

Practical-1

Name: Naik Onkar Chandrakant

Reg No : 2019BEC135

Roll No : A65

AIM:- Write a python program to handle the “Numpy” and “Matplotlib”

library.REQUIRED SOFTWARE :- Google Colab , Pycharm

THEORY :-

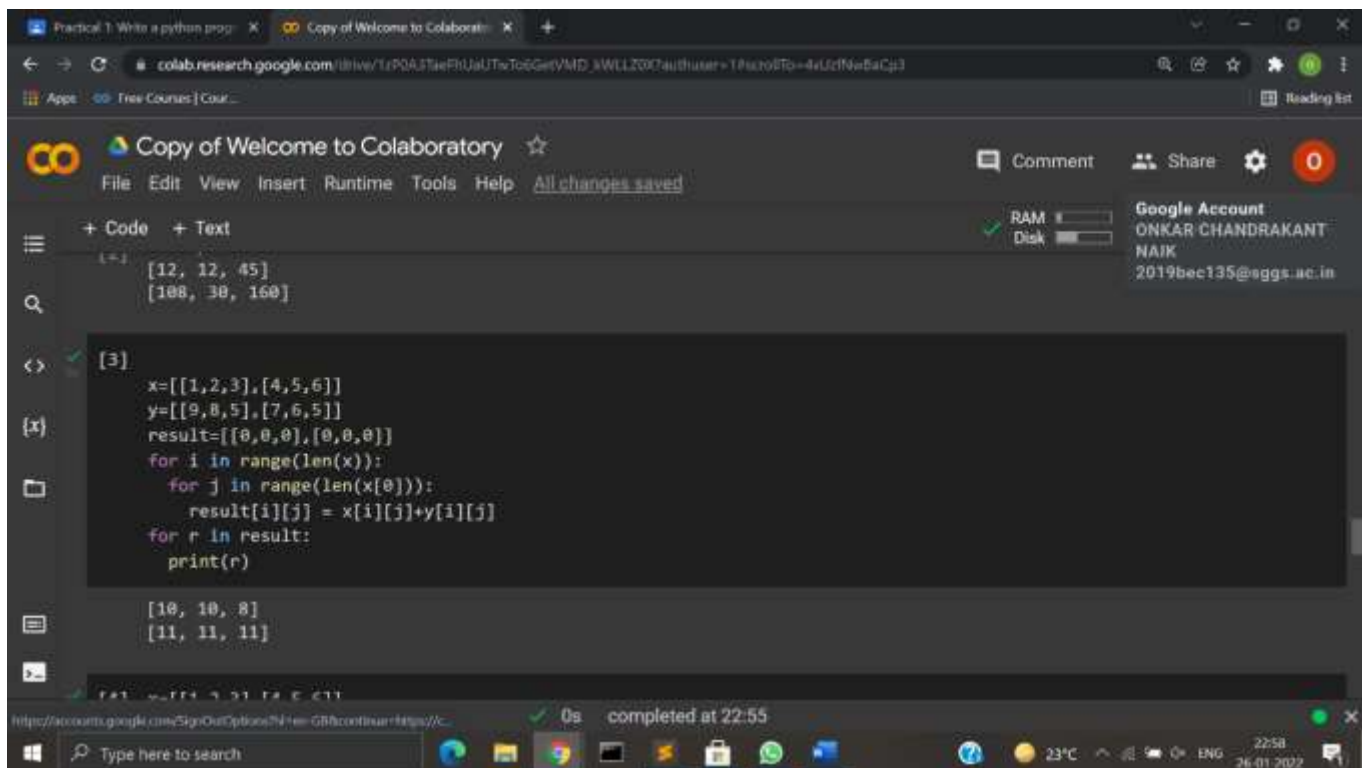
NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software. It contains various features including these important ones:

Matplotlib is a plotting library available for the Python programming language as a component of NumPy, a big data numerical handling resource. Matplotlib uses an object oriented API to embed plots in Python applications. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

1. Define the multi-dimensional arrays (2x2 and 3x3) using Numpy library and perform various operations on it (addition, subtraction, element-wise multiplication, matrix multiplication) without inbuilt function. Print the respective outputs.

Addition:



The screenshot shows the Google Colaboratory interface. The browser address bar displays the URL: `colab.research.google.com/drive/1xPOA3TaePhUaUTwTo6GetVMD_3WLLZ0X7authuser=1Pach06To+4xUdNwBaCp3`. The Colab logo and title "Copy of Welcome to Colaboratory" are at the top. The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. The left sidebar shows icons for Code, Text, and a search icon. The main editor area contains the following Python code:

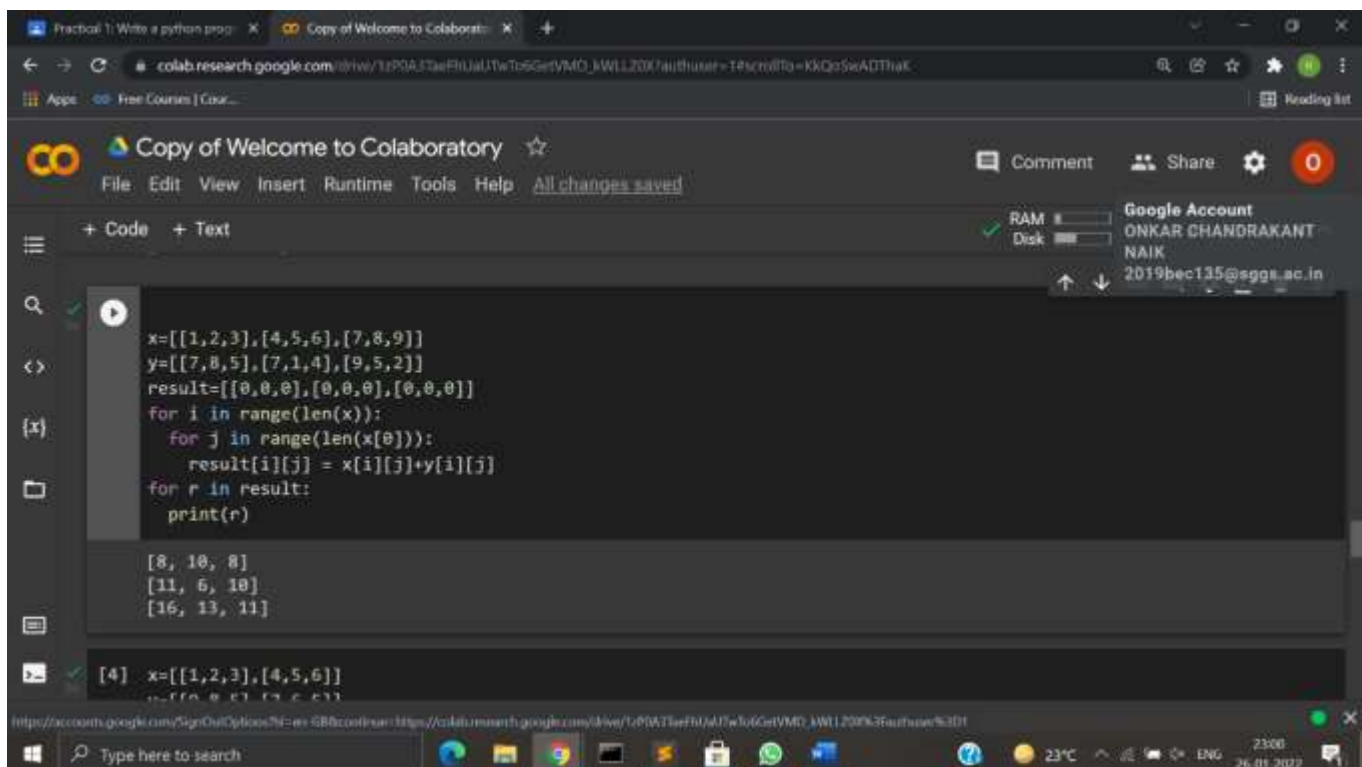
```
[12, 12, 45]
[108, 30, 160]

[3]
x=[[1,2,3],[4,5,6]]
y=[[9,8,5],[7,6,5]]
result=[[0,0,0],[0,0,0]]
for i in range(len(x)):
    for j in range(len(x[0])):
        result[i][j] = x[i][j]+y[i][j]
for r in result:
    print(r)
```

The output of the code is displayed below the editor:

```
[10, 10, 8]
[11, 11, 11]
```

The status bar at the bottom indicates "0s completed at 22:55". The system tray shows the date and time as 22:58 on 26-01-2022.



The screenshot shows the Google Colaboratory interface. The browser address bar displays the URL: `colab.research.google.com/drive/1xPOA3TaePhUaUTwTo6GetVMD_3WLLZ0X7authuser=14scv0lTo+KkQo$wADThak`. The Colab logo and title "Copy of Welcome to Colaboratory" are at the top. The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. The left sidebar shows icons for Code, Text, and a search icon. The main editor area contains the following Python code:

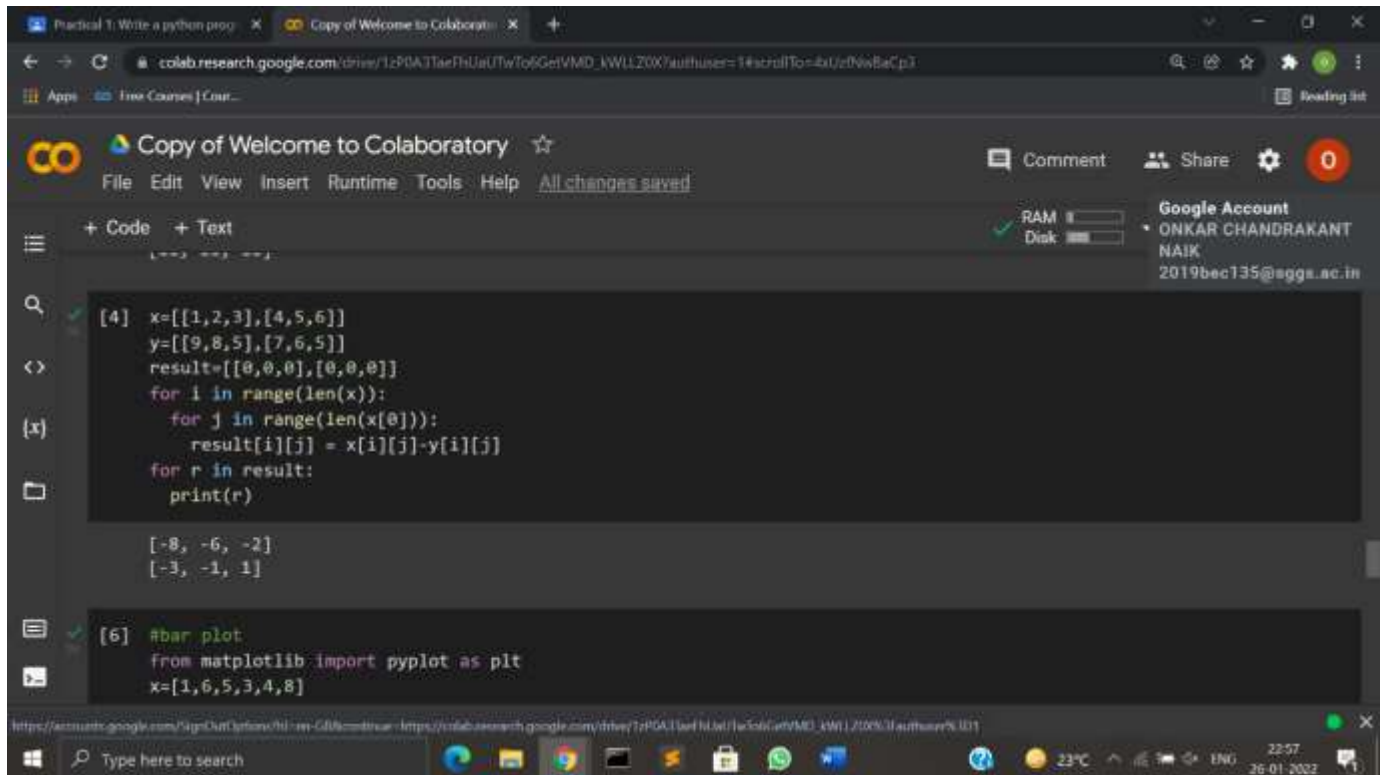
```
x=[[1,2,3],[4,5,6],[7,8,9]]
y=[[7,8,5],[7,1,4],[9,5,2]]
result=[[0,0,0],[0,0,0],[0,0,0]]
for i in range(len(x)):
    for j in range(len(x[0])):
        result[i][j] = x[i][j]+y[i][j]
for r in result:
    print(r)
```

The output of the code is displayed below the editor:

```
[8, 10, 8]
[11, 6, 10]
[16, 13, 11]
```

The status bar at the bottom indicates "0s completed at 22:55". The system tray shows the date and time as 23:00 on 26-01-2022.

Subtraction:

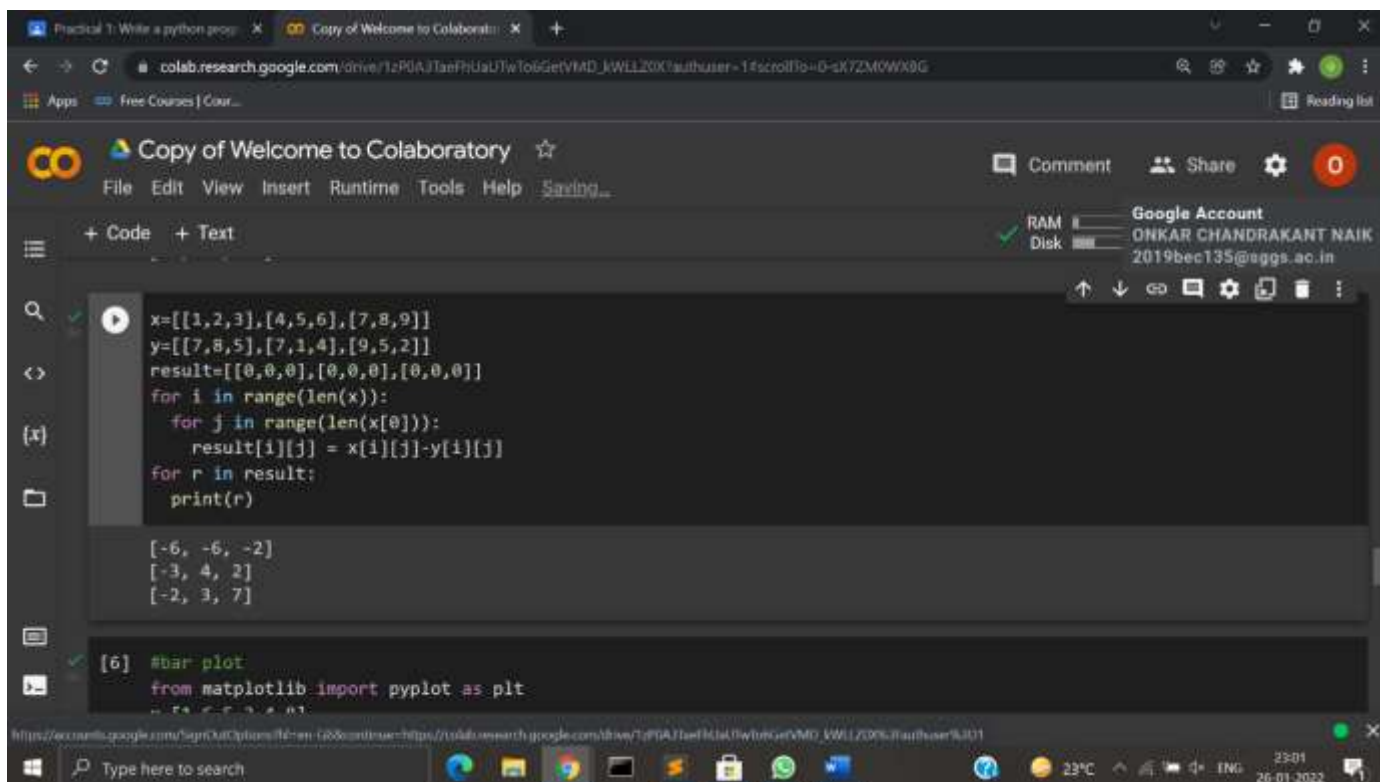


The screenshot shows a Google Colab environment with a notebook titled "Copy of Welcome to Colaboratory". The notebook contains two code cells. The first cell, labeled [4], defines two 2x2 matrices x and y, and performs element-wise subtraction. The second cell, labeled [6], uses matplotlib to plot the resulting matrix. The output of the first cell shows the resulting matrix as a list of lists.

```
[4] x=[[1,2,3],[4,5,6]]
    y=[[9,8,5],[7,6,5]]
    result=[[0,0,0],[0,0,0]]
    for i in range(len(x)):
        for j in range(len(x[0])):
            result[i][j] = x[i][j]-y[i][j]
    for r in result:
        print(r)

[-8, -6, -2]
[-3, -1, 1]
```

```
[6] #bar plot
    from matplotlib import pyplot as plt
    x=[1,6,5,3,4,8]
```



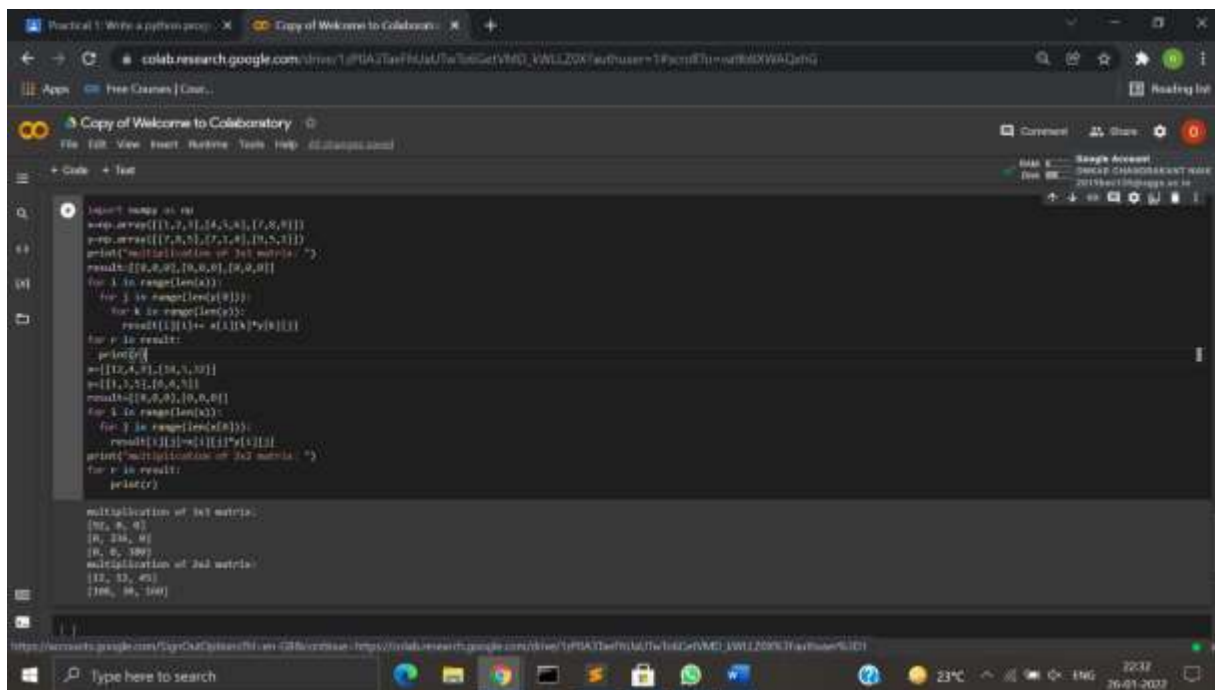
The screenshot shows a Google Colab environment with a notebook titled "Copy of Welcome to Colaboratory". The notebook contains two code cells. The first cell, labeled [4], defines two 3x3 matrices x and y, and performs element-wise subtraction. The second cell, labeled [6], uses matplotlib to plot the resulting matrix. The output of the first cell shows the resulting matrix as a list of lists.

```
[4] x=[[1,2,3],[4,5,6],[7,8,9]]
    y=[[7,8,5],[7,1,4],[9,5,2]]
    result=[[0,0,0],[0,0,0],[0,0,0]]
    for i in range(len(x)):
        for j in range(len(x[0])):
            result[i][j] = x[i][j]-y[i][j]
    for r in result:
        print(r)

[-6, -6, -2]
[-3, 4, 2]
[-2, 3, 7]
```

```
[6] #bar plot
    from matplotlib import pyplot as plt
    x=[1,6,5,3,4,8]
```

Multiplication:



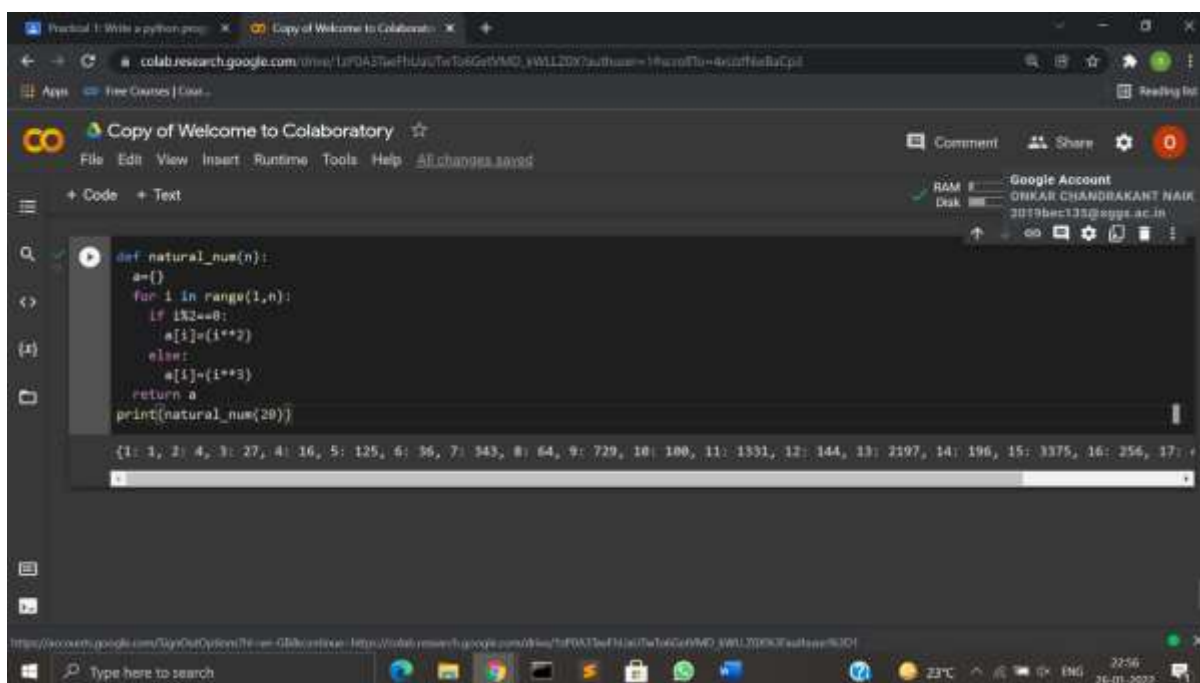
The screenshot shows a Google Colab notebook titled "Copy of Welcome to Colaboratory". The code defines two 3x3 matrices, multiplies them, and prints the result. The output shows the multiplication of two 3x3 matrices, resulting in a 3x3 matrix.

```
import numpy as np
x=np.array([[1,2,3],[4,5,6],[7,8,9]])
y=np.array([[7,8,9],[9,5,1]])
print("multiplication of 3x3 matrix: ")
result=np.zeros([3,3])
for i in range(len(x)):
    for j in range(len(y)):
        for k in range(len(x)):
            result[i][j]=x[i][k]*y[k][j]
for i in result:
    print(i)
x=[[1,2,3],[4,5,6],[7,8,9]]
y=[[7,8,9],[9,5,1]]
result=np.zeros([3,3])
for i in range(len(x)):
    for j in range(len(y)):
        result[i][j]=x[i][k]*y[k][j]
print("multiplication of 3x3 matrix: ")
for i in result:
    print(i)
```

Output:

```
multiplication of 3x3 matrix:
[18, 8, 6]
[18, 8, 30]
multiplication of 3x3 matrix:
[12, 13, 45]
[108, 96, 100]
```

2. Write a python code to generate dictionary which consist of first 20 natural numbers as a keys. The even number keys must have square of number as value while odd number keys must have cube of number as value. Print the dictionary.



The screenshot shows a Google Colab notebook titled "Copy of Welcome to Colaboratory". The code defines a function `natural_num(n)` that generates a dictionary of the first `n` natural numbers. The dictionary keys are natural numbers, and the values are the square of the number for even keys and the cube of the number for odd keys. The function is called with `n=20`, and the resulting dictionary is printed.

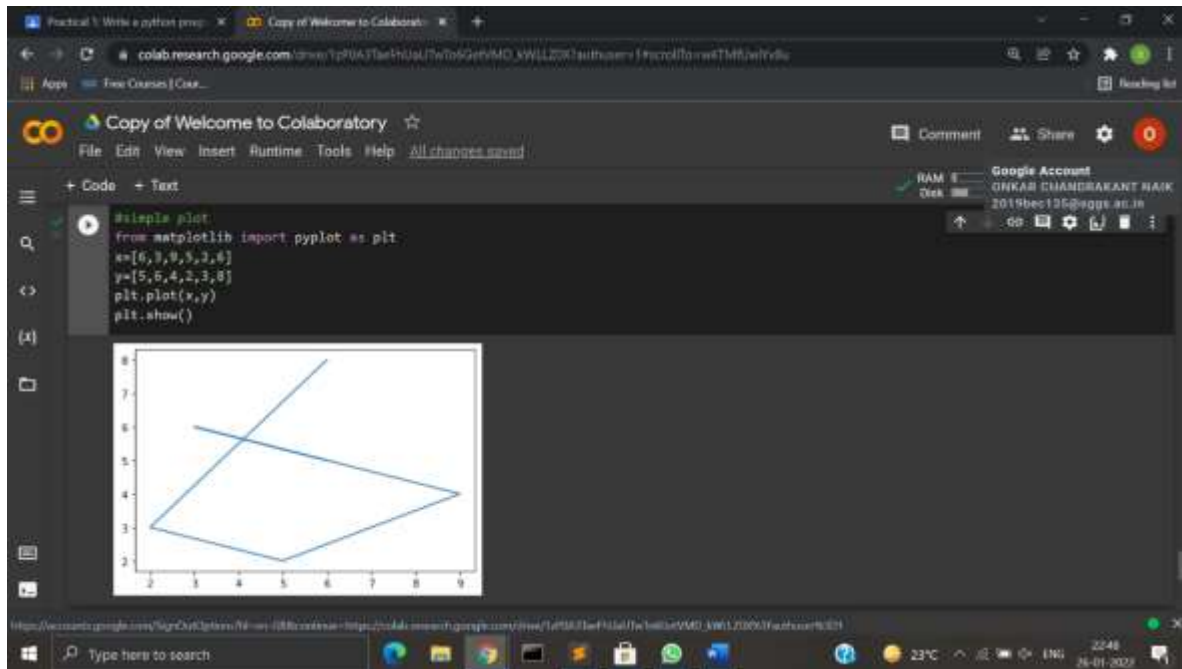
```
def natural_num(n):
    a={}
    for i in range(1,n):
        if i%2==0:
            a[i]=(i**2)
        else:
            a[i]=(i**3)
    return a
print(natural_num(20))
```

Output:

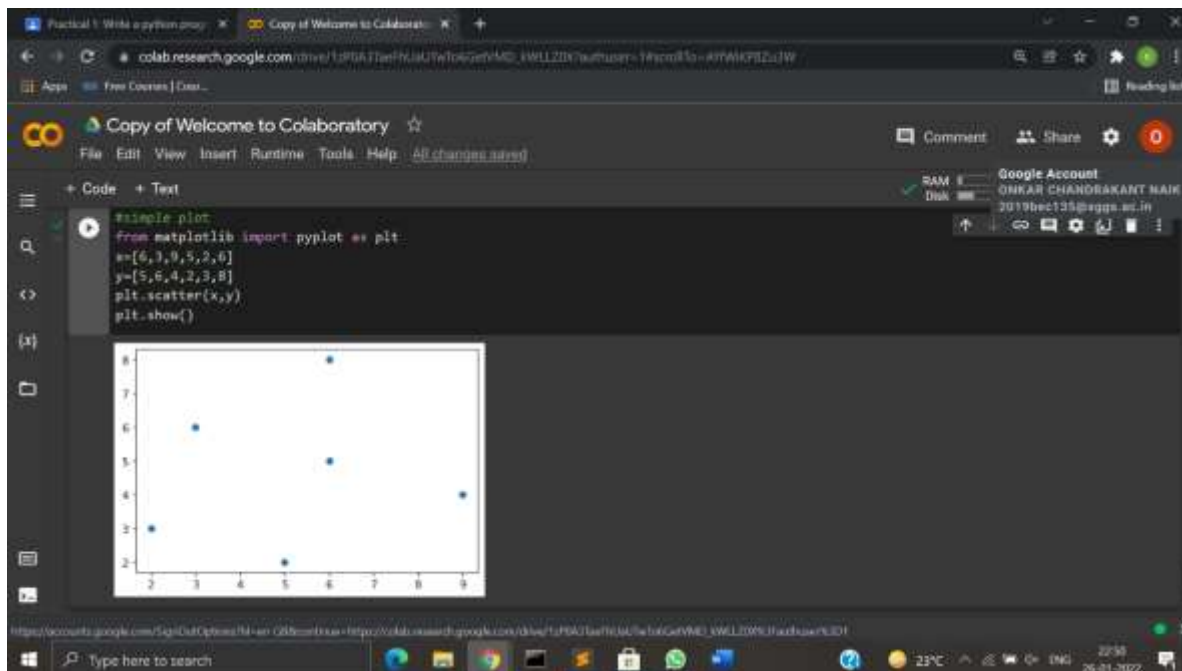
```
{1: 1, 2: 4, 3: 27, 4: 16, 5: 125, 6: 36, 7: 343, 8: 64, 9: 729, 10: 100, 11: 1331, 12: 144, 13: 2197, 14: 196, 15: 3375, 16: 256, 17: 4913, 18: 324, 19: 6859}
```

3. Define two variable having functional dependency. Plot the three graphs (simple plot, scatter plot and bar graph) between two variables using Matplotlib library.

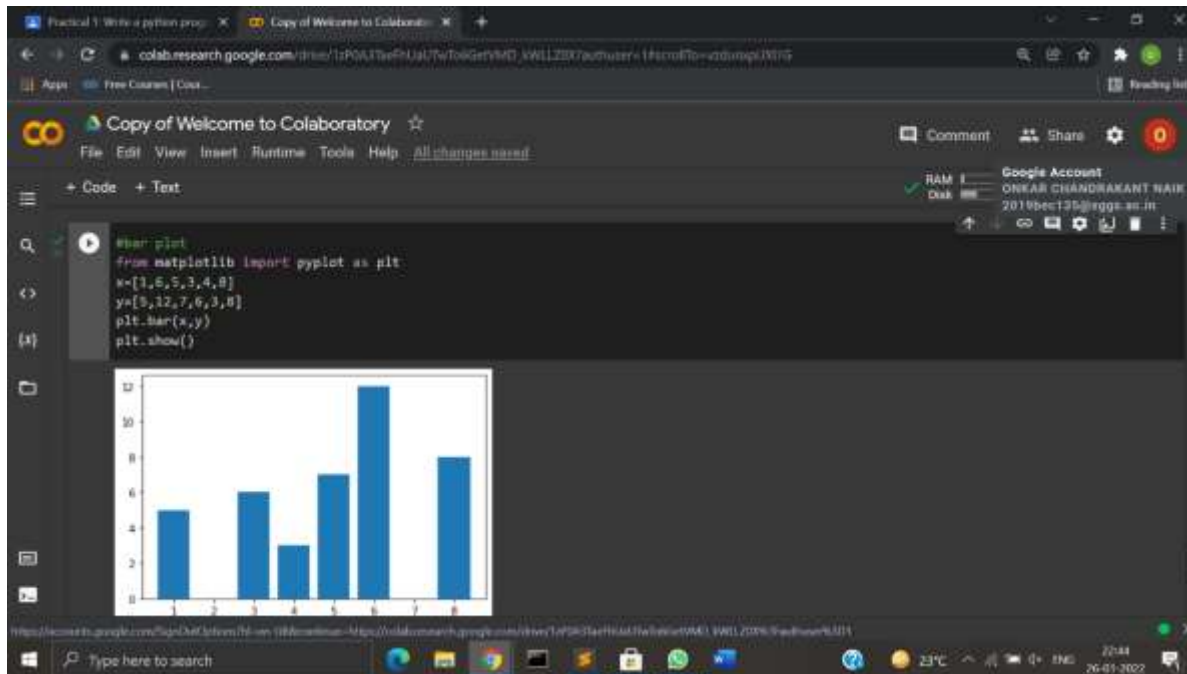
1) Simple Plot:



2) Scatter Plot:



3) Bar plot:



CONCLUSION : - Numpy is a basic Python package that provides an alternative to a regular Python list, a Numpy n-dimensional homogeneous array. A list is a very useful tool offered by Python, as it lets you store values of different types at once, and perform numerous operations on it while Matplotlib is the most powerful visualization library. To use it, you need to import its sub package pyplot.

Practical-2

Name: Naik Onkar Chandrakant

Reg No : 2019BEC135

Roll No : A65

AIM:- Write a python program to plot 2D & 3D Gaussian distribution curve using probability of likelihood formula.

REQUIRED SOFTWARE :- Google Colab ,

Pycharm THEORY :-

The Gaussian distribution is the healthy-studied probability distribution. It is for nonstop-valued random variables. It is as well stated as the normal distribution. Its position makes from the fact that it has many computationally suitable properties.

There are several parts of machine learning that takes advantage from using a Gaussian distribution. Those areas are including;

- Gaussian processes
- Variational inference
- Reinforcement learning

It is similarly broadly used in other application areas such as

- Signal processing such as Kalman filter
- Control, for example linear quadratic regulator
- Statistics, as hypothesis testing

Importance of Gaussian Distribution

It is ever-present as a dataset with finite variance turns into Gaussian as long as a dataset with free feature-probabilities is permitted to raise in size.

It is the most significant probability distribution in statistics as it turns many natural phenomena such as age, height, test-scores, IQ scores, and sum of the rolls of two cubes and so on.

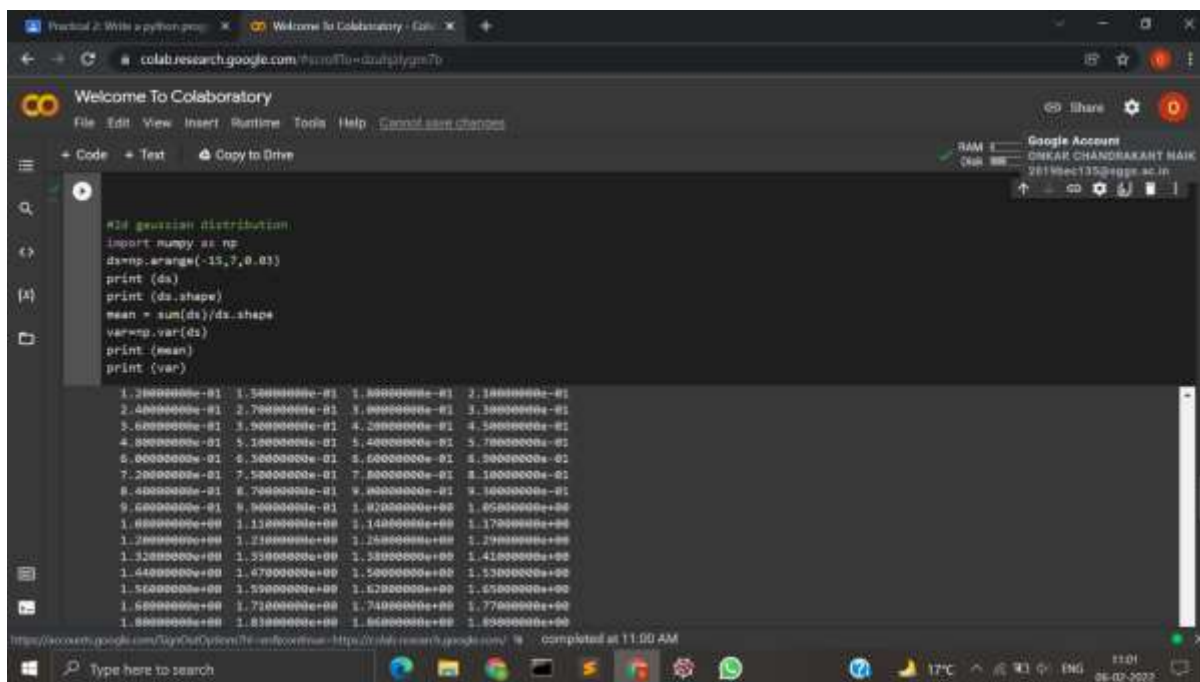
Datasets with Gaussian distributions create valid to a diversity of methods that

decrease under parametric statistics.

The approaches for example propagation of uncertainty and least squares parameter estimation are related only to datasets with normal or normal-like distributions.

Reviews and conclusions resulting from such analysis are intuitive. That also easy to explain to audiences with basic knowledge of statistics.

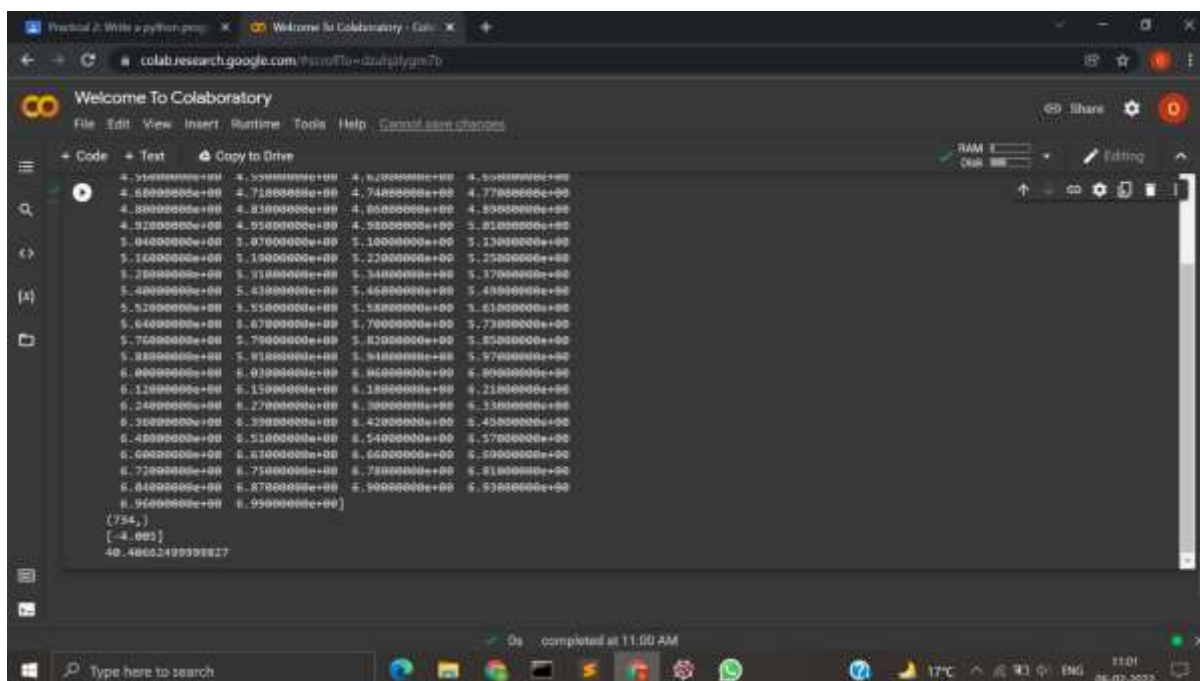
1. Define the data sample x in range (any) for 1D data. Decide mean & variance and print it



The screenshot shows a Google Colab notebook titled "Welcome To Colaboratory". The code cell contains the following Python code:

```
#1d gaussian distribution
import numpy as np
ds=np.arange(-15,7,0.03)
print(ds)
print(ds.shape)
mean = sum(ds)/ds.shape
var=np.var(ds)
print(mean)
print(var)
```

The output of the code is a long list of values representing a 1D Gaussian distribution sample. The values are printed in scientific notation, ranging from approximately -1.5×10^{-1} to 1.8×10^{-1} . The shape of the array is $(734,)$. The mean is -4.001 and the variance is 40.486249999817 .

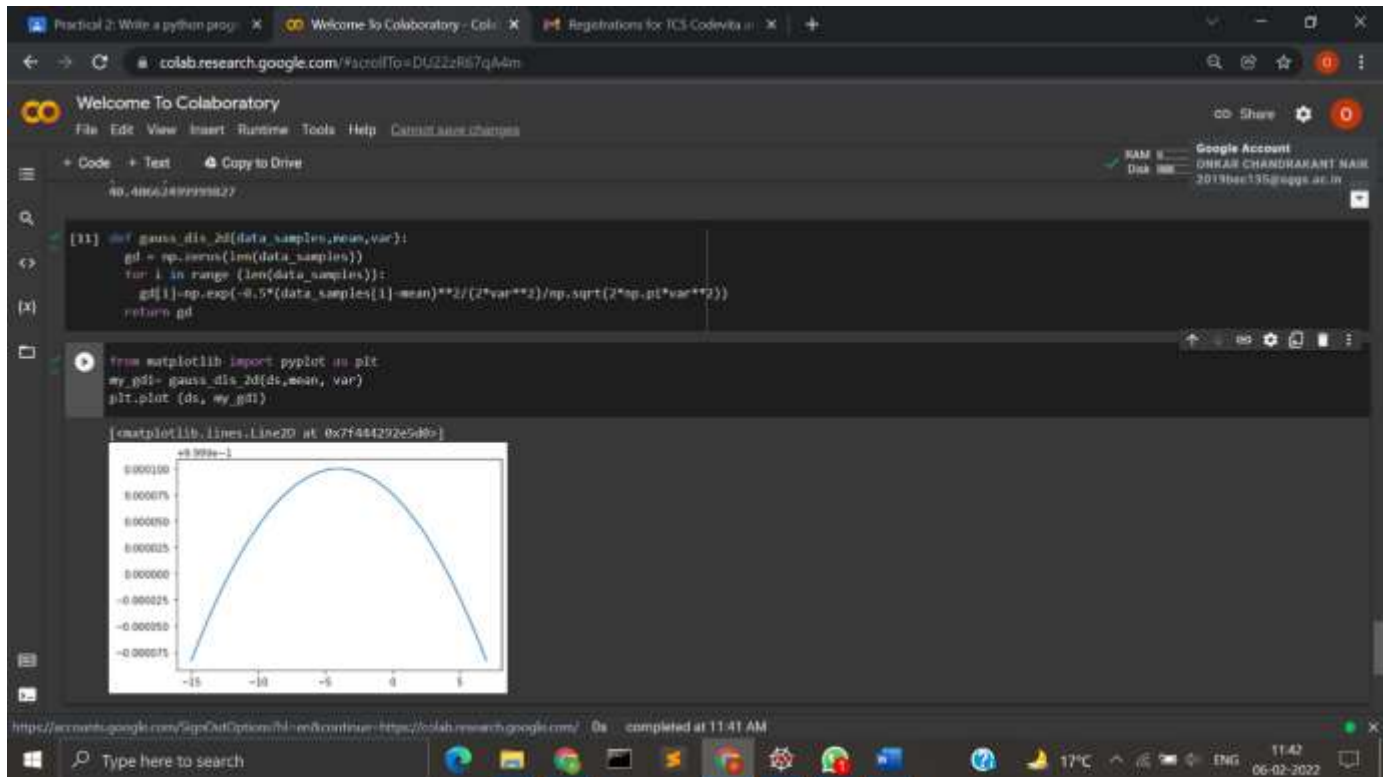


The screenshot shows a Google Colab notebook titled "Welcome To Colaboratory". The code cell contains the following Python code:

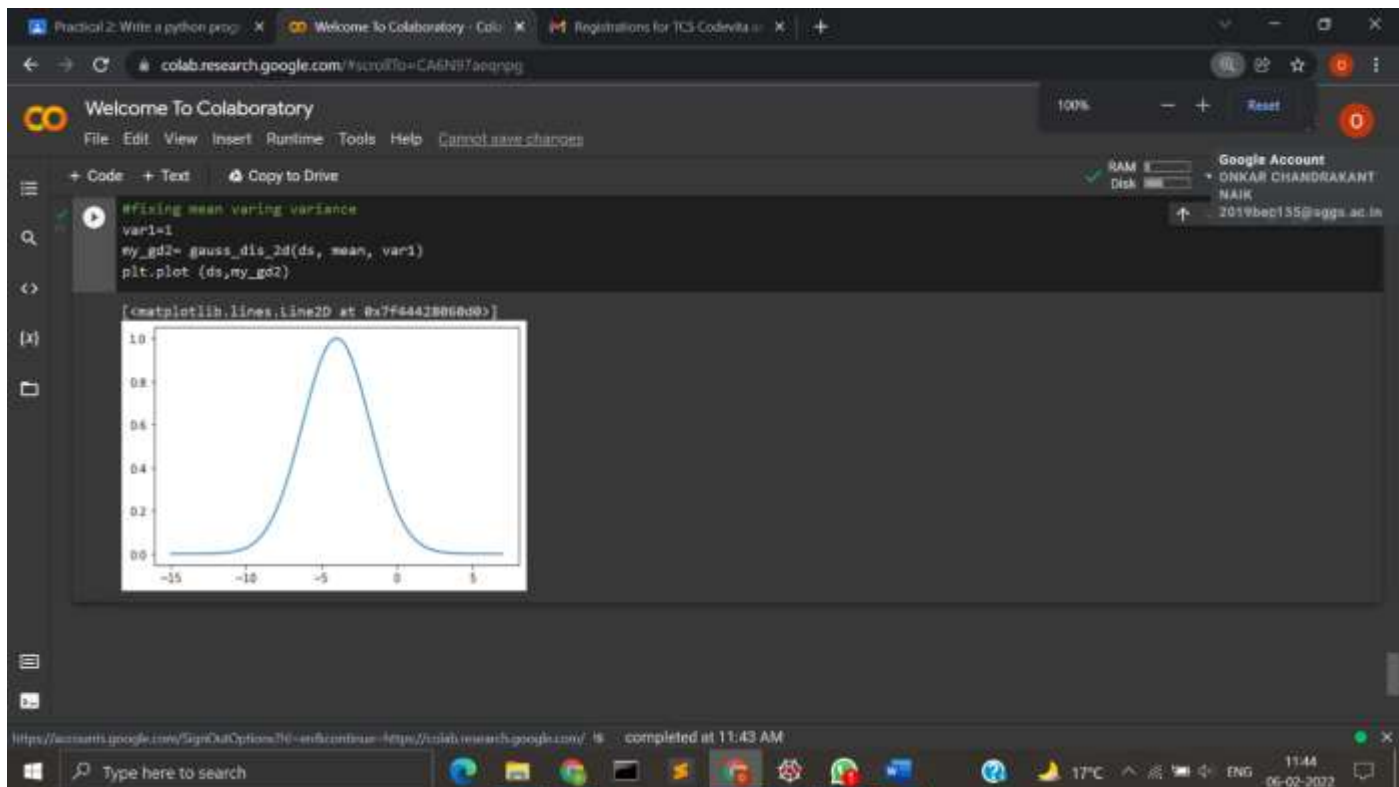
```
#2d gaussian distribution
import numpy as np
ds=np.arange(-15,7,0.03)
print(ds)
print(ds.shape)
mean = sum(ds)/ds.shape
var=np.var(ds)
print(mean)
print(var)
```

The output of the code is a long list of values representing a 2D Gaussian distribution sample. The values are printed in scientific notation, ranging from approximately -1.5×10^{-1} to 1.8×10^{-1} . The shape of the array is $(734,)$. The mean is -4.001 and the variance is 40.486249999817 .

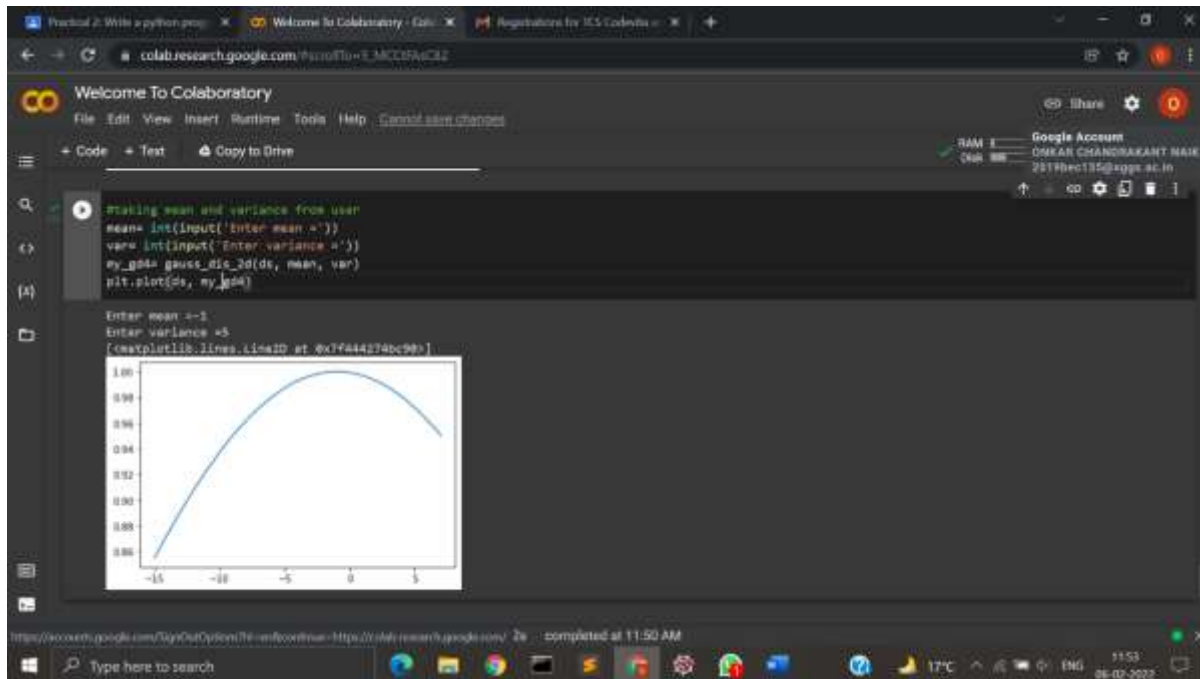
2. Use formula, and plot the 1D Gaussian curve.



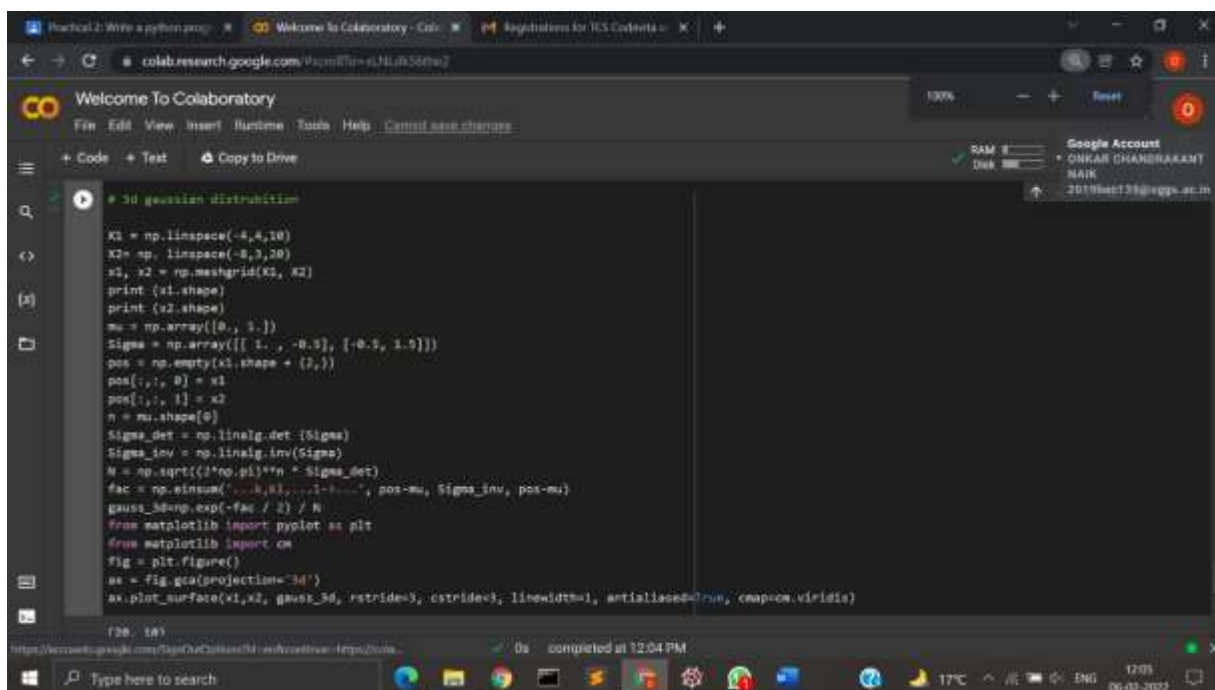
3. Fix the mean and vary the variance and plot the curve. Fix the variance and vary the mean and plot the curve.



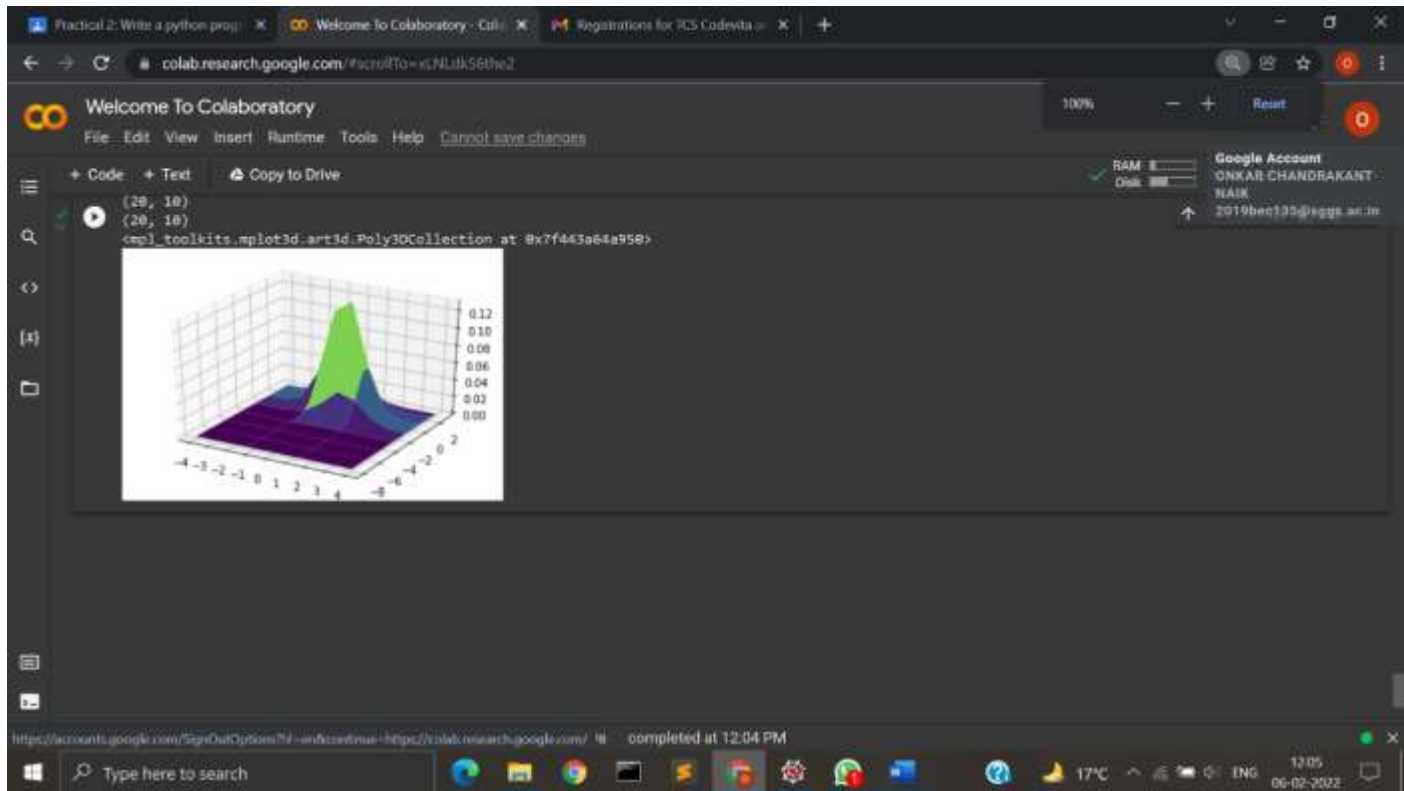
4. Ask the user for mean and variance and plot the 2D Gaussian distribution for it using function definition.



5. Define the data samples x1 and x2 for 2D data. Define $X1, X2 = \text{meshgrid}[x1:x2]$ and print its shape.



6. According to formula, compute pdf and plot 3D graph of Gaussian curve. Give proper labels.



Conclusion: In the given practical we plot 2d and 3d Gaussian distribution curve using probability of likelihood formula. From the graphs we can conclude that the peak of curve depends on mean value and the width of the curve depend on variance.

Practical-3

Name: Naik Onkar Chandrakant

Reg No : 2019BEC135

Roll No : A65

AIM : Write a python program for linear regression modelling.

