Title of Project

BANK CUSTOMER CHURN MODEL

Objective

1. **Data Encoding**
2. **Feature Scaling**
3. **Handling Imbalance Data**
    1. Random Under Sampling
    2. Random Over Sampling
4. **Support Vector Machine Classifier**
5. **Grid Search for Hyperparameter Tunning**

Data Source

https://github.com/YBI-Foundation/Dataset

Double-click (or enter) to edit

Import Library

```
import pandas as pd
```

```
import numpy as np
```

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

```
import seaborn as sns
```

Import Data

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

Analyze Data

```
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

```
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'>
Index: 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
```

Data Encoding

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

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

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

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

```
Gender
Male      5457
Female    4543
Name: count, dtype: int64
```

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

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

```
Num Of Products
1    5084
2    4590
3     266
```

```
4      60
Name: count, dtype: int64
```

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

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

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

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

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

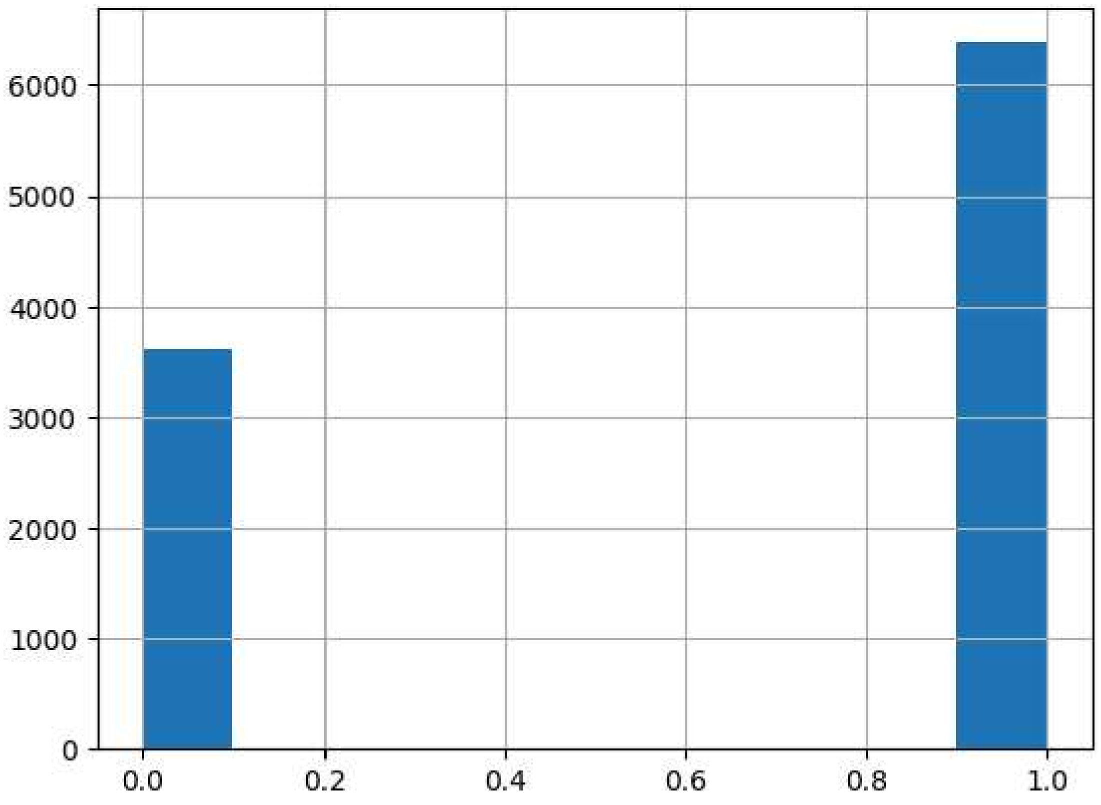
```
df.loc[(df['Balance']==0), 'Churn'].value_counts()
```

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

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

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

```
<Axes: >
```



```
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

```
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')
```

