE:

```
import csv

data=[]
with open(r"C:\Users\admin\Desktop\library.csv") as csvfile:
    next(csvfile)
    for r in csv.reader(csvfile):
        data.append(r)
    print(data)
```

```
num=len(data[0])-1
s=['0']*num
g=[['?' for i in range(len(s))] for j in range(len(s))]

for i in range(0,len(data)):
    if data[i][num]=="yes":
        for j in range(0,num):
            if s[j]=='0' or s[j]==data[i][j]:
                s[j]=data[i][j]
            else:
                s[j]='?'
                g[j][j]='?'

    elif data[i][num]=="no":
        for j in range(0,num):
            if s[j]!=data[i][j]:
                g[j][j]=s[j]
            else:
                g[j][j]="?"
    print("\nSteps of Candidate Elimination Algorithm",i+1)
    print(s)
    print(g)
gh=[]
for i in g:
    for j in i:
        if j!='?':
            gh.append(i)
            break
print("\nFinal specific hypothesis:\n",s)

print("\nFinal general hypothesis:\n",gh)
```

OUTPUT:

```
[['many', 'big', 'no', 'expensive', 'many', 'yes'], ['some', 'small', 'no', 'affordable', 'one', 'no'], ['many', 'small', 'yes', 'affordable', 'many',
'yes'], ['many', 'small', 'no', 'expensive', 'many', 'yes']]

Steps of Candidate Elimination Algorithm 1
['many', 'big', 'no', 'expensive', 'many']
[['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]

Steps of Candidate Elimination Algorithm 2
['many', 'big', 'no', 'expensive', 'many']
[['many', '?', '?', '?', '?'], ['?', 'big', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', 'expensive', '?'], ['?', '?', '?', '?', 'man
y']]

Steps of Candidate Elimination Algorithm 3
['many', '?', '?', '?', 'many']
[['many', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'many']]

Steps of Candidate Elimination Algorithm 4
['many', '?', '?', '?', 'many']
[['many', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', 'many']]

Final specific hypothesis:
 ['many', '?', '?', '?', 'many']

Final general hypothesis:
 [['many', '?', '?', '?', '?'], ['?', '?', '?', '?', 'many']]
```

PROGRAM-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.

TRAINING DATA USED: PlayTennis.csv

	Outlook	Temperature	Humidity	Wind	Play Tennis
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

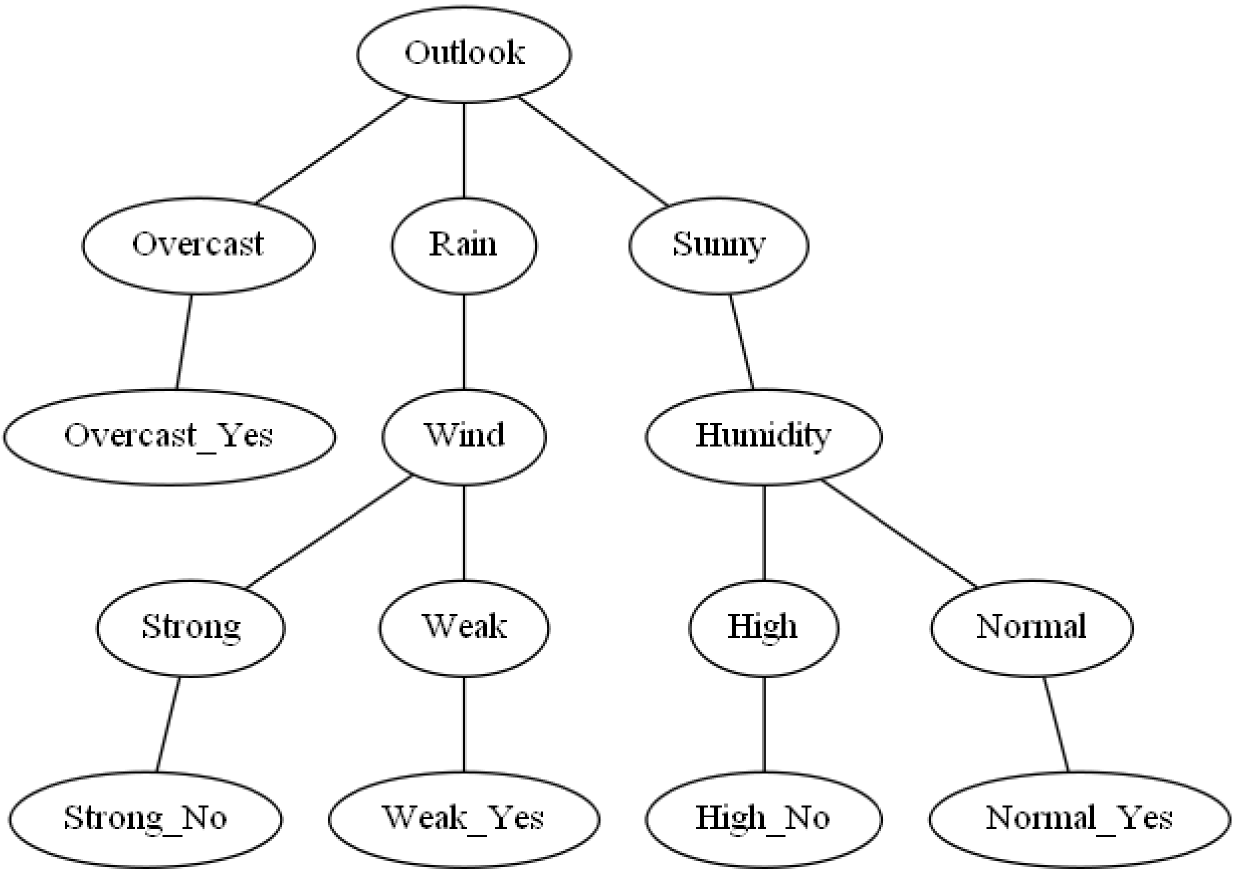
CODE:

```
import pandas as pd
import numpy as np
df = pd.read_csv(r'C:\Users\Ramya\Downloads\PlayTennis.csv')
```

```
print("\n Input Data Set is:\n", df)

eps = np.finfo(float).eps

target=df.keys()[-1]
print('Target: ',target)
attributes=list(df.keys())
attributes.remove(target)
print('Predicting Attributes: ', attributes)

values_in_target = df[target].unique()
print(values_in_target)

def find_entropy_whole(df):
    overall_entropy = 0
    for value in values_in_target:
        p = df[target].value_counts()[value] / len(df[target])
        overall_entropy += -p * np.log2(p)
    return overall_entropy
find_entropy_whole(df)
def find_entropy_of_attribute(df, attribute):
    values_in_attribute = df[attribute].unique()
    entropy_attribute = 0
    for value_in_attribute in values_in_attribute:
        overall_entropy = 0
        for value_in_target in values_in_target:
            num = len(df[attribute][df[attribute] == value_in_attribute][df[target] == value_in_target])
            den = len(df[attribute][df[attribute] == value_in_attribute])
            p = num / (den+eps)
            overall_entropy += -p * np.log2(p+eps)
        p2 = den / len(df)
        entropy_attribute += -p2 * overall_entropy
    return abs(entropy_attribute)

for attribute in df.keys()[:-1]:
    print(f'Entropy of the attribute "{attribute}" is :', find_entropy_of_attribute(df, attribute))

def find_best_attribute_to_divide(df):
    IG = []
    all_attributes = df.keys()[:-1]
    for attribute in all_attributes:
        IG.append(find_entropy_whole(df) - find_entropy_of_attribute(df, attribute))
    print(IG)
    index_of_attribute_with_max_IG = np.argmax(IG)
    best_attribute = all_attributes[index_of_attribute_with_max_IG]
    return best_attribute
find_best_attribute_to_divide(df)

def buildTree(df, tree=None):
```

```
    node = find_best_attribute_to_divide(df)
    attValue = np.unique(df[node])
    if tree is None:
        tree = {}
        tree[node] = {}
    for value in attValue:
        subtable = df[df[node] == value].reset_index(drop=True)
        clValue, counts = np.unique(subtable[target], return_counts=True)
        if len(counts) == 1: #Checking purity of subset
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable) # Calling the function recusively
    return tree

buildTree(df)

import pydot

data = buildTree(df)
def draw(parent_name, child_name):
    edge = pydot.Edge(parent_name, child_name)
    graph.add_edge(edge)

def visit(node, parent=None):
    for k,v in node.items():
        if isinstance(v, dict):
            # We start with the root node whose parent is None
            # we don't want to graph the None node
            if parent:
                draw(parent, k)
            visit(v, k)
        else:
            draw(parent, k)
            # drawing the label using a distinct name
            draw(k, k+'_'+v)

graph = pydot.Dot(graph_type='graph')
visit(data)
graph.write_png('output_graph.png')
```

