# Bank Customer Churn Model

## Objective

The objective of this project is to build a machine learning model to predict customer churn in a bank using the Bank Churn Modeling dataset. The model will be trained, validated, and optimized using various preprocessing techniques and Support Vector Classification (SVC).

## Import Library

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

## Describe Data

```
df = pd.read_csv('https://raw.githubusercontent.com/YBIFoundation/Dataset/main/Bank%20Churn%20Modelling.csv')
```

## Data Visualization

```
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 | Num Of Products | Has Credit Card | Is Active Member | Estimated Salary | Churn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 | 1 | 101348.88 | 1 |
| 1 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 | 1 | 112542.58 | 0 |
| 2 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 | 0 | 113931.57 | 1 |
| 3 | 15701354 | Boni | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 | 0 | 93826.63 | 0 |

Next steps: Generate code with df | View recommended plots

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

```
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':{'Female':1,'Male':0}} , inplace=True)
```

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

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

```
print(df['Has Credit Card'].value_counts())
df.replace({'Has Credit Card':{1:0,0:1}}, inplace=True)

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

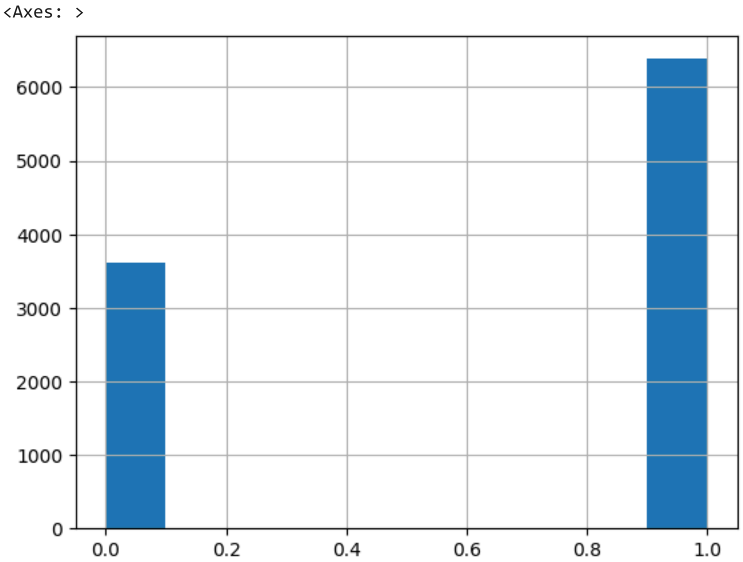
```
Has Credit Card
1    7055
0    2945
Name: count, dtype: int64
Is Active Member
1    5151
0    4849
Name: count, dtype: int64
```

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

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

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

```
<Axes: >
```



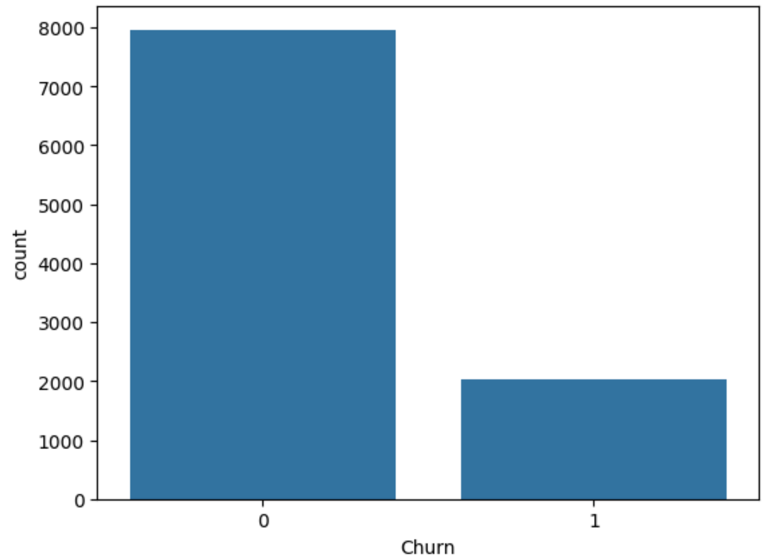
## Define Target Variable (y) and Feature Variables (X)

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

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

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

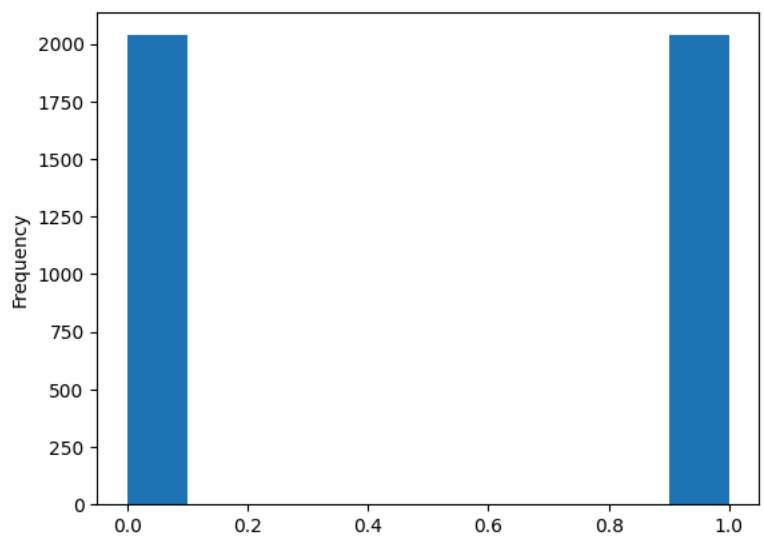
```
df['Churn'].value_counts()
sns.countplot(x='Churn', data=df);
```




RANDOM UNDERSAMPLING

```
from imblearn.under_sampling import RandomUnderSampler
rus=RandomUnderSampler(random_state=2529)
x_rus,y_rus =rus.fit_resample(x,y)
x.shape, y.shape , x_rus.shape, y_rus.shape
x.value_counts() , y.value_counts()
y_rus.value_counts()
y_rus.plot(kind='hist')
```

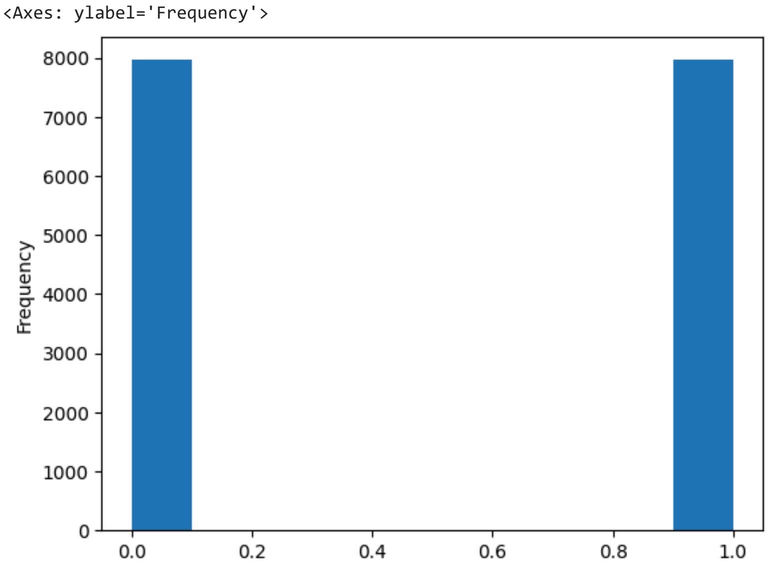
```
<Axes: ylabel='Frequency'>
```




random oversampling

```
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
y.value_counts()
y_ros.value_counts()
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,random_state=2529)
```

```
x_train_rus,x_test_rus,y_train_rus, y_test_rus=train_test_split(x_rus,y_rus)  #split Random Under sample data
```

```
x_train_ros,x_test_ros,y_train_ros, y_test_ros=train_test_split(x_ros,y_ros)  #split Random over sample data
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((7500, 11), (2500, 11), (7500,), (2500,))
```

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

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Assuming 'x' and 'y' are your original data
# Replace 'x' and 'y' with the actual names of your data variables
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)  # Adjust test_size and random_state as needed

sc = StandardScaler()

# Apply scaling to the training and test data
X_train = sc.fit_transform(x_train)  # Scale all features in x_train
X_test = sc.transform(x_test)  # Use the same scaling as applied to x_train
```

## Modeling

```
from sklearn.svm import SVC
svc = SVC()
svc.fit(X_train, y_train)   # Use the correct variable name 'y_train'
Y_pred = svc.predict(X_test)
```

## Prediction

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

```
from sklearn.metrics import confusion_matrix, classification_report

confusion_matrix(y_test, Y_pred)
print(confusion_matrix(y_test, Y_pred))
print("Classification Report:")
print(classification_report(y_test, Y_pred)) # Optionally print a classification report
```

```
[[1585   22]
 [ 292  101]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.99      0.91      1607
           1       0.82      0.26      0.39       393

    accuracy                           0.84      2000
   macro avg       0.83      0.62      0.65      2000
weighted avg       0.84      0.84      0.81      2000
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

```
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=   2.7s
[CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   2.3s
[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=0.1, kernel=rbf; total time=   1.4s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.3s
[CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   2.2s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   2.3s
[CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   2.6s
[CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   3.8s
[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=   2.1s
[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=   1.7s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.8s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.3s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.3s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.3s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.6s
```

▸ **GridSearchCV**

▸ **estimator: SVC**

▸ SVC

```
print(grid.best_estimator_)
grid_predictions = grid.predict(X_test)
confusion_matrix(y_test, grid_predictions)
print(confusion_matrix(y_test, grid_predictions))

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

```
SVC(C=10, class_weight='balanced', gamma=1)
[[1443  164]
 [ 266  127]]
              precision    recall  f1-score   support
```