Classification Assignment

Problem Statement or Requirement:

A requirement from the Hospital, Management asked us to create a predictive model which will predict the Chronic Kidney Disease (CKD) based on the several parameters. The Client has provided the dataset of the same.

- **Identify your problem statement**

  Here the problem statement is to identify whether any individual/patient will have CKD in near future based on several i/p parameters that the hospital has captured. Since the i/p is in numerical formal we can use **MACHINE LEARNING** for providing the solution. Also, since the requirement is clear and we have both the i/p and o/p data handy this will come under **SUPERVISED LEARNING**. Further the o/p is categorical and hence we would go ahead with **CLASSIFICATION**

- **Tell basic info about the dataset (Total number of rows, columns)**

  The dataset that has been provided:

  i. Columnà 25
     1. 13- numerical
     2. 12- categorical

  ii. Rowsà 399

- **Mention the pre-processing method if you're doing any (like converting string to number – nominal data)**

  Since the data is NOMINAL, we should be using **ONE-HOT ENCODING** to update to numerical format for our Python code to handle it

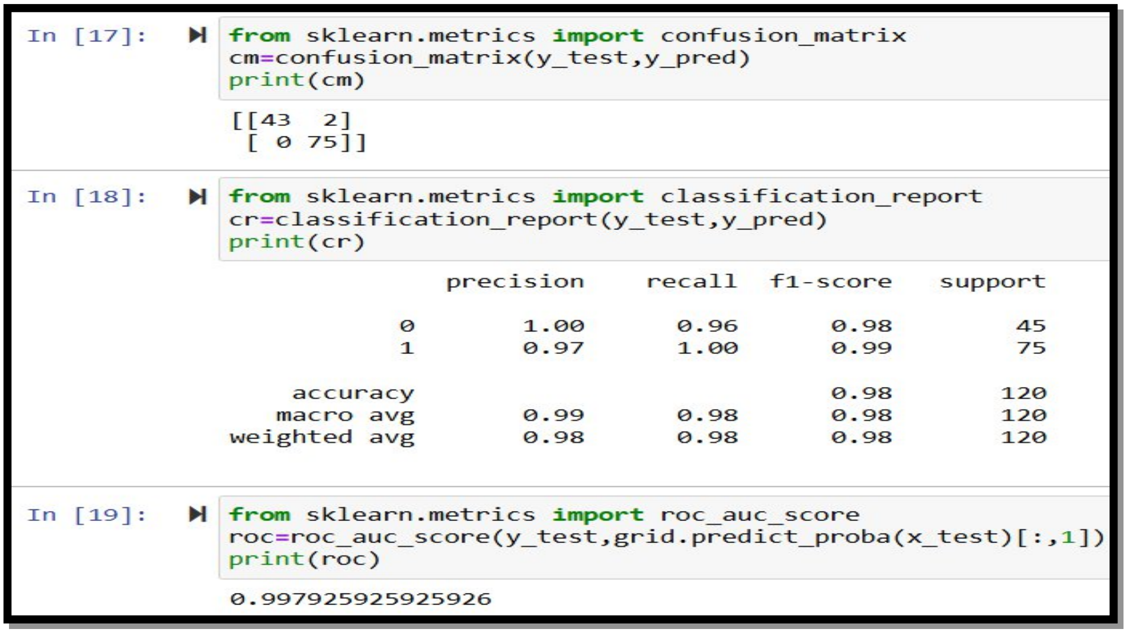
- **Develop a good model with good evaluation metric. You can use any machine learning algorithm; you can create many models. Finally, you have to come up with final model.**

  Used GRIDSERACH for all the 6 algorithms below:

  - ➢ Logistic Regression
  - ➢ SVC
  - ➢ Decision Tree
  - ➢ Random forest
  - ➢ KNN
  - ➢ Naïve Bayes

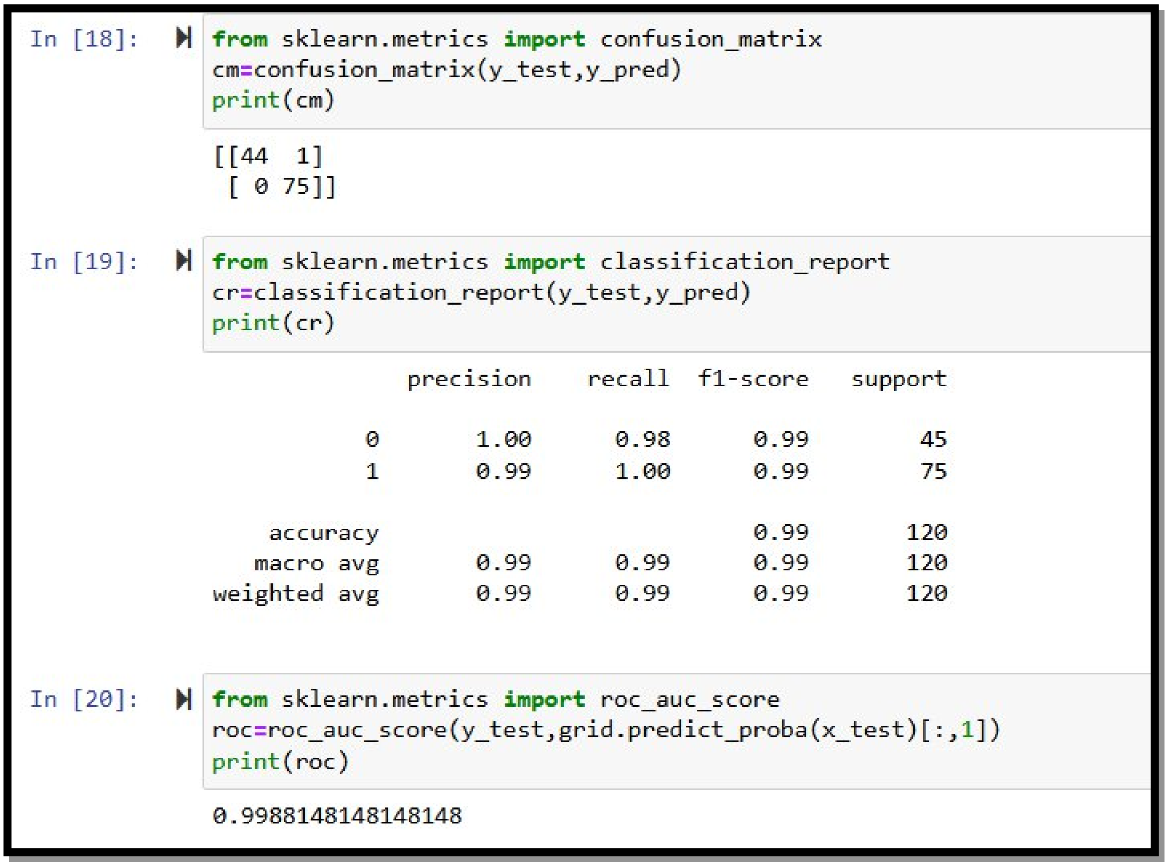
- **All the research values of each algorithm should be documented. (You can make tabulation or screenshot of the results.)**

a) Logistic Regression



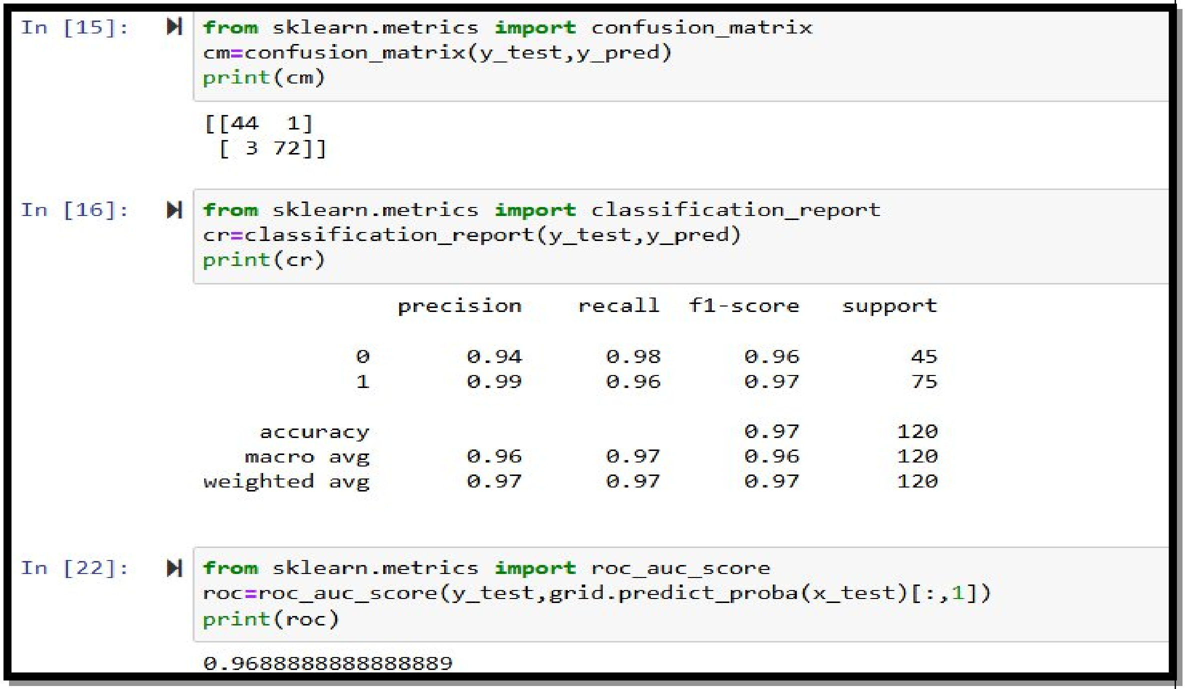
```
In [17]:  from sklearn.metrics import confusion_matrix
          cm=confusion_matrix(y_test,y_pred)
          print(cm)

          [[43  2]
           [ 0 75]]

In [18]:  from sklearn.metrics import classification_report
          cr=classification_report(y_test,y_pred)
          print(cr)

                        precision    recall  f1-score   support

                     0       1.00      0.96      0.98        45
                     1       0.97      1.00      0.99        75

              accuracy                           0.98       120
             macro avg       0.99      0.98      0.98       120
          weighted avg       0.98      0.98      0.98       120

In [19]:  from sklearn.metrics import roc_auc_score
          roc=roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
          print(roc)

          0.997925925925926
```

b) SVC



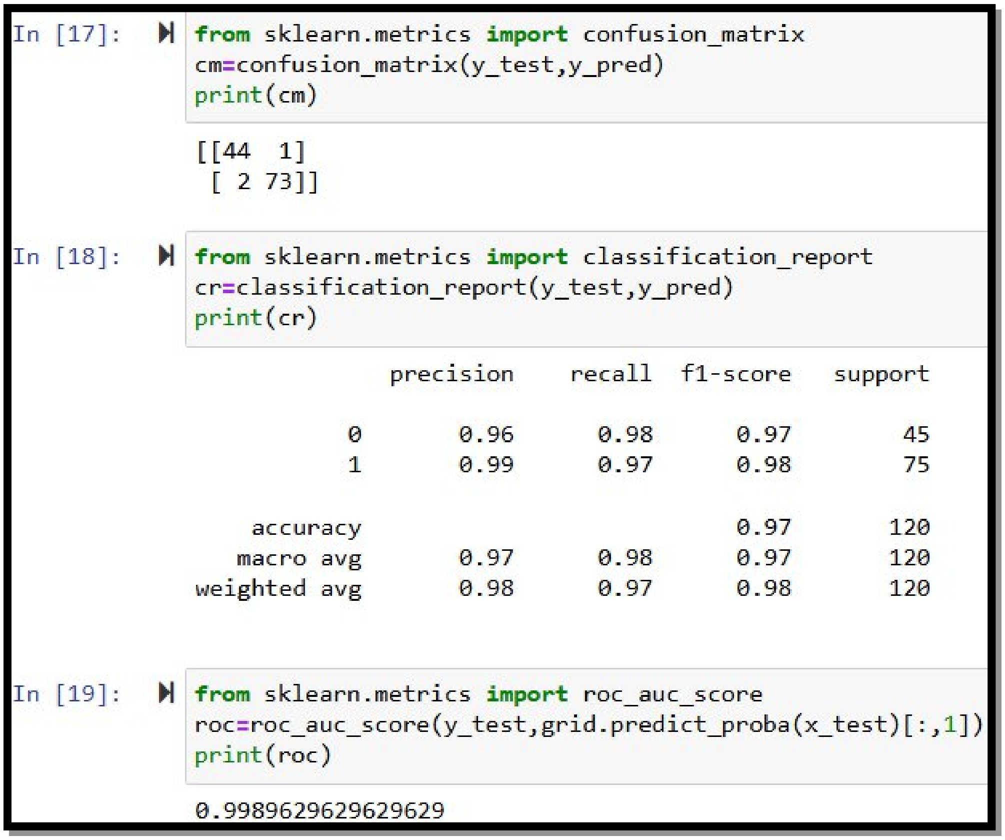
```
In [18]:  from sklearn.metrics import confusion_matrix
          cm=confusion_matrix(y_test,y_pred)
          print(cm)

          [[44  1]
           [ 0 75]]

In [19]:  from sklearn.metrics import classification_report
          cr=classification_report(y_test,y_pred)
          print(cr)

                        precision    recall  f1-score   support

                     0       1.00      0.98      0.99        45
                     1       0.99      1.00      0.99        75

              accuracy                           0.99       120
             macro avg       0.99      0.99      0.99       120
          weighted avg       0.99      0.99      0.99       120

In [20]:  from sklearn.metrics import roc_auc_score
          roc=roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
          print(roc)

          0.9988148148148148
```

c) DecisionTree



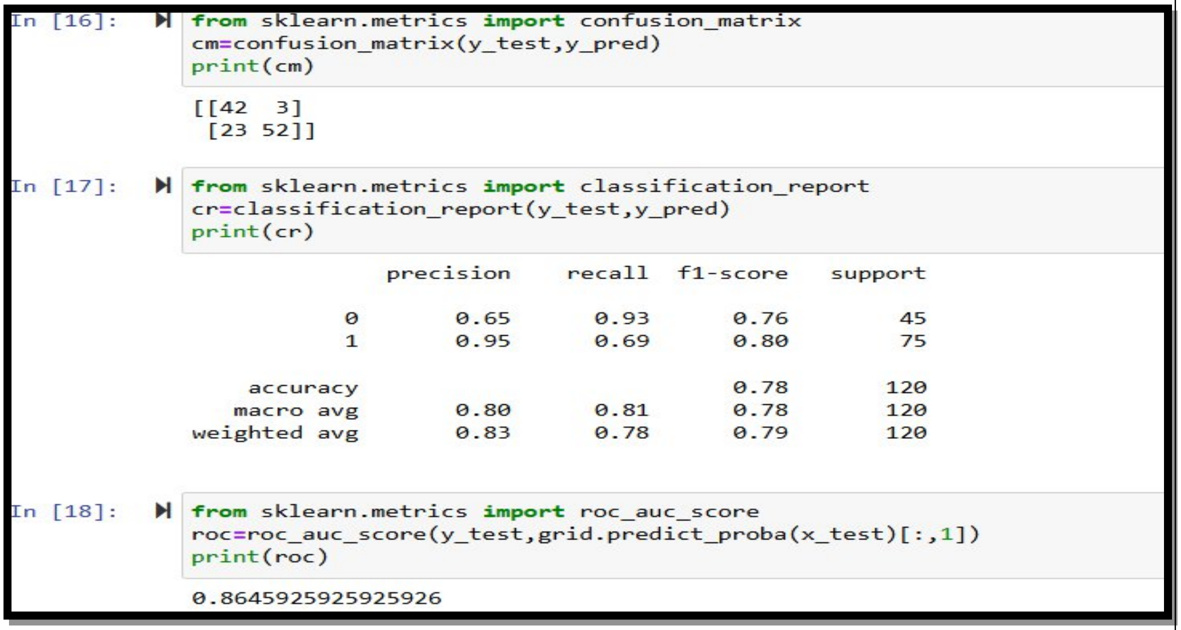
```
In [15]:  from sklearn.metrics import confusion_matrix
          cm=confusion_matrix(y_test,y_pred)
          print(cm)

          [[44  1]
           [ 3 72]]

In [16]:  from sklearn.metrics import classification_report
          cr=classification_report(y_test,y_pred)
          print(cr)

                        precision    recall  f1-score   support

                     0       0.94      0.98      0.96        45
                     1       0.99      0.96      0.97        75

              accuracy                           0.97       120
             macro avg       0.96      0.97      0.96       120
          weighted avg       0.97      0.97      0.97       120


In [22]:  from sklearn.metrics import roc_auc_score
          roc=roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
          print(roc)

          0.9688888888888889
```

d) Random forest



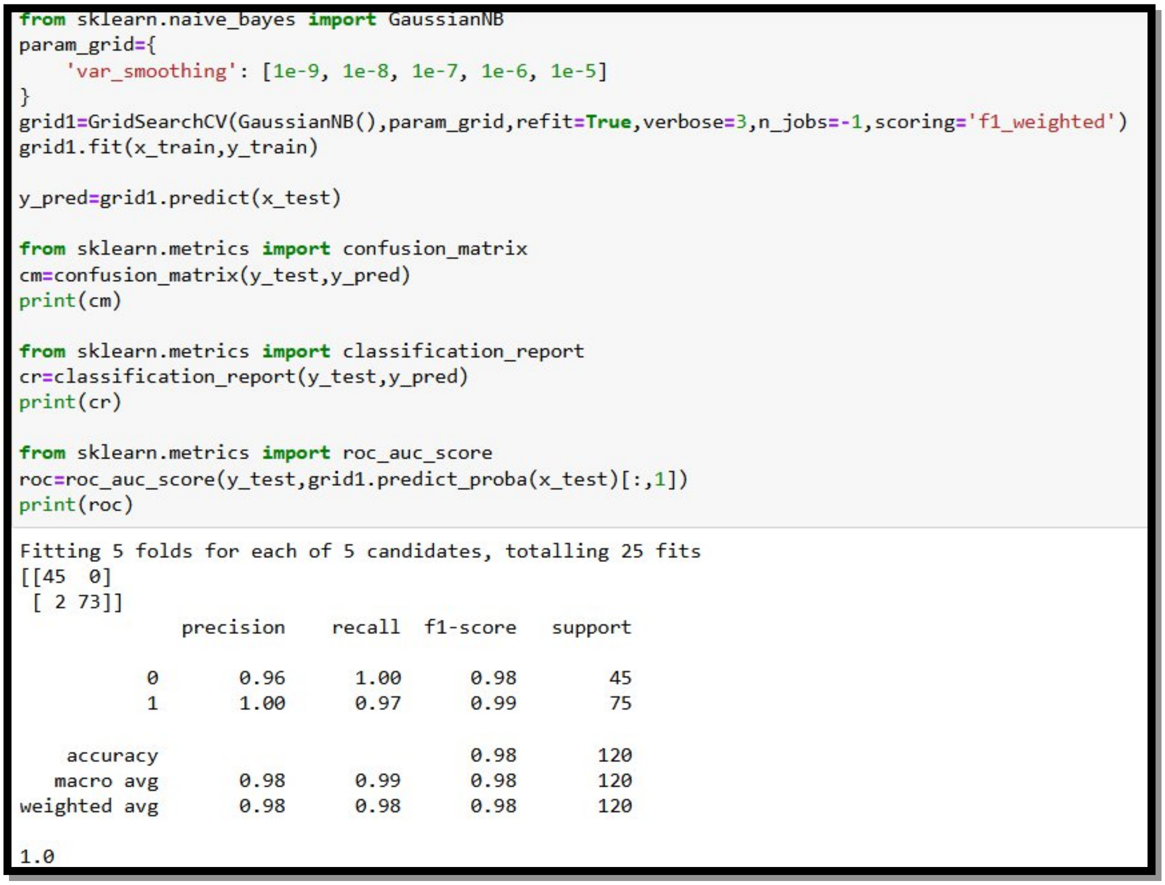
```
In [17]:  from sklearn.metrics import confusion_matrix
          cm=confusion_matrix(y_test,y_pred)
          print(cm)

          [[44  1]
           [ 2 73]]

In [18]:  from sklearn.metrics import classification_report
          cr=classification_report(y_test,y_pred)
          print(cr)

                        precision    recall  f1-score   support

                     0       0.96      0.98      0.97        45
                     1       0.99      0.97      0.98        75

              accuracy                           0.97       120
             macro avg       0.97      0.98      0.97       120
          weighted avg       0.98      0.97      0.98       120


In [19]:  from sklearn.metrics import roc_auc_score
          roc=roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
          print(roc)

          0.9989629629629629
```

e) KNN



```
In [16]:   from sklearn.metrics import confusion_matrix
           cm=confusion_matrix(y_test,y_pred)
           print(cm)

           [[42  3]
            [23 52]]

In [17]:   from sklearn.metrics import classification_report
           cr=classification_report(y_test,y_pred)
           print(cr)

                         precision    recall  f1-score   support

                      0       0.65      0.93      0.76        45
                      1       0.95      0.69      0.80        75

               accuracy                           0.78       120
              macro avg       0.80      0.81      0.78       120
           weighted avg       0.83      0.78      0.79       120


In [18]:   from sklearn.metrics import roc_auc_score
           roc=roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
           print(roc)

           0.8645925925925926
```

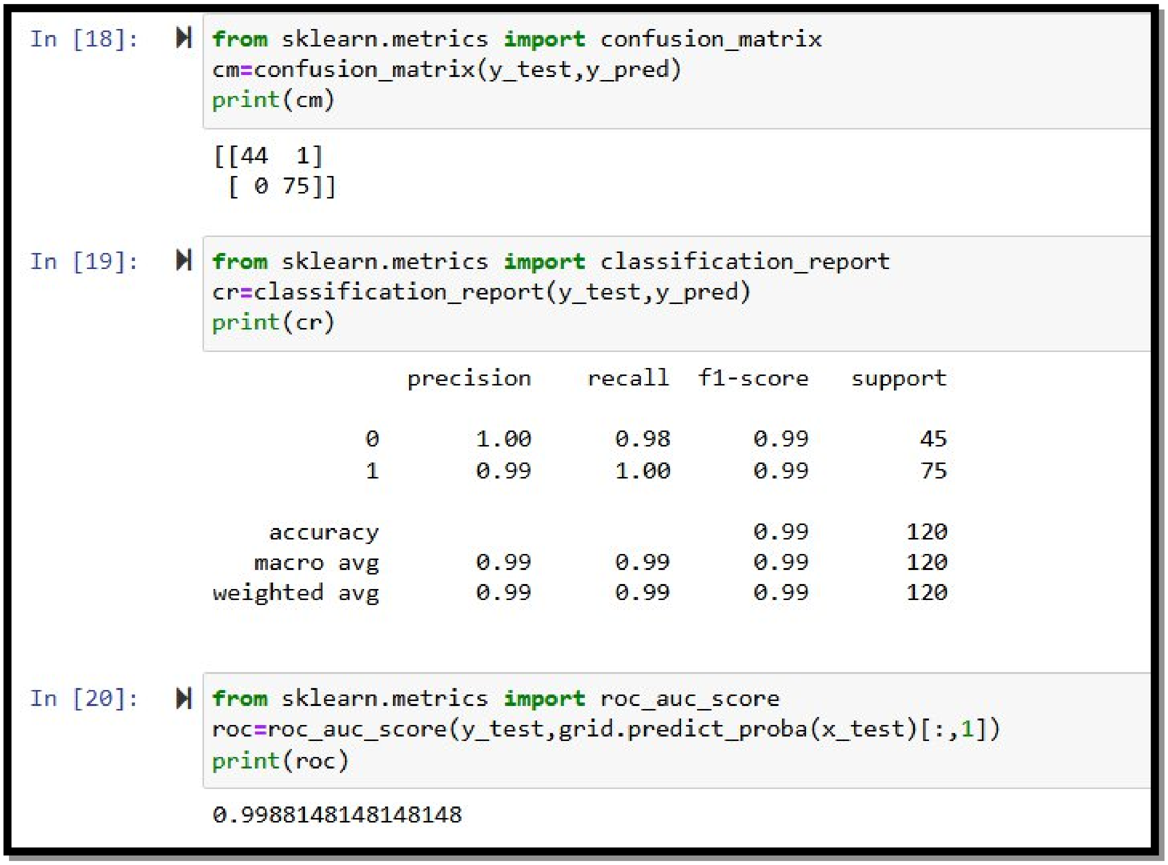
f) **NaïveBayes**



```
from sklearn.naive_bayes import GaussianNB
param_grid={
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
}
grid1=GridSearchCV(GaussianNB(),param_grid,refit=True,verbose=3,n_jobs=-1,scoring='f1_weighted')
grid1.fit(x_train,y_train)

y_pred=grid1.predict(x_test)

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)

from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
print(cr)

from sklearn.metrics import roc_auc_score
roc=roc_auc_score(y_test,grid1.predict_proba(x_test)[:,1])
print(roc)

Fitting 5 folds for each of 5 candidates, totalling 25 fits
[[45  0]
 [ 2 73]]
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        45
           1       1.00      0.97      0.99        75

    accuracy                           0.98       120
   macro avg       0.98      0.99      0.98       120
weighted avg       0.98      0.98      0.98       120

1.0
```

- **Mention your final model, justify why u have chosen the same.**

The final chosen model would be SVC with ACCURACY = 0.99% and ROC=0.998%



```
In [18]:  from sklearn.metrics import confusion_matrix
          cm=confusion_matrix(y_test,y_pred)
          print(cm)

          [[44  1]
           [ 0 75]]

In [19]:  from sklearn.metrics import classification_report
          cr=classification_report(y_test,y_pred)
          print(cr)

                        precision    recall  f1-score   support

                     0       1.00      0.98      0.99        45
                     1       0.99      1.00      0.99        75

              accuracy                           0.99       120
             macro avg       0.99      0.99      0.99       120
          weighted avg       0.99      0.99      0.99       120


In [20]:  from sklearn.metrics import roc_auc_score
          roc=roc_auc_score(y_test,grid.predict_proba(x_test)[:,1])
          print(roc)

          0.9988148148148148
```