

**B.M.S. COLLEGE OF ENGINEERING BENGALURU**  
Autonomous Institute, Affiliated to VTU



Lab Record

**Machine Learning**

*Submitted in partial fulfillment for the 6<sup>th</sup> Semester Laboratory*

Bachelor of Technology in  
Computer Science and Engineering

*Submitted by:*

**Roshan**

**1BM20CS131**

Department of Computer Science and Engineering B.M.S.  
College of Engineering  
Bull Temple Road, Basavanagudi, Bangalore 560 019  
Mar-July-2023

**B.M.S. COLLEGE OF ENGINEERING**  
**DEPARTMENT OF COMPUTER SCIENCE AND**  
**ENGINEERING**



***CERTIFICATE***

This is to certify that the Machine Learning (20CS6PCMAL) laboratory has been carried out by **ROSHAN (1BM20CS131)** during the 6<sup>th</sup> Semester Mar-July-2023.

Signature of the Faculty Incharge:  
Asha G R  
Assistant Professor  
Department of Computer Science and Engineering  
B.M.S. College of Engineering, Bangalore

## Contents

Lab Program	Unit #	Program Details
1	1	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.
2	1	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	1	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	3	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	3	Write a program to construct a Bayesian network considering training data. Use this model to make predictions.
6	3	Apply k-Means algorithm to cluster a set of data stored in a .CSV file.
7	3	Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.
8	4	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.
9	4	Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.
10	4	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

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

```
In [ ]: # Interactive

len_x = int(input('enter no. of samples: '))
x = []
attrib = input('enter attributes: ').split()
n_attrib = len(attrib)
for i in range(len_x):
    X = input(f'enter sample {i}: ').split()
    x.append(X)
y = []
for i in range(len_x):
    Y = input(f'output for sample {i} (true/false): ')
    Y = Y.lower()
    y.append(True if Y == 'true' else False)
```

```
In [4]: # Read from csv

import pandas as pd
import numpy as np

data = pd.read_csv("data.csv")
print(data)
x = np.array(data)[:, :-1]
y = np.array(data)[:, -1]
len_x = len(x)
print(x)
print(y)

      temp     time is_holiday  result
0   hot  afternoon       yes    True
1   hot    morning        no   False
2  cold  afternoon       yes    True
[['hot' 'afternoon' 'yes']
 ['hot' 'morning' 'no']
 ['cold' 'afternoon' 'yes']]
[True False True]
```

```
In [7]: def findAlgo(x, y, len_x):
    h = x[0] # initial generalization

    for i in range(1, len_x):
        if not y[i]:
            continue

        for j, attrib in enumerate(x[i]):
            if h[j] != attrib:
                h[j] = '?'

    return h
```

```
In [8]: findAlgo(x, y, len_x)
Out[8]: array(['?', 'afternoon', 'yes'], dtype=object)
```

## ALGORITHM:

LAB - 2

I) enjoysports.csv

Sky	AirTemp	Humidity	Wind	Water	Forecast	Enjoysport
sunny	warm	normal	strong	warm	some	yes
sunny	warm	high	strong	warm	some	yes
rainy	cold	high	strong	warm	some	no
sunny	warm	high	strong	cold	change	yes

II) Code:-

```
import csv
a = []
with open ('/kaggle/input/enjoysport/enjoysport.csv', 'r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)
print('The total number of training instances are:', len(a))
num_attributes = len(a[0]) - 1
print('The initial hypothesis is :')
hypothesis = ['0'] * num_attributes
print(hypothesis)
for i in range(len(a)):
    if a[i][num_attributes] == 'yes':
        for j in range(0, num_attributes):
            if hypothesis[j] == '0' or hypothesis[j]
```

$hypothesis[j] = a[i][j]$

else:

$hypothesis[j] \rightarrow ?$

print (\* "The hypothesis for the training instance  $i+1$  is: In" format(i+1), hypothesis)

print (\* "The Maximally specific hypothesis for the training instance is")  
print (hypothesis)

ostotamaz

Output:-

11/9.

The total number of training instances are: 5

The initial hypothesis is:

[ '0', '0', '0', '0', '0', '0' ]

The hypothesis for the training instance 1 is:

[ '0', '0', '0', '0', '0', '0', '0' ]

The hypothesis for the training instance 2 is:

[ 'sunny', 'warm', 'normal', 'strong', 'warm', 'semi' ]

[ 'sunny', 'warm', '?', 'strong', 'warm', 'semi' ]

[ 'sunny', 'warm', '?', 'strong', 'warm', 'semi' ]

[ '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.

```
In [1]: import numpy as np
import pandas as pd

In [2]: data = pd.read_csv('../input/enjoysport/enjoysport.csv')
X = data.to_numpy()[:, :-1]
y = data.to_numpy()[:, -1]

In [3]: def candidateElimination(X, y):
    n_attrib = len(X[0])
    specific_h = ['0' for _ in range(n_attrib)]
    general_h = [['?' for _ in range(n_attrib)] for _ in range(n_attrib)]

    print('===== Iteration 0 =====')
    print(f'Specific Boundary: {specific_h}')
    print(f'General Boundary: {general_h}')
    print()

    specific_h = X[0].copy()

    for i, x in enumerate(X):
        print(f'===== Iteration {i + 1} =====')
        print(f'Instance {i + 1}: {x} \t Target: {y[i]}\n')

        if y[i] == 'yes':
            for j in range(n_attrib):
                if x[j] != specific_h[j]:
                    specific_h[j] = '?'
                    general_h[j][j] = '?'
        else:
            for j in range(n_attrib):
                if x[j] != specific_h[j]:
                    general_h[j][j] = specific_h[j]
                else:
                    general_h[j][j] = '?'

        print(f'Specific Boundary: {specific_h}')
        print(f'General Boundary: {general_h}')
        print()

    general_h = [h for h in general_h if h != ['?' for _ in range(n_attrib)]]

    print('===== Result =====')
    print(f'Specific Boundary: {specific_h}')
    print(f'General Boundary: {general_h}')
    print()

    return list(specific_h), list(general_h)
```

```
In [4]: candidateElimination(X, y)

===== Iteration 0 =====
Specific Boundary: ['0', '0', '0', '0', '0', '0']
General Boundary: [[ '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

===== Iteration 1 =====
Instance 1: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same'] Target: yes
Specific Boundary: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
General Boundary: [[ '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

===== Iteration 2 =====
Instance 2: ['sunny' 'warm' 'high' 'strong' 'warm' 'same'] Target: yes
Specific Boundary: ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
General Boundary: [[ '?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

===== Iteration 3 =====
Instance 3: ['rainy' 'cold' 'high' 'strong' 'warm' 'change'] Target: no
Specific Boundary: ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
General Boundary: [[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

===== Iteration 4 =====
Instance 4: ['sunny' 'warm' 'high' 'strong' 'cool' 'change'] Target: yes
Specific Boundary: ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
General Boundary: [[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

===== Result =====
Specific Boundary: ['sunny' 'warm' '? ' strong' '?' '?']
General Boundary: [[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]]

Out[4]: ([['sunny', 'warm', '?', 'strong', '?', '?'], [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]])
```

## ALGORITHM:

PAGE NO. | Usha Gold

LAB-3

II) sea.csv

play	air temp	humidity	wind	water	forecast	Chou sport
sunny	warm	normal	strong	warm	some	yes
sunny	warm	high	strong	warm	some	yes
raining	cold	high	strong	warm	change	no
sunny	warm	high	strong	cold	change	yes

Code:-

```
import numpy as np
import pandas as pd
data = pd.DataFrame (data = pd.read_csv ('kaggle/
    input / dataset / dataset - 1.csv'))
conceptk = np.array (data.iloc[:, 0:-1])
print (conceptk)
target = np.array (data.iloc[:, -1])
print (target)

def learn (conceptk, target):
    print (conceptk)
    specific_h = conceptk [0].copy ()
    print ("Initialization of specific-h (general-h)")
    print (specific_h)
    general_h = [None] * len (specific_h)
    for i in range (len (specific_h)):
        general_h [i] = specific_h [i]
        for j in range (len (specific_h)):
            if target [j] != general_h [j]:
                general_h [j] = '?'
    print (general_h)
```

```
for i, h in enumerate (conuplr):
    if target[i] == 'yes':
        for x in range (len (specific_h)):
            if h[x] != specific_h[x]:
                specific_h[x] = '?'
                general_h[x][x] = '?'
        print (specific_h)
        print (general_h)
```

```
if target[i] == 'no':
    for x in range (len (specific_h)):
        if h[x] != specific_h[x]:
            general_h[x][x] = specific_h[x]
        else:
            general_h[x][x] = '?'
    general_h[x][x] = '?'
```

```
print ('steps of candidate elimination', i+1)
print (specific_h)
```

```
indice = [i for i, val in enumerate (general_h)
          if val == ['?', '?', '?', '?', '?', '?']]
```

```
for i in indice:
    general_h.remove ([ '?', '?', '?', '?', '?', '?'])
return specific_h, general_h
```

```
s_final, g_final = learn (conuplr, target)
```

```
print ('Find specific-h : ', s_final, sep = '\n')
print ('Find general-h : ', g_final, sep = '\n')
```

**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.

```
In [62]: import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log
```

```
In [63]: df = pd.read_csv('../input/playtennis/playtennis.csv')
df
```

```
Out[63]:
```

	outlook	temperature	humidity	wind	play
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

```
In [64]: def find_entropy(df):
    Class = df.keys()[-1]
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy

def find_entropy_attribute(df,attribute):
    Class = df.keys()[-1]
    target_variables = df[Class].unique()
    variables = df[attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
            num = len(df[attribute][df[attribute]==variable][df[Class] ==target_variable])
            den = len(df[attribute][df[attribute]==variable])
            entropy += num/den * find_entropy(df[df[attribute]==variable])
        entropy2 += fraction*entropy
    return entropy2
```

```

        fraction = num/(den+eps)
        entropy += -fraction*log(fraction+eps)
        fraction2 = den/len(df)
        entropy2 += -fraction2*entropy
    return abs(entropy2)

def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
        IG.append(find_entropy(df)-find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]

def get_subtable(df, node,value):
    return df[df[node] == value].reset_index(drop=True)

def buildTree(df,tree=None):
    Class = df.keys()[-1]
    node = find_winner(df)
    attValue = np.unique(df[node])
    if tree is None:
        tree={}
        tree[node] = {}
    for value in attValue:
        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable['play'],return_counts=True)

        if len(counts)==1:
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable)

    return tree

```

In [65]: t = buildTree(df)  
t

Out[65]: {'outlook': {'overcast': 'yes',
 'rain': {'wind': {'strong': 'no', 'weak': 'yes'}},
 'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}

In [ ]:

## ALGORITHM:

3/5/23

LAB-5

import pandas as pd

import math

import numpy as np

data = pd.read\_csv ("./logistics/input/rushan/3-dataset.csv")  
feature = [feat for feat in data]  
features.remove ("answer")

class Node:

def \_\_init\_\_(self):

self.children = {}

self.value = ""

self.isleaf = False

self.pred = "

def entropy (example):

pos = 0.0

neg = 0.0

for row in example.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 learn (concept, target):
    print (concept)
    specific_h = concept[0].copy()
    print ("Initialization of specific-h & general-h")
    print (specific_h)
    general_h = [ '?' for i in range (len (specific_h))]
    for i in range (len (specific_h)):
        for j in range (len (target)):
            if target[j] == 'yes':
                if specific_h[i] != target[j]:
                    specific_h[i] = '?'
                    general_h[i] = '?'
            else:
                specific_h[i] = target[j]
                general_h[i] = '1'
    print (general_h)

```

```

for i, h in enumerate (concept):
    if target[i] == 'yes':
        for x in range (len (specific_h)):
            if h[x] != specific_h[x]:
                specific_h[x] = '?'
                general_h[x] = '?'
            else:
                print (specific_h)
    print (specific_h)

```

Some code

Val = ['?', '?', '?', '?', '?']

Output

Find specific-h:  
 ['sunny', 'Hot', 'High', 'Weak']

Find general-h:  
 [?]

**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.

```
In [1]: import csv
import random
import math

In [2]: def load_csv(filename):
    lines = csv.reader(open(filename, "r"));
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset

def split_dataset(dataset, splitratio):
    trainsize = int(len(dataset) * splitratio);
    trainset = []
    copy = list(dataset);
    while len(trainset) < trainsize:
        index = random.randrange(len(copy));
        trainset.append(copy.pop(index))
    return [trainset, copy]

def separate_by_class(dataset):
    separated = {}
    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 std_dev(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):
    summaries = [(mean(attribute), std_dev(attribute)) for attribute in zip(*dataset)];
    del summaries[-1]
    return summaries

def summarize_by_class(dataset):
    separated = separate_by_class(dataset);
    summaries = {}
    for classvalue, instances in separated.items():
        summaries[classvalue] = summarize(instances)
    return summaries

def calculate_probability(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 calculate_class_probabilities(summaries, inputvector):
    probabilities = {}
```

```

for classvalue, classsummaries in summaries.items():
    probabilities[classvalue] = 1
for i in range(len(classsummaries)):
    mean, stdev = classsummaries[i]
    x = inputvector[i]
    probabilities[classvalue] *= calculate_probability(x, mean, stdev)
return probabilities

def predict(summaries, inputvector):
    probabilities = calculate_class_probabilities(summaries, inputvector)
    bestLabel, bestProb = None, -1
    for classvalue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classvalue
    return bestLabel

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

def get_accuracy(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

```

```

In [3]: splitratio = 0.67
dataset = load_csv('../input/pima-indians-diabetescsv/pima-indians-diabetes.csv');

trainingset, testset = split_dataset(dataset, splitratio)

print(f'Split {len(dataset)} rows into train={len(trainingset)} and test={len(testset)} rows')

summaries = summarize_by_class(trainingset);
predictions = get_predictions(summaries, testset)
accuracy = get_accuracy(testset, predictions)

print(f'Accuracy of the classifier is :{accuracy}%')

Split 768 rows into train=514 and test=254 rows
Accuracy of the classifier is :64.96062992125984%

```

In [ ]:

## ALGORITHM:

17/5/23

CAB-05

Naive Bayes:-

```
import numpy as np  
import pandas as pd  
import os
```

```
for dirname, _, filenames in os.walk('chess'):  
    input):  
        for filename in filenames:  
            print(os.path.join(dirname, filename))
```

```
import csv  
import random  
import math
```

```
def loadcsv(filename):  
    lines = csv.reader(open(filename, "r"))  
    dataset = list(lines)  
    for i in range(len(dataset)):  
        dataset[i] = [float(r) for r 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:  
        index = random.randrange(len(copy))  
        trainset.append(copy.pop(index))  
    return [trainset, copy]
```

<u>outlook</u>	<u>temperature</u>	<u>humidity</u>	<u>wind</u>	<u>answer</u>
sunny	hot	high	weak	no
sunny	hot	high	strong	no
overcast	hot	high	weak	yes
rain	mild	high	weak	yes
rain	cool	normal	weak	yes
rain	cool	normal	strong	no
overcast	cool	normal	strong	yes
Sunny	mild	high	weak	no
sunny	cool	normal	weak	yes
rain	mild	normal	weak	yes
Sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no

Foggy - yes  
Foggy - no

~~Foggy - yes~~ → 6  
~~Foggy - no~~ → 13

# 17-5-23

duration? → depends upon if tidal influenced  
hours? → tides? → considerate? → waves?  
→ (wind) → if water? → flow?

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

```
In [1]: import numpy as np
import pandas as pd
import csv
!pip install pgmpy
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

Collecting pgmpy
  Downloading pgmpy-0.1.14-py3-none-any.whl (331 kB)
[██████████] 331 kB 3.0 MB/s
Requirement already satisfied: pandas in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.2.3)
Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.7/site-packages (from pgmpy) (0.24.1)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.5.4)
Requirement already satisfied: torch in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.7.0)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages (from pgmpy) (4.59.0)
Requirement already satisfied: joblib in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.0.1)
Requirement already satisfied: pyparsing in /opt/conda/lib/python3.7/site-packages (from pgmpy) (2.4.7)
Requirement already satisfied: statsmodels in /opt/conda/lib/python3.7/site-packages (from pgmpy) (0.12.2)
Requirement already satisfied: networkx in /opt/conda/lib/python3.7/site-packages (from pgmpy) (2.5)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from pgmpy) (1.19.5)
Requirement already satisfied: decorator>=4.3.0 in /opt/conda/lib/python3.7/site-packages (from networkx->pgmpy) (4.4.2)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/conda/lib/python3.7/site-packages (from pandas->pgmpy) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.7/site-packages (from pandas->pgmpy) (2021.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas->pgmpy) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/lib/python3.7/site-packages (from scikit-learn->pgmpy) (2.1.0)
Requirement already satisfied: patsy>=0.5 in /opt/conda/lib/python3.7/site-packages (from statsmodels->pgmpy) (0.5.1)
Requirement already satisfied: future in /opt/conda/lib/python3.7/site-packages (from torch->pgmpy) (0.18.2)
Requirement already satisfied: typing_extensions in /opt/conda/lib/python3.7/site-packages (from torch->pgmpy) (3.7.4.3)
Requirement already satisfied: dataclasses in /opt/conda/lib/python3.7/site-packages (from torch->pgmpy) (0.6)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.14
```

```
In [2]: heartDisease = pd.read_csv('../input/heartdisease/heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\nAttributes and datatypes')
print(heartDisease.dtypes)

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

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

```
Attributes and datatypes
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak       float64
slope          int64
ca             object
thal           object
heartdisease  int64
dtype: object
```

```
In [3]: model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'), ('heartdisease','restecg'),('heartdisease','chol')])
```

```
In [4]: print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

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

Learning CPD using Maximum likelihood estimators

Inferencing with Bayesian Network:

```
In [5]: print('\n1. Probability of HeartDisease given evidence = restecg :')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)

print('\n2. Probability of HeartDisease given evidence = cp :')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)

Finding Elimination Order: :  0%| 0/5 [00:00<?, ?it/s]
  0%| 0/5 [00:00<?, ?it/s]
Finding Elimination Order: : 100%|██████████| 5/5 [00:00<00:00, 480.98it/s]

Eliminating: age:  0%| 0/5 [00:00<?, ?it/s]
Eliminating: sex:  0%| 0/5 [00:00<?, ?it/s]
Eliminating: chol: 0%| 0/5 [00:00<?, ?it/s]
Eliminating: exang: 100%|██████████| 5/5 [00:00<00:00, 92.85it/s]
```

```
1. Probability of HeartDisease given evidence = restecg :
+-----+-----+
| 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 :
Finding Elimination Order: : 0% | 0/5 [00:00<?, ?it/s]
0% | 0/5 [00:00<?, ?it/s]
Eliminating: age: 0% | 0/5 [00:00<?, ?it/s]
Eliminating: sex: 0% | 0/5 [00:00<?, ?it/s]
Eliminating: chol: 0% | 0/5 [00:00<?, ?it/s]
Eliminating: exang: 0% | 0/5 [00:00<?, ?it/s]
Eliminating: restecg: 100% | 5/5 [00:00<00:00, 227.41it/s]
+-----+-----+
| heartdisease | phi(heartdisease) |
+-----+-----+
| heartdisease(0) | 0.3610 |
+-----+-----+
| heartdisease(1) | 0.2159 |
+-----+-----+
| heartdisease(2) | 0.1373 |
+-----+-----+
| heartdisease(3) | 0.1537 |
+-----+-----+
| heartdisease(4) | 0.1321 |
+-----+-----+
```

In [ ]:

## ALGORITHM:

10B-07

29/01/23

Bayesian Network :-

```
import numpy as np
import pandas as pd
import csv
from pgm import MaximumLikelihoodEstimation
from pgm.module import BayesianModel
from pgm.inference import VariableElimination

heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?', np.nan)
print("Sample instances from the dataset are given below")
print(heartDisease.head())
print("Attributes & datatypes")
print(heartDisease.dtypes)

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

print("Training PGM using Maximum Likelihood Estimation")
model.fit(heartDisease, estimator = MaximumLikelihoodEstimator)

print("Inference with Bayesian Network : ")
heart_disease_infer = VariableElimination(model)

print("1. Probability of Heart Disease given restecg = resteg")
q1 = heart_disease_infer.infer_query(variables = ['heartdisease'], evidence = {'restecg': 1})
print(q1)

# q1 = Heart Disease but infer query
```

`print (q')`

`print ('2: Probability of Heartdisease given evidence = cp')`  
 $q2 = \text{HeartDisease} \text{ (inf. query variable)} \geq \{ \text{'Heartdisease'} \}$   
 $(\text{evidence} = \{ \text{'cp'} : 2 \})$   
`print (q2)`

→ Output:-

Simple instances from dataset

Attribute is `dotypes`

Learning CPD using Maximum likelihood estimation

Inference with Bayesian Network:

i) Probability of Heart Disease given evidence = routeq  
 $p_{\text{hi}}(\text{Heartdisease})$

Heartdisease(0)	0.1012
Heartdisease(1)	0.000
Heartdisease(2)	0.2392
Heartdisease(3)	0.2015
Heartdisease(4)	0.4581

ii) Probability of heartdisease given evidence = CP

$p_{\text{hi}}(\text{Heartdisease})$

(0)	0.3610
(1)	0.2159
(2)	0.1373
(3)	0.1537
(4)	0.1821

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

```
In [1]: from sklearn import datasets
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, confusion_matrix
```

### Dataset

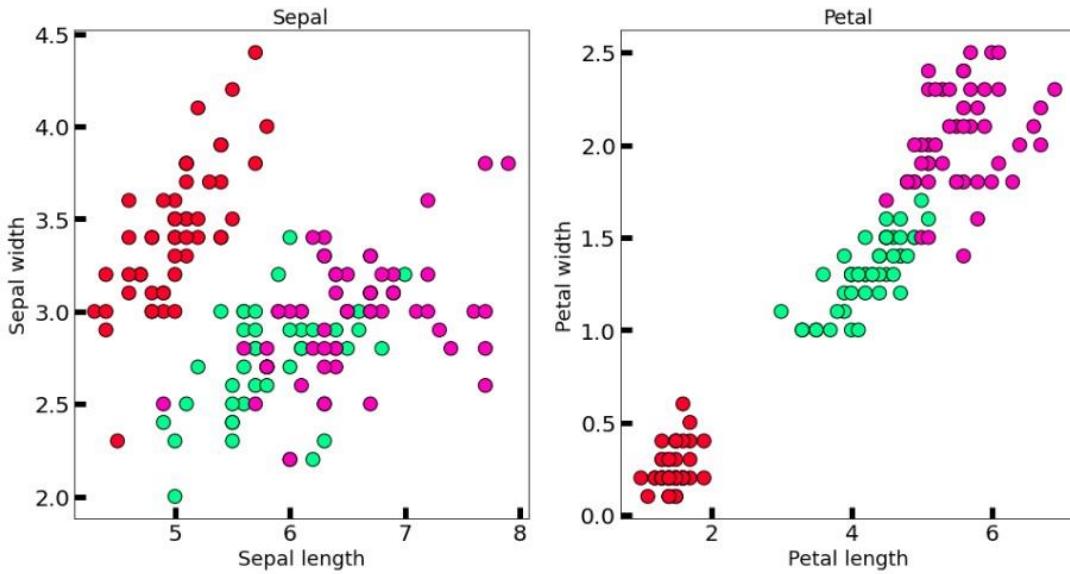
```
In [2]: iris = datasets.load_iris()
iris
```

```
Out[2]: {'data': array([[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],
```

```
In [3]: sepal_X = iris.data[:, :2]
petal_X = iris.data[:, 2:]
y = iris.target
categories = len(iris.target_names)

fig, axes = plt.subplots(1, 2, figsize=(16,8))
axes[0].scatter(sepal_X[:, 0], sepal_X[:, 1], c=y, cmap='gist_rainbow', edgecolor='k', s=150)
axes[1].scatter(petal_X[:, 0], petal_X[:, 1], c=y, cmap='gist_rainbow', edgecolor='k', s=150)
axes[0].set_xlabel('Sepal length', fontsize=18)
axes[0].set_ylabel('Sepal width', fontsize=18)
axes[1].set_xlabel('Petal length', fontsize=18)
axes[1].set_ylabel('Petal width', fontsize=18)
axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[0].set_title('Sepal', fontsize=18)
axes[1].set_title('Petal', fontsize=18)
plt.show()
```



## Model

```
In [4]: model_sepals = KMeans(n_clusters=3)
model_sepals.fit(sepal_X)
model_petal = KMeans(n_clusters=3)
model_petal.fit(petal_X)

Out[4]: KMeans(n_clusters=3)
```

## Analysis

```
In [5]: def plot_centers(sepal_centers, petal_centers):
    plt.scatter([point[0] for point in sepal_centers], [point[1] for point in sepal_centers])
    plt.title('Sepal KMeans Centers')
    plt.show()
```

```

plt.title( f'Petal KMeans Centers {part}' )
plt.show()

def plot_actualvpredicted(X, y, predicted, part):
    fig, axes = plt.subplots(1, 2, figsize=(16,8))
    axes[0].scatter(X[:, 0], X[:, 1], c=y, cmap='gist_rainbow', edgecolor='k', s=150)
    axes[1].scatter(X[:, 0], X[:, 1], c=predicted, cmap='jet', edgecolor='k', s=150)
    axes[0].set_xlabel(f'{part} length', fontsize=18)
    axes[0].set_ylabel(f'{part} width', fontsize=18)
    axes[1].set_xlabel(f'{part} length', fontsize=18)
    axes[1].set_ylabel(f'{part} width', fontsize=18)
    axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
    axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
    axes[0].set_title('Actual', fontsize=18)
    axes[1].set_title('Predicted', fontsize=18)
    plt.show()

def plot_confusion(accuracy, confusion, part):
    print(f'{part} Accuracy: {accuracy}')

    fig, ax = plt.subplots()
    im = ax.imshow(confusion)

    ax.set_xticks(range(categories))
    ax.set_yticks(range(categories))
    ax.set_xticklabels(iris.target_names)
    ax.set_yticklabels(iris.target_names)

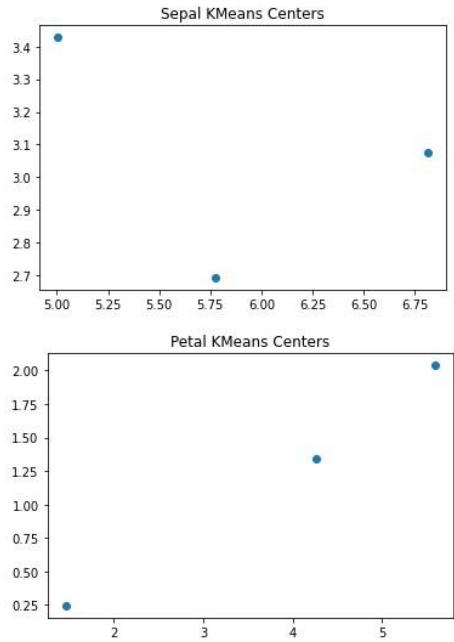
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

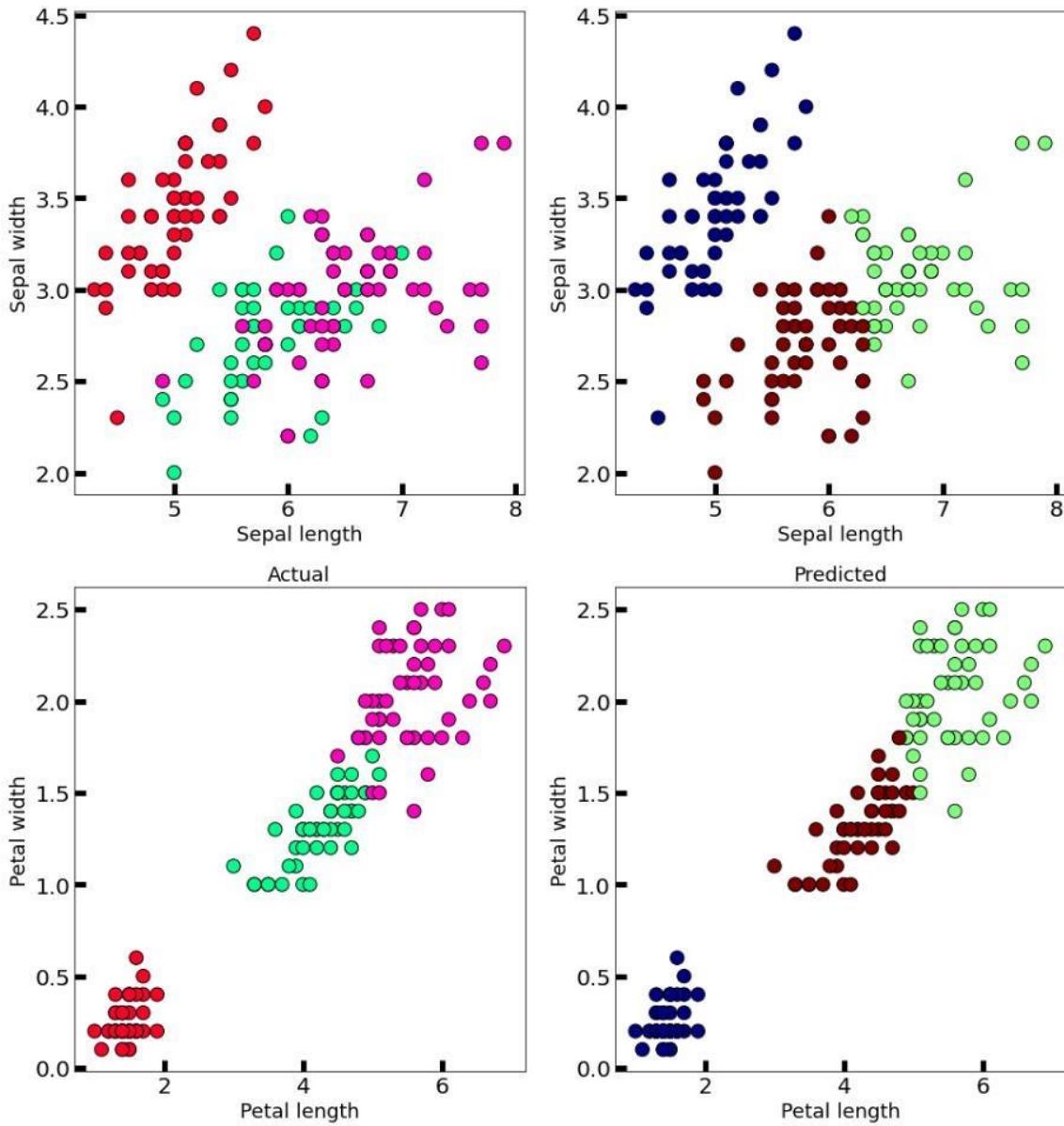
    for i in range(categories):
        for j in range(categories):
            text = ax.text(j, i, confusion[i, j],
                           ha="center", va="center", color="w")

    ax.set_title(f'{part} Confusion Matrix (Actual / Predicted)')
    fig.tight_layout()
    plt.show()

```

In [6]: sepal\_centers = model\_sepal.cluster\_centers\_  
 petal\_centers = model\_petal.cluster\_centers\_  
 plot\_centers(sepal\_centers, petal\_centers)



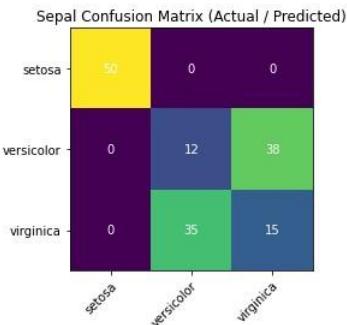


```
In [9]: sepal_accuracy = accuracy_score(y, sepal_labels)
petal_accuracy = accuracy_score(y, petal_labels)

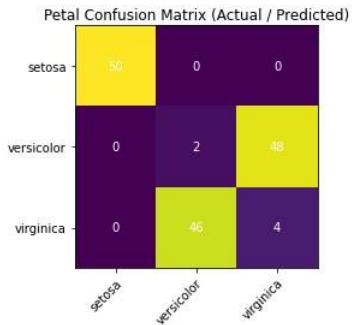
sepal_confusion = confusion_matrix(y, sepal_labels)
petal_confusion = confusion_matrix(y, petal_labels)
```

```
In [10]: plot_confusion(sepal_accuracy, sepal_confusion, 'Sepal')
plot_confusion(petal_accuracy, petal_confusion, 'Petal')
```

Sepal Accuracy: 0.5133333333333333



Petal Accuracy: 0.37333333333333335



## ALGORITHM:

7/6/03 LAB-08

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

def kmeans (X, K, max_iter=100):
    centroids = X [np.random.choice (size = K, replace=False)]
    for _ in range (max_iter):
        clusters = [[ ] for _ in range (K)]
        for x in X:
            distance = (np.linalg.norm (x - centroids))
            for centroid in centroids:
                distance = np.argmin (distance)
                clusters [distance].append (x)
        new_centroids = [ ]
        for cluster in clusters:
            if cluster:
                new_centroids.append (np.mean (cluster, axis=0))
            else:
                new_centroids.append (centroids [clusters.index (cluster)])
        if np.allclose (centroids, new_centroids):
            break
    return
```

return centroids, clusters

```
data = pd.read_csv('Iloggit/input/k-means-data1  
data.csv')
```

X = data.values

k = 3

centroids, clusters = kmeans(X, k)

centroids = np.array(centroids)

clusters = ['r', 'g', 'b']

for i, cluster in enumerate(clusters):

for point in cluster:

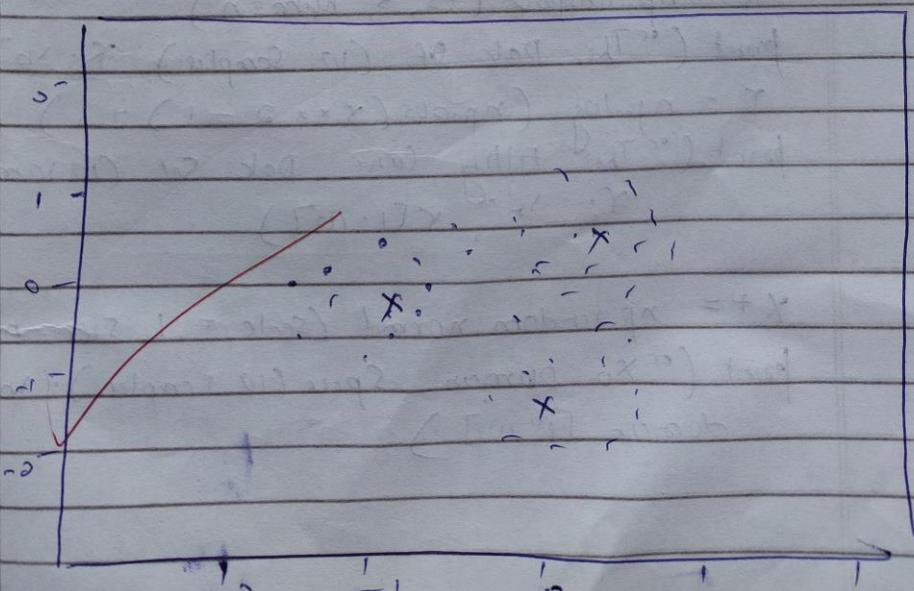
plt.scatter(point[0], point[1], c=clusters[i])

plt.scatter(centroids[0, 0], centroids[1, 0], c='k', marker='x')

plt.show()

Output:-

7/6/23



## 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.

```
In [1]: import matplotlib.pyplot as plt
import pandas as pd
from sklearn.mixture import GaussianMixture
from sklearn.metrics import accuracy_score, confusion_matrix
```

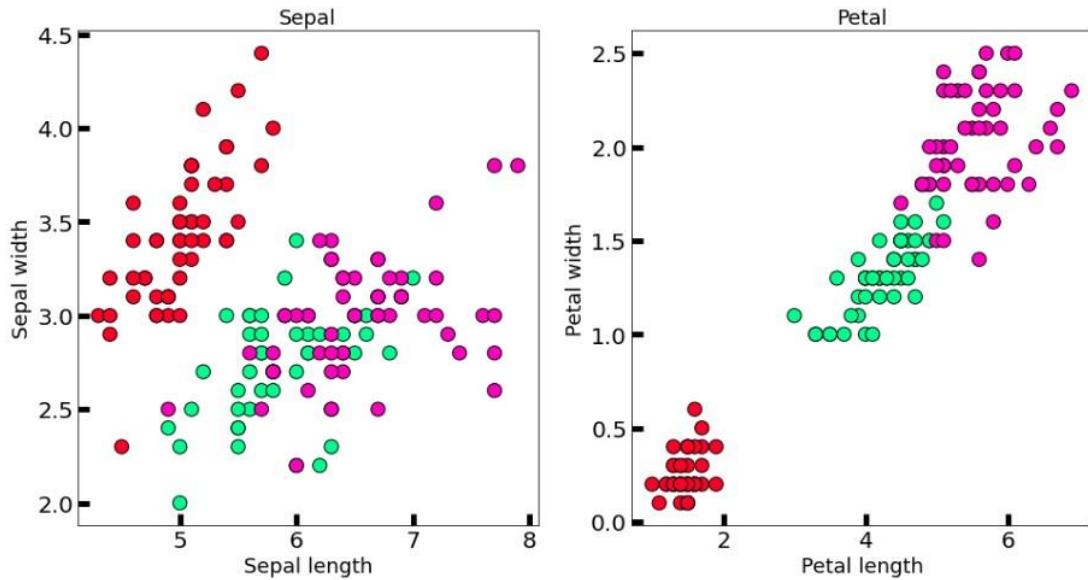
### Data

```
In [2]: data = pd.read_csv('../input/iris/Iris.csv')
data.head()
```

	<b>Id</b>	<b>SepalLengthCm</b>	<b>SepalWidthCm</b>	<b>PetalLengthCm</b>	<b>PetalWidthCm</b>	<b>Species</b>
<b>0</b>	1	5.1	3.5	1.4	0.2	Iris-setosa
<b>1</b>	2	4.9	3.0	1.4	0.2	Iris-setosa
<b>2</b>	3	4.7	3.2	1.3	0.2	Iris-setosa
<b>3</b>	4	4.6	3.1	1.5	0.2	Iris-setosa
<b>4</b>	5	5.0	3.6	1.4	0.2	Iris-setosa

```
In [3]: sepal_X = data[['SepalLengthCm', 'SepalWidthCm']]
petal_X = data[['PetalLengthCm', 'PetalWidthCm']]
y = data['Species'].astype("category").cat.codes
num_cat = y.nunique()
categories = data['Species'].astype("category").cat.categories
```

```
In [4]: fig, axes = plt.subplots(1, 2, figsize=(16,8))
axes[0].scatter(sepal_X.SepalLengthCm, sepal_X.SepalWidthCm, c=y, cmap='gist_rainbow', edgecolor='k', s=150)
axes[1].scatter(petal_X.PetalLengthCm, petal_X.PetalWidthCm, c=y, cmap='gist_rainbow', edgecolor='k', s=150)
axes[0].set_xlabel('Sepal length', fontsize=18)
axes[0].set_ylabel('Sepal width', fontsize=18)
axes[1].set_xlabel('Petal length', fontsize=18)
axes[1].set_ylabel('Petal width', fontsize=18)
axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
axes[0].set_title('Sepal', fontsize=18)
axes[1].set_title('Petal', fontsize=18)
plt.show()
```



## Model

```
In [5]: model_sepel = GaussianMixture(n_components=3)
sepel_labels = model_sepel.fit_predict(sepel_X)
model_petal = GaussianMixture(n_components=3)
petal_labels = model_petal.fit_predict(petal_X)
```

## Analysis

```
In [6]: def plot_actualvpredicted(X, y, predicted, part):
    fig, axes = plt.subplots(1, 2, figsize=(16,8))
    axes[0].scatter(X[f'{part}LengthCm'], X[f'{part}WidthCm'], c=y, cmap='gist_rainbow', edgecolor='k', s=150)
    axes[1].scatter(X[f'{part}LengthCm'], X[f'{part}WidthCm'], c=predicted, cmap='jet', edgecolor='k', s=150)
    axes[0].set_xlabel(f'{part} length', fontsize=18)
    axes[0].set_ylabel(f'{part} width', fontsize=18)
    axes[1].set_xlabel(f'{part} length', fontsize=18)
    axes[1].set_ylabel(f'{part} width', fontsize=18)
    axes[0].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
    axes[1].tick_params(direction='in', length=10, width=5, colors='k', labelsize=20)
    axes[0].set_title('Actual', fontsize=18)
    axes[1].set_title('Predicted', fontsize=18)
    plt.show()
```

```

def plot_confusion(accuracy, confusion, part):
    print(f'{part} Accuracy: {accuracy}')

    fig, ax = plt.subplots()
    im = ax.imshow(confusion)

    ax.set_xticks(range(num_cat))
    ax.set_yticks(range(num_cat))
    ax.set_xticklabels(categories)
    ax.set_yticklabels(categories)

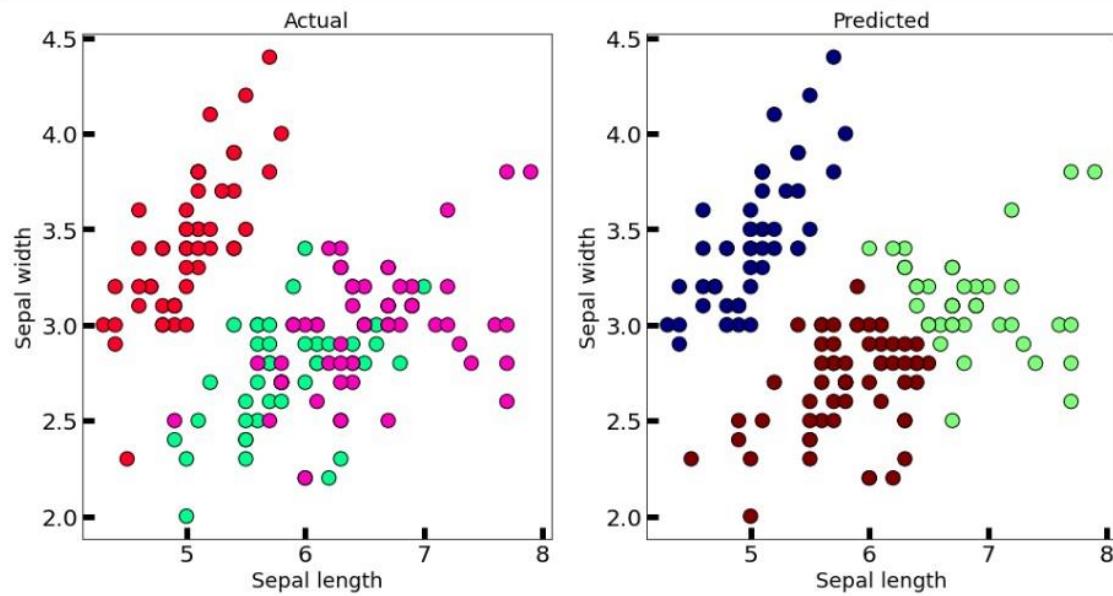
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")

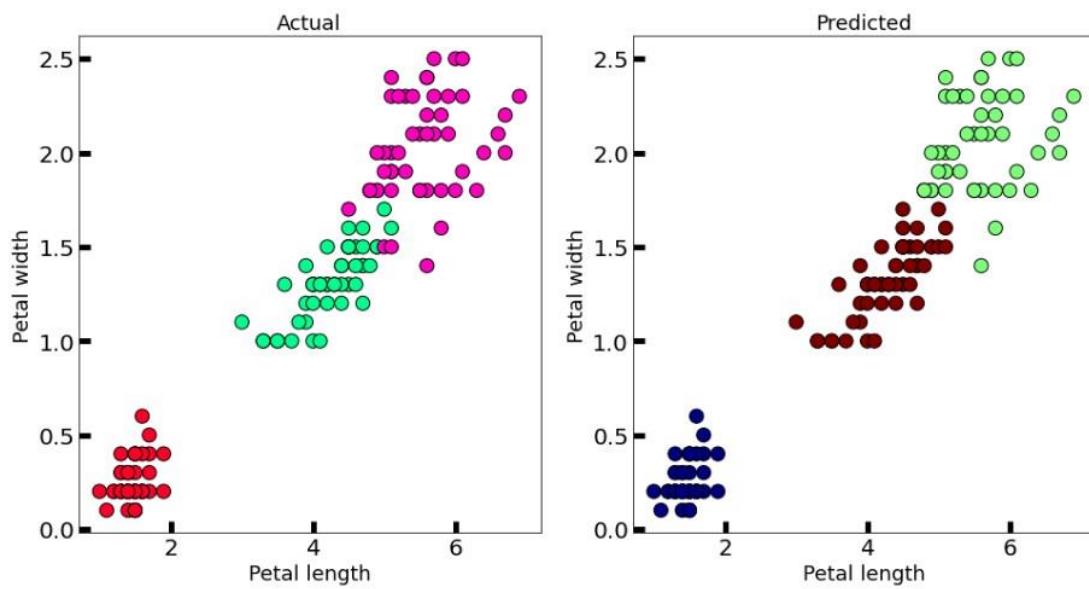
    for i in range(num_cat):
        for j in range(num_cat):
            text = ax.text(j, i, confusion[i, j],
                           ha="center", va="center", color="w")

    ax.set_title(f'{part} Confusion Matrix (Actual / Predicted)')
    fig.tight_layout()
    plt.show()

```

In [7]: `plot_actualvpredicted(sepal_X, y, sepal_labels, 'Sepal')  
plot_actualvpredicted(petal_X, y, petal_labels, 'Petal')`



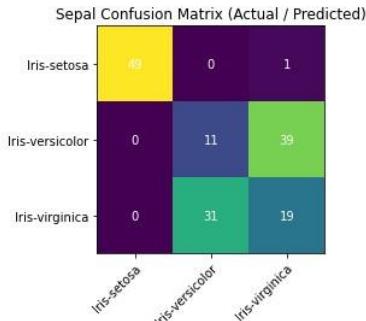


```
In [8]: sepal_accuracy = accuracy_score(y, sepal_labels)
petal_accuracy = accuracy_score(y, petal_labels)

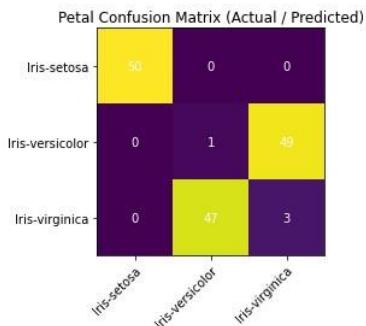
sepal_confusion = confusion_matrix(y, sepal_labels)
petal_confusion = confusion_matrix(y, petal_labels)
```

```
In [9]: plot_confusion(sepal_accuracy, sepal_confusion, 'Sepal')
plot_confusion(petal_accuracy, petal_confusion, 'Petal')
```

Sepal Accuracy: 0.5266666666666666



Petal Accuracy: 0.36



## Comparison with K-Means

When we compare the results of the GaussianMixture model (uses EM algorithm) with that of [K-Means clustering](#) we observe that both give almost equal accuracies:

- K-Means: 0.5133333333333333 (Sepal) & 0.3733333333333335 (Petal)
- GaussianMixture: 0.5333333333333333 (Sepal) & 0.02 (Petal)

GaussianMixture performs slightly better when classified based on the *Sepal* length & width. Likewise K-Means performs slightly better when classified based on the *Petal* length & width.

```
In [ ]:
```

## **ALGORITHM:**

14/6/23

CAB-10

```
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 = ['Target']
```

```
model = KMeans(n_clusters=3)
```

```
model.fit(X)
```

~~```
plt.figure(figsize=(14,7))
```~~~~```
clormaps = np.array(['red', 'teal', 'black'])
```~~

```
plt.subplot(1,2,1)
```

```
plt.scatter(X.Petal_Length, X.Petal_Width, c=clormaps[y.Target], s=100)
```

```
plt.title('Real Classification')
```

```
plt.xlabel('Petal Length')
```

```
plt.ylabel('Petal Width')
```

```
plt.subplot(1,2,2)
```

print("The confusion matrix of EM1: ",  
sm.confusion\_matrix(y, y\_gmm))

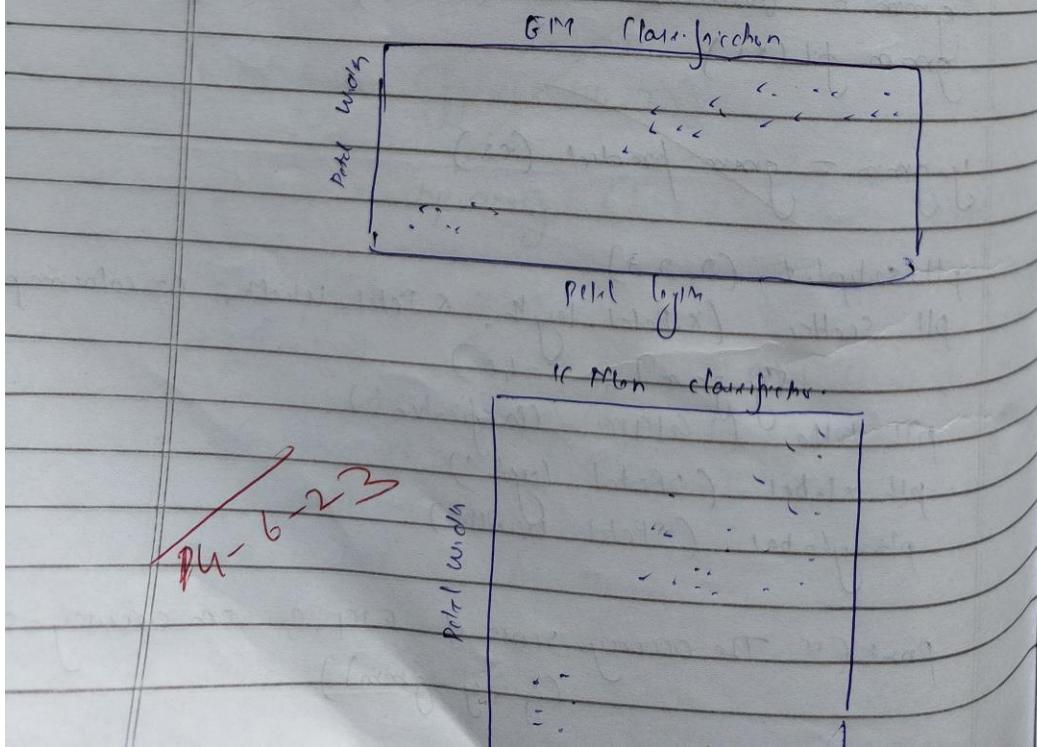
Output:

The accuracy score of k-Means = 0.893733

The confusion matrix of k-Means =  $\begin{bmatrix} 50 & 0 & 0 \\ 0 & 48 & 0 \\ 0 & 14 & 36 \end{bmatrix}$

The accuracy score of EM1: 0.3533333333333333

The confusion matrix of EM1:  $\begin{bmatrix} 50 & 0 & 4 \\ 0 & 48 & 0 \\ 0 & 0 & 0 \end{bmatrix}$



**Program 8:** Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

```
In [1]: import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
```

## Data

```
In [2]: data = pd.read_csv('../input/Iris/Iris.csv')
```

```
In [3]: data.head()
```

```
Out[3]:
```

|          | <b>Id</b> | <b>SepalLengthCm</b> | <b>SepalWidthCm</b> | <b>PetalLengthCm</b> | <b>PetalWidthCm</b> | <b>Species</b> |
|----------|-----------|----------------------|---------------------|----------------------|---------------------|----------------|
| <b>0</b> | 1         | 5.1                  | 3.5                 | 1.4                  | 0.2                 | Iris-setosa    |
| <b>1</b> | 2         | 4.9                  | 3.0                 | 1.4                  | 0.2                 | Iris-setosa    |
| <b>2</b> | 3         | 4.7                  | 3.2                 | 1.3                  | 0.2                 | Iris-setosa    |
| <b>3</b> | 4         | 4.6                  | 3.1                 | 1.5                  | 0.2                 | Iris-setosa    |
| <b>4</b> | 5         | 5.0                  | 3.6                 | 1.4                  | 0.2                 | Iris-setosa    |

```
In [4]: data.describe()
```

```
Out[4]:
```

|              | <b>Id</b>  | <b>SepalLengthCm</b> | <b>SepalWidthCm</b> | <b>PetalLengthCm</b> | <b>PetalWidthCm</b> |
|--------------|------------|----------------------|---------------------|----------------------|---------------------|
| <b>count</b> | 150.000000 | 150.000000           | 150.000000          | 150.000000           | 150.000000          |
| <b>mean</b>  | 75.500000  | 5.843333             | 3.054000            | 3.758667             | 1.198667            |
| <b>std</b>   | 43.445368  | 0.828066             | 0.433594            | 1.764420             | 0.763161            |
| <b>min</b>   | 1.000000   | 4.300000             | 2.000000            | 1.000000             | 0.100000            |
| <b>25%</b>   | 38.250000  | 5.100000             | 2.800000            | 1.600000             | 0.300000            |
| <b>50%</b>   | 75.500000  | 5.800000             | 3.000000            | 4.350000             | 1.300000            |
| <b>75%</b>   | 112.750000 | 6.400000             | 3.300000            | 5.100000             | 1.800000            |
| <b>max</b>   | 150.000000 | 7.900000             | 4.400000            | 6.900000             | 2.500000            |

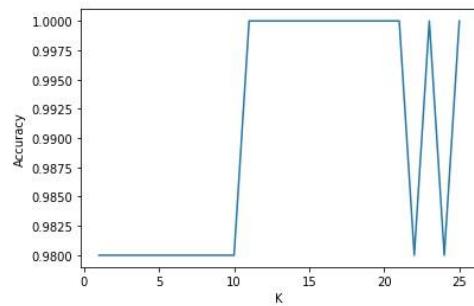
```
In [5]: X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
sepal_X = data[['SepalLengthCm', 'SepalWidthCm']]
petal_X = data[['PetalLengthCm', 'PetalWidthCm']]
y = data['Species'].astype("category").cat.codes
num_cat = y.unique()
categories = data['Species'].astype("category").cat.categories
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

## Model

```
In [6]: scores = []
scores_list = []
for k in range(1, 26):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores[k] = accuracy_score(y_test, y_pred)
    scores_list.append(scores[k])
```

## Analysis

```
In [7]: plt.plot(range(1, 26), scores_list)
plt.xlabel('K')
plt.ylabel('Accuracy')
plt.show()
```



```
In [8]: knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
print(f"Correct Predictions: {accuracy_score(y_test, y_pred)}")
print(f"Wrong Predictions: {1 - accuracy_score(y_test, y_pred)}")
print("Accuracy Metrics:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix
[[19  0  0]
 [ 0 15  0]
 [ 0  1 15]]
Correct Predictions: 0.98
Wrong Predictions: 0.02000000000000018
Accuracy Metrics:
      precision    recall  f1-score   support
          0         1.00     1.00     1.00      19
          1         0.94     1.00     0.97      15
          2         1.00     0.94     0.97      16

   accuracy                           0.98      50
  macro avg       0.98     0.98     0.98      50
weighted avg     0.98     0.98     0.98      50
```

```
In [ ]:
```

## ALGORITHM:

LAB-08

7/6/03

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

def kmeans(X, k, max_iter=100):
    centroids = X[np.random.choice(len(X), size=k, replace=False)]
    for i in range(max_iter):
        clusters = [[ ] for _ in range(k)]
        for x in X:
            distances = [np.linalg.norm(x - centroid) for centroid in centroids]
            cluster_index = np.argmin(distances)
            clusters[cluster_index].append(x)
        new_centroids = []
        for cluster in clusters:
            if cluster:
                new_centroids.append(np.mean(cluster, axis=0))
            else:
                new_centroids.append(centroids[cluster_index])
        if np.allclose(centroids, new_centroids):
            break
    return clusters
```

redu

data = p

X = data

k = 3

centroids

centroids

clusters

for i, j

plt.scatter

plt.show

Output:-

## Program 9: Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
In [1]: import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

### Data

```
In [2]: df = pd.read_csv('../input/housesalesprediction/kc_house_data.csv')
```

```
In [3]: df.head()
```

```
Out[3]:
```

|   | id         | date            | price    | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view | ... | grade | sqft_above |
|---|------------|-----------------|----------|----------|-----------|-------------|----------|--------|------------|------|-----|-------|------------|
| 0 | 7129300520 | 20141013T000000 | 221900.0 | 3        | 1.00      | 1180        | 5650     | 1.0    | 0          | 0    | ... | 7     | 1180       |
| 1 | 6414100192 | 20141209T000000 | 538000.0 | 3        | 2.25      | 2570        | 7242     | 2.0    | 0          | 0    | ... | 7     | 2170       |
| 2 | 5631500400 | 20150225T000000 | 180000.0 | 2        | 1.00      | 770         | 10000    | 1.0    | 0          | 0    | ... | 6     | 770        |
| 3 | 2487200875 | 20141209T000000 | 604000.0 | 4        | 3.00      | 1960        | 5000     | 1.0    | 0          | 0    | ... | 7     | 1050       |
| 4 | 1954400510 | 20150218T000000 | 510000.0 | 3        | 2.00      | 1680        | 8080     | 1.0    | 0          | 0    | ... | 8     | 1680       |

5 rows × 21 columns

```
In [4]: X = np.array(df['sqft_living']).reshape(-1, 1)
y = df['price']
```

```
In [5]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

### Model

```
In [6]: model = LinearRegression()
model.fit(X_train, y_train)
# model.fit(X, y)
```

```
Out[6]: LinearRegression()
```

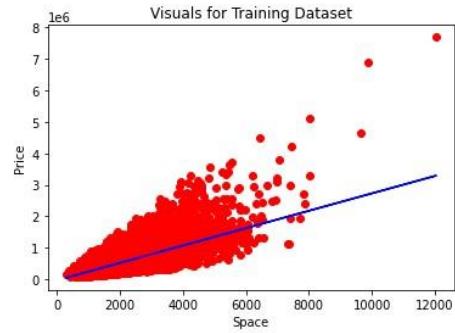
### Analysis

```
In [7]: y_pred = model.predict(X_test)
print(f"Mean Squared Error: {mean_squared_error(y_pred, y_test)}")
```

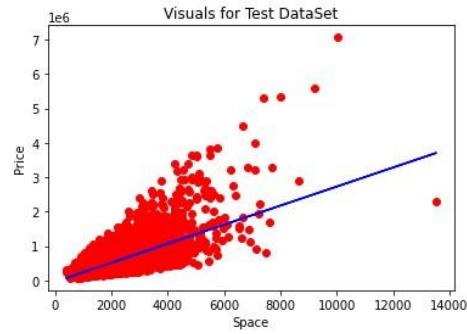
Mean Squared Error: 76715223988.35832

```
In [8]: #Visualizing the training Test Results
plt.scatter(X_train, y_train, color='red')
```

```
In [8]: #Visualizing the training Test Results
plt.scatter(X_train, y_train, color= 'red')
plt.plot(X_train, model.predict(X_train), color = 'blue')
plt.title ("Visuals for Training Dataset")
plt.xlabel("Space")
plt.ylabel("Price")
plt.show()
```



```
In [9]: #Visualizing the Test Results
plt.scatter(X_test, y_test, color= 'red')
plt.plot(X_test, model.predict(X_test), color = 'blue')
plt.title("Visuals for Test DataSet")
plt.xlabel("Space")
plt.ylabel("Price")
plt.show()
```



```
In [ ]:
```

## ALGORITHM:

LHB-06

### Linear Regression:-

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

$x = np.linspace(0, 100, 50)$

$y = x + np.random.normal(2*x + 3, 20)$

plt.xlim = 200

plt.ylim = 200

plt.xlabel = "x"

plt.ylabel = "y"

plt.scatter(x, y)

$X = x$

$Y = y$

$x\_mean = np.mean(X)$

$y\_mean = np.mean(Y)$

len(X)

numerator = 0

denominator = 0

for i in range(1):

numerator += [(x[i] - x\_mean) \* (y[i] - y\_mean)]

denominator += (x[i] - x\_mean) \*\* 2

$b_1 = \frac{\text{numerator}}{\text{denominator}}$

$b_0 = y\_mean - (b_1 * x\_mean)$

print(b1, b0)

$\rightarrow 3.1456 - 9.2805$

$$y - \text{prod} = 60 + 61 * 150$$

print (° for x=150 y=° in)

print (int(y - prod))

→ for x=150

$$y = 1,62$$

**Program 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.

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

## Functions

```
In [2]: # kernel smoothing function
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

# function to return local weight of each training example
def localWeight(point, xmat, ymat, k):
    wt = kernel(point, xmat, k)
    W = (X.T * (wt*X)).I * (X.T * wt * ymat.T)
    return W

# root function that drives the algorithm
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
```

## Data

```
In [3]: #import data
data = pd.read_csv('../input/tipsdata/tips.csv')

# place them in suitable data types
colA = np.array(data.total_bill)
colB = np.array(data.tip)

mcolA = np.mat(colA)
mcolB = np.mat(colB)

m = np.shape(mcolB)[1]
one = np.ones((1, m), dtype = int)

# horizontal stacking
X = np.hstack((one.T, mcolA.T))
print(X.shape)
```

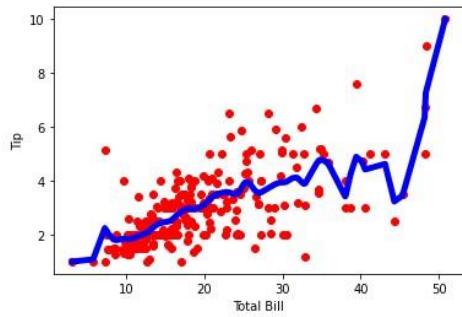
(244, 2)

## Model

```
In [4]: # predicting values using LWLR  
yPred = localWeightRegression(X, mcolB, 0.8)
```

## Analysis

```
In [5]: # plotting the predicted graph  
xsort = X.copy()  
xsort.sort(axis=0)  
plt.scatter(colA, colB, color='red')  
plt.plot(xsort[:, 1], yPred[X[:, 1].argsort(0)], color='blue', linewidth=5)  
plt.xlabel('Total Bill')  
plt.ylabel('Tip')  
plt.show()
```



```
In [ ]:
```

## ALGORITHM:

14/6/23

LAB - 09

```
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, tau):
```

```
x0 = np.r_[1, x0]
```

```
x = np.c_[np.ones(len(x)), x]
```

```
xw = x.T * radial_kernel (x0, x, tau)
```

```
beta = np.linalg.pinv (xw @ x) @ xw @ y
```

```
return x0 @ beta
```

```
def radial_kernel (x0, x, tau):
```

```
return np.exp (-np.sum ((x - x0)**2, axis=1))
```

```
/ (2 * tau * tau))
```

n=1000

```
x = np.linspace (-3, 3, num=n)
```

```
print ("The Data Set (10 samples) is: ", x[1:10])
```

```
T = np.log (np.abs (x**2 - 1) + 1)
```

```
print ("The Fitting Curve Set (10 samples) is: ", T[1:10])
```

```
x_+ = np.random.normal (scale=.1, size=n)
```

```
point ("x0 domain Space (10 samples): ", "
```

```
domain [1:10] )
```

def plot\_lwr (tau):

    prediction = [load regression (x0, x, r, tau)  
    for x0 in domain])]

    plot = figure (plot\_width = 1000, plot\_height = 600)

    plot.title.text = 'tau={:.g} of tau'

    plot.scatter (X, Y, alpha = .3)

    plot.line (domain, prediction, line\_width = 2,  
    color = "#red")

    return plot

show (gridplot ([plot\_lwr (10.1), plot\_lwr (1.1),  
[plot\_lwr (0.1), plot\_lwr (0.01)]]))

Output:-

The Data Set (10 samples) x:

[ -2.99399399 -2.98798799 -2.98198198  
-2.97597998 -2.96 -2.96 -2.95  
-2.95 -2.94 ]

The Fitted Curve Data Set (10 samples) r:

[ 2.13 2.13 2.13 2.12 2.118 2.116 2.11  
2.105 2.101 ]

Normalised (10 samples) r:

[ -3.04 -3.05 -2.98 -2.96 -2.97 -2.97  
-2.93 -2.05 2.95 ]

X0 Domain Samples (10 samples):

[ -2.97 -2.95 -2.93 -2.91 -2.89 2.90  
-2.85 -2.83 -2.81 ]