In [1]:
```python
import pandas as pd
```

In [2]:
```python
import numpy as np
```

In [3]:
```python
import matplotlib.pyplot as plt
```

In [4]:
```python
import seaborn as sns
```

In [5]:
```python
df = pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Bank%20Churn%20Modelling.csv')
```

In [6]:
```python
df.head()
```

Out[6]:

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary	Churn
0	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

In [7]:
```python
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
```

In [8]:
```python
df.duplicated('CustomerId').sum()
```

Out[8]:
```
0
```

In [9]:
```python
df = df.set_index('CustomerId')
```

In [10]:
```python
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
```

In [11]:
```python
df['Geography'].value_counts()
```

Out[11]:
```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [12]:
```python
df.replace({'Geography': {'France': 2,'Germany' : 1, 'Spain': 0}},inplace=True )
```

In [13]:
```python
df['Gender'].value_counts()
```

Out[13]:
```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

In [14]:
```python
df.replace({'Gender': {'Male': 0, 'Female':1}},inplace=True)
```

In [15]:
```python
df['Num Of Products'].value_counts()
```

Out[15]:
```
1    5084
2    4590
3     266
4      60
Name: Num Of Products, dtype: int64
```

In [16]:
```python
df.replace({'Num Of Products': {1: 0,2:1,3:1,4:1}},inplace=True)
```

In [17]:
```python
df['Has Credit Card'].value_counts()
```

Out[17]:
```
1    7055
0    2945
Name: Has Credit Card, dtype: int64
```

In [18]:
```python
df['Is Active Member'].value_counts()
```

Out[18]:
```
1    5151
0    4849
Name: Is Active Member, dtype: int64
```

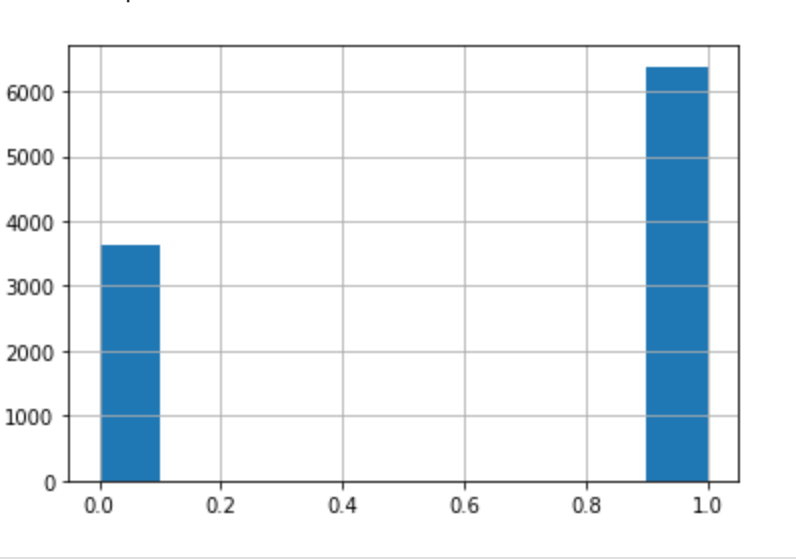
In [19]:
```python
df.loc[(df['Balance']==0), 'Churn'].value_counts()
```

Out[19]:
```
0    3117
1     500
Name: Churn, dtype: int64
```

In [20]:
```python
df['Zero Balance'] =np.where(df['Balance']>0,1,0)
```

In [21]:
```python
df['Zero Balance'].hist()
```

Out[21]:
```
<AxesSubplot:>
```



In [22]:
```python
df.groupby(['Churn', 'Geography']).count()
```

Out[22]:

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

In [23]:
```python
df.columns
```

Out[23]:
```
Index(['Surname', 'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure',
       'Balance', 'Num Of Products', 'Has Credit Card', 'Is Active Member',
       'Estimated Salary', 'Churn', 'Zero Balance'],
      dtype='object')
```

In [24]:
```python
X = df.drop(['Surname', 'Churn'],axis =1)
```

In [25]:
```python
y = df['Churn']
```

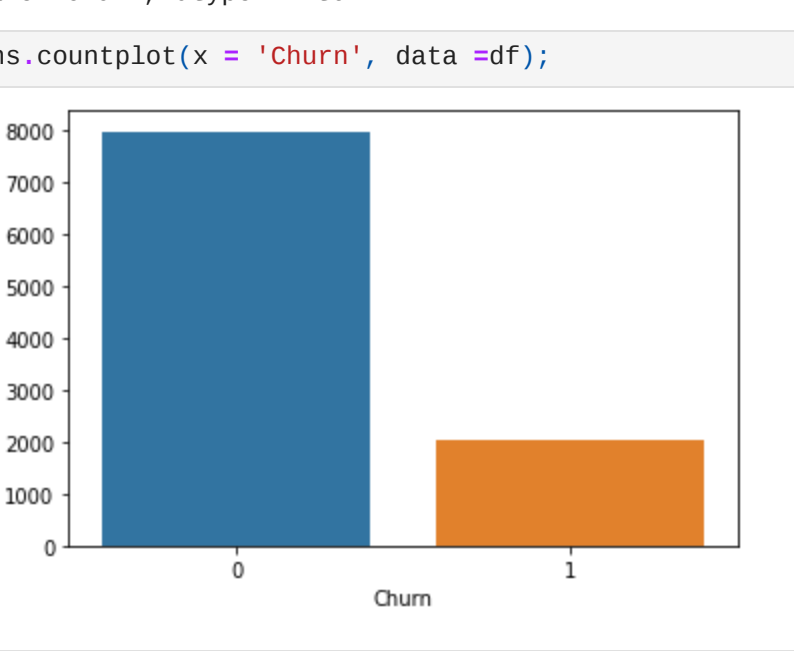
In [26]:
```python
X.shape,y.shape
```

Out[26]:
```
((10000, 11), (10000,))
```

In [27]:
```python
df['Churn'].value_counts()
```

Out[27]:
```
0    7963
1    2037
Name: Churn, dtype: int64
```

In [28]:
```python
sns.countplot(x = 'Churn', data =df);
```



In [29]:
```python
X.shape,y.shape
```

Out[29]:
```
((10000, 11), (10000,))
```

In [31]:
```python
from imblearn.under_sampling import RandomUnderSampler
```

In [32]:
```python
rus =RandomUnderSampler(random_state=192529)
```

In [33]:
```python
X_rus, y_rus = rus.fit_resample(X,y)
```

In [34]:
```python
X_rus.shape,y_rus.shape,X.shape,y.shape
```

Out[34]:
```
((4074, 11), (4074,), (10000, 11), (10000,))
```

In [35]:
```python
y.value_counts()
```

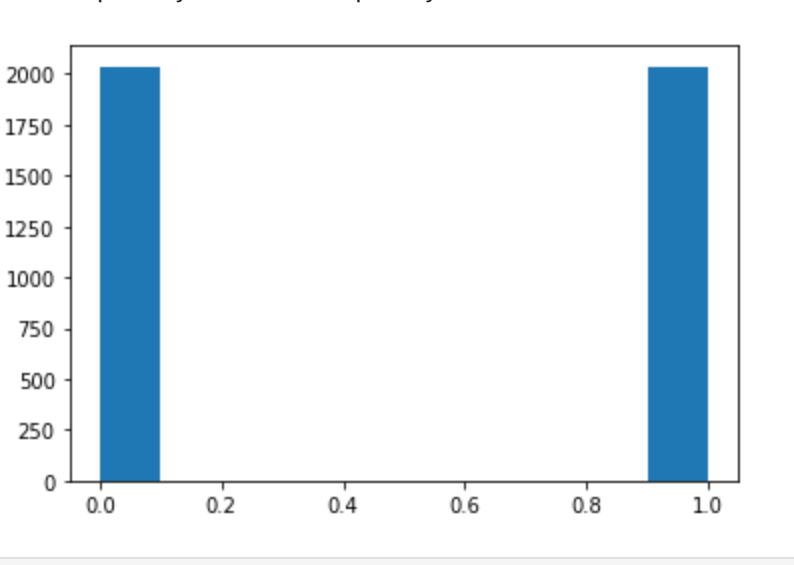
Out[35]:
```
0    7963
1    2037
Name: Churn, dtype: int64
```

In [36]:
```python
y_rus.value_counts()
```

Out[36]:
```
0    2037
1    2037
Name: Churn, dtype: int64
```

In [37]:
```python
y_rus.plot(kind ='hist')
```

Out[37]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [38]:
```python
from imblearn.over_sampling import RandomOverSampler
```

In [39]:
```python
ros = RandomUnderSampler(random_state=72529)
```

In [40]:
```python
X_ros, y_ros = ros.fit_resample(X,y)
```

In [41]:
```python
X_ros.shape, y_ros.shape, X.shape,y.shape
```

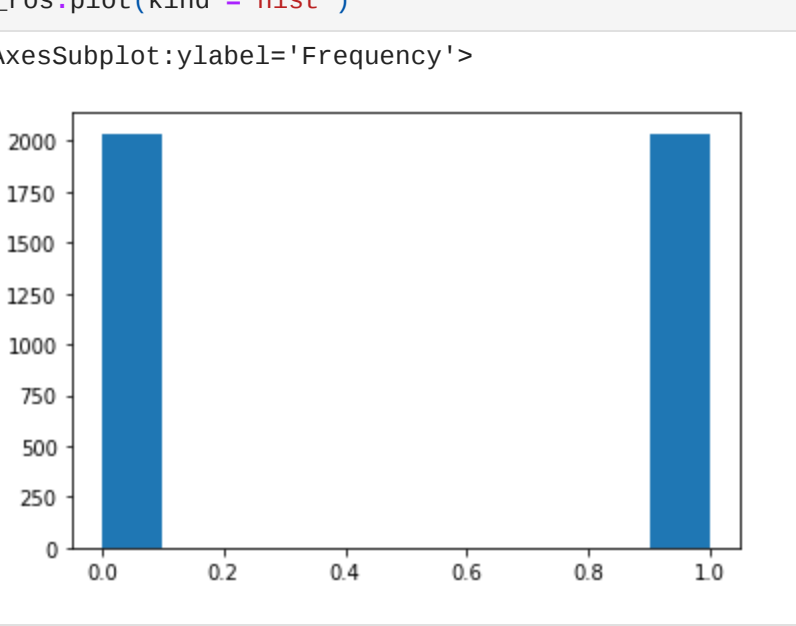
Out[41]:
```
((4074, 11), (4074,), (10000, 11), (10000,))
```

In [42]:
```python
y.value_counts()
```

Out[42]:
```
0    7963
1    2037
Name: Churn, dtype: int64
```

In [43]:
```python
y_ros.plot(kind ='hist')
```

Out[43]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [44]:
```python
from sklearn.model_selection import train_test_split
```

In [45]:
```python
X_train,X_test,y_train, y_test =train_test_split(X,y,test_size=0.3,random_state =72529)
```

In [46]:
```python
X_train_rus,X_test_rus,y_train_rus,y_test_rus = train_test_split(X_rus,y_rus,test_size=0.7)
```

In [47]:
```python
X_train_ros,X_test_ros,y_train_ros,y_test_ros = train_test_split(X_ros,y_ros, test_size=0.7)
```

In [48]:
```python
from sklearn.preprocessing import StandardScaler
```

In [49]:
```python
sc= StandardScaler()
```

In [50]:
```python
X_train[['CreditScore','Age', 'Tenure','Balance','Estimated Salary']] = sc.fit_transform(X_train[['CreditScore','Age','Tenure','Balance','Estimated Salary']])
```

In [52]:
```python
X_test[['CreditScore','Age', 'Tenure','Balance','Estimated Salary']] = sc.fit_transform(X_test[['CreditScore','Age','Tenure','Balance','Estimated Salary']])
```

In [53]:
```python
X_train_rus[['CreditScore','Age', 'Tenure','Balance','Estimated Salary']] = sc.fit_transform(X_train_rus[['CreditScore','Age','Tenure','Balance','Estimated Salary']])
```

