```
1 import pandas as pd
```

```
1 import matplotlib.pyplot as plt
```

```
1 import seaborn as sns
```

```
1 import numpy as np
```

import data

```
1 df=pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Bank%20Churn%20Modellin
2 #df=pd.read_csv('Bank.txt')
```

```
1 df.head()
```

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products
0	15634602	Hargrave	619	France	Female	42	2	0.00	1
1	15647311	Hill	608	Spain	Female	41	1	83807.86	1
2	15619304	Onio	502	France	Female	42	8	159660.80	3
3	15701354	Boni	699	France	Female	39	1	0.00	2

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   CustomerId        10000 non-null  int64  
 1   Surname           10000 non-null  object 
 2   CreditScore       10000 non-null  int64  
 3   Geography         10000 non-null  object 
 4   Gender            10000 non-null  object 
 5   Age               10000 non-null  int64  
 6   Tenure            10000 non-null  int64  
 7   Balance           10000 non-null  float64
 8   Num Of Products   10000 non-null  int64  
 9   Has Credit Card   10000 non-null  int64  
 10  Is Active Member  10000 non-null  int64  
 11  Estimated Salary  10000 non-null  float64
```

```
 12  Churn             10000 non-null  int64
dtypes: float64(2), int64(8), object(3)
memory usage: 1015.8+ KB
```

```
df.duplicated('CustomerId').sum()
```

```
0
```

```
df=df.set_index('CustomerId')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 15634602 to 15628319
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Surname           10000 non-null  object
 1   CreditScore       10000 non-null  int64
 2   Geography         10000 non-null  object
 3   Gender            10000 non-null  object
 4   Age               10000 non-null  int64
 5   Tenure            10000 non-null  int64
 6   Balance           10000 non-null  float64
 7   Num Of Products   10000 non-null  int64
 8   Has Credit Card   10000 non-null  int64
 9   Is Active Member  10000 non-null  int64
 10  Estimated Salary  10000 non-null  float64
 11  Churn             10000 non-null  int64
dtypes: float64(2), int64(7), object(3)
memory usage: 1015.6+ KB
```

encoding

```
df['Geography'].value_counts()
```

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

```
df.replace({'Geography':{'France':2,'Germany':1,'Spain':0}},inplace=True)
```

```
df['Gender'].value_counts()
```

```
Male      5457
Female    4543
```

```
    Name: Gender, dtype: int64
```

```
1 df.replace({'Gender':{'Male':0,'Female':1}},inplace=True)
```

```
1 df['Num Of Products'].value_counts()
```

```
    1    5084
    2    4590
    3     266
    4      60
    Name: Num Of Products, dtype: int64
```

```
1 df.replace({'Num Of Products':{1:0,2:1,3:1,4:1}},inplace=True)
```

```
1 df['Has Credit Card'].value_counts()
```

```
    1    7055
    0    2945
    Name: Has Credit Card, dtype: int64
```

```
1 df['Is Active Member'].value_counts()
```

```
    1    5151
    0    4849
    Name: Is Active Member, dtype: int64
```

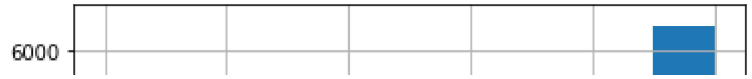
```
1 df.loc[(df['Balance']==0),'Churn'].value_counts()
```

```
    0    3117
    1     500
    Name: Churn, dtype: int64
```

```
1 df['Zero Balance']=np.where(df['Balance']>0,1,0)
```

```
1 df['Zero Balance'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efd06c59eb0>
```




```
1 df.groupby(['Churn','Geography']).count()
```

		Surname	CreditScore	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card
Churn	**Geography**								
0	**0**	2064	2064	2064	2064	2064	2064	2064	2064
	1	1695	1695	1695	1695	1695	1695	1695	1695
	2	4204	4204	4204	4204	4204	4204	4204	4204
1	**0**	413	413	413	413	413	413	413	413
	1	814	814	814	814	814	814	814	814

define label and features

```
1 df.columns
```

```
Index(['Surname', 'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure',
       'Balance', 'Num Of Products', 'Has Credit Card', 'Is Active Member',
       'Estimated Salary', 'Churn', 'Zero Balance'],
      dtype='object')
```

