```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Bank%20Churn%20Modelling.csv')
print(df)
```

```
      CustomerId    Surname  CreditScore Geography  Gender  Age  Tenure  \
0       15634602   Hargrave          619    France  Female   42       2
1       15647311       Hill          608     Spain  Female   41       1
2       15619304       Onio          502    France  Female   42       8
3       15701354       Boni          699    France  Female   39       1
4       15737888   Mitchell          850     Spain  Female   43       2
...          ...        ...          ...       ...     ...  ...     ...
9995    15606229   Obijiaku          771    France    Male   39       5
9996    15569892  Johnstone          516    France    Male   35      10
9997    15584532        Liu          709    France  Female   36       7
9998    15682355  Sabbatini          772   Germany    Male   42       3
9999    15628319     Walker          792    France  Female   28       4

        Balance  Num Of Products  Has Credit Card  Is Active Member  \
0          0.00                1                1                 1
1      83807.86                1                0                 1
2     159660.80                3                1                 0
3          0.00                2                0                 0
4     125510.82                1                1                 1
...         ...              ...              ...               ...
9995       0.00                2                1                 0
9996   57369.61                1                1                 1
9997       0.00                1                0                 1
9998   75075.31                2                1                 0
9999  130142.79                1                1                 0

      Estimated Salary  Churn
0            101348.88      1
1            112542.58      0
2            113931.57      1
3             93826.63      0
4             79084.10      0
...                ...    ...
9995          96270.64      0
9996         101699.77      0
9997          42085.58      1
9998          92888.52      1
9999          38190.78      0

[10000 rows x 13 columns]
```

```
df.head()
```

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	E
0	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	
1	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	
2	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	
3	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	
4	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	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
```

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

```
0
```

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

```
1 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
```

```
1 df.describe()
```

	CreditScore	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000

Encoding

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

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

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

```
1 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
```

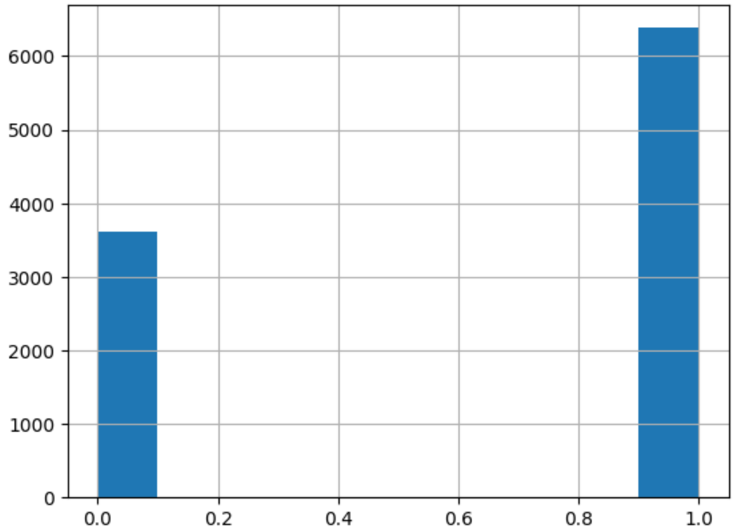
```
1 df.loc[(df['Balance'] == 0), 'Churn'].value_counts()
```

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

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

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

```
<Axes: >
```




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

		Surname	CreditScore	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated
Churn	**Geography**										
0	**0**	2064	2064	2064	2064	2064	2064	2064	2064	2064	
	1	1695	1695	1695	1695	1695	1695	1695	1695	1695	
	2	4204	4204	4204	4204	4204	4204	4204	4204	4204	
1	**0**	413	413	413	413	413	413	413	413	413	
	1	814	814	814	814	814	814	814	814	814	
	2	810	810	810	810	810	810	810	810	810	

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', 'Zer Balance'],
      dtype='object')
```

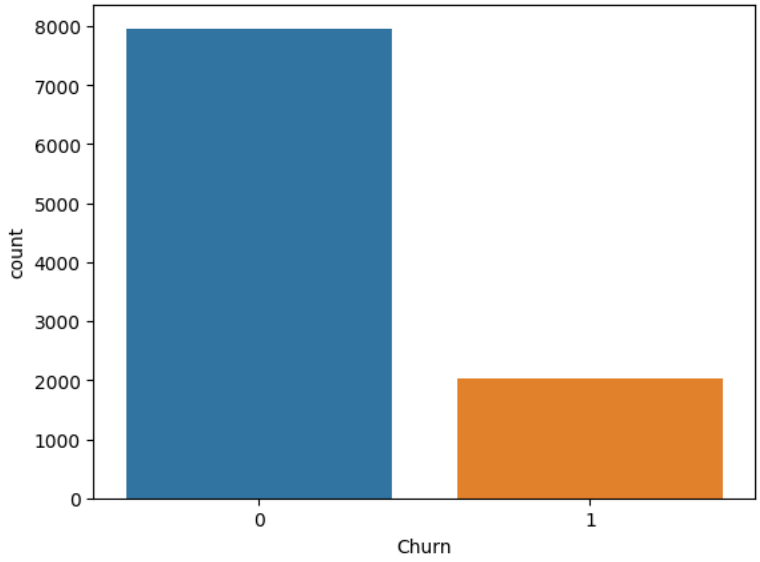
```
1 x = df.drop(['Surname','Churn'], axis = 1)
```

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

```
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')
```

```
<Axes: ylabel='Frequency'>
```

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()
```

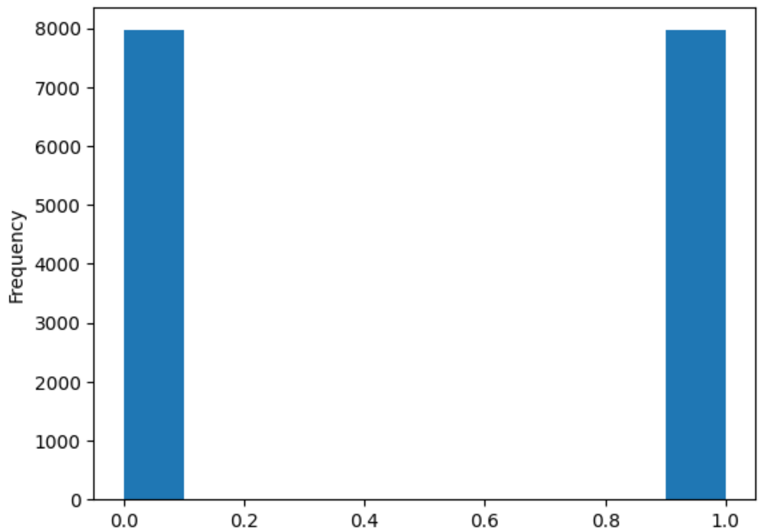
```
0    7963
1    2037
Name: Churn, dtype: int64
```

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

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

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

```
<Axes: ylabel='Frequency'>
```



Train Test Split

```
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,random_state=2529)
```

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,random_state=2529)
```

Standard Features

```
1 from sklearn.preprocessing import StandardScaler
2 sc = StandardScaler()
```

Standard Original Data

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

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

Standardize Random Under Sample Data

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

```
1 x_test_rus[['CreditScore','Age','Tenure','Balance','Estimated Salary']] = sc.fit_transform(x_test_rus[['CreditScore','Age','T
```

Standardize Random over Sample Data

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

```
1 x_test_ros[['CreditScore','Age','Tenure','Balance','Estimated Salary']] = sc.fit_transform(x_test_ros[['CreditScore','Age','T
```

Support Vector Machine Classifier

```
1 from sklearn.svm import SVC
2 svc = SVC()
```

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

```
▾ SVC
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
```

Hyperparameter Tunning

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

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

```
grid = GridSearchCV(SVC(),param_grid,refit = True,verbose = 2, cv = 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, kernel=rbf; total time=   2.6s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.0s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.3s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.1s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.2s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.2s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.3s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.3s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.9s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.0s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.9s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.8s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.3s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.1s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.1s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.0s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.0s
```

▸ **GridSearchCV**

▸ **estimator: SVC**

▸ SVC

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

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

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

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

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

```
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 Under Sampling

```
svc_rus = SVC()
```

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

▾ SVC

SVC()

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

Model Accuracy

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

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

```
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

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

```
grid_rus = GridSearchCV(SVC(),param_grid,refit = True,verbose = 2, cv = 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=   1.2s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   2.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.5s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.4s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   0.4s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   0.4s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.3s
[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**

▸ **estimator: SVC**

▸ SVC

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

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

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

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

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

```
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 Random Over Sampling

```
svc_ros = SVC()
```

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

```
▾ SVC
SVC()
```

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

Model Accuracy

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

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

```
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

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

```
grid_ros = GridSearchCV(SVC(),param_grid,refit = True,verbose = 2, cv = 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=   6.7s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.8s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.8s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   3.5s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   3.7s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   3.0s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.2s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   4.7s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.4s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.4s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.7s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   5.1s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.1s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   2.9s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   2.8s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   4.0s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.9s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.5s
```

```
▸ GridSearchCV
▸ estimator: SVC
  ▸ SVC
```

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

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

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

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

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

```
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
```

Let's 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,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
```

```
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,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
```

```
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
```