In [54]:
```python
X_test_rus[['CreditScore','Age', 'Tenure','Balance','Estimated Salary']] = sc.fit_transform(X_test_rus[['CreditScore','Age','Tenure','Balance','Estimated Salary']])
```

In [55]:
```python
X_train_ros[['CreditScore','Age', 'Tenure','Balance','Estimated Salary']] = sc.fit_transform(X_train_ros[['CreditScore','Age','Tenure','Balance','Estimated Salary']])
```

In [56]:
```python
X_test_ros[['CreditScore','Age', 'Tenure','Balance','Estimated Salary']] = sc.fit_transform(X_test_ros[['CreditScore','Age','Tenure','Balance','Estimated Salary']])
```

In [57]:
```python
from sklearn.svm import SVC
```

In [58]:
```python
svc =SVC()
```

In [59]:
```python
svc.fit(X_train,y_train)
```

Out[59]:
```
▾ SVC
SVC()
```

In [61]:
```python
y_pred = svc.predict(X_test)
```

In [62]:
```python
from sklearn.metrics import confusion_matrix, classification_report
```

In [63]:
```python
confusion_matrix(y_test, y_pred)
```

Out[63]:
```
array([[2330,   34],
       [ 477,  159]], dtype=int64)
```

In [64]:
```python
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.99      0.90      2364
           1       0.82      0.25      0.38       636

    accuracy                           0.83      3000
   macro avg       0.83      0.62      0.64      3000
weighted avg       0.83      0.83      0.79      3000
```

In [65]:
```python
from sklearn.model_selection import GridSearchCV
```

In [66]:
```python
param_grid = {'C': [0.1,1,10],
              'gamma': [1,0.1,0.01],'kernel': ['rbf'],
              'class_weight': ['balanced']}
```

In [67]:
```python
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=   1.4s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
[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=   1.2s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.4s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.1s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.1s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   2.1s
[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=   1.7s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.9s
[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=   1.3s
```

Out[67]:
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [68]:
```python
print(grid.best_estimator_)
```

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

In [69]:
```python
grid_predictions = grid.predict(X_test)
```

In [70]:
```python
confusion_matrix(y_test,grid_predictions)
```

Out[70]:
```
array([[2027,  337],
       [ 311,  325]], dtype=int64)
```

In [71]:
```python
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.86      2364
           1       0.49      0.51      0.50       636

    accuracy                           0.78      3000
   macro avg       0.68      0.68      0.68      3000
weighted avg       0.79      0.78      0.79      3000
```

In [72]:
```python
svc_rus =SVC()
```

In [73]:
```python
svc_rus.fit(X_train_rus,y_train_rus)
```

Out[73]:
```
▾ SVC
SVC()
```

In [74]:
```python
svc.predict(X_test_rus)
```

Out[74]:
```
array([1, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [75]:
```python
confusion_matrix(y_test,grid_predictions)
```

Out[75]:
```
array([[2027,  337],
       [ 311,  325]], dtype=int64)
```

In [76]:
```python
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.86      2364
           1       0.49      0.51      0.50       636

    accuracy                           0.78      3000
   macro avg       0.68      0.68      0.68      3000
weighted avg       0.79      0.78      0.79      3000
```

In [77]:
```python
param_grid = {'C': [0.1,1,10],
              'gamma': [1,0.1,0.01],'kernel': ['rbf'],
              'class_weight': ['balanced']}
```

In [78]:
```python
grid_rus = GridSearchCV(SVC(),param_grid, refit =True,verbose=2,cv= 2)
grid_rus.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=   1.5s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.6s
[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=   1.3s
[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=   1.4s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.4s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.1s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.0s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.3s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.2s
[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=   1.4s
```

Out[78]:
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [79]:
```python
print(grid_rus.best_estimator_)
```

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

In [80]:
```python
grid_predictions_rus = grid_rus.predict(X_test_rus)
```

In [81]:
```python
confusion_matrix(y_test_rus,grid_rus.predict(X_test_rus))
```

Out[81]:
```
array([[1292,  125],
       [ 498,  937]], dtype=int64)
```

In [82]:
```python
print(classification_report(y_test_rus,grid_predictions_rus))
```

```
              precision    recall  f1-score   support

           0       0.72      0.91      0.81      1417
           1       0.88      0.65      0.75      1435

    accuracy                           0.78      2852
   macro avg       0.80      0.78      0.78      2852
weighted avg       0.80      0.78      0.78      2852
```

In [83]:
```python
svc_ros = SVC()
```

In [84]:
```python
svc_ros.fit(X_train_ros,y_train_ros)
```

Out[84]:
```
▾ SVC
SVC()
```

In [86]:
```python
y_pred_ros = svc_ros.predict(X_test_ros)
```

In [87]:
```python
confusion_matrix(y_test_ros,y_pred_ros)
```

Out[87]:
```
array([[1073,  343],
       [ 410, 1026]], dtype=int64)
```

In [88]:
```python
print(classification_report(y_test_ros,y_pred_ros))
```

```
              precision    recall  f1-score   support

           0       0.72      0.76      0.74      1416
           1       0.75      0.71      0.73      1436

    accuracy                           0.74      2852
   macro avg       0.74      0.74      0.74      2852
weighted avg       0.74      0.74      0.74      2852
```

In [89]:
```python
param_grid = {'C': [0.1,1,10],
              'gamma': [1,0.1,0.01],'kernel': ['rbf'],
              'class_weight': ['balanced']}
```

In [90]:
```python
grid_ros = GridSearchCV(SVC(),param_grid, refit =True,verbose=2,cv= 2)
grid_ros.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=   1.5s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.5s
[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=0.1, kernel=rbf; total time=   1.2s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.3s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.3s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.2s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.4s
[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=   0.9s
[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=   1.0s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.4s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.8s
[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=   1.5s
[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=   1.7s
```

Out[90]:
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [91]:
```python
print(grid_ros.best_estimator_)
```

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

In [92]:
```python
grid_predictions_ros = grid_ros.predict(X_test_ros)
```

In [93]:
```python
confusion_matrix(y_test_ros,grid_predictions_ros)
```

Out[93]:
```
array([[1298,  118],
       [ 501,  935]], dtype=int64)
```

In [94]:
```python
print(classification_report(y_test_ros,grid_predictions_ros))
```

```
              precision    recall  f1-score   support

           0       0.72      0.92      0.81      1416
           1       0.89      0.65      0.75      1436

    accuracy                           0.78      2852
   macro avg       0.80      0.78      0.78      2852
weighted avg       0.81      0.78      0.78      2852
```

In [95]:
```python
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.99      0.90      2364
           1       0.82      0.25      0.38       636

    accuracy                           0.83      3000
   macro avg       0.83      0.62      0.64      3000
weighted avg       0.83      0.83      0.79      3000
```

In [96]:
```python
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.86      2364
           1       0.49      0.51      0.50       636

    accuracy                           0.78      3000
   macro avg       0.68      0.68      0.68      3000
weighted avg       0.79      0.78      0.79      3000
```

In [97]:
```python
print(classification_report(y_test_rus,grid_predictions_rus))
```

```
              precision    recall  f1-score   support

           0       0.72      0.91      0.81      1417
           1       0.88      0.65      0.75      1435

    accuracy                           0.78      2852
   macro avg       0.80      0.78      0.78      2852
weighted avg       0.80      0.78      0.78      2852
```

In [98]:
```python
print(classification_report(y_test_ros,y_pred_ros))
```

```
              precision    recall  f1-score   support

           0       0.72      0.76      0.74      1416
           1       0.75      0.71      0.73      1436

    accuracy                           0.74      2852
   macro avg       0.74      0.74      0.74      2852
weighted avg       0.74      0.74      0.74      2852
```

In [99]:
```python
print(classification_report(y_test_ros,grid_predictions_ros))
```

```
              precision    recall  f1-score   support

           0       0.72      0.92      0.81      1416
           1       0.89      0.65      0.75      1436

    accuracy                           0.78      2852
   macro avg       0.80      0.78      0.78      2852
weighted avg       0.81      0.78      0.78      2852
```

In [ ]: