Practical - 1

Aim:- Write a python program to handle the “Numpy” and “Matplotlib” library.

Tool:- VS code

Theory:-

Numpy :-

NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

The NumPy library contains multidimensional array and matrix data structures (you'll find more information about this in later sections). It provides ndarray, a homogeneous n-dimensional array object, with methods to efficiently operate on it.

Matplotlib: -

Matplotlib is a plotting library for Python. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

The Matplotlib module was first written by John D. Hunter. Since 2012, Michael Droettboom has been the principal developer.

Codes and Outputs:-

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.

Code:

```
import numpy as np
# 2-D array
array_2d = np.array([[5, 6], [6, 8]])

# 3-D array
array_3d = np.array([[5,6,1], [5,9,1], [1, 2 ,3]])

# Operations on 2D array
print("\nOperations on 3D array:\n")

#  Addition
addition = array_2d + array_2d

# Subtraction
subtraction = array_2d - array_2d

#  Elementwise Multiplication
element_wise_multiplication = array_2d * array_2d

#  Matrix Multiplication
matrix_multiplication = array_2d @ array_2d

#  Results
print("Addition:\n", addition)
print("Subtraction:\n", subtraction)
print("Element-wise multiplication:\n", element_wise_multiplication)
print("Matrix multiplication:\n", matrix_multiplication)

# Operations on 3D array
print("\nOperations on 3D array:\n")

#  Addition
addition = array_3d + array_3d

# Subtraction
subtraction = array_3d - array_3d

#  Elementwise Multiplication
element_wise_multiplication = array_3d * array_3d

#  Matrix Multiplication
matrix_multiplication = array_3d @ array_3d

#  Results
print("Addition:\n", addition)
print("Subtraction:\n", subtraction)
print("Element-wise multiplication:\n", element_wise_multiplication)
print("Matrix multiplication:\n", matrix_multiplication)
```

Output:

```
Operations on 3D array:

Addition:
 [[10 12]
 [12 16]]
Subtraction:
 [[0 0]
 [0 0]]
Element-wise multiplication:
 [[25 36]
 [36 64]]
Matrix multiplication:
 [[ 61  78]
 [ 78 100]]

Operations on 3D array:

Addition:
 [[10 12  2]
 [10 18  2]
 [ 2  4  6]]
Subtraction:
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
Element-wise multiplication:
 [[25 36  1]
 [25 81  1]
 [ 1  4  9]]
Matrix multiplication:
 [[ 56  86  14]
 [ 71 113  17]
 [ 18  30  12]]
```

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.

Code and Output:

```
#2
dict = {}
for i in range(1,21):
  if i % 2 == 0:
    dict[i] = i*i
  else:
    dict[i] = i*i*i

print(dict)
```

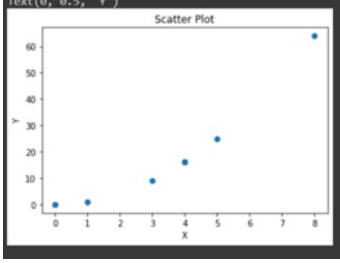
```
{1: 1, 2: 4, 3: 27, 4: 16, 5: 125, 6: 36, 7: 343, 8: 64, 9: 729, 10: 100, 11: 1331, 12: 144, 13: 2197, 14: 196, 15: 3375, 16: 256, 17: 4913, 18: 324, 19: 6859, 20: 400}
```

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

Code and Output:

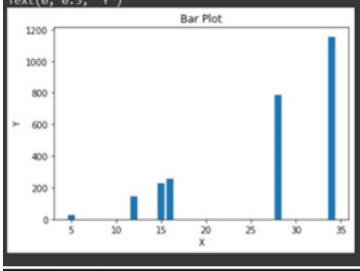
```
x3 = np.random.randint([1,5,4,3,6,7,8,9,2])
y3 = x3**2

plt.scatter(x3,y3)
plt.title("Scatter Plot")
plt.xlabel("X")
plt.ylabel("Y")
```

```
Text(0, 0.5, 'Y')
```



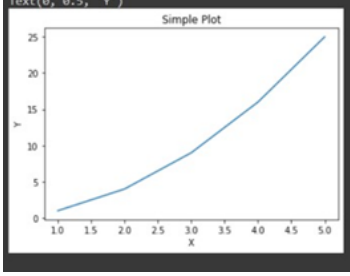

```
x2 = np.random.randint([12,23,31,54,63,45])
y2 = x2**2

plt.bar(x2,y2)
plt.title("Bar Plot")
plt.xlabel("X")
plt.ylabel("Y")
```

```
Text(0, 0.5, 'Y')
```




```
import numpy as np
from matplotlib import pyplot as plt

x1 = np.array([1,2,3,4,5])
y1 = x1**2

plt.plot(x1,y1)
plt.title("Simple Plot")
plt.xlabel("X")
plt.ylabel("Y")
```

```
Text(0, 0.5, 'Y')
```



Conclusion:-

Thus I studied how to handle the “Numpy” and “Matplotlib” libraries.

Practical - 2

Aim :– To write a python program to plot 2D & 3D Gaussian distribution curves using probability of likelihood formula.

Tool:- Google colab

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.

The Gaussian distribution is the backbone of Machine Learning. Every data scientist needs to know during working with linear models. It was learnt by Carl Friedrich Gauss.

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

a. Gaussian processes

b. Variational inference

c. Reinforcement learning

- It is similarly broadly used in other application areas such as

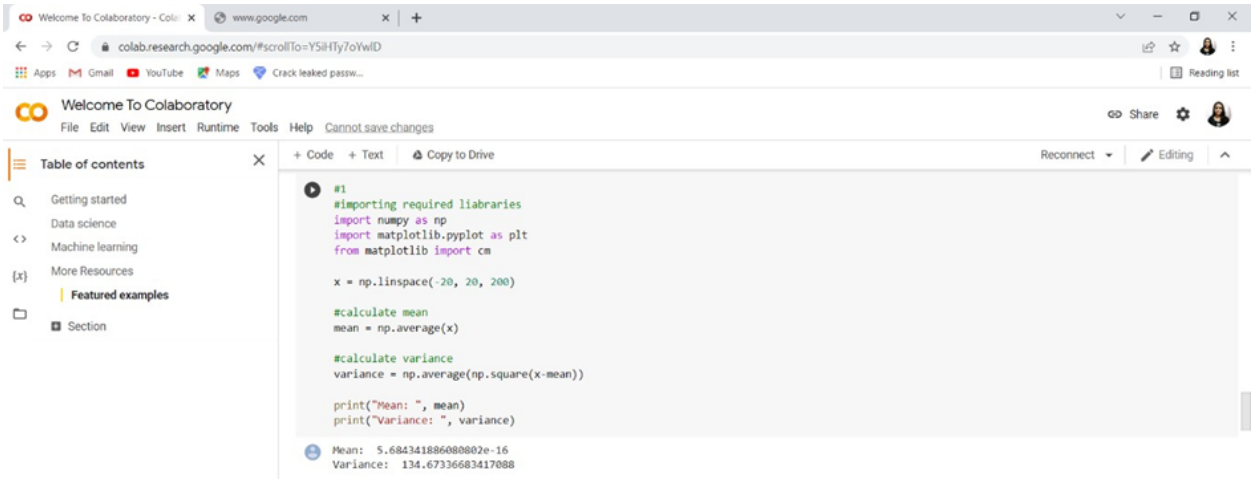
d. Signal processing such as Kalman filter

e. Control, for example linear quadratic regulator

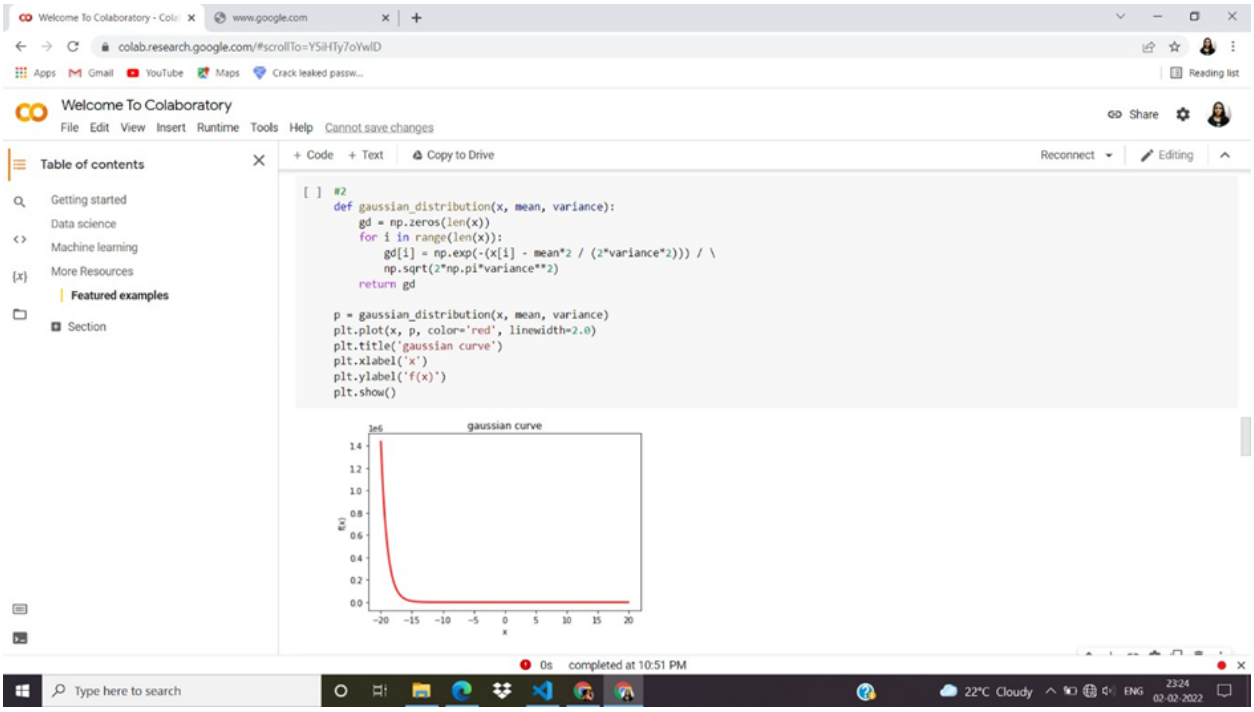
f. Statistics, as hypothesis testing

Codes and Outputs:-

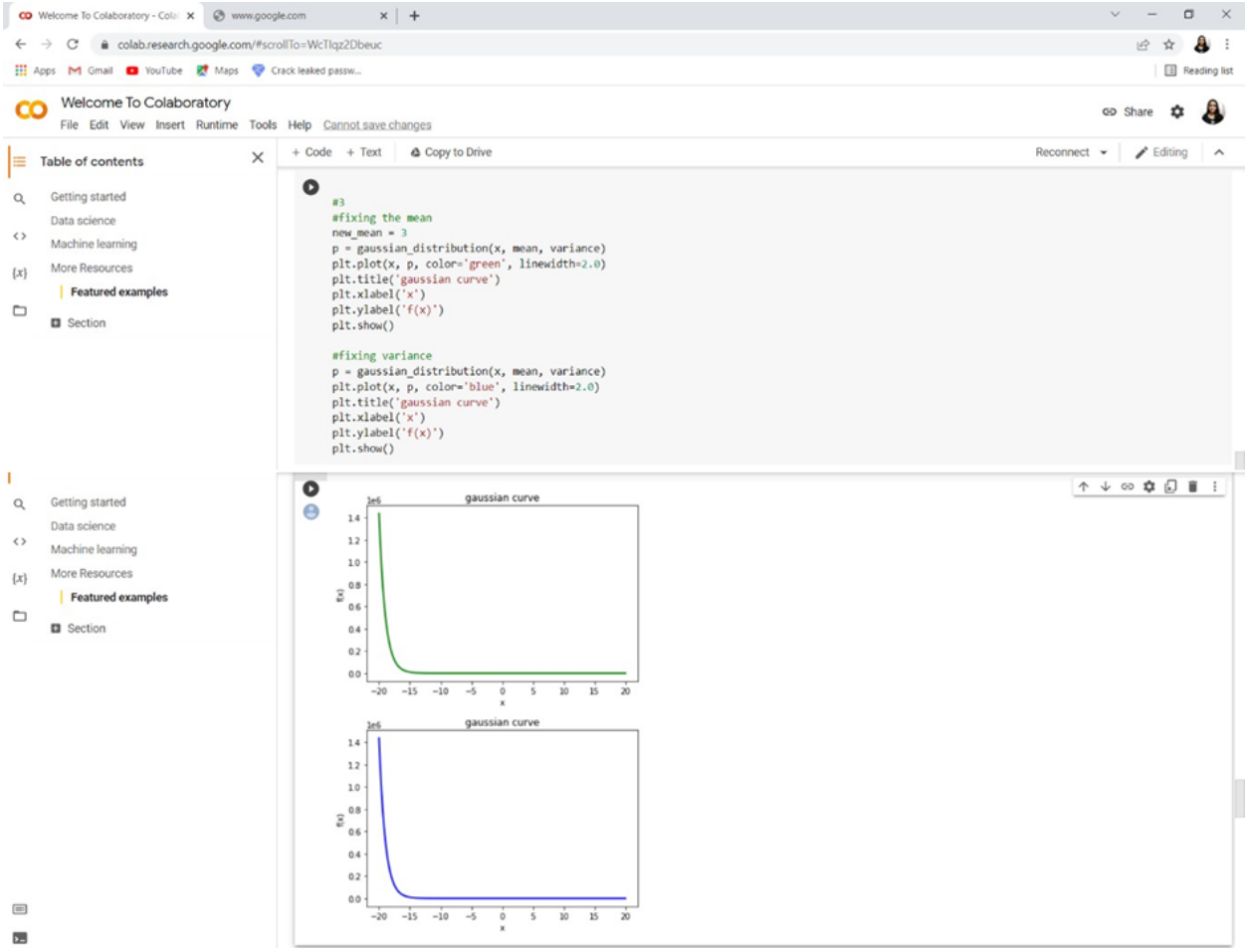
1. Define the data sample x in range (any) for 1D data. Decide mean & variance and print it.



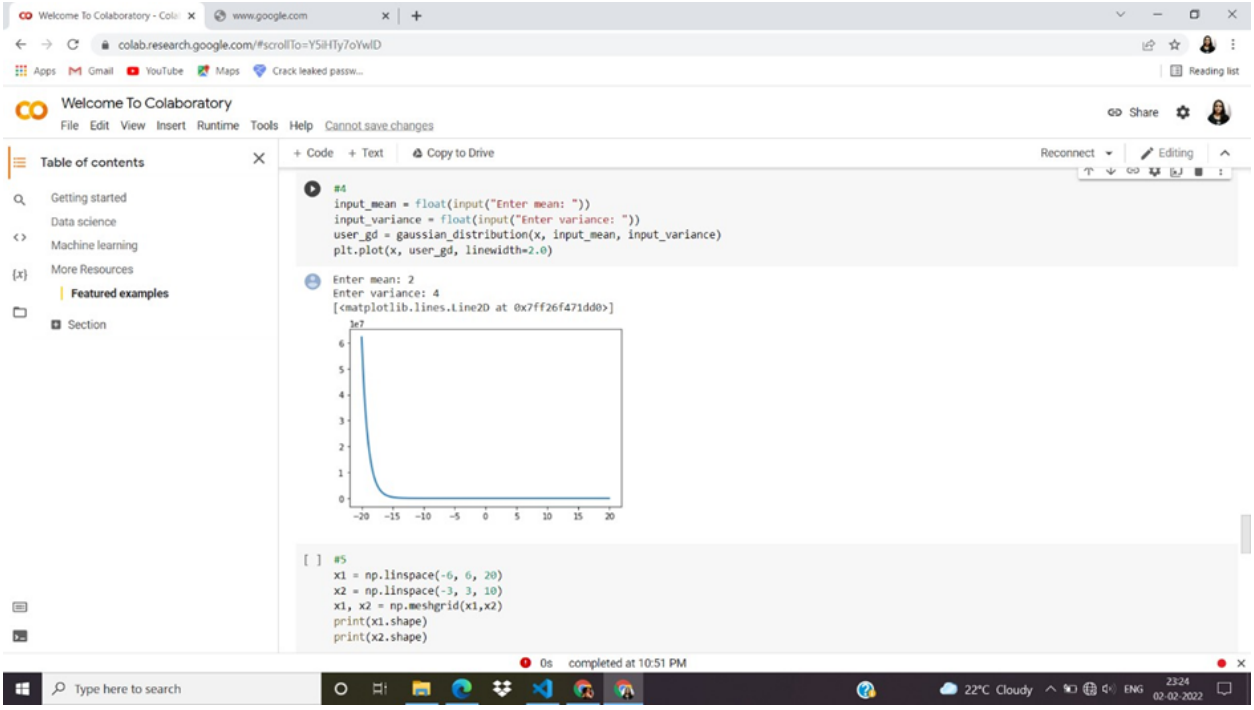
2. Use formula, and plot the 1D Gaussian curve.



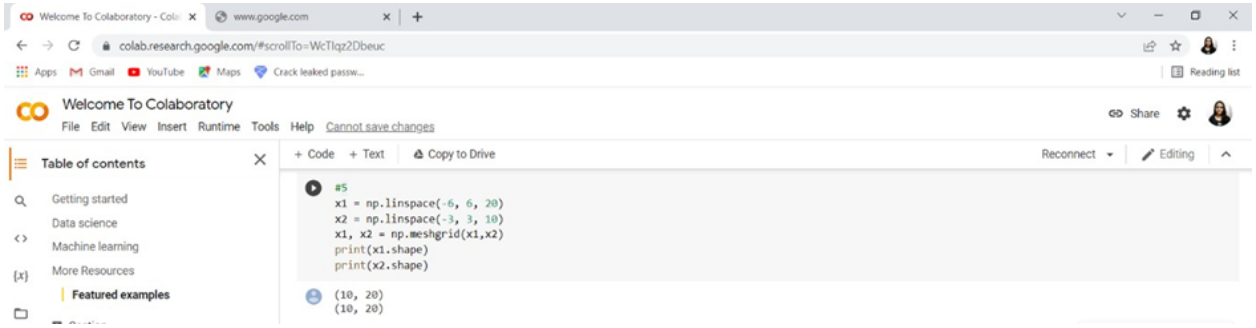
3. Fix the mean and vary the variance and plot the curve. Fix the variance and vary the mean and plot the curve.



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



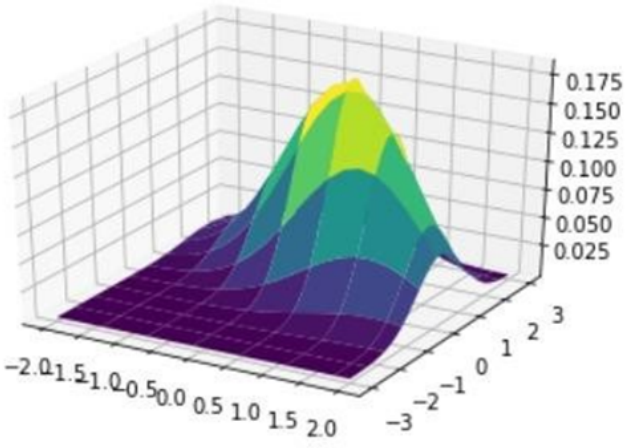
5. Define the data samples x1 and x2 for 2D data. Define X1,X2=meshgrid[x1:x2] and print its shape.



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

```
mu = np.array([0.5, 1.0])
sigma = np.array([[1.0, -0.5], [-0.5, 1.0]])
pos = np.empty(X1.shape + (2,))
pos[:, :, 0] = X1
pos[:, :, 1] = X2
n = mu.shape[0]
sigma_del = np.linalg.det(sigma)
sigma_inv = np.linalg.inv(sigma)
N = np.sqrt((2*np.pi)**n * sigma_del)
fact = np.einsum('...k,kl,...l->...', pos-mu, sigma_inv, pos-mu)
gauss_3d = np.exp(-fact / 2)/N
figure = plt.figure()
ax = figure.gca(projection='3d')
ax.plot_surface(X1, X2, gauss_3d, rstride=3, cstride=3,
 cmap=cm.viridis, antialiased=True)
```

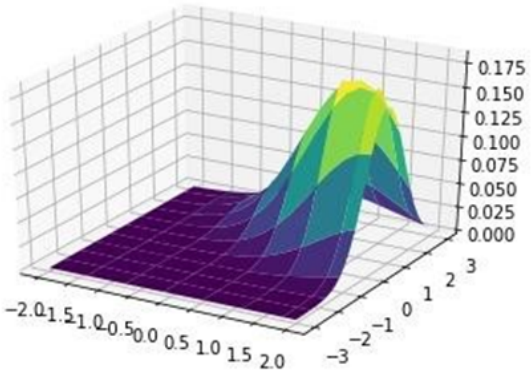
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7ff93592ddd0>
```



7. Define the special purpose “functions” for plotting the 3D Gaussian distribution.

```
def gaussian_distribution_3d(pos, mean1, mean2, sigma1, sigma2, sigma3, sigma4):
 mu = np.array([mean1, mean2])
 sigma = np.array([[sigma1, sigma2], [sigma3, sigma4]])
 sigma_del = np.linalg.det(sigma)
 sigma_inv = np.linalg.inv(sigma)
 N = np.sqrt((2*np.pi)**n * sigma_del)
 fact = np.einsum('...k,kl,...l->...', pos-mu, sigma_inv, pos-mu)
 return np.exp(-fact / 2)/N
mean1 = float(input('Enter mean1: '))
mean2 = float(input('Enter mean2: '))
sigma1 = float(input('Enter sigma1: '))
sigma2 = float(input('Enter sigma2: '))
sigma3 = float(input('Enter sigma3: '))
sigma4 = float(input('Enter sigma4: '))
gauss_3d = gaussian_distribution_3d( pos, mean1, mean2,
 sigma1, sigma2, sigma3, sigma4)
figure = plt.figure()
ax = figure.gca(projection='3d')
ax.plot_surface(X1, X2, gauss_3d, rstride=3, cstride=3,
 linewidth=2.0, antialiased=True, cmap=cm.viridis)
```

```
Enter mean1: 1.5
Enter mean2: 1
Enter sigma1: 1
Enter sigma2: -0.5
Enter sigma3: -0.5
Enter sigma4: 1
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7ff935b04e10>
```



Conclusion:

Thus, using the probability likelihood formula we plotted 2D and 3D curves. By observing graphs, we can conclude that the peak of the curve depends on mean value and the width of the curve depends on variance.

Practical - 3

Aim: To write a python program for Linear Regression Modeling.

Tool: Google colab

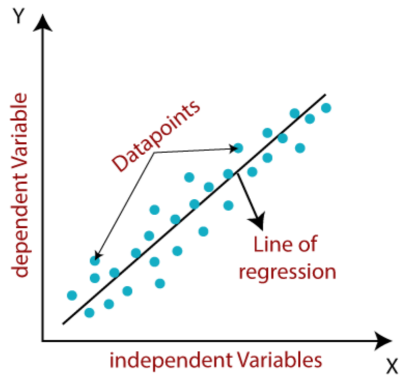
Theory:

Linear Regression Modeling:

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 (y) variables, hence called 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.

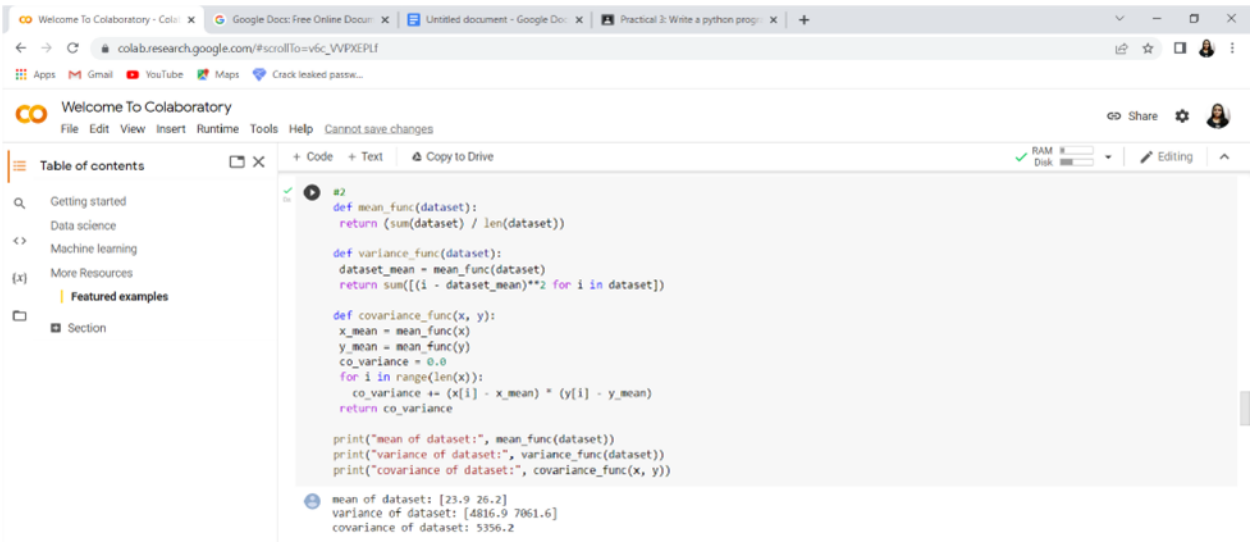
The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Codes and Outputs:

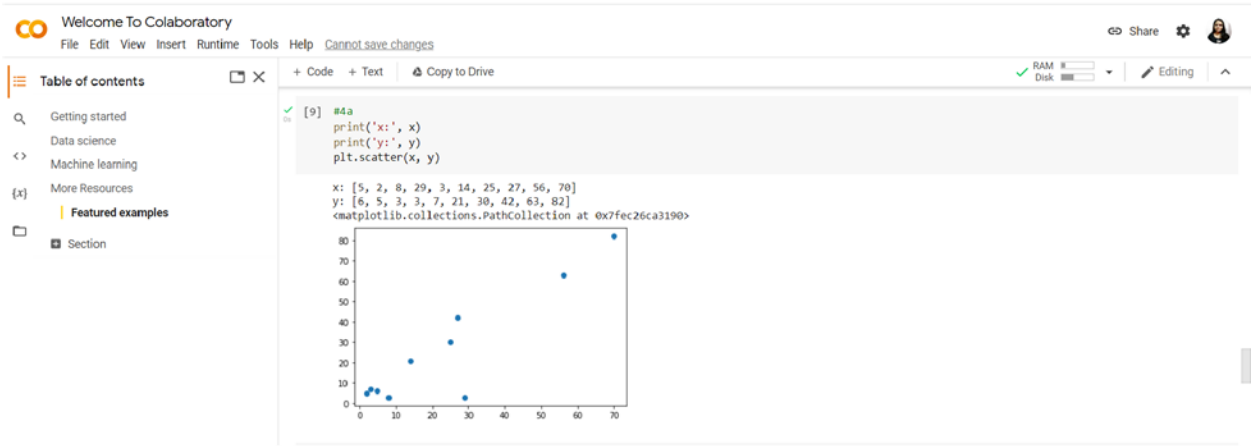
1. Create your own dataset containing (10 rows and 2 columns). Print the dataset.

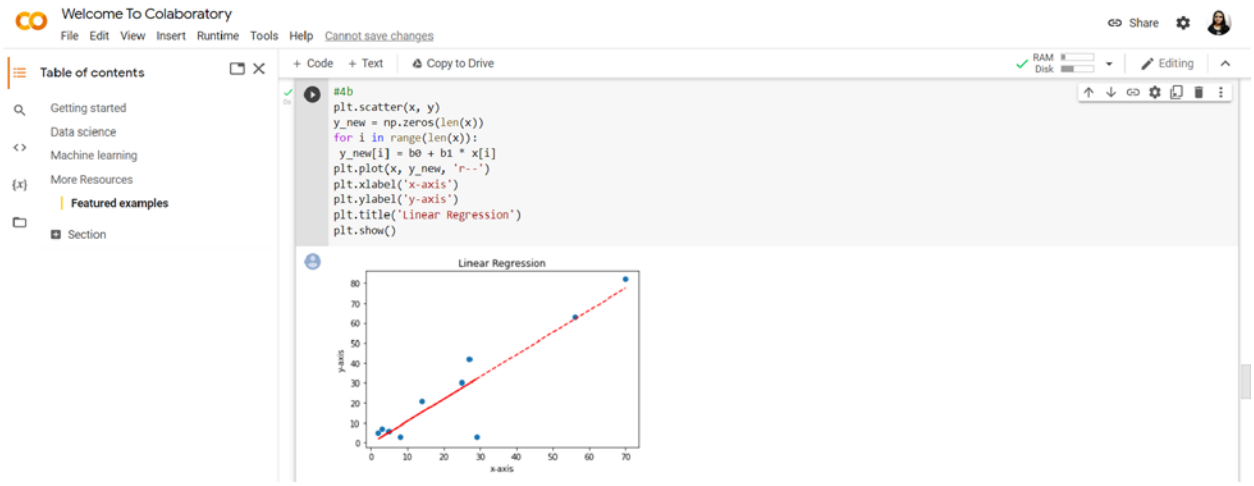
2. Define the functions for computing mean, variance and covariance and also print the respective values for a given dataset.



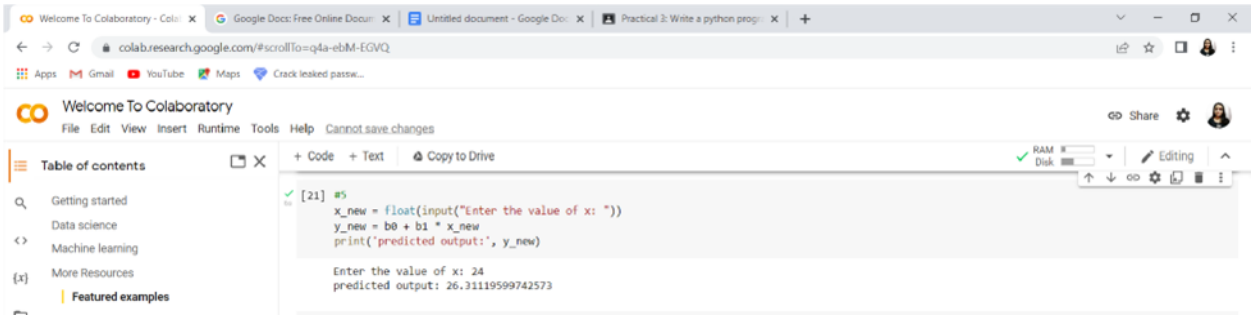
3. Compute the linear regression coefficients using function and print it.

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

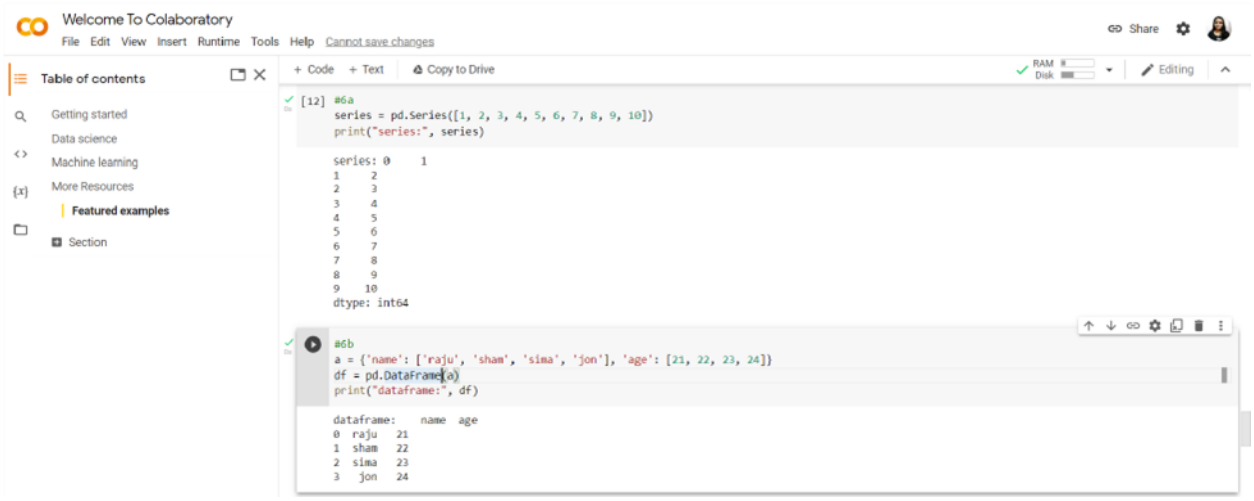




5. Find the predicted output for unknown input value using linear regression coefficients and print it.

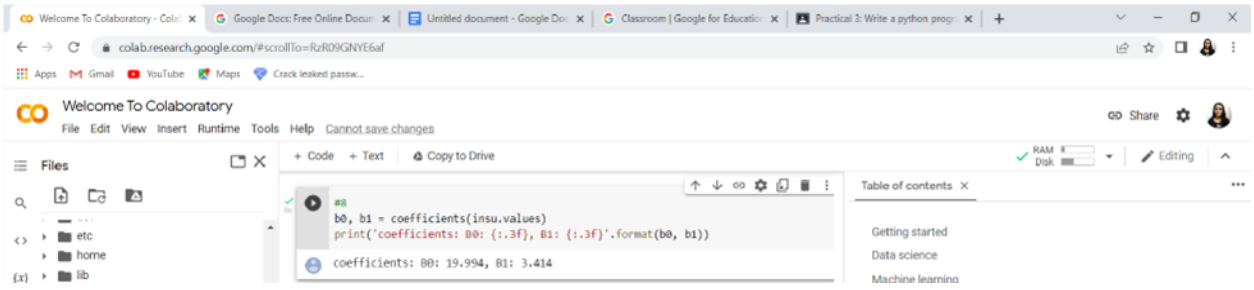


6. Create your own pandas Series and DataFrame and print it.

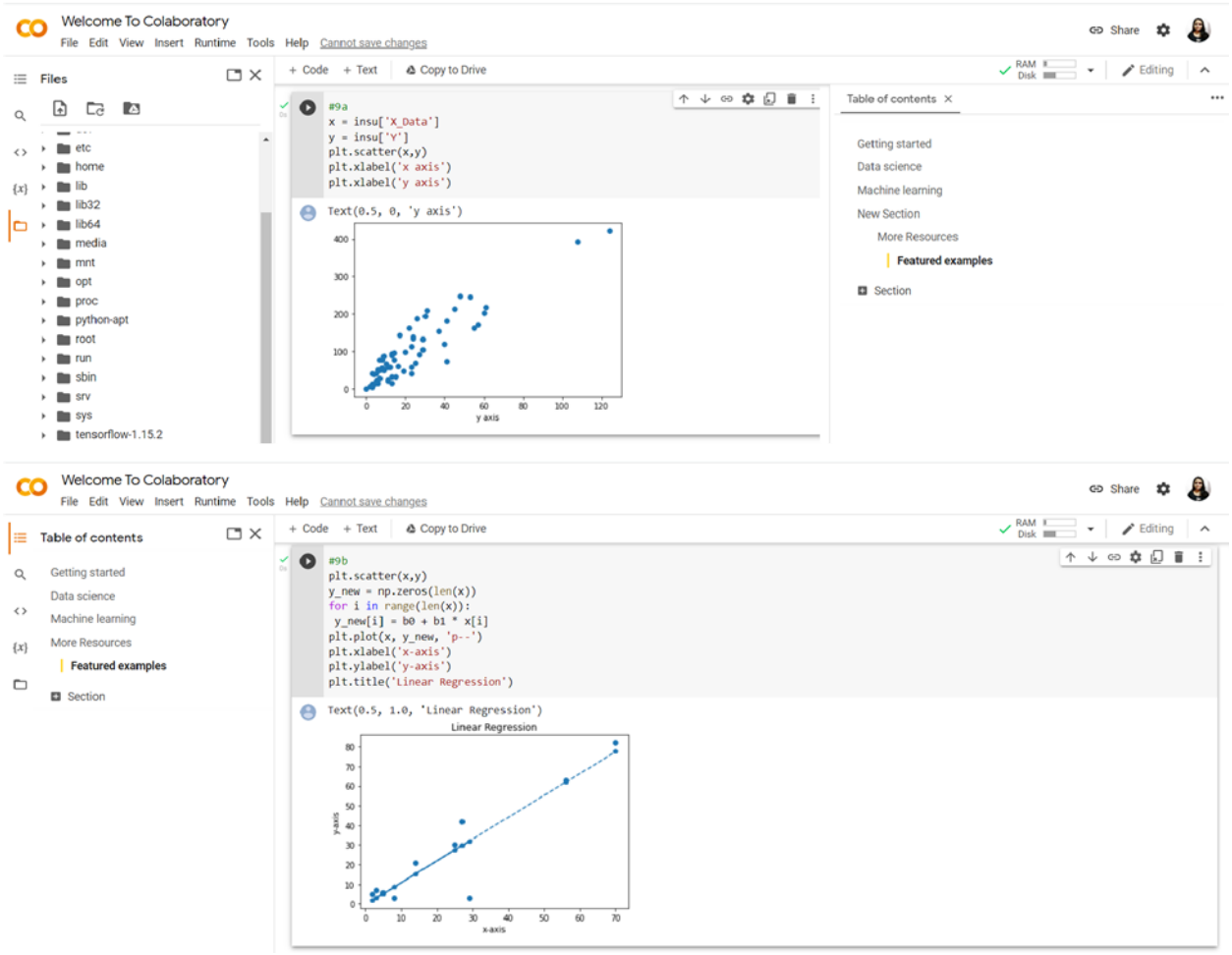


7. Read the insu.csv file using Pandas library. Print the head values. Print the column values. (CSVfile is attached herewith)

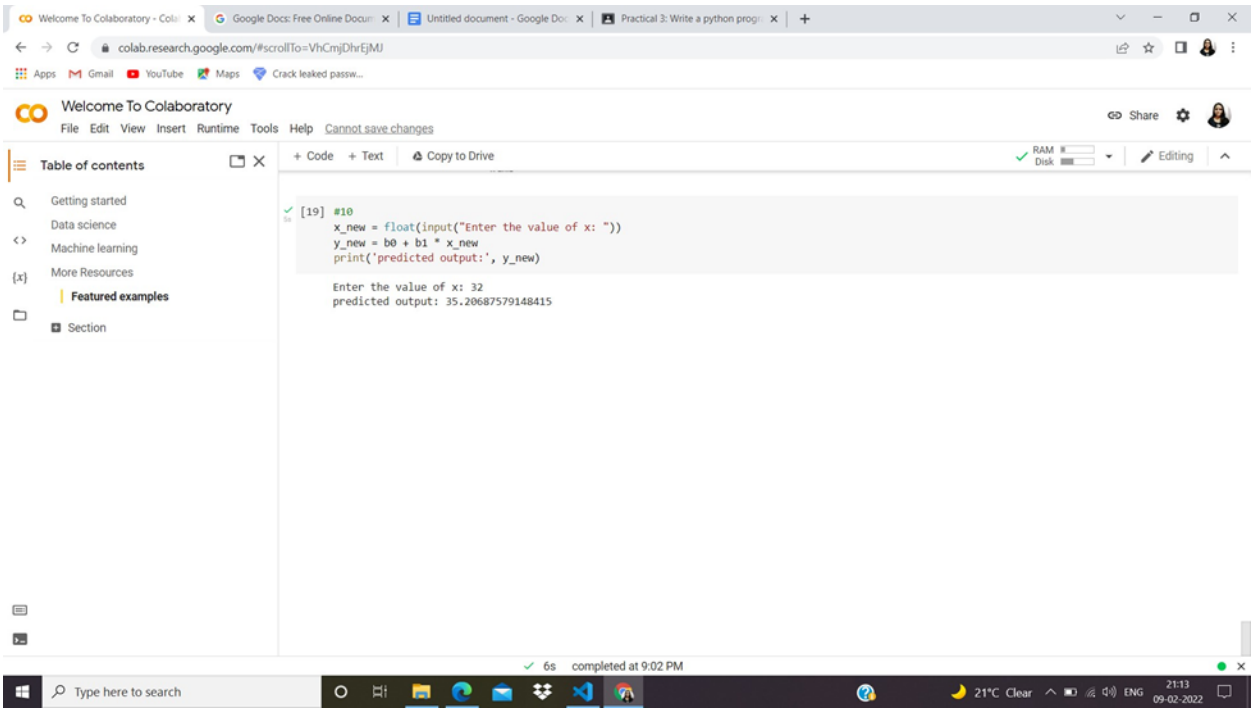
8. Find the linear regression coefficients for given file and print the values.



9. Plot the variables containing the values from the csv file. Also plot the linear regression line using coefficients.



10. Find the predicted output for unknown input values using linear regression coefficients and print it. (for insu.csv dataset)



Conclusion:

I have coded python programs for linear regression modeling and understood that linear regression modeling describes the relationship between a dependent variable and one or more independent variables. It can be used to build the models & predict their response.

Practical - 4

Aim: To write a python program for implementation of Bayes Theorem.

Tool: Google colab

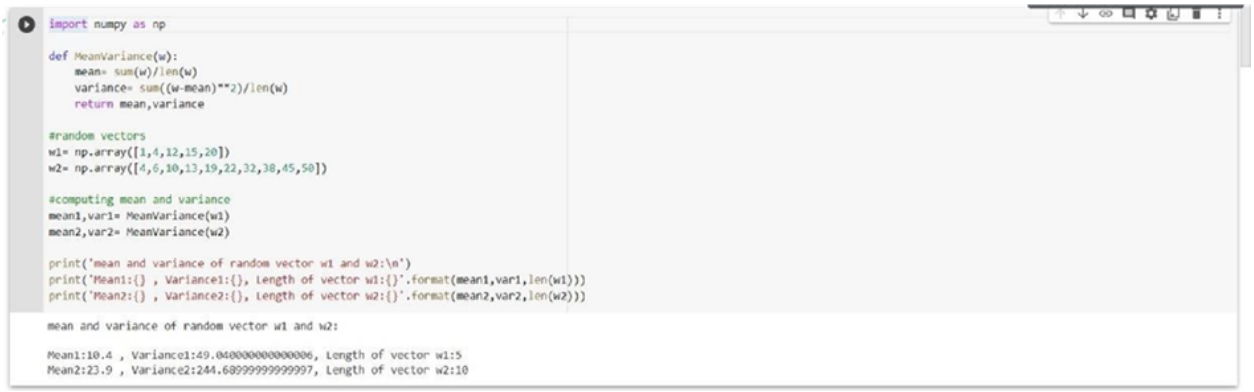
Theory:

Bayes Theorem:

Bayes theorem was given by an English statistician, philosopher, and Presbyterian minister named Mr. Thomas Bayes in the 17th century. Bayes provides their thoughts in decision theory which is extensively used in important mathematics concepts as Probability. Bayes theorem is also widely used in Machine Learning where we need to predict classes precisely and accurately. An important concept of Bayes theorem named Bayesian method is used to calculate conditional probability in a Machine Learning application that includes classification tasks. Further, a simplified version of Bayes theorem (Naïve Bayes classification) is also used to reduce computation time and average cost of the projects.

Bayes theorem is also known with some other name such as Bayes rule or Bayes Law. *Bayes theorem helps to determine the probability of an event with random knowledge*. It is used to calculate the probability of one event occurring while another one has already occurred. It is the best method to relate the condition probability and marginal probability.

1. Define a function for computing the mean and variance of 1D data. Print mean and variance of 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([1,4,12,15,20])
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('Mean1:{} , Variance1:{}, Length of vector w1:{}'.format(mean1,var1,len(w1)))
print('Mean2:{} , Variance2:{}, Length of vector w2:{}'.format(mean2,var2,len(w2)))
```

```
mean and variance of random vector w1 and w2:

Mean1:10.4 , Variance1:49.040000000000006, Length of vector w1:5
Mean2:23.9 , Variance2:244.68999999999997, 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 bayes rule.

```
import numpy as np

w1 = np.array([1,3,8,12])
w2 = np.array([6,9,13,16])
p_w1 =0.6
p_w2=0.4
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_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 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.

```
[6] 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_x= p_x_w* p_w                                        #evidence
        print('data sample: {}  dataset:{} likelihood :{} and evidende:{}'.format(x,w,p_x_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 :0.020540475723085296 and evidende:0.012324285433851176
data sample: 9.6  dataset:[ 1  3  8 12] likelihood :0.019565126404659496 and evidende:0.011739075842795698
data sample: 11.5  dataset:[ 1  3  8 12] likelihood :0.018585791018506888 and evidende:0.011151474611104084
data sample: 7.8  dataset:[ 6  9 13 16] likelihood :0.030723041911599608 and evidende:0.012289216764639843
data sample: 9.6  dataset:[ 6  9 13 16] likelihood :0.028874066993904203 and evidende:0.011549627975616813
data sample: 11.5  dataset:[ 6  9 13 16] likelihood :0.027042960119025017 and evidende:0.010817184047610007
```

4. If w1= [5,6,9,10,14,18,8,4,7,9] with p(w1)=0.7 and w2 =[11,15,17,14,13,8,12,9,10,21] with p(w2)=0.3,find the class for test samples 9.5,11.6,14.5,16.3,13.8 and 12.6 using bayes 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 [[3,5],[8,7],[4,6]] and [[4,7],[8,6],[3,2]].

```
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=\n{}'.format(M1,S1))
print('mean2={}  \ncovariance2=\n{}'.format(M2,S2))
```

```
mean1=[5. 6.]
covariance1=
[[14.  5.]
 [ 5.  2.]]
mean2=[5. 5.]
covariance2=
[[14.  7.]
 [ 7. 14.]]
```

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

```
w1=np.array([[2,4],[4,3],[5,8]]); p_w1=0.65;
w1=np.array([[4,6],[5,7],[7,10]]);p_w2=0.35;
M1,M2,S1,S2= mean_covar(w1,w2)

def check_class(x_new,m1,S1,m2,S2):
  s1_det= np.linalg.det(S1)
  s2_det= np.linalg.det(S2)
  s1_inv= np.linalg.inv(S1)
  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   #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(x_new))
  else:
      print('data sample {} is belonging to class w2'.format(x_new))

x1= np.array([4.5,5.5])
x2= np.array([3.7,7.3])
x3= np.array([5.5,7.3])

check_class(x1,M1,S1,M2,S2)
check_class(x2,M1,S1,M2,S2)
check_class(x3,M1,S1,M2,S2)
```

```
data sample [4.5 5.5] is belonging to class w2
data sample [3.7 7.3] is belonging to class w2
data sample [5.5 7.3] 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):
  s1_det= np.linalg.det(S1)
  s2_det= np.linalg.det(S2)
  s1_inv= np.linalg.inv(S1)
  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) #likelihood
  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) #likelihood
  p_x1= p_x1_w1* p_w1 + p_x1_w2*p_w2                                           #evidence
  print('data sample:{} likelihood (x1/w1):{} (x1/w2) {} evidence: {}'.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):0.0003516178160054805 (x1/w2) 0.0020972948172110637 evidence: 0.0009626044864274346
data sample:[3.7 7.3] likelihood (x1/w1):2.0451482908524873e-11 (x1/w2) 0.0010985692269010665 evidence: 0.0003844992427088371
data sample:[5.5 7.3] likelihood (x1/w1):0.050830097206649155 (x1/w2) 0.0022408069687383 57 evidence: 0.0390290456233803 8
```

8. If w1=[[6,9],[13,10],[8,11],[14,17],[19,17]] with p(w1)=0.63 andw2=[[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.

```
def check_class(x_new,m1,S1,m2,S2):
  s1_det= np.linalg.det(S1)
  s2_det= np.linalg.det(S2)
  s1_inv= np.linalg.inv(S1)
  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)   #likelihood
  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)   #likelihood
  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 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])
x3= np.array([14,19])
x3= 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)
```

```
data sample [7.7 9.9] is belonging to class w1
data sample [11 13] is belonging to class w1
data sample [ 5 30] is belonging to class w1
```

Conclusion: I studied Baye,s theorem with the help of above examples using python programs. Hence, Bayes' theorem is used to figure out conditional probability. In machine learning it can be used as an effective way to predict classes with precision and accuracy.

Practical - 5

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

Tool: Google colab

Theory:

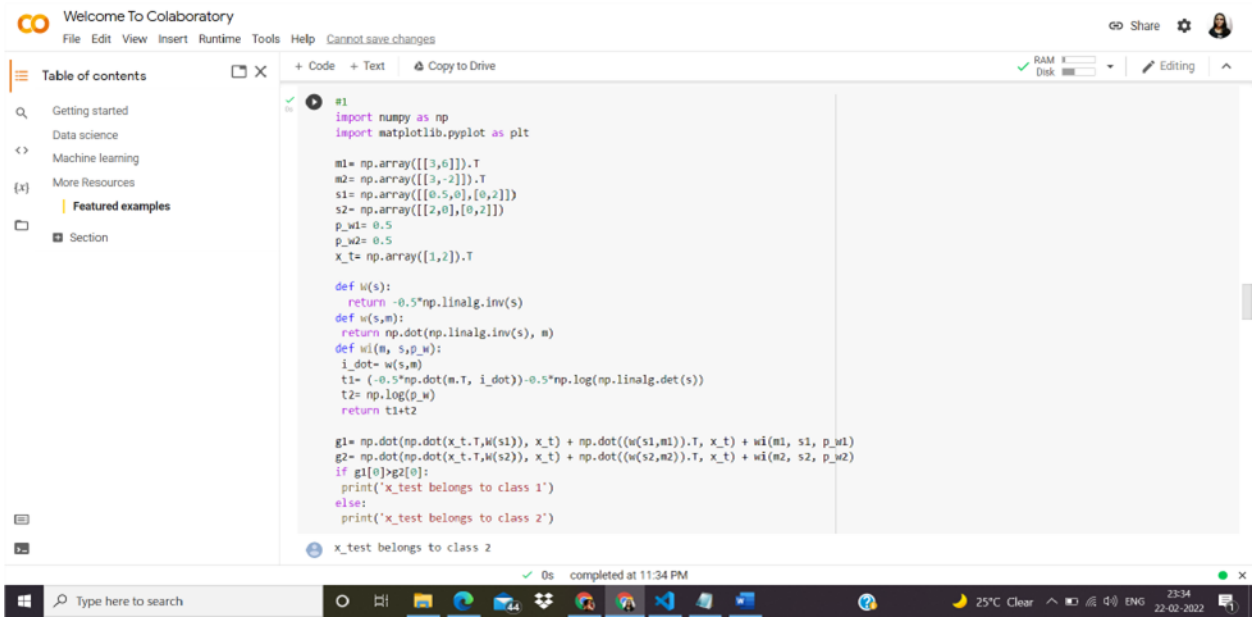
Discriminant function and decision boundary.:

Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).

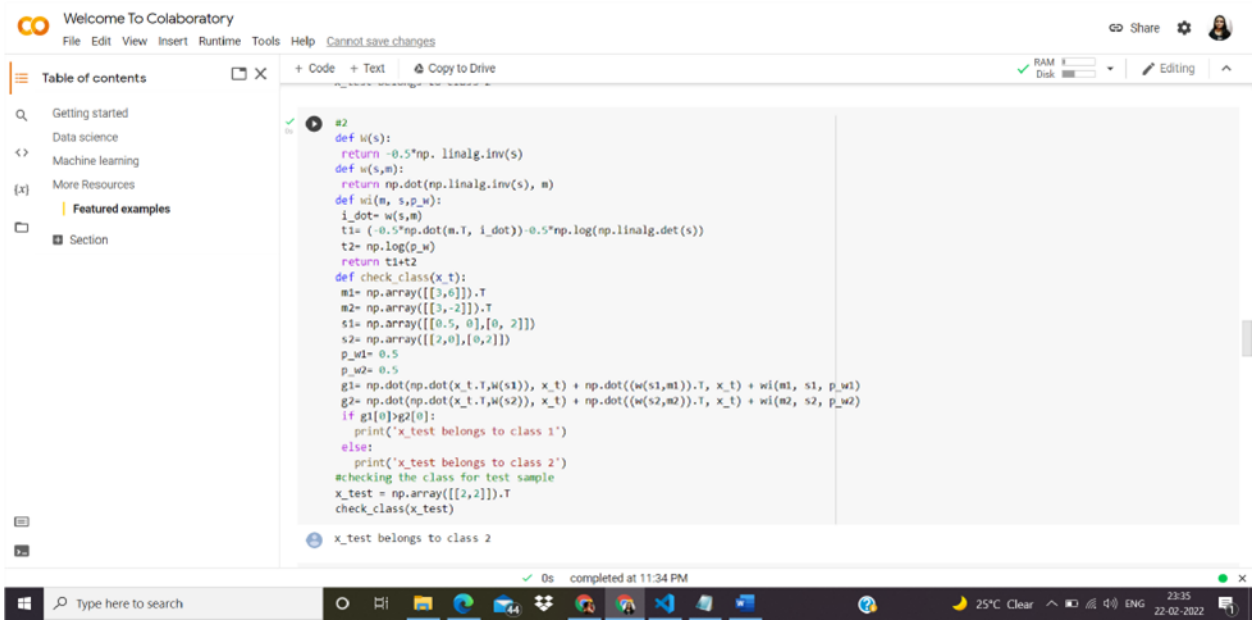
The Decision Boundary separates the data-points into regions, which are actually the classes in which they belong. Importance/Significance of a Decision Boundary: After training a machine Learning Model using a data-set, it is often necessary to visualize the classification of the data-points in Feature Space.

Codes and Outputs:

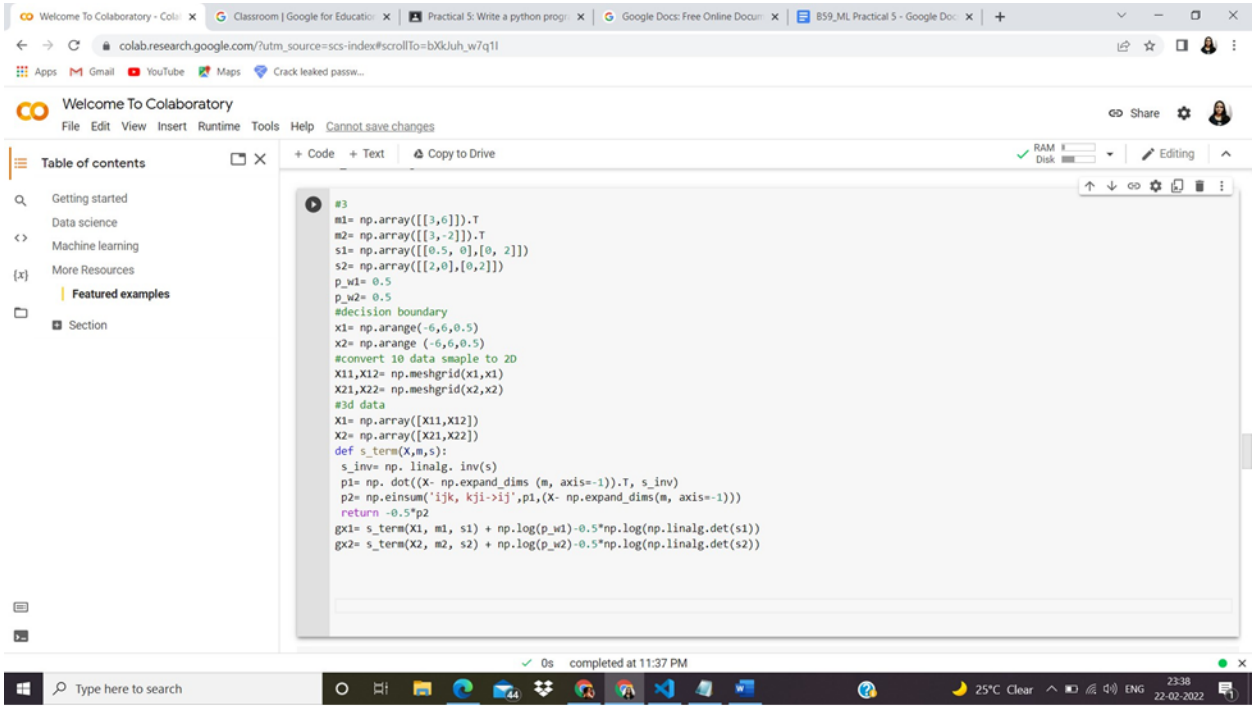
1. Classify test sample x_test = [1, 2]T using discriminant function for given data: m1 = [3, 6]T, m2= [3, -2]T, S1 = [[0.5, 0], [0, 2]], S2 = [[2, 0],[0, 2]], p(w1)=0.5, p(w2)=0.5.

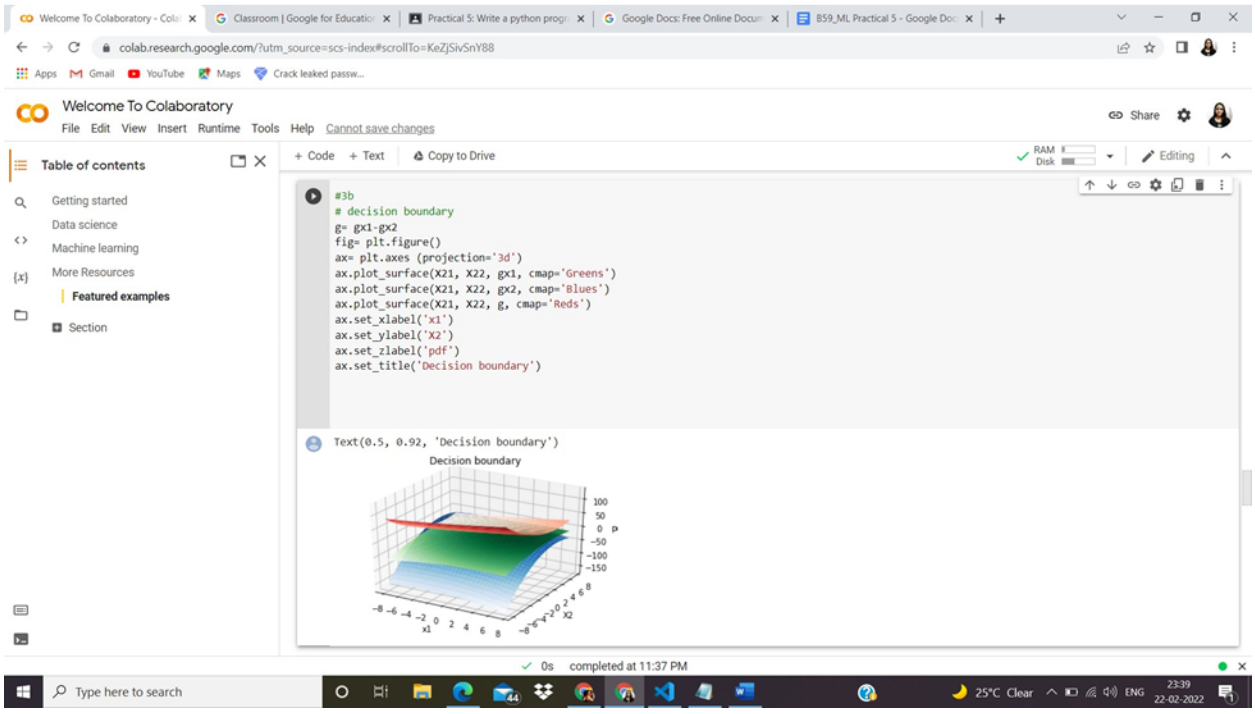


2. Classify test sample x_test = [2, 2]T for above data.

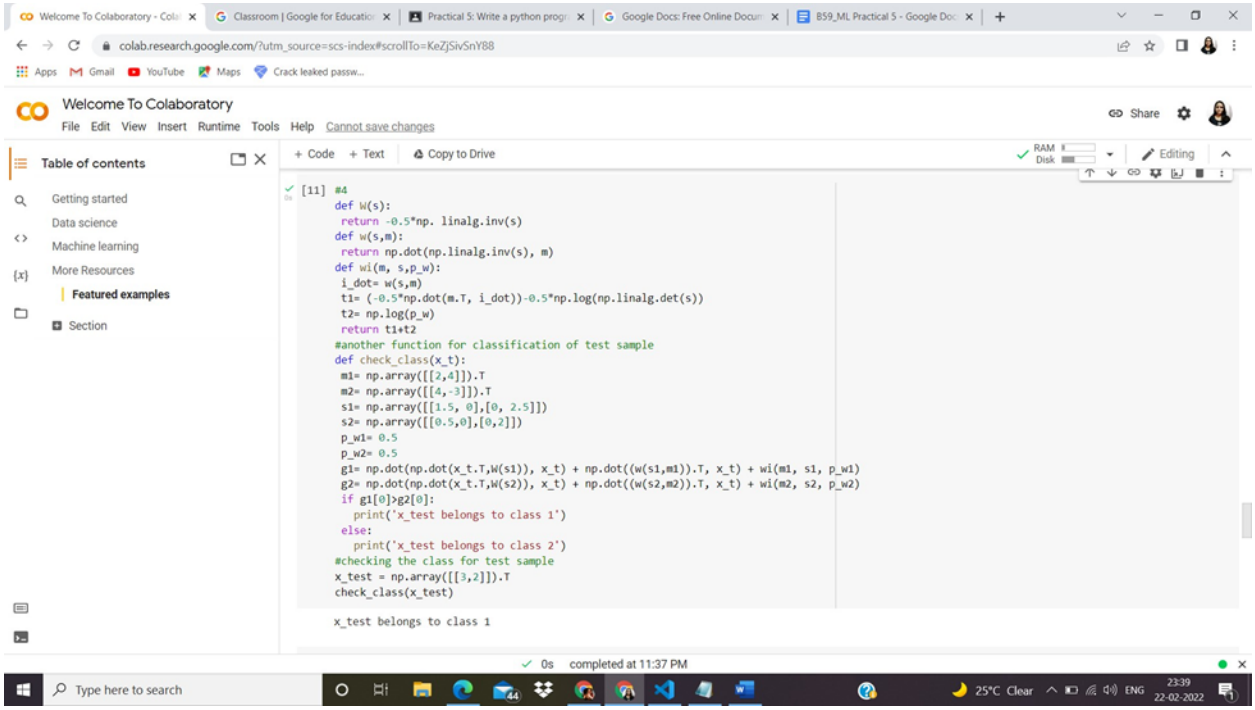


3. Find the Decision boundary for subtask 1. Take x1=np.arange(-6,6,0.5) &x2=np.arange(-6,6,0.5) and use np.meshgrid for converting 1D data to 2D data. Plot the 3D surface of g1(x), g2(x) and g.

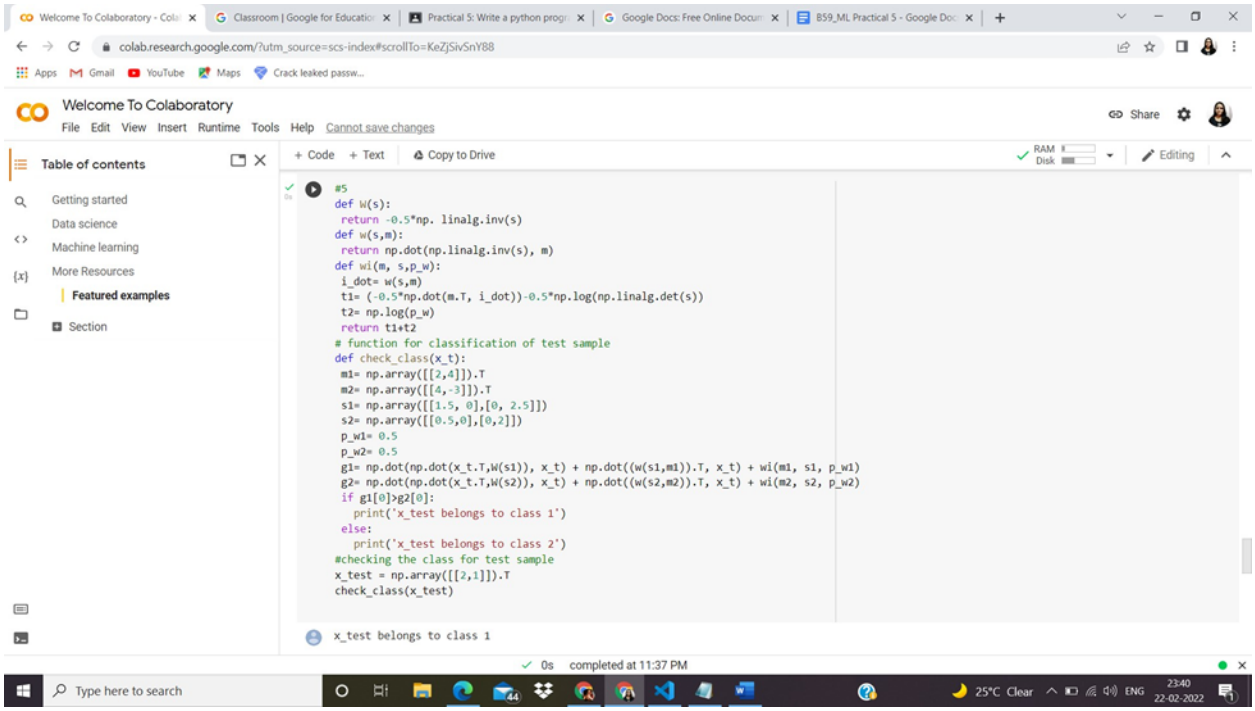




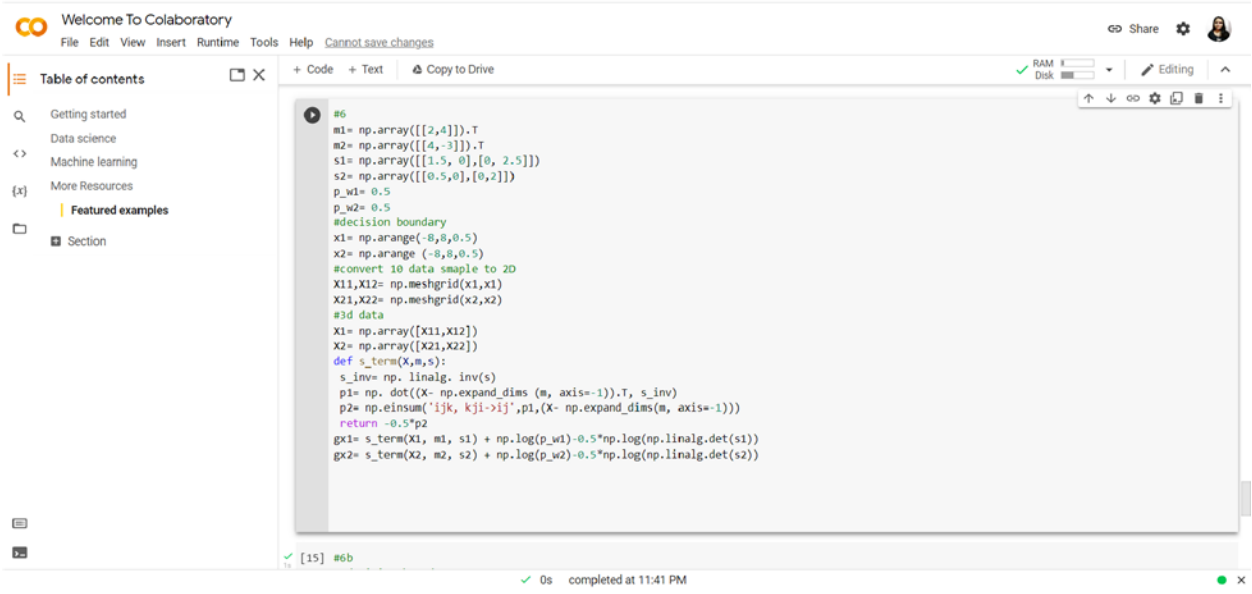
4. Classify test sample x_test = [3, 2]T using discriminant function for given data: m1 = [2, 4]T, m2= [4, -3]T, S1 = [[1.5, 0], [0, 2.5]], S2 = [[0.5, 0],[0, 2]], p(w1)=0.5, p(w2)=0.5.

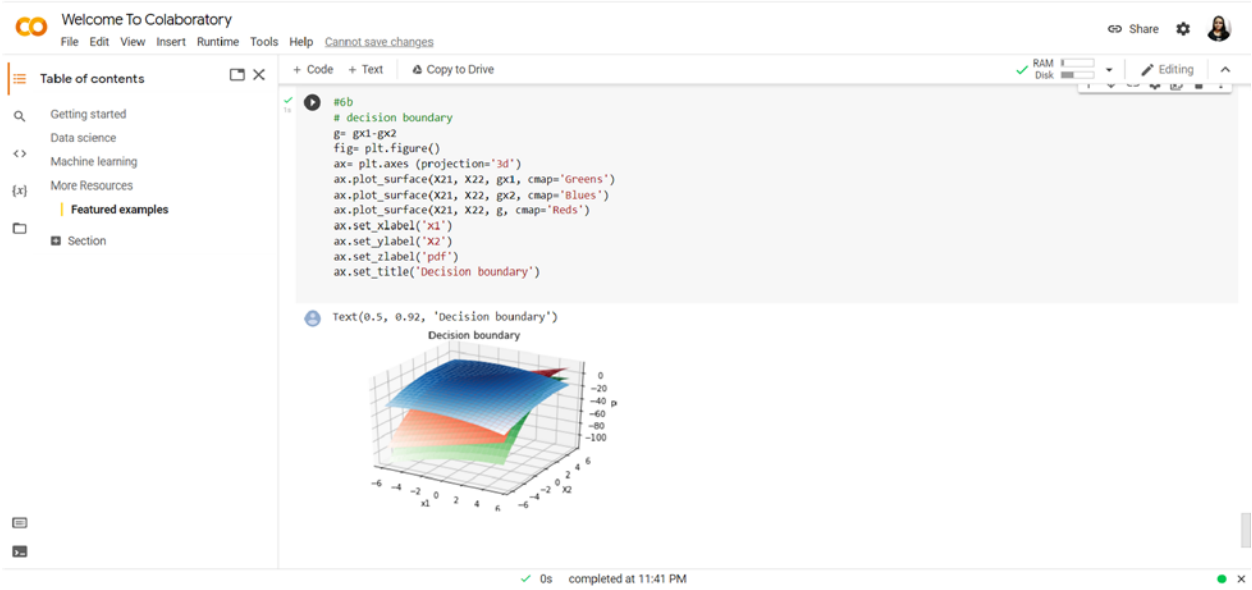


5. Classify test sample x_test = [2, 1]T for above data.



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 g1(x), g2(x) and g.





Conclusion: I concluded and studied that,

1) Discriminant Function Analysis is a dimensionality reduction technique that can be used for supervised classification problems.

   It is used for modeling differences in groups i.e. separating two or more classes.

2) A decision boundary is the line that separates one class from the other.

Practical - 6

Aim: Write a python program to implement Naïve Bayes Classifier.

Tool: Google colab

Theory:

Naïve Bayes Classifier.:

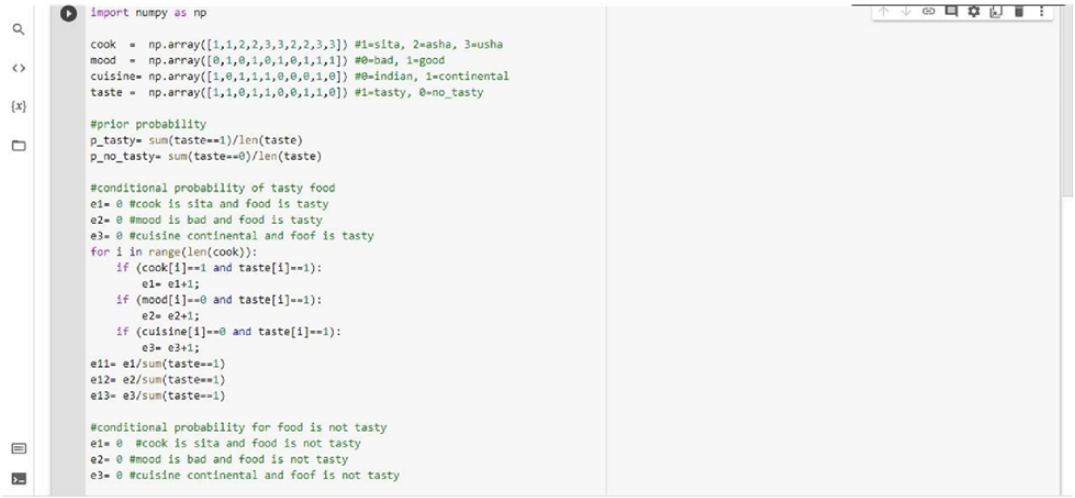
- Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

Codes and Outputs:

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



```
import numpy as np

cook   =  np.array([1,1,2,2,3,3,2,2,3,3]) #1=sita, 2=asha, 3=usha
mood   =  np.array([0,1,0,1,0,1,0,1,1,1]) #0=bad, 1=good
cuisine= np.array([1,0,1,1,1,0,0,0,1,0]) #0=indian, 1=continental
taste =  np.array([1,1,0,1,1,0,0,1,1,0]) #1=tasty, 0=no_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 foof is tasty
for i in range(len(cook)):
    if (cook[i]==1 and taste[i]==1):
        e1= e1+1;
    if (mood[i]==0 and taste[i]==1):
        e2= e2+1;
    if (cuisine[i]==0 and taste[i]==1):
        e3= e3+1;
e11= e1/sum(taste==1)
e12= e2/sum(taste==1)
e13= 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 foof is not tasty
```

```
for i in range(len(cook)):
    if (cook[i]==1 and taste[i]==0):
        e1= e1+1;
    if (mood[i]==0 and taste[i]==0):
        e2= e2+1;
    if (cuisine[i]==0 and taste[i]==0):
        e3= e3+1;
e21= e1/sum(taste==0)
e22= e2/sum(taste==0)
e23= e3/sum(taste==0)

#posterior probability
p_tasty_x= p_tasty*e11*e12*e13
p_no_tasty_x= p_no_tasty*e21*e22*e23

print('condition: Cook is Sita, Mood is Bad and preparing Continental 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 Sita, Mood is Bad and preparing Continental food:
food is tasty for given condition
```

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

```
#conditional probability of tasty food (cook is asha)
e1= 0
for i in range(len(cook)):
    if (cook[i]==2 and taste[i]==1):
        e1= e1+1;
#conditional probability for food is not tasty (cook is asha)
e4= 0
for i in range(len(cook)):
    if (cook[i]==2 and taste[i]==0):
        e1= e1+1;

E1= e1/sum(taste==1) #cook is asha and food is tasty conditional prob.
E2;                  #mood is bad and food is tasty conditional prob.
E3;                  #cuisine is continental & food is tasty conditional prob.
E4= e1/sum(taste==0) #cook is asha and food is not tasty
E5;                  #mood is bad and food is not tasty conditional prob.
E6;                  #cuisine is continental & food is not tasty conditional prob.

#posterior probability
p_tasty_x= p_tasty*E1*E2*E3
p_no_tasty_x= p_no_tasty*E4*E5*E6
print('condition: Cook is Asha, Mood is Bad and preparing Continental 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')
```

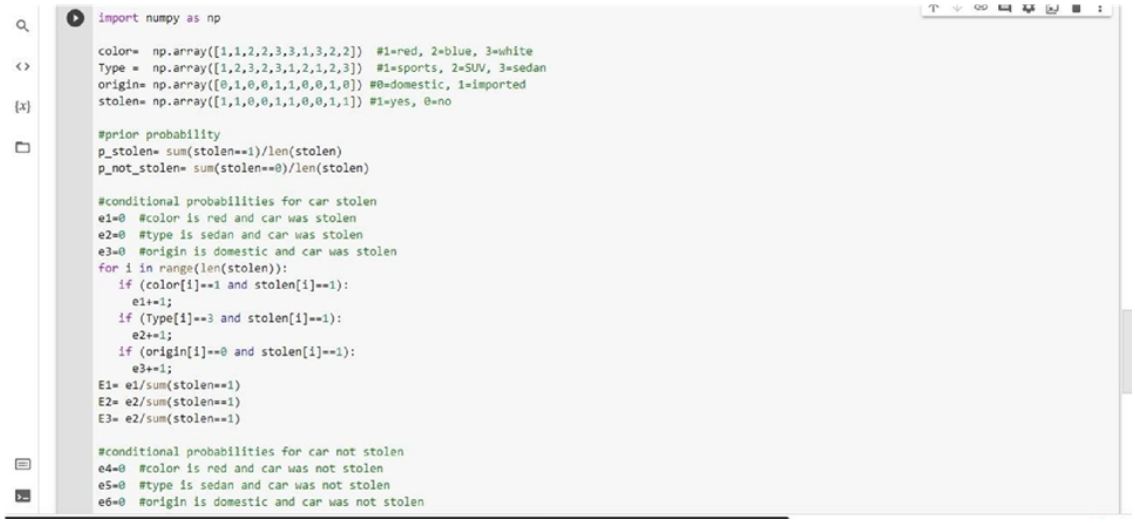
```
condition: Cook is Asha, Mood is Bad and preparing Continental food:
food is not tasty for given condition
```

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

```
#conditional probability of tasty food (cook is usha)
e1= 0
for i in range(len(cook)):
    if (cook[i]==3 and taste[i]==1):
        e1= e1+1;
#conditional probability for food is not tasty (cook is usha)
e4= 0
for i in range(len(cook)):
    if (cook[i]==3 and taste[i]==0):
        e1= e1+1;

E1= e1/sum(taste==1) #cook is usha and food is tasty conditional prob.
E2;                  #mood is bad and food is tasty conditional prob.
E3;                  #cuisine is continental & food is tasty conditional prob.
E4= e1/sum(taste==0) #cook is usha and food is not tasty
E5;                  #mood is bad and food is not tasty conditional prob.
E6;                  #cuisine is continental & food is not tasty conditional prob.

#posterior probability
p_tasty_x= p_tasty*E1*E2*E3
p_no_tasty_x= p_no_tasty*E4*E5*E6

print('condition: Cook is Asha, Mood is Bad and preparing Continental 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 Asha, Mood is Bad and preparing Continental food:
food is not tasty for given condition
```

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



```
import numpy as np

color=  np.array([1,1,2,2,3,3,1,3,2,2])  #1=red, 2=blue, 3=white
Type =  np.array([1,2,3,2,3,3,1,2,1,2,3])  #1=sports, 2=SUV, 3=sedan
origin= np.array([0,1,0,0,1,1,0,0,1,0]) #0=domestic, 1=imported
stolen= np.array([1,1,0,0,1,1,0,0,1,1]) #1=yes, 0=no

#prior probability
p_stolen= sum(stolen==1)/len(stolen)
p_not_stolen= sum(stolen==0)/len(stolen)

#conditional probabilities for car stolen
e1=0  #color is red and car was stolen
e2=0  #type is sedan and car was stolen
e3=0  #origin is domestic and car was stolen
for i in range(len(stolen)):
   if (color[i]==1 and stolen[i]==1):
     e1+=1;
   if (Type[i]==3 and stolen[i]==1):
     e2+=1;
   if (origin[i]==0 and stolen[i]==1):
     e3+=1;
E1= e1/sum(stolen==1)
E2= e2/sum(stolen==1)
E3= e2/sum(stolen==1)

#conditional probabilities for car not stolen
e4=0  #color is red and car was not stolen
e5=0  #type is sedan and car was not stolen
e6=0  #origin is domestic and car was not stolen
```

```
for i in range(len(stolen)):
  if (color[i]==1 and stolen[i]==0):
     e4+=1;
  if (Type[i]==3 and stolen[i]==0):
     e5+=1;
  if (origin[i]==0 and stolen[i]==0):
     e6+=1;
E4= e4/sum(stolen==0)
E5= e5/sum(stolen==0)
E6= e6/sum(stolen==0)

#posterior probability
p_stolen_x = p_stolen*E1*E2*E3
p_not_stolen_x= p_not_stolen*E4*E5*E6
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 Color is White, Type is SUV and Origin is Imported, was car stolen?

```
#conditional probabilities for car stolen and not stolen
e1=0  #color is white and car was stolen
e2=0  #type is SUV and car was stolen
e3=0  #origin is imported and car was stolen
e4=0  #color is white and car was not stolen
e5=0  #type is SUV and car was not stolen
e6=0  #origin is imported and car was not stolen

for i in range(len(stolen)):
   if (color[i]==3 and stolen[i]==1):
     e1+=1;
   if (Type[i]==2 and stolen[i]==1):
     e2+=1;
   if (origin[i]==1 and stolen[i]==1):
     e3+=1;
   if (color[i]==3 and stolen[i]==0):
     e4+=1;
   if (Type[i]==2 and stolen[i]==0):
     e5+=1;
   if (origin[i]==1 and stolen[i]==0):
     e6+=1;

E1= e1/sum(stolen==1)
E2= e2/sum(stolen==1)
E3= e2/sum(stolen==1)
E4= e4/sum(stolen==0)
E5= e5/sum(stolen==0)
E6= e6/sum(stolen==0)

#posterior probability
```

```
#posterior probability
p_stolen_x = p_stolen*E1*E2*E3
p_not_stolen_x= p_not_stolen*E4*E5*E6

print('condition: Color is White, Type is SUV and Origin is Imported :')

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 White, Type is SUV and Origin is Imported :
The car is stolen for given condition
```

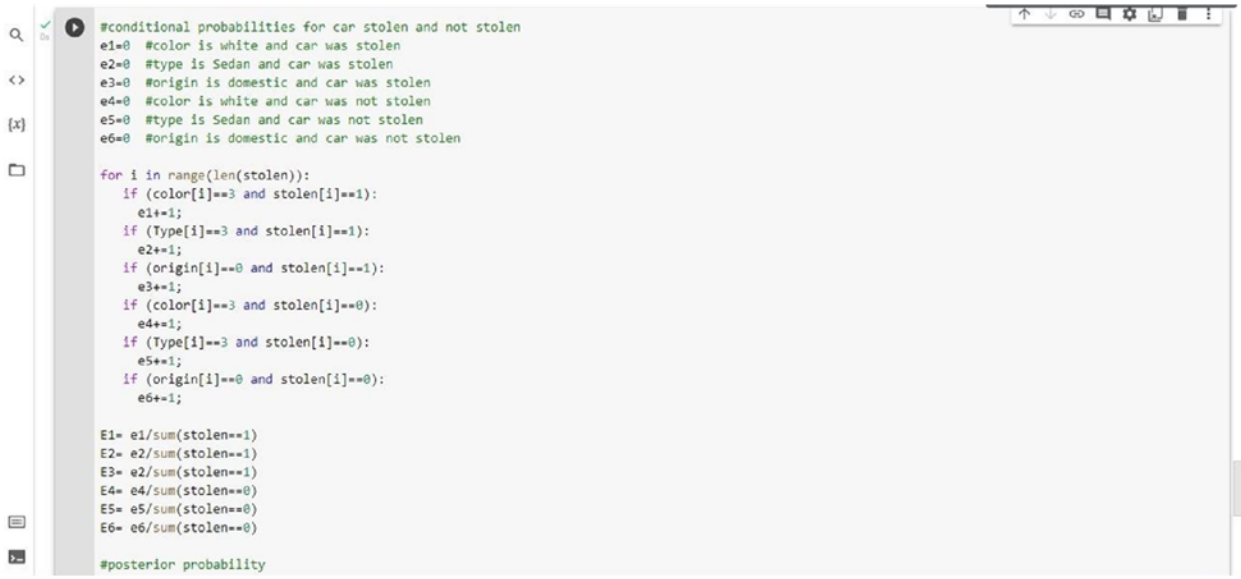
6.If car Color is Blue, Type is Sports and Origin is Domestic, was car stolen?

```
#conditional probabilities for car stolen and not stolen
e1=0  #color is blue and car was stolen
e2=0  #type is Sports and car was stolen
e3=0  #origin is domestic and car was stolen
e4=0  #color is blue and car was not stolen
e5=0  #type is Sports and car was not stolen
e6=0  #origin is domestic and car was not stolen

for i in range(len(stolen)):
   if (color[i]==2 and stolen[i]==1):
     e1+=1;
   if (Type[i]==1 and stolen[i]==1):
     e2+=1;
   if (origin[i]==0 and stolen[i]==1):
     e3+=1;
   if (color[i]==2 and stolen[i]==0):
     e4+=1;
   if (Type[i]==1 and stolen[i]==0):
     e5+=1;
   if (origin[i]==0 and stolen[i]==0):
     e6+=1;

E1= e1/sum(stolen==1)
E2= e2/sum(stolen==1)
E3= e2/sum(stolen==1)
E4= e4/sum(stolen==0)
E5= e5/sum(stolen==0)
E6= e6/sum(stolen==0)

#posterior probability
```

```
#posterior probability
p_stolen_x = p_stolen*E1*E2*E3
p_not_stolen_x= p_not_stolen*E4*E5*E6

print('condition: Color is Blue, Type is Sports 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 Blue, Type is Sports and Origin is Domestic :
The car is not stolen for given condition
```

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

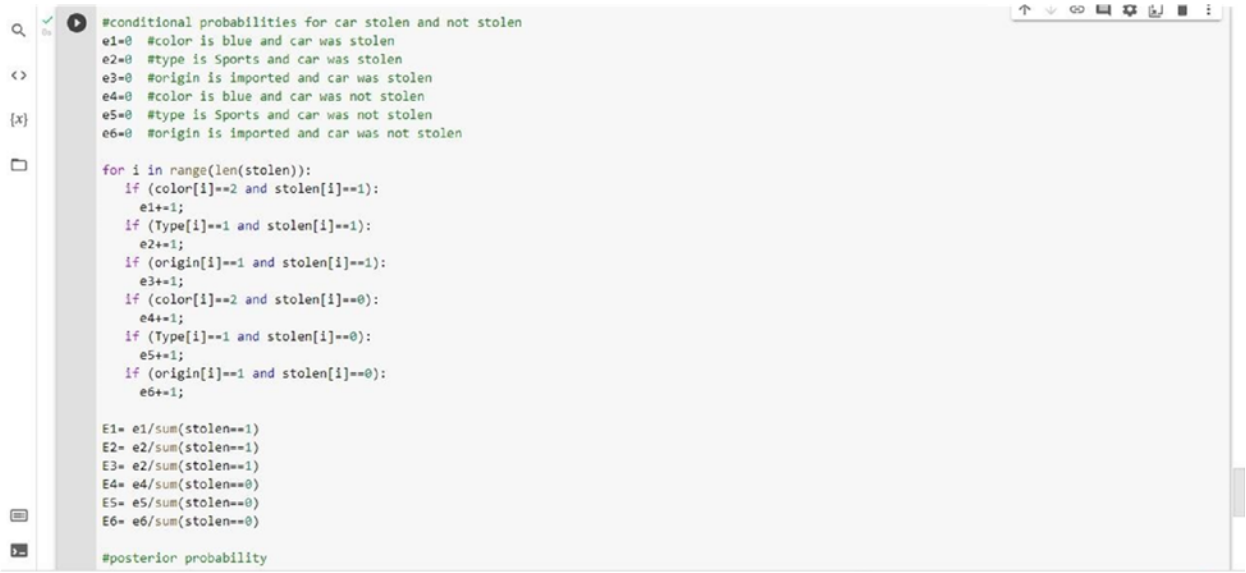


```
#conditional probabilities for car stolen and not stolen
e1=0  #color is white and car was stolen
e2=0  #type is Sedan and car was stolen
e3=0  #origin is domestic and car was stolen
e4=0  #color is white and car was not stolen
e5=0  #type is Sedan and car was not stolen
e6=0  #origin is domestic and car was not stolen

for i in range(len(stolen)):
   if (color[i]==3 and stolen[i]==1):
     e1+=1;
   if (Type[i]==3 and stolen[i]==1):
     e2+=1;
   if (origin[i]==0 and stolen[i]==1):
     e3+=1;
   if (color[i]==3 and stolen[i]==0):
     e4+=1;
   if (Type[i]==3 and stolen[i]==0):
     e5+=1;
   if (origin[i]==0 and stolen[i]==0):
     e6+=1;

E1= e1/sum(stolen==1)
E2= e2/sum(stolen==1)
E3= e2/sum(stolen==1)
E4= e4/sum(stolen==0)
E5= e5/sum(stolen==0)
E6= e6/sum(stolen==0)

#posterior probability
```

```
#posterior probability
p_stolen_x = p_stolen*E1*E2*E3
p_not_stolen_x= p_not_stolen*E4*E5*E6

print('condition: Color is White, 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 White, Type is Sedan and Origin is Domestic :
The car is not stolen for given condition
```

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

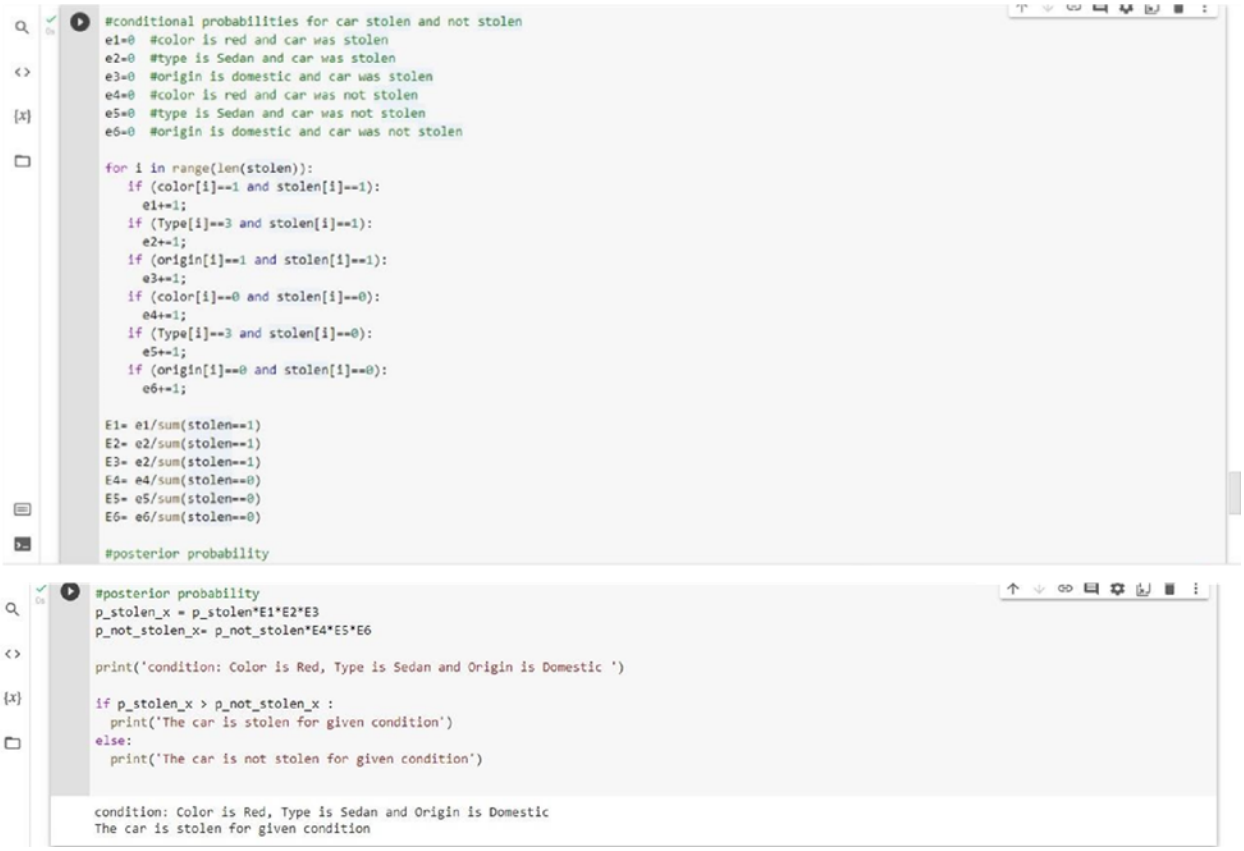


```
#conditional probabilities for car stolen and not stolen
e1=0  #color is blue and car was stolen
e2=0  #type is Sports and car was stolen
e3=0  #origin is imported and car was stolen
e4=0  #color is blue and car was not stolen
e5=0  #type is Sports and car was not stolen
e6=0  #origin is imported and car was not stolen

for i in range(len(stolen)):
   if (color[i]==2 and stolen[i]==1):
     e1+=1;
   if (Type[i]==1 and stolen[i]==1):
     e2+=1;
   if (origin[i]==1 and stolen[i]==1):
     e3+=1;
   if (color[i]==2 and stolen[i]==0):
     e4+=1;
   if (Type[i]==1 and stolen[i]==0):
     e5+=1;
   if (origin[i]==1 and stolen[i]==0):
     e6+=1;

E1= e1/sum(stolen==1)
E2= e2/sum(stolen==1)
E3= e2/sum(stolen==1)
E4= e4/sum(stolen==0)
E5= e5/sum(stolen==0)
E6= e6/sum(stolen==0)

#posterior probability
```

```
#posterior probability
p_stolen_x = p_stolen*E1*E2*E3
p_not_stolen_x= p_not_stolen*E4*E5*E6

print('condition: Color is Blue, Type is Sports and Origin is Imported :')

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 Blue, Type is Sports and Origin is Imported :
The car is stolen for given condition
```

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



```
#conditional probabilities for car stolen and not stolen
e1=0  #color is red and car was stolen
e2=0  #type is Sedan and car was stolen
e3=0  #origin is domestic and car was stolen
e4=0  #color is red and car was not stolen
e5=0  #type is Sedan and car was not stolen
e6=0  #origin is domestic and car was not stolen

for i in range(len(stolen)):
   if (color[i]==1 and stolen[i]==1):
     e1+=1;
   if (Type[i]==3 and stolen[i]==1):
     e2+=1;
   if (origin[i]==1 and stolen[i]==1):
     e3+=1;
   if (color[i]==0 and stolen[i]==0):
     e4+=1;
   if (Type[i]==3 and stolen[i]==0):
     e5+=1;
   if (origin[i]==0 and stolen[i]==0):
     e6+=1;

E1= e1/sum(stolen==1)
E2= e2/sum(stolen==1)
E3= e2/sum(stolen==1)
E4= e4/sum(stolen==0)
E5= e5/sum(stolen==0)
E6= e6/sum(stolen==0)

#posterior probability
```

```
#posterior probability
p_stolen_x = p_stolen*E1*E2*E3
p_not_stolen_x= p_not_stolen*E4*E5*E6

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 stolen for given condition
```

Conclusion:

A) Naïve Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem to predict outcome.

B)Naïve Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem to predict outcome.

C) They are fast and easy to implement.

Practical - 7

Aim: Write a python program to plot Receiver Operating Characteristics (ROC) curves.

Tool: Google colab

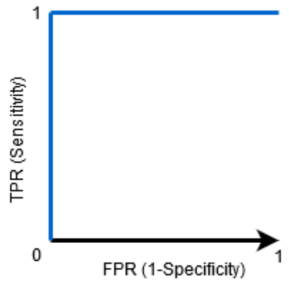
Theory:

Receiver Operating Characteristics (ROC) curves:

Receiver Operating Characteristics (ROC) curves.

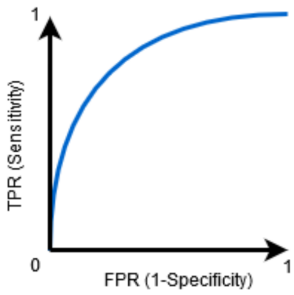
The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that plots the TPR (True Positive Rate) against FPR (False Positive Rate) at various threshold values and essentially separates the 'signal' from the 'noise'. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

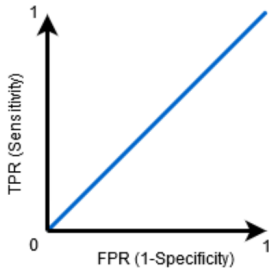


When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points

correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.



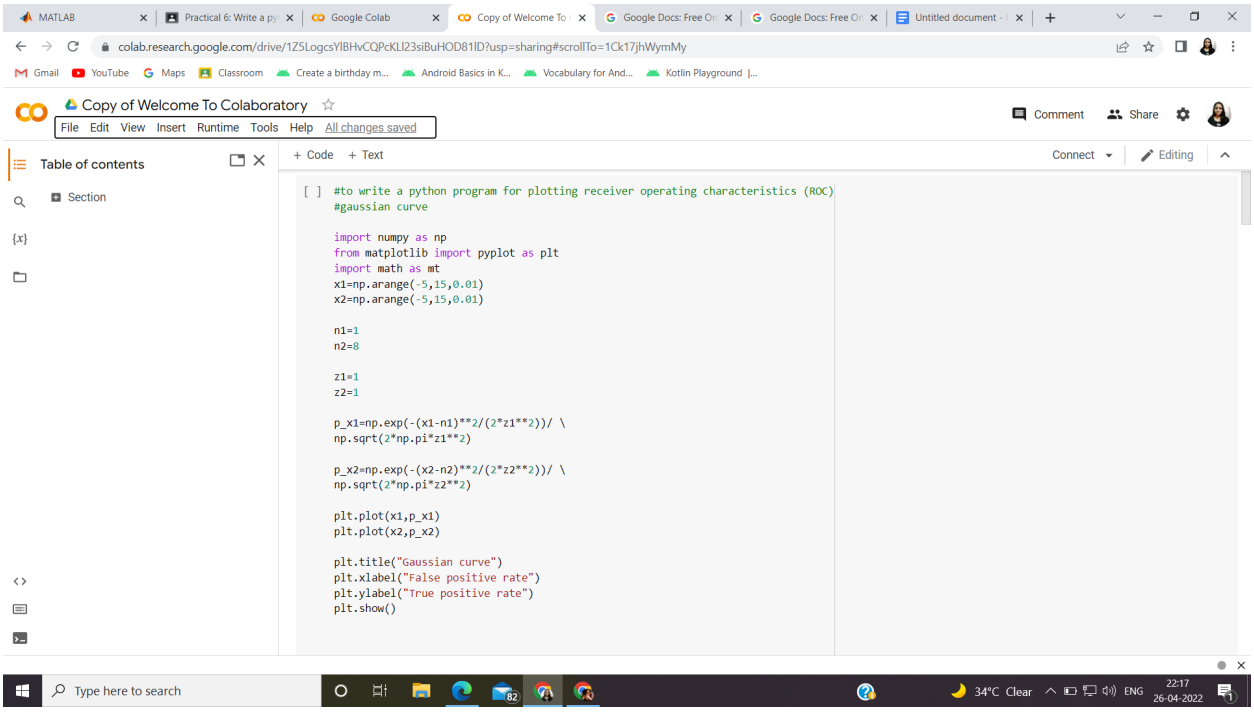
When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.



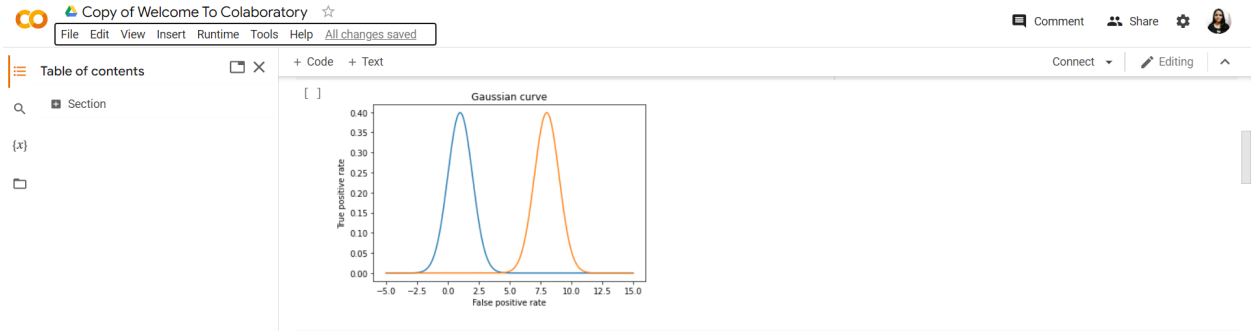
When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning either the classifier is predicting random class or constant class for all the data points.

So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.

Codes and Outputs:



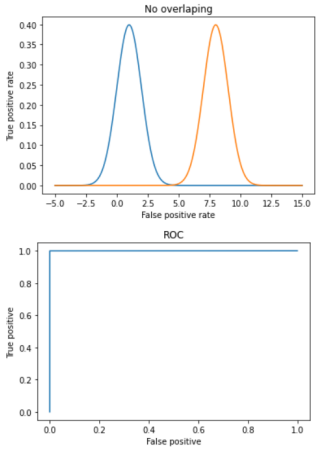
```
#to write a python program for plotting receiver operating characteristics (ROC)
#gaussian curve

import numpy as np
from matplotlib import pyplot as plt
import math as mt
x1=np.arange(-5,15,0.01)
x2=np.arange(-5,15,0.01)

n1=1
n2=8

z1=1
z2=1

p_x1=np.exp(-(x1-n1)**2/(2*z1**2))/ \
np.sqrt(2*np.pi*z1**2)

p_x2=np.exp(-(x2-n2)**2/(2*z2**2))/ \
np.sqrt(2*np.pi*z2**2)

plt.plot(x1,p_x1)
plt.plot(x2,p_x2)

plt.title("Gaussian curve")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```



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

```
#No overlaping
#for ROC

import numpy as np
from matplotlib import pyplot as plt
import math as mt
x1=np.arange(-5,15,0.01)
x2=np.arange(-5,15,0.01)

n1=1
n2=8

z1=1
z2=1

p_x1=np.exp(-(x1-n1)**2/(2*z1**2))/ \
np.sqrt(2*np.pi*z1**2)

p_x2=np.exp(-(x2-n2)**2/(2*z2**2))/ \
np.sqrt(2*np.pi*z2**2)

plt.plot(x1,p_x1)
plt.plot(x2,p_x2)

plt.title("No overlaping")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```

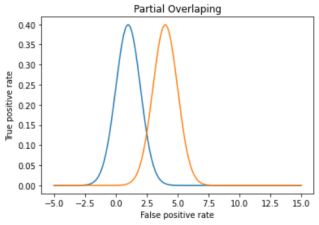
```
tp=np.zeros(len(x1))
fp=np.zeros(len(x1))
for i in range(len(x1)):
    tp[i]=sum((p_x2*(x2>=x2[i])))
    fp[i]=sum((p_x1*(x1>=x1[i])))

plt.plot(fp/100,tp/100)
plt.title("ROC")
plt.xlabel("False positive")
plt.ylabel("True positive")
plt.show()
```

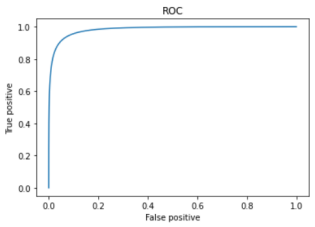


2.Plot the “Partial Overlap” condition of two Gaussian distributions. Also plot the ROC curve for same. Give proper labeling to plot.

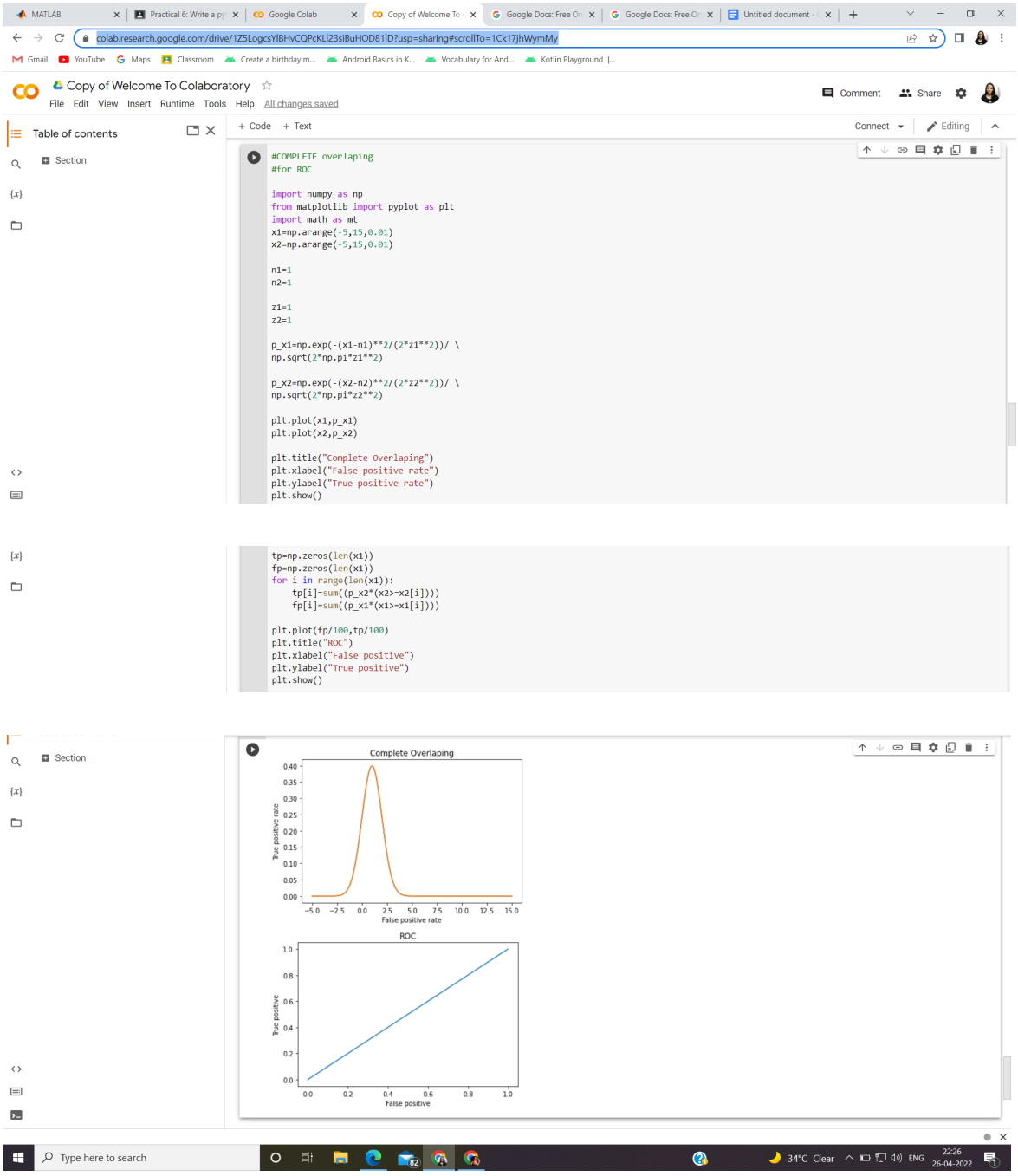
```
#partial overlaping
#for ROC

import numpy as np
from matplotlib import pyplot as plt
import math as mt
x1=np.arange(-5,15,0.01)
x2=np.arange(-5,15,0.01)

n1=1
n2=4

z1=1
z2=1

p_x1=np.exp(-(x1-n1)**2/(2*z1**2))/ \
np.sqrt(2*np.pi*z1**2)

p_x2=np.exp(-(x2-n2)**2/(2*z2**2))/ \
np.sqrt(2*np.pi*z2**2)

plt.plot(x1,p_x1)
plt.plot(x2,p_x2)

plt.title("Partial Overlaping")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.show()
```

```
tp=np.zeros(len(x1))
fp=np.zeros(len(x1))
for i in range(len(x1)):
    tp[i]=sum((p_x2*(x2>=x2[i])))
    fp[i]=sum((p_x1*(x1>=x1[i])))

plt.plot(fp/100,tp/100)
plt.title("ROC")
plt.xlabel("False positive")
plt.ylabel("True positive")
plt.show()
```





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



Conclusion: Thus I concluded that the Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems.

Practical - 8

Aim: Write a python program for Multivariate Regression Modeling.

Tool: Google colab

Theory:

Multivariate Regression Modeling.:

As the name implies, multivariate regression is a technique that estimates a single regression model with more than one outcome variable. When there is more than one predictor variable in a multivariate regression model, the model is a multivariate multiple regression.

Regression is a statistical method used in finance, investing, and other disciplines that attempts to determine the strength and character of the relationship between one dependent variable and a series of other variables(independent variables).

Multivariable Regression is a Machine Learning algorithm that helps establish the relationship between a dependent variable and more than one independent variable.

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 behavior of the response variables depending on how the predictor variables move.

Multivariate multiple regression:

Multivariate multiple regression (MMR) is used to model the linear relationship between more than one independent variable (IV) and more than one dependent variable (DV). MMR is multiple because there is more than one IV. MMR is multivariate because there is more than one DV.

Codes and Outputs:

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

```
#Calculating regression coefficient
def regCoeff(yn,x):
  xt_x =np.matmul(x.transpose(),x) #xT.x
  xt_x_inv = np.linalg.inv(xt_x) #(xT.x)^-1
  xty = np.matmul(x.transpose(),yn) #(xty)
  beta = np.matmul(xt_x_inv,xty) # ((xTx)^-1).xTy)
  return beta

#Power consumtion in kw
def powerConsumption(beta,a,b):
  y1 = beta[0] + a*beta[1]+b*beta[2]
  print("Room Tempt {} No. of person {}.\nThe power consumtion is {} kw".format(a,b,y1[0]))
  return
```

```
# 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.
y = [[0.5],[2.5],[2.2],[1.8]]
yn = np.array(y)
x = np.array([[1,10,2],[1,5,42],[1,4.5,31],[1,6,35]])
beta = regCoeff(yn,x)
print(beta)
```

```
[[ 2.49761588]
 [-0.20751687]
 [ 0.02092015]]
```

```
powerConsumption(beta,7.5,131)
```

```
Room Tempt 7.5 No. of person 131.
The power consumtion is 3.6817792226984043 kw
```

```
powerConsumption(beta,3.5,45)
```

```
Room Tempt 3.5 No. of person 45.
The power consumtion is 2.7127136594256633 kw
```

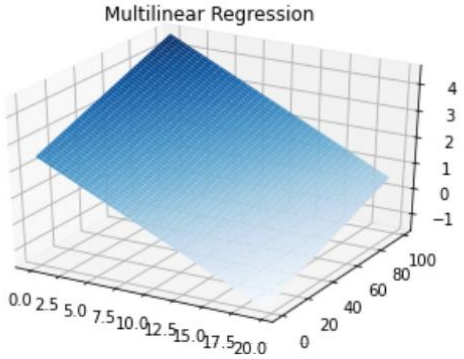
```
powerConsumption(beta,7.2,27)
```

```
Room Tempt 7.2 No. of person 27.
The power consumtion is 1.568338505492359 kw
```

```
#3. Plot the 3D graph for power consumption data using multivariate
# regression coefficients. (fit the hyperplane)
x1 = np.arange(0,20,0.5)
x2 = np.arange(0,100,1)
```

```
#convert 1d data to 2d data
x11,x12 = np.meshgrid(x1,x2)
y_n = beta[0] + x11*beta[1] + x12*beta[2]

fig = plt.figure()
axs = plt.axes(projection = '3d')
axs.plot_surface(x11,x12,y_n,cmap='Blues')
axs.set_title('Multilinear Regression')
```

```
Text(0.5, 0.92, 'Multilinear Regression')
```




```
data = pd.read_csv('insur-2d.csv')
data.insert(0,'Const',1)
data.columns = ["Const", "Input 1", "Input 2", "Output"]
input = np.matrix([data["Const"].tolist(),data["Input 1"].tolist(),data["Input 2"].tolist(
opt = np.matrix(data["Output"].tolist()).T
```

```
beta_N = regCoeff(opt,input)
print("B0: ",beta_N[0],'\nB1: ',beta_N[1],'\nB2: ',beta_N[2])
```

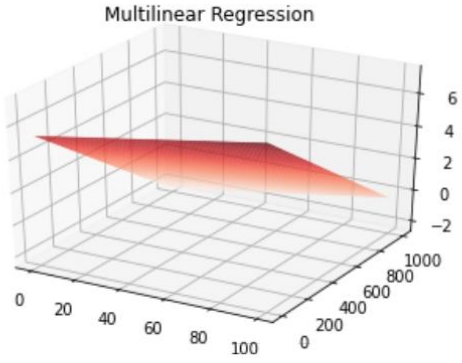
```
B0:  [[5.35320784]]
B1:  [[0.0210913]]
B2:  [[-0.00781811]]
```

```
#5. Predict the three outputs for unknown three pair of inputs.
x1 = np.array([1,7.5,131])
o1 = np.dot(x1,beta_N)
print("The input: {} -> the output: {}".format(x1[1:3],o1[0]))
#o/p 2
x2 = np.array([1,3.5,45])
o2 = np.dot(x2,beta_N)
print("The input: {} -> the output: {}".format(x2[1:3],o2[0]))
#O/P 3
x3 = np.array([1,7.2,27])
o3 = np.dot(x3,beta_N)
print("The input: {} -> the output: {}".format(x3[1:3],o3[0]))
```

```
The input: [  7.5 131. ] -> the output: [[4.48722012]]
The input: [ 3.5 45. ] -> the output: [[5.07521242]]
The input: [ 7.2 27. ] -> the output: [[5.29397621]]
```

```
#6. Plot the 3D graph for given data of csv file using multivariate regression
# coefficients. (fit the hyperplane)
x1 = np.arange(0,100,0.5)
x2 = np.arange(0,1000,1)

#convert 1d data to 2d data
x11,x12 = np.meshgrid(x1,x2)
y_n = float(beta_N[0]) + x11*float(beta_N[1]) + x12*float(beta_N[2])

fig = plt.figure()
axs = plt.axes(projection = '3d')
axs.plot_surface(x11,x12,y_n,cmap='Reds')
axs.set_title('Multilinear Regression')
```

```
Text(0.5, 0.92, 'Multilinear Regression')
```



✓ 1s completed at 11:12 PM

Conclusion:

1) Multivariate regression allows one to have a different view of the relationship between various variables from all the possible angles.
2) It helps you to predict the behavior of the response variables depending on how the predictor variables move.

Practical - 9

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

Tool: Google colab

Theory:

Linear Regression using Gradient Descent algorithm:

A linear regression model attempts to explain the relationship between a dependent (output variables) variable and one or more independent (predictor variable) variables using a straight line.

This straight line is represented using the following formula:

$$y = mx + c$$

Where, y: dependent variable

x: independent variable

m: Slope of the line (For a unit increase in the quantity of X, Y increases by m.1 = m units.)

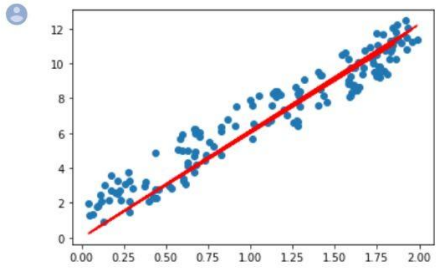
Codes and Outputs:

```
#Priya Kishor Kshirsagar_B59_2019BEC134
```

```
#Write a python program for Linear Regression using Gradient Descent algorithm.
```

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

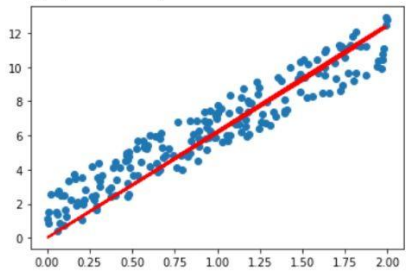
```
#1. Create an own dataset containing (150 random values for input and output) using numpy library.
#Define the function for computing regression coefficients using gradient descent. (consider c=0.001, b0=0.5, b1=1.3, E=0, Emax=0.001

def Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs):
    plt.scatter(x,y)
    y_new=np.array(np.zeros(len(x)));

    for j in range(epochs):
        for i in range(len(x)):
            y_new[i]=b0+b1*x[i];
            E=(y[i]-y_new[i])**2;

            if E> E_max:
                b0=C*(y[i]-y_new[i]);
                b1=b1+C*(y[i]-y_new[i])*x[i];

    plt.plot(x,y_new,color="red")

# for c=0.001, b0=0.5, b1=1.3, E=0, Emax=0.001 and epochs=5000
x=2*np.random.rand(150,1)
y=5*x+3*np.random.rand(150,1)

E_max=0.001
E=0
C=0.01
epochs=5000

b0=0.5
b1=1.3

Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs)
```




```
#2. Print the coefficients. Plot (scatter) the input and output data.
#Also plot the linear regression line using coefficients.

from math import inf
def Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs):
    plt.scatter(x,y)
    y_new=np.array(np.zeros(len(x)));

    for j in range(epochs):
        for i in range(len(x)):
          y_new[i]=b0+b1*x[i];
          E=(y[i]-y_new[i])**2;

          if E> E_max and E!=inf:
              b0=C*(y[i]-y_new[i]);
              b1=b1+C*(y[i]-y_new[i])*x[i];


    plt.plot(x,y_new,color="red")
    print(f"Bita_0={b0}")
    print(f"Bita_1={b1}")
```

```
# for c=0.001, b0=0.5, b1=1.3, E=0, Emax=0.001 and epochs=5000
x=2*np.random.rand(200,1)
y=5*x+3*np.random.rand(200,1)

E_max=0.001
E=0
C=0.01
epochs=5000

b0=0.5
b1=1.3

Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs)
```

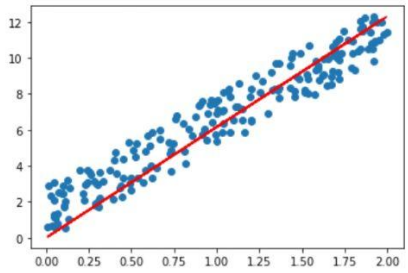
```
Bita_0=[0.00640505]
Bita_1=[6.22225886]
```




```
#3. Compute the linear regression coefficients using  gradient descent function for given data and print it for various learning cons
#Also print the regression line for various learning constant and various epochs.
```

```
#For c=0.001 & epochs=1000

x=2*np.random.rand(200,1)
y=5*x+3*np.random.rand(200,1)

E_max=0.001
E=0
C=0.001
epochs=1000

b0=0.5
b1=1.3

Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs)
```

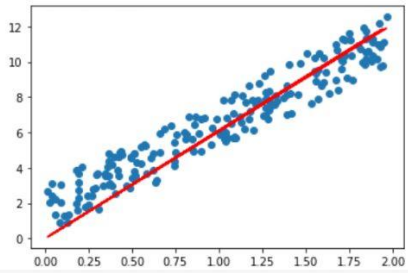
```
Bita_0=[3.30237619e-05]
Bita_1=[6.15412364]
```




```
#For c=0.005 & epochs=2000

x=2*np.random.rand(200,1)
y=5*x+3*np.random.rand(200,1)
E_max=0.001
E=0
C=0.005
epochs=2000

b0=0.5
b1=1.3
```

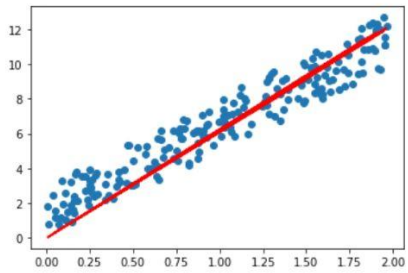
```
Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs)
```

```
Bita_0=[0.00794556]
Bita_1=[6.09440153]
```




```
#For c=0.01 epochs=3000

x=2*np.random.rand(200,1)
y=5*x+3*np.random.rand(200,1)

E_max=0.001
E=0
C=0.01
epochs=3000

b0=0.5
b1=1.3

Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs)
```

```
Bita_0=[-0.0078852]
Bita_1=[6.10419053]
```




```
#For c=0.05 & epochs=5000

x=2*np.random.rand(200,1)
y=5*x+3*np.random.rand(200,1)


E_max=0.001
E=0
C=0.05
epochs=5000

b0=0.5
b1=1.3

Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs)
```

```
Bita_0=[-0.05852399]
Bita_1=[5.94671977]
```

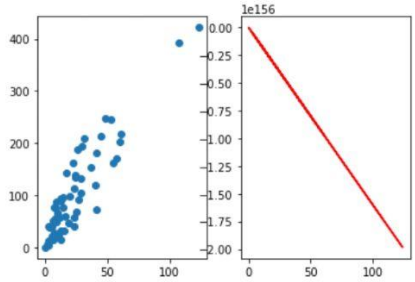
```
#4. Read the attached insu.csv file using Pandas library. Find the linear regression coefficients using gradient descent for given fi
#(Initialize the value as per your convenience)

from math import inf
def Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs):
    y_new=np.array(np.zeros(len(x)));

    for j in range(epochs):
        for i in range(len(x)):
          y_new[i]=b0+b1*x[i];
          E=(y[i]-y_new[i])**2;

          if E> E_max and E!=inf:
              b0=C*(y[i]-y_new[i]);
              b1=b1+C*(y[i]-y_new[i])*x[i];

    print(f"Bita_0={b0}")
    print(f"Bita_1={b1}")

data=pd.read_csv("/content/insu (1).csv")

x=np.array([data['X_Data']])
y=np.array([data['Y']])


x=x.T
y=y.T

E_max=0.001
E=0
C=0.005
epochs=5000

b0=1.5
b1=0.5
Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:10: RuntimeWarning: overflow encountered in square
  # Remove the CWD from sys.path while we load stuff.
Bita_0=[-2.58867204e+149]
Bita_1=[-4.74082577e+150]
```

```
#5. Plot the variables containing the values from csv file. Also plot the linear regression line using coefficients.

from math import inf
def Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs):

    y_new=np.array(np.zeros(len(x)));

    for j in range(epochs):
        for i in range(len(x)):
          y_new[i]=b0+b1*x[i];
          E=(y[i]-y_new[i])**2;

          if E> E_max and E!=inf:
              b0=C*(y[i]-y_new[i]);
              b1=b1+C*(y[i]-y_new[i])*x[i];

    plt.subplot(1,2,2)
    plt.plot(x,y_new,color='red')
    print(f"Bita_0={b0}")
    print(f"Bita_1={b1}")

data=pd.read_csv("/content/insu (1).csv")
x=np.array([data['X_Data']])
y=np.array([data['Y']])

x=x.T
y=y.T
E_max=0.01
E=0
C=0.05
epochs=1000

b0=0.5
b1=1.3
```

```
b0 = np.float128(b0)
b1 = np.float128(b1)
plt.subplot(1,2,1)
plt.scatter(x,y)
Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: RuntimeWarning: overflow encountered :
  # This is added back by InteractiveShellApp.init_path()
Bita_0=[0.00994515]
Bita_1=[-1.59965707e+154]
```




```
#6.Check the coefficient values computed by co-variance method and gradient descent method for insu.csv file.

from math import inf
def Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs):

    y_new=np.array(np.zeros(len(x)));

    for j in range(epochs):
        for i in range(len(x)):
          y_new[i]=b0+b1*x[i];
          E=(y[i]-y_new[i])**2;

          if E> E_max and E!=inf:
              b0=C*(y[i]-y_new[i]);
              b1=b1+C*(y[i]-y_new[i])*x[i];

    print(f"Bita_0={b0}")
    print(f"Bita_1={b1}")

x=2*np.random.rand(200,1)
y=5*x+3*np.random.rand(200,1)

E_max=0.01
E=0
C=0.05
epochs=1000

b0=0.5
b1=1.3
b0 = np.float128(b0)
b1 = np.float128(b1)
Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs)
```

```
Bita_0=[-0.08699529]
Bita_1=[6.20964105]
```

```
#7. Find the predicted output for unknown input values using linear regression coefficients and print it.(for insu.csv dataset)

from math import inf
def Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs,input):

    y_new=np.array(np.zeros(len(x)));

    for j in range(epochs):
        for i in range(len(x)):
          y_new[i]=b0+b1*x[i];
          E=(y[i]-y_new[i])**2;

          if E> E_max and E!=inf:
              b0=C*(y[i]-y_new[i]);
              b1=b1+C*(y[i]-y_new[i])*x[i];

    y_pred=b0+b1*input;
```

```
    print(f"Predicted output = {y_pred} for the input = {input}")

data=pd.read_csv("/content/insu (1).csv")

x=np.array([data['X_Data']])
y=np.array([data['Y']])

x=x.T
y=y.T

x=2*np.random.rand(200,1)
y=2*x+3*np.random.rand(200,1)
E_max=0.01
E=0
C=0.05
epochs=1000

b0=0.5
b1=1.3


Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs,5.10)
Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs,10.90)
Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs,50.10)
Descent_Gradiant(x,y,E_max,E,C,b0,b1,epochs,5.10)
```

```
Predicted output = [16.97713977] for the input = 5.1
Predicted output = [36.23229243] for the input = 10.9
Predicted output = [166.37056558] for the input = 50.1
Predicted output = [16.97713977] for the input = 5.1
```

✓ 16s completed at 3:25 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

Conclusion:

Thus I studied Linear Regression using the Gradient Descent algorithm. I also studied how a linear regression model attempts to explain the relationship between a dependent (output variables) variable and one or more independent (predictor variable) variables using a straight line.

Practical - 10

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

Tool: Google colab

Theory:

Parzen Window:

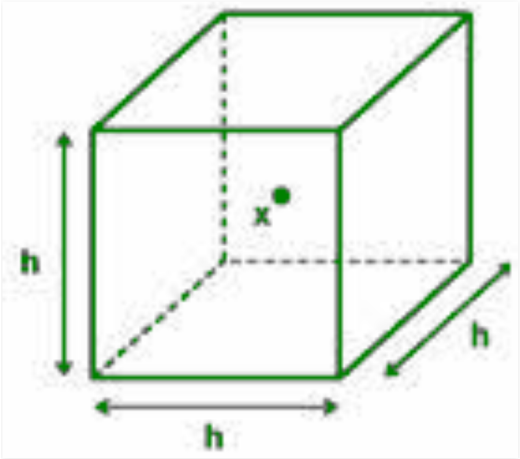
The Parzen window is a non-parametric density estimation technique. Density estimation in Pattern Recognition can be achieved by using the approach of the Parzen Windows. Parzen window density estimation technique is a kind of generalization of the histogram technique.

It is used to derive a density function, $f(x)$.

$f(x)$ is used to implement a Bayes Classifier. When we have a new sample feature 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.

An n-dimensional hypercube is considered which is assumed to possess k-data samples.



Codes and Outputs:

```
#Priya Kishor Kshirsagar_B59_2019BEC134
```

```
#Practical 10
```

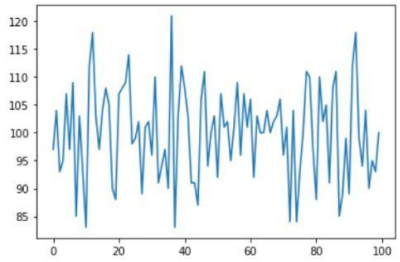
```
#To write a python program to implement Parzen Window Density estimation for any randomly generated signal.
```

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

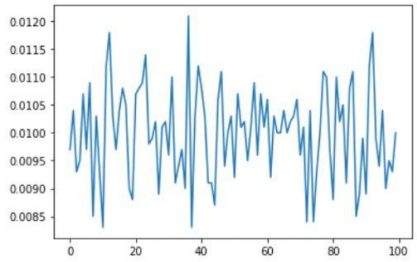
```
#1.Generate a random signal of size 10,000 in the range 0 to 99.

x=np.random.randint(100,size=10000)
print(x)
```

```
[70 77 86 ...  4 19 43]
```

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

def Count_Occurence(x):
    y=np.zeros(100);

    for i in range(100):
        for j in range(10000):
              if(x[j]==i):
                  y[i]=y[i]+1;
    plt.plot(y)

Count_Occurence(x)
```



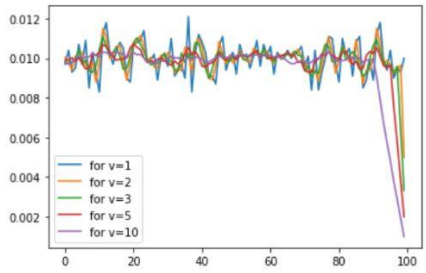

```
#3.Plot the probability of each sample.

def Count_Prob(x):
    y=np.zeros(100);

    for i in range(100):
        for j in range(len(x)):
              if(x[j]==i):
                  y[i]=y[i]+1;
        y[i]/=len(x)
    plt.plot(y)

Count_Prob(x)
```



```
#4.Define the function to compute the Parzen window density using formula.
#(pdf = k/n/V , where k-> no. of samples fall within window, n-> total samples, V->width of window)

def Parzen_window(w,x):
    y=np.zeros(100);

    for i in range(100):
        for j in range(len(x)):
            for k in range(w):
                if(x[j]==i+k):
                    y[i]=y[i]+1;
        y[i]=(y[i]/len(x))/w;
    print(y)

Parzen_window(3,x)
```

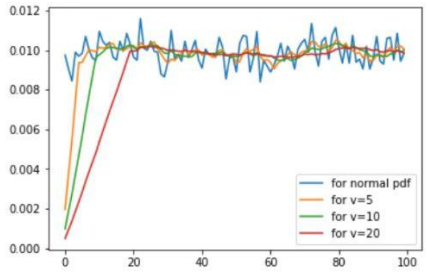
```
 [0.0098     0.00973333 0.00983333 0.00996667 0.01043333 0.0097
  0.0099     0.00936667 0.0093     0.0096     0.01043333 0.0111
  0.0106     0.01013333 0.0103     0.01056667 0.0101     0.00943333
  0.0095     0.0101     0.0108     0.01103333 0.0107     0.01036667
  0.00996667 0.00966667 0.00973333 0.00973333 0.00996667 0.01026667
  0.0099     0.00983333 0.0094     0.00936667 0.01026667 0.0098
  0.01023333 0.00993333 0.01076667 0.01076667 0.01006667 0.0095
  0.00896667 0.00946667 0.01013333 0.01036667 0.01016667 0.0099
  0.00983333 0.01006667 0.01       0.01033333 0.00993333 0.00993333
  0.01016667 0.0102     0.0104     0.01013333 0.01046667 0.00996667
  0.01003333 0.00983333 0.0101     0.01013333 0.01013333 0.0102
  0.01016667 0.01036667 0.01016667 0.0101     0.00936667 0.00963333
  0.00906667 0.00936667 0.00923333 0.01013333 0.0107     0.0106
  0.00983333 0.00983333 0.01       0.01056667 0.00993333 0.01013333
  0.01033333 0.01013333 0.0095     0.0091     0.00923333 0.01
  0.01063333 0.01096667 0.01036667 0.0099     0.0096     0.00963333
  0.00926667 0.0096     0.00643333 0.00333333]
```

```
#5.Plot the normal pdf of signal and Parzen window density function of signal for various window size
#(V=1,2,3,5,10)

def Parzen_window(w,x):
    y=np.zeros(100);

    for i in range(100):
        for j in range(len(x)):
            for k in range(w):
                if(x[j]==i+k):
                    y[i]=y[i]+1;
        y[i]=(y[i]/len(x))/w;

    plt.plot(y,label=f"for v={w}")
    plt.legend()

Parzen_window(1,x)
Parzen_window(2,x)
Parzen_window(3,x)
Parzen_window(5,x)
Parzen_window(10,x)
```




```
#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).

x=np.random.randint(100,201,size=20000)

def Parzen_window(w,x):
    y=np.zeros(100);
```

```
    for i in range(100,201):
        for j in range(len(x)):
            for k in range(w):
                if(x[j]==i+k):
                    y[100-i]=y[100-i]+1;
        y[100-i]=(y[100-i]/len(x))/w;

    if(w==1):
      plt.plot(y,label=f"for normal pdf")
      plt.legend()
    else:
      plt.plot(y,label=f"for v={w}")
      plt.legend()

Parzen_window(1,x)
Parzen_window(5,x)
Parzen_window(10,x)
Parzen_window(20,x)
```



✓ 0s completed at 11:08 PM

Conclusion:

Thus I concluded that the Parzen window density estimation technique takes sample input data value and returns the density estimate of the given data sample.

Practical - 11

Aim: Write a python program for implementation of k-nearest neighbor algorithm.

Tool: Google colab

Theory:

k-nearest neighbor algorithm.:

- K-Nearest Neighbor 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.

- Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Codes and outputs:

```
#Priya Kishor Kshirsagar_B59_2019BEC134
#PRACTICAL 11
```

```
#Write a python program for implementation of k-nearest neighbor algorithm.
```

```
import numpy as np


def KNN_for_1D(w1,w2,xtest,k):
    count1=0
    count2=0
    d1=np.zeros(len(w1))
    d2=np.zeros(len(w2))

    for i in range(len(w1)):
        d1[i]=abs(xtest-w1[i])
        d2[i]=abs(xtest-w2[i])

    D=[]
    D=np.append(d1,d2)
    D=sorted(D)

    for i in range(k):
        for j in range(len(d1)):
            if(D[i]==d1[j]):
                count1+=1

            if(D[i]==d2[j]):
                count2+=1

    if(count1<count2):
        print(f"Assigning class: w2 for the xtest={xtest} & k={k}")

    else:
        print(f"Assigning class: w1 for the xtest={xtest} & k={k}")



def KNN_for_2D(w1,w2,xtest,k):
    count1=0
    count2=0
    d1=np.zeros(len(w1))
    d2=np.zeros(len(w2))

    for i in range(len(w1)):
        d1[i]=abs(np.sqrt( ((xtest[0]-w1[i][0])**2) + ((xtest[1]-w1[i][1])**2) ))
        d2[i]=abs(np.sqrt( ((xtest[0]-w2[i][0])**2) + ((xtest[1]-w2[i][1])**2) ))

    D=[]
    D=np.append(d1,d2)
    D=sorted(D)
```

```
    for i in range(k):
        for j in range(len(d1)):
            if(D[i]==d1[j]):
                count1+=1

            if(D[i]==d2[j]):
                count2+=1

    if(count1<count2):
        print(f"Assigning class: w2 for the xtest={xtest} & k={k}")

    else:
        print(f"Assigning class: w1 for the xtest={xtest} & k={k}")
```

```
#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 & 4.1

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]

print("Xtest=3.3")
KNN_for_1D(w1,w2,3.3,3)
KNN_for_1D(w1,w2,3.3,5)
KNN_for_1D(w1,w2,3.3,7)

print("\nXtest=1.9")
KNN_for_1D(w1,w2,1.9,3)
KNN_for_1D(w1,w2,1.9,5)
KNN_for_1D(w1,w2,1.9,7)

print("\nXtest=8.6")
KNN_for_1D(w1,w2,8.6,3)
KNN_for_1D(w1,w2,8.6,5)
KNN_for_1D(w1,w2,8.6,7)

print("\nXtest=4.1")
KNN_for_1D(w1,w2,4.1,3)
KNN_for_1D(w1,w2,4.1,5)
KNN_for_1D(w1,w2,4.1,7)
```

```
Xtest=3.3
Assigning class: w1 for the xtest=3.3 & k=3
Assigning class: w1 for the xtest=3.3 & k=5
Assigning class: w1 for the xtest=3.3 & k=7

Xtest=1.9
Assigning class: w1 for the xtest=1.9 & k=3
Assigning class: w1 for the xtest=1.9 & k=5
Assigning class: w1 for the xtest=1.9 & k=7

Xtest=8.6
```

```
Assigning class: w2 for the xtest=8.6 & k=3
Assigning class: w2 for the xtest=8.6 & k=5
Assigning class: w2 for the xtest=8.6 & k=7

Xtest=4.1
Assigning class: w2 for the xtest=4.1 & k=3
Assigning class: w1 for the xtest=4.1 & k=5
Assigning class: w1 for the xtest=4.1 & k=7
```

```
#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

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]]

xtest1=[1.7,3.2];
xtest2=[1.2,4.5];
xtest3=[2.6,3.8]
xtest4=[0.8, 2.5]

print("Xtest=[1.7,3.2]")
KNN_for_2D(w1,w2,xtest1,3)
KNN_for_2D(w1,w2,xtest1,5)
KNN_for_2D(w1,w2,xtest1,7)

print("\nXtest=[1.2,4.5]")
KNN_for_2D(w1,w2,xtest2,3)
KNN_for_2D(w1,w2,xtest2,5)
KNN_for_2D(w1,w2,xtest2,7)

print("\nXtest=[2.6,3.8]")
KNN_for_2D(w1,w2,xtest3,3)
KNN_for_2D(w1,w2,xtest3,5)
KNN_for_2D(w1,w2,xtest3,7)

print("\nXtest=[0.8, 2.5]")
KNN_for_2D(w1,w2,xtest4,3)
KNN_for_2D(w1,w2,xtest4,5)
KNN_for_2D(w1,w2,xtest4,7)
```

```
Xtest=[1.7,3.2]
Assigning class: w1 for the xtest=[1.7, 3.2] & k=3
Assigning class: w1 for the xtest=[1.7, 3.2] & k=5
Assigning class: w1 for the xtest=[1.7, 3.2] & k=7

Xtest=[1.2,4.5]
Assigning class: w1 for the xtest=[1.2, 4.5] & k=3
Assigning class: w1 for the xtest=[1.2, 4.5] & k=5
Assigning class: w1 for the xtest=[1.2, 4.5] & k=7

Xtest=[2.6,3.8]
Assigning class: w1 for the xtest=[2.6, 3.8] & k=3
Assigning class: w1 for the xtest=[2.6, 3.8] & k=5
Assigning class: w1 for the xtest=[2.6, 3.8] & k=7
```

```
Xtest=[0.8, 2.5]
Assigning class: w1 for the xtest=[0.8, 2.5] & k=3
Assigning class: w1 for the xtest=[0.8, 2.5] & k=5
Assigning class: w1 for the xtest=[0.8, 2.5] & k=7
```

```
#3.Generate the random 1D data consisting of 100 samples in any desired range for two clas
#Vary the ‘k’ value [3,5,7,9] and find the class for 5 random test sample.

w1=8*np.random.rand(100);
w2=12*np.random.rand(100)

xtest1=1.56
xtest2=4.20
xtest3=0.156
xtest4=2.145
xtest5=3.234

print("xtest=1.56")
KNN_for_1D(w1,w2,xtest1,3)
KNN_for_1D(w1,w2,xtest1,5)
KNN_for_1D(w1,w2,xtest1,7)
KNN_for_1D(w1,w2,xtest1,9)

print("\nxtest=4.20")
KNN_for_1D(w1,w2,xtest2,3)
KNN_for_1D(w1,w2,xtest2,5)
KNN_for_1D(w1,w2,xtest2,7)
KNN_for_1D(w1,w2,xtest2,9)

print("\nxtest=0.156")
KNN_for_1D(w1,w2,xtest3,3)
KNN_for_1D(w1,w2,xtest3,5)
KNN_for_1D(w1,w2,xtest3,7)
KNN_for_1D(w1,w2,xtest3,9)

print("\nxtest=2.145")
KNN_for_1D(w1,w2,xtest4,3)
KNN_for_1D(w1,w2,xtest4,5)
KNN_for_1D(w1,w2,xtest4,7)
KNN_for_1D(w1,w2,xtest4,9)

print("\nxtest=3.234")
KNN_for_1D(w1,w2,xtest5,3)
KNN_for_1D(w1,w2,xtest5,5)
KNN_for_1D(w1,w2,xtest5,7)
KNN_for_1D(w1,w2,xtest5,9)
```

```
xtest=1.56
Assigning class: w2 for the xtest=1.56 & k=3
Assigning class: w1 for the xtest=1.56 & k=5
Assigning class: w1 for the xtest=1.56 & k=7
Assigning class: w1 for the xtest=1.56 & k=9

xtest=4.20
Assigning class: w1 for the xtest=4.2 & k=3
```

```
Assigning class: w1 for the xtest=4.2 & k=5
Assigning class: w2 for the xtest=4.2 & k=7
Assigning class: w2 for the xtest=4.2 & k=9

xtest=0.156
Assigning class: w1 for the xtest=0.156 & k=3
Assigning class: w1 for the xtest=0.156 & k=5
Assigning class: w1 for the xtest=0.156 & k=7
Assigning class: w1 for the xtest=0.156 & k=9

xtest=2.145
Assigning class: w2 for the xtest=2.145 & k=3
Assigning class: w2 for the xtest=2.145 & k=5
Assigning class: w2 for the xtest=2.145 & k=7
Assigning class: w2 for the xtest=2.145 & k=9

xtest=3.234
Assigning class: w1 for the xtest=3.234 & k=3
Assigning class: w2 for the xtest=3.234 & k=5
Assigning class: w2 for the xtest=3.234 & k=7
Assigning class: w1 for the xtest=3.234 & k=9
```

Conclusion:

1) Thus I concluded that the K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
2) K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

Practical - 12

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

Tool: Google colab

Theory:

K-Means Clustering Algorithm:

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what K-means clustering algorithm is, how the algorithm works, along with the Python implementation of k-means clustering.

What is the K-Means Clustering Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of predefined 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.

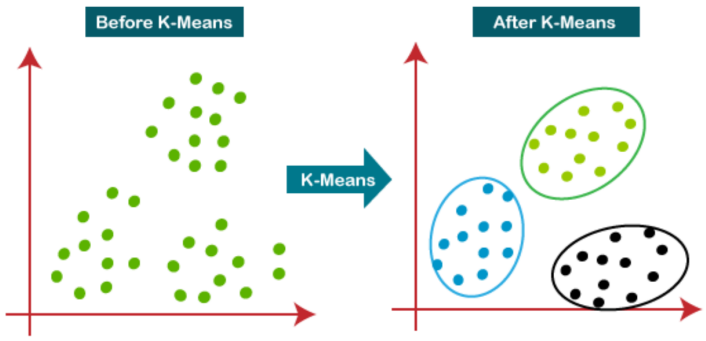
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

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 data points with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Codes and outputs:

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

def U_Distance(xc1,xc2,x):
    d1=np.zeros(len(x))
    d2=np.zeros(len(x))

    for i in range(len(x)):
        d1[i]=abs(np.sqrt( ((xc1[0]-x[i][0])**2) + ((xc1[1]-x[i][1])**2) ))
        d2[i]=abs(np.sqrt( ((xc2[0]-x[i][0])**2) + ((xc2[1]-x[i][1])**2) ))
    return d1,d2;

def K_Clustering(x,u):

    xc1=np.array([0,0])
    xc2=np.array([0,0])

    while(1):
        u_new=np.zeros((len(u),len(u[0])))
        x1=0
        x2=0
        count1=0
        count2=0
        for i in range(len(u[0])):
            if(u[0][i]==1):
                count1+=1;
                x1+=x[i]

            else:
                count2+=1;
                x2+=x[i]

        xc1=x1/count1
        xc2=x2/count2


        d1,d2=U_Distance(xc1,xc2,x);

        for i in range(len(d1)):
            if(d1[i]<d2[i]):
                u_new[0][i]=1
                u_new[1][i]=0

            else:
                u_new[0][i]=0
                u_new[1][i]=1

        comparison = u == u_new
        equal_arrays = comparison.all()
        if(equal_arrays):
            break;

        u=u_new
```

```
    print("u=")
    print(u_new)
```

```
x=np.array([[3, 1],[4, 2],[1, 1],[6, 3],[2, 6],[-1, 2]]);
u=np.array([[0, 0, 1, 1, 0, 0], [1, 1, 0, 0, 1, 1]])

K_Clustering(x,u)
```

```
u=
[[1. 1. 0. 1. 0. 0.]
 [0. 0. 1. 0. 1. 1.]]
```

```
x=np.array([[-1, 1],[-2, -3],[1, 3],[-2, 1],[5, 1],[4, -2],[3, -1],[3, 3]]);
u=np.array([[1, 0, 0, 1, 0, 0, 1, 1], [0, 1, 1, 0, 1, 1, 0, 0]])

K_Clustering(x,u)
```

```
u=
[[1. 0. 1. 1. 0. 0. 0. 1.]
 [0. 1. 0. 0. 1. 1. 1. 0.]]
```

```
x=np.array([[-1, -1],[1, 3],[4, 1],[3, -2],[-1, 2],[-2, 1],[5, -3],[1, -1]]);
u=np.array([[1, 1, 1, 0, 0, 0, 0, 1], [0, 0, 0, 1, 1, 1, 1, 0]])

K_Clustering(x,u)
```

```
u=
[[1. 1. 0. 0. 1. 1. 0. 0.]
 [0. 0. 1. 1. 0. 0. 1. 1.]]
```

+ Code + Text

```
x=np.array([[1, 1],[-1, 4],[-2, 1],[3, 4],[-3, -2],[5, 4],[2, -2],[-4, -1]]);
u=np.array([[1, 0, 1, 0, 0, 1, 0, 1], [0, 1, 0, 1, 1, 0, 1, 0]])

K_Clustering(x,u)
```

```
u=
[[0. 1. 1. 0. 1. 0. 0. 1.]
 [1. 0. 0. 1. 0. 1. 1. 0.]]
```

Conclusion:

1) K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters.
2) K-means clustering algorithm can be used to solve the clustering problems in machine learning or data science.