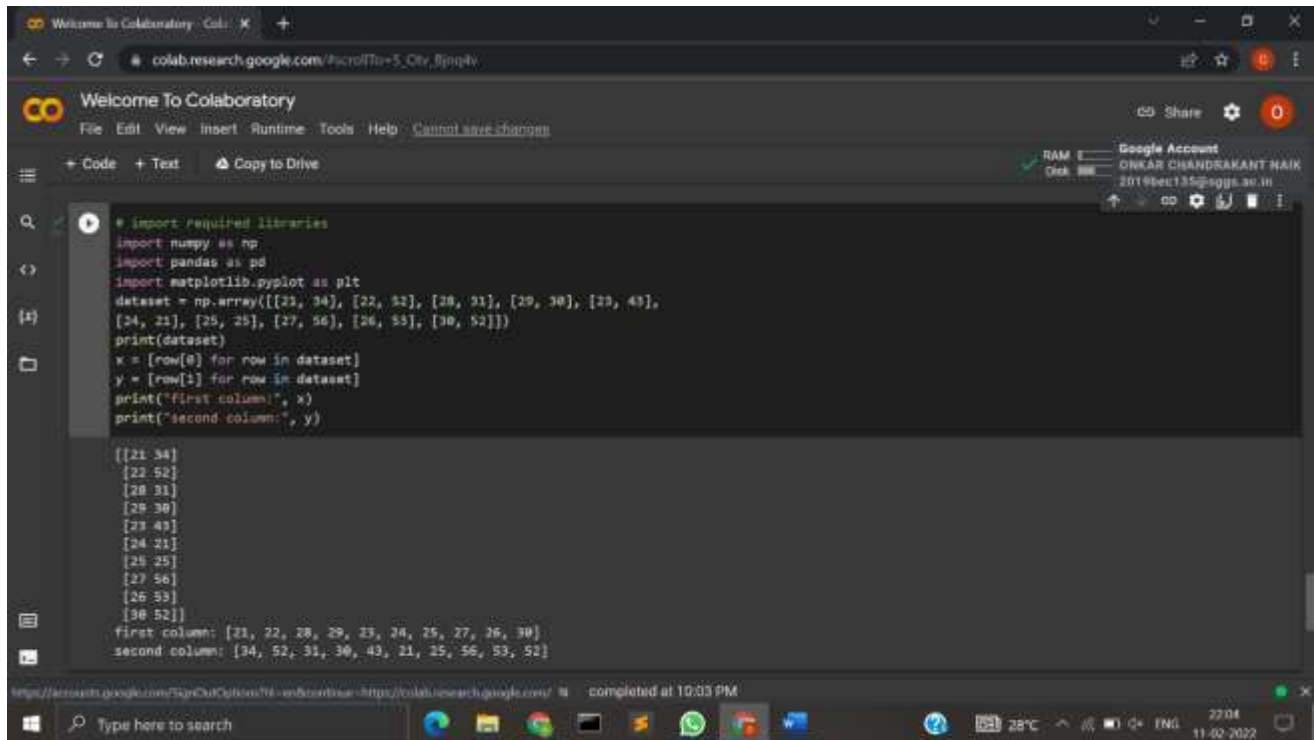
SOFTWARE :- Google Colab , Pycharm

THEORY :-

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price**, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (x) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

1. Create own dataset containing (10 rows and 2 column). Print the dataset.



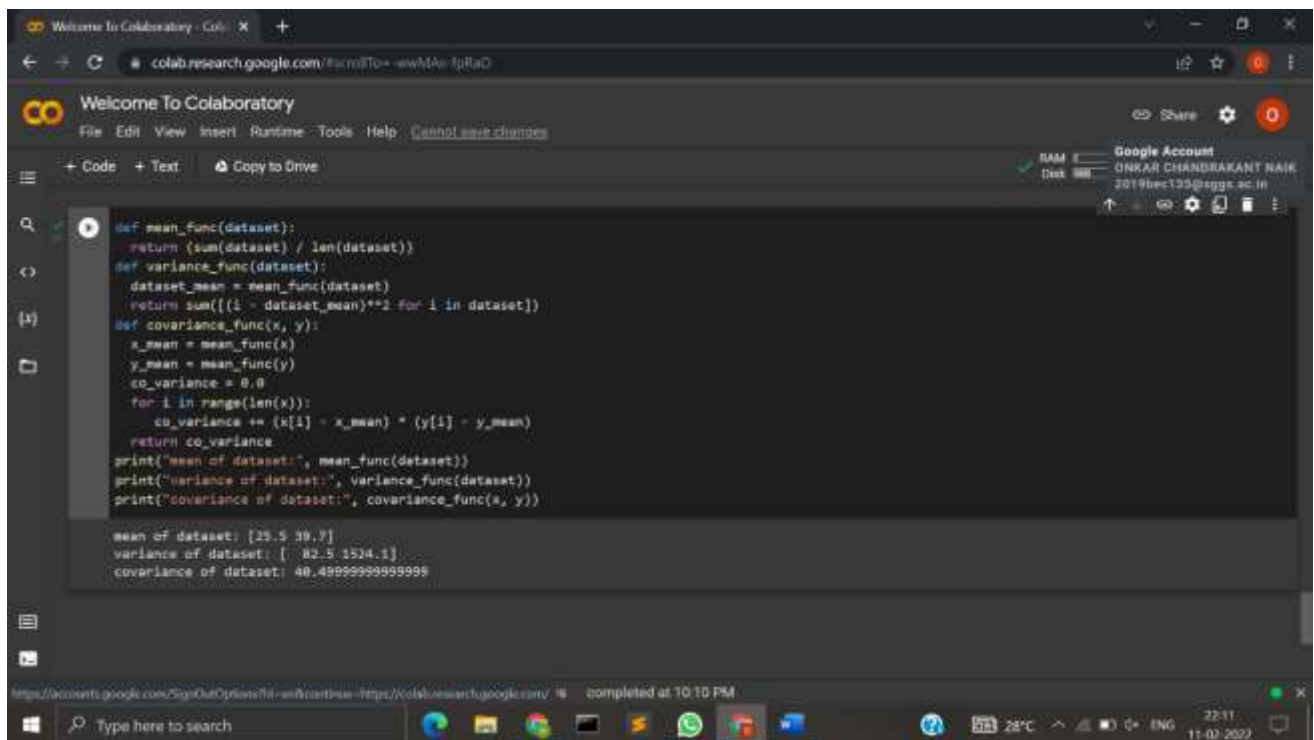
The screenshot shows a Google Colab interface with a Jupyter notebook. The code in the notebook is as follows:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset = np.array([[21, 34], [22, 52], [28, 31], [29, 30], [23, 43],
                    [24, 21], [28, 25], [27, 56], [26, 53], [30, 52]])
print(dataset)
x = [row[0] for row in dataset]
y = [row[1] for row in dataset]
print("first column:", x)
print("second column:", y)
```

The output of the code is:

```
[[21 34]
 [22 52]
 [28 31]
 [29 30]
 [23 43]
 [24 21]
 [28 25]
 [27 56]
 [26 53]
 [30 52]]
first column: [21, 22, 28, 29, 23, 24, 25, 27, 26, 30]
second column: [34, 52, 31, 30, 43, 21, 25, 56, 53, 52]
```

2. Define the functions for computing mean, variance and co-variance and also print the respective values for given dataset.



The screenshot shows a Google Colab interface with a Jupyter notebook. The code in the notebook is as follows:

```
def mean_func(dataset):
    return (sum(dataset) / len(dataset))
def variance_func(dataset):
    dataset_mean = mean_func(dataset)
    return sum([(i - dataset_mean)**2 for i in dataset])
def covariance_func(x, y):
    x_mean = mean_func(x)
    y_mean = mean_func(y)
    co_variance = 0.0
    for i in range(len(x)):
        co_variance += (x[i] - x_mean) * (y[i] - y_mean)
    return co_variance
print("mean of dataset:", mean_func(dataset))
print("variance of dataset:", variance_func(dataset))
print("covariance of dataset:", covariance_func(x, y))
```

The output of the code is:

```
mean of dataset: [25.5 39.7]
variance of dataset: [ 82.5 1524.1]
covariance of dataset: 40.49999999999999
```

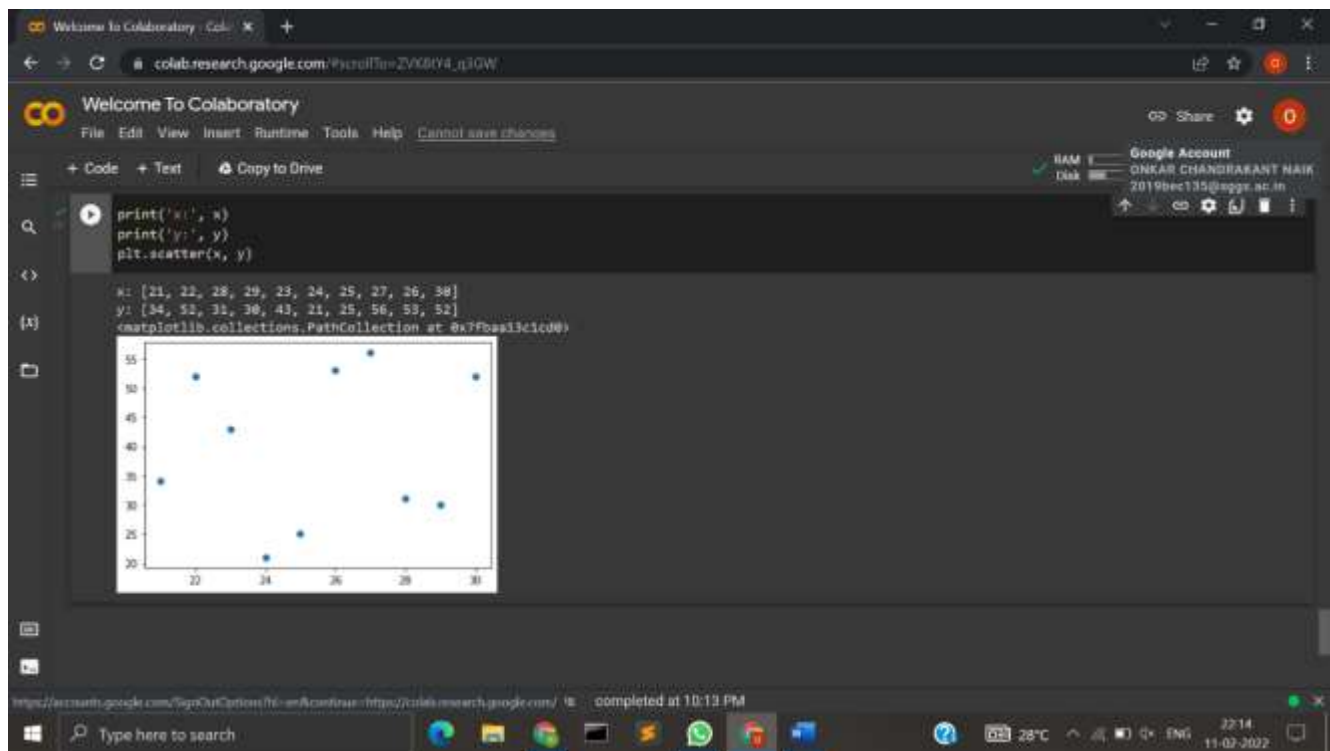
3. Compute the linear regression coefficients using function and print it.

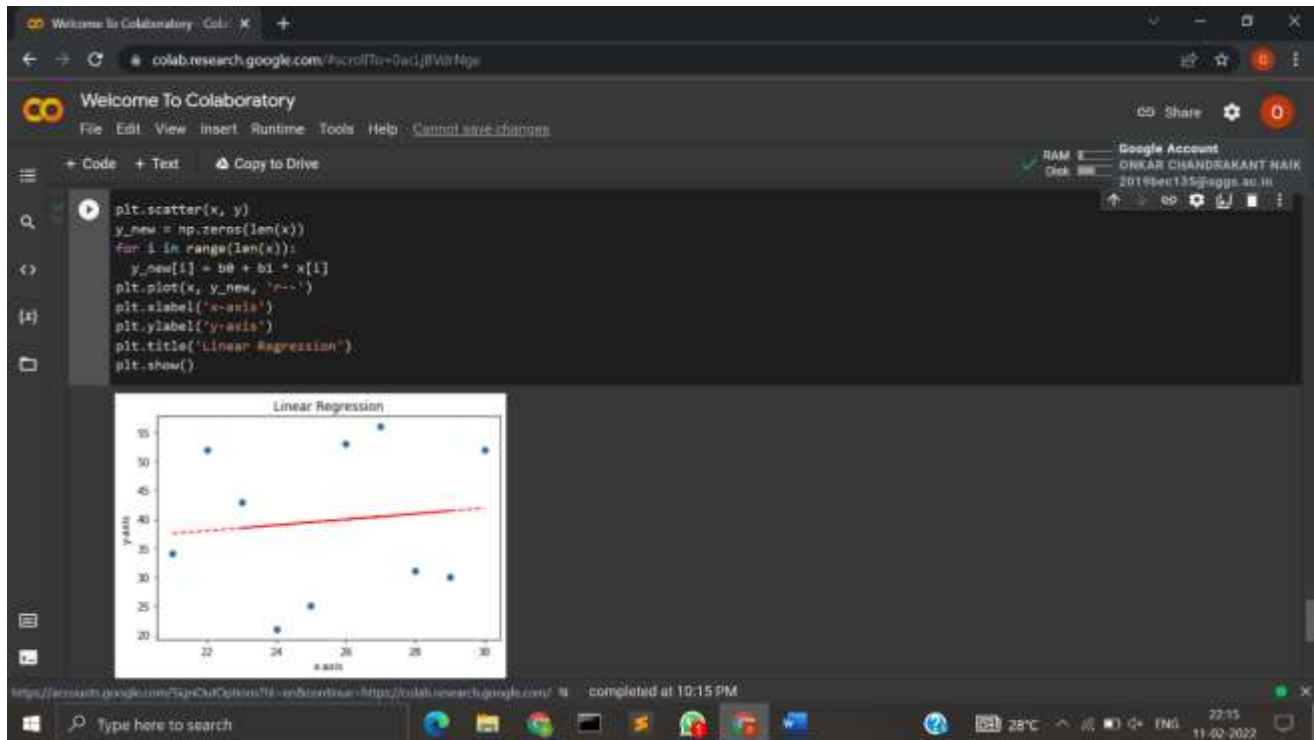
```
def coefficients(dataset):
    x = [row[0] for row in dataset]
    y = [row[1] for row in dataset]
    x_mean, y_mean = mean_func(x), mean_func(y)
    b1 = covariance_func(x, y) / variance_func(x)
    b0 = y_mean - b1 * x_mean
    return [b0, b1]

b0, b1 = coefficients(dataset)
print('coefficients: B0: {:.3f}, B1: {:.3f}'.format(b0, b1))
```

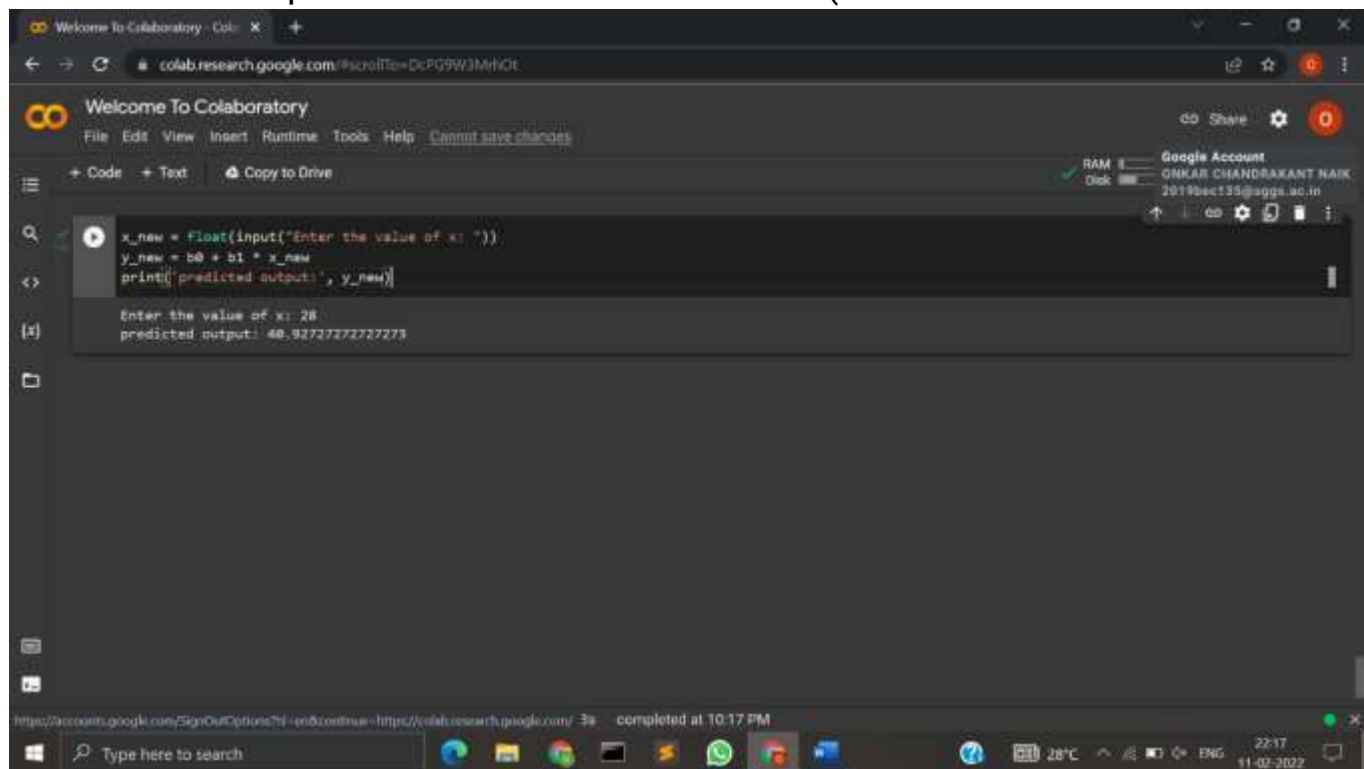
coefficients: B0: 27.182, B1: 0.491

4. Plot the variables containing the values from dataset. Also plot the linear regression line using coefficients.

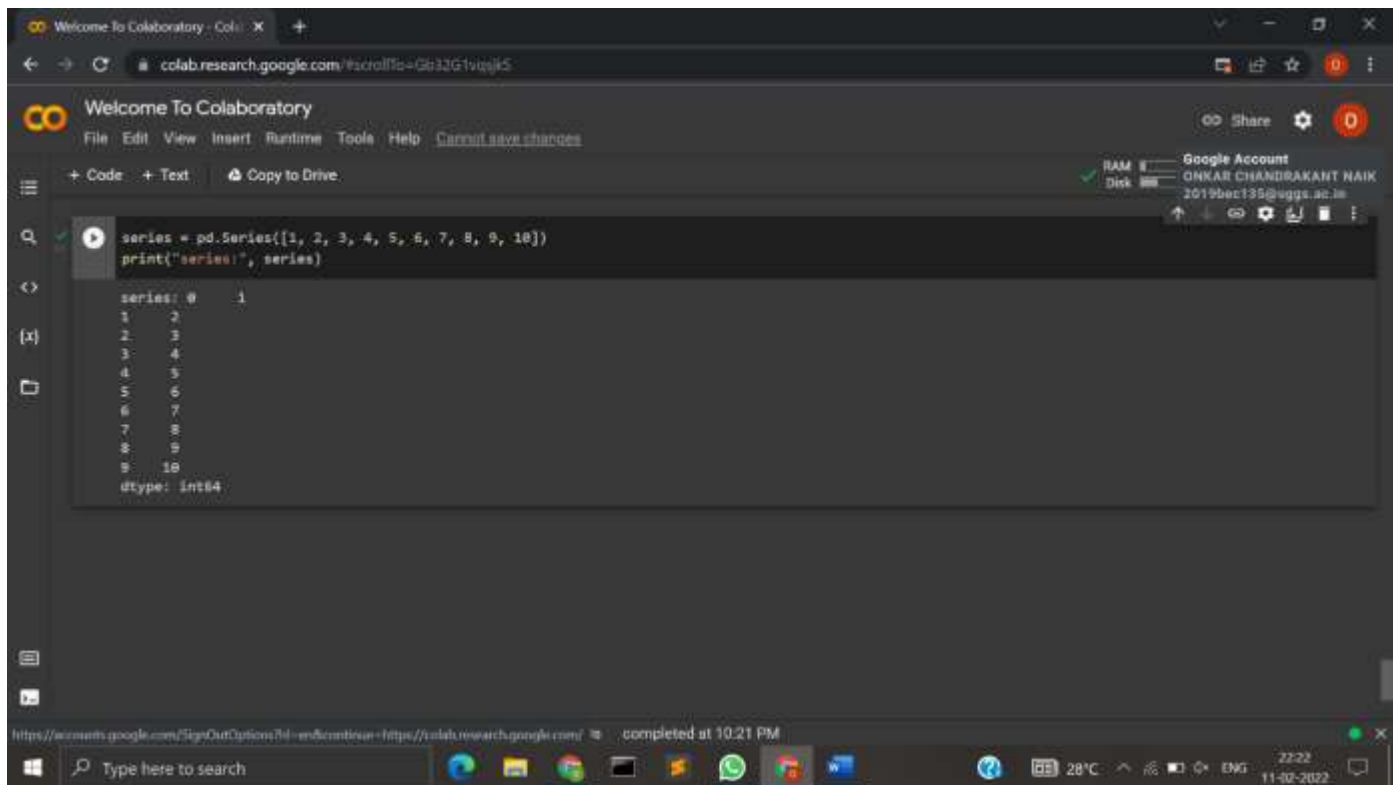




5. Find the predicted output for unknown input value using linear regression coefficients and print it. (for own dataset)



6. Create your own pandas Series and DataFrame and print it.



The screenshot shows the Google Colaboratory web interface. The code editor contains the following Python code:

```
series = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
print("series:", series)
```

The output of the code is displayed below the editor:

```
series: 0      1
        1      2
        2      3
        3      4
        4      5
        5      6
        6      7
        7      8
        8      9
        9     10
dtype: int64
```

The interface also shows the Google Account menu with the name ONKAR CHANDRAKANT NAIK and the email 2019bec135@eggs.ac.in. The status bar at the bottom indicates the session was completed at 10:21 PM.

CONCLUSION :- It is algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

Practical-4

Onkar Chandrakant Naik

2019BEC135

A65

AIM: Write a Python program for the implementation of the Bayes

Theorem.REQUIRED OUTPUT :- Google Colab , Pycharm

THEORY :-

Bayes theorem is one of the most popular machine learning concepts that helps to calculate the probability of occurring one event with uncertain knowledge while other one has already occurred.

Bayes' theorem can be derived using product rule and conditional probability of event X with known event Y:

- According to the product rule we can express as the probability of event X with known event Y as follows;
 1. $P(X \cap Y) = P(X|Y) P(Y)$
- Further, the probability of event Y with known event X:
 1. $P(X \cap Y) = P(Y|X) P(X)$

Mathematically, Bayes theorem can be expressed by combining both equations on right handside. We will get:

Here, both events X and Y are independent events which means probability of outcome of both events does not depend on another.

The above equation is called as Bayes Rule or Bayes Theorem.

- $P(X|Y)$ is called as **posterior**, which we need to calculate. It is defined as updated probability after considering the evidence.
- $P(Y|X)$ is called the likelihood. It is the probability of evidence when hypothesis is true.

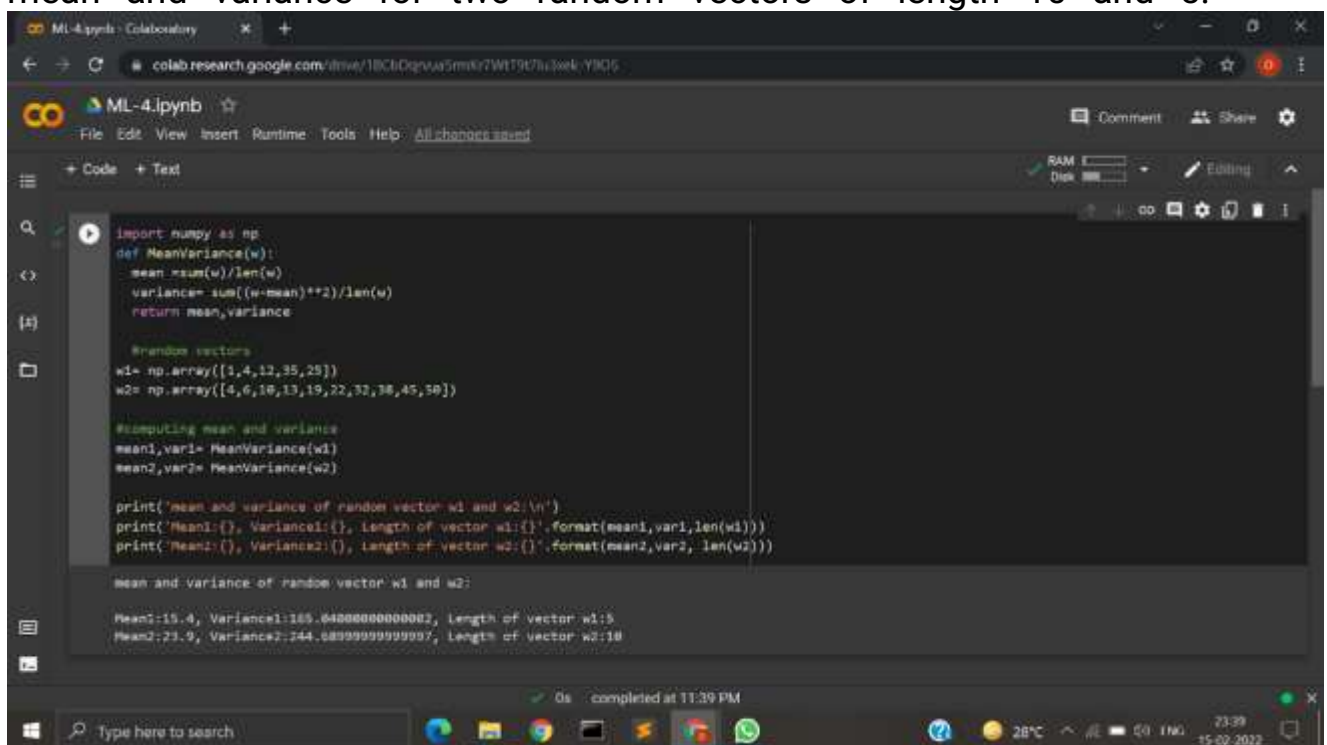
- $P(X)$ is called the **prior probability**, probability of hypothesis before considering the evidence

$P(Y)$ is called marginal probability. It is defined as the probability of evidence under any consideration.

Hence, Bayes Theorem can be written as:

$$\text{posterior} = \text{likelihood} * \text{prior} / \text{evidence}$$

1. Define a function for computing the mean and variance of 1D data. Print mean and variance for two random vectors of length 10 and 5.



