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

In [2]:
```
df = pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Bank%20Churn%20Modelling.csv')
df.head()
```

Out[2]:

	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 [3]:
```
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 [4]:
```
df.duplicated('CustomerId')
```

Out[4]:
```
0       False
1       False
2       False
3       False
4       False
        ...
9995    False
9996    False
9997    False
9998    False
9999    False
Length: 10000, dtype: bool
```

In [5]:
```
df.duplicated('CustomerId').sum()
```

Out[5]:
```
0
```

In [6]:
```
df = df.set_index('CustomerId')
```

In [7]:
```
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

In [8]:
```
df['Geography'].value_counts()
```

Out[8]:
```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [9]:
```
df.replace({'Geography': {'France':2, 'Germany':1, 'Spain':0}}, inplace = True)
```

In [10]:
```
df
```

Out[10]:

CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary	Churn
15634602	Hargrave	619	2	Female	42	2	0.00	1	1	1	101348.88	1
15647311	Hill	608	0	Female	41	1	83807.86	1	0	1	112542.58	0
15619304	Onio	502	2	Female	42	8	159660.80	3	1	0	113931.57	1
15701354	Boni	699	2	Female	39	1	0.00	2	0	0	93826.63	0
15737888	Mitchell	850	0	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
15606229	Obijiaku	771	2	Male	39	5	0.00	2	1	0	96270.64	0
15569892	Johnstone	516	2	Male	35	10	57369.61	1	1	1	101699.77	0
15584532	Liu	709	2	Female	36	7	0.00	1	0	1	42085.58	1
15682355	Sabbatini	772	1	Male	42	3	75075.31	2	1	0	92888.52	1
15628319	Walker	792	2	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

In [11]:
```
df.replace({'Gender': {'Female':1, 'Male':0}}, inplace = True)
```

In [12]:
```
df['Num Of Products'].value_counts()
```

Out[12]:
```
1    5084
2    4590
3     266
4      60
Name: Num Of Products, dtype: int64
```

In [13]:
```
df.replace({'Num Of Products': {1:0, 2:1, 3:1, 4:1}}, inplace = True)
```

In [14]:
```
df
```

Out[14]:

CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary	Churn
15634602	Hargrave	619	2	1	42	2	0.00	0	1	1	101348.88	1
15647311	Hill	608	0	1	41	1	83807.86	0	0	1	112542.58	0
15619304	Onio	502	2	1	42	8	159660.80	1	1	0	113931.57	1
15701354	Boni	699	2	1	39	1	0.00	1	0	0	93826.63	0
15737888	Mitchell	850	0	1	43	2	125510.82	0	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...
15606229	Obijiaku	771	2	0	39	5	0.00	1	1	0	96270.64	0
15569892	Johnstone	516	2	0	35	10	57369.61	0	1	1	101699.77	0
15584532	Liu	709	2	1	36	7	0.00	0	0	1	42085.58	1
15682355	Sabbatini	772	1	0	42	3	75075.31	1	1	0	92888.52	1
15628319	Walker	792	2	1	28	4	130142.79	0	1	0	38190.78	0

10000 rows × 12 columns

In [15]:
```
df['Has Credit Card'].value_counts()
```

Out[15]:
```
1    7055
0    2945
Name: Has Credit Card, dtype: int64
```

In [16]:
```
df['Is Active Member'].value_counts()
```

Out[16]:
```
1    5151
0    4849
Name: Is Active Member, dtype: int64
```

In [17]:
```
df.loc[(df['Balance']==0), 'Churn'].value_counts()
```

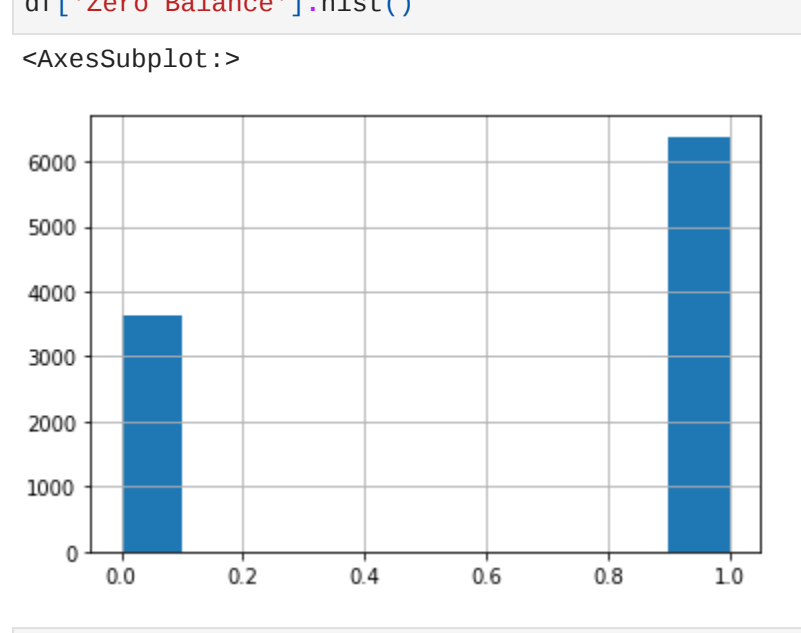
Out[17]:
```
0    3117
1     500
Name: Churn, dtype: int64
```

In [18]:
```
import numpy as np
```

In [19]:
```
df['Zero Balance'] = np.where(df['Balance']>0, 1, 0)
```

In [20]:
```
df['Zero Balance'].hist()
```

Out[20]:
```
<AxesSubplot:>
```



In [21]:
```
df.groupby (['Churn','Geography']).count()
```

Out[21]:

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

Define Label and Features

In [22]:
```
df.columns
```

Out[22]:
```
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 [23]:
```
X = df.drop(['Surname','Churn'], axis = 1)
```

In [24]:
```
Y = df['Churn']
```

In [25]:
```
X.shape , Y.shape
```

Out[25]:
```
((10000, 11), (10000,))
```

Handling Imbalance Data

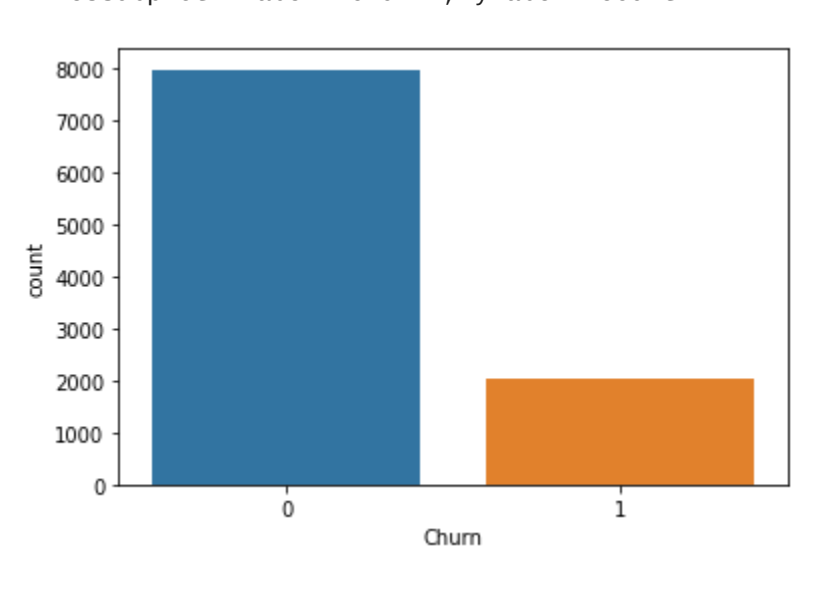
In [26]:
```
df['Churn'].value_counts()
```

