

```

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

```

```
df.head()
```

	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	

	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	

	Estimated Salary	Churn
0	101348.88	1
1	112542.58	0
2	113931.57	1
3	93826.63	0
4	79084.10	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
```

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

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

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

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

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

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

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

```
Num Of Products
```

```
1    5084
```

```
2    4590
```

```
3     266
```

```
4      60
```

```
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
```

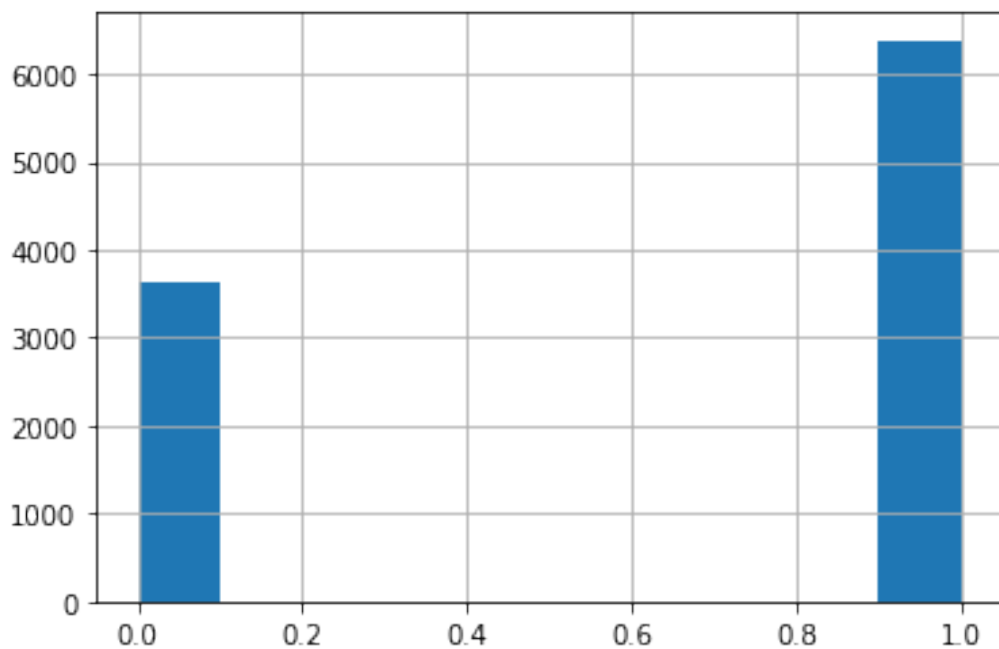
```
1      500
```

```
Name: Churn, dtype: int64
```

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f03b055d110>
```



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

Balance \ Churn Geography		Surname	CreditScore	Gender	Age	Tenure	
0	0	4204	4204	4204	4204	4204	4204
	1	1695	1695	1695	1695	1695	1695
	2	2064	2064	2064	2064	2064	2064
1	0	810	810	810	810	810	810
	1	814	814	814	814	814	814
	2	413	413	413	413	413	413

Churn	Geography	Num Of Products	Has Credit Card	Is Active Member \
0	0	4204	4204	4204
	1	1695	1695	1695
	2	2064	2064	2064
1	0	810	810	810
	1	814	814	814
	2	413	413	413

Estimated Salary Zero Balance

Churn	Geography		
0	0	4204	4204
	1	1695	1695
	2	2064	2064
1	0	810	810
	1	814	814
	2	413	413

```
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')
```

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

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

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

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

```
## Undersampling and Oversampling
```

