In [1]:

```
import pandas as pd
```

In [2]:

```
df = pd.read_csv("bank.csv")
df
```

Out[2]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
0	59	0	1	1	0	2343	1	0	2	5	8	1042	1	-1	0	3	1
1	56	0	1	1	0	45	0	0	2	5	8	1467	1	-1	0	3	1
2	41	9	1	1	0	1270	1	0	2	5	8	1389	1	-1	0	3	1
3	55	7	1	1	0	2476	1	0	2	5	8	579	1	-1	0	3	1
4	54	0	1	2	0	184	0	0	2	5	8	673	2	-1	0	3	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11157	33	1	2	0	0	1	1	0	0	20	0	257	1	-1	0	3	0
11158	39	7	1	1	0	733	0	0	2	16	6	83	4	-1	0	3	0
11159	32	9	2	1	0	29	0	0	0	19	1	156	2	-1	0	3	0
11160	43	9	1	1	0	0	0	1	0	8	8	9	2	172	5	0	0
11161	34	9	1	1	0	0	0	0	0	9	5	628	1	-1	0	3	0

11162 rows × 17 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11162 entries, 0 to 11161
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   age        11162 non-null  int64
 1   job        11162 non-null  int64
 2   marital    11162 non-null  int64
 3   education  11162 non-null  int64
 4   default    11162 non-null  int64
 5   balance    11162 non-null  int64
 6   housing    11162 non-null  int64
 7   loan       11162 non-null  int64
 8   contact    11162 non-null  int64
 9   day        11162 non-null  int64
 10  month      11162 non-null  int64
 11  duration   11162 non-null  int64
 12  campaign   11162 non-null  int64
 13  pdays      11162 non-null  int64
 14  previous   11162 non-null  int64
 15  poutcome   11162 non-null  int64
 16  deposit    11162 non-null  int64
dtypes: int64(17)
memory usage: 1.4 MB
```

In [4]:

```
#EDA
```

In [5]:

```
x = df.iloc[:,:-1]

y = df.iloc[:,-1]
```

In [6]:

```
x
```

Out[6]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
0	59	0	1	1	0	2343	1	0	2	5	8	1042	1	-1	0	3
1	56	0	1	1	0	45	0	0	2	5	8	1467	1	-1	0	3
2	41	9	1	1	0	1270	1	0	2	5	8	1389	1	-1	0	3
3	55	7	1	1	0	2476	1	0	2	5	8	579	1	-1	0	3
4	54	0	1	2	0	184	0	0	2	5	8	673	2	-1	0	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11157	33	1	2	0	0	1	1	0	0	20	0	257	1	-1	0	3
11158	39	7	1	1	0	733	0	0	2	16	6	83	4	-1	0	3
11159	32	9	2	1	0	29	0	0	0	19	1	156	2	-1	0	3
11160	43	9	1	1	0	0	0	1	0	8	8	9	2	172	5	0
11161	34	9	1	1	0	0	0	0	0	9	5	628	1	-1	0	3

11162 rows × 16 columns

In [7]:

```
y
```

Out[7]:

```
0        1
1        1
2        1
3        1
4        1
       ..
11157    0
11158    0
11159    0
11160    0
11161    0
Name: deposit, Length: 11162, dtype: int64
```

In [8]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.3,random_state=1)
```

In [9]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

In [10]:

```
from sklearn.metrics import classification_report, accuracy_score
```

In [11]:

```
def mymodel(model):

    #Model Creation
    model.fit(xtrain,ytrain)
    ypred = model.predict(xtest)

    #Checkeing Bais and Variance

    train = model.score(xtrain,ytrain)
    test = model.score(xtest,ytest)

    print(f"Training Acuuracy:- {train}\n Testing Accuracy:- {test}")

    #Model Evaluation

    print(classification_report(ytest,ypred))
    return model
```

In [12]:

```
knn = mymodel(KNeighborsClassifier())
```

```
Training Acuuracy:- 0.81889159093818
 Testing Accuracy:- 0.7539564048969841
              precision    recall  f1-score   support

           0       0.76      0.78      0.77      1760
           1       0.75      0.72      0.74      1589

    accuracy                           0.75      3349
   macro avg       0.75      0.75      0.75      3349
weighted avg       0.75      0.75      0.75      3349
```

In [13]:

```
logreg = mymodel(LogisticRegression())
```

```
Training Acuuracy:- 0.755535645718674
 Testing Accuracy:- 0.7659002687369364
              precision    recall  f1-score   support

           0       0.77      0.80      0.78      1760
           1       0.76      0.73      0.75      1589

    accuracy                           0.77      3349
   macro avg       0.77      0.76      0.76      3349
weighted avg       0.77      0.77      0.77      3349


C:\Users\hp\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/
linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

In [14]:

```
svm = mymodel(SVC())
```

```
Training Acuuracy:- 0.7382567515678996
 Testing Accuracy:- 0.7414153478650344
              precision    recall  f1-score   support

           0       0.72      0.84      0.77      1760
           1       0.78      0.63      0.70      1589

    accuracy                           0.74      3349
   macro avg       0.75      0.74      0.74      3349
weighted avg       0.75      0.74      0.74      3349
```

In [15]:

```
dt = mymodel(DecisionTreeClassifier())
```

```
Training Acuuracy:- 1.0
 Testing Accuracy:- 0.7868020304568528
              precision    recall  f1-score   support

           0       0.80      0.80      0.80      1760
           1       0.78      0.77      0.77      1589

    accuracy                           0.79      3349
   macro avg       0.79      0.79      0.79      3349
weighted avg       0.79      0.79      0.79      3349
```

In [16]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
xtrain = sc.fit_transform(xtrain)
xtest = sc.transform(xtest)
```

In [17]:

```
knn = mymodel(KNeighborsClassifier())
```

```
Training Acuuracy:- 0.8466658133879432
 Testing Accuracy:- 0.7748581666169005
              precision    recall  f1-score   support

           0       0.77      0.81      0.79      1760
           1       0.78      0.73      0.76      1589

    accuracy                           0.77      3349
   macro avg       0.78      0.77      0.77      3349
weighted avg       0.78      0.77      0.77      3349
```

In [18]:

```
logreg = mymodel(LogisticRegression())
```

```
Training Acuuracy:- 0.7973889671061052
 Testing Accuracy:- 0.7975515079128098
              precision    recall  f1-score   support

           0       0.80      0.82      0.81      1760
           1       0.79      0.77      0.78      1589

    accuracy                           0.80      3349
   macro avg       0.80      0.80      0.80      3349
weighted avg       0.80      0.80      0.80      3349
```

In [19]:

```
svm = mymodel(SVC())
```

```
Training Acuuracy:- 0.8447459362600793
 Testing Accuracy:- 0.8163630934607345
              precision    recall  f1-score   support

           0       0.84      0.80      0.82      1760
           1       0.79      0.84      0.81      1589

    accuracy                           0.82      3349
   macro avg       0.82      0.82      0.82      3349
weighted avg       0.82      0.82      0.82      3349
```

In [20]:

```
dt = mymodel(DecisionTreeClassifier())
```

```
Training Acuuracy:- 1.0
 Testing Accuracy:- 0.792773962376829
              precision    recall  f1-score   support

           0       0.80      0.81      0.80      1760
           1       0.78      0.78      0.78      1589

    accuracy                           0.79      3349
   macro avg       0.79      0.79      0.79      3349
weighted avg       0.79      0.79      0.79      3349
```

Hyperparameter Tuning

In [21]:

```
dt1 = mymodel(DecisionTreeClassifier(max_depth=10))
```

```
Training Acuuracy:- 0.899910405734033
 Testing Accuracy:- 0.8163630934607345
              precision    recall  f1-score   support

           0       0.82      0.83      0.83      1760
           1       0.81      0.81      0.81      1589

    accuracy                           0.82      3349
   macro avg       0.82      0.82      0.82      3349
weighted avg       0.82      0.82      0.82      3349
```

In [22]:

```
for i in range(1,50):
    dt = DecisionTreeClassifier(max_depth=i)
    dt.fit(xtrain,ytrain)
    train =dt.score(xtrain,ytrain)
    test =dt.score(xtest,ytest)

    print(f"{i}  {train}    {test}")
```

```
1  0.7108665045437093    0.7121528814571514
2  0.7108665045437093    0.7121528814571514
3  0.7757583514655062    0.771275007464915
4  0.7985408933828235    0.78501045088086
5  0.8141558940227825    0.7990444908928038
6  0.8273390503007807    0.8011346670647954
7  0.8446179444515551    0.8160644968647357
8  0.8627927812619992    0.8193490594207226
9  0.8758479457314732    0.8083009853687668
10  0.8993984384999361    0.8169602866527321
11  0.9124536029694099    0.8199462526127202
12  0.9313963906309997    0.8190504628247238
13  0.9470113912709587    0.808002388772768
14  0.9609624984001024    0.8002388772767991
15  0.9724817611672852    0.7975515079128098
16  0.9800332778702163    0.797252911316811
17  0.985920901062332    0.7894893998208421
18  0.9907845897862537    0.792773962376829
19  0.9943683604249328    0.7859062406688564
20  0.9968002047868937    0.787996416840848
21  0.9983361064891847    0.786503433860854
22  0.9988480737232817    0.7882950134368468
23  0.999488032765903    0.7841146610928635
24  0.9998720081914757    0.7900865930128397
25  1.0    0.7885936100328457
26  1.0    0.7891908032248433
27  1.0    0.7844132576888624
28  1.0    0.7924753657808301
29  1.0    0.7856076440728575
30  1.0    0.7873992236488504
31  1.0    0.7856076440728575
32  1.0    0.7823230815168707
33  1.0    0.7891908032248433
34  1.0    0.7859062406688564
35  1.0    0.7930725589728277
36  1.0    0.7900865930128397
37  1.0    0.787996416840848
38  1.0    0.78501045088086
39  1.0    0.7799343087488803
40  1.0    0.7853090474768588
41  1.0    0.7897879964168408
42  1.0    0.7891908032248433
43  1.0    0.7868020304568528
44  1.0    0.7882950134368468
45  1.0    0.7838160644968647
46  1.0    0.7868020304568528
47  1.0    0.7862048372648551
48  1.0    0.7885936100328457
49  1.0    0.7856076440728575
```

In [23]:

```
dt2 = mymodel(DecisionTreeClassifier(max_depth=7))
```

```
Training Acuuracy:- 0.8446179444515551
 Testing Accuracy:- 0.8151687070767393
              precision    recall  f1-score   support

           0       0.82      0.83      0.82      1760
           1       0.81      0.80      0.80      1589

    accuracy                           0.82      3349
   macro avg       0.81      0.81      0.81      3349
weighted avg       0.82      0.82      0.82      3349
```

In [24]:

```
dt3 = mymodel(DecisionTreeClassifier(min_samples_leaf=10))
```

```
Training Acuuracy:- 0.8813515934980162
 Testing Accuracy:- 0.8133771275007465
              precision    recall  f1-score   support

           0       0.82      0.83      0.82      1760
           1       0.81      0.80      0.80      1589

    accuracy                           0.81      3349
   macro avg       0.81      0.81      0.81      3349
weighted avg       0.81      0.81      0.81      3349
```

In [25]:

```
for i in range(1,75):
    dt = DecisionTreeClassifier(min_samples_leaf=i)
    dt.fit(xtrain,ytrain)
    train =dt.score(xtrain,ytrain)
    test =dt.score(xtest,ytest)

    print(f"{i}  {train}    {test}")
```

```
1  1.0    0.7862048372648551
2  0.9618584410597721    0.7730665870409077
3  0.9449635223345706    0.7868020304568528
4  0.9273006527582235    0.7897879964168408
5  0.9141174964802252    0.7891908032248433
6  0.9041341354153334    0.7951627351448194
7  0.898374504031742    0.8068080023887728
8  0.8905670037117625    0.8133771275007465
9  0.885831306796365    0.8115855479247537
10  0.8813515934980162    0.814272917288743
11  0.8767438883911429    0.8127799343087488
12  0.8744400358377064    0.815765900268737
13  0.8708562651990273    0.8127799343087488
14  0.8689363880711635    0.8130785309047477
15  0.8680404454114936    0.815765900268737
16  0.8653526174324844    0.8178560764407286
17  0.864200691155766    0.8232308151687071
18  0.8629207730705235    0.8199462526127202
19  0.8613848713682324    0.8220364287847118
20  0.8606169205170869    0.8286055538966856
21  0.8592090106233201    0.8268139743206927
22  0.8565211826443108    0.8277097641086891
23  0.8551132727505439    0.829501343684682
24  0.8537053628567771    0.828008360704688
25  0.8534493792397286    0.8289041504926844
26  0.8520414693459618    0.8262167811286951
27  0.8524254447715346    0.8262167811286951
28  0.850121592218098    0.8256195879366975
29  0.849097657749904    0.8274111675126904
30  0.8482017150902342    0.8238280083607047
31  0.847561756047613    0.8262167811286951
32  0.847561756047613    0.8262167811286951
33  0.847561756047613    0.8250223947447
34  0.8471777806220402    0.826515377724694
35  0.8471777806220402    0.8253209913406987
36  0.8471777806220402    0.8253209913406987
37  0.8471777806220402    0.8259181845326963
38  0.8467938051964674    0.8256195879366975
39  0.8453858953027006    0.8283069573006868
40  0.8452579034941764    0.828008360704688
41  0.8446179444515551    0.8292027470886831
42  0.843082042749264    0.8286055538966856
43  0.8416741328554973    0.8289041504926844
44  0.8411621656214002    0.8253209913406987
45  0.8411621656214002    0.8253209913406987
46  0.8411621656214002    0.8253209913406987
47  0.8411621656214002    0.8253209913406987
48  0.8410341738128759    0.8253209913406987
49  0.8409061820043517    0.8253209913406987
50  0.8407781901958274    0.8250223947447
51  0.840522206578779    0.8277097641086891
52  0.840522206578779    0.8277097641086891
53  0.8377063867912453    0.8247237981487011
54  0.837066427748624    0.8235294117647058
55  0.8366824523230513    0.8244252015527023
56  0.8366824523230513    0.8244252015527023
57  0.8364264687060028    0.8238280083607047
58  0.8352745424292846    0.8214392355927143
59  0.8352745424292846    0.8214392355927143
60  0.8352745424292846    0.8214392355927143
61  0.8352745424292846    0.8214392355927143
62  0.8352745424292846    0.8214392355927143
63  0.8352745424292846    0.8214392355927143
64  0.8352745424292846    0.8214392355927143
65  0.8347625751951875    0.8217378321887131
66  0.8306668373224113    0.8160644968647357
67  0.8306668373224113    0.8163630934607345
68  0.8323307308332267    0.8196476560167214
69  0.8314347881735569    0.8184532696327261
70  0.8314347881735569    0.8184532696327261
71  0.8314347881735569    0.8175574798447298
72  0.8314347881735569    0.8175574798447298
73  0.8313067963650327    0.8178560764407286
74  0.8311788045565084    0.8178560764407286
```

In [26]:

```
dt4 = mymodel(DecisionTreeClassifier(min_samples_leaf=57))
```

```
Training Acuuracy:- 0.8364264687060028
 Testing Accuracy:- 0.8238280083607047
              precision    recall  f1-score   support

           0       0.85      0.80      0.83      1760
           1       0.79      0.85      0.82      1589

    accuracy                           0.82      3349
   macro avg       0.82      0.82      0.82      3349
weighted avg       0.83      0.82      0.82      3349
```

In [27]:

```
dt5= mymodel(DecisionTreeClassifier(max_depth=7,min_samples_leaf=57))
```

```
Training Acuuracy:- 0.8220913861512863
 Testing Accuracy:- 0.8121827411167513
              precision    recall  f1-score   support

           0       0.82      0.82      0.82      1760
           1       0.80      0.81      0.80      1589

    accuracy                           0.81      3349
   macro avg       0.81      0.81      0.81      3349
weighted avg       0.81      0.81      0.81      3349
```

Entropy

In [28]:

```
dt6 = mymodel(DecisionTreeClassifier(criterion="entropy",max_depth=7,min_samples_leaf=57))
```

```
Training Acuuracy:- 0.8159477793421221
 Testing Accuracy:- 0.8088981785607644
              precision    recall  f1-score   support

           0       0.81      0.83      0.82      1760
           1       0.81      0.78      0.80      1589

    accuracy                           0.81      3349
   macro avg       0.81      0.81      0.81      3349
weighted avg       0.81      0.81      0.81      3349
```

In [29]:

```
for i in range(1,100):
    dt = DecisionTreeClassifier(criterion="entropy",min_samples_leaf=i)
    dt.fit(xtrain,ytrain)
    train =dt.score(xtrain,ytrain)
    test =dt.score(xtest,ytest)

    print(f"{i}  {train}    {test}")
```

```
1  1.0    0.7823230815168707
2  0.9705618840394215    0.7793371155568827
3  0.9517470881863561    0.7891908032248433
4  0.9328043005247664    0.794266945356823
5  0.921029054140535    0.7933711555688265
6  0.9091258159477793    0.7930725589728277
7  0.9027262255215667    0.8091967751567632
8  0.8931268398822475    0.8085995819647656
9  0.8850633559452195    0.8083009853687668
10  0.8816075771150647    0.806509405792774
11  0.8778958146678613    0.8118841445207524
12  0.8749520030718034    0.8151687070767393
13  0.870728273390503    0.8232308151687071
14  0.8670165109432997    0.8211406389967154
15  0.8639447075387175    0.8259181845326963
16  0.8618968386023295    0.818751866228725
17  0.8601049532829899    0.8196476560167214
18  0.8585690515806988    0.820244849208719
19  0.8583130679636504    0.8232308151687071
20  0.8576731089210291    0.8241266049567035
21  0.855625239984641    0.8238280083607047
22  0.8542173300908742    0.818751866228725
23  0.8529374120056317    0.817258883248731
24  0.8503775758351465    0.8175574798447298
25  0.8492256495584283    0.8175574798447298
26  0.8480737232817099    0.8193490594207226
27  0.8476897478561372    0.8154673036727381
28  0.8474337642390887    0.815765900268737
29  0.8461538461538461    0.8181546730367274
30  0.8457698707282734    0.8169602866527321
31  0.8460258543453218    0.8169602866527321
32  0.8452579034941764    0.8193490594207226
33  0.8452579034941764    0.8193490594207226
34  0.8450019198771279    0.8208420424007167
35  0.8450019198771279    0.8208420424007167
36  0.8444899526430308    0.8196476560167214
37  0.843082042749264    0.8235294117647058
38  0.843082042749264    0.8238280083607047
39  0.8426980673236912    0.8229322185727083
40  0.8424420837066428    0.8229322185727083
41  0.8441059772174581    0.8229322185727083
42  0.8441059772174581    0.8235294117647058
43  0.8441059772174581    0.8244252015527023
44  0.8403942147702547    0.8175574798447298
45  0.8402662229617305    0.8193490594207226
46  0.8402662229617305    0.8193490594207226
47  0.8402662229617305    0.8217378321887131
48  0.8402662229617305    0.8217378321887131
49  0.8378343785997696    0.8232308151687071
50  0.8371944195571484    0.8223350253807107
51  0.8371944195571484    0.8208420424007167
52  0.8371944195571484    0.8208420424007167
53  0.837066427748624    0.8175574798447298
54  0.837066427748624    0.817258883248731
55  0.8368104441315756    0.8169602866527321
56  0.83604249328043    0.8181546730367274
57  0.83604249328043    0.8178560764407286
58  0.83604249328043    0.8181546730367274
59  0.8359145014719058    0.8184532696327261
60  0.83604249328043    0.8184532696327261
61  0.835530526046333    0.8160644968647357
62  0.8351465506207603    0.8211406389967154
63  0.8351465506207603    0.8208420424007167
64  0.834506591578139    0.8199462526127202
65  0.8337386407269934    0.8175574798447298
66  0.8337386407269934    0.8178560764407286
67  0.8328426980673237    0.8166616900567334
68  0.8324587226417509    0.8175574798447298
69  0.8324587226417509    0.8178560764407286
70  0.8324587226417509    0.8175574798447298
71  0.8324587226417509    0.8178560764407286
72  0.8270830666837322    0.8139743206927441
73  0.8270830666837322    0.8139743206927441
74  0.826955074875208    0.814272917288743
75  0.8268270830666837    0.814272917288743
76  0.8266990912581594    0.814272917288743
77  0.8265710994496352    0.814272917288743
78  0.8265710994496352    0.8136757240967453
79  0.8265710994496352    0.8136757240967453
80  0.825931140407014    0.814272917288743
81  0.825931140407014    0.8139743206927441
82  0.825931140407014    0.8139743206927441
83  0.8258031485984897    0.814272917288743
84  0.8256751567899654    0.8139743206927441
85  0.8255471649814412    0.8139743206927441
86  0.8254191731729169    0.8139743206927441
87  0.8252911813643927    0.8136757240967453
88  0.8251631895558684    0.8136757240967453
89  0.8213234353001407    0.8053150194087787
90  0.8211954434916166    0.8053150194087787
91  0.8210674516830923    0.8053150194087787
```

```
92  0.8206834762575195    0.8053150194087787
93  0.8206834762575195    0.8053150194087787
94  0.8206834762575195    0.8056136160047775
95  0.8206834762575195    0.8053150194087787
96  0.8206834762575195    0.8056136160047775
97  0.8197875335978497    0.8047178262167811
98  0.8197875335978497    0.80501642281278
99  0.8197875335978497    0.80501642281278
```

In [30]:

```
dt7 = mymodel(DecisionTreeClassifier(criterion="entropy",min_samples_leaf=52))
```

```
Training Acuuracy:- 0.8371944195571484
 Testing Accuracy:- 0.8208420424007167
              precision    recall  f1-score   support

           0       0.83      0.82      0.83      1760
           1       0.81      0.82      0.81      1589

    accuracy                           0.82      3349
   macro avg       0.82      0.82      0.82      3349
weighted avg       0.82      0.82      0.82      3349
```

In [31]:

```
dt8 = mymodel(DecisionTreeClassifier(criterion="entropy",max_depth=7))
```

```
Training Acuuracy:- 0.8282349929604506
 Testing Accuracy:- 0.806509405792774
              precision    recall  f1-score   support

           0       0.81      0.83      0.82      1760
           1       0.81      0.78      0.79      1589

    accuracy                           0.81      3349
   macro avg       0.81      0.81      0.81      3349
weighted avg       0.81      0.81      0.81      3349
```

In [32]:

```
for i in range(1,100):
    dt = DecisionTreeClassifier(criterion="entropy",max_depth=i)
    dt.fit(xtrain,ytrain)
    train =dt.score(xtrain,ytrain)
    test =dt.score(xtest,ytest)

    print(f"{i}  {train}    {test}")
```

```
1  0.7108665045437093    0.7121528814571514
2  0.7108665045437093    0.7121528814571514
3  0.757071547420965    0.7584353538369663
4  0.7849737616792525    0.774260973424903
5  0.7940611800844746    0.7784413257688862
6  0.8103161397670549    0.792773962376829
7  0.8282349929604506    0.8056136160047775
8  0.8476897478561372    0.808002388772768
9  0.8652246256239601    0.8154673036727381
10  0.8795597081786766    0.815765900268737
11  0.8961986432868296    0.8083009853687668
12  0.911941635735313    0.8041206330247835
13  0.928580570843466    0.8029262466407883
14  0.9412517598873672    0.7951627351448194
15  0.9545629079738897    0.7999402806808003
16  0.9633943427620633    0.7918781725888325
17  0.971841802124664    0.7924753657808301
18  0.9793933188275951    0.7868020304568528
19  0.9845129911685652    0.7906837862048373
20  0.9884807372328172    0.7876978202448492
21  0.9896326635095354    0.782024484920872
22  0.9921924996800204    0.786503433860854
23  0.9934724177652631    0.7856076440728575
24  0.99398438499936    0.7859062406688564
25  0.9947523358505056    0.7847118542848611
26  0.9953922948931269    0.783517467900866
27  0.9971841802124664    0.7847118542848611
28  0.9975681556380391    0.7838160644968647
29  0.9980801228721362    0.783517467900866
30  0.998464098297709    0.7859062406688564
31  0.9992320491488544    0.7844132576888624
32  0.9996160245744272    0.7841146610928635
33  0.9998720081914757    0.7862048372648551
34  1.0    0.7769483427888922
35  1.0    0.7826216781128695
36  0.9998720081914757    0.7760525530008958
37  1.0    0.7891908032248433
38  1.0    0.7853090474768588
39  1.0    0.782024484920872
40  1.0    0.7894893998208421
41  1.0    0.7814272917288743
42  1.0    0.7862048372648551
43  1.0    0.7847118542848611
44  1.0    0.78501045088086
45  1.0    0.7805315019408778
46  1.0    0.786503433860854
47  1.0    0.7805315019408778
48  1.0    0.7856076440728575
49  1.0    0.7844132576888624
50  1.0    0.7826216781128695
51  1.0    0.7862048372648551
52  1.0    0.7873992236488504
53  1.0    0.7856076440728575
54  1.0    0.78501045088086
55  1.0    0.7817258883248731
56  1.0    0.7847118542848611
57  1.0    0.7823230815168707
58  1.0    0.7826216781128695
59  1.0    0.7847118542848611
60  1.0    0.7868020304568528
61  1.0    0.7823230815168707
62  1.0    0.7868020304568528
63  1.0    0.786503433860854
64  1.0    0.7817258883248731
65  1.0    0.7826216781128695
66  1.0    0.7808300985368767
67  1.0    0.7814272917288743
68  1.0    0.7808300985368767
69  1.0    0.7856076440728575
70  1.0    0.7876978202448492
71  1.0    0.7862048372648551
72  1.0    0.782024484920872
73  1.0    0.783517467900866
74  1.0    0.7844132576888624
75  1.0    0.783517467900866
76  1.0    0.7799343087488803
77  1.0    0.7811286951328755
78  1.0    0.7859062406688564
79  1.0    0.7897879964168408
80  1.0    0.7814272917288743
81  1.0    0.783517467900866
82  1.0    0.7823230815168707
83  1.0    0.7868020304568528
84  1.0    0.7859062406688564
85  1.0    0.7868020304568528
86  1.0    0.7826216781128695
87  1.0    0.7862048372648551
88  1.0    0.7793371155568827
89  1.0    0.7796357121528814
90  1.0    0.7817258883248731
91  1.0    0.7862048372648551
```

```
92  1.0    0.7841146610928635
93  1.0    0.7814272917288743
94  1.0    0.7859062406688564
95  1.0    0.7808300985368767
96  1.0    0.7853090474768588
97  1.0    0.7832188713048671
98  1.0    0.7894893998208421
99  1.0    0.7844132576888624
```

In [33]:

```
dt8 = mymodel(DecisionTreeClassifier(criterion="entropy",max_depth=7))
```

```
Training Acuuracy:- 0.8282349929604506
 Testing Accuracy:- 0.8068080023887728
              precision    recall  f1-score   support

           0       0.81      0.83      0.82      1760
           1       0.81      0.78      0.79      1589

    accuracy                           0.81      3349
   macro avg       0.81      0.81      0.81      3349
weighted avg       0.81      0.81      0.81      3349
```

In [34]:

```
dt9 = mymodel(DecisionTreeClassifier(criterion="entropy",max_depth=7,min_samples_leaf=52))
```

```
Training Acuuracy:- 0.8167157301932676
 Testing Accuracy:- 0.808002388772768
              precision    recall  f1-score   support

           0       0.81      0.83      0.82      1760
           1       0.81      0.78      0.79      1589

    accuracy                           0.81      3349
   macro avg       0.81      0.81      0.81      3349
weighted avg       0.81      0.81      0.81      3349
```

Final Result

In [35]:

```
dt10= mymodel(DecisionTreeClassifier(max_depth=7,min_samples_leaf=57))
```

```
Training Acuuracy:- 0.8220913861512863
 Testing Accuracy:- 0.8121827411167513
              precision    recall  f1-score   support

           0       0.82      0.82      0.82      1760
           1       0.80      0.81      0.80      1589

    accuracy                           0.81      3349
   macro avg       0.81      0.81      0.81      3349
weighted avg       0.81      0.81      0.81      3349
```

In [36]:

```
# gini

from sklearn.model_selection import cross_val_score

cvs = cross_val_score(dt10,x,y,cv=5,scoring="accuracy")

print(f"Avg Accuracy :- {cvs.mean()}\n STD :- {cvs.std()}")
```

```
Avg Accuracy :- 0.7855257645580226
 STD :- 0.035880717688939065
```

In [37]:

```
# entropy

from sklearn.model_selection import cross_val_score

cvs = cross_val_score(dt9,x,y,cv=5,scoring="accuracy")

print(f"Avg Accuracy :- {cvs.mean()}\n STD :- {cvs.std()}")
```

```
Avg Accuracy :- 0.7812244485214451
 STD :- 0.034056608505377076
```

GridSearchCV

In [38]:

```
list(range(1,50,5))
```

Out[38]:

```
[1, 6, 11, 16, 21, 26, 31, 36, 41, 46]
```

In [39]:

```
parameter = {

            "criterion":["gini","entropy"],
            "max_depth":list(range(1,50,5)),
            "min_samples_leaf":list(range(1,50,5))

}
```

In [40]:

```
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(DecisionTreeClassifier(),parameter,verbose=2)
grid.fit(xtrain,ytrain)
```

```
[CV] END ...criterion=gini, max_depth=11, min_samples_leaf=1; total time=   0.0s
[CV] END ...criterion=gini, max_depth=11, min_samples_leaf=1; total time=   0.0s

[CV] END ...criterion=gini, max_depth=11, min_samples_leaf=1; total time=   0.0s
[CV] END ...criterion=gini, max_depth=11, min_samples_leaf=6; total time=   0.0s
[CV] END ...criterion=gini, max_depth=11, min_samples_leaf=6; total time=   0.0s
[CV] END ...criterion=gini, max_depth=11, min_samples_leaf=6; total time=   0.0s
[CV] END ...criterion=gini, max_depth=11, min_samples_leaf=6; total time=   0.0s
[CV] END ...criterion=gini, max_depth=11, min_samples_leaf=6; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=11; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=11; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=11; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=11; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=16; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=16; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=16; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=16; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=16; total time=   0.0s
[CV] END ..criterion=gini, max_depth=11, min_samples_leaf=21; total time=   0.0s
```

In [41]:

```
grid.best_params_
```

Out[41]:

```
{'criterion': 'gini', 'max_depth': 41, 'min_samples_leaf': 31}
```

In [42]:

```
grid.best_score_
```

Out[42]:

```
0.8261899086018467
```

In [43]:

```
grid.best_estimator_
```

Out[43]:

```
DecisionTreeClassifier(max_depth=41, min_samples_leaf=31)
```

In [44]:

```
dt= mymodel(grid.best_estimator_)
```

```
Training Acuuracy:- 0.847561756047613
 Testing Accuracy:- 0.8262167811286951
              precision    recall  f1-score   support

           0       0.85      0.81      0.83      1760
           1       0.80      0.85      0.82      1589

    accuracy                           0.83      3349
   macro avg       0.83      0.83      0.83      3349
weighted avg       0.83      0.83      0.83      3349
```

In [45]:

```
from sklearn.ensemble import BaggingClassifier
bg=BaggingClassifier(DecisionTreeClassifier(),n_estimators=10)
bg.fit(xtrain,ytrain)
ypred=bg.predict(xtest)

train=bg.score(xtrain,ytrain)
test=bg.score(xtest,ytest)

print(f"Training Accuracy:{train}\n Testing Accuracy:{test}")
print(classification_report(ytest,ypred))
```

```
Training Accuracy:0.9916805324459235
 Testing Accuracy:0.8297999402806808
              precision    recall  f1-score   support

           0       0.84      0.83      0.84      1760
           1       0.82      0.82      0.82      1589

    accuracy                           0.83      3349
   macro avg       0.83      0.83      0.83      3349
weighted avg       0.83      0.83      0.83      3349
```

In [46]:

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(xtrain,ytrain)
ypred=rf.predict(xtest)

train=rf.score(xtrain,ytrain)
test=rf.score(xtest,ytest)

print(f"Training Accuracy:{train}\n Testing Accuracy:{test}")
print(classification_report(ytest,ypred))
```

```
Training Accuracy:1.0
 Testing Accuracy:0.8423409973126307
              precision    recall  f1-score   support

           0       0.87      0.82      0.85      1760
           1       0.81      0.87      0.84      1589

    accuracy                           0.84      3349
   macro avg       0.84      0.84      0.84      3349
weighted avg       0.84      0.84      0.84      3349
```

In [47]:

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=100,criterion="entropy",min_samples_leaf=1,max_depth=21)
rf.fit(xtrain,ytrain)
ypred=rf.predict(xtest)

train=rf.score(xtrain,ytrain)
test=rf.score(xtest,ytest)

print(f"Training Accuracy:{train}\n Testing Accuracy:{test}")
print(classification_report(ytest,ypred))
```

```
Training Accuracy:0.9998720081914757
 Testing Accuracy:0.8453269632726187
              precision    recall  f1-score   support

           0       0.88      0.82      0.85      1760
           1       0.81      0.88      0.84      1589

    accuracy                           0.85      3349
   macro avg       0.85      0.85      0.85      3349
weighted avg       0.85      0.85      0.85      3349
```

In [48]:

```
list(range(1,50,5))
parameter = {

            "criterion":["gini","entropy"],
            "max_depth":list(range(1,50,5)),
            "min_samples_leaf":list(range(1,50,5))

}
```

In [49]:

```
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(RandomForestClassifier(),parameter,verbose=2)
grid.fit(xtrain,ytrain)
```

```
[CV] END ..criterion=gini, max_depth=21, min_samples_leaf=41; total time=   0.4s
[CV] END ..criterion=gini, max_depth=21, min_samples_leaf=41; total time=   0.4s
[CV] END ..criterion=gini, max_depth=21, min_samples_leaf=41; total time=   0.4s
[CV] END ..criterion=gini, max_depth=21, min_samples_leaf=46; total time=   0.4s
[CV] END ..criterion=gini, max_depth=21, min_samples_leaf=46; total time=   0.4s
[CV] END ..criterion=gini, max_depth=21, min_samples_leaf=46; total time=   0.4s
[CV] END ..criterion=gini, max_depth=21, min_samples_leaf=46; total time=   0.4s
[CV] END ..criterion=gini, max_depth=21, min_samples_leaf=46; total time=   0.4s
[CV] END ...criterion=gini, max_depth=26, min_samples_leaf=1; total time=   0.7s
[CV] END ...criterion=gini, max_depth=26, min_samples_leaf=1; total time=   0.6s
[CV] END ...criterion=gini, max_depth=26, min_samples_leaf=1; total time=   0.7s
[CV] END ...criterion=gini, max_depth=26, min_samples_leaf=1; total time=   0.6s
[CV] END ...criterion=gini, max_depth=26, min_samples_leaf=1; total time=   0.6s
[CV] END ...criterion=gini, max_depth=26, min_samples_leaf=6; total time=   0.5s
[CV] END ...criterion=gini, max_depth=26, min_samples_leaf=6; total time=   0.5s
[CV] END ...criterion=gini, max_depth=26, min_samples_leaf=6; total time=   0.5s
[CV] END ...criterion=gini, max_depth=26, min_samples_leaf=6; total time=   0.5s
[CV] END ..criterion=gini, max_depth=26, min_samples_leaf=11; total time=   0.5s
```

In [50]:

```
grid.best_params_
```

Out[50]:

```
{'criterion': 'gini', 'max_depth': 46, 'min_samples_leaf': 1}
```

In [51]:

```
grid.best_estimator_
```

Out[51]:

```
RandomForestClassifier(max_depth=46)
```

In [63]:

```
dt= mymodel(grid.best_estimator_)
```

```
Training Acuuracy:- 1.0
 Testing Accuracy:- 0.8447297700806211
              precision    recall  f1-score   support

           0       0.88      0.82      0.85      1760
           1       0.81      0.87      0.84      1589

    accuracy                           0.84      3349
   macro avg       0.85      0.85      0.84      3349
weighted avg       0.85      0.84      0.84      3349
```

Ensemble Learning :Boosting

In [53]:

```
!pip install XGBOOST
```

```
Collecting XGBOOST
  Downloading xgboost-1.7.3-py3-none-win_amd64.whl (89.1 MB)
Requirement already satisfied: scipy in c:\users\hp\anaconda3\lib\site-packages (from XGBOOST) (1.7.1)
Requirement already satisfied: numpy in c:\users\hp\anaconda3\lib\site-packages (from XGBOOST) (1.20.3)
Installing collected packages: XGBOOST
Successfully installed XGBOOST-1.7.3
```

In [54]:

```
from xgboost import XGBClassifier
```

In [57]:

```
mymodel(XGBClassifier(max_depth=2))
```

```
Training Acuuracy:- 0.8590810188147958
 Testing Accuracy:- 0.8456255598686175
              precision    recall  f1-score   support

           0       0.87      0.83      0.85      1760
           1       0.82      0.86      0.84      1589

    accuracy                           0.85      3349
   macro avg       0.85      0.85      0.85      3349
weighted avg       0.85      0.85      0.85      3349
```

Out[57]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=2, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [59]:

```
from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier
```

In [61]:

```
mymodel(AdaBoostClassifier())
```

```
Training Acuuracy:- 0.8263151158325868
 Testing Accuracy:- 0.826515377724694
              precision    recall  f1-score   support

           0       0.82      0.85      0.84      1760
           1       0.83      0.80      0.81      1589

    accuracy                           0.83      3349
   macro avg       0.83      0.83      0.83      3349
weighted avg       0.83      0.83      0.83      3349
```

Out[61]:

```
AdaBoostClassifier()
```

In [62]:

```
mymodel(GradientBoostingClassifier())
```

```
Training Acuuracy:- 0.8585690515806988
 Testing Accuracy:- 0.8378620483726485
              precision    recall  f1-score   support

           0       0.87      0.82      0.84      1760
           1       0.81      0.86      0.83      1589

    accuracy                           0.84      3349
   macro avg       0.84      0.84      0.84      3349
weighted avg       0.84      0.84      0.84      3349
```

Out[62]:

```
GradientBoostingClassifier()
```

Conclusion

We Found that after applying these Boosting ,Bagging and hyperparameter Tuning we can find our best Fit Data with a accuracy of close to 84 and only 16 percent of data is giving error.

In [ ]: