```
#Bank Churn Model Using Ml



# 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://raw.githubusercontent.com/YBI-Foundation/Dataset/main/Bank%20Churn%20Modelling.csv')


#Analyse Data

df.head()
```

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member
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

```
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
```

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

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

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

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

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

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

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

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

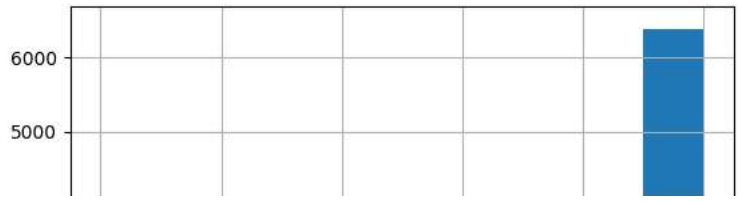
```
df.loc[(df['Balance']==0),'Churn'].value_counts()
```

```
0    3117
1     500
Name: Churn, 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	Is Active Member	Estimat Sal
Churn	**Geography**										
0	**0**	2064	2064	2064	2064	2064	2064	2064	2064	2064	2(
	1	1695	1695	1695	1695	1695	1695	1695	1695	1695	1(
	2	4204	4204	4204	4204	4204	4204	4204	4204	4204	4
1	**0**	413	413	413	413	413	413	413	413	413	
	1	814	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']
```

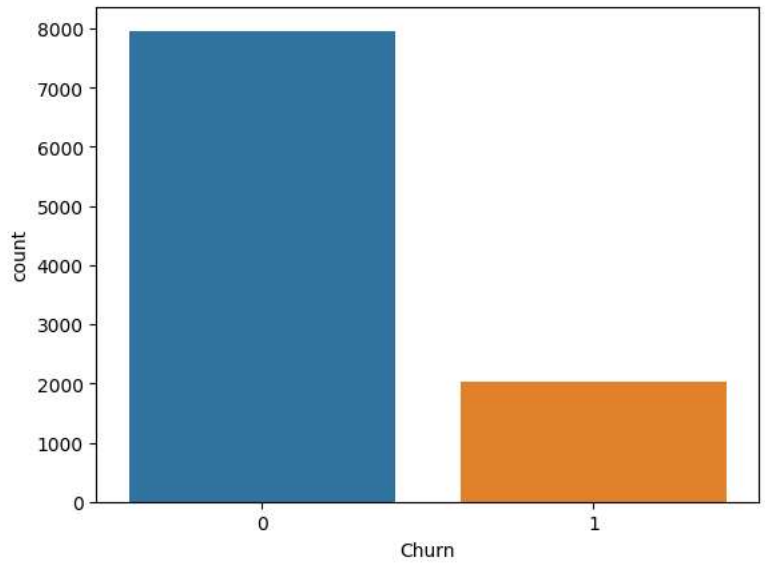
```
X.shape,y.shape
```

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

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

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

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



```
#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()
```

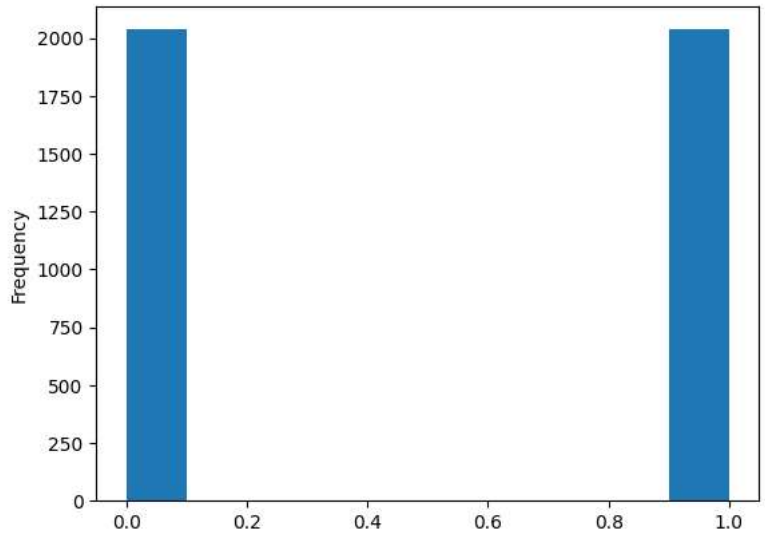
```
0    7963
1    2037
Name: Churn, dtype: int64
```

```
y_rus.value_counts()
```

```
0    2037
1    2037
Name: Churn, 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
```

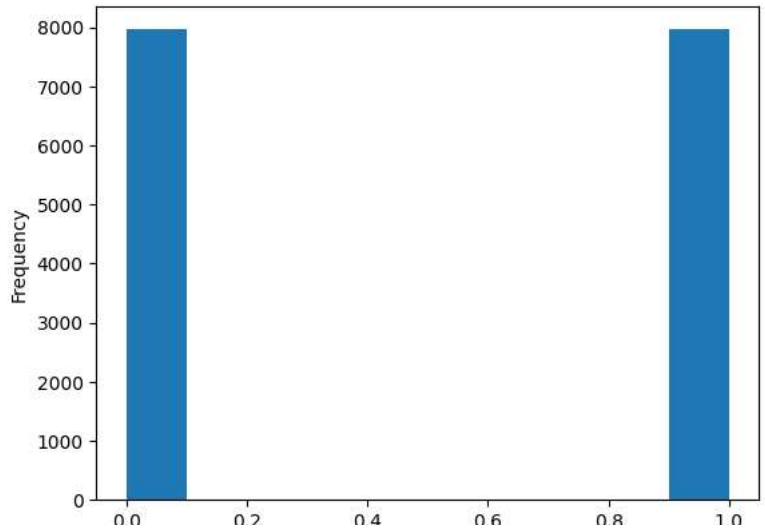
```
((15926, 11), (15926,), (10000, 11), (10000,))
```

```
y.value_counts()
```

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

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

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




```
#Train Test Split
from sklearn.model_selection import train_test_split


#Split Orignal Data
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
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

from sklearn.preprocessing import StandardScaler


sc=StandardScaler()


#Standardize Orignal Data

X_train[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(X_train[['CreditScore','Age','Tenure','Balance','Estima



#Standardize Orignal Data

X_test[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(X_test[['CreditScore','Age','Tenure','Balance','Estimate


#Standardize Random Under Sample Data

X_train_rus[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(X_train_rus[['CreditScore','Age','Tenure','Balance'
X_test_rus[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(X_test_rus[['CreditScore','Age','Tenure','Balance','



#standardize Random over Sample Data
X_train_ros[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(X_train_ros[['CreditScore','Age','Tenure','Balance'
X_test_rus[['CreditScore','Age','Tenure','Balance','Estimated Salary']]=sc.fit_transform(X_test_rus[['CreditScore','Age','Tenure','Balance','



X_train_rus.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary

X_train_ros.head()

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimate Salar
3292	-1.570296	2	1	-0.012695	-1.362155	1.542957	1	1	0	1.42697
1681	0.324677	0	1	-0.855625	-0.676186	-1.315602	1	1	1	-0.70382
12685	0.232489	2	0	1.017554	1.038738	1.445272	0	0	0	-0.76067
8656	0.109572	0	0	-0.949284	-1.362155	0.057618	0	1	1	0.65299

X_test_ros.head()

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary	Z Bala
15904	749	1	0	54	3	144768.94	0	1	0	93336.30	
7215	548	1	0	32	2	98986.28	0	1	1	55867.38	
1704	654	2	1	29	4	96974.97	0	0	1	141404.07	
12499	850	2	1	42	8	0.00	0	1	0	19632.64	

X_test_rus.head()

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary
1480	1.330783	0	0	-0.816414	1.418574	-1.283758	1	1	0	-1.566655
3012	0.434908	2	1	0.285452	1.418574	-1.283758	1	0	0	0.067755
2555	0.342231	1	1	0.928207	-1.419734	1.065661	0	0	0	0.185799
549	0.795318	2	0	-0.357303	-1.064946	-1.283758	0	0	0	0.383623

```
#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)

y_pred=svc.predict(X_test)

#Model Accuracy
from sklearn.metrics import confusion_matrix,classification_report

confusion_matrix(y_test,y_pred)
```

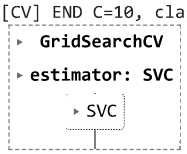
```
array([[2374,   45],
       [ 421,  160]])
```

```
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
```

```
#Hyperparameter Tunning
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=   1.5s
[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=   1.1s
[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=   2.2s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, 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=0.01, 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=   1.3s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
[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
```




```
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([[2166,  253],
       [ 365,  216]])
```

```
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
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([[476, 150],
       [164, 433]])
```

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

```
                   precision    recall  f1-score   support

               0       0.74      0.76      0.75       626
               1       0.74      0.73      0.73       597

        accuracy                           0.74      1223
       macro avg       0.74      0.74      0.74      1223
    weighted avg       0.74      0.74      0.74      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=   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=   0.3s
[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=   0.3s
[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
estimator: SVC
SVC

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

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

```
grid_predictions_rus =grid_rus.predict(X_test_rus)


confusion_matrix(y_test_rus,grid_predictions_rus)
```

```
array([[462, 164],
       [170, 427]])
```

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

```
                   precision    recall  f1-score   support

                0       0.73      0.74      0.73       626
                1       0.72      0.72      0.72       597

         accuracy                           0.73      1223
        macro avg       0.73      0.73      0.73      1223
     weighted avg       0.73      0.73      0.73      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([[2375,    0],
       [2403,    0]])
```

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

```
                   precision    recall  f1-score   support

                0       0.50      1.00      0.66      2375
                1       0.00      0.00      0.00      2403

         accuracy                           0.50      4778
        macro avg       0.25      0.50      0.33      4778
     weighted avg       0.25      0.50      0.33      4778

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
```

```
#Hyperparameter Tunning
```

```
#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=   3.8s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   4.8s
[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=   2.7s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   3.1s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   4.2s
[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=   3.2s
```

```
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([[2375,    0],
       [2403,    0]])
```

SVC

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

```
                   precision    recall  f1-score   support

               0       0.50      1.00      0.66      2375
               1       0.00      0.00      0.00      2403

        accuracy                           0.50      4778
       macro avg       0.25      0.50      0.33      4778
    weighted avg       0.25      0.50      0.33      4778

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
```

```
#Lets Compare
```

```
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
```

```
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
```

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

```
                   precision    recall  f1-score   support

               0       0.74      0.76      0.75       626
               1       0.74      0.73      0.73       597

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

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

```
                  precision    recall  f1-score   support

               0       0.73      0.74      0.73       626
               1       0.72      0.72      0.72       597

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

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

```
                  precision    recall  f1-score   support

               0       0.50      1.00      0.66      2375
               1       0.00      0.00      0.00      2403

        accuracy                           0.50      4778
       macro avg       0.25      0.50      0.33      4778
    weighted avg       0.25      0.50      0.33      4778

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
```

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

```
                  precision    recall  f1-score   support

               0       0.50      1.00      0.66      2375
               1       0.00      0.00      0.00      2403

        accuracy                           0.50      4778
       macro avg       0.25      0.50      0.33      4778
    weighted avg       0.25      0.50      0.33      4778

/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
```

✓ 0s completed at 9:02 PM