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

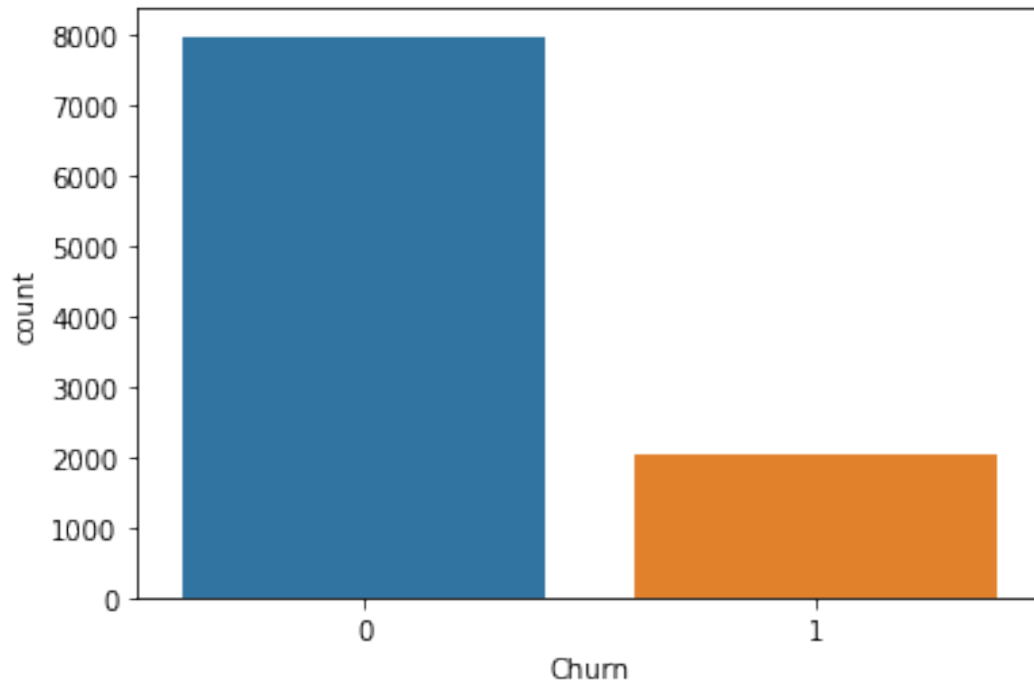
```
0    7963
```

```
1    2037
```

```
Name: Churn, dtype: int64
```

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f03b02d0710>
```



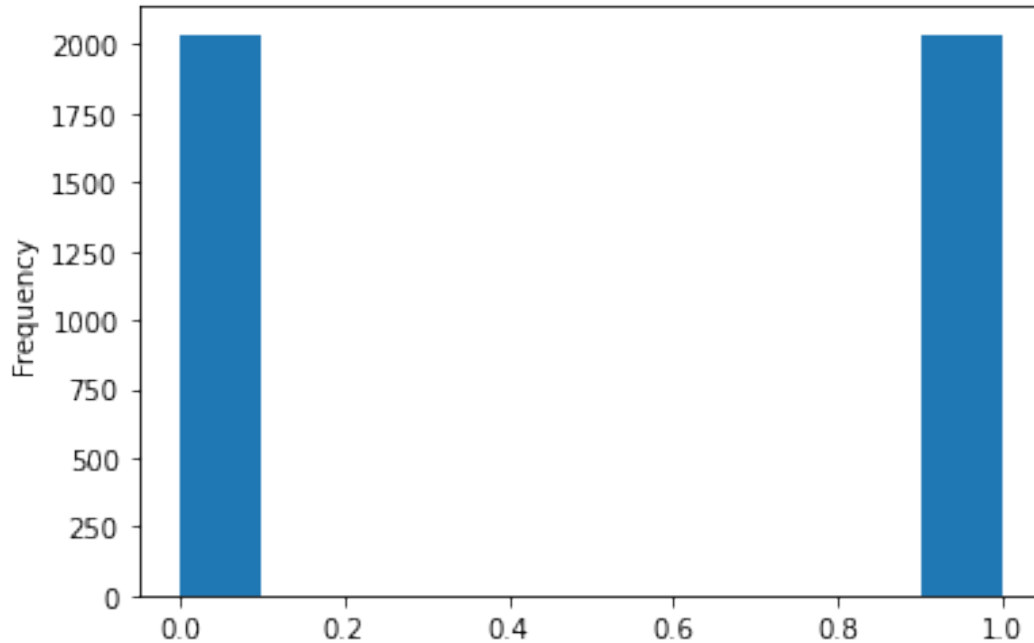
```
X.shape,y.shape
((10000, 11), (10000,))
```

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()
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')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f03b0236310>