```
1 x=df.drop(['Surname','Churn'],axis=1)
```

```
1 y=df['Churn']
```

```
1 x.shape,y.shape
```

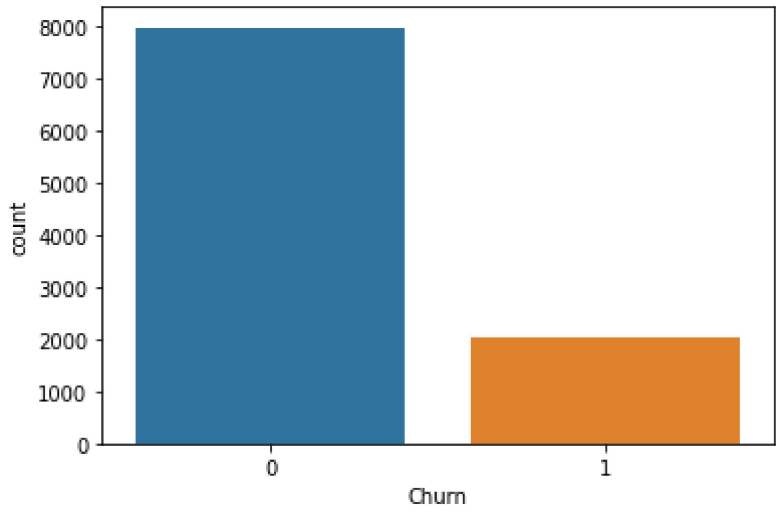
```
((10000, 11), (10000,))
```

handling imbalance data

```
1 df['Churn'].value_counts()
```

```
0    7963
1    2037
Name: Churn, dtype: int64
```

```
1 sns.countplot(x='Churn',data=df);
```




```
1 x.shape,y.shape
```

```
((10000, 11), (10000,))
```

random under sampling

```
1 from imblearn.under_sampling import RandomUnderSampler
```

```
1 rus=RandomUnderSampler(random_state=2529)
```

```
1 x_rus,y_rus=rus.fit_resample(x,y)
```

```
1 x_rus.shape,y_rus.shape,x.shape,y.shape
```

```
((4074, 11), (4074,), (10000, 11), (10000,))
```

```
1 y.value_counts()
```

```
0    7963
1    2037
Name: Churn, dtype: int64
```

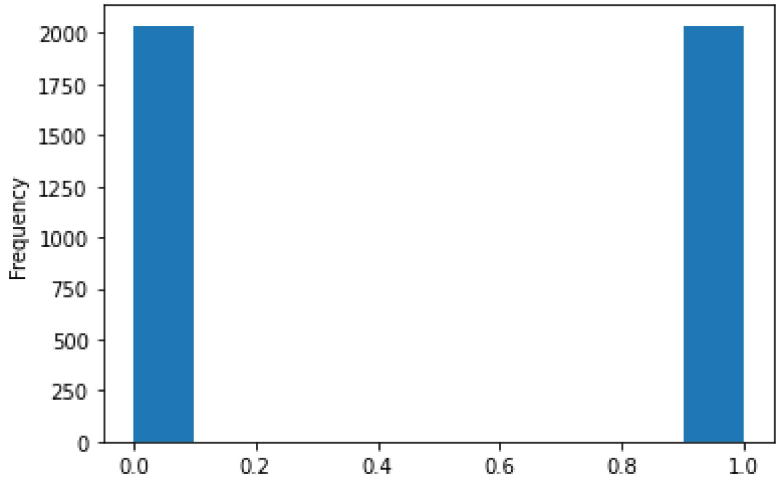
```
1 y_rus.value_counts()
```

```
0    2037
1    2037
```

```
Name: Churn, dtype: int64
```

```
1 y_rus.plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efd051e4eb0>
```



random over sampling

```
1 from imblearn.over_sampling import RandomOverSampler
```

```
1 ros=RandomOverSampler(random_state=2529)
```

```
1 x_ros,y_ros=ros.fit_resample(x,y)
```

```
1 x_ros.shape,y_ros.shape,x.shape,y.shape
```

```
((15926, 11), (15926,), (10000, 11), (10000,))
```

```
1 y.value_counts()
```

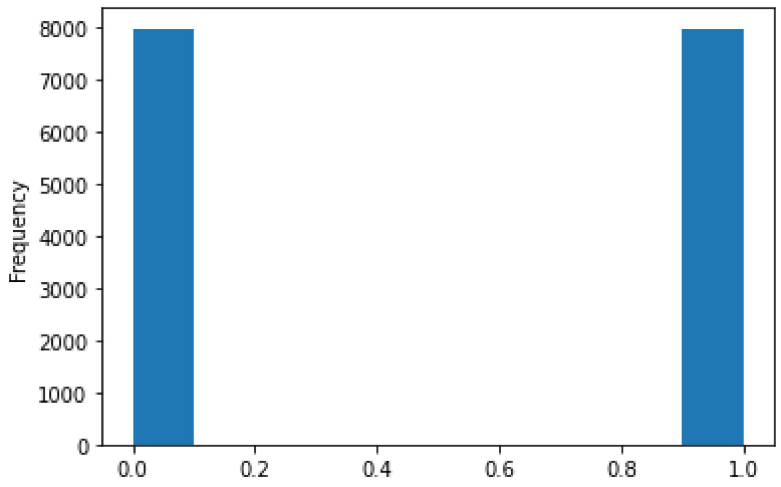
```
0    7963
1    2037
Name: Churn, dtype: int64
```

```
1 y_ros.value_counts()
```

```
1    7963
0    7963
Name: Churn, dtype: int64
```

```
1 y_ros.plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efd04fc75e0>
```




```
1 df.head()
```

CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	C
15634602	Hargrave	619	2	1	42	2	0.00	0	
15647311	Hill	608	0	1	41	1	83807.86	0	
15619304	Onio	502	2	1	42	8	159660.80	1	
15701354	Boni	699	2	1	39	1	0.00	1	

train split data

```
1 from sklearn.model_selection import train_test_split
```

split original data

```
1 x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=2529)
```

split random under sample data

```
1 x_train_rus,x_test_rus,y_train_rus,y_test_rus=train_test_split(x_rus,y_rus,test_size=0.3,r
```

split random over sample data

```
1 x_train_ros,x_test_ros,y_train_ros,y_test_ros=train_test_split(x_ros,y_ros,test_size=0.3,r
```

standardized features

```
1 from sklearn.preprocessing import StandardScaler
```

```
1 sc=StandardScaler()
```

standardized original data

```
1 x_train[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(x_tr
```

```
1 x_test[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(x_tes
```

standardized random under sample data

```
1 x_train_rus[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(
```

```
1 x_test_rus[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(x
```

standardized random over sample data

```
1 x_train_ros[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(
```

```
1 x_test_ros[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(x
```

```
1 x_train_rus.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Ac Me
1014	-1.301006	1	0	-0.019928	1.026218	0.624803	0	0	
3723	1.216940	1	0	0.640697	-0.678763	0.661166	0	1	
234	0.635875	2	1	-1.152426	1.367214	-1.328359	1	0	

```
1 x_train_ros.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	A M
13139	0.587468	0	1	1.108470	-0.675574	-1.325006	1	0	
11842	1.028833	1	0	0.547861	1.371995	0.465209	0	0	
9560	0.382182	2	0	-0.012748	0.006949	0.771989	1	1	
14762	0.577204	0	1	0.547861	-0.675574	-0.291961	0	0	

```
1 x_test_rus.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Ac Me
770	1.752046	2	0	-0.777063	-1.025147	0.841169	1	1	
3840	-0.606250	2	0	-0.401394	0.720543	0.076794	1	1	
2732	-0.902314	0	1	0.162109	-0.326871	0.239316	1	1	
1727	0.629048	0	1	-1.152732	-0.676009	0.638218	0	1	

```
1 x_test_ros.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	A M
7302	-1.595700	2	1	2.917462	0.355672	0.144460	0	1	
2486	0.370116	0	1	-0.487662	0.701008	-1.343204	1	1	
13035	-0.500167	1	0	1.687834	1.046344	0.583257	1	0	
335	-1.657132	2	0	-1.149770	0.701008	1.624845	0	1	

support vector machine classifier

```
1 from sklearn.svm import SVC
```

```
1 svc=SVC()
```

```
1 svc.fit(x_train,y_train)
```

```
SVC()
```

```
1 y_pred=svc.predict(x_test)
```

model accuracy

```
1 from sklearn.metrics import confusion_matrix,classification_report
```

```
1 confusion_matrix(y_test,y_pred)
```

```
array([[2381,   33],
       [ 436,  150]])
```

```
1 print(classification_report(y_test,y_pred))
```

```
                precision    recall  f1-score   support

             0       0.85      0.99      0.91      2414
             1       0.82      0.26      0.39       586

      accuracy                           0.84      3000
     macro avg       0.83      0.62      0.65      3000
  weighted avg       0.84      0.84      0.81      3000
```

```
1 from sklearn.model_selection import GridSearchCV
```

```
1 param_grid={'C':[0.1,1,10],
2             'gamma':[1,0.1,0.01],
3             'class_weight':['balanced']}
```

```
1 grid=GridSearchCV(SVC(),param_grid,refit=True,verbose=2,cv=2)
2 grid.fit(x_train,y_train)
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits
[CV] END ..............C=0.1, class_weight=balanced, gamma=1; total time=   1.4s
[CV] END ..............C=0.1, class_weight=balanced, gamma=1; total time=   1.4s
[CV] END ............C=0.1, class_weight=balanced, gamma=0.1; total time=   1.0s
[CV] END ............C=0.1, class_weight=balanced, gamma=0.1; total time=   1.0s
[CV] END ...........C=0.1, class_weight=balanced, gamma=0.01; total time=   1.1s
[CV] END ...........C=0.1, class_weight=balanced, gamma=0.01; total time=   1.1s
[CV] END ................C=1, class_weight=balanced, gamma=1; total time=   1.2s
[CV] END ................C=1, class_weight=balanced, gamma=1; total time=   1.2s
[CV] END ..............C=1, class_weight=balanced, gamma=0.1; total time=   0.9s
[CV] END ..............C=1, class_weight=balanced, gamma=0.1; total time=   0.9s
[CV] END ............C=1, class_weight=balanced, gamma=0.01; total time=   1.0s
[CV] END ............C=1, class_weight=balanced, gamma=0.01; total time=   1.0s
[CV] END ...............C=10, class_weight=balanced, gamma=1; total time=   1.2s
[CV] END ...............C=10, class_weight=balanced, gamma=1; total time=   1.2s
[CV] END ............C=10, class_weight=balanced, gamma=0.1; total time=   0.9s
[CV] END ............C=10, class_weight=balanced, gamma=0.1; total time=   1.0s
[CV] END ............C=10, class_weight=balanced, gamma=0.01; total time=   0.9s
[CV] END ............C=10, class_weight=balanced, gamma=0.01; total time=   1.0s
GridSearchCV(cv=2, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'class_weight': ['balanced'],
                         'gamma': [1, 0.1, 0.01]},
             verbose=2)
```

```
1 print(grid.best_estimator_)
```

```
SVC(C=10, class_weight='balanced', gamma=1)
```

```
1 grid_predictions=grid.predict(x_test)
```

```
1 confusion_matrix(y_test,grid_predictions)
```

```
array([[2159,  255],
       [ 343,  243]])
```

```
1 print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.86      0.89      0.88      2414
           1       0.49      0.41      0.45       586

    accuracy                           0.80      3000
   macro avg       0.68      0.65      0.66      3000
weighted avg       0.79      0.80      0.79      3000
```

model with random forest classifier

```
1 from sklearn.ensemble import RandomForestRegressor
```

model with under smapling

```
1 svc_rus=SVC()
```

```
1 svc_rus.fit(x_train_rus,y_train_rus)
```

```
SVC()
```

```
1 y_pred_rus=svc_rus.predict(x_test_rus)
```

model accuracy

```
1 confusion_matrix(y_test_rus,y_pred_rus)
```

```
array([[470, 157],
       [174, 422]])
```

```
1 print(classification_report(y_test_rus,y_pred_rus))
```

```
                precision    recall  f1-score   support

             0       0.73      0.75      0.74       627
             1       0.73      0.71      0.72       596

      accuracy                           0.73      1223
     macro avg       0.73      0.73      0.73      1223
  weighted avg       0.73      0.73      0.73      1223
```

hyperparameter tunning

```
1 param_grid={'C':[0.1,1,10],
2              'gamma':[1,0.1,0.01],
3              'kernel':['rbf'],
4              'class_weight':['balanced']}
```

```
4               class_weight :[ balanced ]}
```

```
1 grid_rus=GridSearchCV(SVC(),param_grid,refit=True,verbose=2,cv=2)
2 grid_rus.fit(x_train_rus,y_train_rus)
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   0.2s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   0.2s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.2s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.2s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.2s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.2s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   0.2s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   0.2s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.2s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.2s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.2s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.2s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   0.2s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   0.2s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.2s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.2s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.2s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.2s
GridSearchCV(cv=2, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'class_weight': ['balanced'],
                         'gamma': [1, 0.1, 0.01], 'kernel': ['rbf']},
             verbose=2)
```

```
1 print(grid_rus.best_estimator_)
```

```
SVC(C=1, class_weight='balanced', gamma=0.1)
```

```
1 grid_predictions_rus=grid_rus.predict(x_test_rus)
```

```
1 confusion_matrix(y_test_rus,grid_predictions_rus)
```

```
array([[476, 151],
       [172, 424]])
```

```
1 print(classification_report(y_test_rus,grid_predictions_rus))
```

```
              precision    recall  f1-score   support

           0       0.73      0.76      0.75       627
           1       0.74      0.71      0.72       596

    accuracy                           0.74      1223
   macro avg       0.74      0.74      0.74      1223
weighted avg       0.74      0.74      0.74      1223
```

model with over sampling

```
1 svc_ros=SVC()
```

```
1 svc_ros.fit(x_train_ros,y_train_ros)
```

```
SVC()
```

```
1 y_pred_ros=svc_ros.predict(x_test_ros)
```

model accuracy

```
1 confusion_matrix(y_test_ros,y_pred_ros)
```

```
array([[1823,  556],
       [ 626, 1773]])
```

```
1 print(classification_report(y_test_ros,y_pred_ros))
```

```
                precision    recall  f1-score   support

             0       0.74      0.77      0.76      2379
             1       0.76      0.74      0.75      2399

      accuracy                           0.75      4778
     macro avg       0.75      0.75      0.75      4778
  weighted avg       0.75      0.75      0.75      4778
```

hyperparameter tunning

```
1 param_grid={'C':[0.1,1,10],
2             'gamma':[1,0.1,0.01],
3             'kernel':['rbf'],
4             'class_weight':['balanced']}
```

```
1 grid_ros=GridSearchCV(SVC(),param_grid,refit=True,verbose=2,cv=2)
2 grid_ros.fit(x_train_ros,y_train_ros)
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.6s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.6s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.6s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.5s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.9s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.8s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.0s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.0s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.3s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.3s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.5s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.5s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   2.8s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   2.8s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.5s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.5s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.5s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.4s
GridSearchCV(cv=2, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'class_weight': ['balanced'],
                         'gamma': [1, 0.1, 0.01], 'kernel': ['rbf']},
             verbose=2)
```

```
1 print(grid_ros.best_estimator_)
```

```
SVC(C=10, class_weight='balanced', gamma=1)
```

```
1 grid_predictions_ros=grid_ros.predict(x_test_ros)
```

```
1 confusion_matrix(y_test_ros,grid_predictions_ros)
```

```
array([[2047,  332],
       [  68, 2331]])
```

```
1 print(classification_report(y_test_ros,grid_predictions_ros))
```

```
              precision    recall  f1-score   support

           0       0.97      0.86      0.91      2379
           1       0.88      0.97      0.92      2399

    accuracy                           0.92      4778
   macro avg       0.92      0.92      0.92      4778
weighted avg       0.92      0.92      0.92      4778
```

compare

```
1 print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.85      0.99      0.91      2414
               1       0.82      0.26      0.39       586

        accuracy                           0.84      3000
       macro avg       0.83      0.62      0.65      3000
    weighted avg       0.84      0.84      0.81      3000
```

```
1 print(classification_report(y_test,grid_predictions))
```

```
                  precision    recall  f1-score   support

               0       0.86      0.89      0.88      2414
               1       0.49      0.41      0.45       586

        accuracy                           0.80      3000
       macro avg       0.68      0.65      0.66      3000
    weighted avg       0.79      0.80      0.79      3000
```

```
1 print(classification_report(y_test_rus,grid_predictions_rus))
```

```
                  precision    recall  f1-score   support

               0       0.73      0.76      0.75       627
               1       0.74      0.71      0.72       596

        accuracy                           0.74      1223
       macro avg       0.74      0.74      0.74      1223
    weighted avg       0.74      0.74      0.74      1223
```

```
1 print(classification_report(y_test_ros,grid_predictions_ros))
```

```
                  precision    recall  f1-score   support

               0       0.97      0.86      0.91      2379
               1       0.88      0.97      0.92      2399

        accuracy                           0.92      4778
       macro avg       0.92      0.92      0.92      4778
    weighted avg       0.92      0.92      0.92      4778
```

✓ 0s completed at 12:24 PM