Load Fisher Iris Data and understand the dataset. Divide 70% of the sample at random and keep it as training set. Plot the values (histogram) of each feature for all training sample and pick two features (out of 4) that you feel can be modelled using Gaussian distribution.

In [102…

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from math import pi,sqrt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

#newdataset
col_name=['SepalLength','SepalWidth','PetalLength','PetalWidth','Species']
iris=pd.read_csv('C:/Users/ashme/Mtech AI/Machine Learning/Lab Assignment/23-10-20 L
iris

#Preprocessing the last row to make it different classes
def name_class(ar):
    if ar == "setosa":
        ar=1
    elif ar == "versicolor":
        ar=2
    else: ar = 3
    return ar

iris['Species']=iris.Species.apply(name_class)
iris
#Preprocessing Done
```

Out[102…

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	5.1	3.5	1.4	0.2	1
2	4.9	3.0	1.4	0.2	1
3	4.7	3.2	1.3	0.2	1
4	4.6	3.1	1.5	0.2	1
5	5.0	3.6	1.4	0.2	1
...	...	...	...	...	...
146	6.7	3.0	5.2	2.3	3
147	6.3	2.5	5.0	1.9	3
148	6.5	3.0	5.2	2.0	3
149	6.2	3.4	5.4	2.3	3
150	5.9	3.0	5.1	1.8	3

150 rows × 5 columns

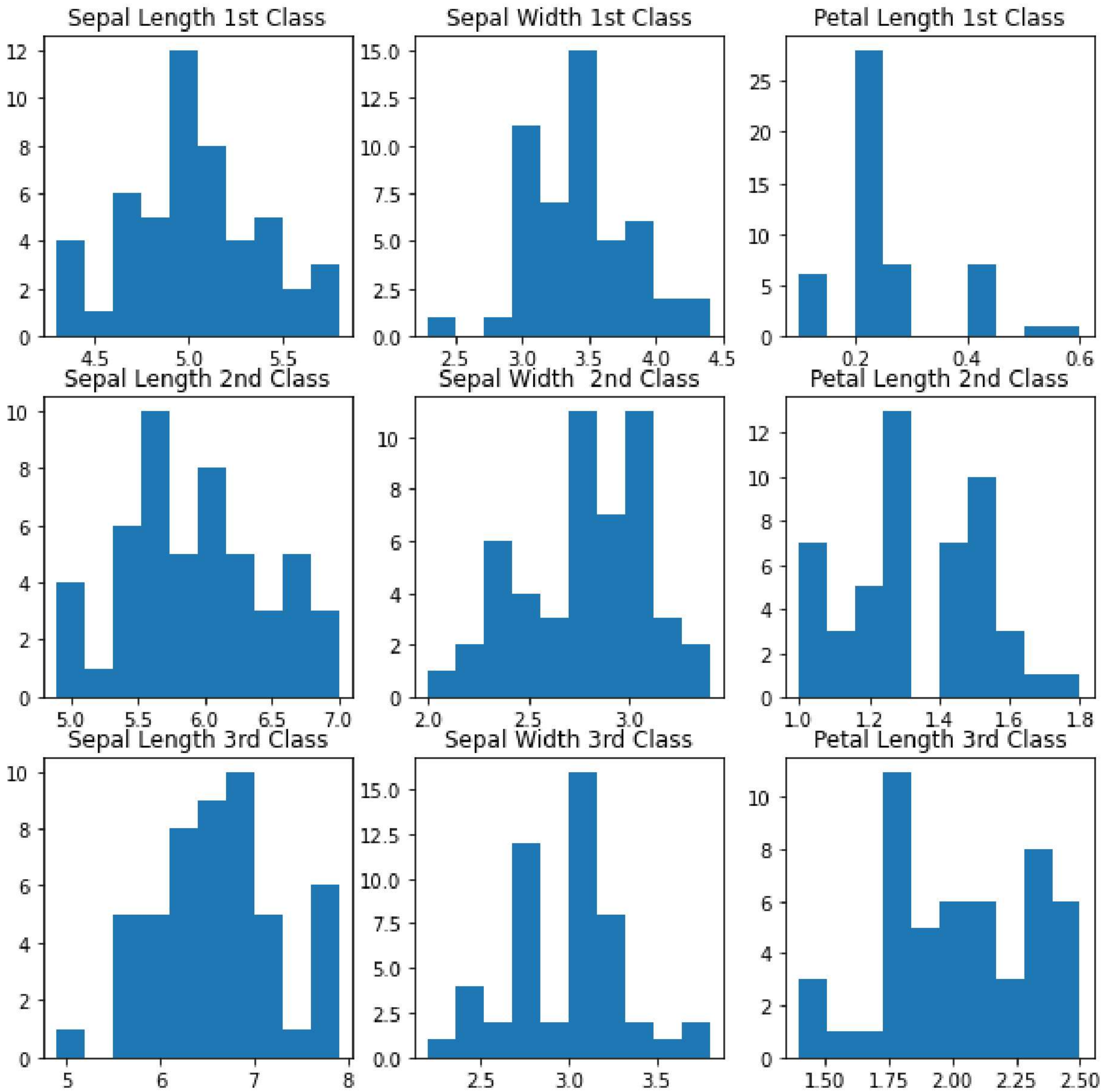
In [103…

```
fig,a=plt.subplots(3,3,figsize=(10,10))
a[0,0].hist(iris.iloc[:50,0])
a[0,0].set_title("Sepal Length 1st Class")
a[0,1].hist(iris.iloc[:50,1])
a[0,1].set_title("Sepal Width 1st Class")
a[0,2].hist(iris.iloc[:50,3])
a[0,2].set_title("Petal Length 1st Class")
a[1,0].hist(iris.iloc[50:100,0])
```

```
a[1,0].set_title("Sepal Length 2nd Class")
a[1,1].hist(iris.iloc[50:100,1])
a[1,1].set_title("Sepal Width  2nd Class")
a[1,2].hist(iris.iloc[50:100,3])
a[1,2].set_title("Petal Length 2nd Class")
a[2,0].hist(iris.iloc[100:150,0])
a[2,0].set_title("Sepal Length 3rd Class")
a[2,1].hist(iris.iloc[100:150,1])
a[2,1].set_title("Sepal Width 3rd Class")
a[2,2].hist(iris.iloc[100:150,3])
a[2,2].set_title("Petal Length 3rd Class")
```

Out[103…

```
Text(0.5, 1.0, 'Petal Length 3rd Class')
```



From this Histogram we can see that the Sepal Length and Sepal Width of all the three classes have a normal distribution. So we can pick these two features for our Model Building.

Question 1:

Now assume that the samples (with the above selected features) follows normal distribution. Design a Bayes classifier for the 30% test sample and report the accuracy.

In [104…

```
new_data=iris.drop(columns=['PetalLength','PetalWidth'])
new_data
```

Out[104…

	SepalLength	SepalWidth	Species
1	5.1	3.5	1
2	4.9	3.0	1
3	4.7	3.2	1
4	4.6	3.1	1
5	5.0	3.6	1
...	...	...	...
146	6.7	3.0	3
147	6.3	2.5	3
148	6.5	3.0	3
149	6.2	3.4	3
150	5.9	3.0	3

150 rows × 3 columns

In [106…

```
#Training and test for Class 1
train_class1=new_data.iloc[:50,:]
train_class2=new_data.iloc[50:100,:]
train_class3=new_data.iloc[100:150,:]
#Targets
#target1=new_data.iloc[:50,2]
#target2=new_data.iloc[50:100,2]
#target3=new_data.iloc[100:150,2]


#Splitting into train and test
X1_train,X1_test=train_test_split(train_class1,test_size=0.3)
X2_train,X2_test=train_test_split(train_class2,test_size=0.3)
X3_train,X3_test=train_test_split(train_class3,test_size=0.3)

#Create Training sets
X1_train=X1_train.drop(columns='Species')
X2_train=X2_train.drop(columns='Species')
X3_train=X3_train.drop(columns='Species')
```

In [107…

```
#Create Test set
X_test=pd.concat([X1_test,X2_test,X3_test]).sample(frac=1)
Y_test=X_test["Species"]
X_test=X_test.drop(columns="Species")
```

In [108…

```
#Function for finding mu and covariance
def training_func(X1):
    mu =X1.mean()
    cov= X1.cov()
    cov_inv=np.linalg.inv(cov)
    cov_det=np.linalg.det(cov)

    return mu,cov_inv,cov_det
```

In [109…

```
#test_class1.iloc[0,:].shape
```

In [110…

```
#Finding the Likelihood
```

```
def likelihood(test_case,train_c):
    mu1,cov_inv1,cov_det1= training_func(train_c)

    #using corrected equation
    P_like=np.exp(-.5*(np.transpose(test_case - mu1).dot(cov_inv1).dot(test_case-mu1

    return P_like


#Give each from combined X_test to all the three different likehoods and find thier
# 3 classes. Highest wins.
```

In [111…

```
#im=likelihood(class3.iloc[5,:],X1_train)
#im
```

In [112…

```
#prior=len(X1_train)/(len(X1_train)+len(X2_train)+len(X3_train))
```

In [113…

```
#Prior Probablity
prior_prob1=len(X1_train)/(len(X1_train)+len(X2_train)+len(X3_train))
prior_prob2=len(X2_train)/(len(X1_train)+len(X2_train)+len(X3_train))
prior_prob3=len(X2_train)/(len(X1_train)+len(X2_train)+len(X3_train))

#Defining posterior probability

def posterior(test_case_p,train_c1,train_c2,train_c3):

    like1=likelihood(test_case_p,train_c1)
    like2=likelihood(test_case_p,train_c2)
    like3=likelihood(test_case_p,train_c3)
    #print(like1,like2,like3)


    post1=prior_prob1*like1
    post2=prior_prob2*like2
    post3=prior_prob3*like3



    #print(post1,post2,post3)
    if post1>post2 and post1>post3:
        out = 1
    elif post2>post1 and post2>post3:
        out = 2
    elif post3>post1 and post3>post2:
        out = 3

    return out
```

In [120…

```
#P1=posterior(class3.iloc[],X1_train,X2_train,X3_train)
#P1
```

In [115…

```
#Getting the values for the test case
def test(test_f,train_f1,train_f2,train_f3):
    y_pred1=[]
    for i in range(test_f.shape[0]):
        t_p= test_f.iloc[i,:]
        out_pred= posterior(t_p,train_f1,train_f2,train_f3)
        y_pred1.append(out_pred)
    return y_pred1
```

In [116…

```
Y_pred=test(X_test,X1_train,X2_train,X3_train)
Y_pred
```

Out[116…
```
[3,
 2,
 2,
 2,
 1,
 2,
 1,
 3,
 2,
 3,
 1,
 1,
 1,
 1,
 3,
 3,
 1,
 1,
 2,
 3,
 3,
 3,
 2,
 2,
 2,
 1,
 3,
 1,
 3,
 3,
 3,
 2,
 2,
 1,
 3,
 1,
 3,
 1,
 3,
 3,
 1,
 2,
 3,
 1,
 2]
```

In [117…
```
#Our prediction using Baseyian
r=confusion_matrix(Y_test,Y_pred)
r
```

Out[117…
```
array([[15,  0,  0],
       [ 0, 10,  5],
       [ 0,  3, 12]], dtype=int64)
```

In [118…
```
test_acc=(r[0,0]+r[1,1]+r[2,2])/len(Y_test)
test_acc
```

Out[118…
```
0.8222222222222222
```

In [119…
```
accuracy_score(Y_test,Y_pred)
```

Out[119…
```
0.8222222222222222
```

Question 2:

Repeat step 1, but this time build a naïve Bayes classifier.

In [39]:

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.naive_bayes import GaussianNB

col_name=['SepalLength','SepalWidth','PetalLength','PetalWidth','Species']
iris=pd.read_csv('C:/Users/ashme/Mtech AI/Machine Learning/Lab Assignment/23-10-20 L
iris

#Preprocessing the last row to make it different classes
def name_class(ar):
    if ar == "setosa":
        ar=1
    elif ar == "versicolor":
        ar=2
    else: ar = 3
    return ar

iris['Species']=iris.Species.apply(name_class)
iris
#Preprocessing Done
```

Out[39]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	5.1	3.5	1.4	0.2	1
2	4.9	3.0	1.4	0.2	1
3	4.7	3.2	1.3	0.2	1
4	4.6	3.1	1.5	0.2	1
5	5.0	3.6	1.4	0.2	1
...	...	...	...	...	...
146	6.7	3.0	5.2	2.3	3
147	6.3	2.5	5.0	1.9	3
148	6.5	3.0	5.2	2.0	3
149	6.2	3.4	5.4	2.3	3
150	5.9	3.0	5.1	1.8	3

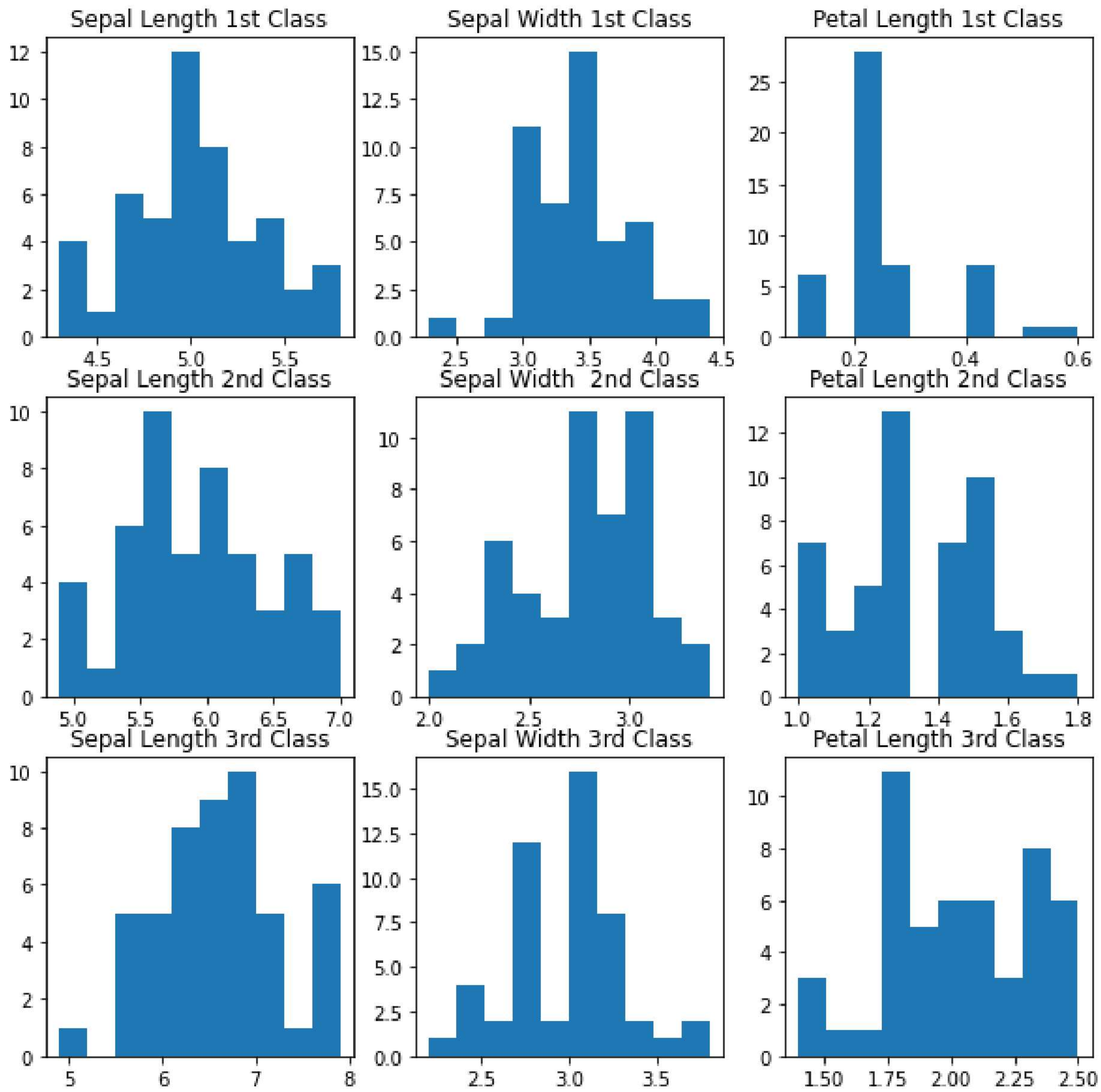
150 rows × 5 columns

In [40]:

```
fig,a=plt.subplots(3,3,figsize=(10,10))
a[0,0].hist(iris.iloc[:50,0])
a[0,0].set_title("Sepal Length 1st Class")
a[0,1].hist(iris.iloc[:50,1])
a[0,1].set_title("Sepal Width 1st Class")
a[0,2].hist(iris.iloc[:50,3])
a[0,2].set_title("Petal Length 1st Class")
a[1,0].hist(iris.iloc[50:100,0])
```

```
a[1,0].set_title("Sepal Length 2nd Class")
a[1,1].hist(iris.iloc[50:100,1])
a[1,1].set_title("Sepal Width  2nd Class")
a[1,2].hist(iris.iloc[50:100,3])
a[1,2].set_title("Petal Length 2nd Class")
a[2,0].hist(iris.iloc[100:150,0])
a[2,0].set_title("Sepal Length 3rd Class")
a[2,1].hist(iris.iloc[100:150,1])
a[2,1].set_title("Sepal Width 3rd Class")
a[2,2].hist(iris.iloc[100:150,3])
a[2,2].set_title("Petal Length 3rd Class")
```

Out[40]: Text(0.5, 1.0, 'Petal Length 3rd Class')



In [41]:
```
new_data=iris.drop(columns=['PetalLength','PetalWidth'])
new_data
```

Out[41]:

	SepalLength	SepalWidth	Species
1	5.1	3.5	1
2	4.9	3.0	1
3	4.7	3.2	1
4	4.6	3.1	1
5	5.0	3.6	1

	SepalLength	SepalWidth	Species
...	...	...	...
146	6.7	3.0	3
147	6.3	2.5	3
148	6.5	3.0	3
149	6.2	3.4	3
150	5.9	3.0	3

150 rows × 3 columns

In [42]:
```
train_class1=new_data.iloc[:50,:]
train_class2=new_data.iloc[50:100,:]
train_class3=new_data.iloc[100:150,:]

#Splitting into train and test
X1_train,X1_test=train_test_split(train_class1,test_size=0.3)
X2_train,X2_test=train_test_split(train_class2,test_size=0.3)
X3_train,X3_test=train_test_split(train_class3,test_size=0.3)

#Create Training sets
X_train=pd.concat([X1_train,X2_train,X3_train]).sample(frac=1)
Y_train=X_train["Species"]
X_train=X_train.drop(columns="Species")
```

In [43]:
```
#Create Test set
X_test=pd.concat([X1_test,X2_test,X3_test]).sample(frac=1)
Y_test=X_test["Species"]
X_test=X_test.drop(columns="Species")
```

In [44]:
```
gnb = GaussianNB()
gnb_classifier = gnb.fit(X_train, Y_train)
```

In [45]:
```
#testing the training set using inbuilt classifier
Y_pred = gnb_classifier.predict(X_train)
```

In [46]:
```
cm_train = confusion_matrix(Y_train, Y_pred)
cm_train
```

Out[46]:
```
array([[34,  1,  0],
       [ 0, 26,  9],
       [ 0, 10, 25]], dtype=int64)
```

In [54]:
```
accuracy=accuracy_score(Y_train,Y_pred)
accuracy
print('Training Accuracy '+str(accuracy*100)+" %")
```

```
Training Accuracy 80.95238095238095 %
```

In [48]:
```
#testing using testing case
gnb_classifier = gnb.fit(X_train, Y_train)
Y_pred1 = gnb_classifier.predict(X_test)
```

In [49]:
```
cm_train = confusion_matrix(Y_test, Y_pred1)
cm_train
```

Out[49]:
```
array([[15,  0,  0],
       [ 0, 10,  5],
```

```
        [ 0,  4, 11]], dtype=int64)
```

In [72]:

```
accuracy=accuracy_score(Y_test,Y_pred1)
accuracy
print('Testing Accuracy '+str(accuracy*100)+" %")
```

```
Testing Accuracy 80.0 %
```

Building a Naive Base using Formula

In [57]:

```
#Function for finding mu and covariance
def training_func(X1):
    mu =X1.mean()
    sig=np.std(X1)

    return mu,sig
```

In [58]:

```
#training_func(X1_train.iloc[0,:2])
```

In [59]:

```
#X1_test.SepalWidth
```

In [60]:

```
def likelihood(test_case,train_c):
    train_a=train_c.iloc[:,0]
    train_b=train_c.iloc[:,1]
    test_a=test_case.SepalLength
    test_b=test_case.SepalWidth
    mu1,sig1= training_func(train_a)
    mu2,sig2= training_func(train_b)

    #using corrected equation
    P_like1=np.exp(-0.5*((test_a-mu1)**2)/(sig1**2))/(np.sqrt(2*3.14*(sig1**2)))
    P_like2=np.exp(-0.5*((test_b-mu2)**2)/(sig2**2))/(np.sqrt(2*3.14*(sig2**2)))

    P_like=P_like1*P_like2
    return P_like
```

In [61]:

```
#likelihood(X1_test.iloc[5,:2],X1_train)
```

In [62]:

```
prior_prob1=len(X1_train)/(len(X1_train)+len(X2_train)+len(X3_train))
prior_prob2=len(X2_train)/(len(X1_train)+len(X2_train)+len(X3_train))
prior_prob3=len(X2_train)/(len(X1_train)+len(X2_train)+len(X3_train))

#Defining posterior probability

def posterior(test_case_p,train_c1,train_c2,train_c3):

    like1=likelihood(test_case_p,train_c1)
    like2=likelihood(test_case_p,train_c2)
    like3=likelihood(test_case_p,train_c3)
    #print(like1,like2,like3)


    post1=prior_prob1*like1
    post2=prior_prob2*like2
    post3=prior_prob3*like3



    #print(post1,post2,post3)
    if post1>post2 and post1>post3:
        out = 1
    elif post2>post1 and post2>post3:
        out = 2
```

```
    else:
        out = 3

    return out
```

In [63]:
```
#P1=posterior(X3_test.iloc[5,:2],X1_train,X2_train,X3_train)
#P1
```

In [64]:
```
def test(test_f,train_f1,train_f2,train_f3):
    y_pred1=[]
    for i in range(test_f.shape[0]):
        t_p= test_f.iloc[i,:2]
        out_pred= posterior(t_p,train_f1,train_f2,train_f3)
        y_pred1.append(out_pred)
    return y_pred1
```

In [65]:
```
#Our prediction using Naive Baseyian
Y_pred_1=test(X_test,X1_train,X2_train,X3_train)
Y_pred_1
```

Out[65]:
```
[2,
 2,
 1,
 1,
 3,
 3,
 1,
 2,
 1,
 3,
 1,
 1,
 2,
 3,
 1,
 3,
 2,
 1,
 3,
 1,
 3,
 3,
 3,
 3,
 2,
 2,
 2,
 2,
 1,
 3,
 2,
 3,
 1,
 3,
 3,
 3,
 2,
 1,
 1,
 2,
 1,
 2,
 1,
 3,
 2]
```

In [66]:
```
#Testing Accuracy
r=confusion_matrix(Y_test,Y_pred_1)
r
```

Out[66]:
```
array([[15,  0,  0],
       [ 0, 10,  5],
       [ 0,  4, 11]], dtype=int64)
```

In [73]:
```
test_acc=(r[0,0]+r[1,1]+r[2,2])/len(Y_test)
test_acc
print('Testing Accuracy '+str(test_acc*100)+" %")
```

```
Testing Accuracy 80.0 %
```

In [70]:
```
accuracy1=accuracy_score(Y_test,Y_pred_1)
print('Testing Accuracy '+str(accuracy1*100)+" %")
```

```
Testing Accuracy 80.0 %
```

Inference

1. We have implemented Naive Base classifier and tested the accuracy to be 80% for the chosen features.

In [ ]: