```
import numpy as np
```

```
import pandas as pd
```

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

```
import seaborn as sns
```

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

```
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.head()
```

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Pr
0	15634602	Hargrave	619	France	Female	42	2	0.00	
1	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	15619304	Onio	502	France	Female	42	8	159660.80	
3	15701354	Boni	699	France	Female	39	1	0.00	

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

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

```
#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
```

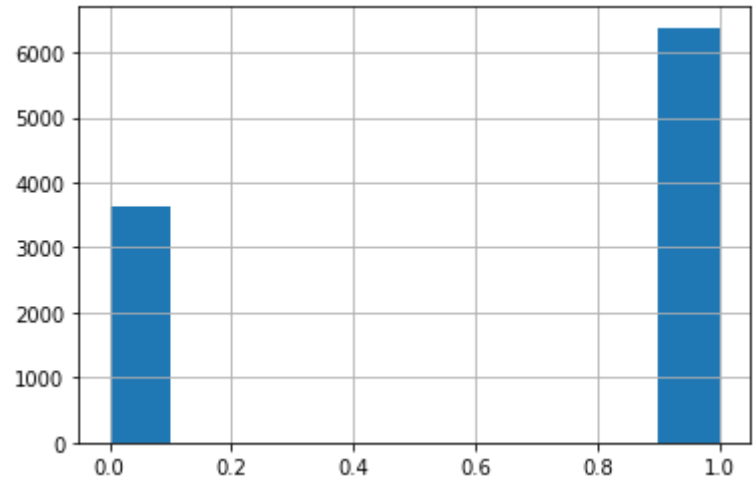
```
df.loc[(df['Balance']==0),'Churn'].value_counts()
```

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

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe7ae2348d0>
```




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

```
#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
```

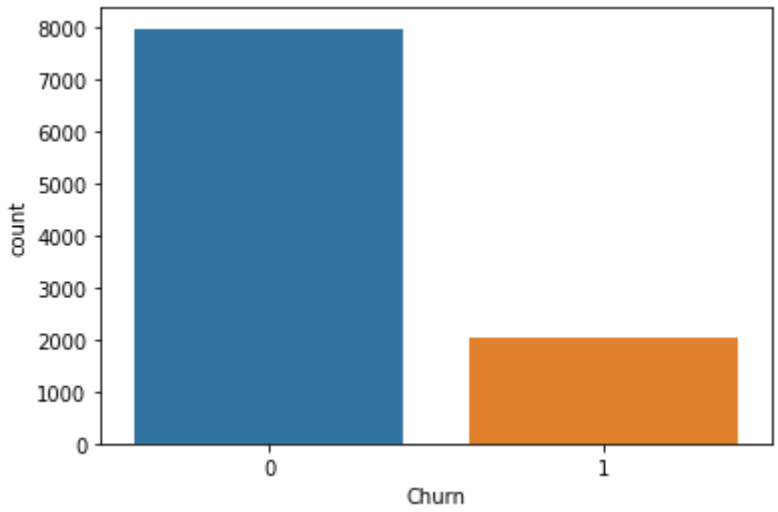
```
((10000, 11), (10000,))
```

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

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe7ae2404d0>
```




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

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

```
#random under Sanpling
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()
```

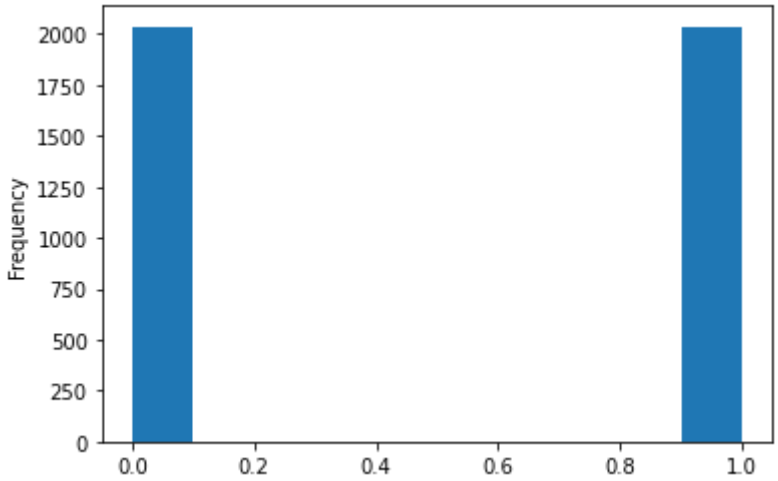
```
0    7963
1    2037
Name: Churn, dtype: int64
```

```
y_rus.value_counts()
```

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe7ac00f390>
```




```
#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
```

```
#Train test splt
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=2529) #split
```

```
X_train_rus,X_test_rus,y_train_rus,y_test_rus=train_test_split(X_rus,y_rus,test_size=0.3,r
```

```
X_train_ros,X_test_ros,y_train_ros,y_test_ros=train_test_split(X_ros,y_ros,test_size=0.3,r
```

```
#standardize Features
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
```

```
#standardize original dataset
```

```
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 inder 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()
```

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

```
#Model accuracy
from sklearn.metrics import confusion_matrix,classification_report
```

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

```
array([[2390,   24],
       [ 459,  127]])
```

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

```
                  precision    recall  f1-score   support

               0       0.84      0.99      0.91      2414
               1       0.84      0.22      0.34       586

        accuracy                           0.84      3000
       macro avg       0.84      0.60      0.63      3000
    weighted avg       0.84      0.84      0.80      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=   1.7s
[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=   1.4s
```

```
[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=   1.1s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.2s
[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=   1.4s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.4s
[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([[2016,  398],
       [ 255,  331]])
```

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

```
                  precision    recall  f1-score   support

               0       0.89      0.84      0.86      2414
               1       0.45      0.56      0.50       586

        accuracy                           0.78      3000
       macro avg       0.67      0.70      0.68      3000
    weighted avg       0.80      0.78      0.79      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)
```

```
#Model Accuracy


confusion_matrix(y_test_rus,y_pred_rus)
```

```
array([[457, 170],
       [202, 394]])
```

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

```
                  precision    recall  f1-score   support

               0       0.69      0.73      0.71       627
               1       0.70      0.66      0.68       596

        accuracy                           0.70      1223
       macro avg       0.70      0.69      0.70      1223
    weighted avg       0.70      0.70      0.70      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=   0.3s
[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=   0.2s
[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([[462, 165],
       [206, 390]])
```

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

```
                  precision    recall  f1-score   support

               0       0.69      0.74      0.71       627
               1       0.70      0.65      0.68       596

        accuracy                           0.70      1223
       macro avg       0.70      0.70      0.70      1223
    weighted avg       0.70      0.70      0.70      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)
```

```
#model accuracy
```

```
confusion_matrix(y_test_ros,y_pred_ros)
```

```
array([[1807,  572],
       [ 744, 1655]])
```

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

```
                  precision    recall  f1-score   support

               0       0.71      0.76      0.73      2379
               1       0.74      0.69      0.72      2399

        accuracy                           0.72      4778
       macro avg       0.73      0.72      0.72      4778
```

```
weighted avg       0.73      0.72      0.72      4778
```

```
#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=   4.3s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   4.3s
[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=   3.2s
[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=   3.6s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.5s
[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=   2.9s
[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=   3.5s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   3.5s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   3.2s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   3.2s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   3.1s
[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_predictions_ros=grid_ros.predict(X_test_ros)


confusion_matrix(y_test_ros,grid_predictions_ros)
```

```
array([[1990,  389],
       [  92, 2307]])
```

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

```
                precision    recall  f1-score   support

             0       0.96      0.84      0.89      2379
             1       0.86      0.96      0.91      2399

      accuracy                           0.90      4778
```

```
           macro avg       0.91      0.90      0.90      4778
        weighted avg       0.91      0.90      0.90      4778
```

✓ 0s completed at 12:15 AM