```
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_ros.value_counts()
```

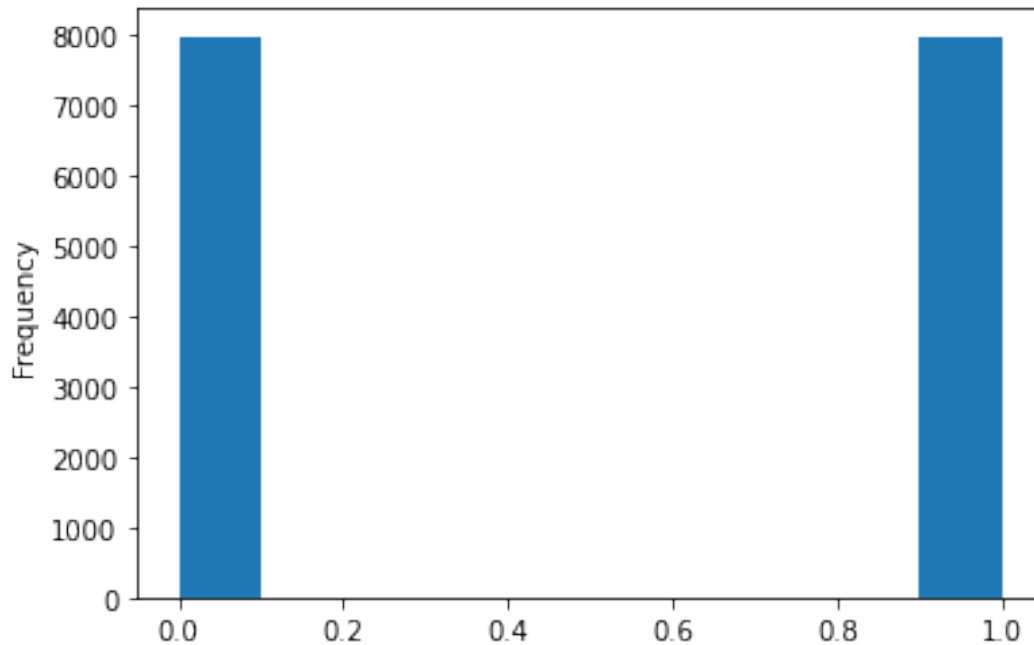
```
1    7963
```

```
0    7963
```

```
Name: Churn, dtype: int64
```

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

<matplotlib.axes._subplots.AxesSubplot at 0x7f03b02320d0>



```
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 Under Sampling data

```
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 Over Sampling Data

```
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)
```

Standardizing Data

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

```
ss = StandardScaler()
```

```
X_train[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']]  
= ss.fit_transform(X_train[['CreditScore', 'Age',  
'Tenure','Balance','Estimated Salary']])
```

```
X_test[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']]  
= ss.fit_transform(X_test[['CreditScore', 'Age',  
'Tenure','Balance','Estimated Salary']])
```



```
X_train_rus[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']] = ss.fit_transform(X_train_rus[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']])
```

```
X_test_rus[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']] = ss.fit_transform(X_test_rus[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']])
```

```
X_train_ros[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']] = ss.fit_transform(X_train_ros[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']])
```

```
X_test_ros[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']] = ss.fit_transform(X_test_ros[['CreditScore', 'Age', 'Tenure','Balance','Estimated Salary']])
```

Support Vector Classifier

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

```
svc=SVC()
```

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

```
SVC()
```

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

Model Accuracy

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

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