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

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

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

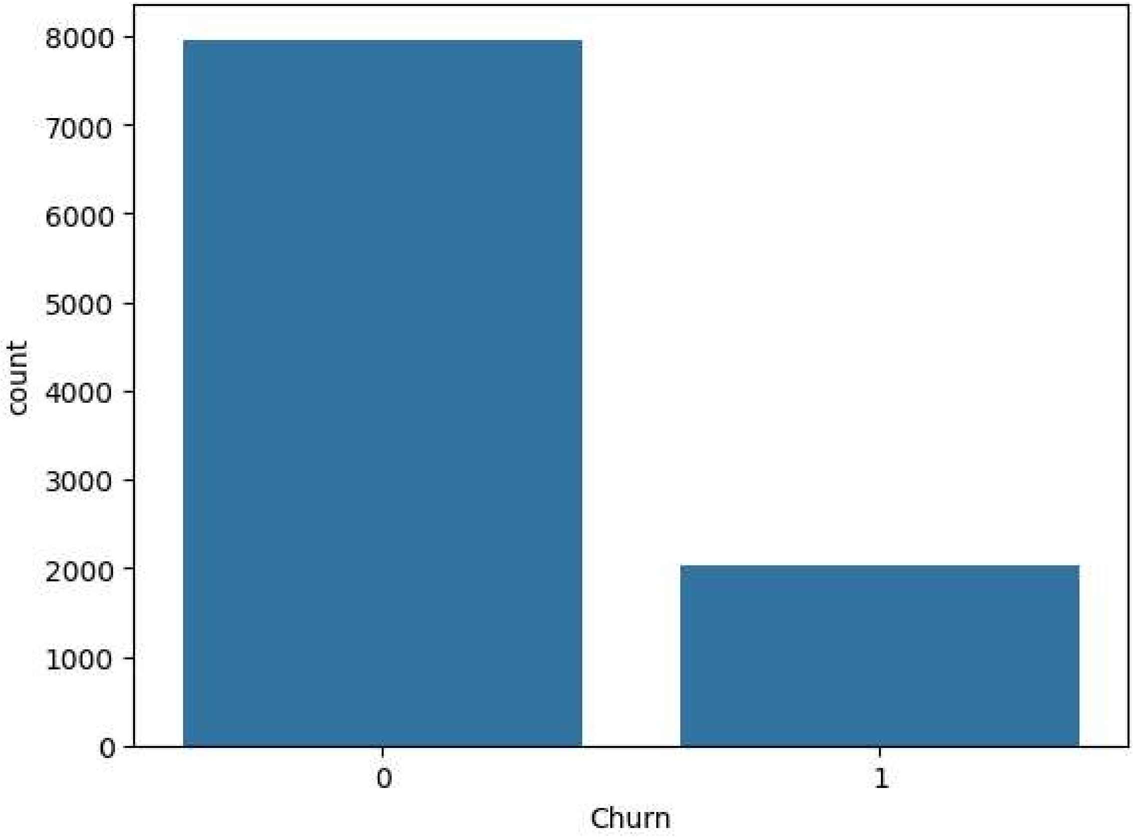
```
((10000, 11), (10000,))
```

Undersampling and Oversampling

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

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

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



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

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

Random Under Sampling

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

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

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

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

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

```
y.value_counts()
```

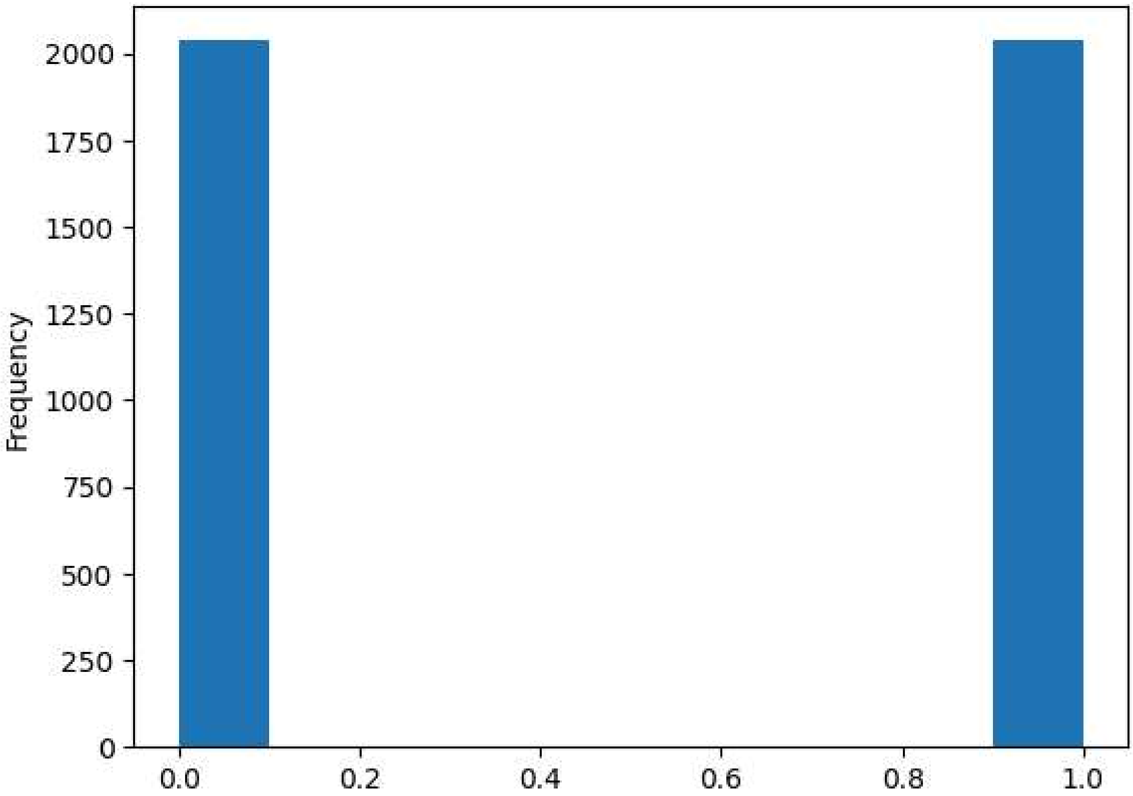
```
Churn
0    7963
1    2037
Name: count, dtype: int64
```

```
y_rus.value_counts()
```

