Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Importing the dataset

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

Get Information of Dataframe

```
df.info()   #gives column name, count,  not null category, D-type(data
type)

<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.describe()   #gives the linear relation of each column with another
column

         CustomerId   CreditScore           Age        Tenure
Balance  \
count  1.000000e+04  10000.000000  10000.000000  10000.000000
10000.000000
mean   1.569094e+07    650.528800     38.921800      5.012800
76485.889288
std    7.193619e+04     96.653299     10.487806      2.892174
```

```
62397.405202
min    1.556570e+07    350.000000     18.000000      0.000000
0.000000
25%    1.562853e+07    584.000000     32.000000      3.000000
0.000000
50%    1.569074e+07    652.000000     37.000000      5.000000
97198.540000
75%    1.575323e+07    718.000000     44.000000      7.000000
127644.240000
max    1.581569e+07    850.000000     92.000000     10.000000
250898.090000

       Num Of Products  Has Credit Card  Is Active Member  Estimated
Salary  \
count     10000.000000      10000.00000      10000.000000
10000.000000
mean          1.530200          0.70550          0.515100
100090.239881
std           0.581654          0.45584          0.499797
57510.492818
min           1.000000          0.00000          0.000000
11.580000
25%           1.000000          0.00000          0.000000
51002.110000
50%           1.000000          1.00000          1.000000
100193.915000
75%           2.000000          1.00000          1.000000
149388.247500
max           4.000000          1.00000          1.000000
199992.480000

              Churn
count  10000.000000
mean       0.203700
std        0.402769
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
```

```
df.head(10)
```

```
             Surname  CreditScore  Geography  Gender  Age  Tenure
Balance  \
CustomerId

15634602    Hargrave          619          2       1   42       2
0.00
15647311        Hill          608          0       1   41       1
```

```
83807.86
15619304        Onio          502          2       1   42       8
159660.80
15701354        Boni          699          2       1   39       1
0.00
15737888    Mitchell          850          0       1   43       2
125510.82
15574012         Chu          645          0       0   44       8
113755.78
15592531    Bartlett          822          2       0   50       7
0.00
15656148      Obinna          376          1       1   29       4
115046.74
15792365          He          501          2       0   44       4
142051.07
15592389          H?          684          2       0   27       2
134603.88

            Num Of Products  Has Credit Card  Is Active Member  \
CustomerId
15634602                  0                1                 1
15647311                  0                0                 1
15619304                  1                1                 0
15701354                  1                0                 0
15737888                  0                1                 1
15574012                  1                1                 0
15592531                  1                1                 1
15656148                  1                1                 0
15792365                  1                0                 1
15592389                  0                1                 1

            Estimated Salary  Churn
CustomerId
15634602           101348.88      1
15647311           112542.58      0
15619304           113931.57      1
15701354            93826.63      0
15737888            79084.10      0
15574012           149756.71      1
15592531            10062.80      0
15656148           119346.88      1
15792365            74940.50      0
15592389            71725.73      0
```

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

```
0    5084
1    4916
Name: Num Of Products, dtype: int64
```

```python
df.isnull().sum() #(df.isna().sum() gives same result)
#gives the sum of all null values columns-wise
```

```
CustomerId          0
Surname             0
CreditScore         0
Geography           0
Gender              0
Age                 0
Tenure              0
Balance             0
Num Of Products     0
Has Credit Card     0
Is Active Member    0
Estimated Salary    0
Churn               0
dtype: int64
```

```python
df.nunique()   #gives total no. of unique entries
```

```
CustomerId          10000
Surname              2932
CreditScore           460
Geography               3
Gender                  2
Age                    70
Tenure                 11
Balance              6382
Num Of Products         4
Has Credit Card         2
Is Active Member        2
Estimated Salary     9999
Churn                   2
dtype: int64
```

```python
df.columns  #give column names in the dataframe
```

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

```python
df.shape
```

```
(10000, 13)
```

```python
df.duplicated('CustomerId').sum()
```

```
0
```

```python
df=df.set_index('CustomerId')
```

#Encoding

```
df.dtypes
```

```
Surname              object
CreditScore           int64
Geography            object
Gender               object
Age                   int64
Tenure                int64
Balance             float64
Num Of Products       int64
Has Credit Card       int64
Is Active Member      int64
Estimated Salary    float64
Churn                 int64
dtype: object
```

```
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)    #we
are clubbing 2,3,4 as 3 and 4 product has very less data
```

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

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

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

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

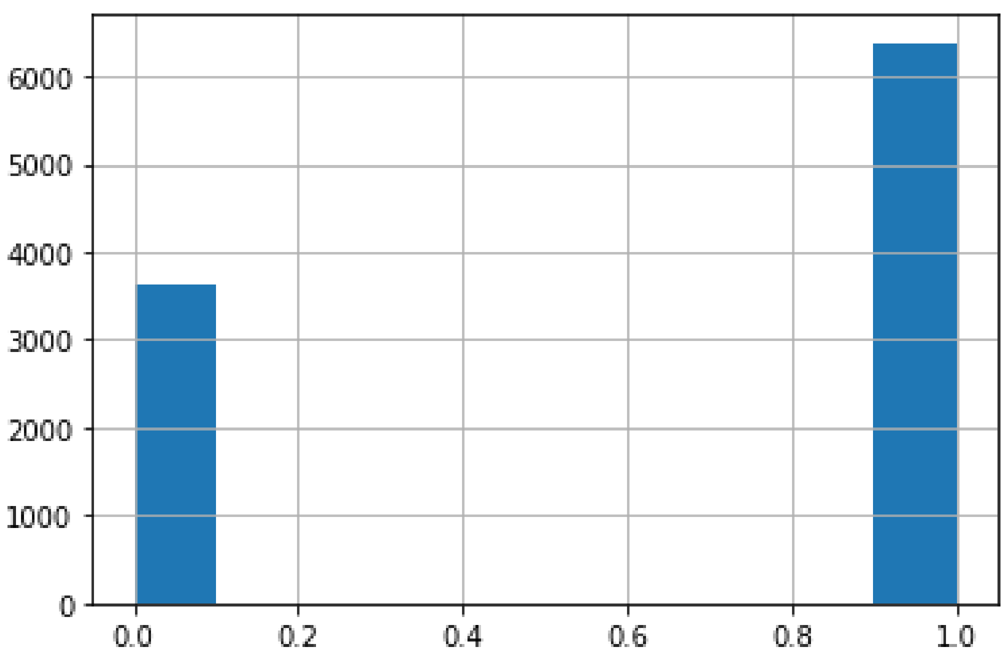
```
df[df['Balance']==0]['Churn'].value_counts()
```

```
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 0x7fb9bcb2e250>
```




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

```
                  Surname  CreditScore  Gender   Age  Tenure
Balance  \
Churn Geography

0     0              2064         2064    2064  2064    2064     2064

      1              1695         1695    1695  1695    1695     1695

      2              4204         4204    4204  4204    4204     4204

1     0               413          413     413   413     413      413

      1               814          814     814   814     814      814

      2               810          810     810   810     810      810


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

                  Estimated Salary  Zero Balance
Churn Geography
0     0                       2064          2064
      1                       1695          1695
      2                       4204          4204
1     0                        413           413
      1                        814           814
      2                        810           810
```

#Define Label and Features

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

y=df['Churn']

X.shape, y.shape

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

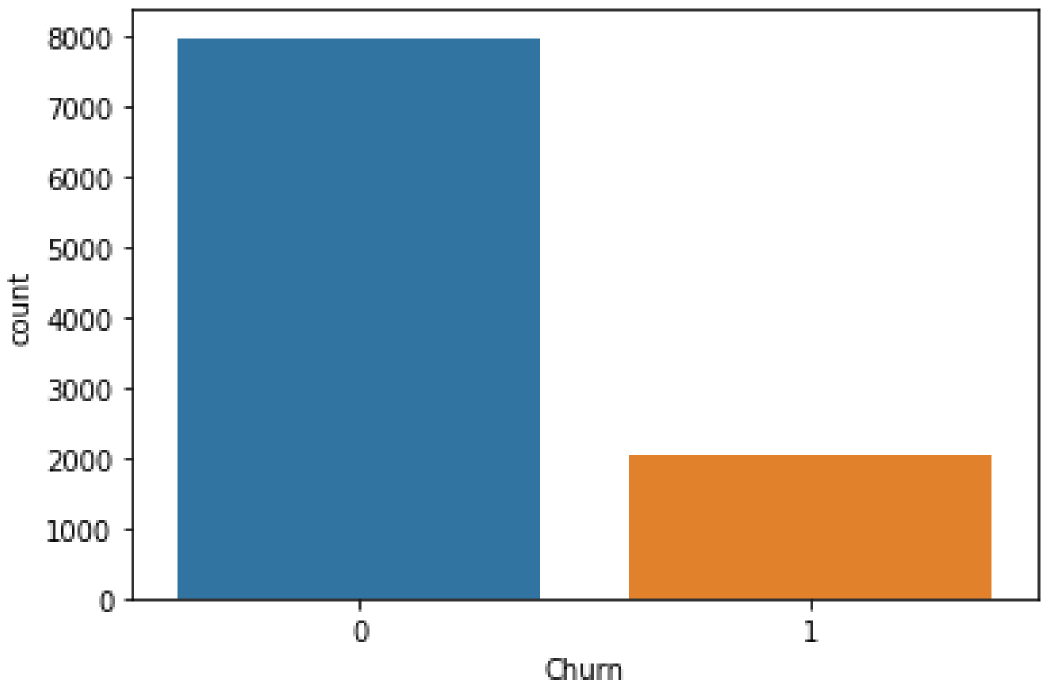
Handling Imbalance Data

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

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

```
  sns.countplot(x='Churn',data=df)    #the target 'Churn' have
imbalanced data - no of examples in class of target is not same
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb9bc81c550>
```




```
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()
```

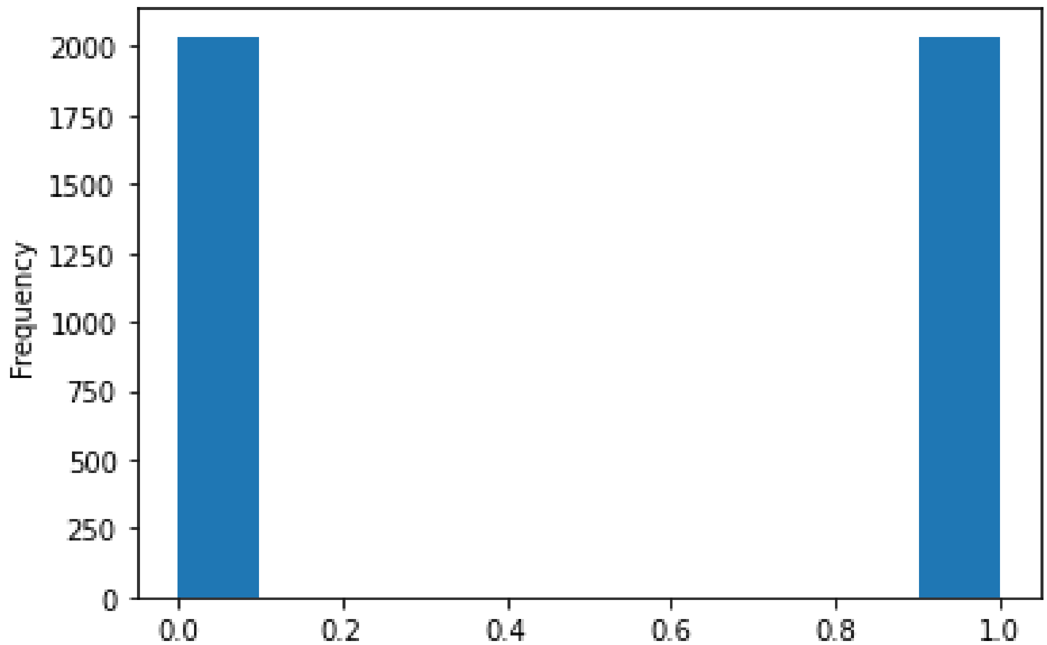
```
0    7963
1    2037
Name: Churn, dtype: int64
```

```
y_rus.value_counts()
```

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

```
y_rus.plot(kind='hist')   #Now both the class of Target have same no. of examples
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb9bab038d0>
```



#Random Over Sampling

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

```
ros=RandomOverSampler(random_state=23)
```

```
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()
```

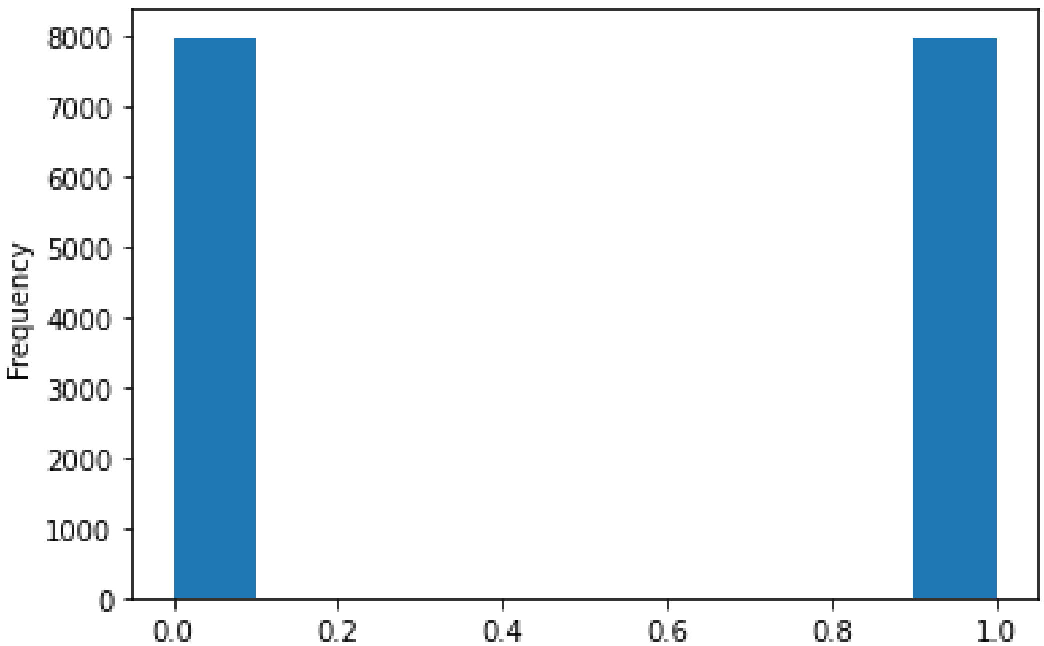
```
0    7963
1    2037
Name: Churn, dtype: int64
```

```
y_ros.value_counts()
```

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

```
y_ros.plot(kind='hist')    #Now both the class of Target have same no. of examples
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb9baa7d110>
```



Splitting the dataset into the Training set and Test set

```
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 = 1)
```

##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, random_state = 1)
```

##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, random_state = 1)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler    #STANDARDIZATION
sc = StandardScaler()
```

```
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')
```

##Standardize Original Data

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

```
X_test[['CreditScore','Gender', 'Age', 'Tenure',
       'Balance', 'Estimated
Salary']]=sc.transform(X_test[['CreditScore','Gender', 'Age',
'Tenure',
       'Balance', 'Estimated Salary']])
```

##Standardize Random Under Sample Data

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

```
X_test_rus[['CreditScore','Gender', 'Age', 'Tenure',
       'Balance', 'Estimated
Salary']]=sc.transform(X_test_rus[['CreditScore','Gender', 'Age',
'Tenure',
       'Balance', 'Estimated Salary']])
```

##Standardize Random Over Sample Data

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

```
X_test_ros[['CreditScore','Gender', 'Age', 'Tenure',
       'Balance', 'Estimated
Salary']]=sc.transform(X_test_ros[['CreditScore','Gender', 'Age',
```

```
'Tenure',
       'Balance', 'Estimated Salary']])
```

#Support Vector Machine Classifier

```
from sklearn.svm import SVC

svc=SVC()

svc.fit(X_train,y_train)

SVC()

y_pred= svc.predict(X_test)

y_pred.shape

(3000,)

y_pred

array([0, 0, 0, ..., 0, 0, 0])
```

#Model Evaluation

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

confusion_matrix(y_test,y_pred)

array([[2333,   40],
       [ 455,  172]])

accuracy_score(y_test,y_pred)

0.835

print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.84      0.98      0.90      2373
           1       0.81      0.27      0.41       627

    accuracy                           0.83      3000
   macro avg       0.82      0.63      0.66      3000
weighted avg       0.83      0.83      0.80      3000
```

#GridSearch CV on original Data

```
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=   1.7s
[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=   1.3s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   1.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   1.4s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time=   1.5s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time=   1.5s
[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=   1.2s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   1.2s
[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=   1.2s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   1.2s
[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=0.1, class_weight='balanced', gamma=1)
```

```
grid_predictions= grid.predict(X_test)
```

```
confusion_matrix(y_test,grid_predictions)
```

```
array([[2000,  373],
       [ 290,  337]])
```

```
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.86      2373
           1       0.47      0.54      0.50       627

    accuracy                           0.78      3000
   macro avg       0.67      0.69      0.68      3000
weighted avg       0.79      0.78      0.78      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([[471, 151],
       [159, 442]])
```

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

```
              precision    recall  f1-score   support

           0       0.75      0.76      0.75       622
           1       0.75      0.74      0.74       601

    accuracy                           0.75      1223
   macro avg       0.75      0.75      0.75      1223
weighted avg       0.75      0.75      0.75      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=   0.5s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total
time=   0.8s
[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=   0.2s
[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=   0.4s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time=   0.6s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time=   0.7s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   0.7s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   0.4s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   0.4s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   0.7s
[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=   0.6s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   0.7s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   0.3s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   0.4s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   0.4s

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([[476, 146],
       [155, 446]])
```

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

```
              precision    recall  f1-score   support

           0       0.75      0.77      0.76       622
           1       0.75      0.74      0.75       601

    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

```
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([[1774,  548],
       [ 602, 1854]])
```

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

```
              precision    recall  f1-score   support

           0       0.75      0.76      0.76      2322
           1       0.77      0.75      0.76      2456

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

#Hyperparameter Tuning

```
param_grid_ros= {'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=   6.7s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total
time=   4.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   3.2s
[CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   3.1s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   3.6s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   3.5s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time=   3.8s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total
time=   3.7s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   2.9s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   2.8s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   3.2s
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total
time=   3.0s
[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=   3.3s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total
time=   3.1s
[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=   2.9s

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_predictions_ros= grid_ros.predict(X_test_ros)
```

```
confusion_matrix(y_test_ros,grid_predictions_ros)
```

```
array([[1999,  323],
       [  44, 2412]])
```

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

```
              precision    recall  f1-score   support

           0       0.98      0.86      0.92      2322
           1       0.88      0.98      0.93      2456

    accuracy                           0.92      4778
   macro avg       0.93      0.92      0.92      4778
weighted avg       0.93      0.92      0.92      4778
```

#Comparing Accuracy per Step

```
print(classification_report(y_test,y_pred))   #original data
```

```
              precision    recall  f1-score   support

           0       0.84      0.98      0.90      2373
           1       0.81      0.27      0.41       627

    accuracy                           0.83      3000
   macro avg       0.82      0.63      0.66      3000
weighted avg       0.83      0.83      0.80      3000
```

```
print(classification_report(y_test,grid_predictions))   #original data
Hypertuned
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.86      2373
           1       0.47      0.54      0.50       627
```

```
    accuracy                           0.78      3000
   macro avg       0.67      0.69      0.68      3000
weighted avg       0.79      0.78      0.78      3000
```

```
print(classification_report(y_test_rus,y_pred_rus))   #under sampling
data
```

```
              precision    recall  f1-score   support

           0       0.75      0.76      0.75       622
           1       0.75      0.74      0.74       601

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

```
print(classification_report(y_test_rus,grid_predictions_rus)) #under
sampling hypertuned
```

```
              precision    recall  f1-score   support

           0       0.75      0.77      0.76       622
           1       0.75      0.74      0.75       601

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

```
print(classification_report(y_test_ros,y_pred_ros)) #over sampling
```

```
              precision    recall  f1-score   support

           0       0.75      0.76      0.76      2322
           1       0.77      0.75      0.76      2456

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

```
print(classification_report(y_test_ros,grid_predictions_ros)) #over
sampling hypertuned
```

```
              precision    recall  f1-score   support

           0       0.98      0.86      0.92      2322
           1       0.88      0.98      0.93      2456
```

accuracy			0.92	4778
macro avg	0.93	0.92	0.92	4778
weighted avg	0.93	0.92	0.92	4778