OUTPUT:



PROGRAM-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.

GAUSSIAN NAIVE BAYES

TRAINING DATA USED: pima_indian.csv

num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	diabetes
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1

...

<u>CODE:</u>

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

df = pd.read_csv(r"C:\Users\admin\Desktop\pima_indian.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 # these are factors for the prediction
y = df[predicted_class_names].values # this is what we want to predict
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)

print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)


# Training Naive Bayes (NB) classifier on training data.

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

#printing Confusion matrix, accuracy, Precision and Recall

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
print('\n The value of Precision', metrics.precision_score(ytest,predicted))
print('\n The value of Recall', metrics.recall_score(ytest,predicted))
```

print("Predicted Value for individual Test Data:", predictTestData)

OUTPUT:

```
 the total number of Training Data : (514, 1)

 the total number of Test Data : (254, 1)

 Confusion matrix
[[136  28]
 [ 31  59]]

 Accuracy of the classifier is 0.7677165354330708

 The value of Precision 0.6781609195402298

 The value of Recall 0.6555555555555556
Predicted Value for individual Test Data: [1]
```

MULTINOMIAL NAIVE BAYES

TRAINING DATA USED: species.csv

1	color	legs	height	species
2	white	3	short	M
3	green	2	tall	M
4	green	3	short	M
5	white	3	short	M
6	green	2	short	H
7	white	2	tall	H
8	white	2	tall	H
9	white	2	short	H
10	green	3	tall	H

CODE:

```
import pandas as pd
import numpy as np

dataset = pd.read_csv(r'C:\Users\admin\Desktop\species.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
X[:,2] = le.fit_transform(X[:,2])

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 = 0)

class MultiNB(object):
```

```
    def __init__(self):
        self.priors = None
        self.params = None
        self.unique_labels = None

    def fit(self, X, y, alpha=1.0):
        assert ((alpha <= 1.0) and (alpha > 0.0)), "ERROR: smoothing parameter alpha should have value
within [0.0, 1.0]!"
        self.unique_labels = np.unique(y)
        self.params = np.zeros(shape = (X.shape[1], len(self.unique_labels)))
        self.priors = np.zeros(shape = (len(self.unique_labels),))

        for ix,label in enumerate(self.unique_labels):
            # Boolean mask for extracting training samples corresponding to label
            mask = (y == label)

            # Add-1 smoothing; verified numerically that probabilities column-sum to 1
            token_counts_in_label = (np.sum(X[mask, :], axis=0) + alpha)
            total_tokens_in_label = np.sum(X[mask, :]) + X.shape[1] * alpha
            self.params[:, ix] = token_counts_in_label / total_tokens_in_label
            self.priors[ix] = np.sum(mask)/len(y)

    def predict_log_likelihood(self, X):

        log_params = np.log(self.params)
        log_likelihoods = np.dot(X, log_params)
        return log_likelihoods

    def predict(self, X):

        log_likelihoods = self.predict_log_likelihood(X)
        index_to_label = np.argmax(log_likelihoods, axis=1)
        pred_y = np.asarray([self.unique_labels[index] for index in index_to_label])

        return pred_y

like = MultiNB()
like.fit(X_train,y_train)

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
y_pred = like.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

OUTPUT:

```
print(y_pred)
```

```
['M' 'M']

[[0 1]
 [0 1]]
              precision    recall  f1-score   support

           H       0.00      0.00      0.00         1
           M       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

PROGRAM-5

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

TRAINING DATA USED: heart_disease.csv

1	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
2	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
3	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
4	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
5	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
6	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
7	56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
8	62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
9	57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
10	63	1	4	130	254	0	2	147	0	1.4	2	1	7	2

…

CODE:

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

heartDisease = pd.read_csv(r"C:\Users\admin\Desktop\heart_disease.csv")
heartDisease = heartDisease.replace('?',np.nan)
```

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

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

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

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

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

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

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

OUTPUT:

```
 Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1.Probability of HeartDisease given evidence= restecg :1
  0%|          | 0/4 [00:00<?, ?it/s]
  0%|          | 0/4 [00:00<?, ?it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+
```

```
2.Probability of HeartDisease given evidence= cp:2
 0%|          | 0/3 [00:00<?, ?it/s]
 0%|          | 0/3 [00:00<?, ?it/s]
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

PROGRAM-6

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

TRAINING DATA USED: income.csv

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

...

CODE:

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline

df = pd.read_csv('C:/Users/admin/downloads/income.csv')
df.head(10)
```

	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

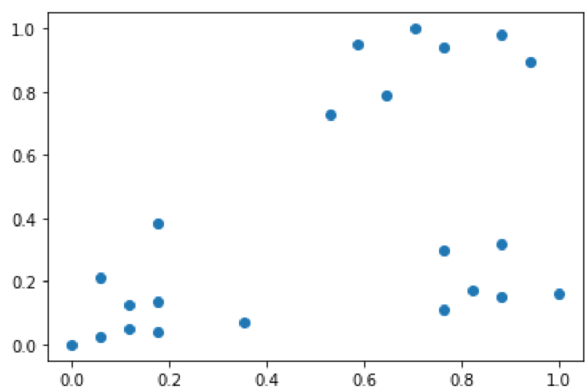
```
scaler = MinMaxScaler()
scaler.fit(df[['Age']])
df[['Age']] = scaler.transform(df[['Age']])
scaler.fit(df[['Income($)']])
df[['Income($)']] = scaler.transform(df[['Income($)']])
df.head(10)
```

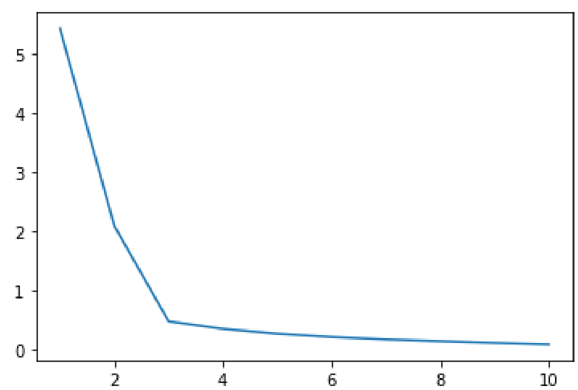
	Name	Age	Income($)
0	Rob	0.058824	0.213675
1	Michael	0.176471	0.384615
2	Mohan	0.176471	0.136752
3	Ismail	0.117647	0.128205
4	Kory	0.941176	0.897436
5	Gautam	0.764706	0.940171
6	David	0.882353	0.982906
7	Andrea	0.705882	1.000000
8	Brad	0.588235	0.948718
9	Angelina	0.529412	0.726496

```
plt.scatter(df['Age'], df['Income($)'])
k_range = range(1, 11)
sse = []
for k in k_range:
 kmc = KMeans(n_clusters=k)
 kmc.fit(df[['Age', 'Income($)']])
 sse.append(kmc.inertia_)
sse
plt.xlabel = 'Number of Clusters'
plt.ylabel = 'Sum of Squared Errors'
plt.plot(k_range, sse)
km = KMeans(n_clusters=3)
km
y_predict = km.fit_predict(df[['Age', 'Income($)']])
y_predict
```

df['cluster'] = y_predict
df.head()
df0 = df[df.cluster == 0]
df1 = df[df.cluster == 1]
df2 = df[df.cluster == 2]
km.cluster_centers_
p1 = plt.scatter(df0['Age'], df0['Income($)'],color='red')
p2 = plt.scatter(df1['Age'], df1['Income($)'],color='blue')
p3 = plt.scatter(df2['Age'], df2['Income($)'],color='green')
c = plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color='black')
plt.legend((p1, p2, p3, c),
 ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))

OUTPUT:



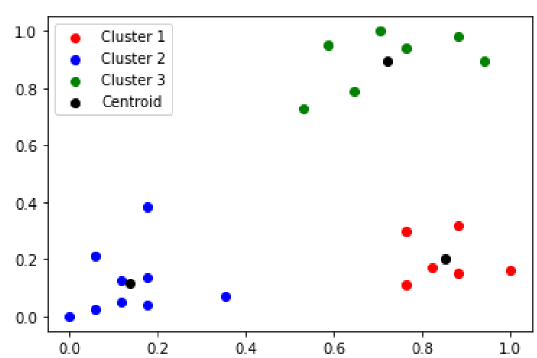


```
y_predict = km.fit_predict(df[['Age', 'Income($)']])
y_predict
```

```
array([1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0])
```

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

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



PROGRAM-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.

TRAINING DATA USED: load_iris (dataset.csv)

1	**5.1**	**3.5**	**1.4**	**0.2**	**Iris-setosa**
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa

...

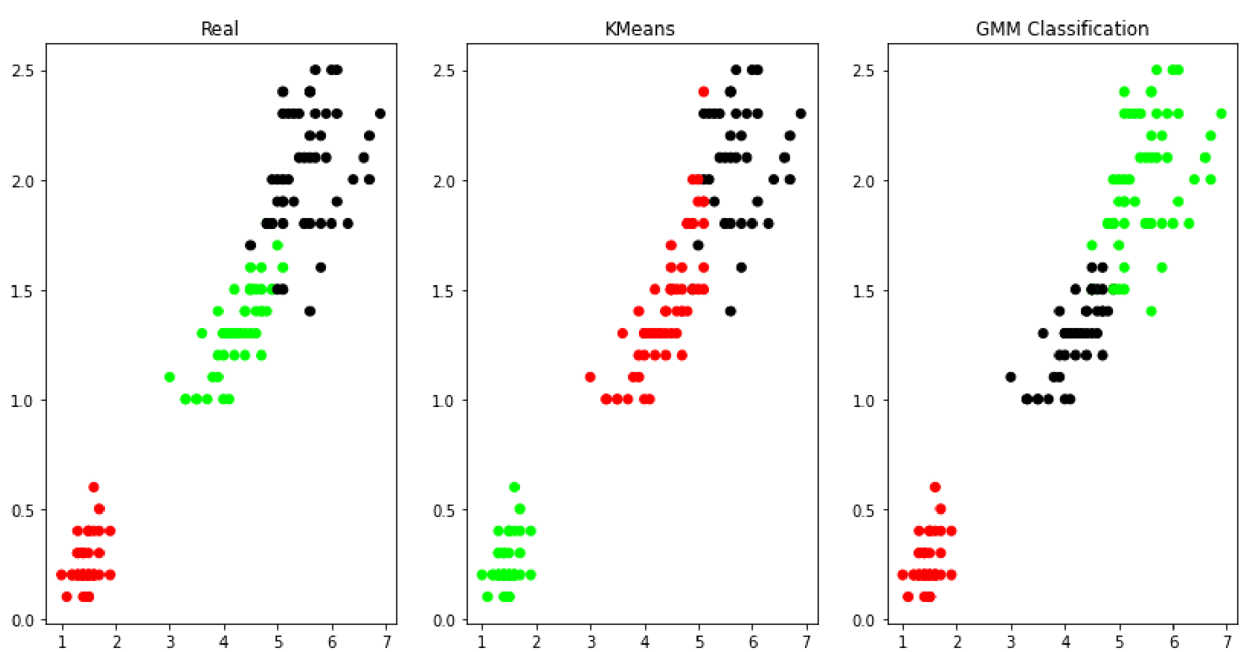
CODE:

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

names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']

dataset = pd.read_csv("C:/Users/admin/Downloads/dataset.csv", names=names)

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

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])


plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])


model=KMeans(n_clusters=3, random_state=0).fit(X)
```

```
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])
print('The confusion matrix of K-Mean:\n',metrics.confusion_matrix(y, model.labels_))
print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))



gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])
print('The confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_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:  0.36666666666666664
The Confusion matrix of EM:
  [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
```



PROGRAM-8

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

<u>TRAINING DATA USED:</u> load_iris

```
data = datasets.load_iris()
df = pd.DataFrame(data=data.data, columns=data.feature_names)
df.head(10)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1

<u>CODE (built-in):</u>

```
import numpy as np
import pandas as pd
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
x =  data.data
y = data.target
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=42)

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)

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))
```

CODE:

```
from scipy import stats
def EuclideanDistance(x1, x2):
    return np.sum((x1 - x2) ** 2, axis=1)
class KNN:
    def __init__(self, k, distance_metric=EuclideanDistance, task_type='Classification'):
        self._k = k
        self._distance_metric = distance_metric
        self._task_type = task_type
    def fit(self, X, y):
        self._X = X
        self._y = y
    def predict(self, newExample):
        distance_vector = self._distance_metric(self._X, newExample)
        k_nearest_neighbors_indices = np.argpartition(distance_vector, self._k)[:self._k]
        k_nearest_neighbors = self._y[k_nearest_neighbors_indices]
        if self._task_type == 'Classification':
            label = stats.mode(k_nearest_neighbors)[0]
        else:
            label = k_nearest_neighbors.mean()
        return label, k_nearest_neighbors_indices
    def eval(self, X_test, y_test):
        y_pred = np.zeros(y_test.shape)
        for i in range(y_test.shape[0]):
            y_predicted[i], _ = self.predict(X_test[i,:])
        if self._task_type == 'Classification': # for all examples
            error = np.mean(y_test == y_pred, axis=0)
        else:
            error_vector = y_predicted - y_test
            error = np.sqrt((error_vector.T @ error_vector) / error_vector.ravel().shape[0])
        return error
```

OUTPUT:

```
Confusion Matrix
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
pred = KNN(k=5)
pred.fit(X_train,y_train)
```

```
pred.predict(X_test[0])
```

```
(array([1], dtype=int64), array([ 67, 263, 538, 276, 490], dtype=int64))
```

```
y_test[0]
```

```
1
```

PROGRAM-9

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

TRAINING DATA USED: fetch_california_housing

```
from sklearn.datasets import fetch_california_housing
X, y = fetch_california_housing(return_X_y=True,as_frame=True)
```

```
X.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

```
y.head()
```

```
0    4.526
1    3.585
2    3.521
3    3.413
4    3.422
Name: MedHouseVal, dtype: float64
```

CODE:

```
def __init__(self):
   self.t0 = 200
   self.t1 = 100000
def predict (self, X):
   return X@w
def loss (self, X, y):
   e = y - self.predict(X)
   return 0.5 *(e.T @ e)
def rmse(self,X, y):
   return np.sqrt(2/X.shape[0] * self.loss(X, y))
def fit(self, X, y):
   self.w = np.linalg.pinv(X) @ y
   return self.w
def calculate_gradient(self, X, y):
```

```
    return X.T @ (self.predict(X) - y)
def update_weights(self, grad, lr):
    return (self.w - lr * grad)
def learning_schedule(self, t):
    return self.t0 / (self.t0 + self.t1)
def gd(self, X, y, num_epochs, lr):
    self.w = np.zeros(X.shape[1])
    self.w_all = list()
    self.err_all = list()
    for i in range(epochs):
        dJdw = calculate_gradient(X, y)
        self.w_all.append(self.w)
        self.err_all.append(self.loss(X, y))
        self.w = self.update_weights(dJdw, lr)
    return self.w
```

CODE: (built-in)

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

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)
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

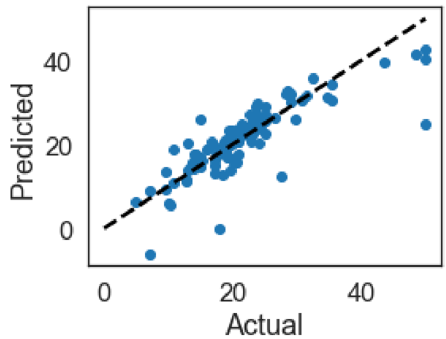
OUTPUT:

```
y_pred = linreg.predict(X_test)
pd.DataFrame(data={'Actuals': y_test, 'Predictions': y_pred})
```

	Actuals	Predictions
20046	0.47700	0.719123
3024	0.45800	1.764017
15663	5.00001	2.709659
20484	2.18600	2.838926
9814	2.78000	2.604657
...	...	...
15362	2.63300	1.991746
16623	2.66800	2.249839
18086	5.00001	4.468770
2144	0.72300	1.187511
3665	1.51500	2.009403

4128 rows × 2 columns

```
plt.figure(figsize=(4, 3))
plt.scatter(y_test, y_pred)
plt.plot([0, 50], [0, 50], '--k')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.show()
```



PROGRAM-10

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

TRAINING DATA USED: tips.csv

1	total_bill	tip	sex	smoker	day	time	size
2	16.99	1.01	Female	No	Sun	Dinner	2
3	10.34	1.66	Male	No	Sun	Dinner	3
4	21.01	3.5	Male	No	Sun	Dinner	3
5	23.68	3.31	Male	No	Sun	Dinner	2
6	24.59	3.61	Female	No	Sun	Dinner	4
7	25.29	4.71	Male	No	Sun	Dinner	4
8	8.77	2.0	Male	No	Sun	Dinner	2
9	26.88	3.12	Male	No	Sun	Dinner	4
10	15.04	1.96	Male	No	Sun	Dinner	2

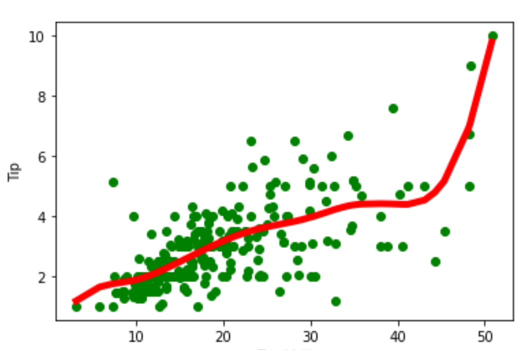
...

CODE:

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

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

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

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

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='green')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'red', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show();

data = pd.read_csv('tips.csv')
bill = np.array(data.total_bill) # We use only Bill amount and Tips data
tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

# increase k to get smooth curves
ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```

<u>OUTPUT:</u>



CODE (built-in):

```
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

def local_regression(x0, X, Y, tau):# add bias term
x0 = np.r_[1, x0] # Add one to avoid the loss in information
X = np.c_[np.ones(len(X)), X]

# fit model: normal equations with kernel
xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W

beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product

# predict value
return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernal Bias Function

n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])

def plot_lwr(tau):
# prediction through regression
prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
plot = figure(plot_width=400, plot_height=400)
plot.title.text='tau=%g' % tau
plot.scatter(X, Y, alpha=.3)
plot.line(domain, prediction, line_width=2, color='red')
return plot

show(gridplot([[plot_lwr(10.), plot_lwr(1.)],
[plot_lwr(0.1), plot_lwr(0.01)]]))
```

OUTPUT:

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-3.07038137 -2.97903806 -3.07809225 -2.93627863 -2.95209929 -3.03687263
 -2.8601589  -2.83440865 -2.98620123]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```