```
array([[2380,  34],
       [ 410, 176]])
```

```
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.85	0.99	0.91	2414
1	0.84	0.30	0.44	586
accuracy			0.85	3000
macro avg	0.85	0.64	0.68	3000
weighted avg	0.85	0.85	0.82	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= 1.8s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total
time= 1.8s
[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= 1.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 1.5s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 1.5s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time= 1.6s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time= 1.6s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time= 1.1s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time= 1.1s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 1.3s
[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= 1.5s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total
time= 1.5s
[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= 1.2s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 1.2s

```

```

GridSearchCV(cv=2, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'class_weight':

```

```

['balanced'],
                                'gamma': [1, 0.1, 0.01], 'kernel': ['rbf']},
                                verbose=2)
print(grid.best_estimator_)
SVC(C=1, class_weight='balanced', gamma=1)
grid_prediction = grid.predict(X_test)
confusion_matrix(y_test,grid_prediction)
array([[2121, 293],
       [ 273, 313]])
print(classification_report(y_test,grid_prediction))

```

	precision	recall	f1-score	support
0	0.89	0.88	0.88	2414
1	0.52	0.53	0.53	586
accuracy			0.81	3000
macro avg	0.70	0.71	0.70	3000
weighted avg	0.81	0.81	0.81	3000

##Model With Random Under sampling

```

svc_rus = SVC()
svc_rus.fit(X_train_rus,y_train_rus)
SVC()
y_pred_rus = svc_rus.predict(X_test_rus)
confusion_matrix(y_test_rus,y_pred_rus)
array([[489, 138],
       [181, 415]])
print(classification_report(y_test_rus,y_pred_rus))

```

	precision	recall	f1-score	support
0	0.73	0.78	0.75	627
1	0.75	0.70	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

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= 0.3s  
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total  
time= 0.3s  
[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= 0.2s  
[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= 0.3s  
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total  
time= 0.3s  
[CV] END ....C=1, class_weight=balanced, gamma=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=0.1, 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= 0.3s  
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total  
time= 0.3s  
[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(cv=2, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'class_weight':
['balanced'],
                        'gamma': [1, 0.1, 0.01], 'kernel': ['rbf']},
             verbose=2)

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([[490, 137],
       [185, 411]])
print(classification_report(y_test_rus,grid_predictions_rus))

```

	precision	recall	f1-score	support
0	0.73	0.78	0.75	627
1	0.75	0.69	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

Random Over Sampling Model

```

svc_ros =SVC()

svc_ros.fit(X_train_ros,y_train_ros)

SVC()

y_pred_ros = svc_ros.predict(X_test_ros)
confusion_matrix(y_test_ros,y_pred_ros)
array([[1928, 451],
       [ 599, 1800]])
print(classification_report(y_test_ros,y_pred_ros))

```

	precision	recall	f1-score	support
0	0.76	0.81	0.79	2379
1	0.80	0.75	0.77	2399
accuracy			0.78	4778

macro avg	0.78	0.78	0.78	4778
weighted avg	0.78	0.78	0.78	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= 8.7s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total
time= 8.1s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time= 6.3s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time= 5.8s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 6.9s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 5.7s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time= 3.7s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time= 3.8s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time= 2.8s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time= 2.7s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 3.3s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 3.1s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total
time= 3.4s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total
time= 3.4s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total
time= 3.0s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total
time= 2.9s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 3.0s
```

```
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total
time= 3.0s
GridSearchCV(cv=2, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'class_weight':
['balanced'],
                        'gamma': [1, 0.1, 0.01], 'kernel': ['rbf']},
             verbose=2)
print(grid_ros.best_estimator_)
SVC(C=10, class_weight='balanced', gamma=1)
grid_pred_ros = grid_ros.predict(X_test_ros)
confusion_matrix(y_test_ros,grid_pred_ros)
array([[2084, 295],
       [ 59, 2340]])
print(classification_report(y_test_ros,grid_pred_ros))
```

	precision	recall	f1-score	support
0	0.97	0.88	0.92	2379
1	0.89	0.98	0.93	2399
accuracy			0.93	4778
macro avg	0.93	0.93	0.93	4778
weighted avg	0.93	0.93	0.93	4778

This Model has the best Accuracy