Out[26]:
```
0    7963
1    2037
Name: Churn, dtype: int64
```

In [27]:
```
import seaborn as sns
```

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

Out[28]:
```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



Random Under Sampling

In [29]:
```
from imblearn.under_sampling import RandomUnderSampler
```

In [30]:
```
rus = RandomUnderSampler(random_state = 2529)
```

In [31]:
```
X_rus,Y_rus = rus.fit_resample(X,Y)
```

In [32]:
```
X_rus.shape, Y_rus.shape, X.shape, Y.shape
```

Out[32]:
```
((4074, 11), (4074,), (10000, 11), (10000,))
```

In [33]:
```
Y.value_counts()
```

Out[33]:
```
0    7963
1    2037
Name: Churn, dtype: int64
```

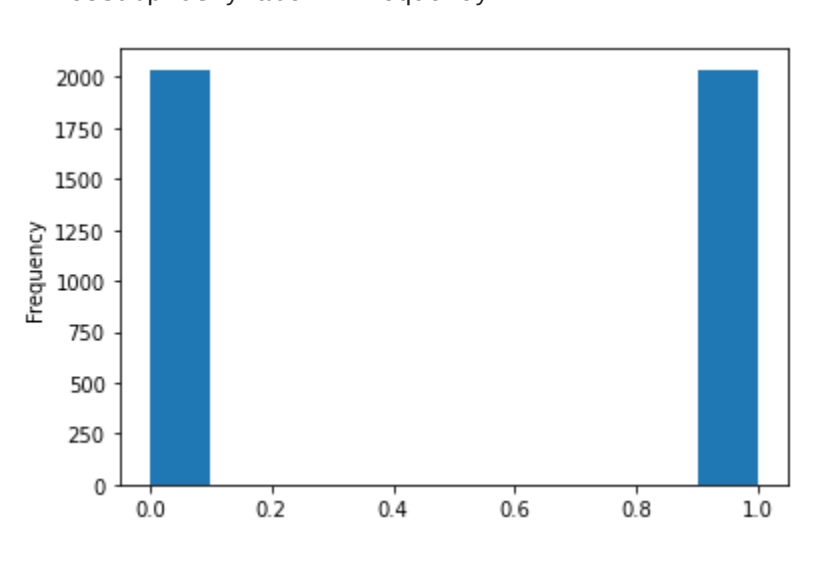
In [34]:
```
Y_rus.value_counts()
```

Out[34]:
```
0    2037
1    2037
Name: Churn, dtype: int64
```

In [35]:
```
import matplotlib.pyplot as plt
```

In [36]:
```
Y_rus.plot(kind = 'hist')
```

Out[36]:
```
<AxesSubplot:ylabel='Frequency'>
```



Random Over Sampling

In [37]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [38]:
```
ros = RandomOverSampler(random_state = 2529)
```

In [39]:
```
X_ros,Y_ros = ros.fit_resample(X,Y)
```

In [40]:
```
X_ros.shape, Y_ros.shape, X.shape, Y.shape
```

Out[40]:
```
((15926, 11), (15926,), (10000, 11), (10000,))
```

In [41]:
```
Y.value_counts()
```

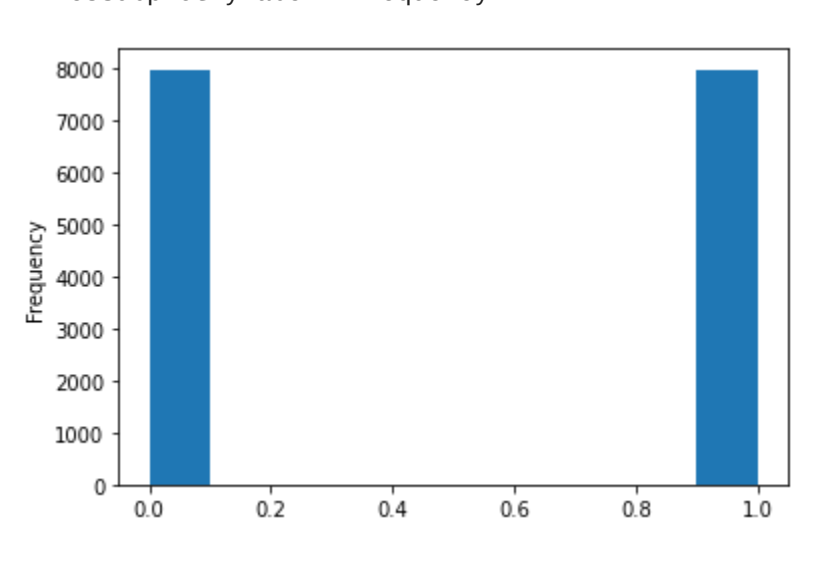
Out[41]:
```
0    7963
1    2037
Name: Churn, dtype: int64
```

In [42]:
```
Y_ros.value_counts()
```

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

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

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



Test Train Split

Split Orignal Data

In [44]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.3 , random_state = 25)
```

Split Random Under Sample Data

In [45]:
```
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 = 25)
```

Split Random Over Sample Data

In [46]:
```
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 = 25)
```

Standardize Features

In [47]:
```
from sklearn.preprocessing import StandardScaler
```

In [48]:
```
sc = StandardScaler()
```

Standardize Orignal Data

In [49]:
```
X_train[['CreditScore', 'Age','Tenure','Balance','Estimated Salary']]  = sc.fit_transform(X_train[['CreditScore', 'Age','Tenure','Balance','Estimated Salary']])
```

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

Standardize Random Under Sample Data

In [51]:
```
X_train_rus[['CreditScore', 'Age','Tenure','Balance','Estimated Salary']]  = sc.fit_transform(X_train_rus[['CreditScore', 'Age','Tenure','Balance','Estimated Salary']])
```

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

Standardize Random Over Sample Data

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

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

Modelling

In [55]:
```
from sklearn.svm import SVC
```

In [56]:
```
model = SVC()
```

In [57]:
```
model.fit (X_train,Y_train)
```

Out[57]:
```
▾ SVC
SVC()
```

In [58]:
```
Y_pred = model.predict(X_test)
```

Model Accuracy

In [59]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [60]:
```
confusion_matrix(Y_test,Y_pred)
```

Out[60]:
```
array([[2374,   45],
       [ 421,  160]], dtype=int64)
```

In [61]:
```
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.98      0.91      2419
           1       0.78      0.28      0.41       581

    accuracy                           0.84      3000
   macro avg       0.81      0.63      0.66      3000
weighted avg       0.84      0.84      0.81      3000
```

In [62]:
```
from sklearn.model_selection import GridSearchCV
```

In [63]:
```
param_grid = {'C': [0.1,1,10],
              'gamma': [1,0.1,0.01],
              'kernel': ['rbf'],
              'class_weight': ['balanced']}
```

In [64]:
```
grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 2, cv = 2)
```

In [65]:
```
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=   0.9s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.9s
[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=   1.1s
[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=   1.2s
[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=   1.0s
[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=   1.2s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.1s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.9s
[CV] END .C=10, class_weight=balanced, gamma=0.1, 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=   0.8s
```

Out[65]:
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [69]:
```
print(grid.best_estimator_)
```

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

In [70]:
```
grid_predictions = grid.predict(X_test)
```

In [71]:
```
confusion_matrix(Y_test, grid_predictions)
```

Out[71]:
```
array([[2166,  253],
       [ 365,  216]], dtype=int64)
```

In [72]:
```
print(classification_report(Y_test, grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      2419
           1       0.46      0.37      0.41       581

    accuracy                           0.79      3000
   macro avg       0.66      0.63      0.64      3000
weighted avg       0.78      0.79      0.79      3000
```

Model with Random Under Sampling

In [73]:
```
svc_rus = SVC()
```

In [74]:
```
svc_rus.fit(X_train_rus,Y_train_rus)
```

Out[74]:
```
▾ SVC
SVC()
```

In [75]:
```
Y_pred_rus = svc_rus.predict(X_test_rus)
```

Model Accuracy

In [76]:
```
confusion_matrix(Y_test_rus, Y_pred_rus)
```

Out[76]:
```
array([[483, 120],
       [172, 448]], dtype=int64)
```

In [77]:
```
print(classification_report(Y_test_rus, Y_pred_rus))
```

```
              precision    recall  f1-score   support

           0       0.74      0.80      0.77       603
           1       0.79      0.72      0.75       620

    accuracy                           0.76      1223
   macro avg       0.76      0.76      0.76      1223
weighted avg       0.76      0.76      0.76      1223
```

Hyperparameter Tunning

In [78]:
```
param_grid = {'C': [0.1,1,10],
              'gamma': [1,0.1,0.01],
              'kernel': ['rbf'],
              'class_weight': ['balanced']}
```

In [79]:
```
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=   0.1s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   0.1s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.1s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.1s
[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=   0.1s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.1s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.1s
[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=   0.1s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   0.1s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.0s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.1s
```

Out[79]:
```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [80]:
```
print(grid_rus.best_estimator_)
```

```
SVC(C=10, class_weight='balanced', gamma=0.01)
```

In [81]:
```
grid_predictions_rus = grid_rus.predict(X_test_rus)
```

In [82]:
```
confusion_matrix(Y_test_rus,grid_predictions_rus)
```

Out[82]:
```
array([[456, 147],
       [161, 459]], dtype=int64)
```

In [83]:
```
print(classification_report(Y_test_rus,grid_predictions_rus))
```

```
              precision    recall  f1-score   support

           0       0.74      0.76      0.75       603
           1       0.76      0.74      0.75       620

    accuracy                           0.75      1223
   macro avg       0.75      0.75      0.75      1223
weighted avg       0.75      0.75      0.75      1223
```

Model with Random Over Sampling

In [84]:
```
svc_ros = SVC()
```

In [85]:
```
svc_ros.fit(X_train_ros,Y_train_ros)
```

Out[85]:
```
▾ SVC
SVC()
```

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

Model Accuracy

In [87]:
```
confusion_matrix(Y_test_ros, Y_pred_ros)
```

Out[87]:
```
array([[1854,  572],
       [ 551, 1801]], dtype=int64)
```

In [88]:
```
print(classification_report(Y_test_ros, Y_pred_ros))
```

```
              precision    recall  f1-score   support

           0       0.77      0.76      0.77      2426
           1       0.76      0.77      0.76      2352

    accuracy                           0.76      4778
   macro avg       0.76      0.76      0.76      4778
weighted avg       0.77      0.76      0.76      4778
```

Hyperparameter Tunning

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

In [90]:
```
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=   4.0s
[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=   2.7s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   3.2s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   3.2s
[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=   3.0s
[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=   2.9s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.6s
[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=   2.7s
[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=   2.4s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.7s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.6s
```

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

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

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

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

In [93]:
```
confusion_matrix(Y_test_ros,grid_predictions_ros)
```

Out[93]:
```
array([[2064,  362],
       [  75, 2277]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

           0       0.96      0.85      0.90      2426
           1       0.86      0.97      0.91      2352

    accuracy                           0.91      4778
   macro avg       0.91      0.91      0.91      4778
weighted avg       0.91      0.91      0.91      4778
```

In [ ]:
```
# ROS model is the best among all 3 models
```