Pattern Recognition Lab Assignment

Submitted by: Yukti Khurana, 2017UCP1234
Submitted to: Dr Deepak Ranjan Nayak
Semester-8 2021

DAY-1

1. Write MATLAB/Python program to generate the following distributions.

- Normal distribution

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

- Uniform distribution

$$p(x) = \frac{1}{b-a} \text{ if } a < x < b;\ 0 \text{ otherwise} \tag{2}$$

- Exponential distribution

$$p(x) = \lambda e^{-\lambda x} \text{ if } x \geq 0;\ 0 \text{ otherwise} \tag{3}$$

- Poisson distribution

$$p(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}; \lambda > 0 \tag{4}$$

1. NORMAL DISTRIBUTION

CODE

```
# NORMAL DISTRIBUTION

import random
import math
import matplotlib.pyplot as plt

def data_mean(samples):
    return sum(samples)/(1.0*len(samples))

def data_varience(samples, data_mean):
    summ = 0
    for x in samples:
        t1 = abs(x-data_mean)
        t2 = pow(t1,2)
        summ+=t2
    return summ*1.0/len(samples)

# function to apply normal distribution
def Normal_Distribution(x, mean,var):
    t1 = 2 * math.pi * var
    t2 = 1.0 / pow(t1, 0.5)
    t3 = ((-1.0) * pow(x - mean, 2)) / 2 * var
    t4 = pow(math.e, t3)
    return (t2*t4)
```

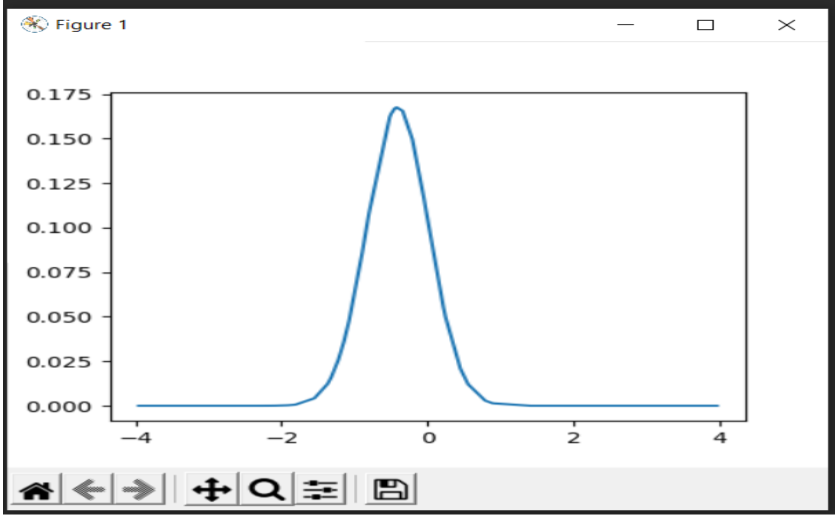
```
# function to create n samples points
def generate_random_samples(start, end, n):
    if (start>=end):
        return
    samples = []
    for i in range(n):
        samples.append(random.uniform(start,end))
    return samples

def main():
    start = -4
    end = 4
    n = 100
    samples = generate_random_samples(start,end,n)
    samples.sort()
    mean = data_mean(samples)
    var = data_varience(samples,mean)
    print("Mean = ",mean)
    print("Varience = ",var)
    distr = []
    for i in samples:
        px = Normal_Distribution(i,mean,var)
        distr.append(px)
    plt.plot(samples,distr)
    plt.show()

main()
```

INPUT

- For range of numbers:
    a. start = -4
    b. end = 4
- Number of Samples
  n = 100

OUTPUT

```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Use
Mean =  -0.40932932242863423
Varience =  5.659856976268552
```



2. UNIFORM DISTRIBUTION

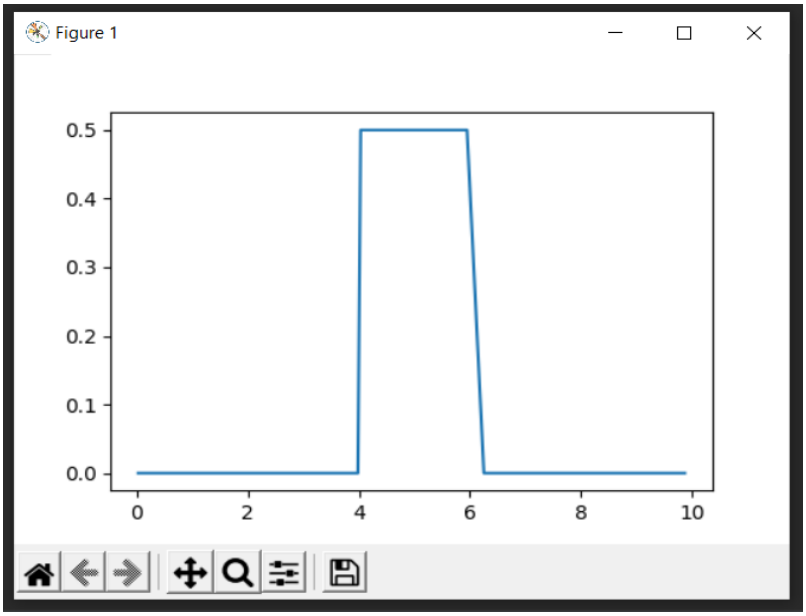
CODE

```python
# UNIFORM DISTRIBUTION
import random
import matplotlib.pyplot as plt

# function to apply uniform distribution
def Uniform_Distribution(x, a, b):
    if (a<x<b):
        return 1.0/(b-a)
    return 0

# function to create n samples points
def generate_random_samples(start, end, n):
    if (start >= end):
        return
    samples = []
    for i in range(n):
        samples.append(random.uniform(start, end))
    return samples


def main():
    start = 0
    end = 10
    n = 100
    samples = generate_random_samples(start, end, n)
    samples.sort()
    a = 4
    b = 6
    distr = []
    for i in samples:
        px = Uniform_Distribution(i, a, b)
        distr.append(px)
    plt.plot(samples, distr)
    plt.show()

main()
```

INPUT

- For range of numbers:
    a. start = 0
    b. end = 10
- Number of Samples
  n = 100
- a=4
- b=6

OUTPUT



3. EXPONENTIAL DISTRIBUTION

CODE

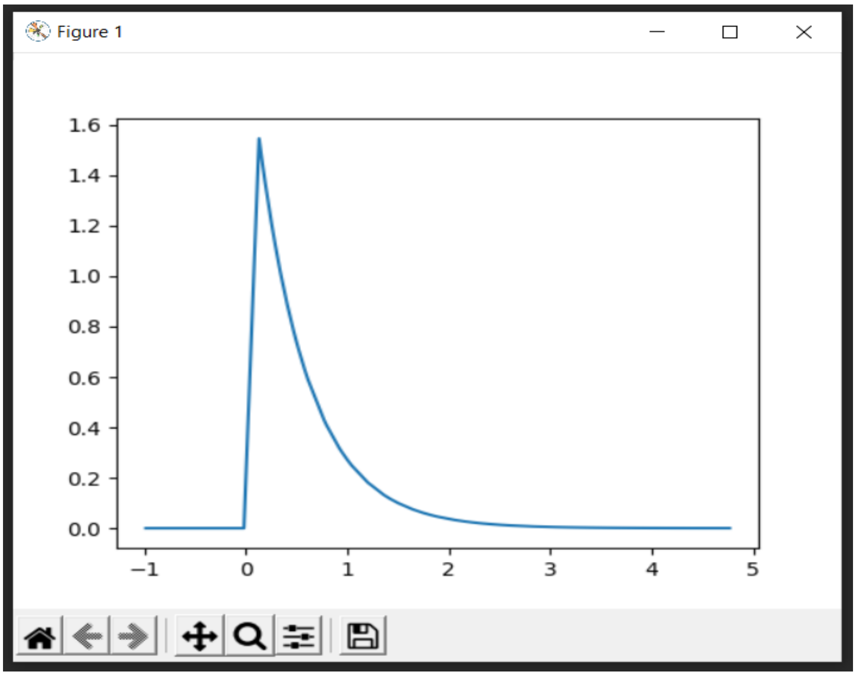
```
# Exponential DISTRIBUTION

import random
import math
import matplotlib.pyplot as plt

# function to apply exponential distribution
def Exponential_Distribution(x, lambd):
    if (x>=0):
        t1 = (-1.0)*lambd*x
        t2 = pow(math.e, t1)
        return lambd*t2
    return 0


# function to create n samples points
def generate_random_samples(start, end, n):
    if (start >= end):
        return
    samples = []
    for i in range(n):
        samples.append(random.uniform(start, end))
    return samples


def main():
    start = -1
    end = 5
    n = 100
    samples = generate_random_samples(start, end, n)
    samples.sort()
```

```
    lambd = 2
    distr = []
    for i in samples:
        px = Exponential_Distribution(i, lambd)
        distr.append(px)
    plt.plot(samples, distr)
    plt.show()


main()
```

INPUT

- For range of numbers:
    a. start = -1
    b. end = 5
- Number of Samples
  n = 100

OUTPUT



4. POISSON DISTRIBUTION

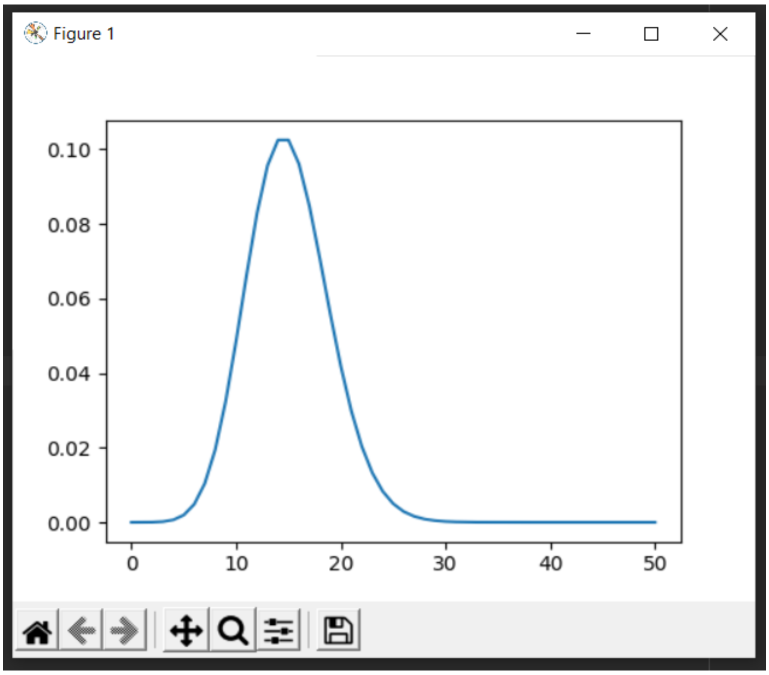
CODE

```
# POISSON DISTRIBUTION

import random
import math
import matplotlib.pyplot as plt

# function to apply poisson distribution
def Poisson_Distribution(x, lambd):
    t1 = pow(lambd,x)
    t2 = pow(math.e,-1.0*lambd)
    t3 = math.factorial(x)
    return (1.0*t1*t2)/t3

# function to create n samples points
def generate_random_samples(start, end, n):
    if (start >= end):
        return
    samples = []
    for i in range(n):
        samples.append(random.randint(start, end))
    return samples


def main():
    start = 0
    end = 50
    n = 300
    samples = generate_random_samples(start, end, n)
    samples.sort()
    lambd = 15
    distr = []
    for i in samples:
        px = Poisson_Distribution(i, lambd)
        distr.append(px)
    plt.plot(samples, distr)
    plt.show()


main()
```

INPUT

- For range of numbers:
    a. start = 0
    b. end = 50
- Lambda = 15
- Number of Samples
  n = 300

OUTPUT



2. Generate $N = 500$ 2-D random data points and plot its corresponding Gaussian PDF. $\left(p(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}\right)$

CODE

```
"""
Generate N = 500 2-D random data points and plot its corresponding Gaussian
PDF.
"""
import matplotlib.pyplot as plt
import random
import numpy as np
import math


def calculateMean(samples):
    s = np.array(samples)
    m = np.mean(s, axis=0)
    return np.array(m)

def Covarience(samples, mean, N, d):
    cov = np.zeros((d,d))
    for i in range(N):
        # for each sample
        x = samples[i]
        # difference with the mean of data
        t1 = x-mean
        # to multiply transpose of sample point with sample point
        q1 = t1.reshape(-1,1)
        q2 = t1.reshape(1,-1)
        q = np.matmul(q1,q2)
        # add to covarience matrix
        cov += q
    # divide by number of samples minus one
    c = 1.0/(N-1)
    cov = np.multiply(cov,c)
    return cov

def GaussianPx(x, mean, Cov, d):
    # determinant of Covariance matrix
    cov_det = np.linalg.det(Cov)
    cov_inv = np.linalg.inv(Cov)

    #print("inv = ",cov_inv)

    # difference of x (data point) and mean of distribution, and its
transpose
    x1 = np.array(x - mean).reshape(1,-1)
    x2 = np.array(x - mean).reshape(-1, 1)

    #print("x1 = ",x1)
    t1 = 1.0/((2*math.pi)**(d/2.0))
    t2 = 1.0/(cov_det**0.5)
    t3 = np.matmul(x1,cov_inv)
    t4 = np.matmul(t3,x2)
    px = t1 * t2 * pow(math.e,-0.5*t4)
```

```
    return px

# function to create n samples points
def d_dim_samples(start, end, N, d):
    if (start>=end):
        return
    # list of d-dimensional samples
    samples = []
    for i in range(N):
        t = []
        for j in range(d):
            t.append(random.uniform(start,end))
        samples.append(np.array(t))
    return samples


def main():
    start = -10
    end = 10
    # variables that can be changed according to the user
    N = 500
    # d = no of dimensions or features of x
    d = 2
    samples = d_dim_samples(start, end, N, d)
    print(samples)
    print("\nNumber of Data Points = ", N)
    print("Number of Dimensions/features = ",d)
    print("\nData points : ")
    cnt=0
    for i in samples:
        cnt+=1
        print(cnt," ",i)
    print()
    print("\nMean: ")
    mean = calculateMean(samples)
    print(mean)

    print("\nCovariance: ")
    Cov = Covarience(samples, mean, N, d)
    print(Cov)
    print()
    distr = []
    for i in samples:
        px = GaussianPx(i, mean, Cov, d)
        distr.append(px[0][0])

    # 3D Graph to visualize the Gaussian Distribution
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    # Data for a three-dimensional line
    zvals = np.array(distr)
    xvals = []
    yvals = []
    for i in samples:
        xvals.append(i[0])
        yvals.append(i[1])
    xvals = np.array(xvals)
    yvals = np.array(yvals)
    ax.scatter3D(xvals, yvals, zvals, c=zvals, cmap='Oranges')
    plt.title("Gaussian PDF by Yukti Khurana")
    plt.xlabel('Feature-1')
```

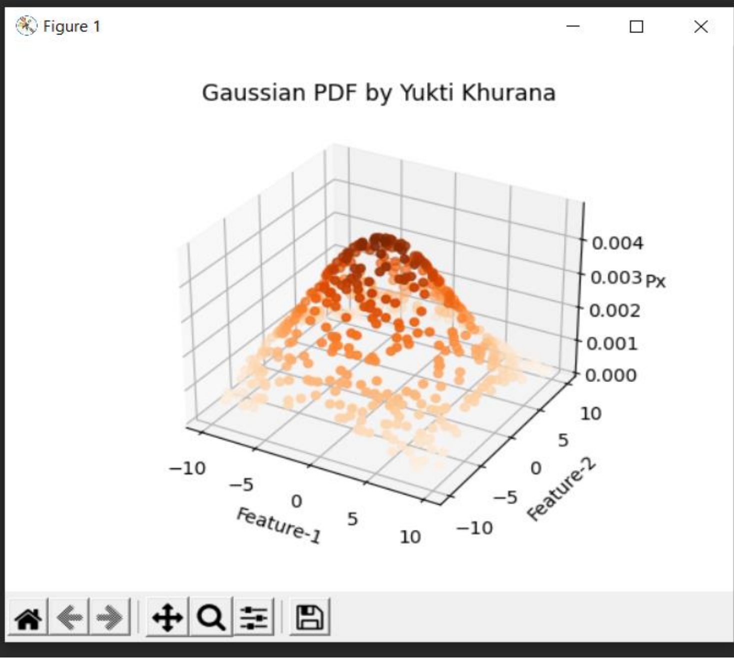
```
    plt.ylabel('Feature-2')
    ax.set_zlabel('Px')
    plt.show()

main()
```

INPUT

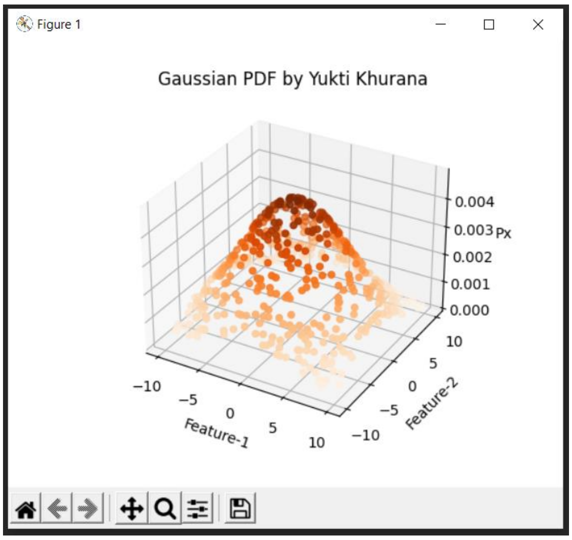
- For range of numbers:
    a. start = -10
    b. end = 10
- Number of features/ dimensions (d) = 2
- Number of Samples
  n = 500

OUTPUT

```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/GaussianPDF.py"
[array([ 6.1213267 , -9.26436834]), array([-7.38262466, -3.64102787]), array([-4.08934019,  8.20071895]), array([ 0.6146448 ,

Number of Data Points =  500
Number of Dimensions/features =  2

Data points :
1    [ 6.1213267   -9.26436834]
2    [-7.38262466 -3.64102787]
3    [-4.08934019  8.20071895]
4    [ 0.6146448   -6.14062887]
5    [-6.7820863   -4.20780706]
6    [-8.12037719  8.18378685]
7    [ 4.9719346  -7.7956138]
8    [9.6245299 9.9945177]
9    [-8.27722868 -5.85864622]
10    [-4.01639114  0.92350845]
11    [4.96841217 4.58509284]
12    [-4.19477519  0.88014418]
13    [ 7.89382619 -8.28532725]
14    [6.80397391 0.8655564 ]
15    [-5.6356563   -9.19561117]
16    [ 9.38425164 -2.36889062]
17    [-5.63538595 -2.63915758]
18    [-4.02731295 -2.11021848]
19    [ 0.3123114   -0.47259944]
20    [-9.27766637 -6.63197471]
```




```
484    [6.33428483 3.18618806]
485    [3.0523313   9.57046583]
486    [-9.26658077  2.44590705]
487    [-0.53532924 -6.57239221]
488    [7.58327565 4.40792036]
489    [-6.27466991 -7.98710303]
490    [5.00348713 3.08010376]
491    [ 9.89900335 -8.86421538]
492    [-0.12845716  3.8349374 ]
493    [-8.8491827   -4.00756721]
494    [-8.38186708  1.87903604]
495    [ 7.6531147   -8.16782386]
496    [-3.22711088  1.04988706]
497    [-1.957499    -9.46662502]
498    [9.93601771 8.5956194 ]
499    [-8.2778972   -2.72630725]
500    [6.96904327 4.15864931]


Mean:
[ 0.03205416 -0.62021893]

Covariance:
[[34.70285001 -0.09550851]
 [-0.09550851 32.57890477]]
```



3. Given a set of d-dimensional samples. Write a program to find out covariance matrix.
$$\left(\Sigma = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)(x_i - \mu)^T\right)$$

CODE

```
"""
Given a set of d-dimensional samples. Write a program to find out
covariance matrix.
"""

# Calculating Covariance
import random
import numpy as np

def calculateMean(samples):
    s = np.array(samples)
    m = np.mean(s, axis=0)
    return np.array(m)

def Covarience(samples, mean, N, d):
    cov = np.zeros((d,d))
    for i in range(N):
        # for each sample
        x = samples[i]
        # difference with the mean of data
        t1 = x-mean
        # to multiply transpose of sample point with sample point
        q1 = t1.reshape(-1,1)
        q2 = t1.reshape(1,-1)
        q = np.matmul(q1,q2)
        # add to covarience matrix
        cov += q
    # divide by number of samples minus one
    c = 1.0/(N-1)
    cov = np.multiply(cov,c)
    return cov

# function to create n samples points
def d_dim_samples(start, end, N, d):
    if (start>=end):
        return
    # list of d-dimensional samples
    samples = []
    for i in range(N):
        t = []
        for j in range(d):
            t.append(random.uniform(start,end))
        samples.append(np.array(t))
    return samples


def main():
    # variables that can be changed according to the user
    N = 10
    # d = no of dimensions or features of x
    d = 3
    start = -10
```

```
    end = 10
    samples = d_dim_samples(start, end, N, d)
    print("\nNumber of Data Points = ", N)
    print("Number of Dimensions/features = ",d)
    print("\nData points : ")
    cnt=0
    for i in samples:
        cnt+=1
        print(cnt," ",i)
    print()
    mean = calculateMean(samples)
    print("Mean:\n ",mean)
    print()
    Cov = Covarience(samples, mean, N, d)
    print("Covariance:\n ",Cov)
    print()


main()
```

INPUT (mentioned values can be changed to be anything)

- For range of numbers:
    a. start = -10
    b. end = 10
- Number of features/ dimensions (d) = 3
- Number of Samples
  N = 10

OUTPUT

```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/Covariance.py"

Number of Data Points =  10
Number of Dimensions/features =  3

Data points :
1   [ 7.8460332  -8.7950896  -7.76525199]
2   [ 0.23864387 -7.26476336 -0.15187597]
3   [ 4.77000191 -8.57305474  9.2230809 ]
4   [ 1.72978931 -9.73293302  8.63126296]
5   [-4.73087527  1.20437514  5.36023908]
6   [2.46669231 5.81022325 7.30168243]
7   [-7.59682209 -6.0537791  -3.81200186]
8   [ 2.61673506 -2.55030379 -5.4616757 ]
9   [ 1.86054685  1.52794841 -3.63871206]
10   [ 8.86424147 -7.40832317  5.57157628]

Mean:
  [ 1.80649866 -4.18357     1.52583241]

Covariance:
  [[25.53211938 -7.52995324  2.259455  ]
 [-7.52995324 28.76827159  1.02294514]
 [ 2.259455    1.02294514 40.81057306]]


Process finished with exit code 0
```

DAY-2

1. Generate $N = 500$ 2-D data points that are distributed according to the Gaussian distribution $N(m, \Sigma)$, with mean $m = [0, 0]^T$ and covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{bmatrix}$$

for the following cases

- $\sigma_1^2 = \sigma_2^2 = 1,\ \sigma_{12} = 0$
- $\sigma_1^2 = \sigma_2^2 = 0.2,\ \sigma_{12} = 0$
- $\sigma_1^2 = \sigma_2^2 = 2,\ \sigma_{12} = 0$
- $\sigma_1^2 = 0.2,\ \sigma_2^2 = 2,\ \sigma_{12} = 0$
- $\sigma_1^2 = 2,\ \sigma_2^2 = 0.2,\ \sigma_{12} = 0$
- $\sigma_1^2 = \sigma_2^2 = 1,\ \sigma_{12} = 0.5$
- $\sigma_1^2 = 0.3,\ \sigma_2^2 = 2,\ \sigma_{12} = 0.5$
- $\sigma_1^2 = 0.3,\ \sigma_2^2 = 2,\ \sigma_{12} = -0.5$

Plot each data set and comment the shape of the clusters formed by the data points.

CODE

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

# Mean of the Gaussian distribution
# by Yukti Khurana

Mean = [0, 0]
print("\nMean: ")
print(Mean)
print()

# Covariance matrices
Cov1 = [[1, 0], [0, 1]]
Cov2 = [[0.2, 0], [0, 0.2]]
Cov3 = [[2, 0], [0, 2]]
Cov4 = [[0.2, 0], [0, 2]]
Cov5 = [[2, 0], [0, 0.2]]
Cov6 = [[1, 0.5], [0.5, 1]]
Cov7 = [[0.3, 0.5], [0.5, 2]]
Cov8 = [[0.3, -0.5], [-0.5, 2]]

Cov_matrices = [Cov1, Cov2, Cov3, Cov4, Cov5, Cov6, Cov7, Cov8]

for i in range(8):
    print("Covariance Matrix-",i+1,":")
```

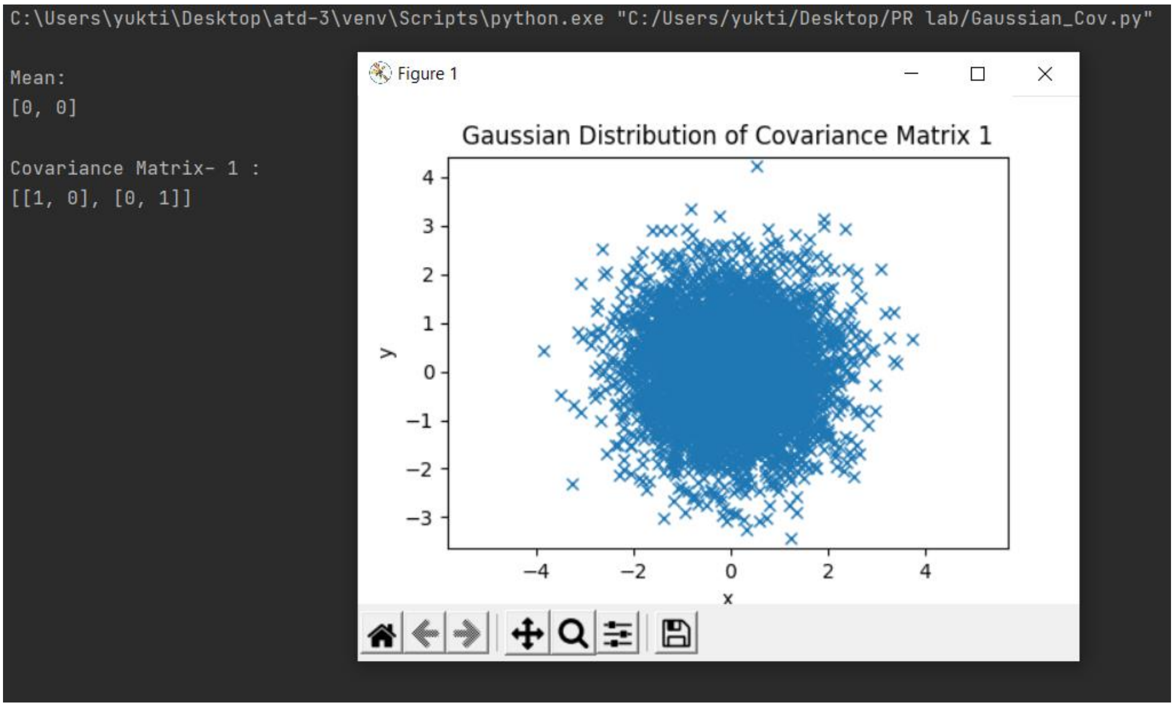
```
    print(Cov_matrices[i])
    #generating random numbers following the Gaussian distribution
    x, y = np.random.multivariate_normal(Mean, Cov_matrices[i], 5000).T
    plt.plot(x, y, 'x')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.axis('equal')
    s = "Gaussian Distribution of Covariance Matrix "+str(i+1)
    plt.title(s)
    plt.show()
    print()
```

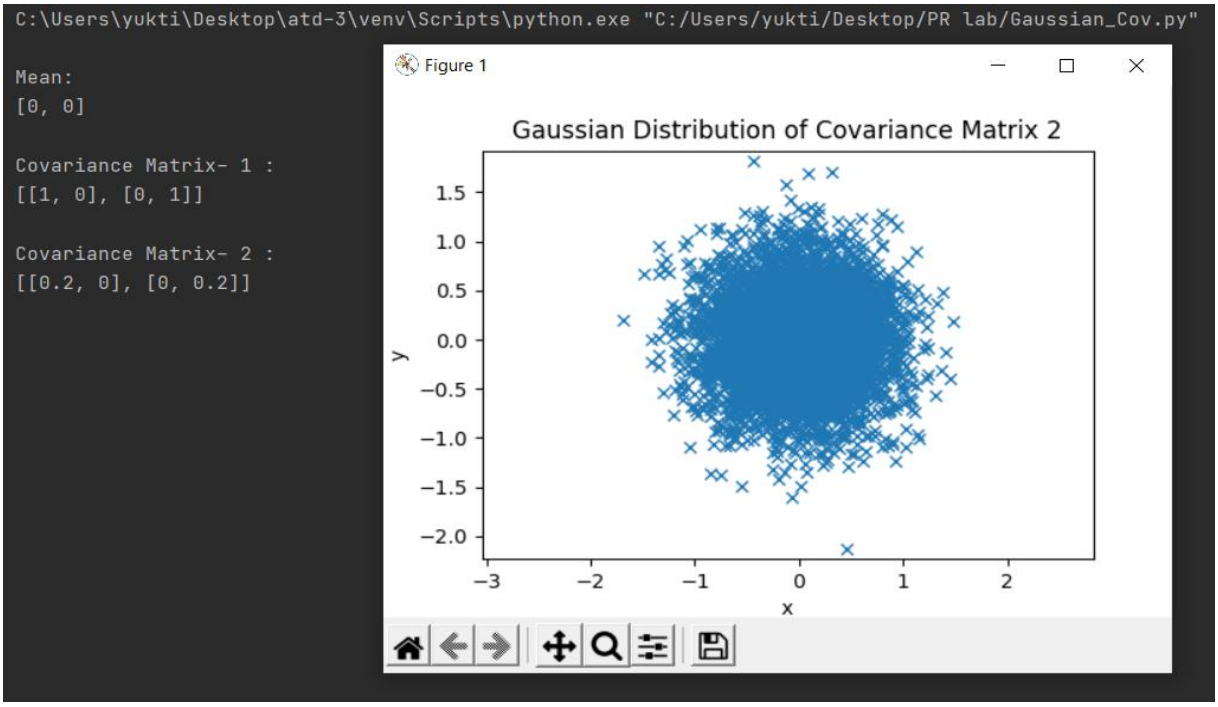
INPUT

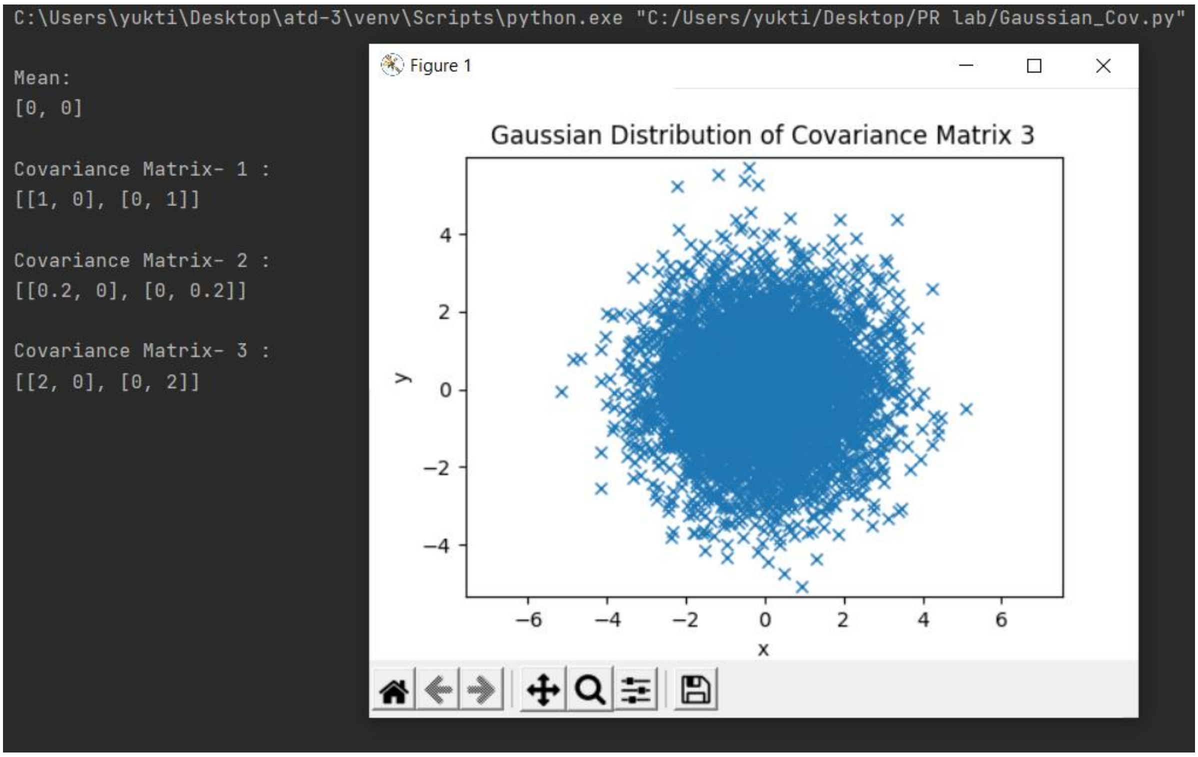
Cov1 = [[1, 0], [0, 1]]
Cov2 = [[0.2, 0], [0, 0.2]]
Cov3 = [[2, 0], [0, 2]]
Cov4 = [[0.2, 0], [0, 2]]
Cov5 = [[2, 0], [0, 0.2]]
Cov6 = [[1, 0.5], [0.5, 1]]
Cov7 = [[0.3, 0.5], [0.5, 2]]
Cov8 = [[0.3, -0.5], [-0.5, 2]]

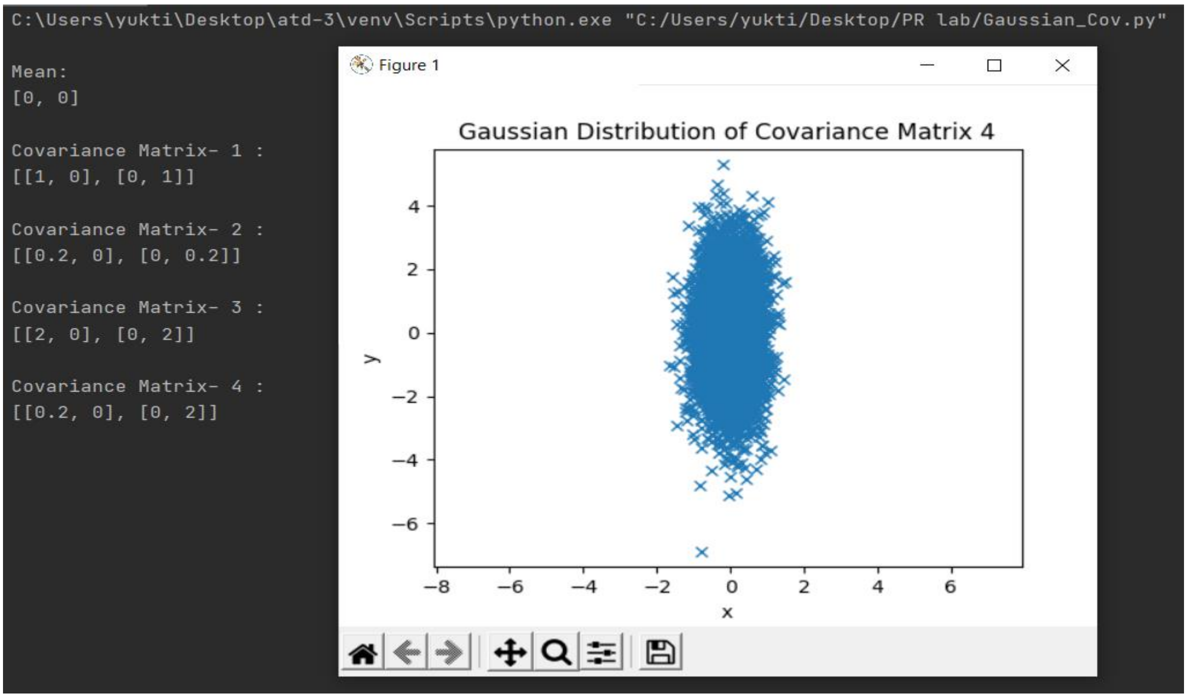
Any number and type of Covariance matrices can be entered in this code

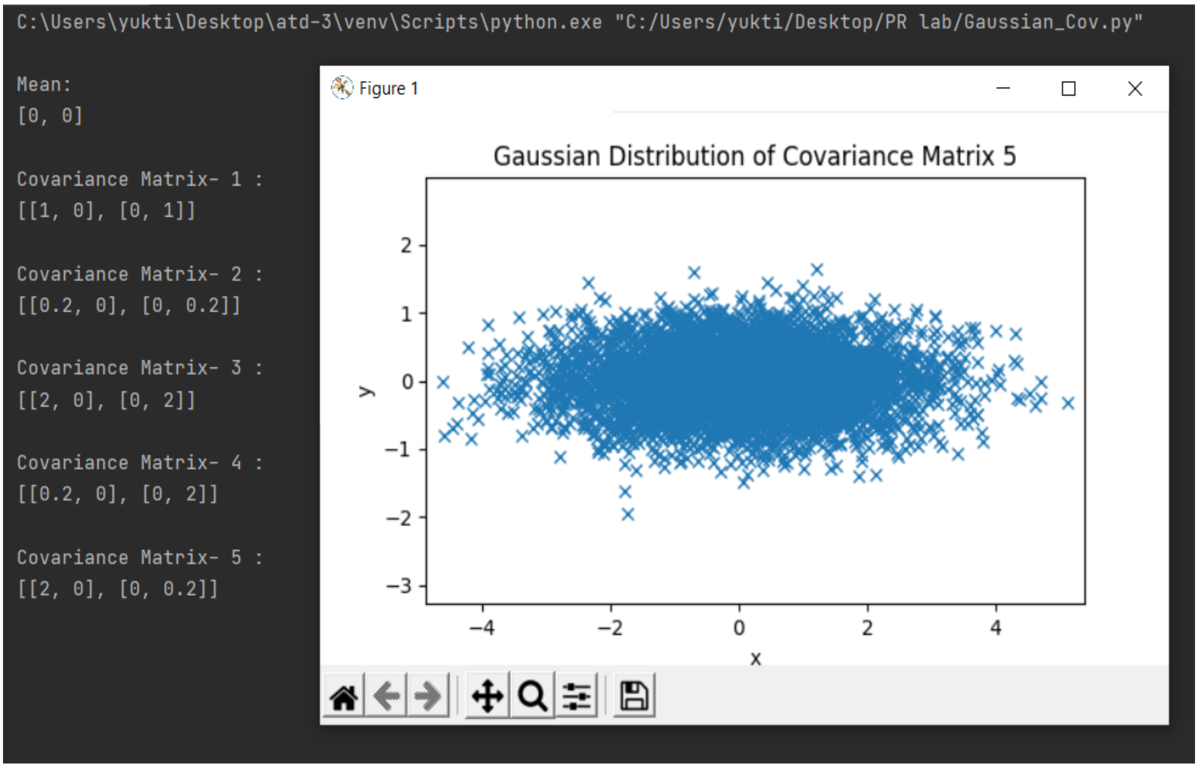
OUTPUT

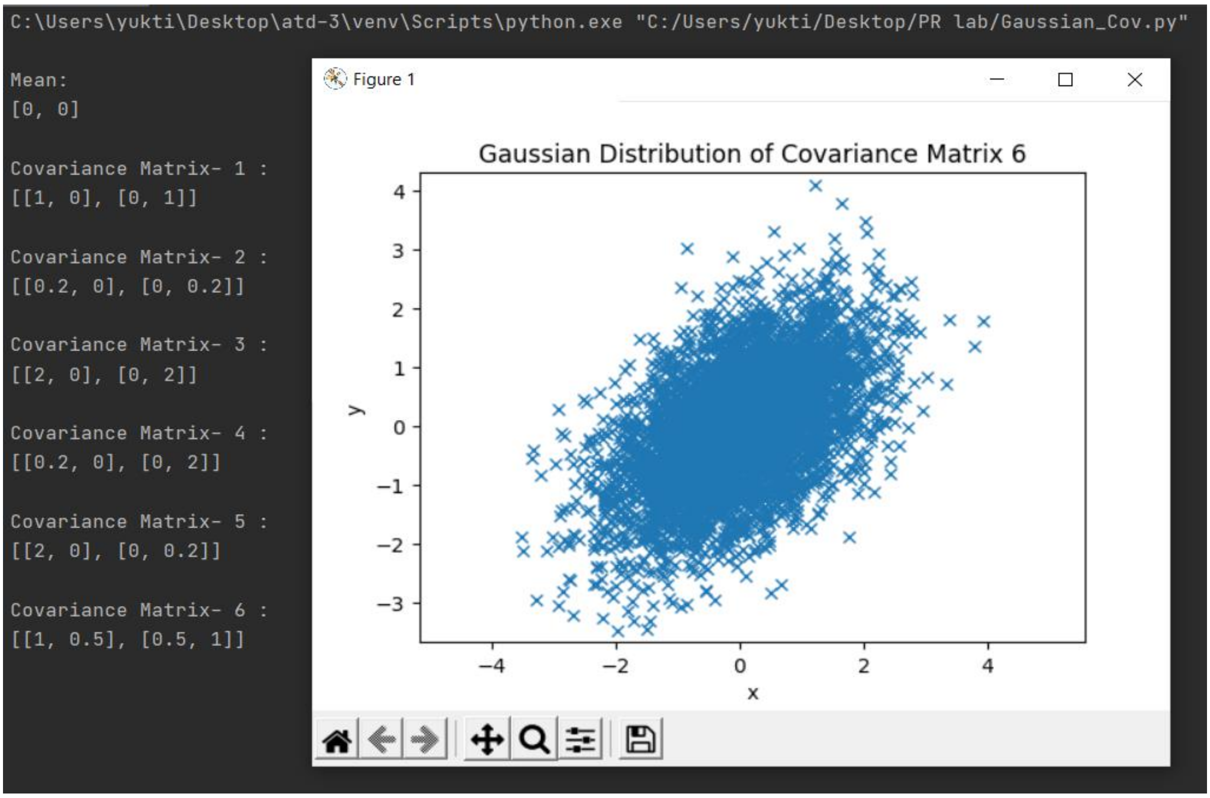


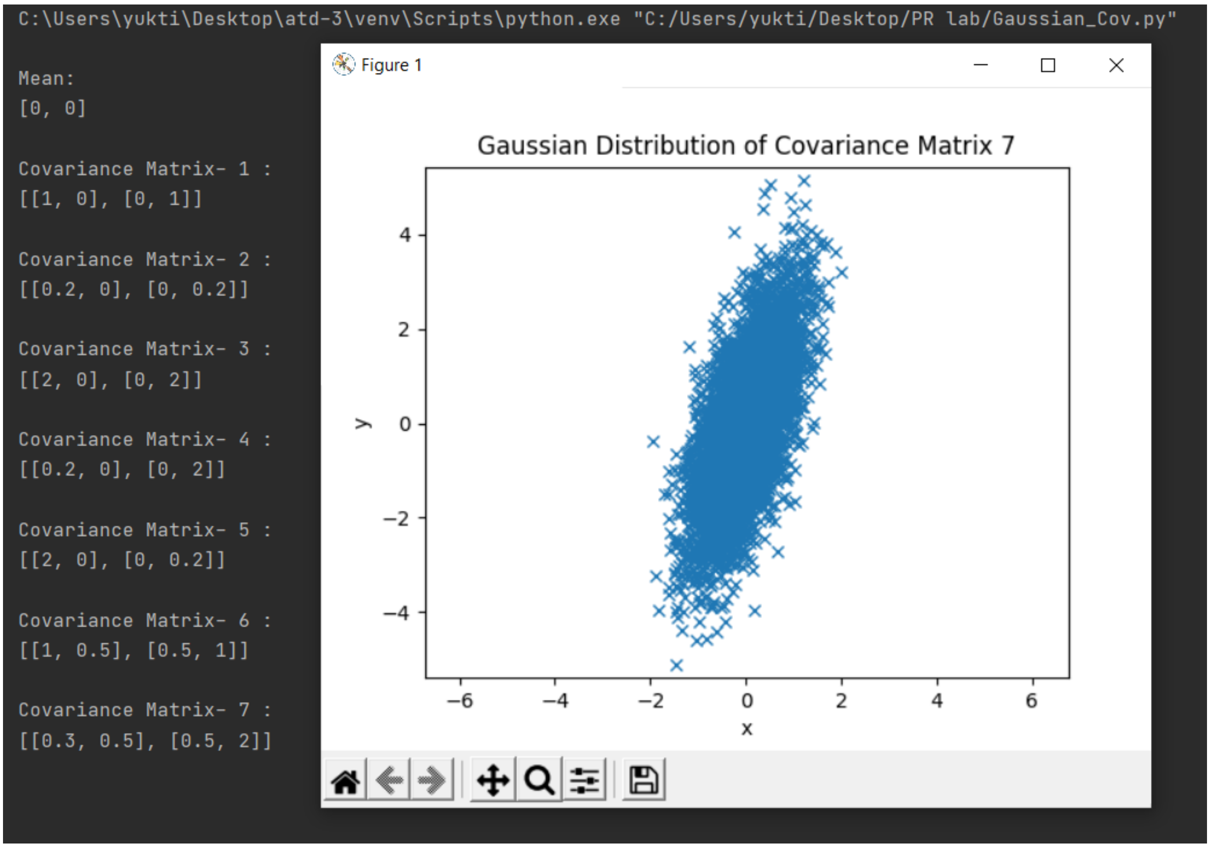

C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/Gaussian_Cov.py"
Mean:
[0, 0]
Covariance Matrix- 1 :
[[1, 0], [0, 1]]
Covariance Matrix- 2 :
[[0.2, 0], [0, 0.2]]
Figure 1
Gaussian Distribution of Covariance Matrix 2
y
x

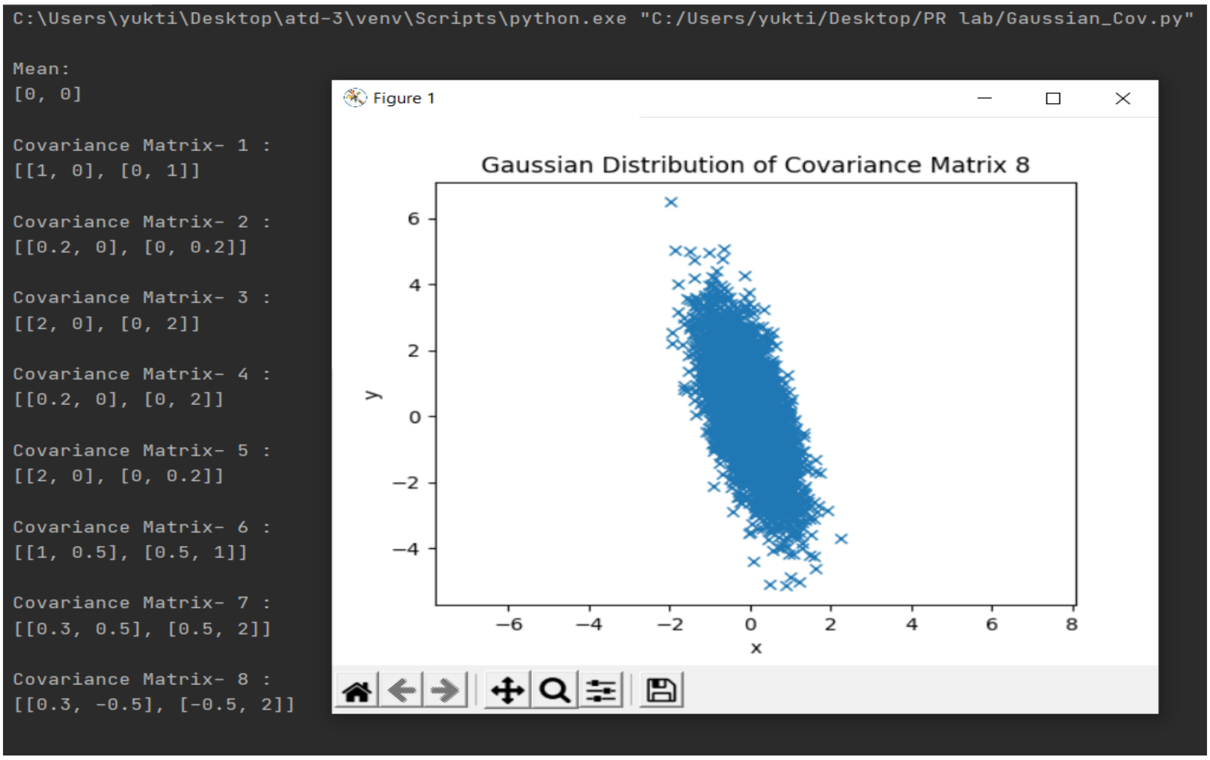

C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/Gaussian_Cov.py"
Mean:
[0, 0]
Covariance Matrix- 1 :
[[1, 0], [0, 1]]
Covariance Matrix- 2 :
[[0.2, 0], [0, 0.2]]
Covariance Matrix- 3 :
[[2, 0], [0, 2]]
Figure 1
Gaussian Distribution of Covariance Matrix 3
y
x


C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/Gaussian_Cov.py"
Mean:
[0, 0]
Covariance Matrix- 1 :
[[1, 0], [0, 1]]
Covariance Matrix- 2 :
[[0.2, 0], [0, 0.2]]
Covariance Matrix- 3 :
[[2, 0], [0, 2]]
Covariance Matrix- 4 :
[[0.2, 0], [0, 2]]
Figure 1
Gaussian Distribution of Covariance Matrix 4
y
x


C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/Gaussian_Cov.py"
Mean:
[0, 0]
Covariance Matrix- 1 :
[[1, 0], [0, 1]]
Covariance Matrix- 2 :
[[0.2, 0], [0, 0.2]]
Covariance Matrix- 3 :
[[2, 0], [0, 2]]
Covariance Matrix- 4 :
[[0.2, 0], [0, 2]]
Covariance Matrix- 5 :
[[2, 0], [0, 0.2]]
Figure 1
Gaussian Distribution of Covariance Matrix 5
y
x


C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/Gaussian_Cov.py"
Mean:
[0, 0]
Covariance Matrix- 1 :
[[1, 0], [0, 1]]
Covariance Matrix- 2 :
[[0.2, 0], [0, 0.2]]
Covariance Matrix- 3 :
[[2, 0], [0, 2]]
Covariance Matrix- 4 :
[[0.2, 0], [0, 2]]
Covariance Matrix- 5 :
[[2, 0], [0, 0.2]]
Covariance Matrix- 6 :
[[1, 0.5], [0.5, 1]]
Figure 1
Gaussian Distribution of Covariance Matrix 6
x
y


C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/Gaussian_Cov.py"
Mean:
[0, 0]
Covariance Matrix- 1 :
[[1, 0], [0, 1]]
Covariance Matrix- 2 :
[[0.2, 0], [0, 0.2]]
Covariance Matrix- 3 :
[[2, 0], [0, 2]]
Covariance Matrix- 4 :
[[0.2, 0], [0, 2]]
Covariance Matrix- 5 :
[[2, 0], [0, 0.2]]
Covariance Matrix- 6 :
[[1, 0.5], [0.5, 1]]
Covariance Matrix- 7 :
[[0.3, 0.5], [0.5, 2]]
Figure 1
Gaussian Distribution of Covariance Matrix 7
x
y



Conclusion

Based on the resulting plots of different covariance matrices shown above, we can deduce the following about covariance of the distribution and shape of the clusters formed –

1. When the two coordinated of x are uncorrelated i.e., **σ12 = 0** (which is the case in covariance matrices – 1, 2 and 3), the shape of the cluster formed by data vectors is "**Spherical**" in nature.
2. When the two coordinates of x are uncorrelated i.e., **σ12 = 0** and their **variances are unequal**, the data vectors form "ellipsoidal" shaped clusters. As we can observe in covariance matrices – 4 and 5.
    a. The coordinate with the highest variance corresponds to the "major axis" of the ellipsoidal-shaped cluster, while the coordinate with the lowest variance corresponds to its "minor axis." In addition, the major and minor axes of the cluster are parallel to the axes.
    b. Therefore, we can see that covariance-4 ellipsoidal cluster is parallel to y-axis as its highest variance is 2 which corresponds too y-axis.
    c. And the covariance-5 ellipsoidal cluster is parallel to x-axis, as its highest variance is 2, parallel to x-axis.
3. When the two coordinates of x are correlated (σ12! = 0), the major and minor axes of the ellipsoidal-shaped cluster are no longer parallel to the axes. This can be observed in covariance matrices – 6,7 and 8.
    a. The degree of rotation with respect to the axes depends on the value of σ12.
    b. The effect of the value of σ12, whether positive or negative is apparent from above plots. When σ12 is positive, the clusters tilt towards right, while towards left when it is negative as in covariance matrix-8. (σ12= -0.5). Also, the spherical nature of matrix-6 is because covariances are equal. In matrix-7 shape is ellipsoid again as variances are not equal.

2. Consider a c-class classification task in the d-dimensional space, where the data in all classes are distributed according to Gaussian distribution $N(m_i, S_i)$, $i = 1, 2, \ldots, c$.

   For a given $m_i = [m_1, m_2, \ldots, m_d]$ and

$$S = \begin{bmatrix} \cdots & \cdots \\ \cdots & \cdots \end{bmatrix}_{d\times d}$$

   Design a Bayesian classifier to classify a d-dimensional data into one of the c-classes. Assume $\sum P(w_i) = 1$, $i = 1, 2, \ldots, c$.

CODE

```
import csv
import numpy as np
import math

# Dataset filenames
TrainFile = "train-1.csv"  # or "train-2.csv"
TestFile = "test-1.csv"    # or "test-2.csv"

# ----------------------------------HELPER FUNCTIONS-----------------------
---------------------------------------------
# Load data from csv file
def load_csv(filename):
    lines = csv.reader(open(filename, "r", encoding='utf-8-sig'))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    dataset = np.asarray(dataset, dtype=np.float32)
    return dataset


# Seperate data by class
def class_sorted_data(dataset):
    classes = np.unique(dataset[:, np.size(dataset, 1) - 1])
    sortedclassdata = []
    for i in range(len(classes)):
        item = classes[i]
        itemindex = np.where(dataset[:, np.size(dataset, 1) - 1] == item)
# index  of rows with label class[i]
        singleclassdataset = dataset[itemindex, 0:np.size(dataset, 1) - 1]
# array  of data for class[i]
        sortedclassdata.append(np.matrix(singleclassdataset))
# matrix of data for class[i]
    return sortedclassdata, classes


# posterior prob = likelihood * prior probability
# function to calculate prior probability
def prior_prob(dataset, sortedclassdata):
    priorprob = []
    for i in range(len(sortedclassdata)):
        priorprob.append(len(sortedclassdata[i])/len(dataset))
    return priorprob


# function to find mean and covariance, so that likelihood can be
calculated
def find_mean(sortedclassdata):
    classmeans = []
```

```
    for i in range(len(sortedclassdata)):
        classmeans.append(sortedclassdata[i].mean(0))
    return classmeans


def find_covariance(sortedclassdata, classmeans):
    cov = []
    # total number of data points (rows) per class
    ndpc = len(sortedclassdata[0])
    for i in range(len(classmeans)):
        xn = np.transpose(sortedclassdata[i])
        mean_class = np.transpose(classmeans[i])
        tempvariance = sum([(xn[:, x] - mean_class) * np.transpose(xn[:, x]
- mean_class) for x in range(int(ndpc))])
        tempvariance = tempvariance / (ndpc - 1)
        cov.append(tempvariance)
    return cov


# find likelihood, given a gaussian distribuition
# and knowing the mean and variance, or in this case, the covariance
def find_n_class_probability(dataset, classmeans, covariance, priorProb,
classes):
    expo = []
    nclassprob = []
    probabilityofclass = []
    datasetDimensions = len(covariance[0])
    testdatasetMatrix = np.matrix(dataset)
    datasetTranspose = np.transpose(testdatasetMatrix[:,0:len(dataset[0])-
1])
    for i in range(len(dataset)):
        x = datasetTranspose[:, i]
        for j in range(len(classmeans)):
            determinate = np.linalg.det(covariance[j])
            if determinate == 0:
                addValue = 0.006*np.identity(datasetDimensions)
                covariance[j] = addValue + covariance[j]
                determinate = np.linalg.det(covariance[j])
            exponent = (-0.5)*np.transpose(x-
np.transpose(classmeans[j]))*np.linalg.inv(covariance[j])*(x-
np.transpose(classmeans[j]))
            expo.append(exponent)
            nprobabilityofclass =
priorProb[j]*(1/((2*math.pi)**(datasetDimensions/2)))*(1/(determinate**0.5)
)*math.exp(expo[j])
            probabilityofclass.append(nprobabilityofclass)
        arrayprob = np.array(probabilityofclass)
        nclassprob.append(classes[np.argmax(arrayprob)])
        probabilityofclass = []
        expo = []
    return nclassprob


def Accuracy(nclassprob, dataset):
    Classes = np.transpose([np.asarray(nclassprob, dtype=np.float32)])
    Truth = np.transpose([np.asarray(dataset[:, dataset.shape[1]-1])])
    validate = np.equal(Classes, Truth)
    accuracy = 100 * (np.sum(validate) / dataset.shape[0])
    return accuracy

def convert_covariance_to_naive(matrix):
```

```
    numofclasses = len(matrix)
    numoffeatures = len(matrix[0])
    for i in range(numofclasses):
        for j in range(numoffeatures):
            for k in range(numoffeatures):
                if j != k:
                    matrix[i][j, k] = 0
    print("Converted covariance to Naive Bayes")
    return matrix
#---------------------------------------------------------------------------
---------------------------------------------
print("\n**************************************************BAYESIAN
CLASSIFIER**************************************")
# loading Training data
trainingData = load_csv(TrainFile)
# loading Testing data
testingData = load_csv(TestFile)
testingData = testingData[0:1000]

no_classes = np.transpose([np.asarray(testingData[:, testingData.shape[1]-
1])])
no_classes = np.unique(no_classes)
print("\nClass Labels for this dataset - ")
for i in no_classes:
    print(i,end=" ")
print()