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

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

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



Random Over Sampling

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

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

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

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

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

```
y.value_counts()
```

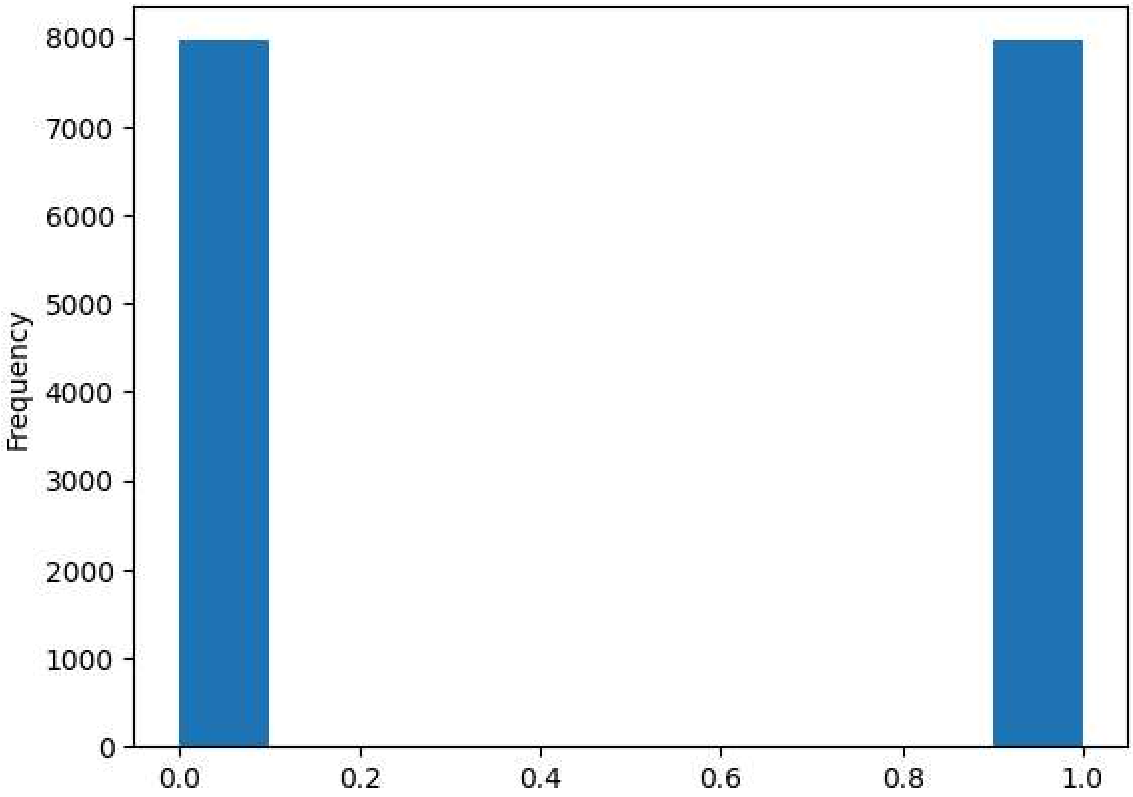
```
Churn
0    7963
1    2037
Name: count, dtype: int64
```

```
y_ros.value_counts()
```

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

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

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



Train Test Split

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

Split Original Data

```
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

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

Split Random Over Sample Data

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

Standardize Features

Double-click (or enter) to edit

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

```
sc = StandardScaler()
```

Standardize Original Data

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

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

Standardize Random Under Sample Data

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

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

Standardize Random Over Sample Data

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

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

Support Vector Machine Classifier

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

```
svc = SVC()
```

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

```
▾ SVC
SVC()
```

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

Model Accuracy

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

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

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

```
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 Tuning

```
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=   2.9s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.7s
[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=0.1, kernel=rbf; total time=   2.0s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.6s
[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=   1.4s
[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=   1.5s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.9s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.3s
[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=   1.3s
[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=   1.1s
```

▸ **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))
```

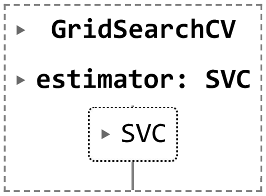
```
              precision    recall  f1-score   support

           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 Tuning

```
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=   0.4s
[CV] END ..C=0.1, class_weight=balanced, gamma=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=0.1, kernel=rbf; total time=   0.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   0.3s
[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
```



```
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 Tuning

```
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=   6.0s
[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=   2.7s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   3.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   3.9s
[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=   3.1s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   4.1s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   3.1s
[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=   3.7s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   9.2s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=  10.0s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   6.4s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   7.1s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   7.3s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   6.2s
```

▸ **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)
```