```
import numpy as np

def MeanVariance(w):
    mean = sum(w)/len(w)
    variance = sum((w-mean)**2)/len(w)
    return mean, variance

# Random vectors
w1 = np.array([1,4,12,35,25])
w2 = np.array([4,6,10,13,19,22,32,38,45,50])

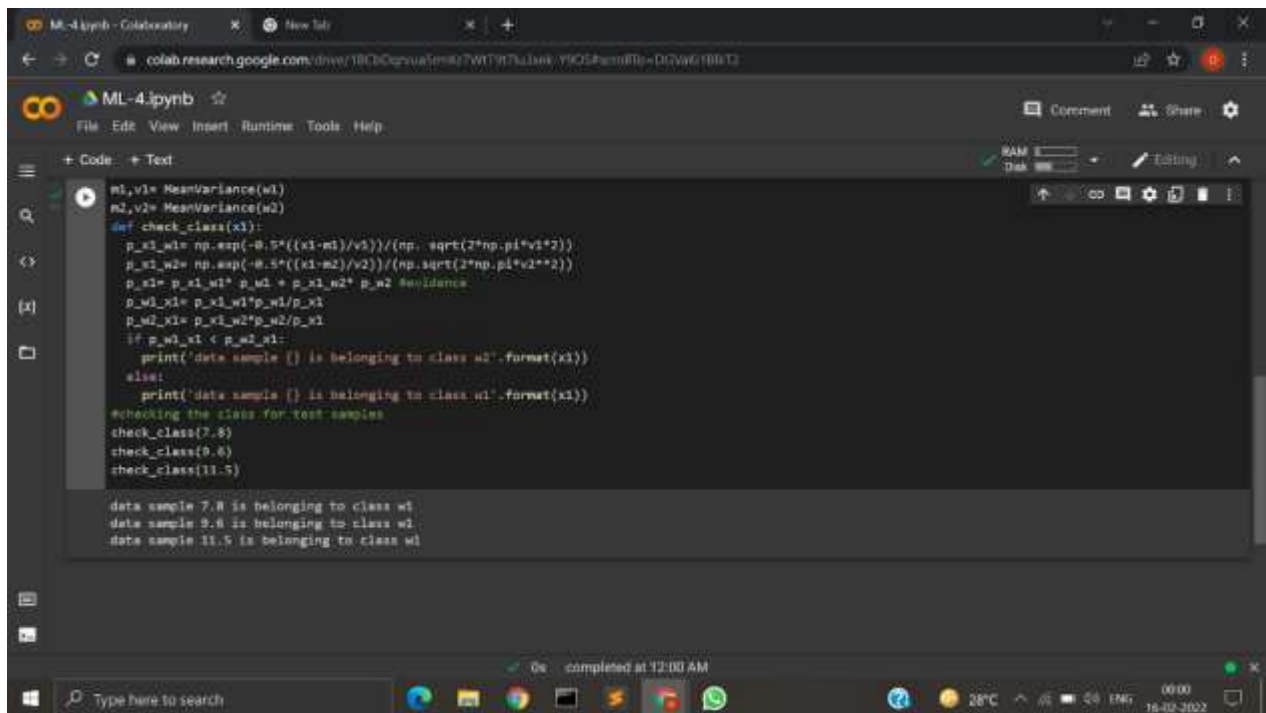
# Computing mean and variance
mean1, var1 = MeanVariance(w1)
mean2, var2 = MeanVariance(w2)

print('Mean and variance of random vector w1 and w2:\n')
print('Mean: {}, Variance: {}, Length of vector w1: {}'.format(mean1, var1, len(w1)))
print('Mean: {}, Variance: {}, Length of vector w2: {}'.format(mean2, var2, len(w2)))
```

Mean and variance of random vector w1 and w2:

Mean: 15.4, Variance: 165.64000000000002, Length of vector w1: 5
Mean: 23.9, Variance: 244.68000000000007, Length of vector w2: 10

2. If $w1 = [1,3,8,12]$ with $p(w1) = 0.6$ and $w2 = [6,9,13,16]$ with $p(w2) = 0.4$, find the class for test samples 7.8, 9.6 and 11.5 using baye's rule.

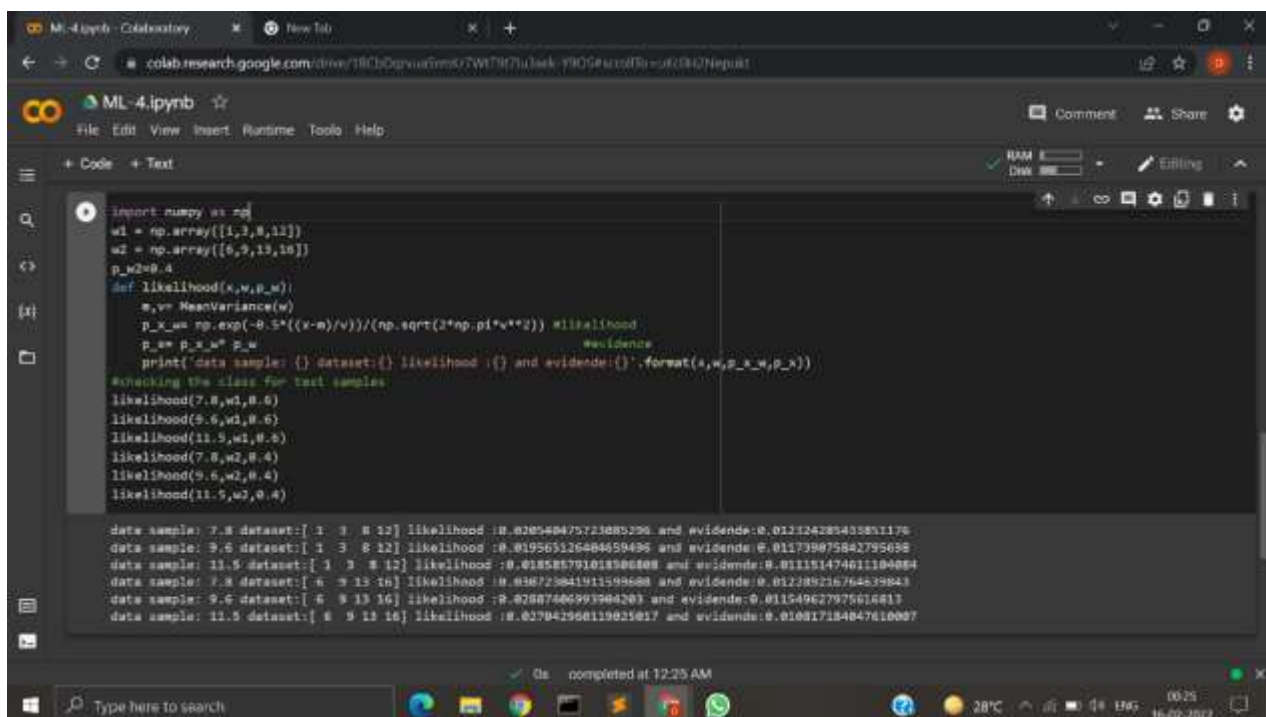


```

m1,v1= MeanVariance(w1)
m2,v2= MeanVariance(w2)
def check_class(x1):
    p_x1_w1= np.exp(-0.5*((x1-m1)/v1))/(np.sqrt(2*np.pi*v1**2))
    p_x1_w2= np.exp(-0.5*((x1-m2)/v2))/(np.sqrt(2*np.pi*v2**2))
    p_w1= p_x1_w1* p_w1 + p_x1_w2* p_w2 #evidance
    p_w1_x1= p_x1_w1*p_w1/p_x1
    p_w2_x1= p_x1_w2*p_w2/p_x1
    if p_w1_x1 < p_w2_x1:
        print('data sample {} is belonging to class w2'.format(x1))
    else:
        print('data sample {} is belonging to class w1'.format(x1))
#checking the class for test samples
check_class(7.8)
check_class(9.6)
check_class(11.5)

data sample 7.8 is belonging to class w1
data sample 9.6 is belonging to class w1
data sample 11.5 is belonging to class w1
  
```

3. Define the function to compute the probability of likelihood and evidence for given 1D data and 1D test sample.

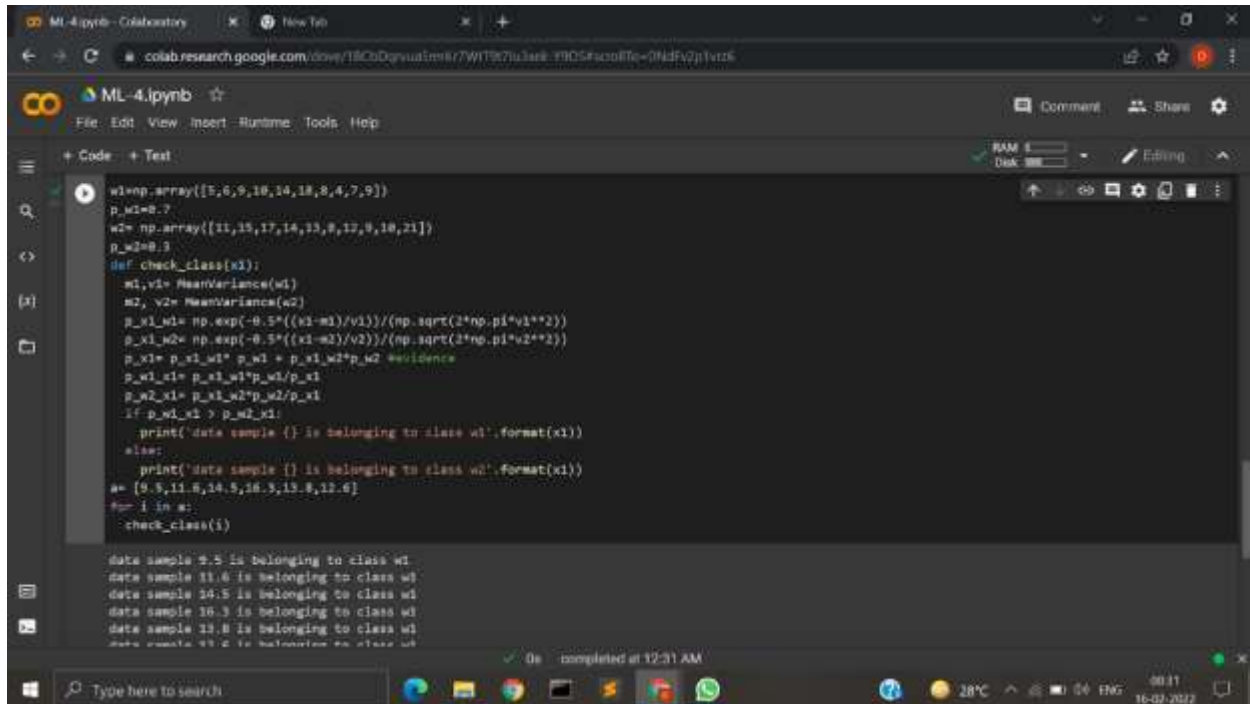


```

import numpy as np
w1 = np.array([1,3,8,12])
w2 = np.array([6,9,13,16])
p_w2=0.4
def likelihood(x,w,p_w):
    m,v= MeanVariance(w)
    p_x_w= np.exp(-0.5*((x-m)/v))/(np.sqrt(2*np.pi*v**2)) #likelihood
    p_w= p_x_w* p_w #evidance
    print('data sample: {} dataset: {} likelihood: {} and evidance: {}'.format(x,w,p_w,p_x))
#checking the class for test samples
likelihood(7.8,w1,0.6)
likelihood(9.6,w1,0.6)
likelihood(11.5,w1,0.6)
likelihood(7.8,w2,0.4)
likelihood(9.6,w2,0.4)
likelihood(11.5,w2,0.4)

data sample: 7.8 dataset:[ 1  3  8 12] likelihood :0.020540475723885396 and evidance:0.0122428543381176
data sample: 9.6 dataset:[ 1  3  8 12] likelihood :0.019565126404459436 and evidance:0.011739075842795698
data sample: 11.5 dataset:[ 1  3  8 12] likelihood :0.018585791018906808 and evidance:0.01151474011104084
data sample: 7.8 dataset:[ 6  9 13 16] likelihood :0.030725041915996808 and evidance:0.012289216764439843
data sample: 9.6 dataset:[ 6  9 13 16] likelihood :0.02807406993984203 and evidance:0.011549627975616813
data sample: 11.5 dataset:[ 6  9 13 16] likelihood :0.017042968119025017 and evidance:0.010817184047610087
  
```

4. If $w_1 = [5, 6, 9, 10, 14, 18, 8, 4, 7, 9]$ with $p(w_1) = 0.7$ and $w_2 = [11, 15, 17, 14, 13, 8, 12, 9, 10, 21]$ with $p(w_2) = 0.3$, find the class for test samples 9.5, 11.6, 14.5, 16.3, 13.8 and 12.6 using baye's rule by defining function.

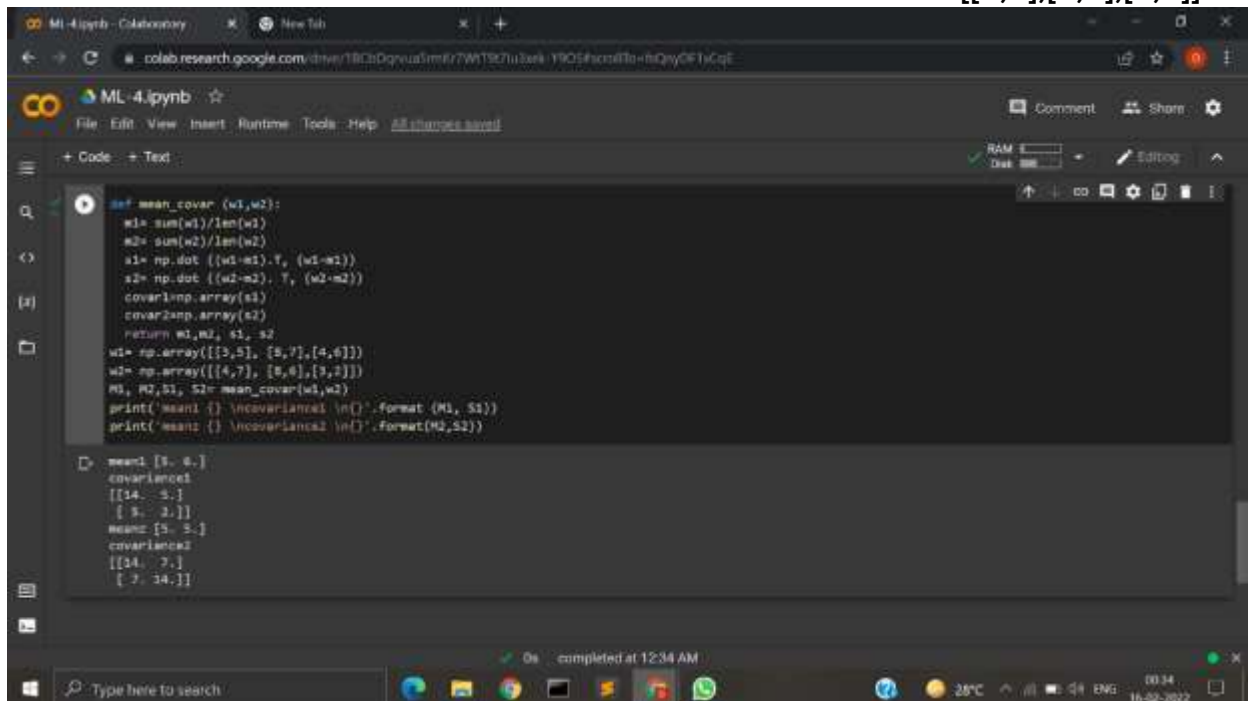


```

w1=np.array([5,6,9,10,14,18,8,4,7,9])
p_w1=0.7
w2= np.array([11,15,17,14,13,8,12,9,10,21])
p_w2=0.3
def check_class(x1):
    m1,v1= MeanVariance(w1)
    m2,v2= MeanVariance(w2)
    p_x1_w1= np.exp(-0.5*((x1-m1)/v1))/(np.sqrt(2*np.pi*v1**2))
    p_x1_w2= np.exp(-0.5*((x1-m2)/v2))/(np.sqrt(2*np.pi*v2**2))
    p_x1= p_x1_w1* p_w1 + p_x1_w2*p_w2 #evidence
    p_w1_x1= p_x1_w1*p_w1/p_x1
    p_w2_x1= p_x1_w2*p_w2/p_x1
    if p_w1_x1 > p_w2_x1:
        print('data sample {} is belonging to class w1'.format(x1))
    else:
        print('data sample {} is belonging to class w2'.format(x1))
a= [9.5,11.6,14.5,16.3,13.8,12.6]
for i in a:
    check_class(i)

data sample 9.5 is belonging to class w1
data sample 11.6 is belonging to class w1
data sample 14.5 is belonging to class w1
data sample 16.3 is belonging to class w1
data sample 13.8 is belonging to class w1
data sample 12.6 is belonging to class w1
  
```

5. Define the function for computing the mean and covariance matrix for 2D data. Print the individual mean & co-variance matrix for $\begin{bmatrix} 3, 5 \\ 8, 7 \end{bmatrix}, \begin{bmatrix} 4, 6 \\ 4, 7 \end{bmatrix}, \begin{bmatrix} 8, 6 \\ 3, 2 \end{bmatrix}$.



```

def mean_covar(w1,w2):
    m1= sum(w1)/len(w1)
    m2= sum(w2)/len(w2)
    s1= np.dot((w1-m1).T, (w1-m1))
    s2= np.dot((w2-m2).T, (w2-m2))
    covar1=np.array(s1)
    covar2=np.array(s2)
    return m1,m2, s1, s2

w1= np.array([[3,5], [8,7],[4,6]])
w2= np.array([[4,7], [8,6],[3,2]])
M1, M2, S1, S2= mean_covar(w1,w2)
print('mean1 {} \ncovariance1 {}'.format(M1, S1))
print('mean2 {} \ncovariance2 {}'.format(M2, S2))

mean1 [1.  4.]
covariance1
[[14.  5.]
 [ 5.  2.]]
mean2 [5.  5.]
covariance2
[[14.  7.]
 [ 7. 14.]]
  
```

6. If $w1 = \begin{bmatrix} 2, 4 \\ 4, 3 \\ 5, 8 \end{bmatrix}$ with $p(w1) = 0.65$ and $w2 = \begin{bmatrix} 4, 6 \\ 5, 7 \\ 7, 10 \end{bmatrix}$ with $p(w2) = 0.35$, find the class for test samples $[4.5, 5.5]$, $[3.7, 5.8]$ and $[5.5, 7.3]$.

```

p_w1=0.65;
w1=np.array([[4,6],[5,7],[7,10]]);
p_w2=0.35;
M1,M2,S1= mean_covar(w1,w2)
def check_class(x_new,M1,S1,M2,S2):
    x1_det= np.linalg.det(S1)
    x2_det= np.linalg.det(S2)
    x1_inv= np.linalg.pinv(S1)
    x2_inv= np.linalg.pinv(S2)
    inner_prod1= np.dot((x_new-M1),T_ (x1_inv))
    p_x1_w1= np.exp(0.5*np.dot(inner_prod1,(x_new-M1)))/np.sqrt(2*np.pi*x1_det)
    inner_prod2= np.dot((x_new-M2),T_ (x2_inv))
    p_x1_w2= np.exp(0.5*np.dot(inner_prod2,(x_new-M2)))/np.sqrt(2*np.pi*x2_det)
    p_x1= p_x1_w1*p_w1 + p_x1_w2*p_w2
    p_x2= p_x1_w1*p_w1/p_x1
    p_x2_w2= p_x1_w2*p_w2/p_x1
    if p_x1 > p_x2:
        print('data sample {} is belonging to class w1'.format(x_new))
    else:
        print('data sample {} is belonging to class w2'.format(x_new))
w1= np.array([4,5,5])
w2= np.array([3,7,7])
x1= np.array([5,5,7,3])
check_class(w1,M1, S1, M2,S2)
check_class(w2,M1,S1,M2,S2)
check_class(x1,M1, S1, M2,S2)

```

data sample [4.5 5.5] is belonging to class w1
data sample [3.7 5.8] is belonging to class w1

7. Define the function to compute the probability of likelihood and evidence for given 2D data and 2D test samples.

```

def likelihood_ev(x_new,M1,S1,M2,S2):
    x1_det= np.linalg.det(S1)
    x2_det= np.linalg.det(S2)
    x1_inv= np.linalg.pinv(S1)
    x2_inv= np.linalg.pinv(S2)
    inner_prod1= np.dot((x_new-M1),T_ (x1_inv))
    p_x1_w1= np.exp(-0.5*np.dot(inner_prod1,(x_new-M1)))/np.sqrt(2*np.pi*x1_det)
    inner_prod2= np.dot((x_new-M2),T_ (x2_inv))
    p_x1_w2= np.exp(-0.5*np.dot(inner_prod2,(x_new-M2)))/np.sqrt(2*np.pi*x2_det)
    p_x1= p_x1_w1*p_w1 + p_x1_w2*p_w2
    p_x2= p_x1_w1*p_w1/p_x1
    p_x2_w2= p_x1_w2*p_w2/p_x1
    likelihood= print('data sample {} likelihood (x1/w1): {} (x2/w2): {}'.format(x_new,p_x1_w1,p_x1_w2,p_x1))
    likelihood_ev(x1,M1, S1, M2, S2)
    likelihood_ev(x2,M1, S1, M2, S2)
    likelihood_ev(x3,M1,S1,M2,S2)

```

data sample [4.5 5.5] likelihood (x1/w1): 0.000316178108054889 (x2/w2) 0.0117497915407262 evidence: 0.0114907869742079
data sample [3.7 5.8] likelihood (x1/w1): 2.0413402908524873e-11 (x2/w2) 0.02946791840950888 evidence: 0.0071635170157004951
data sample [5.5 7.3] likelihood (x1/w1): 0.018810097906640155 (x2/w2) 0.0260070436173125 evidence: 0.047580728450457904

8. If $w1 = [[6,9],[13,10],[8,11],[14,17],[19,17]]$ with $p(w1) = 0.63$ and $w2 = [[11,8],[16,19],[21,18],[22,25],[15,13]]$ with $p(w2) = 0.37$, find the class for test samples $[7.7,9.9],[11,13],[13,16],[14,19],[5,30]$ using baye's rule by defining function.

```

x1_new = np.random.randn(2)
s1_det = np.linalg.det(S1)
s1_inv = np.linalg.inv(S1)
s2_det = np.linalg.det(S2)
s2_inv = np.linalg.inv(S2)
inner_prod1 = np.dot((x_new-M1),T, {s1_inv})
p_x1_w1 = np.exp(-0.5*np.dot(inner_prod1, {x_new-M1}))/np.sqrt(2*np.pi*s1_det)
inner_prod2 = np.dot((x_new-M2),T, {s2_inv})
p_x1_w2 = np.exp(-0.5*np.dot(inner_prod2, {x_new-M2}))/np.sqrt(2*np.pi*s2_det)
p_x1 = p_x1_w1*p_w1 + p_x1_w2*p_w2
p_w1_x1 = p_x1_w1*p_w1/p_x1
p_w2_x1 = p_x1_w2*p_w2/p_x1
if p_x1_w1 < p_x1_w2:
    print('data sample {} is belonging to class w2'.format(x_new))
else:
    print('data sample {} is belonging to class w1'.format(x_new))
w1 = np.array([[6,9],[13,10],[8,11],[14,17],[19,17]])
w2 = np.array([[11,8],[16,19],[21,18],[22,25],[15,13]])
M1, M2, S1, S2 = mean_covar(w1,w2)
x1 = np.array([7.7,9.9])
x2 = np.array([11,13])
x3 = np.array([13,16])
x4 = np.array([14,19])
x5 = np.array([5,30])
check_class(x1,M1, S1, M2, S2)
check_class(x2,M1, S1, M2, S2)
check_class(x3,M1,S1, M2, S2)
check_class(x4,M1,S1, M2, S2)
check_class(x5,M1,S1, M2, S2)

```

data sample [7.7 9.9] is belonging to class w1
data sample [11 13] is belonging to class w1
data sample [13 16] is belonging to class w2
data sample [14 19] is belonging to class w1

CONCLUSION :- Bayes' Theorem calculates the conditional probability of an event, based on the values of specific related known probabilities. At its simplest, Bayes' Theorem takes a test result and relates it to the conditional probability of that test result given other related events.

Practical-5

Name: Naik Onkar Chandrakant

Reg No : 2019BEC135

Roll No : A65

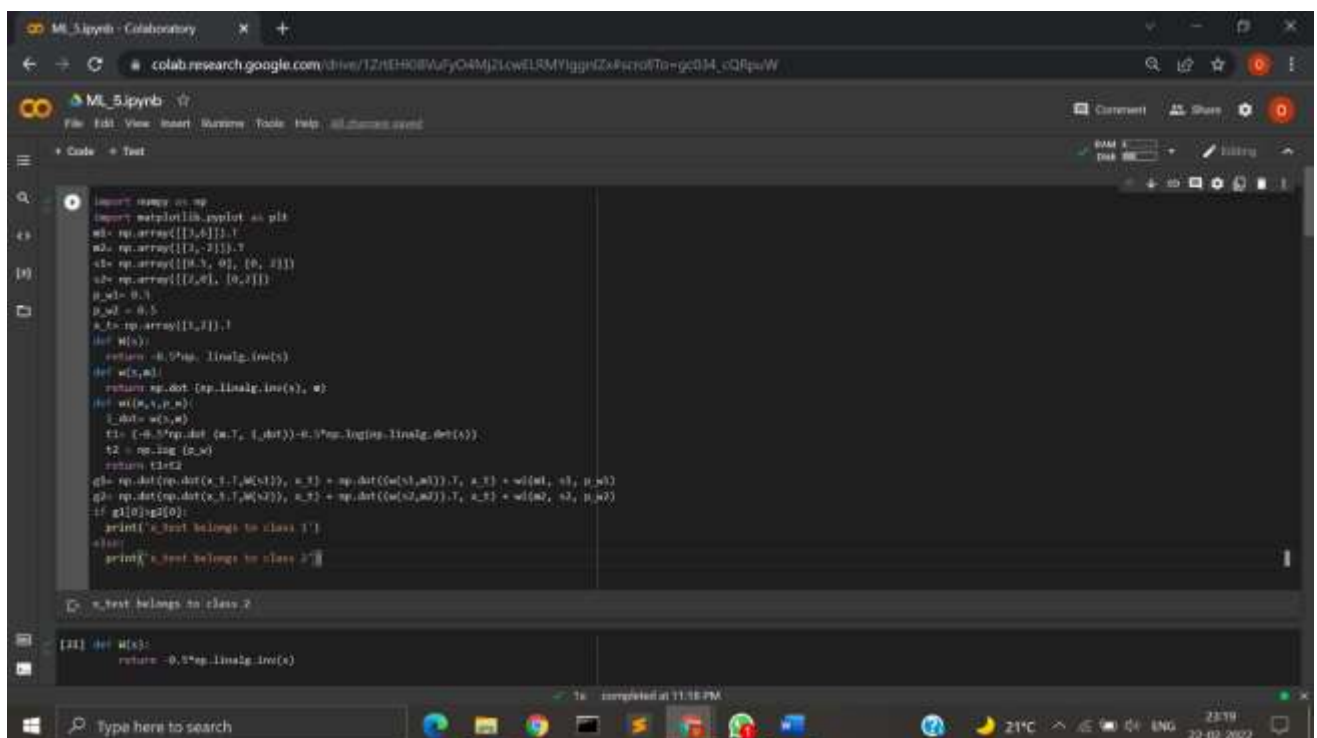
Aim: Write a python program to implement discriminant function and decision boundary.

REQUIRED SOFTWARE :- Google

colab , PycharmTHEORY :-

Aim: Write a python program to implement discriminant function and decision boundary. A decision boundary is a line (in the case of 2D) or a plane (in the case of 3D) that separates the feature space into regions, each of which is associated with a class. All samples of one class are on one side of that line, and all samples of the other class are on the opposite side of the line. The line *separates* one class from the other. If you have more than two features, the decision boundary is not a line, but a (hyper)-plane in the dimension of your feature space.

1. Classify test sample $x_{\text{test}} = [1, 2]^T$ using discriminant function for given data: $m_1 = [3, 6]^T$, $m_2 = [3, -2]^T$, $S_1 = [[0.5, 0], [0, 2]]$, $S_2 = [[2, 0], [0, 2]]$, $p(w_1)=0.5$, $p(w_2)=0.5$.

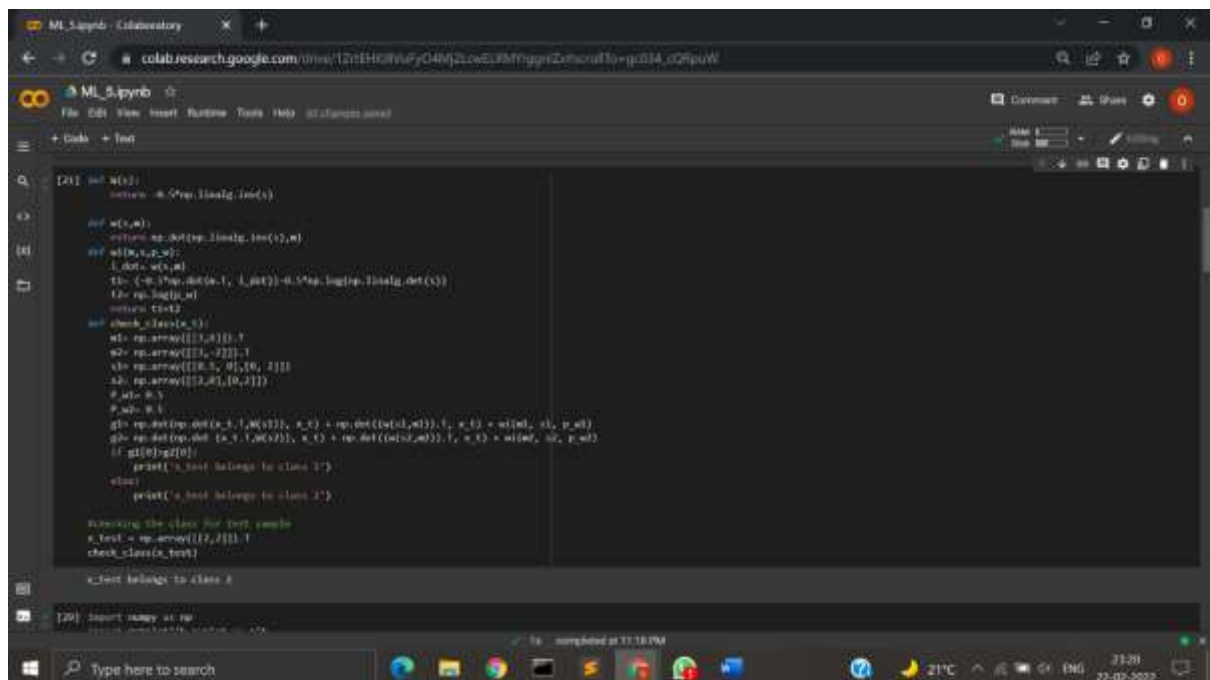


```
import numpy as np
import matplotlib.pyplot as plt
m1 = np.array([3,6])
m2 = np.array([3,-2])
S1 = np.array([0.5, 0], [0, 2])
S2 = np.array([2, 0], [0, 2])
p_w1 = 0.5
p_w2 = 0.5
x_test = np.array([1,2])
def W(x):
    return -0.5*np.linalg.log(S1)
def w(x,m):
    return np.dot(np.linalg.inv(S1), x)
def w1(x,p_w1):
    l1 = -0.5*np.dot(x.T, l1.dot()) - 0.5*np.log(np.linalg.det(S1))
    l2 = np.log(p_w1)
    return l1+l2
g1 = np.dot(np.dot(x.T, W(S1)), x) + np.dot((w(S1,m1)).T, x) + w(m1, S1, p_w1)
g2 = np.dot(np.dot(x.T, W(S2)), x) + np.dot((w(S2,m2)).T, x) + w(m2, S2, p_w2)
if g1[0]>g2[0]:
    print('x_test belongs to class 1')
else:
    print('x_test belongs to class 2')
```

11 def W(x):
12 return -0.5*np.linalg.log(S1)

1x completed at 11:18 PM

2. Classify test sample $x_{\text{test}} = [2, 2]^T$ for above data.



```

[28] def w(x):
    return 0.5*np.linalg.log(x)

def w(x,w):
    return np.dot(np.linalg.log(x),w)

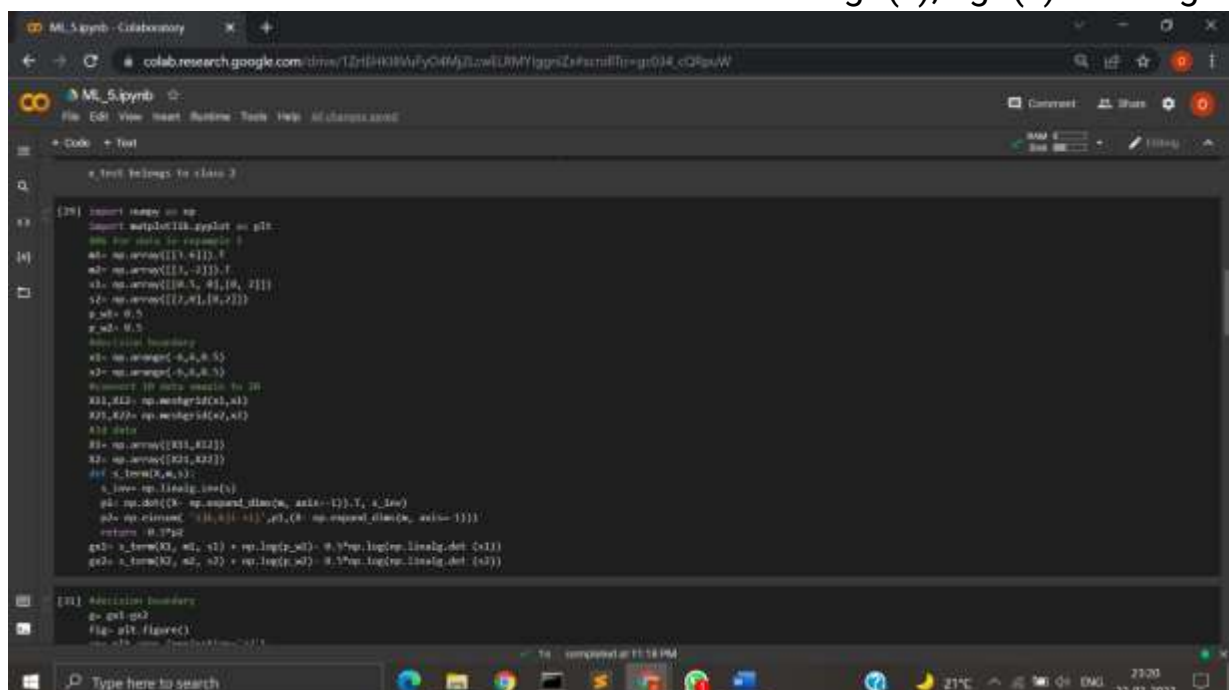
def w(x,w):
    L_data = w(x,w)
    L1 = (-0.5*np.dot(w-1, L_data))-0.5*np.log(np.linalg.det(x))
    L2 = np.log(L1,w)
    return L1+L2

def check_class(x,y):
    w1 = np.array([1,1,1],1)
    w2 = np.array([1,1,1],1)
    w3 = np.array([1,0,1],1)
    w4 = np.array([1,0,1],1)
    w5 = 0.5
    w6 = 0.5
    g1 = np.dot(np.dot(x,1,w1), x,y) + np.dot(np.dot(x,w2),1, x,y) + w3(w1, x1, p,w1)
    g2 = np.dot(np.dot(x,1,w2), x,y) + np.dot(np.dot(x,w3),1, x,y) + w4(w2, w3, p,w2)
    if g1[0]>g2[0]:
        print('x test belongs to class 1')
    else:
        print('x test belongs to class 2')

#testing the class for test sample
x_test = np.array([1,1,1],1)
check_class(x_test)

x_test belongs to class 2
  
```

3. Find the Decision boundary for subtask 1. Take $x1=np.arange(-6,6,0.5)$ & $x2=np.arange(-6,6,0.5)$ and use `np.meshgrid` for converting 1D data to 2D data. Plot the 3D surface of $g1(x)$, $g2(x)$ and g .



```

[29] import numpy as np
import matplotlib.pyplot as plt

#data for class 1 response 1
w1 = np.array([1,1,1],1)
w2 = np.array([1,1,1],1)
w3 = np.array([1,0,1],1)
w4 = np.array([1,0,1],1)
w5 = 0.5
w6 = 0.5

#decision boundary
x1 = np.arange(-6,6,0.5)
x2 = np.arange(-6,6,0.5)
#convert 1D data matrix to 2D
X1,X2 = np.meshgrid(x1,x2)
X1,X2 = np.meshgrid(x2,x1)
#1D data
w1 = np.array([X1,X2])
w2 = np.array([X2,X2])
def x_test(x,w):
    g1 = np.linalg.log(x)
    g2 = np.dot(x, np.array([w1,w2,w3,w4,w5,w6]))
    g3 = np.dot(x, [1,1,1],1) + w1(w1, x1, p,w1)
    return 0.5*g1
g1 = x_test(X1, w1, x1) + np.log(x,w1) - 0.5*np.log(np.linalg.det(x))
g2 = x_test(X2, w2, x2) + np.log(x,w2) - 0.5*np.log(np.linalg.det(x))

[30] #Decision boundary
g1 = g1-g2
fig=plt.figure()
plt.plot(g1)
plt.show()
  
```


5. Classify test sample $x_{\text{test}} = [2, 1]^T$ for above data.

The screenshot shows a Google Colab notebook titled "ML_S.pyrb". The notebook contains a Python function named `linear_regression` that takes a list of (x, y) pairs and returns the coefficients of the best-fit line. The code is executed in a cell, and the output shows the coefficients [0.5, 1.0]. The notebook interface includes a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". The status bar at the bottom indicates the notebook is "completed at 17:18 PM".

```
[40] def linear_regression(x, y):
    # Initialize parameters
    w0 = 0
    w1 = 0
    # Compute the mean of x and y
    x_mean = sum(x)/len(x)
    y_mean = sum(y)/len(y)
    # Compute the covariance and variance of x
    cov_x = sum((x[i] - x_mean)*(y[i] - y_mean)) / (len(x) - 1)
    var_x = sum((x[i] - x_mean)**2) / (len(x) - 1)
    # Compute the slope of the best-fit line
    w1 = cov_x / var_x
    # Compute the intercept of the best-fit line
    w0 = y_mean - w1 * x_mean
    # Return the coefficients
    return [w0, w1]

# Test the function
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
coeffs = linear_regression(x, y)
print(coeffs)
```

Output:

```
[0.5, 1.0]
```

The notebook is titled "ML_S.pyrb" and the status bar at the bottom indicates it is "completed at 17:18 PM".

6. Find the Decision boundary for subtask 4. Take $x1=np.arange(-8,8,0.5)$ & $x2=np.arange(-8,8,0.5)$ and use `np.meshgrid` for converting 1D data to 2D data. Plot the 3D surface of $g_1(x)$, $g_2(x)$ and g .

The screenshot shows a Google Colab notebook titled "ML_5.pyrb". The notebook has two code cells. The first cell, labeled [46], imports numpy and defines a function `def plot1D(x)` that generates a 1D dataset with 1000 points, splits it into training and testing sets, and plots the data. The second cell, labeled [47], imports matplotlib and displays the plot using `plt.show()`. The notebook interface shows the file explorer on the left, the code editor in the center, and the output area at the bottom. The status bar at the bottom indicates the notebook is "completed at 11:18 PM".

```
[46] import numpy as np
import matplotlib.pyplot as plt

# Set the seed for reproducibility
s1 = np.random.seed(1)
s2 = np.random.seed(2)

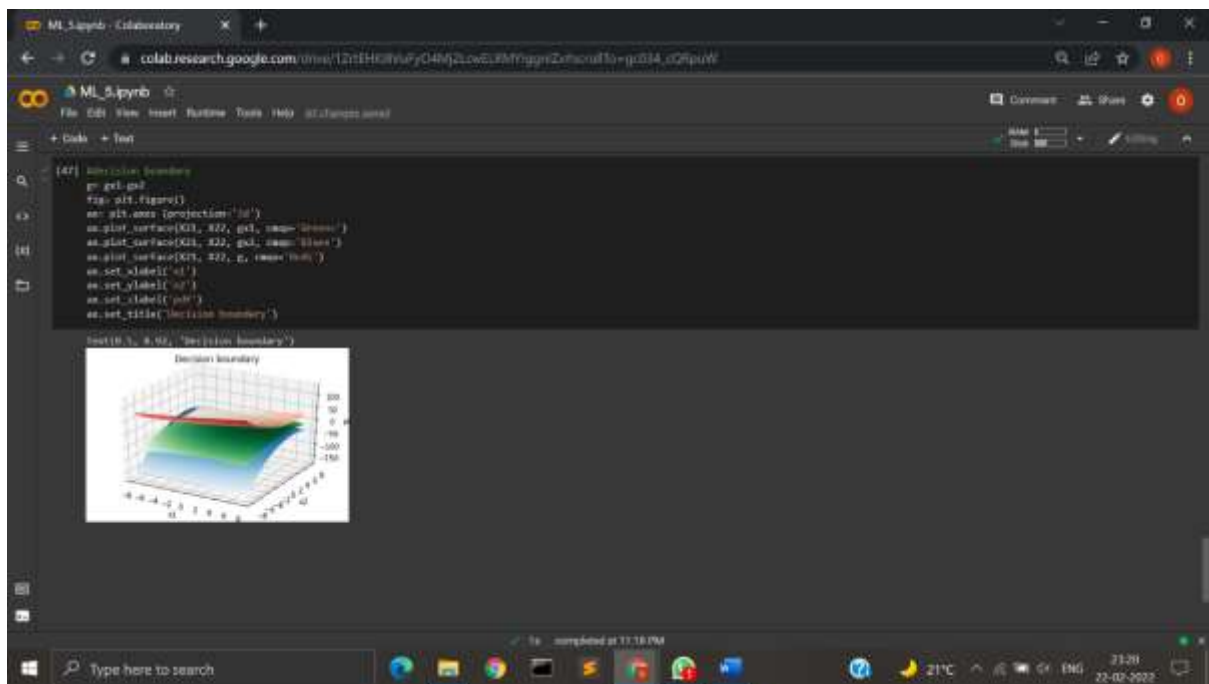
# Generate data
x = np.linspace(0, 1, 1000)
y = np.random.randn(1000)

# Split the data into training and testing sets
x_train = x[:800]
y_train = y[:800]
x_test = x[800:]
y_test = y[800:]

# Plot the data
plt.figure(figsize=(10, 5))
plt.scatter(x_train, y_train, color='blue', label='Training Data')
plt.scatter(x_test, y_test, color='red', label='Testing Data')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()

[47] import matplotlib.pyplot as plt

# Display the plot
plt.show()
```



CONCLUSION:- A decision boundary or decision surface is a hyper surface that partitions the underlying vector space into two sets, one for each class. The classifier will classify all the points on one side of the decision boundary as belonging to one class and all those on the otherside as belonging to the other class.

Practical-6

Name: Naik Onkar Chandrakant

Reg No : 2019BEC135

Roll No : A65

AIM: - Write a python program to implement

Naïve Bayes Classifier.

SOFTWARE :- Google colab , Pycharm

THEORY:-

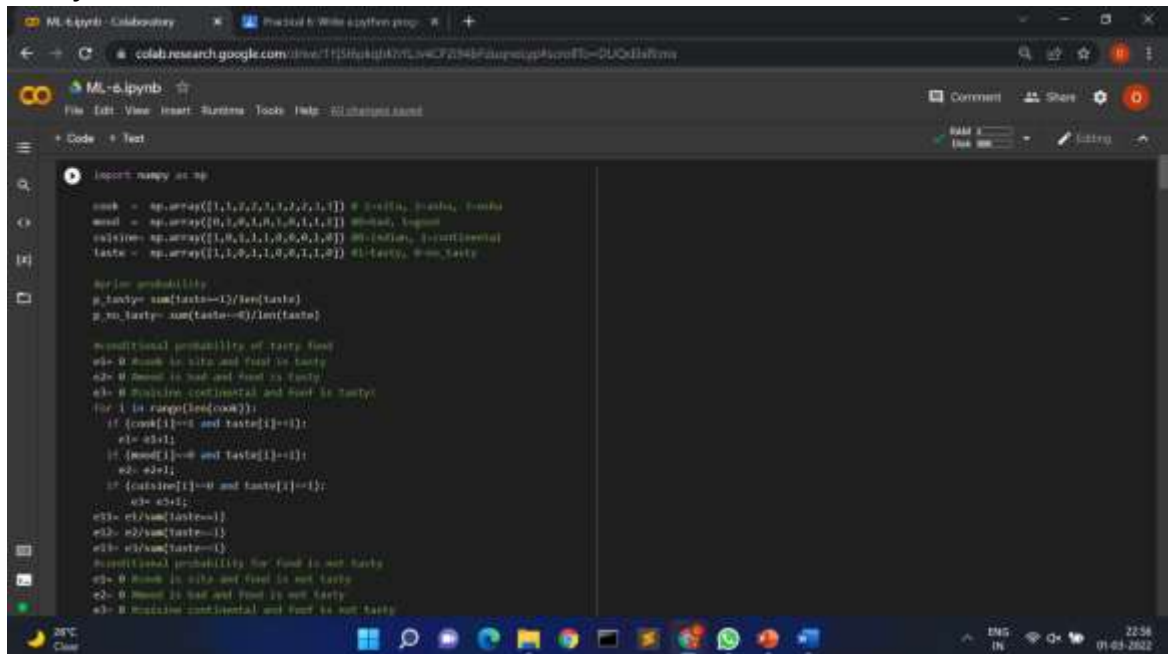
Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible efficacy of naive Bayes classifiers.^[6] Still, a comprehensive comparison with other classification algorithms in 2006 showed that Bayes classification is outperformed by other approaches, such as boosted trees or random forests.^[7]

An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

1.If Cook is Sita, Mood is Bad and preparing Continental food, is food tasty?



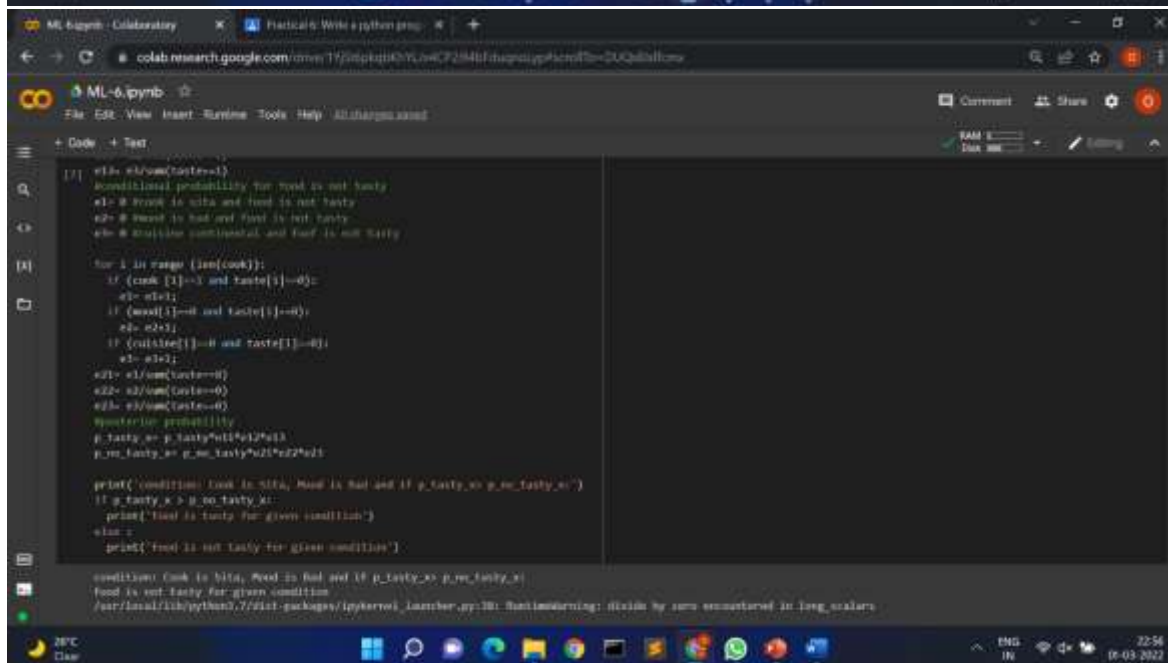
```
import numpy as np

cook = np.array([1,1,2,2,3,3,3,3,1,1]) # 0-sita, 1-mood, 2-mood
mood = np.array([0,1,0,1,0,1,0,1,1,1]) # mood, 0-good, 1-bad
cuisine = np.array([1,0,1,1,1,0,0,0,1,0]) # cuisine, 1-continental, 0-indian
taste = np.array([1,1,0,1,1,0,0,1,1,0]) # 0-tasty, 1-not_tasty

# prior probability
p_tasty = sum(taste==1)/len(taste)
p_no_tasty = sum(taste==0)/len(taste)

# conditional probability of tasty food
e1= 0 # Cook is sita and food is tasty
e2= 0 # Mood is bad and food is tasty
e3= 0 # Cuisine continental and food is tasty
for i in range(len(cook)):
    if (cook[i]==1 and taste[i]==1):
        e1+=1
    if (mood[i]==1 and taste[i]==1):
        e2+=1
    if (cuisine[i]==1 and taste[i]==1):
        e3+=1
e1= e1/sum(taste==1)
e2= e2/sum(taste==1)
e3= e3/sum(taste==1)

# conditional probability for food is not tasty
e1= 0 # Cook is sita and food is not tasty
e2= 0 # Mood is bad and food is not tasty
e3= 0 # Cuisine continental and food is not tasty
```



```
with np.sum(taste==1)
# conditional probability for food is not tasty
e1= 0 # Cook is sita and food is not tasty
e2= 0 # Mood is bad and food is not tasty
e3= 0 # Cuisine continental and food is not tasty
for i in range (len(cook)):
    if (cook [i]==1 and taste[i]==0):
        e1+=1
    if (mood[i]==1 and taste[i]==0):
        e2+=1
    if (cuisine[i]==1 and taste[i]==0):
        e3+=1
e1= e1/sum(taste==0)
e2= e2/sum(taste==0)
e3= e3/sum(taste==0)

# posterior probability
p_tasty_x = p_tasty*e1*e2*e3
p_no_tasty_x = p_no_tasty*e1*e2*e3

print('condition: Cook is Sita, Mood is Bad and if p_tasty_x > p_no_tasty_x:')
if p_tasty_x > p_no_tasty_x:
    print('food is tasty for given condition')
else:
    print('food is not tasty for given condition')

condition: Cook is Sita, Mood is Bad and if p_tasty_x > p_no_tasty_x
food is not tasty for given condition
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:30: RuntimeWarning: divide by zero encountered in long_scalars
```


2.If Cook is Asha for above case, is food tasty?

[illegible]

3.If Cook is Usha, Mood is Bad and preparing Continental food, is food tasty?

The screenshot shows a Jupyter Notebook in Google Colab. The notebook is titled "Practical 6: Write a python prog...". The code in the notebook is as follows:

```

condition: Cook is Alibi, Food is Bad and preparing (conditional) Food.
Food is not tasty for given condition.

#conditional probability of tasty food (cook is alibi)
for i in range(len(cook)):
    if (cook[i]==i and taste[i]==1):
        c1+=1
#conditional probability for food is not tasty (cook is alibi)
c4=0
for i in range(len(cook)):
    if (cook[i]==i and taste[i]==0):
        c1+=1

E1: c1/(sum(taste==1))
E2: c2/(sum(taste==1))
E3: c3/(sum(taste==1))
E4: c4/(sum(taste==0))
E5: c5/(sum(taste==0))
E6: c6/(sum(taste==0))

#Bayesian probability
p_tasty_x = p_taste*(E1/E2)
p_no_tasty_x = p_no_taste*(E4/E5)
print('condition: Cook is Alibi, Food is Bad and preparing (conditional) Food:')
if p_tasty_x > p_no_tasty_x:
    print('Food is Tasty for given condition')
else:
    print('Food is not Tasty for given condition')

condition: Cook is Alibi, Food is Bad and preparing (conditional) Food.
Food is not Tasty for given condition.

```

4.If car Colour is Red, Type is Sedan and Origin is Domestic, was car stolen?

```

import numpy as np
color = np.array([1,1,2,2,1,2,1,2,2]) # 0-red, 1-blue, 2-white
type = np.array([1,2,1,2,1,2,1,2,1]) # 1-sedan, 2 SUV, 3-truck
origin = np.array([0,1,0,0,1,1,0,0,1]) # 0-domestic, 1-Imported
stolen = np.array([1,1,0,0,1,1,0,0,1]) # 1-yes, 0-no
prior_probabilty
p_stolen = sum(stolen)/len(stolen)
p_not_stolen = sum(stolen==0)/len(stolen)
#conditional probabilities for car stolen
#0-0 sedan is red and car was stolen
#0-1 sedan is sedan and car was stolen
#0-2 truck is domestic and car was stolen
for i in range(len(stolen)):
    if (color[i]==1 and stolen[i]==1):
        #0-0
        #0-1
        #0-2
    elif (color[i]==2 and stolen[i]==1):
        #1-0
        #1-1
        #1-2
    elif (color[i]==0 and stolen[i]==1):
        #2-0
        #2-1
        #2-2
    #conditional probabilities for car not stolen
    #0-0 sedan is red and car was not stolen
    #0-1 sedan is sedan and car was not stolen
    #0-2 truck is domestic and car was not stolen
    for i in range(len(stolen)):
        if (color[i]==1 and stolen[i]==0):
            #0-0
            if (type[i]==1 and stolen[i]==0):
                #0-1
            if (origin[i]==0 and stolen[i]==0):
                #0-2
            #0-3
        elif (color[i]==2 and stolen[i]==0):
            #1-0
            #1-1
            #1-2
        elif (color[i]==0 and stolen[i]==0):
            #2-0
            #2-1
            #2-2

```

```

#0-0 sedan is red and car was not stolen
#0-1 sedan is sedan and car was not stolen
#0-2 truck is domestic and car was not stolen
for i in range(len(stolen)):
    if (color[i]==1 and stolen[i]==0):
        #0-0
        if (type[i]==1 and stolen[i]==0):
            #0-1
        if (origin[i]==0 and stolen[i]==0):
            #0-2
        #0-3
    elif (color[i]==2 and stolen[i]==0):
        #1-0
        #1-1
        #1-2
    elif (color[i]==0 and stolen[i]==0):
        #2-0
        #2-1
        #2-2
#posterior probability
p_stolen_x = p_stolen*(1/3)*(1/3)
p_not_stolen_x = p_not_stolen*(1/3)*(1/3)
print('condition: color is red, type is sedan and origin is domestic:')
if p_stolen_x > p_not_stolen_x:
    print('the car is stolen for given condition')
else:
    print('the car is not stolen for given condition')

```

condition: Color is Red, Type is Sedan and Origin is Domestic:
the car is not stolen for given condition

5.If car Colour is White, Type is SUV and Origin is Imported, was car stolen?

[illegible]

6.If car Colour is Blue, Type is Sports and Origin is Domestic, was car stolen?

The screenshot shows a Google Colab notebook with the following content:

```

def conditional_probabilities_for_car_insurance_and_age_model:
    color = blue and car was stolen and height is normal and car was stolen
    age = 18 and car was stolen and height is normal and car was stolen
    color = blue and car was not stolen
    age = 18 and car was not stolen
    height = normal and car
    for i in range (len (stolen)):
        if (color[i]==2 and stolen[i]==1):
            stolen[i]
        if (age[i]==1 and stolen[i]==1):
            stolen[i]
        if (height[i]==1 and stolen[i]==1):
            stolen[i]
        if (color[i]==2 and stolen[i]==1):
            stolen[i]
        if (age[i]==1 and stolen[i]==1):
            stolen[i]
        if (height[i]==1 and stolen[i]==1):
            stolen[i]
    P1= x1/sum(stolen==1)
    P2= x2/sum(stolen==1)
    P3= x3/sum(stolen==1)
    P4= x4/sum(stolen==1)
    P5= x5/sum(stolen==1)
    P6= x6/sum(stolen==1)
    P7= x7/sum(stolen==1)
    P8= x8/sum(stolen==1)
    P9= x9/sum(stolen==1)
    P10= x10/sum(stolen==1)
    P11= x11/sum(stolen==1)
    P12= x12/sum(stolen==1)
    P13= x13/sum(stolen==1)
    P14= x14/sum(stolen==1)
    P15= x15/sum(stolen==1)
    P16= x16/sum(stolen==1)
    P17= x17/sum(stolen==1)
    P18= x18/sum(stolen==1)
    P19= x19/sum(stolen==1)
    P20= x20/sum(stolen==1)
    P21= x21/sum(stolen==1)
    P22= x22/sum(stolen==1)
    P23= x23/sum(stolen==1)
    P24= x24/sum(stolen==1)
    P25= x25/sum(stolen==1)
    P26= x26/sum(stolen==1)
    P27= x27/sum(stolen==1)
    P28= x28/sum(stolen==1)
    P29= x29/sum(stolen==1)
    P30= x30/sum(stolen==1)
    P31= x31/sum(stolen==1)
    P32= x32/sum(stolen==1)
    P33= x33/sum(stolen==1)
    P34= x34/sum(stolen==1)
    P35= x35/sum(stolen==1)
    P36= x36/sum(stolen==1)
    P37= x37/sum(stolen==1)
    P38= x38/sum(stolen==1)
    P39= x39/sum(stolen==1)
    P40= x40/sum(stolen==1)
    P41= x41/sum(stolen==1)
    P42= x42/sum(stolen==1)
    P43= x43/sum(stolen==1)
    P44= x44/sum(stolen==1)
    P45= x45/sum(stolen==1)
    P46= x46/sum(stolen==1)
    P47= x47/sum(stolen==1)
    P48= x48/sum(stolen==1)
    P49= x49/sum(stolen==1)
    P50= x50/sum(stolen==1)
    P51= x51/sum(stolen==1)
    P52= x52/sum(stolen==1)
    P53= x53/sum(stolen==1)
    P54= x54/sum(stolen==1)
    P55= x55/sum(stolen==1)
    P56= x56/sum(stolen==1)
    P57= x57/sum(stolen==1)
    P58= x58/sum(stolen==1)
    P59= x59/sum(stolen==1)
    P60= x60/sum(stolen==1)
    P61= x61/sum(stolen==1)
    P62= x62/sum(stolen==1)
    P63= x63/sum(stolen==1)
    P64= x64/sum(stolen==1)
    P65= x65/sum(stolen==1)
    P66= x66/sum(stolen==1)
    P67= x67/sum(stolen==1)
    P68= x68/sum(stolen==1)
    P69= x69/sum(stolen==1)
    P70= x70/sum(stolen==1)
    P71= x71/sum(stolen==1)
    P72= x72/sum(stolen==1)
    P73= x73/sum(stolen==1)
    P74= x74/sum(stolen==1)
    P75= x75/sum(stolen==1)
    P76= x76/sum(stolen==1)
    P77= x77/sum(stolen==1)
    P78= x78/sum(stolen==1)
    P79= x79/sum(stolen==1)
    P80= x80/sum(stolen==1)
    P81= x81/sum(stolen==1)
    P82= x82/sum(stolen==1)
    P83= x83/sum(stolen==1)
    P84= x84/sum(stolen==1)
    P85= x85/sum(stolen==1)
    P86= x86/sum(stolen==1)
    P87= x87/sum(stolen==1)
    P88= x88/sum(stolen==1)
    P89= x89/sum(stolen==1)
    P90= x90/sum(stolen==1)
    P91= x91/sum(stolen==1)
    P92= x92/sum(stolen==1)
    P93= x93/sum(stolen==1)
    P94= x94/sum(stolen==1)
    P95= x95/sum(stolen==1)
    P96= x96/sum(stolen==1)
    P97= x97/sum(stolen==1)
    P98= x98/sum(stolen==1)
    P99= x99/sum(stolen==1)
    P100= x100/sum(stolen==1)
    P101= x101/sum(stolen==1)
    P102= x102/sum(stolen==1)
    P103= x103/sum(stolen==1)
    P104= x104/sum(stolen==1)
    P105= x105/sum(stolen==1)
    P106= x106/sum(stolen==1)
    P107= x107/sum(stolen==1)
    P108= x108/sum(stolen==1)
    P109= x109/sum(stolen==1)
    P110= x110/sum(stolen==1)
    P111= x111/sum(stolen==1)
    P112= x112/sum(stolen==1)
    P113= x113/sum(stolen==1)
    P114= x114/sum(stolen==1)
    P115= x115/sum(stolen==1)
    P116= x116/sum(stolen==1)
    P117= x117/sum(stolen==1)
    P118= x118/sum(stolen==1)
    P119= x119/sum(stolen==1)
    P120= x120/sum(stolen==1)
    P121= x121/sum(stolen==1)
    P122= x122/sum(stolen==1)
    P123= x123/sum(stolen==1)
    P124= x124/sum(stolen==1)
    P125= x125/sum(stolen==1)
    P126= x126/sum(stolen==1)
    P127= x127/sum(stolen==1)
    P128= x128/sum(stolen==1)
    P129= x129/sum(stolen==1)
    P130= x130/sum(stolen==1)
    P131= x131/sum(stolen==1)
    P132= x132/sum(stolen==1)
    P133= x133/sum(stolen==1)
    P134= x134/sum(stolen==1)
    P135= x135/sum(stolen==1)
    P136= x136/sum(stolen==1)
    P137= x137/sum(stolen==1)
    P138= x138/sum(stolen==1)
    P139= x139/sum(stolen==1)
    P140= x140/sum(stolen==1)
    P141= x141/sum(stolen==1)
    P142= x142/sum(stolen==1)
    P143= x143/sum(stolen==1)
    P144= x144/sum(stolen==1)
    P145= x145/sum(stolen==1)
    P146= x146/sum(stolen==1)
    P147= x147/sum(stolen==1)
    P148= x148/sum(stolen==1)
    P149= x149/sum(stolen==1)
    P150= x150/sum(stolen==1)
    P151= x151/sum(stolen==1)
    P152= x152/sum(stolen==1)
    P153= x153/sum(stolen==1)
    P154= x154/sum(stolen==1)
    P155= x155/sum(stolen==1)
    P156= x156/sum(stolen==1)
    P157= x157/sum(stolen==1)
    P158= x158/sum(stolen==1)
    P159= x159/sum(stolen==1)
    P160= x160/sum(stolen==1)
    P161= x161/sum(stolen==1)
    P162= x162/sum(stolen==1)
    P163= x163/sum(stolen==1)
    P164= x164/sum(stolen==1)
    P165= x165/sum(stolen==1)
    P166= x166/sum(stolen==1)
    P167= x167/sum(stolen==1)
    P168= x168/sum(stolen==1)
    P169= x169/sum(stolen==1)
    P170= x170/sum(stolen==1)
    P171= x171/sum(stolen==1)
    P172= x172/sum(stolen==1)
    P173= x173/sum(stolen==1)
    P174= x174/sum(stolen==1)
    P175= x175/sum(stolen==1)
    P176= x176/sum(stolen==1)
    P177= x177/sum(stolen==1)
    P178= x178/sum(stolen==1)
    P179= x179/sum(stolen==1)
    P180= x180/sum(stolen==1)
    P181= x181/sum(stolen==1)
    P182= x182/sum(stolen==1)
    P183= x183/sum(stolen==1)
    P184= x184/sum(stolen==1)
    P185= x185/sum(stolen==1)
    P186= x186/sum(stolen==1)
    P187= x187/sum(stolen==1)
    P188= x188/sum(stolen==1)
    P189= x189/sum(stolen==1)
    P190= x190/sum(stolen==1)
    P191= x191/sum(stolen==1)
    P192= x192/sum(stolen==1)
    P193= x193/sum(stolen==1)
    P194= x194/sum(stolen==1)
    P195= x195/sum(stolen==1)
    P196= x196/sum(stolen==1)
    P197= x197/sum(stolen==1)
    P198= x198/sum(stolen==1)
    P199= x199/sum(stolen==1)
    P200= x200/sum(stolen==1)
    P201= x201/sum(stolen==1)
    P202= x202/sum(stolen==1)
    P203= x203/sum(stolen==1)
    P204= x204/sum(stolen==1)
    P205= x205/sum(stolen==1)
    P206= x206/sum(stolen==1)
    P207= x207/sum(stolen==1)
    P208= x208/sum(stolen==1)
    P209= x209/sum(stolen==1)
    P210= x210/sum(stolen==1)
    P211= x211/sum(stolen==1)
    P212= x212/sum(stolen==1)
    P213= x213/sum(stolen==1)
    P214= x214/sum(stolen==1)
    P215= x215/sum(stolen==1)
    P216= x216/sum(stolen==1)
    P217= x217/sum(stolen==1)
    P218= x218/sum(stolen==1)
    P219= x219/sum(stolen==1)
    P220= x220/sum(stolen==1)
    P221= x221/sum(stolen==1)
    P222= x222/sum(stolen==1)
    P223= x2
```

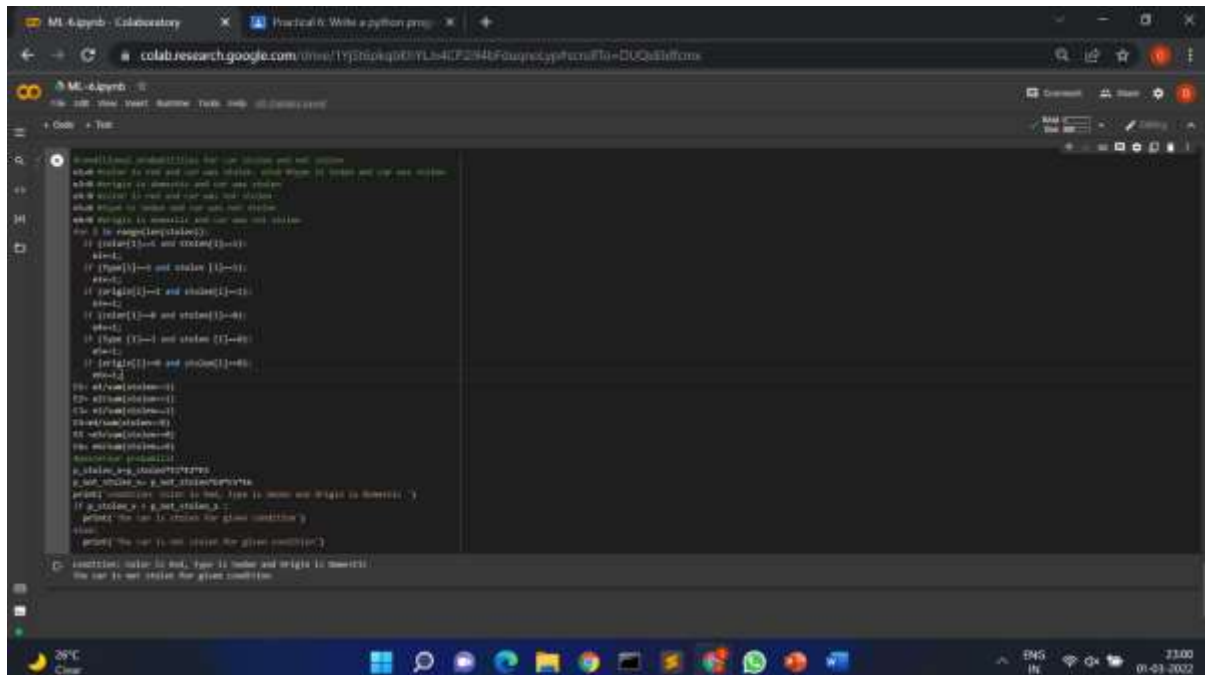
7.If car Colour is White, Type is Sedan and Origin is Domestic, was car stolen?

[illegible]

8.If car Colour is Bleu, Type is Sports and Origin is Imported, was car stolen?

[illegible]

9.If car Colour is Red, Type is Sedan and Origin is Domestic, was car stolen?



```
from collections import defaultdict
def car_stolen_and_not_stolen:
    """Return a list of cars and whether they were stolen or not.
    Each element is a tuple (car, stolen)."""
    # Read the data from the file
    with open('car_data.csv', 'r') as f:
        lines = f.readlines()
    # Parse the data
    cars = []
    for line in lines:
        # Split the line into fields
        fields = line.strip().split(',')
        # Create a car object
        car = {'color': fields[0], 'type': fields[1], 'origin': fields[2], 'stolen': fields[3]}
        cars.append(car)
    # Return the list of cars
    return cars

# Create a Naive Bayes classifier
def naive_bayes_classifier(cars):
    """Return a Naive Bayes classifier for car theft prediction.
    The classifier will take a car's color, type, and origin as input
    and return the probability of the car being stolen or not stolen.
    """
    # Count the number of cars for each combination of color, type, and origin
    counts = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0)))
    for car in cars:
        color = car['color']
        type = car['type']
        origin = car['origin']
        stolen = car['stolen']
        counts[color][type][origin] += 1 if stolen else 0

    # Calculate the prior probabilities
    total_cars = len(cars)
    prior_stolen = sum(counts[color][type][origin] for color, type, origin in counts) / total_cars
    prior_not_stolen = 1 - prior_stolen

    # Calculate the likelihoods
    likelihood_stolen = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0)))
    likelihood_not_stolen = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0)))
    for color, type, origin in counts:
        count_stolen = counts[color][type][origin]
        count_not_stolen = counts[color][type][origin] - count_stolen
        likelihood_stolen[color][type][origin] = count_stolen / (count_stolen + count_not_stolen)
        likelihood_not_stolen[color][type][origin] = count_not_stolen / (count_stolen + count_not_stolen)

    # Calculate the posterior probabilities
    posterior_stolen = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0)))
    posterior_not_stolen = defaultdict(lambda: defaultdict(lambda: defaultdict(lambda: 0)))
    for color, type, origin in counts:
        count_stolen = counts[color][type][origin]
        count_not_stolen = counts[color][type][origin] - count_stolen
        posterior_stolen[color][type][origin] = prior_stolen * likelihood_stolen[color][type][origin]
        posterior_not_stolen[color][type][origin] = prior_not_stolen * likelihood_not_stolen[color][type][origin]

    # Return the posterior probabilities
    return posterior_stolen, posterior_not_stolen

# Test the classifier
cars = car_stolen_and_not_stolen()
posterior_stolen, posterior_not_stolen = naive_bayes_classifier(cars)

# Example car
car = {'color': 'red', 'type': 'sedan', 'origin': 'domestic'}

# Calculate the probability of the car being stolen
posterior_stolen_prob = posterior_stolen[car['color']][car['type']][car['origin']]

# Calculate the probability of the car not being stolen
posterior_not_stolen_prob = posterior_not_stolen[car['color']][car['type']][car['origin']]

# Print the results
print('The car is more likely to be stolen than not stolen.' if posterior_stolen_prob > posterior_not_stolen_prob else 'The car is more likely to be not stolen than stolen.')
print('The car is more likely to be stolen than not stolen.' if posterior_stolen_prob > posterior_not_stolen_prob else 'The car is more likely to be not stolen than stolen.')
```

CONCLUSION: -

Hence, Naïve Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem to predict the outcome. It is a family of algorithms where all of them share a common principle. Naïve Bayes algorithms are mostly used in sentiment analysis, spam filtering, recommendation systems, etc. They are fast and easy to implement

Practical-7

Name : Naik Onkar Chandrakant

Reg. No. : 2019BEC135

Roll No. : A65

AIM : - Write a python program to plot Receiver Operating

Characteristics (ROC) curves.REQUIRED SOFTWARE :- Google

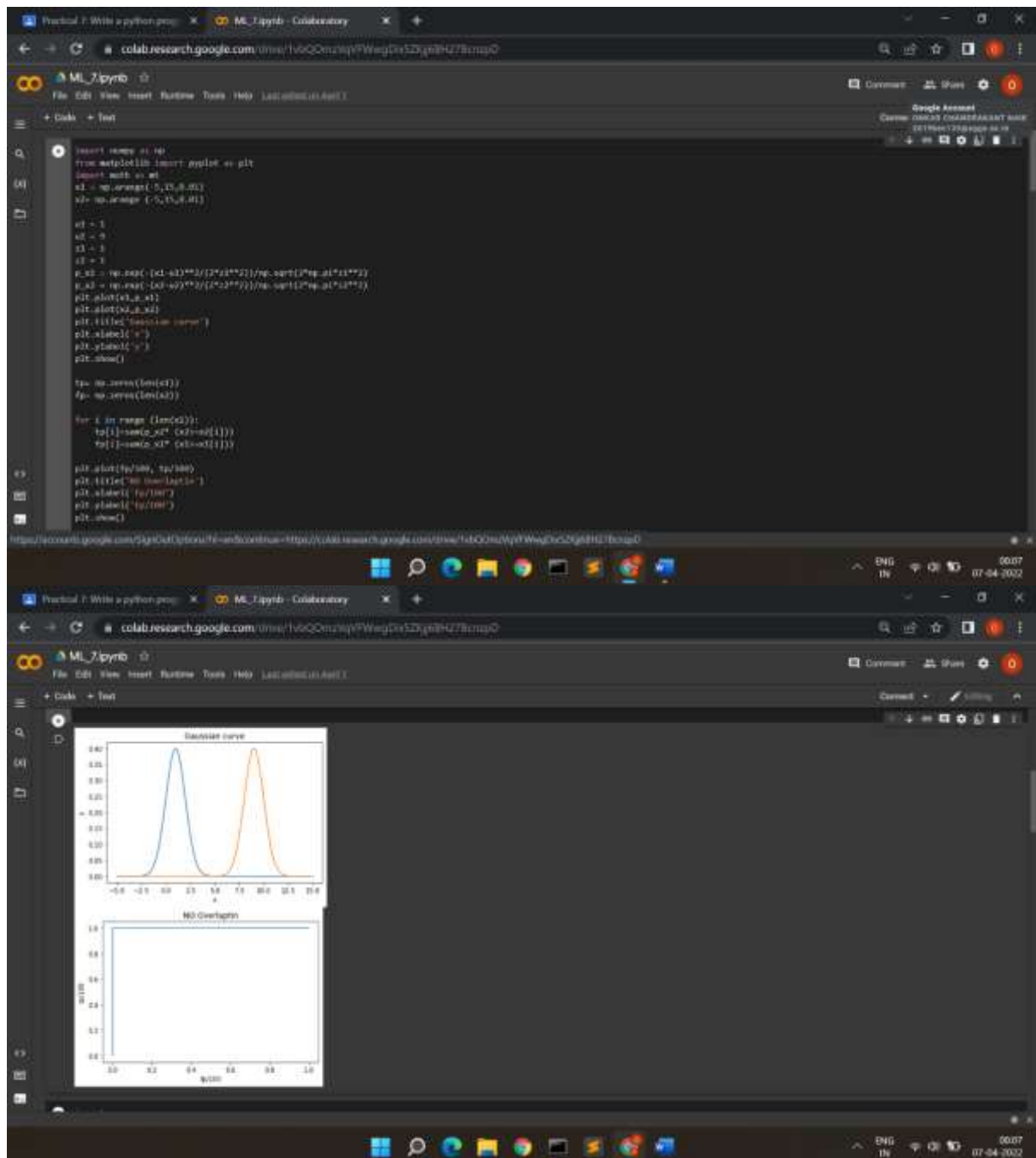
colab , Pycharm

THEORY :-

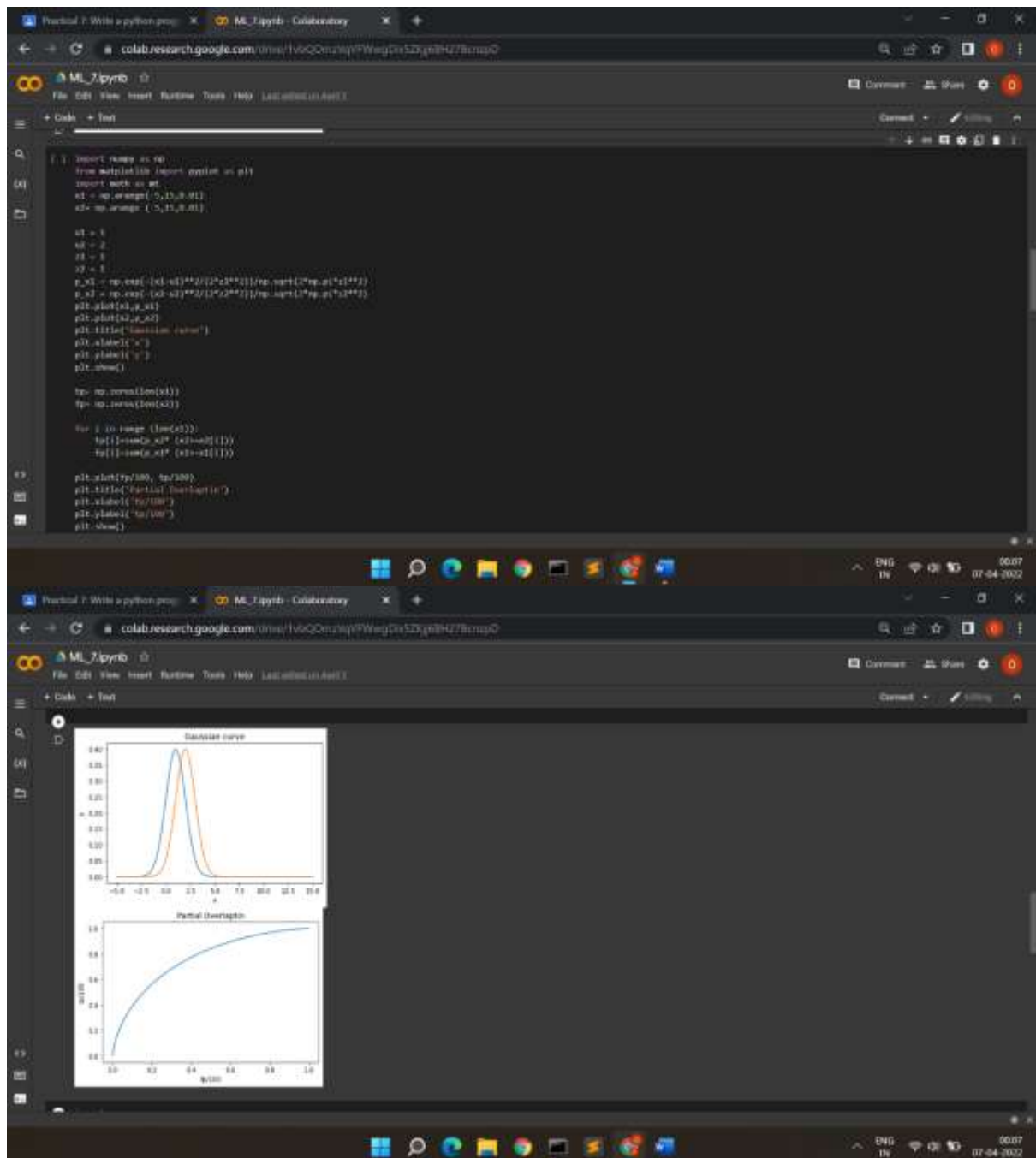
ROC curve, also known as Receiver Operating Characteristics Curve, is a metric used to measure the performance of a classifier model. The ROC curve depicts the rate of true positives with respect to the rate of false positives, therefore highlighting the sensitivity of the classifier model. The ROC is also known as a relative operating characteristic curve, as it is a comparison of two operating characteristics, the True Positive Rate and the False Positive Rate, as the criterion changes. An ideal classifier will have a ROC where the graph would hit a true positive rate of 100% with zero false positives. We generally measure how many correct positive classifications are being gained with an increment in the rate of false positives.

ROC curve can be used to select a threshold for a classifier, which maximizes the true positives and in turn minimizes the false positives. ROC Curves help determine the exact trade-off between the true positive rate and false-positive rate for a model using different measures of probability thresholds. ROC curves are more appropriate to be used when the observations present are balanced between each class. This method was first used in signal detection but is now also being used in many other areas such as medicine, radiology, natural hazards other than machine learning. A discrete classifier returns only the predicted class and gives a single point on the ROC space. But for probabilistic classifiers, which give a probability or score that reflects the degree to which an instance belongs to one class rather than another, we can create a curve by changing the threshold for the score.

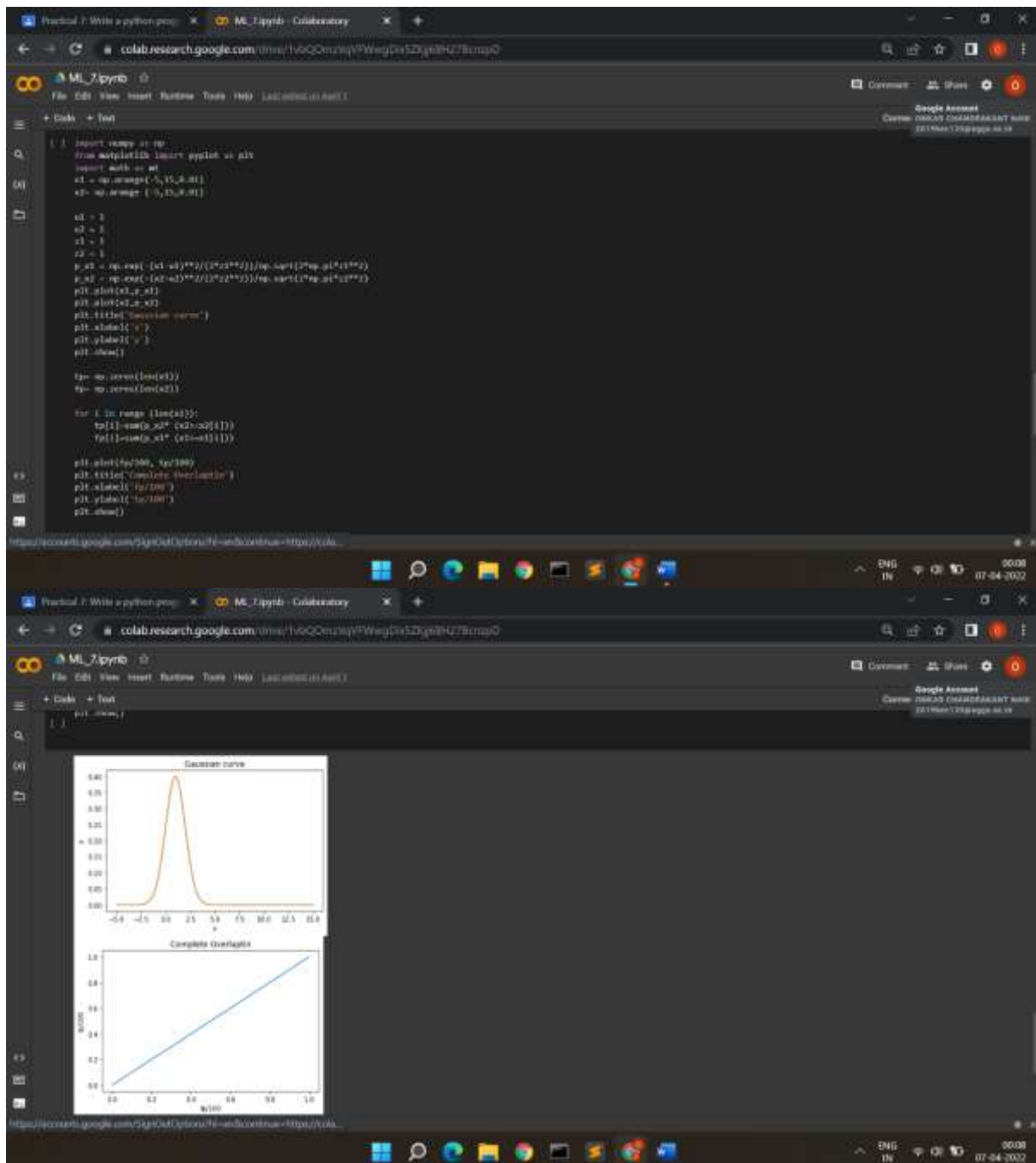
1) Plot the “No Overlap” condition of two Gaussian distributions. Also plot the ROC curve for same. Give proper labeling to plot.



2) Plot the “Partial Overlap” condition of two Gaussian distributions. Also plot the ROC curve for same. Give proper labeling to plot.



3) Plot the "Complete Overlap" condition of two Gaussian distributions. Also plot the ROC curve for same. Give proper labeling to plot.



CONCLUSION :- An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives

Practical-8

Name: Naik Onkar Chandrakant

Reg No : 2019BEC135

Roll No : A65

AIM :- Write a python program for Multivariate Regression Modeling

SOFTWARE :- Google colab , Pycharm

THEORY :-

Multivariate regression is a technique used to measure the degree to which the various independent variable and various dependent variables are linearly related to each other. The relation is said to be linear due to the correlation between the variables. Once the multivariate regression is applied to the dataset, this method is then used to predict the behaviour of the response variable based on its corresponding predictor variables.

Multivariate regression is commonly used as a supervised algorithm in machine learning, a model to predict the behaviour of dependent variables and multiple independent variables.

Characteristics of multivariate regression

- Multivariate regression allows one to have a different view of the relationship between various variables from all the possible angles.
- It helps you to predict the behaviour of the response variables depending on how the predictor variables move.
- Multivariate regression can be applied to various machine learning fields, economic, science and medical research studies.

1. Consider the attached dataset (Mult_reg.png) of power consumption containing (4 rows and 3 column). Define the function for computing the regression coefficients. Print out the coefficients.

```

import numpy as np

x=np.array([[18.5],[2.5],[1.8]])
y=np.array([11,10,11,5,9],[1,4,5,3],[1,6,8,1])
x=np.matmul(x.transpose(),x)
print(np.linalg.inv(x))
print(p)

x=np.matmul(x.transpose(),y)
print(q)
beta=np.matmul(p,q)
print('beta[0] = ',beta[0])
print('beta[1] = ',beta[1])
print('beta[2] = ',beta[2])

[[ 3.7519555e+01 -3.8521905e+00 -5.8817072e-01]
 [ -3.8521905e+00  3.8860540e-01  4.7977932e-07]
 [ -5.8817072e-01  4.7977932e-07  7.3683802e-03]]
[[ 7. ]
 [ 8.4]
 [132.7]]
beta[0] = [2.40701348]
beta[1] = [-4.20701687]
beta[2] = [0.40690815]
  
```

2. Define the function for computing energy consumption for given regression coefficients as inputs as follows. Print the output describing energy required (KW) for given temp and no. of persons.

I. temp = 7.5 and No. of persons in room = 131

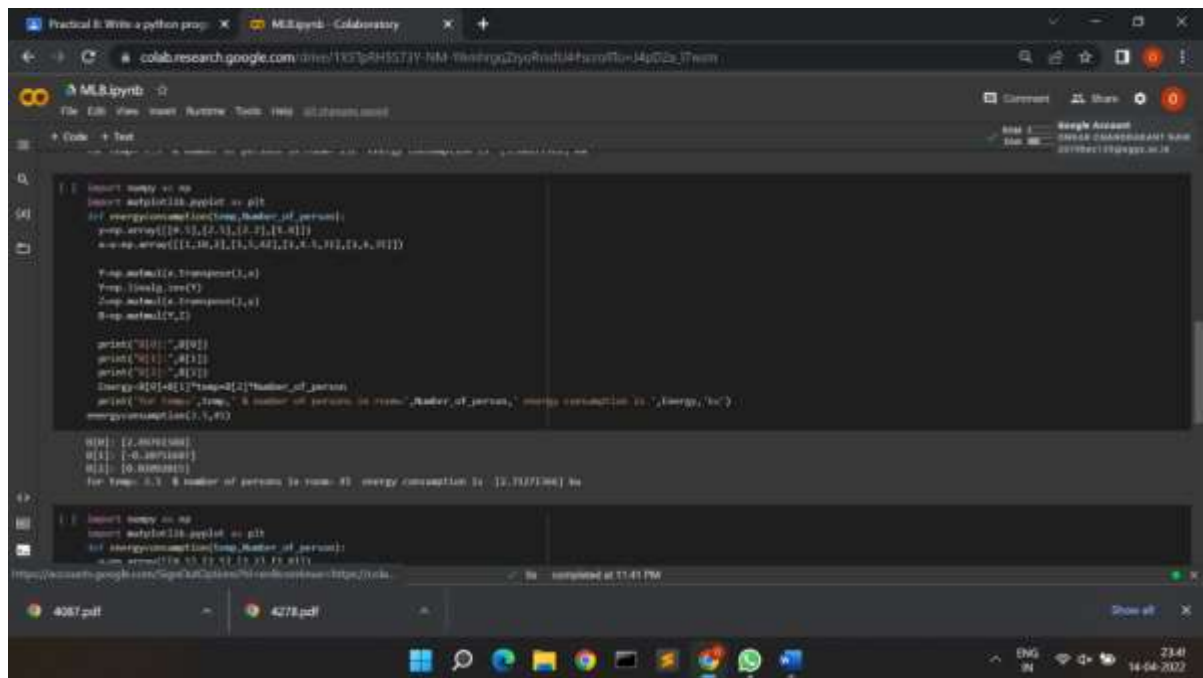
```

def energyconsumption(temp,number_of_person):
    x=np.array([[18.5],[2.5],[1.8]])
    y=np.array([11,10,11,5,9],[1,4,5,3],[1,6,8,1])
    x=np.matmul(x.transpose(),x)
    y=np.linalg.inv(y)
    z=np.matmul(x.transpose(),y)
    beta=np.matmul(y,z)

    print('beta[0] = ',beta[0])
    print('beta[1] = ',beta[1])
    print('beta[2] = ',beta[2])
    energy=beta[0]+beta[1]*temp+beta[2]*number_of_person
    print('for temp ',temp,' & number of persons in room ',number_of_person,' energy consumption is ',energy,'kw')
    energyconsumption(7.5,131)

beta[0] = [2.40701348]
beta[1] = [-4.20701687]
beta[2] = [0.40690815]
for temp 7.5 & number of persons in room 131, energy consumption is [2.62177422] kw
  
```

II. temp = 3.5 and No. of persons in room = 45



```
[ ] import numpy as np
import matplotlib.pyplot as plt

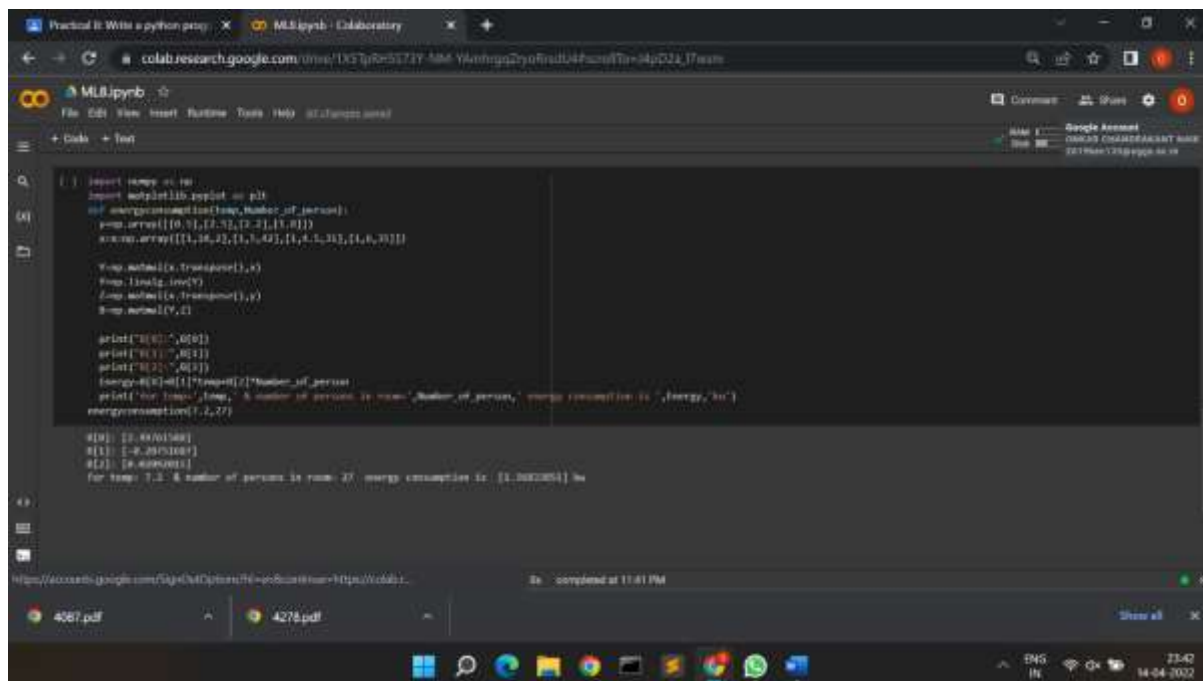
def energyconsumption(temp,number_of_persons):
    temp.array([10,12,14,16,18,20])
    no_of_persons.array([1,2,3,4,5,6,7,8,9,10])

    temp.autofmt(x.transpose(),x)
    temp.linlog.yes(y)
    temp.autofmt(x.transpose(),y)
    temp.autofmt(y,z)

    print("X[0]:",X[0])
    print("X[1]:",X[1])
    print("X[2]:",X[2])
    temp=X[0]+X[1]*temp+X[2]*number_of_persons
    print("for temp:",temp," & number of persons in room:",number_of_persons," energy consumption is :",energy,"kWh")
    energyconsumption(3.5,45)

X[0]: [2.8091588]
X[1]: [-0.0071081]
X[2]: [0.0000015]
for temp: 3.5 & number of persons in room: 45 energy consumption is : [12.71271961] kWh
```

III. temp = 7.2 and No. of persons in room = 27



```
[ ] import numpy as np
import matplotlib.pyplot as plt

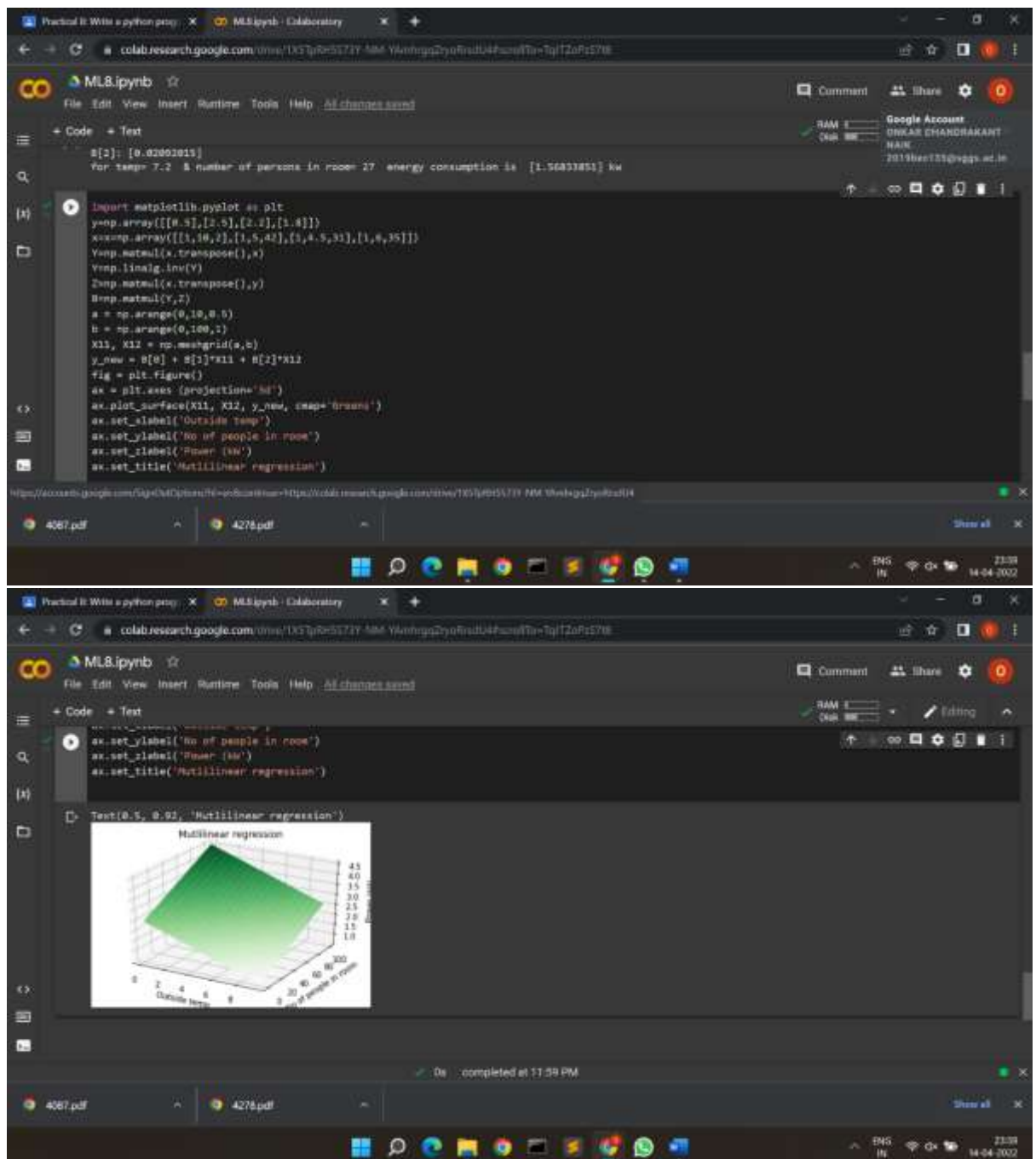
def energyconsumption(temp,number_of_persons):
    temp.array([10,12,14,16,18,20])
    no_of_persons.array([1,2,3,4,5,6,7,8,9,10])

    temp.autofmt(x.transpose(),x)
    temp.linlog.yes(y)
    temp.autofmt(x.transpose(),y)
    temp.autofmt(y,z)

    print("X[0]:",X[0])
    print("X[1]:",X[1])
    print("X[2]:",X[2])
    temp=X[0]+X[1]*temp+X[2]*number_of_persons
    print("for temp:",temp," & number of persons in room:",number_of_persons," energy consumption is :",energy,"kWh")
    energyconsumption(7.2,27)

X[0]: [2.8091588]
X[1]: [-0.0071081]
X[2]: [0.0000015]
for temp: 7.2 & number of persons in room: 27 energy consumption is : [11.3022854] kWh
```

3. Plot the 3D graph for power consumption data using multivariate regression coefficients. (fit the hyperplane)



CONCLUSION :- Multivariate Regression is a supervised machine learning algorithm involving multiple data variables for analysis. Multivariate regression is an extension of multiple regression with one dependent variable and multiple independent variables. Based on the number of independent variables, we try to predict the output.

Practical-9

Name: Naik Onkar Chandrakant

Reg No : 2019BEC135

Roll No : A65

AIM :- Write a python program for Linear Regression using Gradient Descent algorithm.

REQUIRED SOFTWARE :- Google

Colab , Pycharm

THEORY:-
Gradient descent is a name for a generic class of computer algorithms which minimize a function. These algorithms achieve this end by starting with initial parameter values and iteratively moving towards a set of parameter values that minimize some cost function or metric—that's the descent part. The movement toward best-fit is achieved by taking the derivative of the variable or variables involved, towards the direction with the lowest (calculus-defined) gradient—that's the gradient part.

Gradient descent is an important concept in computer science, and an illustrative example of why CS has kind of overtaken statistics in importance when it comes to machine learning: it's a general-purpose tool that can be used to "brute force" an optimal solution in a wide range of scenarios, which doesn't have the elegance, closed-form solution, and unfortunate sheer mathematical inpalatability of a statistical solution.

Ordinary linear regression is a good and simple way of demonstrating how gradient descent works. We start with some error function. We could use any metric we want, but in OLS the obvious one is the residual sum of squares.

Given a sequence of points, y_i , and a sequence of points predicted by our model, \hat{y}_i , RSS is:

$$\text{error}(m, b) = \sum_{i=1}^n (y_i - \hat{y}_i)$$

Our objective is to minimize this value. Inserting our linear regression model in for the \hat{y}_i predictions, and assuming (for the sake of simplicity) that we're doing regression on only one variable, we get:

$$\text{RSS} = \sum_{i=1}^n (y_i - (mx_i + b))^2$$

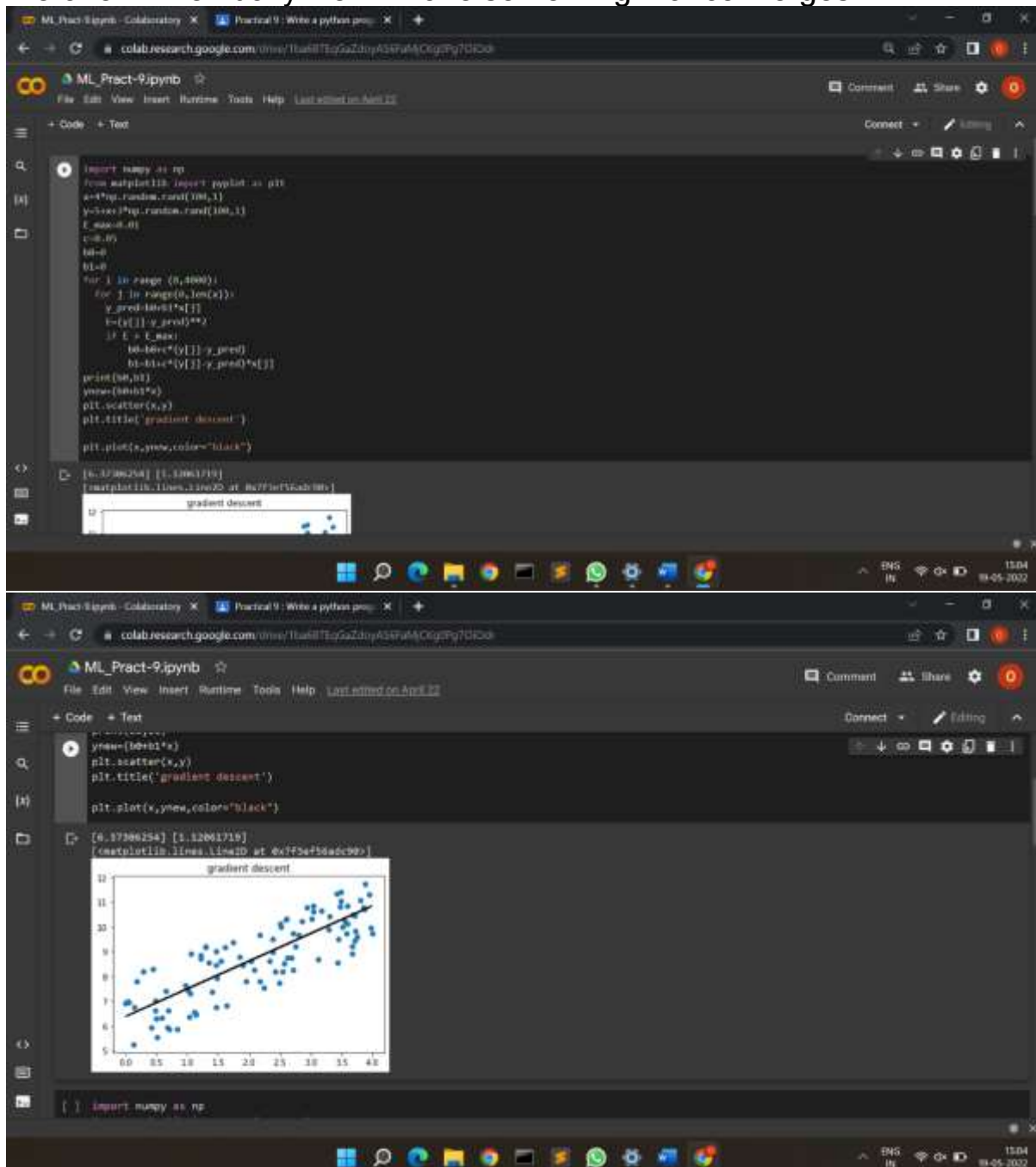
$(m x_i + b)^2$ Where b is the intercept and m is the slope of the line of best fit.

Now we need to take the gradient. Since this is an equation of two variables

(b and m) the gradient will consist of two partial derivatives. Hence the gradient is:

$$\left(\frac{\partial}{\partial b}(\text{RSS}), \frac{\partial}{\partial m}(\text{RSS}) \right) = \left(-2 \sum_{i=1}^n (y_i - (m x_i + b)), -2 \sum_{i=1}^n x_i (y_i - (m x_i + b)) \right)$$

To solve, take a step in the negative gradient direction every iteration. Eventually we will have something that converges.



```
ML Pract 9.ipynb - Colaboratory
Practical 9: Write a python prog.
colab.research.google.com/drive/7ba68T1qGaZduyAS8faMCKg0lg7QDQd

ML_Pract-9.ipynb
File Edit View Insert Runtime Tools Help Last edited on April 22

+ Code + Text
Connect Editing

import numpy as np
from matplotlib import pyplot as plt
def gradientdescent(x,y):
    E_max=0.01
    c=0.05
    b0=0
    b1=0
    for i in range(0,4000):
        for j in range(0,len(x)):
            y_pred=b0+b1*x[j]
            E=(y[j]-y_pred)**2
            if E > E_max:
                b0=b0+c*(y[j]-y_pred)
                b1=b1+c*(y[j]-y_pred)*x[j]
        print(b0,b1)
    ynew=(b0+b1*x)
    plt.scatter(x,y)
    plt.title('gradient descent')
    plt.plot(x,ynew,color="black")

x=np.random.rand(100,1)
y=5*x+3*np.random.rand(100,1)
gradientdescent(x,y)
```

```
ML Pract 9.ipynb - Colaboratory
Practical 9: Write a python prog.
colab.research.google.com/drive/7ba68T1qGaZduyAS8faMCKg0lg7QDQd

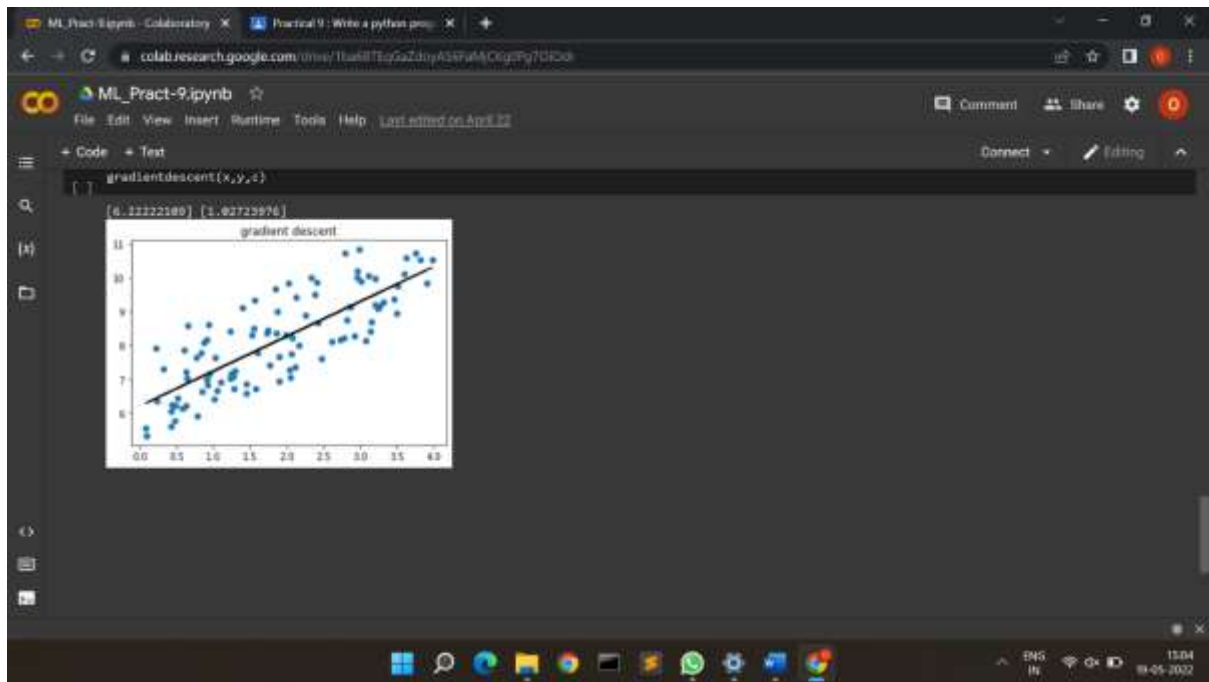
ML_Pract-9.ipynb
File Edit View Insert Runtime Tools Help Last edited on April 22

+ Code + Text
Connect Editing

from matplotlib import pyplot as plt
def gradientdescent(x,y,c):
    E_max=0.01

    E=0
    b0=0
    b1=0
    for i in range(0,4000):
        for j in range(0,len(x)):
            y_pred=b0+b1*x[j]
            E=(y[j]-y_pred)**2
            if E > E_max:
                b0=b0+c*(y[j]-y_pred)
                b1=b1+c*(y[j]-y_pred)*x[j]
        print(b0,b1)
    ynew=(b0+b1*x)
    plt.scatter(x,y)
    plt.title('gradient descent')
    plt.plot(x,ynew,color="black")

x=np.random.rand(100,1)
y=5*x+3*np.random.rand(100,1)
c=0.2
gradientdescent(x,y,c)
```



CONCLUSION :- The goal of the gradient descent algorithm is to minimize the given function (say cost function). To achieve this goal, it performs two steps iteratively is to Compute the gradient (slope), the first order derivative of the function at that point and to Make a step (move) in the direction opposite to the gradient, opposite direction of slope increase from the current point by alpha times the gradient at that point

Practical-10

Name: Naik Onkar Chandrakant

Reg No : 2019BEC135

Roll No : A65

AIM : - Write a python program to implement Parzen Window Density estimation for any randomly generated signal.

REQUIRED SOFTWARE:- Google

colab ,

HEORY :-

Parzen windows classification is a technique for nonparametric density estimation, which can also be used for classification. Using a given kernel function, the technique approximates a given training set distribution via a linear combination of kernels centered on the observed points. In this work, we separately approximate densities for each of the two classes, and we assign a test point to the class with maximal posterior probability.

The resulting algorithm is extremely simple and closely related to support vector machines. The decision function is

$$f(\mathbf{X}) = \text{sign}(\sum y_i K(\mathbf{X}_i, \mathbf{X})),$$

where the kernel function K is the radial basis function of Equation 2, without normalization applied to the inputs. As for the radial basis SVM, a constant is added to the kernel function whenever the two inputs are identical (Equation 3).

The Parzen windows classification algorithm does not require any training phase; however, the lack of sparseness makes the test phase quite slow. Furthermore, although asymptotical

convergence guarantees on the performance of Parzen windows classifiers exist [Duda and Hart, 1973], no such guarantees exist for finite sample sizes.

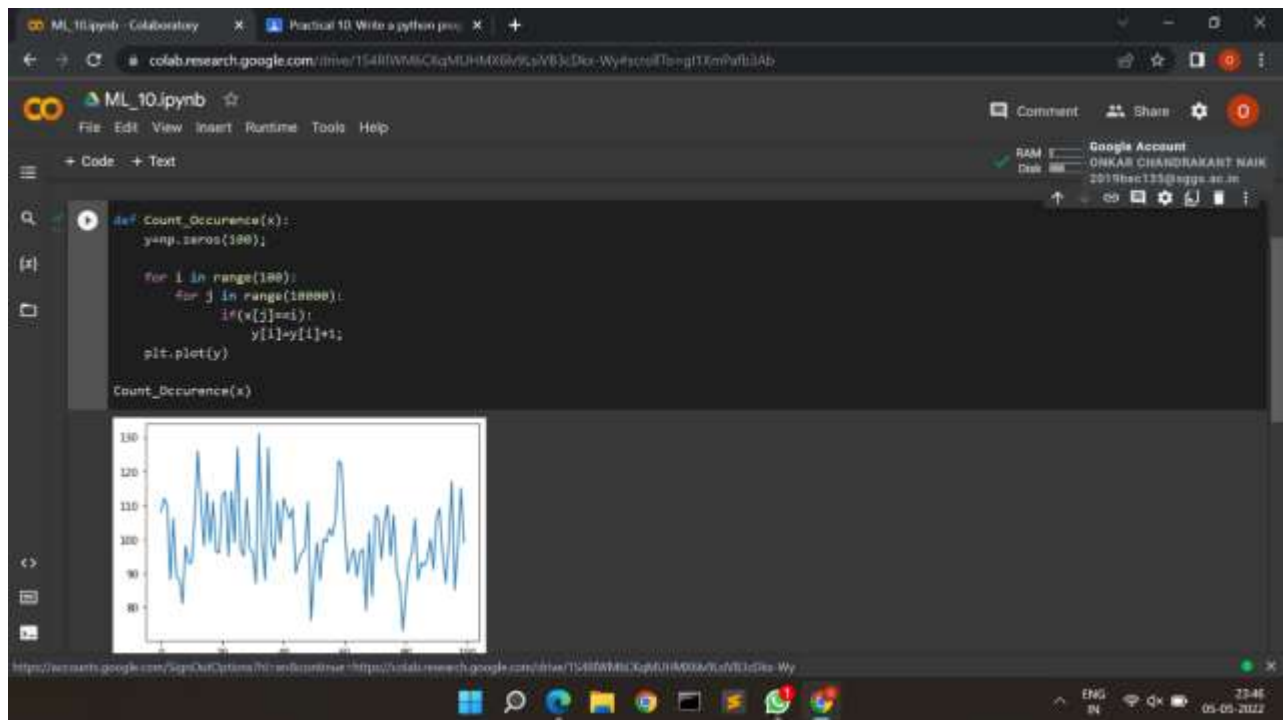
Parzen windows can be regarded as a generalization of k -nearest neighbor techniques. Rather than choosing the k nearest neighbors of a test point and labelling the test point with the weighted majority of its neighbors' votes, one can consider all points in the voting scheme and assign their weight by means of the kernel function. With Gaussian kernels, the weight decreases exponentially with the square of the distance, so far away points are practically irrelevant. The width σ of the Gaussian determines the relative weighting of near and far points. Tuning this parameter controls the predictive power of the system. We have empirically optimized the value of σ .

1. Generate a random signal of size 10,000 in the range 0 to 99.

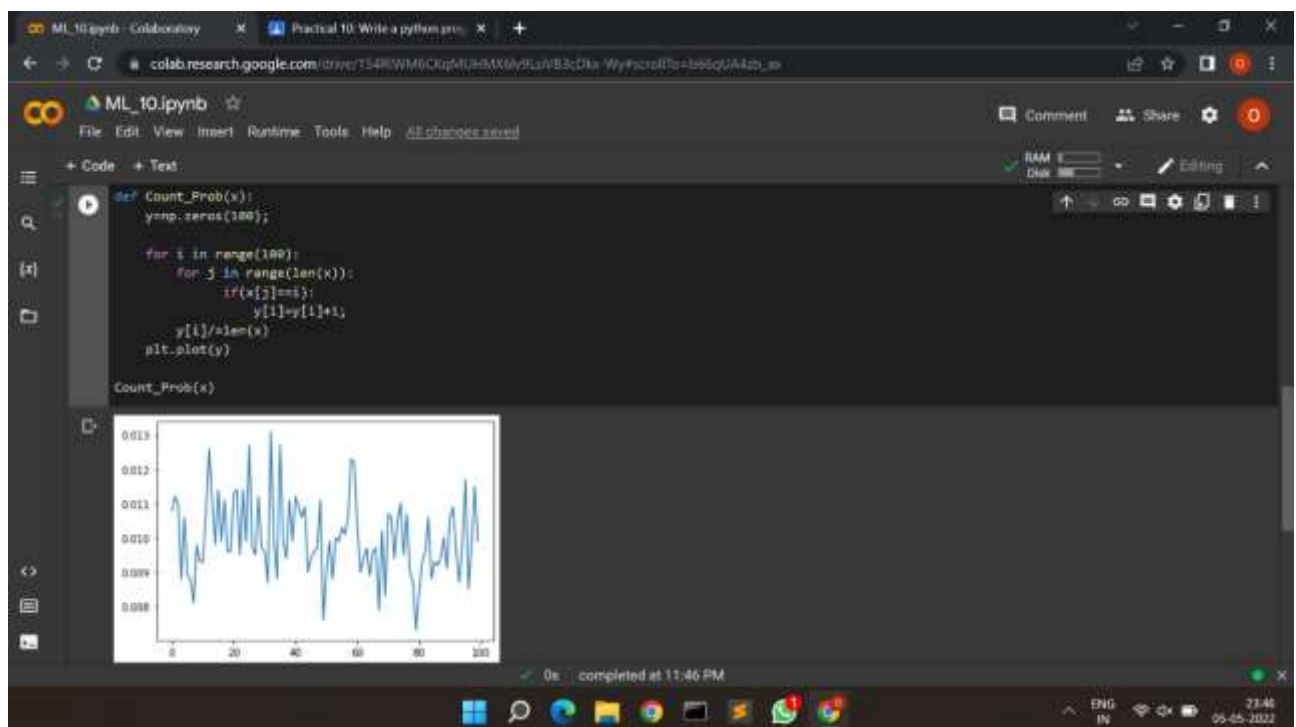
```
import numpy as np
import matplotlib.pyplot as plt
x=np.random.randint(100,size=10000)
print(x)
```

[90 28 87 ... 97 84 54]

2.Count the occurrence of each sample from signal. Plot the occurrence of each sample.



3.Plot the probability of each sample.



The screenshot shows a Google Colab notebook with the following code:

```
def Person_window(x,y):
    y=y.series(100),

    for i in range(100):
        for j in range(100):
            for k in range(10):
                y[i][j]=y[i][j]+k
                y[i][j]=y[i][j]+k
                y[i][j]=(y[i][j]+k)/k
            print(y)

    Person_window(x,y)
```

The output of the function is a 100x100x100 array of results, displayed as a 10x10 grid of values. The values are small floats, ranging from approximately 0.00000000 to 0.00000000.

At the bottom of the notebook, the URL is shown: https://colab.research.google.com/github/15481WMCup/MUHMXXB9LWELDx_WyWczuTf-wk1JAUJockF

The screenshot shows a Google Colab notebook with the following code:

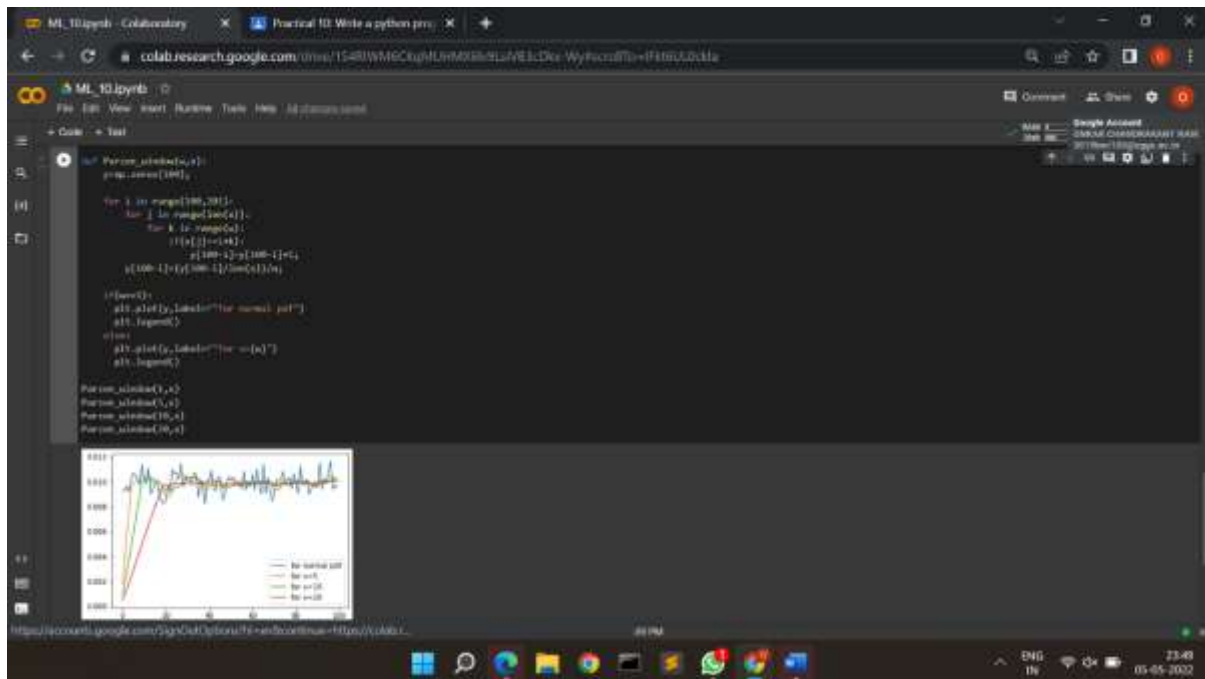
```
def Perceptron_weights(w,x):  
    y=w*x*cos(100)  
  
    for i in range(100):  
        for j in range(len(x)):  
            for k in range(x):  
                t(x[j])+w(k)  
                x(j)+w(j)+2  
                y(j)=(t(j)+m(x))/2  
  
    plot.plot(y,label=i**"for w={j}"  
    plt.legend()
```

The code is then called for different input values:

```
Perceptron_weights(1,x)  
Perceptron_weights(2,x)  
Perceptron_weights(3,x)  
Perceptron_weights(4,x)  
Perceptron_weights(5,x)  
Perceptron_weights(10,x)
```

The resulting plot shows the output of the function for these inputs. The legend indicates the input values: w=1 (blue), w=2 (orange), w=3 (green), w=4 (red), w=5 (purple), and w=10 (brown). The plot shows that the output increases as the input increases, and the slope of the line increases with the input value.

6. Generate a random data of size 20,000 in the range 100 to 200. Plot the normal pdf of data and Parzen window density function of data for various window size (V=1,5,10,20).



CONCLUSION :- It is used to derive a density function $f(X)$ is used to implement a Bayes Classifier. When we have a new sample feature $f(x)$ and when there is a need to compute the value of the class conditional densities $f(x)$ is used $f(X)$ takes sample input data value and returns the density estimate of the given data sample.

Practical-11

Name: Naik Onkar Chandrakant

Reg No : 2019BEC135

Roll No : A65

AIM : - Write a python program for implementation of K-nearest neighbor algorithm.

REQUIRED SOFTWARE:- Google

colab , Pycharm THEORY :-

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

1. Classify the test sample for given 1D data using knn with k values as 3, 5 and 7. $w1 = [1.4, 3.6, 4.7, 2.8, 7.4, 5.3, 6.7, 3.2, 4.8]$, $w2 = [4.5, 3.7, 2.4, 7.8, 9.3, 7.0, 6.4, 8.3, 5.1]$, $x_test = 3.3, 1.9, 8.6 \text{ \& } 4.1$

```

class0 = 0
w1 = [1.4, 3.6, 4.7, 2.8, 7.4, 5.3, 6.7, 3.2, 4.8]
w2 = [4.5, 3.7, 2.4, 7.8, 9.3, 7.0, 6.4, 8.3, 5.1]
x_test = [3.3, 1.9, 8.6, 4.1]

def dist(x1, x2):
    return sqrt(sum((x1[i] - x2[i])**2 for i in range(len(x1))))

def knn(x, k):
    dists = []
    for i in range(len(w1)):
        dists.append((dist(x, w1[i]), i))
    for i in range(len(w2)):
        dists.append((dist(x, w2[i]), i))
    dists.sort()
    k_nearest = dists[:k]
    class0_count = 0
    class1_count = 0
    for i in range(k):
        if k_nearest[i][1] == 0:
            class0_count += 1
        else:
            class1_count += 1
    if class0_count > class1_count:
        return 0
    else:
        return 1

print("Class 0 for the value 'x_test' is: ", x_test)
print("Class 1 for the value 'x_test' is: ", x_test)

```

```

print("Class 0 for the value 'x_test' is: ", x_test)

w1 = [1.4, 3.6, 4.7, 2.8, 7.4, 5.3, 6.7, 3.2, 4.8]
w2 = [4.5, 3.7, 2.4, 7.8, 9.3, 7.0, 6.4, 8.3, 5.1]

def dist(x1, x2):
    return sqrt(sum((x1[i] - x2[i])**2 for i in range(len(x1))))

def knn(x, k):
    dists = []
    for i in range(len(w1)):
        dists.append((dist(x, w1[i]), i))
    for i in range(len(w2)):
        dists.append((dist(x, w2[i]), i))
    dists.sort()
    k_nearest = dists[:k]
    class0_count = 0
    class1_count = 0
    for i in range(k):
        if k_nearest[i][1] == 0:
            class0_count += 1
        else:
            class1_count += 1
    if class0_count > class1_count:
        return 0
    else:
        return 1

for k in [3, 5, 7]:
    print("k = ", k)
    for i in range(len(x_test)):
        print("x_test = ", x_test[i], " predicted class = ", knn(x_test[i], k))

```

2. Classify the test sample for given 2D data using knn with k values as 3, 5 and 7. $w1 = \{[1, 1.7], [1, 2], [2, 3], [1.5, 3.1], [2.4, 3.6], [1.8, 2.7]\}$, $w2 = \{[1, 1.5], [1.5, 2], [0.5, 1], [0.9, 1.6], [3, 4.1], [2, 2.7]\}$, $x_test = [1.7, 3.2], [1.2, 4.5], [2.6, 3.8] \& [0.8, 2.5]$

The first screenshot shows the initial code in a Google Colab notebook. The code defines a function `findMaxSum` that takes an array `arr` and its size `n` as input. It calculates the prefix sum of the array and then iterates over all possible subarrays to find the maximum sum. The code is as follows:

```
int findMaxSum(int arr[], int n) {
    int sum = 0;
    int maxSum = 0;
    for (int i = 0; i < n; i++) {
        sum += arr[i];
        maxSum = max(maxSum, sum);
    }
    return maxSum;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
    findMaxSum(arr, n);
    return 0;
}
```

The second screenshot shows the completed code. The code defines a function `findMaxSum` that takes an array `arr` and its size `n` as input. It calculates the prefix sum of the array and then iterates over all possible subarrays to find the maximum sum. The code is as follows:

```
int findMaxSum(int arr[], int n) {
    int sum = 0;
    int maxSum = 0;
    for (int i = 0; i < n; i++) {
        sum += arr[i];
        maxSum = max(maxSum, sum);
    }
    return maxSum;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int n = sizeof(arr) / sizeof(arr[0]);
    findMaxSum(arr, n);
    return 0;
}
```

The top screenshot shows a Google Colab notebook titled "ML_Practice11.ipynb". The code is as follows:

```

1 # 1. 데이터 로드
2 # 2. 데이터 전처리
3 # 3. 모델 학습
4 # 4. 모델 평가
5 # 5. 모델 배포
6 # 6. 모델 모니터링
7 # 7. 모델 업데이트
8 # 8. 모델 삭제
9 # 9. 모델 백업
10 # 10. 모델 복구
11 # 11. 모델 테스트
12 # 12. 모델 배포
13 # 13. 모델 모니터링
14 # 14. 모델 업데이트
15 # 15. 모델 삭제
16 # 16. 모델 백업
17 # 17. 모델 복구
18 # 18. 모델 테스트
19 # 19. 모델 배포
20 # 20. 모델 모니터링
21 # 21. 모델 업데이트
22 # 22. 모델 삭제
23 # 23. 모델 백업
24 # 24. 모델 복구
25 # 25. 모델 테스트
26 # 26. 모델 배포
27 # 27. 모델 모니터링
28 # 28. 모델 업데이트
29 # 29. 모델 삭제
30 # 30. 모델 백업
31 # 31. 모델 복구
32 # 32. 모델 테스트
33 # 33. 모델 배포
34 # 34. 모델 모니터링
35 # 35. 모델 업데이트
36 # 36. 모델 삭제
37 # 37. 모델 백업
38 # 38. 모델 복구
39 # 39. 모델 테스트
40 # 40. 모델 배포
41 # 41. 모델 모니터링
42 # 42. 모델 업데이트
43 # 43. 모델 삭제
44 # 44. 모델 백업
45 # 45. 모델 복구
46 # 46. 모델 테스트
47 # 47. 모델 배포
48 # 48. 모델 모니터링
49 # 49. 모델 업데이트
50 # 50. 모델 삭제
51 # 51. 모델 백업
52 # 52. 모델 복구
53 # 53. 모델 테스트
54 # 54. 모델 배포
55 # 55. 모델 모니터링
56 # 56. 모델 업데이트
57 # 57. 모델 삭제
58 # 58. 모델 백업
59 # 59. 모델 복구
60 # 60. 모델 테스트
61 # 61. 모델 배포
62 # 62. 모델 모니터링
63 # 63. 모델 업데이트
64 # 64. 모델 삭제
65 # 65. 모델 백업
66 # 66. 모델 복구
67 # 67. 모델 테스트
68 # 68. 모델 배포
69 # 69. 모델 모니터링
70 # 70. 모델 업데이트
71 # 71. 모델 삭제
72 # 72. 모델 백업
73 # 73. 모델 복구
74 # 74. 모델 테스트
75 # 75. 모델 배포
76 # 76. 모델 모니터링
77 # 77. 모델 업데이트
78 # 78. 모델 삭제
79 # 79. 모델 백업
80 # 80. 모델 복구
81 # 81. 모델 테스트
82 # 82. 모델 배포
83 # 83. 모델 모니터링
84 # 84. 모델 업데이트
85 # 85. 모델 삭제
86 # 86. 모델 백업
87 # 87. 모델 복구
88 # 88. 모델 테스트
89 # 89. 모델 배포
90 # 90. 모델 모니터링
91 # 91. 모델 업데이트
92 # 92. 모델 삭제
93 # 93. 모델 백업
94 # 94. 모델 복구
95 # 95. 모델 테스트
96 # 96. 모델 배포
97 # 97. 모델 모니터링
98 # 98. 모델 업데이트
99 # 99. 모델 삭제
100 # 100. 모델 백업
101 # 101. 모델 복구
102 # 102. 모델 테스트
103 # 103. 모델 배포
104 # 104. 모델 모니터링
105 # 105. 모델 업데이트
106 # 106. 모델 삭제
107 # 107. 모델 백업
108 # 108. 모델 복구
109 # 109. 모델 테스트
110 # 110. 모델 배포
111 # 111. 모델 모니터링
112 # 112. 모델 업데이트
113 # 113. 모델 삭제
114 # 114. 모델 백업
115 # 115. 모델 복구
116 # 116. 모델 테스트
117 # 117. 모델 배포
118 # 118. 모델 모니터링
119 # 119. 모델 업데이트
120 # 120. 모델 삭제
121 # 121. 모델 백업
122 # 122. 모델 복구
123 # 123. 모델 테스트
124 # 124. 모델 배포
125 # 125. 모델 모니터링
126 # 126. 모델 업데이트
127 # 127. 모델 삭제
128 # 128. 모델 백업
129 # 129. 모델 복구
130 # 130. 모델 테스트
131 # 131. 모델 배포
132 # 132. 모델 모니터링
133 # 133. 모델 업데이트
134 # 134. 모델 삭제
135 # 135. 모델 백업
136 # 136. 모델 복구
137 # 137. 모델 테스트
138 # 138. 모델 배포
139 # 139. 모델 모니터링
140 # 140. 모델 업데이트
141 # 141. 모델 삭제
142 # 142. 모델 백업
143 # 143. 모델 복구
144 # 144. 모델 테스트
145 # 145. 모델 배포
146 # 146. 모델 모니터링
147 # 147. 모델 업데이트
148 # 148. 모델 삭제
149 # 149. 모델 백업
150 # 150. 모델 복구
151 # 151. 모델 테스트
152 # 152. 모델 배포
153 # 153. 모델 모니터링
154 # 154. 모델 업데이트
155 # 155. 모델 삭제
156 # 156. 모델 백업
157 # 157. 모델 복구
158 # 158. 모델 테스트
159 # 159. 모델 배포
160 # 160. 모델 모니터링
161 # 161. 모델 업데이트
162 # 162. 모델 삭제
163 # 163. 모델 백업
164 # 164. 모델 복구
165 # 165. 모델 테스트
166 # 166. 모델 배포
167 # 167. 모델 모니터링
168 # 168. 모델 업데이트
169 # 169. 모델 삭제
170 # 170. 모델 백업
171 # 171. 모델 복구
172 # 172. 모델 테스트
173 # 173. 모델 배포
174 # 174. 모델 모니터링
175 # 175. 모델 업데이트
176 # 176. 모델 삭제
177 # 177. 모델 백업
178 # 178. 모델 복구
179 # 179. 모델 테스트
180 # 180. 모델 배포
181 # 181. 모델 모니터링
182 # 182. 모델 업데이트
183 # 183. 모델 삭제
184 # 184. 모델 백업
185 # 185. 모델 복구
186 # 186. 모델 테스트
187 # 187. 모델 배포
188 # 188. 모델 모니터링
189 # 189. 모델 업데이트
190 # 190. 모델 삭제
191 # 191. 모델 백업
192 # 192. 모델 복구
193 # 193. 모델 테스트
194 # 194. 모델 배포
195 # 195. 모델 모니터링
196 # 196. 모델 업데이트
197 # 197. 모델 삭제
198 # 198. 모델 백업
199 # 199. 모델 복구
200 # 200. 모델 테스트
201 # 201. 모델 배포
202 # 202. 모델 모니터링
203 # 203. 모델 업데이트
204 # 204. 모델 삭제
205 # 205. 모델 백업
206 # 206. 모델 복구
207 # 207. 모델 테스트
208 # 208. 모델 배포
209 # 209. 모델 모니터링
210 # 210. 모델 업데이트
211 # 211. 모델 삭제
212 # 212. 모델 백업
213 # 213. 모델 복구
214 # 214. 모델 테스트
215 # 215. 모델 배포
216 # 216. 모델 모니터링
217 # 217. 모델 업데이트
218 # 218. 모델 삭제
219 # 219. 모델 백업
220 # 220. 모델 복구
221 # 221. 모델 테스트
222 # 222. 모델 배포
223 # 223. 모델 모니터링
224 # 224. 모델 업데이트
225 # 225. 모델 삭제
226 # 226. 모델 백업
227 # 227. 모델 복구
228 # 228. 모델 테스트
229 # 229. 모델 배포
230 # 230. 모델 모니터링
231 # 231. 모델 업데이트
232 # 232. 모델 삭제
233 # 233. 모델 백업
234 # 234. 모델 복구
235 # 235. 모델 테스트
236 # 236. 모델 배포
237 # 237. 모델 모니터링
238 # 238. 모델 업데이트
239 # 239. 모델 삭제
240 # 240. 모델 백업
241 # 241. 모델 복구
242 # 242. 모델 테스트
243 # 243. 모델 배포
244 # 244. 모델 모니터링
245 # 245. 모델 업데이트
246 # 246. 모델 삭제
247 # 247. 모델 백업
248 # 248. 모델 복구
249 # 249. 모델 테스트
250 # 250. 모델 배포
251 # 251. 모델 모니터링
252 # 252. 모델 업데이트
253 # 253. 모델 삭제
254 # 254. 모델 백업
255 # 255. 모델 복구
256 # 256. 모델 테스트
257 # 257. 모델 배포
258 # 258. 모델 모니터링
259 # 259. 모델 업데이트
260 # 260. 모델 삭제
261 # 261. 모델 백업
262 # 262. 모델 복구
263 # 263. 모델 테스트
264 # 264. 모델 배포
265 # 265. 모델 모니터링
266 # 266. 모델 업데이트
267 # 267. 모델 삭제
268 # 268. 모델 백업
269 # 269. 모델 복구
270 # 270. 모델 테스트
271 # 271. 모델 배포
272 # 272. 모델 모니터링
273 # 273. 모델 업데이트
274 # 274. 모델 삭제
275 # 275. 모델 백업
276 # 276. 모델 복구
277 # 277. 모델 테스트
278 # 278. 모델 배포
279 # 279. 모델 모니터링
280 # 280. 모델 업데이트
281 # 281. 모델 삭제
282 # 282. 모델 백업
283 # 283. 모델 복구
284 # 284. 모델 테스트
285 # 285. 모델 배포
286 # 286. 모델 모니터링
287 # 287. 모델 업데이트
288 # 288. 모델 삭제
289 # 289. 모델 백업
290 # 290. 모델 복구
291 # 291. 모델 테스트
292 # 292. 모델 배포
293 # 293. 모델 모니터링
294 # 294. 모델 업데이트
295 # 295. 모델 삭제
296 # 296. 모델 백업
297 # 297. 모델 복구
298 # 298. 모델 테스트
299 # 299. 모델 배포
300 # 300. 모델 모니터링
301 # 301. 모델 업데이트
302 # 302. 모델 삭제
303 # 303. 모델 백업
304 # 304. 모델 복구
305 # 305. 모델 테스트
306 # 306. 모델 배포
307 # 307. 모델 모니터링
308 # 308. 모델 업데이트
309 # 309. 모델 삭제
310 # 310. 모델 백업
311 # 311. 모델 복구
312 # 312. 모델 테스트
313 # 313. 모델 배포
314 # 314. 모델 모니터링
315 # 315. 모델 업데이트
316 # 316. 모델 삭제
317 # 317. 모델 백업
318 # 318. 모델 복구
319 # 319. 모델 테스트
320 # 320. 모델 배포
321 # 321. 모델 모니터링
322 # 322. 모델 업데이트
323 # 323. 모델 삭제
324 # 324. 모델 백업
325 # 325. 모델 복구
326 # 326. 모델 테스트
327 # 327. 모델 배포
328 # 328. 모델 모니터링
329 # 329. 모델 업데이트
330 # 330. 모델 삭제
331 # 331. 모델 백업
332 # 332. 모델 복구
333 # 333. 모델 테스트
334 # 334. 모델 배포
335 # 335. 모델 모니터링
336 # 336. 모델 업데이트
337 # 337. 모델 삭제
338 # 338. 모델 백업
339 # 339. 모델 복구
340 # 340. 모델 테스트
341 # 341. 모델 배포
342 # 342. 모델 모니터링
343 # 343. 모델 업데이트
344 # 344. 모델 삭제
345 # 345. 모델 백업
346 # 346. 모델 복구
347 # 34
```

CONCLUSION :- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K-NN algorithm.

Practical-12

Naik Onkar Chandrakant

A65

2019BEC135

AIM :- Write a python program to partition the data into two groups using K-means clustering algorithm.

REQUIRED SOFTWARE :- Google colab ,

PycharmTHEORY :-

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science.

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if $K=2$, there will be two clusters, and for $K=3$, there will be three clusters, and so on.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.

- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

1. $X = \{[3, 1], [4, 2], [1, 1], [6, 3], [2, 6], [-1, 2]\}$, $U = \{[0, 0, 1, 1, 0, 0], [1, 1, 0, 0, 1, 1]\}$

```

def distance(x1, x2, U):
    for i in range(len(x1)):
        d1 = abs(x1[i] - x2[i])
        d2 = abs(x1[i] - x2[i])
        d3 = abs(x1[i] - x2[i])
        d4 = abs(x1[i] - x2[i])
        d5 = abs(x1[i] - x2[i])
        d6 = abs(x1[i] - x2[i])
        d7 = abs(x1[i] - x2[i])
        d8 = abs(x1[i] - x2[i])
        d9 = abs(x1[i] - x2[i])
        d10 = abs(x1[i] - x2[i])
        d11 = abs(x1[i] - x2[i])
        d12 = abs(x1[i] - x2[i])
        d13 = abs(x1[i] - x2[i])
        d14 = abs(x1[i] - x2[i])
        d15 = abs(x1[i] - x2[i])
        d16 = abs(x1[i] - x2[i])
        d17 = abs(x1[i] - x2[i])
        d18 = abs(x1[i] - x2[i])
        d19 = abs(x1[i] - x2[i])
        d20 = abs(x1[i] - x2[i])
        d21 = abs(x1[i] - x2[i])
        d22 = abs(x1[i] - x2[i])
        d23 = abs(x1[i] - x2[i])
        d24 = abs(x1[i] - x2[i])
        d25 = abs(x1[i] - x2[i])
        d26 = abs(x1[i] - x2[i])
        d27 = abs(x1[i] - x2[i])
        d28 = abs(x1[i] - x2[i])
        d29 = abs(x1[i] - x2[i])
        d30 = abs(x1[i] - x2[i])
        d31 = abs(x1[i] - x2[i])
        d32 = abs(x1[i] - x2[i])
        d33 = abs(x1[i] - x2[i])
        d34 = abs(x1[i] - x2[i])
        d35 = abs(x1[i] - x2[i])
        d36 = abs(x1[i] - x2[i])
        d37 = abs(x1[i] - x2[i])
        d38 = abs(x1[i] - x2[i])
        d39 = abs(x1[i] - x2[i])
        d40 = abs(x1[i] - x2[i])
        d41 = abs(x1[i] - x2[i])
        d42 = abs(x1[i] - x2[i])
        d43 = abs(x1[i] - x2[i])
        d44 = abs(x1[i] - x2[i])
        d45 = abs(x1[i] - x2[i])
        d46 = abs(x1[i] - x2[i])
        d47 = abs(x1[i] - x2[i])
        d48 = abs(x1[i] - x2[i])
        d49 = abs(x1[i] - x2[i])
        d50 = abs(x1[i] - x2[i])
        d51 = abs(x1[i] - x2[i])
        d52 = abs(x1[i] - x2[i])
        d53 = abs(x1[i] - x2[i])
        d54 = abs(x1[i] - x2[i])
        d55 = abs(x1[i] - x2[i])
        d56 = abs(x1[i] - x2[i])
        d57 = abs(x1[i] - x2[i])
        d58 = abs(x1[i] - x2[i])
        d59 = abs(x1[i] - x2[i])
        d60 = abs(x1[i] - x2[i])
        d61 = abs(x1[i] - x2[i])
        d62 = abs(x1[i] - x2[i])
        d63 = abs(x1[i] - x2[i])
        d64 = abs(x1[i] - x2[i])
        d65 = abs(x1[i] - x2[i])
        d66 = abs(x1[i] - x2[i])
        d67 = abs(x1[i] - x2[i])
        d68 = abs(x1[i] - x2[i])
        d69 = abs(x1[i] - x2[i])
        d70 = abs(x1[i] - x2[i])
        d71 = abs(x1[i] - x2[i])
        d72 = abs(x1[i] - x2[i])
        d73 = abs(x1[i] - x2[i])
        d74 = abs(x1[i] - x2[i])
        d75 = abs(x1[i] - x2[i])
        d76 = abs(x1[i] - x2[i])
        d77 = abs(x1[i] - x2[i])
        d78 = abs(x1[i] - x2[i])
        d79 = abs(x1[i] - x2[i])
        d80 = abs(x1[i] - x2[i])
        d81 = abs(x1[i] - x2[i])
        d82 = abs(x1[i] - x2[i])
        d83 = abs(x1[i] - x2[i])
        d84 = abs(x1[i] - x2[i])
        d85 = abs(x1[i] - x2[i])
        d86 = abs(x1[i] - x2[i])
        d87 = abs(x1[i] - x2[i])
        d88 = abs(x1[i] - x2[i])
        d89 = abs(x1[i] - x2[i])
        d90 = abs(x1[i] - x2[i])
        d91 = abs(x1[i] - x2[i])
        d92 = abs(x1[i] - x2[i])
        d93 = abs(x1[i] - x2[i])
        d94 = abs(x1[i] - x2[i])
        d95 = abs(x1[i] - x2[i])
        d96 = abs(x1[i] - x2[i])
        d97 = abs(x1[i] - x2[i])
        d98 = abs(x1[i] - x2[i])
        d99 = abs(x1[i] - x2[i])
    return d1 + d2 + d3 + d4 + d5 + d6 + d7 + d8 + d9 + d10 + d11 + d12 + d13 + d14 + d15 + d16 + d17 + d18 + d19 + d20 + d21 + d22 + d23 + d24 + d25 + d26 + d27 + d28 + d29 + d30 + d31 + d32 + d33 + d34 + d35 + d36 + d37 + d38 + d39 + d40 + d41 + d42 + d43 + d44 + d45 + d46 + d47 + d48 + d49 + d50 + d51 + d52 + d53 + d54 + d55 + d56 + d57 + d58 + d59 + d60 + d61 + d62 + d63 + d64 + d65 + d66 + d67 + d68 + d69 + d70 + d71 + d72 + d73 + d74 + d75 + d76 + d77 + d78 + d79 + d80 + d81 + d82 + d83 + d84 + d85 + d86 + d87 + d88 + d89 + d90 + d91 + d92 + d93 + d94 + d95 + d96 + d97 + d98 + d99

```

```

# Final centroids
[[0.0, 0.0, 1.0, 1.0, 0.0, 0.0], [1.0, 1.0, 0.0, 0.0, 1.0, 1.0]]

# Total iterations required to find final clusters: 0

```


2. $X = \{[-1, 1], [-2, -3], [1, 3], [-2, 1], [5, 1], [4, -2], [3, -1], [3, 3]\}$, $U = \{[1, 0, 0, 1, 0, 0, 1, 1], [0, 1, 1, 0, 1, 1, 0, 0]\}$

[illegible]

```
MI_Practical-12.pyrb - Colab - X  UnredKiprb - Colabentry X +
colab.research.google.com/.../MI_Practical-12.pyrb
+ Code + File
151 def f (n, range(10000)):
152     f1=f2=f3=0
153     n,rem=n//10,n%10
154     if n:
155         f1,f2,f3=f2,f3,f1+n
156     count=count+1
157     print(f"count: {count}")
158     print(f1)
159     if (len(range(10000,100000))>f1):
160         print(f"max: {f1}")
161         print(f1)
162         print(f"Total iterations required to form final list: {count}")
163         break
164     return f1
165
166 arr=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
167 arr=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068

```

3. $X = \{-1, -1\}, [1, 3], [4, 1], [3, -2], [-1, 2], [-2, 1], [5, -3], [1, -1]\}$, $U = \{[1, 1, 1, 0, 0, 0, 0, 1], [0, 0, 0, 1, 1, 1, 1, 0]\}$

```

def dist(x1, y1, x2, y2):
    return ((x2-x1)**2 + (y2-y1)**2)**0.5

def dist(x1, y1, x2, y2):
    return ((x2-x1)**2 + (y2-y1)**2)**0.5

def main():
    # Define the points in X and U
    X = [[-1, -1], [1, 3], [4, 1], [3, -2], [-1, 2], [-2, 1], [5, -3], [1, -1]]
    U = [[1, 1, 1, 0, 0, 0, 0, 1], [0, 0, 0, 1, 1, 1, 1, 0]]

    # Calculate the distance between the points in X and U
    for i in range(len(X)):
        for j in range(len(U)):
            dist = dist(X[i][0], X[i][1], U[j][0], U[j][1])
            print(f"Distance between {X[i]} and {U[j]} is {dist}")

    # Print the total number of iterations required to find the final clusters
    print("Total iterations required to find final clusters: 4")

if __name__ == "__main__":
    main()

```

```

def main():
    # Define the points in X and U
    X = [[-1, -1], [1, 3], [4, 1], [3, -2], [-1, 2], [-2, 1], [5, -3], [1, -1]]
    U = [[1, 1, 1, 0, 0, 0, 0, 1], [0, 0, 0, 1, 1, 1, 1, 0]]

    # Calculate the distance between the points in X and U
    for i in range(len(X)):
        for j in range(len(U)):
            dist = dist(X[i][0], X[i][1], U[j][0], U[j][1])
            print(f"Distance between {X[i]} and {U[j]} is {dist}")

    # Print the total number of iterations required to find the final clusters
    print("Total iterations required to find final clusters: 4")

if __name__ == "__main__":
    main()

```

4. $X = \{[1, 1], [-1, 4], [-2, 1], [3, 4], [-3, -2], [5, 4], [2, -2], [-4, -1]\}$, $U = \{[1, 0, 1, 0, 0, 1, 0, 1], [0, 1, 0, 1, 1, 0, 1, 0]\}$

The image displays two screenshots of a Google Colab notebook titled 'ML_Practical-12.ipynb'. The first screenshot shows the initial code for K-means clustering, including data initialization and the first iteration of the algorithm. The second screenshot shows the final output of the clustering process, displaying the centroids and the assignment of data points to clusters.

```

# Initial data points
X = [[1, 1], [-1, 4], [-2, 1], [3, 4], [-3, -2], [5, 4], [2, -2], [-4, -1]]
U = [[1, 0, 1, 0, 0, 1, 0, 1], [0, 1, 0, 1, 1, 0, 1, 0]]

# K-means algorithm
def kmeans(X, U, k):
    # Initialize centroids
    centroids = X[:k].copy()
    # Iterate until convergence
    for i in range(100):
        # Calculate distances
        dists = np.zeros((len(X), k))
        for j in range(k):
            dists[:, j] = np.linalg.norm(X - centroids[j], axis=1)
        # Assign points to clusters
        labels = np.argmin(dists, axis=1)
        # Calculate new centroids
        new_centroids = np.zeros((k, len(X[0])))
        for j in range(k):
            new_centroids[j] = np.mean(X[labels == j], axis=0)
        centroids = new_centroids
    return centroids, labels

# Run K-means
centroids, labels = kmeans(X, U, 2)

# Print results
print("Centroids:")
print(centroids)
print("Labels:")
print(labels)

```

The output of the K-means algorithm shows the final centroids and the assignment of data points to clusters. The centroids are approximately $[-0.5, 2.5]$ and $[2.5, -0.5]$. The labels for the data points are $[1, 0, 1, 0, 0, 1, 0, 1]$ and $[0, 1, 0, 1, 1, 0, 1, 0]$.

CONCLUSION :- It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabelled dataset on its own without the need for any training. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.