# getting sorted classes
sortclassdata, classes = class_sorted_data(trainingData)
# caculating Prior Probability of all classes
priorProb = prior_prob(trainingData, sortclassdata)
# calculating mean
#print("\n Mean by class of dataset - ")
meansbyclass = find_mean(sortclassdata)
#print(meansbyclass)
#print("\n Covariance - ")
# finding covariance
covariance = find_covariance(sortclassdata, meansbyclass)
#print(covariance)

print("\nBayes Classifer on Training Data\n")
nclassprob_train = find_n_class_probability(trainingData, meansbyclass,
covariance, priorProb, classes)
accuracy_train = Accuracy(nclassprob_train, trainingData)
print("Probable Classes assigned to Train Data samples by bayes classifier:
")
print("Train data size = ",len(nclassprob_train))
print(nclassprob_train)
print("Accuracy of Model on Training data  = ",round(accuracy_train,3),
"%")

print("\nBayes Classifer on Testing Data\n")
nclassprob_test = find_n_class_probability(testingData, meansbyclass,
covariance, priorProb, classes)
accuracy_test = Accuracy(nclassprob_test, testingData)
print("Test data size = ",len(nclassprob_test))
print("Probable Classes assigned to Test Data samples by bayes classifier:
")
print(nclassprob_test)
print("Accuracy of Model on Tsting data = ", round(accuracy_test,3), "%")
```

INPUT

	Feature-1	Feature-2	Class Label
0	3.305623	-1.112026	1
1	0.946703	0.345605	0
2	2.983970	-1.438217	1
3	-0.354255	-0.106622	0
4	0.748614	6.131478	1
...	...	...	...
394	0.193821	4.800389	1
395	-0.040993	-2.048766	0
396	3.024696	6.437311	1
397	0.364309	4.485938	0
398	2.825232	-2.160753	1

399 rows × 3 columns

Figure 1. Train Data-1

	Feature-1	Feature-2	Class Label
0	1.763668	-2.330878	1
1	0.549524	6.180992	0
2	1.018734	4.148406	1
3	-0.374638	-2.427788	0
4	1.230412	0.187759	1
...	...	...	...
95	-0.729839	-2.518054	0
96	2.858256	2.595878	1
97	0.833202	1.699691	0
98	0.436381	5.154859	1
99	-1.198550	-1.139769	0

100 rows × 3 columns

Figure 2. Test Data-1

	F1	F2	F3	F4	F4	F5	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	Class Label
0	1	0	0	1	1	1	1	0	0.0	0.0	0.000000	9	0.090900	0.739130	0.869565	6.000000	2
1	1	0	0	0	1	1	1	0	0.0	0.0	3.000000	0	4.421052	0.258824	1.636364	4.111111	3
2	1	1	0	0	0	0	0	0	0.0	0.0	0.739130	0	18.548390	1.022727	0.949152	0.346667	4
3	1	1	0	1	0	1	0	0	0.0	0.0	0.375000	1	0.000000	1.228261	1.333333	1.265625	5
4	0	1	0	0	0	1	0	0	0.0	0.0	2.333333	0	13.645160	1.186813	1.391304	0.431035	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2994	0	1	0	0	0	0	0	0	0.0	0.0	0.444444	1	16.960000	0.634921	1.000000	1.000000	6
2995	1	1	1	0	0	1	1	1	24.0	0.0	0.451613	1	1.064516	0.374302	0.507246	2.696970	7
2996	1	1	1	1	0	0	0	0	0.0	0.0	2.500000	1	0.428571	1.180328	1.821429	1.259259	8
2997	1	1	0	0	0	1	0	2	6.5	22.5	0.933333	2	4.068965	0.970803	1.333333	1.155844	9
2998	1	1	0	1	0	1	0	1	10.0	0.0	0.750000	2	0.000000	2.666667	2.583333	0.724138	10

2999 rows × 17 columns

Figure 3. Train Data-2

	F1	F2	F3	F4	F4	F5	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	Class Label
0	1	1	1	1	1	1	1	0	0.0	0.0	0.000000	16	0.096800	0.750000	0.700000	5.000000	2
1	1	0	0	0	0	0	0	0	0.0	0.0	1.600000	0	2.631579	0.369697	3.363636	1.380000	3
2	1	1	0	0	0	1	0	1	7.5	0.0	1.812500	0	7.655172	1.920455	1.642857	1.013514	4
3	1	1	0	1	0	0	0	0	0.0	0.0	0.333333	2	0.000000	2.203125	1.162162	0.442623	5
4	0	1	0	0	0	0	0	0	0.0	0.0	0.538462	0	32.000000	2.000000	1.875000	1.695652	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	1	1	0	0	0	1	1	1	19.5	0.0	0.441177	3	5.680000	0.127072	0.145161	1.596154	7
996	1	1	1	1	0	0	0	0	0.0	0.0	1.818182	0	0.064500	2.094340	2.538461	0.238095	8
997	1	1	1	0	0	1	0	2	8.5	22.5	1.107143	1	0.148148	1.361702	1.758065	1.841270	9
998	1	1	1	1	0	1	0	1	9.0	0.0	0.900000	0	0.074100	2.358974	2.594594	2.068182	10
999	1	1	0	0	0	1	0	1	17.0	0.0	0.750000	3	5.173913	0.771429	1.642857	1.711340	1

1000 rows × 17 columns

Figure 4. Test Data-2

OUTPUT

Dataset-1 Output

```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/BayesClassifier.py"

***************************************************BAYESIAN CLASSIFIER***************************************

Class Labels for this dataset -
0.0 1.0

 Mean by class of dataset -
[matrix([[0.12044239, 0.13603354]], dtype=float32), matrix([[2.04392  , 1.9988643]], dtype=float32)]

 Covariance -
[matrix([[0.9023751, 1.2711079],
        [1.2711079, 8.359462 ]], dtype=float32), matrix([[ 0.9055061, -1.0031031],
        [-1.0031031,  8.778898 ]], dtype=float32)]

Bayes Classifer on Training Data

Probable Classes assigned to Train Data samples by bayes classifier:
Train data size =  400
[0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0,
Accuracy of Model on Training data  =  86.0 %

Bayes Classifer on Testing Data

Test data size =  100
Probable Classes assigned to Test Data samples by bayes classifier:
[0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0,
Accuracy of Model on Tsting data =  87.0 %

Process finished with exit code 0
```

Dataset-2 Output

```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/BayesClassifier.py"

***************************************************BAYESIAN CLASSIFIER***************************************

Class Labels for this dataset -
1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0

Bayes Classifer on Training Data

Probable Classes assigned to Train Data samples by bayes classifier:
Train data size =  3000
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 1.0, 8.0, 9.0, 10.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6
Accuracy of Model on Training data  =  91.333 %

Bayes Classifer on Testing Data

Test data size =  1000
Probable Classes assigned to Test Data samples by bayes classifier:
[1.0, 2.0, 3.0, 4.0, 8.0, 6.0, 7.0, 4.0, 9.0, 10.0, 1.0, 2.0, 3.0, 4.0, 8.0, 7.0, 7.0, 10.0, 9.0, 8.0, 1.0, 2.0, 3.0, 4.0, 10.0,
Accuracy of Model on Tsting data =  87.3 %

Process finished with exit code 0
```

DAY-3

1. For a given dataset (e.g., iris data set) D of size $N \times M$ with N: number of samples and M: number of features, design a Bayesian classifier to classify the test data. Divide the data set into training and testing data according to random percentage split.

CODE

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
import math

print("\nBAYESIAN CLASSIFIER FOR IRIS DATASET BY YUKTI KHURANA\n")
# using covarience formula
def getCovarience(x, mean_point, M):
    N = len(x)
    # initialise the matrix with zeroes
    cov = np.zeros(shape = (M, M))
    for i in range(N):
        xim = np.matrix(x[i] - mean_point)
        cov += np.matmul(xim.T, xim)
    cov /= (N-1)
    return cov

# here no of features = 4, so dimensions M = 4
# Gaussian Theory Formula is used
def get_prob(sample, mean, cov, class_prob, M):
    cov_inv = np.linalg.inv(cov)
    cov_det = np.linalg.det(cov)

    t1 = (1/(2*math.pi)**M)*cov_det**0.5
    t2 = math.exp(-0.5*np.matmul((np.matmul(sample - mean, cov_inv)),
(sample - mean)))

    return t1*t2*class_prob

def accuracy(actual, predicted):
   correct = 0
   for i in range(len(actual)):
      if actual[i] == predicted[i]:
         correct += 1
   return correct / float(len(actual)) * 100.0

# loading the iris dataset
iris = datasets.load_iris()
target_iris_names =  list(iris.target_names)

# defining training data
X = iris.data   # input features
y = iris.target  # target features


# STRATIFIED SPLIT TO TRAIN AND TEST DATA
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size =
0.80, random_state =39, stratify = y)

# Initializing the count of each of all classes of flowers -
```

```
Versicolor,Setosa,Virginica
# let the total number of classes be 'C'
#C = len(target_iris_names)
C = len(np.unique(np.array(y_test)))
print("Number of Classes = ",C)

# set the total number of features/ dimensions
M = X.shape[1]
print("Number of Features = ",M)

# count of each class
class_cnts = [0]*C
for i in range(C):
    # if y_train is 0 means its class 1
    class_cnts[i] = np.count_nonzero(y_train == i)

# to get a matrix with class count number of rows and 4 number of columns
because there are 4 number of features
xmatrices = []
for  i in range(C):
    x = np.zeros(shape = (class_cnts[i], M), dtype = float)
    xmatrices.append(x)

# index of three classes
class_indices = [0]*C

for i in range(len(X_train)):
    # do for all classes
    for c in range(C):
        if (y_train[i] == c):
            xmatrices[c][class_indices[c]] = X_train[i].tolist()
            class_indices[c]+=1



# finding the mean of the all classes
class_means = []
for i in range(C):
    class_means.append(np.mean(xmatrices[i], axis = 0))


# getting the covarience matrix of each class
class_covs = []
for i in range(C):
    class_covs.append( getCovarience(xmatrices[i], class_means[i], M) )

# checking if all the covariance matrices are equal or not


#  calculating the class probabilities
class_probs = []
for i in range(C):
    class_probs.append( len(xmatrices[i]) / len(X_train) )


# declare the y_prediction np array of the length of test
y_pred = np.zeros(len(X_test))


# Testing
data_classes = [i for i in range(C)]
```

```
for z in range(len(X_test)):
    class_post_probs = []
    i = X_test[z]

    # calculate the probabilities for each class
    for c in range(C):
        cur_prob = get_prob(i, class_means[c], class_covs[c],
class_probs[c], M)
        class_post_probs.append(cur_prob)

    # finding the maximum probability as per bayesian decision theory
    max_prob = max(class_post_probs)
    for c in range(C):
        if (max_prob == class_post_probs[c]):
            y_pred[z] = data_classes[c]
            break

print("Predicted Class Labels for given Dataset = \n" + str(y_pred))
print()

for i in range(len(y_pred)):
    print("Test data = ", X_test[i])
    print(i," Predicted Value = iris ",target_iris_names[int(y_pred[i])])


print("\nAccuracy of Bayes model = ",round(accuracy(y_test, y_pred),3),
"%")
```

INPUT

Iris dataset is the input. Any other dataset can also be used.

OUTPUT

```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/generalized_bayes.py"

BAYESIAN CLASSIFIER FOR IRIS DATASET BY YUKTI KHURANA

Number of Classes =  3
Number of Features =  4
Predicted Class Labels for given Dataset =
[1. 0. 0. 2. 2. 1. 2. 1. 1. 0. 1. 2. 1. 2. 0. 2. 2. 1. 0. 2. 0. 0. 1. 2.
 0. 0. 2. 1. 2. 0.]

Test data =  [5.5 2.6 4.4 1.2]
0  Predicted Value = iris  versicolor
Test data =  [5.  3.6 1.4 0.2]
1  Predicted Value = iris  setosa
Test data =  [4.6 3.4 1.4 0.3]
2  Predicted Value = iris  setosa
Test data =  [5.7 2.5 5.  2. ]
3  Predicted Value = iris  virginica
Test data =  [6.5 3.  5.8 2.2]
4  Predicted Value = iris  virginica
Test data =  [5.6 2.5 3.9 1.1]
5  Predicted Value = iris  versicolor
Test data =  [6.  2.7 5.1 1.6]
6  Predicted Value = iris  virginica
Test data =  [6.  3.4 4.5 1.6]
7  Predicted Value = iris  versicolor
Test data =  [6.  2.2 4.  1. ]
8  Predicted Value = iris  versicolor
```

```
Test data =  [5.9 3.  5.1 1.8]
19  Predicted Value = iris  virginica
Test data =  [4.9 3.1 1.5 0.1]
20  Predicted Value = iris  setosa
Test data =  [4.5 2.3 1.3 0.3]
21  Predicted Value = iris  setosa
Test data =  [6.6 3.  4.4 1.4]
22  Predicted Value = iris  versicolor
Test data =  [6.7 3.3 5.7 2.5]
23  Predicted Value = iris  virginica
Test data =  [5.4 3.4 1.5 0.4]
24  Predicted Value = iris  setosa
Test data =  [5.1 3.5 1.4 0.2]
25  Predicted Value = iris  setosa
Test data =  [6.  3.  4.8 1.8]
26  Predicted Value = iris  virginica
Test data =  [4.9 2.4 3.3 1. ]
27  Predicted Value = iris  versicolor
Test data =  [6.3 3.4 5.6 2.4]
28  Predicted Value = iris  virginica
Test data =  [5.4 3.4 1.7 0.2]
29  Predicted Value = iris  setosa

Accuracy of Bayes model =  96.667 %

Process finished with exit code 0
```

2. For a given dataset (e.g., iris data set) D of size $N \times M$ with N: number of samples and M: number of features, design a (1) Euclidean distance classifier and (2) Mahalanobis distance classifier to classify the test data. Comment on the results. Assume classes are modeled by Gaussian distributions and classes to be equiprobable.

CODE

(1) Euclidean Distance Classifier Code

```
"""
EUCLIDEAN DISTANCE CLASSIFIER

The optimal Bayesian classifier is significantly simplified
under the following assumptions:

ϖ The classes are equiprobable.
ϖ The data in all classes follow Gaussian distributions.
ϖ The covariance matrix is the same for all classes.
ϖ The covariance matrix is diagonal and all elements
across the diagonal are equal.
That is, C = σ2I, I is the identity matrix.

"""


from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np
```

```python
print("EUCLIDEAN CLASSIFIER FOR IRIS DATASET BY YUKTI KHURANA\n")
# using covarience formula
def getCovarience(x, mean_point, M):
    N = len(x)
    # initialise the matrix with zeroes
    cov = np.zeros(shape = (M, M))
    for i in range(N):
        xim = np.matrix(x[i] - mean_point)
        cov += np.matmul(xim.T, xim)
    cov /= (N-1)
    return cov

# here no of features = 4, so dimensions M = 4
# Gaussian Theory Formula is used
def get_prob(sample, mean):
    var = (sample-mean)
    dist = (np.matmul(var, var.T))**0.5
    return dist

def accuracy(actual, predicted):
   correct = 0
   for i in range(len(actual)):
      if actual[i] == predicted[i]:
         correct += 1
   return correct / float(len(actual)) * 100.0

# loading the iris dataset
iris = datasets.load_iris()
target_iris_names =  list(iris.target_names)

# defining training data
X = iris.data    # input features
y = iris.target  # target features


# STRATIFIED SPLIT TO TRAIN AND TEST DATA
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size =
0.80, random_state = 41, stratify = y)

# Initializing the count of each of all classes of flowers -
Versicolor,Setosa,Virginica
# let the total number of classes be 'C'
#C = len(target_iris_names)
C = len(np.unique(np.array(y_test)))

# set the total number of features/ dimensions
M = X.shape[1]

print("Number of Classes = ",C)
print("Number of Features = ",M)

# count of each class
class_cnts = [0]*C

for i in range(C):
    # if y_train is 0 means its class 1
    class_cnts[i] = np.count_nonzero(y_train == i)

# to get a matrix with class count number of rows and 4 number of columns
because there are 4 number of features
xmatrices = []
```

```python
for  i in range(C):
    x = np.zeros(shape = (class_cnts[i], M), dtype = float)
    xmatrices.append(x)

# index of three classes
class_indices = [0]*C

for i in range(len(X_train)):
    # do for all classes
    for c in range(C):
        if (y_train[i] == c):
            xmatrices[c][class_indices[c]] = X_train[i].tolist()
            class_indices[c]+=1



# finding the mean of the all classes
class_means = []
for i in range(C):
    class_means.append(np.mean(xmatrices[i], axis = 0))


# getting the covarience matrix of each class
class_covs = []
for i in range(C):
    class_covs.append( getCovarience(xmatrices[i], class_means[i], M) )

# checking if the covariance is same for all classes or not
if (np.min(class_covs) == np.max(class_covs)):
    print("The covarience matrix is same for all three classes!")
#else:
    #print("The covarience matrix is NOT same for classes!\n\n")

#  calculating the class probabilities
class_probs = []
for i in range(C):
    class_probs.append( len(xmatrices[i]) / len(X_train) )


# declare the y_prediction np array of the length of test
y_pred = np.zeros(len(X_test))


# Testing
data_classes = [i for i in range(C)]


for z in range(len(X_test)):
    class_post_probs = []
    i = X_test[z]
    # calculate the probabilities for each class
    for c in range(C):
        cur_prob = get_prob(i, class_means[c])
        class_post_probs.append(cur_prob)
    # finding the min distance for mahalonobis
    max_prob = min(class_post_probs)
    for c in range(C):
        if (max_prob == class_post_probs[c]):
            y_pred[z] = data_classes[c]
            break
```

```
print("Predicted Class Labels for given Dataset = \n" + str(y_pred))
print()

for i in range(len(y_pred)):
    print("Test data = ", X_test[i])
    print(i," Predicted Value = iris ",target_iris_names[int(y_pred[i])])


print("\nAccuracy of model = ",round(accuracy(y_test, y_pred),3),"%")
```

(2) Mahalonobis Distance Classifier Code

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np


print("\nMAHALONOBIS CLASSIFIER FOR IRIS DATASET BY YUKTI KHURANA\n")
# using covarience formula
def getCovarience(x, mean_point, M):
    N = len(x)
    # initialise the matrix with zeroes
    cov = np.zeros(shape = (M, M))
    for i in range(N):
        xim = np.matrix(x[i] - mean_point)
        cov += np.matmul(xim.T, xim)
    cov /= (N-1)
    return cov

# here no of features = 4, so dimensions M = 4
# Gaussian Theory Formula is used
def get_prob(sample, mean, cov, class_prob, M):
    var = (sample - mean)
    dist1 = np.matmul(var, np.linalg.inv(cov))
    dist2 = np.matmul(dist1, var.T)
    return dist2 ** 0.5

def accuracy(actual, predicted):
   correct = 0
   for i in range(len(actual)):
      if actual[i] == predicted[i]:
         correct += 1
   return correct / float(len(actual)) * 100.0

# loading the iris dataset
iris = datasets.load_iris()
target_iris_names =  list(iris.target_names)

# defining training data
X = iris.data   # input features
y = iris.target  # target features


# STRATIFIED SPLIT TO TRAIN AND TEST DATA
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size =
0.80, random_state = 41, stratify = y)

# Initializing the count of each of all classes of flowers -
```

```
Versicolor,Setosa,Virginica
# let the total number of classes be 'C'
#C = len(target_iris_names)
C = len(np.unique(np.array(y_test)))

# set the total number of features/ dimensions
M = X.shape[1]

print("Number of Classes = ",C)
print("Number of Features = ",M)
# count of each class
class_cnts = [0]*C
for i in range(C):
    # if y_train is 0 means its class 1
    class_cnts[i] = np.count_nonzero(y_train == i)

# to get a matrix with class count number of rows and 4 number of columns
because there are 4 number of features
xmatrices = []
for  i in range(C):
    x = np.zeros(shape = (class_cnts[i], M), dtype = float)
    xmatrices.append(x)

# index of three classes
class_indices = [0]*C

for i in range(len(X_train)):
    # do for all classes
    for c in range(C):
        if (y_train[i] == c):
            xmatrices[c][class_indices[c]] = X_train[i].tolist()
            class_indices[c]+=1



# finding the mean of the all classes
class_means = []
for i in range(C):
    class_means.append(np.mean(xmatrices[i], axis = 0))


# getting the covarience matrix of each class
class_covs = []
for i in range(C):
    class_covs.append( getCovarience(xmatrices[i], class_means[i], M) )

# checking if the covariance is same for all classes or not
if (np.min(class_covs) == np.max(class_covs)):
    print("The covarience matrix is same for all three classes!")
#else:
    #print("The covarience matrix is NOT same for classes!\n\n")

#  calculating the class probabilities
class_probs = []
for i in range(C):
    class_probs.append( len(xmatrices[i]) / len(X_train) )


# declare the y_prediction np array of the length of test
y_pred = np.zeros(len(X_test))
```

```
# Testing
data_classes = [i for i in range(C)]


for z in range(len(X_test)):
    class_post_probs = []
    i = X_test[z]
    # calculate the probabilities for each class
    for c in range(C):
        cur_prob = get_prob(i, class_means[c], class_covs[c],
class_probs[c], M)
        class_post_probs.append(cur_prob)
    # finding the min distance for mahalonobis
    max_prob = min(class_post_probs)
    for c in range(C):
        if (max_prob == class_post_probs[c]):
            y_pred[z] = data_classes[c]
            break

print("Predicted Class Labels for IRIS Dataset = \n" + str(y_pred))
print()

for i in range(len(y_pred)):
    print("Test data = ", X_test[i])
    print(i," Predicted Flower = iris ",target_iris_names[int(y_pred[i])])


print("\nAccuracy of model = ",round(accuracy(y_test, y_pred),3), "%")
```

INPUT

Iris dataset is used. Any other dataset can also be used.

INFERENCE

The optimal Bayesian classifier is simplified under the following assumptions:

1. The classes are equiprobable.
2. The data in all classes follow Gaussian distribution.
3. The covariance matrix is the same for all classes.
4. The covariance matrix is diagonal and all elements across the diagonal are equal i.e. is, S=σ2I, where I is the identity matrix.

Under these assumptions, it turns out that the optimal Bayesian classifier is equivalent to the minimum Euclidean Distance Classifier. That is, given an unknown x, assign it to class i, if

$$\sqrt{(\boldsymbol{x}-\boldsymbol{\mu}_i)(\boldsymbol{x}-\boldsymbol{\mu}_i)^T} < \sqrt{(\boldsymbol{x}-\boldsymbol{\mu}_j)(\boldsymbol{x}-\boldsymbol{\mu}_j)^T} \;\; \forall \boldsymbol{i} \neq \boldsymbol{j}$$

And Mahalonobis Distance Classifier (not having the 4th condition)

$$\sqrt{(\boldsymbol{x}-\boldsymbol{\mu}_i)\boldsymbol{S}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T} < \sqrt{(\boldsymbol{x}-\boldsymbol{\mu}_j)\boldsymbol{S}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)^T} \;\; \forall \boldsymbol{i} \neq \boldsymbol{j}$$

As we can observe from the above results, that mahalonobis and euclidean classifiers give similar probability results as the Bayes model for class predictions as Euclidean and Mahalonobis classifiers are specialized/restrictive versions of the Bayesian classifier. On meeting their specific conditions, Bayes can be reduced to both these. However, given the tight set of assumptions under which

Eucledian works, it might not be practical to use in every dataset and thus not very efficient. Although, Euclidean classifier is often used, even if we know that the previously stated assumptions are not valid, because of its simplicity. It assigns a pattern to the class whose mean is closest to it with respect to the Euclidean norm.

OUTPUT

(1) Euclidean Distance Classifier Output

```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/generalized_Euclidean.py"
EUCLIDEAN CLASSIFIER FOR IRIS DATASET BY YUKTI KHURANA

Number of Classes =  3
Number of Features =  4
Predicted Class Labels for given Dataset =
[1. 1. 1. 0. 0. 2. 2. 1. 1. 1. 1. 1. 1. 1. 2. 2. 0. 1. 0. 2. 0. 0. 2. 1.
 0. 0. 0. 0. 2. 2.]

Test data =  [5.6 2.9 3.6 1.3]
0  Predicted Value = iris  versicolor
Test data =  [6.6 2.9 4.6 1.3]
1  Predicted Value = iris  versicolor
Test data =  [5.6 3.  4.1 1.3]
2  Predicted Value = iris  versicolor
Test data =  [4.9 3.1 1.5 0.2]
3  Predicted Value = iris  setosa
Test data =  [5.  3.4 1.6 0.4]
4  Predicted Value = iris  setosa
Test data =  [6.1 2.6 5.6 1.4]
5  Predicted Value = iris  virginica
Test data =  [6.3 3.4 5.6 2.4]
6  Predicted Value = iris  virginica
Test data =  [5.9 3.2 4.8 1.8]
7  Predicted Value = iris  versicolor
Test data =  [5.2 2.7 3.9 1.4]
8  Predicted Value = iris  versicolor
Test data =  [6.2 2.9 4.3 1.3]
9  Predicted Value = iris  versicolor
Test data =  [6.4 3.2 4.5 1.5]
10  Predicted Value = iris  versicolor
```

```
17  Predicted Value = iris  versicolor
Test data =  [5.  3.2 1.2 0.2]
18  Predicted Value = iris  setosa
Test data =  [6.3 2.8 5.1 1.5]
19  Predicted Value = iris  virginica
Test data =  [5.1 3.3 1.7 0.5]
20  Predicted Value = iris  setosa
Test data =  [4.9 3.  1.4 0.2]
21  Predicted Value = iris  setosa
Test data =  [6.5 3.  5.5 1.8]
22  Predicted Value = iris  virginica
Test data =  [5.  2.  3.5 1. ]
23  Predicted Value = iris  versicolor
Test data =  [5.1 3.5 1.4 0.2]
24  Predicted Value = iris  setosa
Test data =  [5.4 3.9 1.3 0.4]
25  Predicted Value = iris  setosa
Test data =  [5.1 3.7 1.5 0.4]
26  Predicted Value = iris  setosa
Test data =  [4.8 3.  1.4 0.1]
27  Predicted Value = iris  setosa
Test data =  [6.5 3.2 5.1 2. ]
28  Predicted Value = iris  virginica
Test data =  [7.3 2.9 6.3 1.8]
29  Predicted Value = iris  virginica

Accuracy of model =  93.333 %

Process finished with exit code 0
```

(2) Mahalonobis Distance Classifier Output

```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/generalized_Mahalanobis.py"

MAHALONOBIS CLASSIFIER FOR IRIS DATASET BY YUKTI KHURANA

Number of Classes =  3
Number of Features =  4
Predicted Class Labels for IRIS Dataset =
[1. 1. 1. 0. 0. 2. 2. 2. 1. 1. 1. 1. 2. 2. 2. 2. 0. 1. 0. 1. 0. 0. 2. 1.
 0. 0. 0. 0. 2. 2.]

Test data =  [5.6 2.9 3.6 1.3]
0  Predicted Flower = iris  versicolor
Test data =  [6.6 2.9 4.6 1.3]
1  Predicted Flower = iris  versicolor
Test data =  [5.6 3.  4.1 1.3]
2  Predicted Flower = iris  versicolor
Test data =  [4.9 3.1 1.5 0.2]
3  Predicted Flower = iris  setosa
Test data =  [5.  3.4 1.6 0.4]
4  Predicted Flower = iris  setosa
Test data =  [6.1 2.6 5.6 1.4]
5  Predicted Flower = iris  virginica
Test data =  [6.3 3.4 5.6 2.4]
6  Predicted Flower = iris  virginica
Test data =  [5.9 3.2 4.8 1.8]
7  Predicted Flower = iris  virginica
Test data =  [5.2 2.7 3.9 1.4]
8  Predicted Flower = iris  versicolor
Test data =  [6.2 2.9 4.3 1.3]
9  Predicted Flower = iris  versicolor
Test data =  [6.4 3.2 4.5 1.5]
```

```
17  Predicted Flower = iris  versicolor
Test data =  [5.  3.2 1.2 0.2]
18  Predicted Flower = iris  setosa
Test data =  [6.3 2.8 5.1 1.5]
19  Predicted Flower = iris  versicolor
Test data =  [5.1 3.3 1.7 0.5]
20  Predicted Flower = iris  setosa
Test data =  [4.9 3.  1.4 0.2]
21  Predicted Flower = iris  setosa
Test data =  [6.5 3.  5.5 1.8]
22  Predicted Flower = iris  virginica
Test data =  [5.  2.  3.5 1. ]
23  Predicted Flower = iris  versicolor
Test data =  [5.1 3.5 1.4 0.2]
24  Predicted Flower = iris  setosa
Test data =  [5.4 3.9 1.3 0.4]
25  Predicted Flower = iris  setosa
Test data =  [5.1 3.7 1.5 0.4]
26  Predicted Flower = iris  setosa
Test data =  [4.8 3.  1.4 0.1]
27  Predicted Flower = iris  setosa
Test data =  [6.5 3.2 5.1 2. ]
28  Predicted Flower = iris  virginica
Test data =  [7.3 2.9 6.3 1.8]
29  Predicted Flower = iris  virginica

Accuracy of model =  93.333 %

Process finished with exit code 0
```

DAY-4

1. Consider a dataset of your choice and implement k-means clustering algorithm.

CODE

Kmeans_clustering.py

```
"""
K MEANS CLUSTERING ALGORITHM
-> Clustering the data into k different clusters
-> using unsupervised learning as the dataset is unlabeled
-> Each sample is assigned to the cluster with the nearest mean

"""

import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime

# so that we can reproduce the data later
np.random.seed(42)

# to find th Euclidean Distance between two vectors
# this will help in calculating the distance between each data point and
cluster centers
def Euclidean_Dist(x1,x2):
    return np.sqrt( np.sum((x1-x2)**2))

class KMeans:
    num_plots=0
    # setting default values for the class
    # if k value not provided by user, k will be 5 and no of iterations
will be 100
    def __init__(self, K=5, max_iters=100, plot_steps=False):

        # initializing values
        self.K = K
        self.max_iters = max_iters
        self.plot_steps = plot_steps

        # list of sample indices for each cluster
        # for each cluster we initialize an empty list
        self.clusters = [ [] for _ in range(self.K) ]
        # mean feature vector for each cluster (actual samples)
        self.centroids = []

    # this method will input a list of chosen centroids
    # and assign each data point/sample to its nearest centroid and return
the clusters created
    def _create_clusters(self, centroids):
        # creating an empty list of K-lists for clusters
        clusters = [ [] for _ in range(self.K)]
        for index, sample in enumerate(self.X):
            # find the nearest centroid for current sample
            centroid_index  = self._nearest_centroid(sample,centroids)
            clusters[centroid_index].append(index)
        return clusters
```

```
    # returns the index of centroid in list which is nearest to the given
sample
    def _nearest_centroid(self, sample, centroids):
        distances = [ Euclidean_Dist(sample,c) for c in centroids ]
        min_dist_index = np.argmin(distances)
        return min_dist_index

    def _get_centroids(self, clusters):
        # creating an array filled with zeroes with tuple of K and number
of features
        centroids = np.zeros((self.K, self.n_features))
        # calculating the new centroid as the mean of all samples in the
cluster
        # clusters is the list of lists
        for cluster_index, cluster in enumerate(clusters):
            # finding mean of samples in the current cluster
            cluster_mean = np.mean(self.X[cluster],axis=0)
            centroids[cluster_index] = cluster_mean
        return centroids

    def _is_converged(self, old_centroids, new_centroids):
        dist = [Euclidean_Dist(old_centroids[i], new_centroids[i]) for i in
range(self.K)]
        # if there is no change in the distances of the old and new
centroids, means the algorithm has converged
        if(sum(dist) == 0):
            return True
        return False

    def _get_cluster_labels(self, clusters):
        # creating an empty NumPy array for storing the label of each
sample
        cluster_labels = [-1 for _ in range(self.n_samples)]
        for cluster_index, cluster in enumerate(clusters):
            for sample_index in cluster:
                cluster_labels[sample_index] = int(cluster_index)
        for i in range(len(cluster_labels)):
            cluster_labels[i] = "CLuster-"+str(cluster_labels[i])
        return cluster_labels

    def predict(self, X):
        # for storing data
        self.X = X
        # number of samples and features
        self.n_samples, self.n_features = X.shape

        # initialize the centroids
        # to randomly pick some samples
        # it will pick a random choice between 0 and number of samples
        # In KMeans algorithm initially, random samples are made centroids
and gradually optimization is done and new centroids selected
        random_sample_indices = np.random.choice(self.n_samples, self.K,
replace=False) # this will be an array of size self.K
        self.centroids = [self.X[i] for i in random_sample_indices]

        # optimization
        for _ in range(self.max_iters):
            # update clusters
            self.clusters = self._create_clusters(self.centroids)
            if self.plot_steps:
                self.plot_data()
```

```
            # update centroids
            old_centroids = self.centroids
            self.centroids =  self._get_centroids(self.clusters)
            print("----------------------------------------------------------------------------
-----------------------------------------")
            print("\nClusters : ")
            print(self.clusters)
            print("\n Old Centroids : ")
            print(old_centroids)
            print("\n New Centroids : ")
            print(self.centroids)
            print()

            if self.plot_steps:
                self.plot_data()

            # checking for convergence of algorithm
            if self._is_converged(old_centroids, self.centroids):
                # we can end the clustering algorithm now
                break

        # return cluster_labels
        return self._get_cluster_labels(self.clusters)

    def plot_data(self):
        KMeans.num_plots+=1
        figure, x = plt.subplots(figsize=(12,8))
        for i,index in enumerate(self.clusters):
            # .T is for the transpose of the array
            point = self.X[index].T
            # to plot all the points in different colors for different
clusters
            x.scatter(*point)
        # showing the centroids as markers to differentiate between them
and normal data points
        for point in self.centroids:
            x.scatter(*point, marker="x",color="black",linewidth=2)
        plt.show()
        plt.title("K-Means Clustering Graph by Yukti Khurana: "+"Plot
Number - "+str(KMeans.num_plots))
        plt.ylabel('Y-Position')
        plt.xlabel('X-Position')
        now = datetime.now()
        dt_string = now.strftime("%m%Y%H%M%S")
        name = str(dt_string)+" Plot Number -
"+str(KMeans.num_plots)+".png"
        #plt.show(block=False)
        plt.savefig("PlotImages/"+name)
        #plt.pause(3)
        plt.close()
```

Kmeans_testing.py

```
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from Kmeans_clustering import KMeans

print("\n************************************KMEANS CLUSTERING BY YUKTI
```

```
KHURANA**************************")
sample_no = 50
center_no = 3
feature_no = 2
X, y = make_blobs(centers=center_no , n_samples=sample_no,
n_features=feature_no, shuffle=True, random_state=42)
print("Number of samples = ", sample_no)
print("Number of features = ", feature_no)
print("Number of cluster centers/k = ", center_no)

clusters_len = len(np.unique(y))
k = KMeans(K=clusters_len, max_iters=150, plot_steps=True)
y_pred = k.predict(X)
print("\nThe class assigned to each datapoint : ")
print(y_pred)
k.plot_data()
```

INPUT

- Random Dataset is created at runtime using make_blobs. Any dataset can be used on this code
- Number of samples = 500
- Number of centers/k = 4
- Number of features = 2

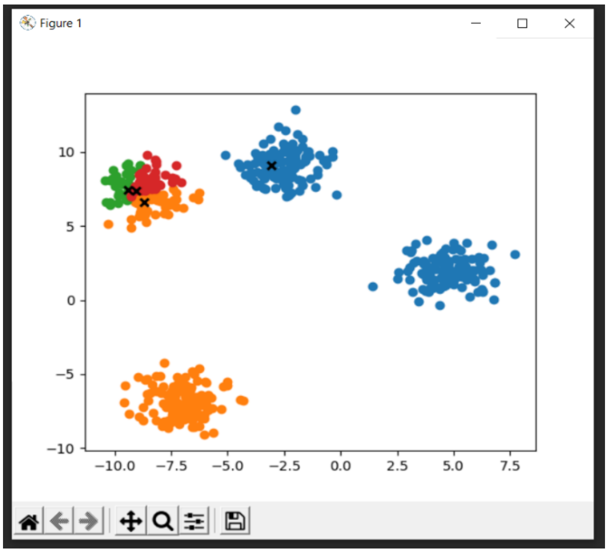
All the values above can be altered.

OUTPUT

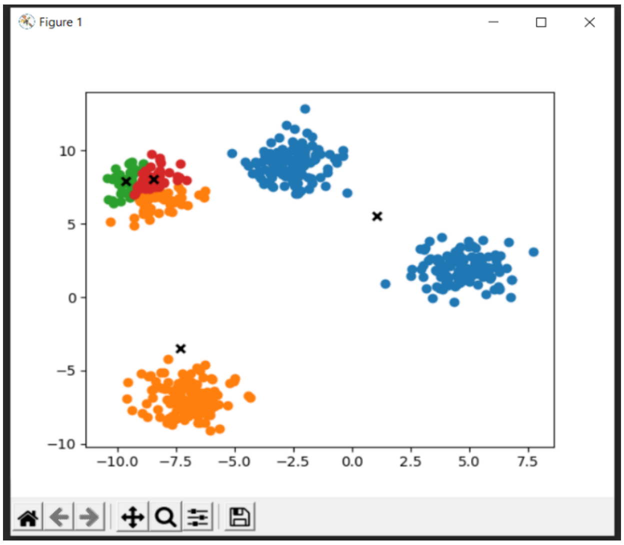
```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/Kmeans_tesing.py"

************************************KMEANS CLUSTERING BY YUKTI KHURANA****************************
Number of samples =  500
Number of features =  2
Number of cluster centers/k =  4
-------------------------------------------------------------------------------------------------

Clusters :
[[2, 3, 8, 10, 11, 12, 13, 15, 16, 19, 20, 21, 22, 29, 31, 32, 33, 36, 37, 41, 42, 43, 44, 45, 46, 47, 51, 54, 55, 56, 61,

 Old Centroids :
[array([-3.05358035,  9.12520872]), array([-8.71022334,  6.64247126]), array([-9.41396558,  7.44553273]), array([-9.0592919

 New Centroids :
[[ 1.05980458  5.50814533]
 [-7.30150873 -3.48732756]
 [-9.62524775  7.88630253]
 [-8.44203539  8.05797364]]

-------------------------------------------------------------------------------------------------

Clusters :
[[2, 3, 8, 10, 11, 12, 13, 15, 16, 20, 21, 22, 29, 31, 36, 37, 41, 42, 43, 44, 45, 46, 47, 54, 56, 61, 64, 68, 70, 76, 78,

 Old Centroids :
[[ 1.05980458  5.50814533]
 [-7.30150873 -3.48732756]
 [-9.62524775  7.88630253]
 [-8.44203539  8.05797364]]
```

```
 New Centroids :
[[ 2.03990148  4.7026632 ]
 [-7.0009649  -6.90445754]
 [-9.52444594  7.369551  ]
 [-6.4984573   8.14631416]]

-----------------------------------------------------------------------------------------------------

Clusters :
[[2, 8, 10, 11, 12, 15, 20, 22, 36, 37, 41, 44, 45, 46, 47, 54, 64, 68, 70, 78, 81, 88, 92, 94, 97, 100, 101, 102, 116, 121,

 Old Centroids :
[[ 2.03990148  4.7026632 ]
 [-7.0009649  -6.90445754]
 [-9.52444594  7.369551  ]
 [-6.4984573   8.14631416]]

 New Centroids :
[[ 4.27144311  2.46994348]
 [-7.0009649  -6.90445754]
 [-8.97484473  7.40821974]
 [-3.58476804  8.87554591]]

-----------------------------------------------------------------------------------------------------
```

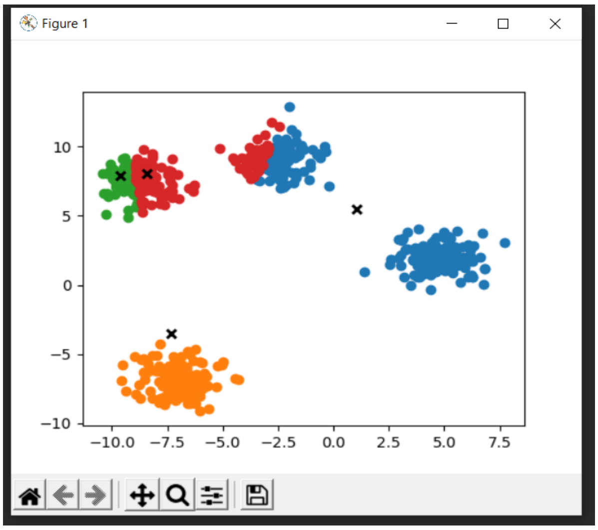
```
 New Centroids :
[[ 4.72182456  1.9238556 ]
 [-7.0009649  -6.90445754]
 [-8.68153751  7.4554521 ]
 [-2.60221541  9.09243505]]

-----------------------------------------------------------------------------------------------------

Clusters :
[[2, 8, 10, 12, 15, 20, 22, 36, 37, 41, 45, 46, 47, 54, 64, 68, 70, 78, 81, 88, 92, 94, 97, 100, 101, 102, 116, 121, 123, 125, 126, 131,

 Old Centroids :
[[ 4.72182456  1.9238556 ]
 [-7.0009649  -6.90445754]
 [-8.68153751  7.4554521 ]
 [-2.60221541  9.09243505]]

 New Centroids :
[[ 4.72182456  1.9238556 ]
 [-7.0009649  -6.90445754]
 [-8.68153751  7.4554521 ]
 [-2.60221541  9.09243505]]


The class assigned to each datapoint :
['CLuster-2', 'CLuster-1', 'CLuster-0', 'CLuster-3', 'CLuster-1', 'CLuster-1', 'CLuster-2', 'CLuster-1', 'CLuster-0', 'CLuster-1', 'CLust

Process finished with exit code 0
```



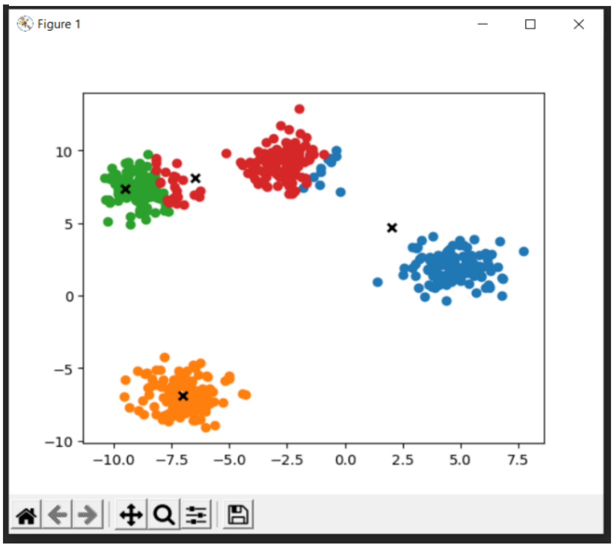
Clustering Step-1



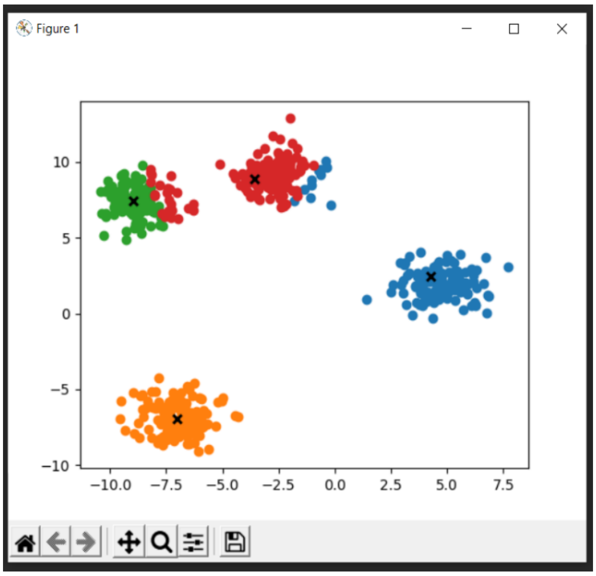
Clustering Step-2



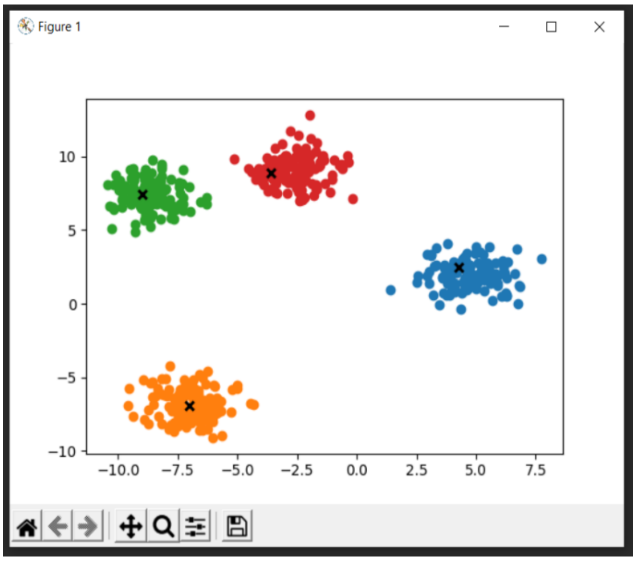
Clustering Step-3



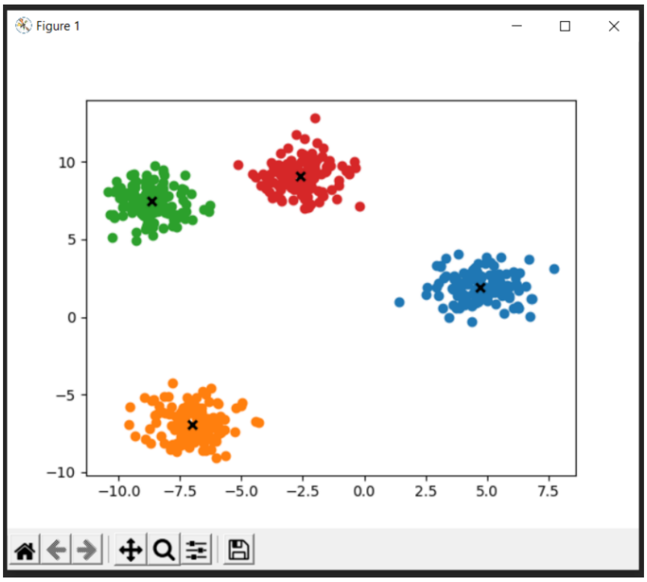
Clustering Step-4



Clustering Step-5



Clustering Step-6



Clustering Step-7 Final Clusters Formed

2. Consider a dataset of your choice and implement agglomerative clustering algorithm.

CODE

```
"""
Consider a dataset of your choice and implement agglomerative clustering
algorithm
"""

import numpy as np
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

# creating dataset to apply clustering on
X, y = make_blobs(centers=4, n_samples=15, n_features=3, shuffle=True,
random_state=42)
X = np.array(X)

print("************************************** Agglomerative Clustering
Algorithm *****************************************************\n")

def compute_distance(samples):
    # Creates a matrix of distances between individual samples and clusters
attained at a particular step
    Distance_mat = np.zeros((len(samples), len(samples)))
    for i in range(Distance_mat.shape[0]):
        for j in range(Distance_mat.shape[0]):
            if i != j:
                Distance_mat[i, j] = float(distance_calculate(samples[i],
samples[j]))
            else:
                Distance_mat[i, j] = 10 ** 4
    return Distance_mat
```

```
def intersampledist(s1, s2):
    if str(type(s2[0])) != '<class \'list\'>':
        s2 = [s2]
    if str(type(s1[0])) != '<class \'list\'>':
        s1 = [s1]
    m = len(s1)
    n = len(s2)
    dist = []
    if n >= m:
        for i in range(n):
            for j in range(m):
                if (len(s2[i]) >= len(s1[j])) and str(type(s2[i][0]) !=
'<class \'list\'>'):
                    dist.append(interclusterdist(s2[i], s1[j]))
                else:
                    dist.append(np.linalg.norm(np.array(s2[i]) -
np.array(s1[j])))
    else:
        for i in range(m):
            for j in range(n):
                if (len(s1[i]) >= len(s2[j])) and str(type(s1[i][0]) !=
'<class \'list\'>'):
                    dist.append(interclusterdist(s1[i], s2[j]))
                else:
                    dist.append(np.linalg.norm(np.array(s1[i]) -
np.array(s2[j])))
    return min(dist)

def interclusterdist( cl, sample):
    if sample[0] != '<class \'list\'>':
        sample = [sample]
    dist = []
    for i in range(len(cl)):
        for j in range(len(sample)):
            dist.append(np.linalg.norm(np.array(cl[i]) -
np.array(sample[j])))
    return min(dist)

def distance_calculate(sample1, sample2):
    dist = []
    for i in range(len(sample1)):
        for j in range(len(sample2)):
            try:
                dist.append(np.linalg.norm(np.array(sample1[i]) -
np.array(sample2[j])))
            except:
                dist.append(intersampledist(sample1[i], sample2[j]))
    return min(dist)


t = [[i] for i in range(X.shape[0])]
samples = [[list(X[i])] for i in range(X.shape[0])]
m = len(samples)

while m > 1:
    print('Current number of Clusters    :- ', m)
    Distance_mat = compute_distance(samples)
    sample_ind_needed = np.where(Distance_mat == Distance_mat.min())[0]
    value_to_add = samples.pop(sample_ind_needed[1])
    samples[sample_ind_needed[0]].append(value_to_add)
```

```
    print("Combining Clusters:  ")
    print(t[sample_ind_needed[0]]," and ", t[sample_ind_needed[1]])
    t[sample_ind_needed[0]].append(t[sample_ind_needed[1]])
    t[sample_ind_needed[0]] = [t[sample_ind_needed[0]]]
    v = t.pop(sample_ind_needed[1])
    m = len(samples)
    print("Current Status         :", t)
    print("Cluster attained:      :", t[sample_ind_needed[0]])
    print("Size after clustering  :", m)
    print('\n')

print("The algorithm has converged!!")
Z = linkage(X, 'single')
dn = dendrogram(Z)
plt.title("Agglomerative Clustering - Dendogram")
plt.show()
```

INPUT

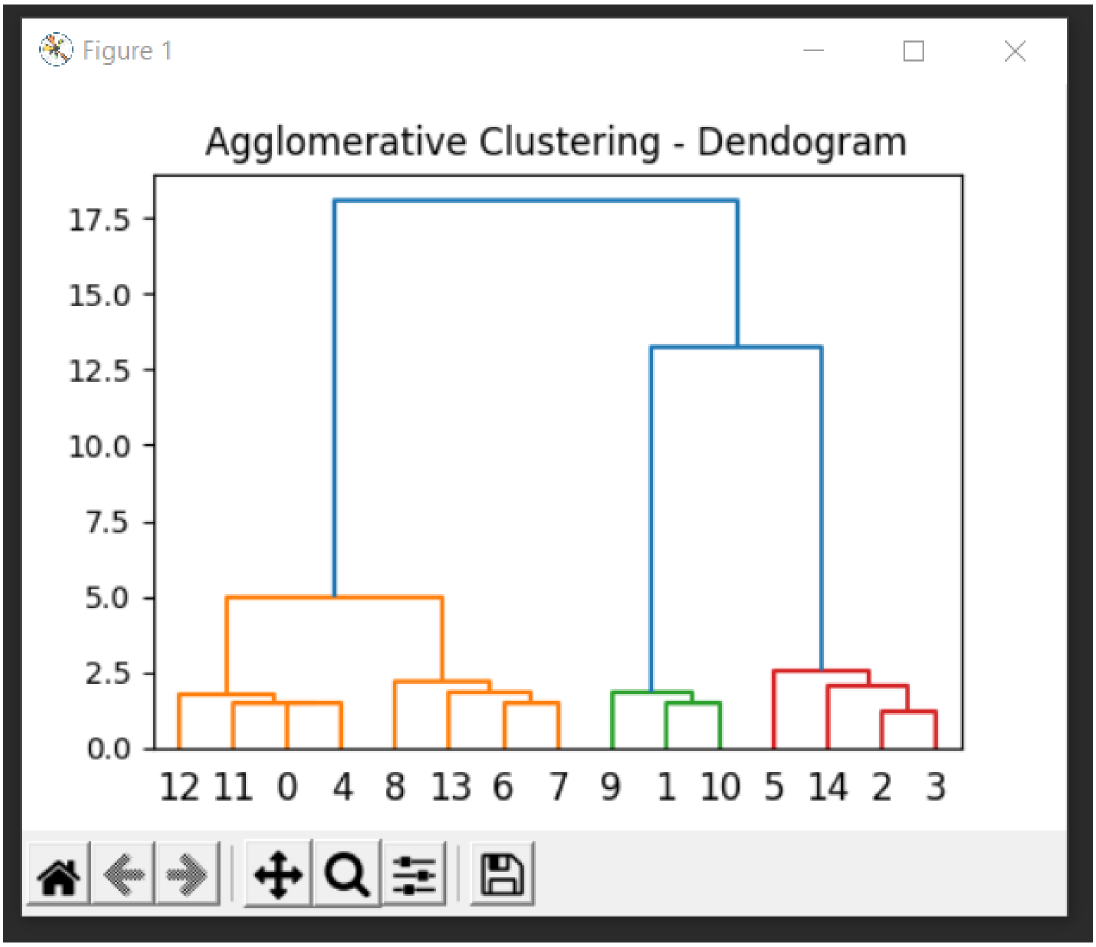
- Random Dataset is created at runtime using make_blobs. Any dataset can be used on this code
- Number of samples = 15
- Number of centers = 4
- Number of features = 3

OUTPUT

```
C:\Users\yukti\Desktop\atd-3\venv\Scripts\python.exe "C:/Users/yukti/Desktop/PR lab/ac.py"
************************************** Agglomerative Clustering Algorithm *****************************************************

Current number of Clusters   :-  15
Combining Clusters:
[2]  and  [3]
Current Status         : [[0], [1], [[2, [3]]], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]]
Cluster attained:      : [[2, [3]]]
Size after clustering  : 14


Current number of Clusters   :-  14
Combining Clusters:
[0]  and  [4]
Current Status         : [[[0, [4]]], [1], [[2, [3]]], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]]
Cluster attained:      : [[0, [4]]]
Size after clustering  : 13


Current number of Clusters   :-  13
Combining Clusters:
[6]  and  [7]
Current Status         : [[[0, [4]]], [1], [[2, [3]]], [5], [[6, [7]]], [8], [9], [10], [11], [12], [13], [14]]
Cluster attained:      : [[6, [7]]]
Size after clustering  : 12
```

```
Current number of Clusters   :-  12
Combining Clusters:
[[0, [4]]]  and  [11]
Current Status         : [[[[0, [4]], [11]]], [1], [[2, [3]]], [5], [[6, [7]]], [8], [9], [10], [12], [13], [14]]
Cluster attained:      : [[[0, [4]], [11]]]
Size after clustering  : 11


Current number of Clusters   :-  11
Combining Clusters:
[1]  and  [10]
Current Status         : [[[[0, [4]], [11]]], [[1, [10]]], [[2, [3]]], [5], [[6, [7]]], [8], [9], [12], [13], [14]]
Cluster attained:      : [[1, [10]]]
Size after clustering  : 10


Current number of Clusters   :-  10
Combining Clusters:
[[[0, [4]], [11]]]  and  [12]
Current Status         : [[[[[0, [4]], [11]], [12]]], [[1, [10]]], [[2, [3]]], [5], [[6, [7]]], [8], [9], [13], [14]]
Cluster attained:      : [[[[0, [4]], [11]], [12]]]
Size after clustering  : 9
```

```
Current Status         : [[[[[0, [4]], [11]], [12]]], [[[1, [10]], [9]]], [[[[2, [3]], [14]], [5]]], [[[[6, [7]], [13]], [8]]]]
Cluster attained:      : [[[[2, [3]], [14]], [5]]]
Size after clustering  : 4


Current number of Clusters   :-  4
Combining Clusters:
[[[[0, [4]], [11]], [12]]]  and  [[[[6, [7]], [13]], [8]]]
Current Status         : [[[[[[0, [4]], [11]], [12]], [[[[6, [7]], [13]], [8]]]]], [[[1, [10]], [9]]], [[[[2, [3]], [14]], [5]]]]
Cluster attained:      : [[[[[0, [4]], [11]], [12]], [[[[6, [7]], [13]], [8]]]]]
Size after clustering  : 3


Current number of Clusters   :-  3
Combining Clusters:
[[[[[0, [4]], [11]], [12]], [[[[6, [7]], [13]], [8]]]]]  and  [[[[2, [3]], [14]], [5]]]
Current Status         : [[[[[[[0, [4]], [11]], [12]], [[[[6, [7]], [13]], [8]]]], [[[[2, [3]], [14]], [5]]]]], [[[1, [10]], [9]]]]
Cluster attained:      : [[[[[[0, [4]], [11]], [12]], [[[[6, [7]], [13]], [8]]]], [[[[2, [3]], [14]], [5]]]]]
Size after clustering  : 2


Current number of Clusters   :-  2
Combining Clusters:
[[[[[[0, [4]], [11]], [12]], [[[[6, [7]], [13]], [8]]]], [[[[2, [3]], [14]], [5]]]]]  and  [[[1, [10]], [9]]]
Current Status         : [[[[[[[[0, [4]], [11]], [12]], [[[[6, [7]], [13]], [8]]]], [[[[2, [3]], [14]], [5]]]], [[[1, [10]], [9]]]]]]
Cluster attained:      : [[[[[[[0, [4]], [11]], [12]], [[[[6, [7]], [13]], [8]]]], [[[[2, [3]], [14]], [5]]]], [[[1, [10]], [9]]]]]
Size after clustering  : 1


The algorithm has converged!!
```



Dendogram showing clusters formed