In [ ]:

# Predicting Custermer Churn for Bank Customers

# Predicting Churn for Bank Customers

## Libraries importing

In [1]:
```
# Data Visualisation Libraries
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
sns.set_style('darkgrid')
```

In [2]:
```
df = pd.read_csv(r"C:\Users\SSD\Downloads\Churn_Modelling.csv")
```

In [3]:
```
df
```

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | B |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | 0. |
| **1** | 2 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 83 |
| **2** | 3 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 15 |
| **3** | 4 | 15701354 | Boni | 699 | France | Female | 39 | 1 | 0. |
| **4** | 5 | 15737888 | Mitchell | 850 | Spain | Female | 43 | 2 | 12 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **9995** | 9996 | 15606229 | Obijiaku | 771 | France | Male | 39 | 5 | 0. |
| **9996** | 9997 | 15569892 | Johnstone | 516 | France | Male | 35 | 10 | 57 |
| **9997** | 9998 | 15584532 | Liu | 709 | France | Female | 36 | 7 | 0. |
| **9998** | 9999 | 15682355 | Sabbatini | 772 | Germany | Male | 42 | 3 | 75 |
| **9999** | 10000 | 15628319 | Walker | 792 | France | Female | 28 | 4 | 13 |

10000 rows × 14 columns

In [4]:
```
df.head()
```

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Bala |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | 0.00 |
| 1 | 2 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 83807 |
| 2 | 3 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 15966 |
| 3 | 4 | 15701354 | Boni | 699 | France | Female | 39 | 1 | 0.00 |
| 4 | 5 | 15737888 | Mitchell | 850 | Spain | Female | 43 | 2 | 1255 |

In [5]:
```
df.shape
```

```
(10000, 14)
```

In [6]:
```
df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1, inplace=True)
df.columns
```

```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

In [7]:
```
df.shape
```

```
(10000, 11)
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CreditScore      10000 non-null  int64  
 1   Geography        10000 non-null  object 
 2   Gender           10000 non-null  object 
 3   Age              10000 non-null  int64  
 4   Tenure           10000 non-null  int64  
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64  
 7   HasCrCard        10000 non-null  int64  
 8   IsActiveMember   10000 non-null  int64  
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64  
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

In [9]:
```
#Thankfully, there are no missing values in our DataFrame.
```

The describe() method gives us a statistical summary of the numerical features:

In [10]:
```
df.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | ma |
|---|---|---|---|---|---|---|---|---|
| **CreditScore** | 10000.0 | 650.528800 | 96.653299 | 350.00 | 584.00 | 652.000 | 718.0000 | 85 |
| **Age** | 10000.0 | 38.921800 | 10.487806 | 18.00 | 32.00 | 37.000 | 44.0000 | 92 |
| **Tenure** | 10000.0 | 5.012800 | 2.892174 | 0.00 | 3.00 | 5.000 | 7.0000 | 10 |
| **Balance** | 10000.0 | 76485.889288 | 62397.405202 | 0.00 | 0.00 | 97198.540 | 127644.2400 | 25 |
| **NumOfProducts** | 10000.0 | 1.530200 | 0.581654 | 1.00 | 1.00 | 1.000 | 2.0000 | 4.0 |
| **HasCrCard** | 10000.0 | 0.705500 | 0.455840 | 0.00 | 0.00 | 1.000 | 1.0000 | 1.0 |
| **IsActiveMember** | 10000.0 | 0.515100 | 0.499797 | 0.00 | 0.00 | 1.000 | 1.0000 | 1.0 |
| **EstimatedSalary** | 10000.0 | 100090.239881 | 57510.492818 | 11.58 | 51002.11 | 100193.915 | 149388.2475 | 19 |
| **Exited** | 10000.0 | 0.203700 | 0.402769 | 0.00 | 0.00 | 0.000 | 0.0000 | 1.0 |

In [11]:
```
#Cheking for missing numbers
```

In [12]:
```
df.isnull().sum()
```

```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

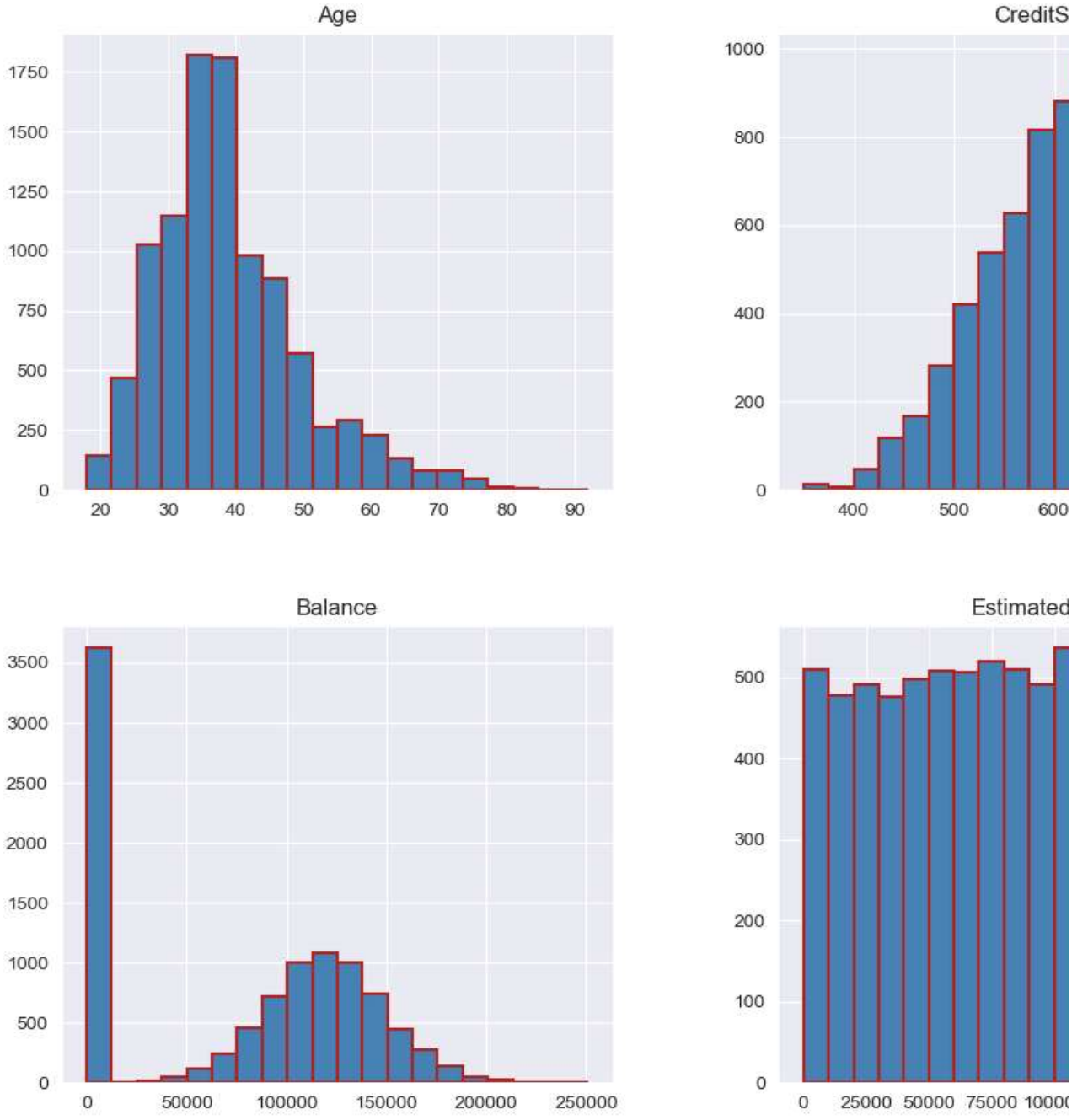
## Visulization

In [13]:
```
continuous = ['Age', 'CreditScore', 'Balance', 'EstimatedSalary']
categorical = ['Geography', 'Gender', 'Tenure', 'NumOfProducts', 'HasCrCard', 'I

print('Continuous: ', ', '.join(continuous))
print('Categorical: ', ', '.join(categorical))
```

```
Continuous:  Age, CreditScore, Balance, EstimatedSalary
Categorical:  Geography, Gender, Tenure, NumOfProducts, HasCrCard, IsActiveMember
```

In [14]:
```
df[continuous].hist(figsize=(12, 10),
                    bins=20,
                    layout=(2, 2),
                    color='steelblue',
                    edgecolor='firebrick',
                    linewidth=1.5);
```



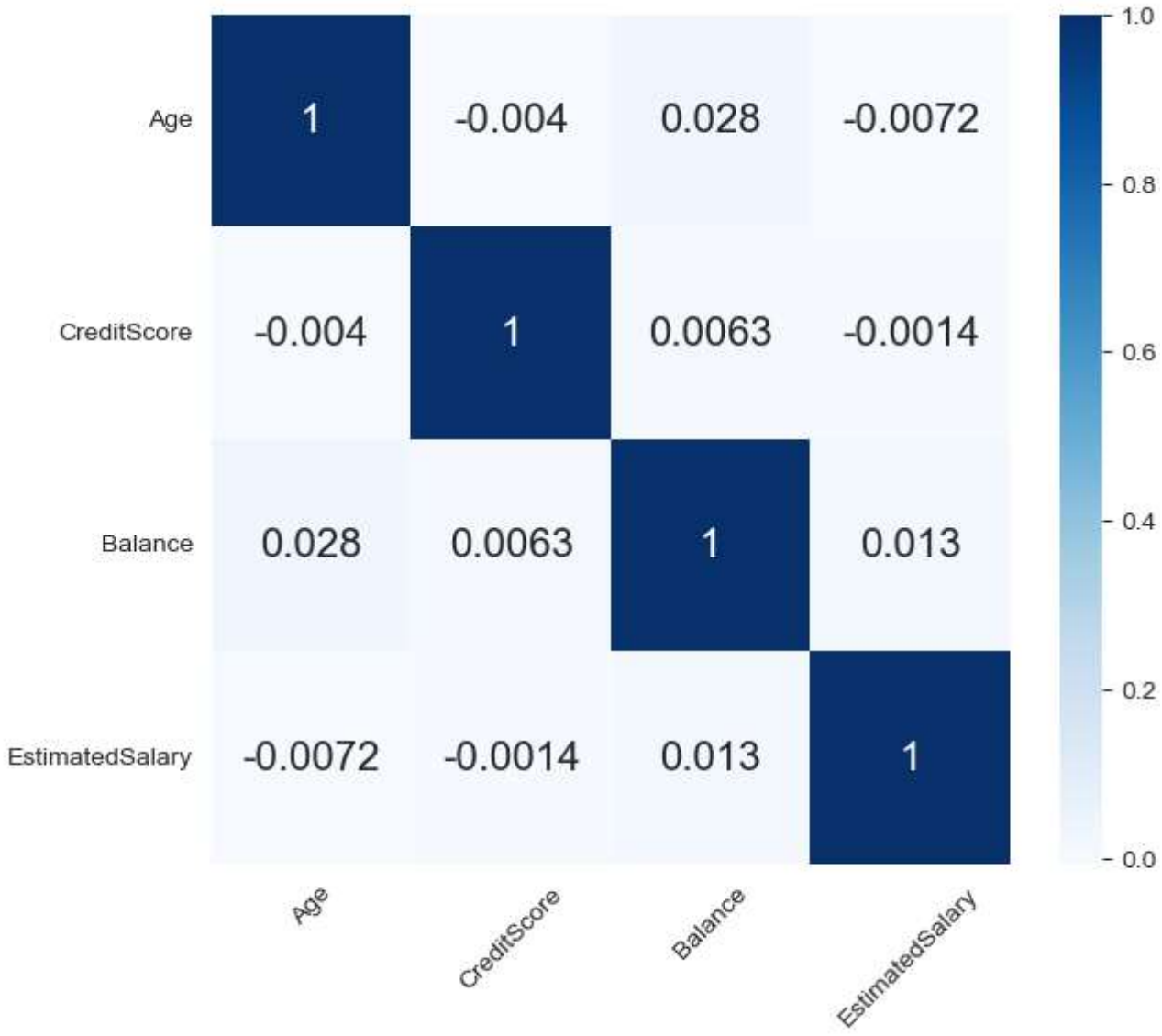

'Age' is slightly tail-heavy, i.e. it extends more further to the right of the median tha
'CreditScore' are above 600, If we ignore the first bin, 'Balance' follows a fairly nor
distribution of 'EstimatedSalary' is more or less uniform and provides little informat

## Looking for Correlations

In [15]:
```
fig, ax = plt.subplots(figsize=(7, 6))

sns.heatmap(df[continuous].corr(),
            annot=True,
            annot_kws={'fontsize': 16},
            cmap='Blues',
            ax=ax)

ax.tick_params(axis='x', rotation=45)
ax.tick_params(axis='y', rotation=360);
```



In [16]:
```
# Create subsets
df_churned = df[df['Exited'] == 1]
df_retained = df[df['Exited'] == 0]
```

In [17]:
```
df_churned
```

| | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard |
|---|---|---|---|---|---|---|---|---|
| **0** | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 |
| **2** | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 |
| **5** | 645 | Spain | Male | 44 | 8 | 113755.78 | 2 | 1 |
| **7** | 376 | Germany | Female | 29 | 4 | 115046.74 | 4 | 1 |
| **16** | 653 | Germany | Male | 58 | 1 | 132602.88 | 1 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **9981** | 498 | Germany | Male | 42 | 3 | 152039.70 | 1 | 1 |
| **9982** | 655 | Germany | Female | 46 | 7 | 137145.12 | 1 | 1 |
| **9991** | 597 | France | Female | 53 | 4 | 88381.21 | 1 | 1 |
| **9997** | 709 | France | Female | 36 | 7 | 0.00 | 1 | 0 |
| **9998** | 772 | Germany | Male | 42 | 3 | 75075.31 | 2 | 1 |

2037 rows × 11 columns

In [18]:
```
df_retained
```

| | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard |
|---|---|---|---|---|---|---|---|---|
| **1** | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 |
| **3** | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 |
| **4** | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | 1 |
| **6** | 822 | France | Male | 50 | 7 | 0.00 | 2 | 1 |
| **8** | 501 | France | Male | 44 | 4 | 142051.07 | 2 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **9993** | 644 | France | Male | 28 | 7 | 155060.41 | 1 | 1 |
| **9994** | 800 | France | Female | 29 | 2 | 0.00 | 2 | 0 |
| **9995** | 771 | France | Male | 39 | 5 | 0.00 | 2 | 1 |
| **9996** | 516 | France | Male | 35 | 10 | 57369.61 | 1 | 1 |
| **9999** | 792 | France | Female | 28 | 4 | 130142.79 | 1 | 1 |

7963 rows × 11 columns

In [19]:

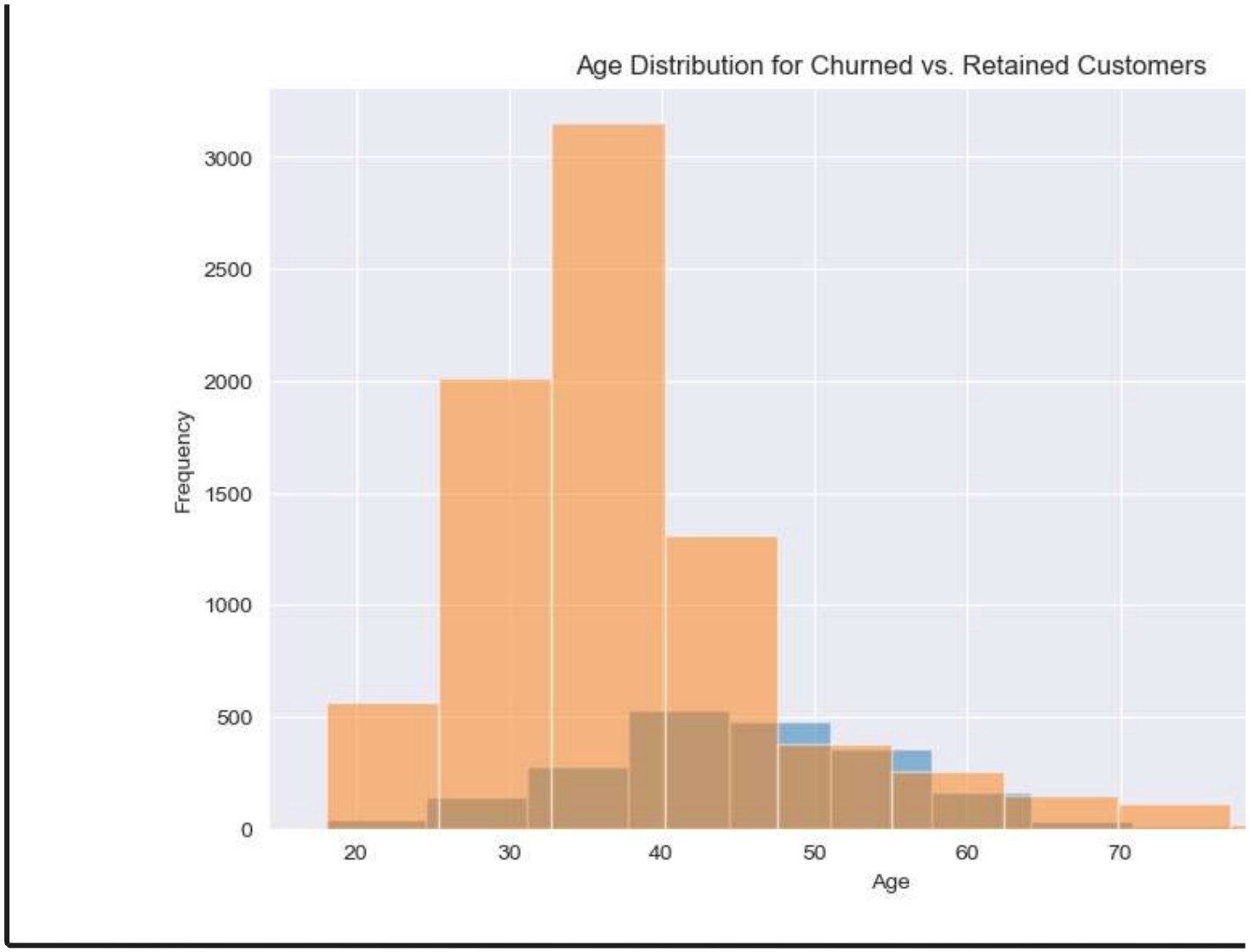
```
# Plotting
plt.figure(figsize=(10, 6))

# Plot age distribution for churned customers
plt.hist(df_churned['Age'], bins=10, alpha=0.5, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['Age'], bins=10, alpha=0.5, label='Retained')

plt.title('Age Distribution for Churned vs. Retained Customers')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```


Age Distribution for Churned vs. Retained Customers
Frequency
3000
2500
2000
1500
1000
500
0
20
30
40
50
60
70
Age

In [20]:

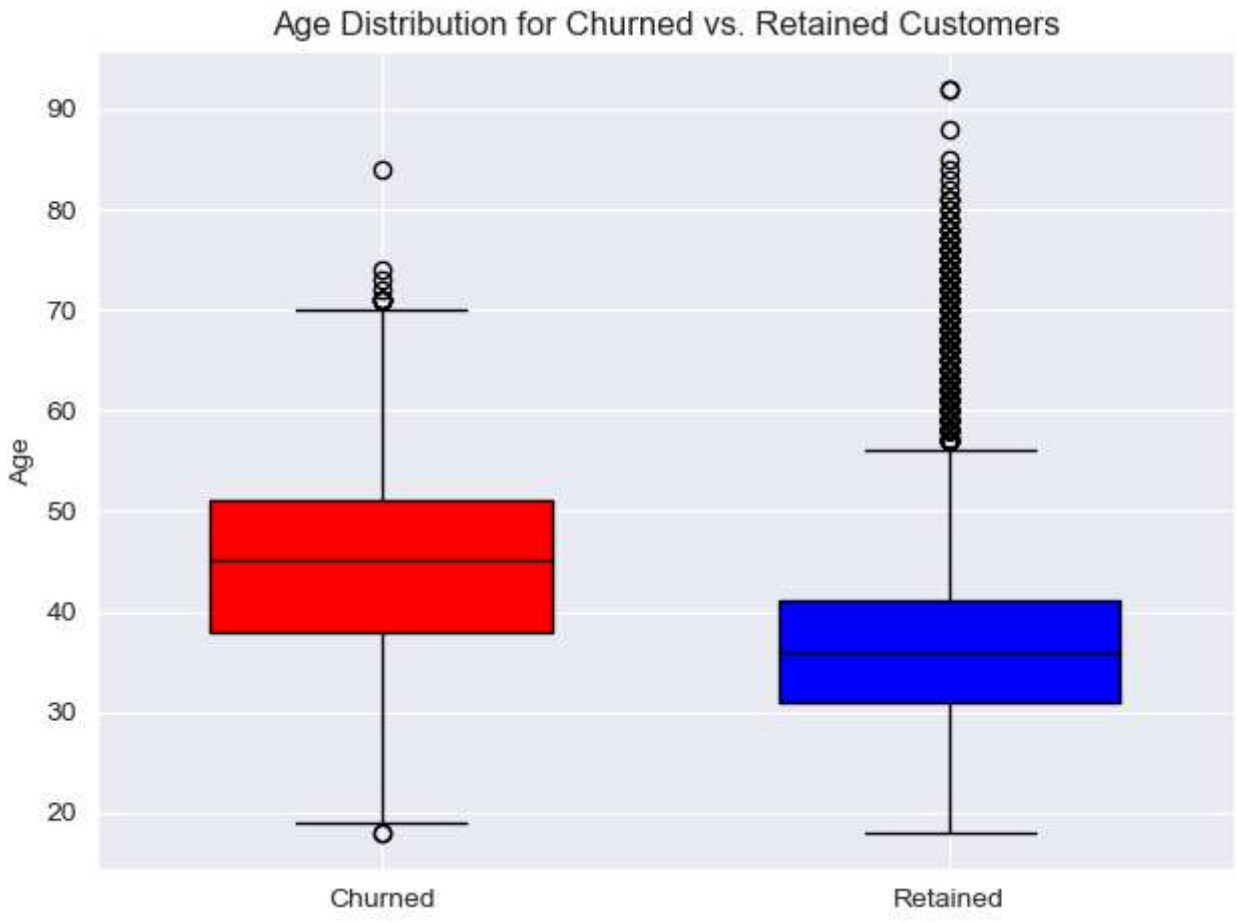
```
# Create boxplot for churned customers' credit scores
plt.boxplot(df_churned['Age'], positions=[1], widths=0.6, patch_artist=True, bo

# Create boxplot for retained customers' credit scores
plt.boxplot(df_retained['Age'], positions=[2], widths=0.6, patch_artist=True, bo

# Adjust x-axis labels
plt.xticks([1, 2], ['Churned', 'Retained'])

plt.title('Age Distribution for Churned vs. Retained Customers')
plt.ylabel('Age')
plt.grid(True)

# Show the plot
plt.tight_layout()
plt.show()
```



In [21]:

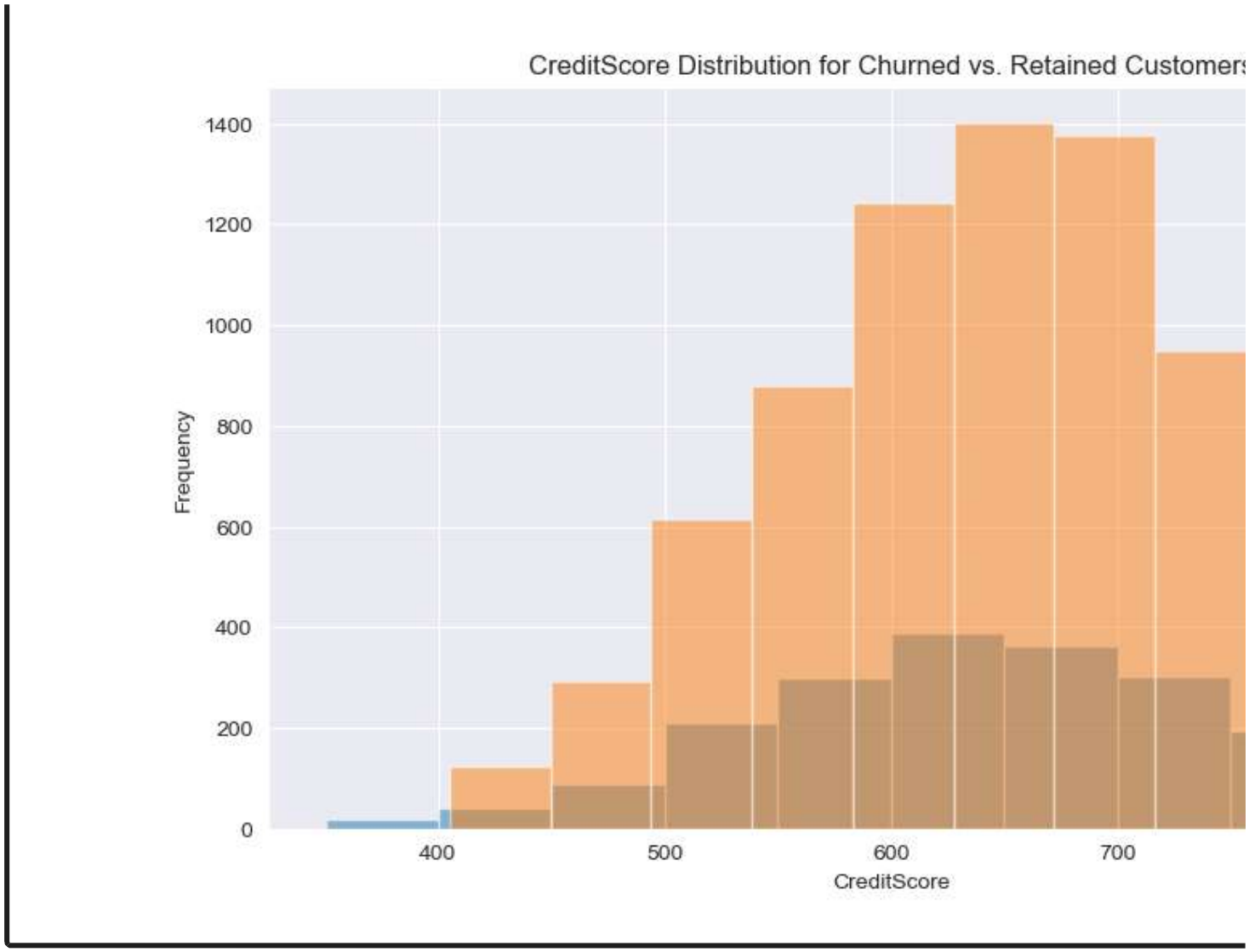
```
# Plotting
plt.figure(figsize=(10, 6))

# Plot age distribution for churned customers
plt.hist(df_churned['CreditScore'], bins=10, alpha=0.5, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['CreditScore'], bins=10, alpha=0.5, label='Retained')

plt.title('CreditScore Distribution for Churned vs. Retained Customers')
plt.xlabel('CreditScore')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```


CreditScore Distribution for Churned vs. Retained Customers
Frequency
0
200
400
600
800
1000
1200
1400
400
500
600
700
CreditScore

In [22]:

```
# Create boxplot for churned customers' credit scores
plt.boxplot(df_churned['CreditScore'], positions=[1], widths=0.6, patch_artist=T

# Create boxplot for retained customers' credit scores
plt.boxplot(df_retained['CreditScore'], positions=[2], widths=0.6, patch_artist=

# Adjust x-axis labels
plt.xticks([1, 2], ['Churned', 'Retained'])

plt.title('CreditScore Distribution for Churned vs. Retained Customers')
plt.ylabel('CreditScore')
plt.grid(True)

# Show the plot
plt.tight_layout()
plt.show()
```

In [23]:

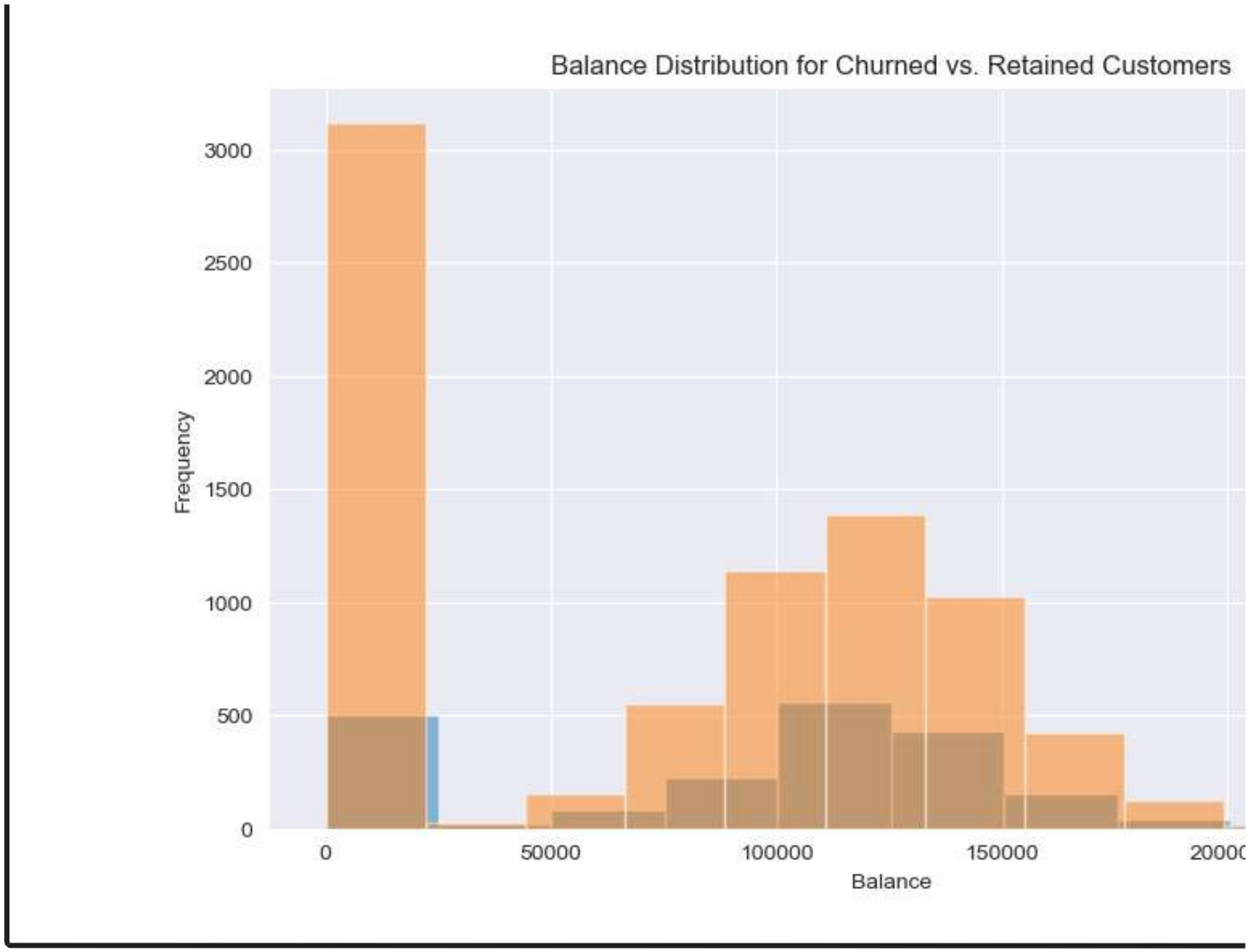
```
# Plotting
plt.figure(figsize=(10, 6))

# Plot age distribution for churned customers
plt.hist(df_churned['Balance'], bins=10, alpha=0.5, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['Balance'], bins=10, alpha=0.5, label='Retained')

plt.title('Balance Distribution for Churned vs. Retained Customers')
plt.xlabel('Balance')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```


Balance Distribution for Churned vs. Retained Customers
Frequency
3000
2500
2000
1500
1000
500
0
0
50000
100000
150000
Balance

In [24]:

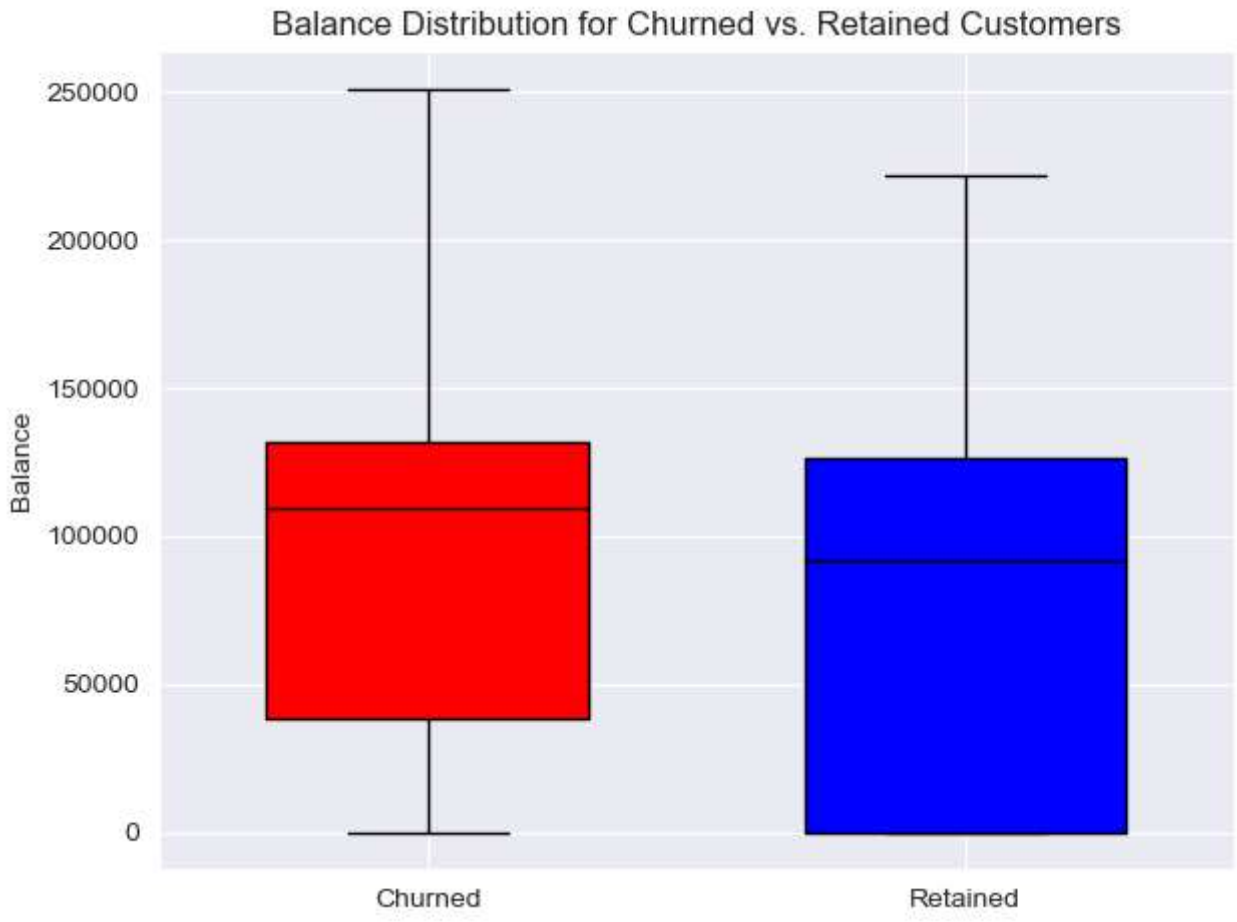
```
# Create boxplot for churned customers' credit scores
plt.boxplot(df_churned['Balance'], positions=[1], widths=0.6, patch_artist=True,

# Create boxplot for retained customers' credit scores
plt.boxplot(df_retained['Balance'], positions=[2], widths=0.6, patch_artist=True

# Adjust x-axis labels
plt.xticks([1, 2], ['Churned', 'Retained'])

plt.title('Balance Distribution for Churned vs. Retained Customers')
plt.ylabel('Balance')
plt.grid(True)

# Show the plot
plt.tight_layout()
plt.show()
```



In [25]:

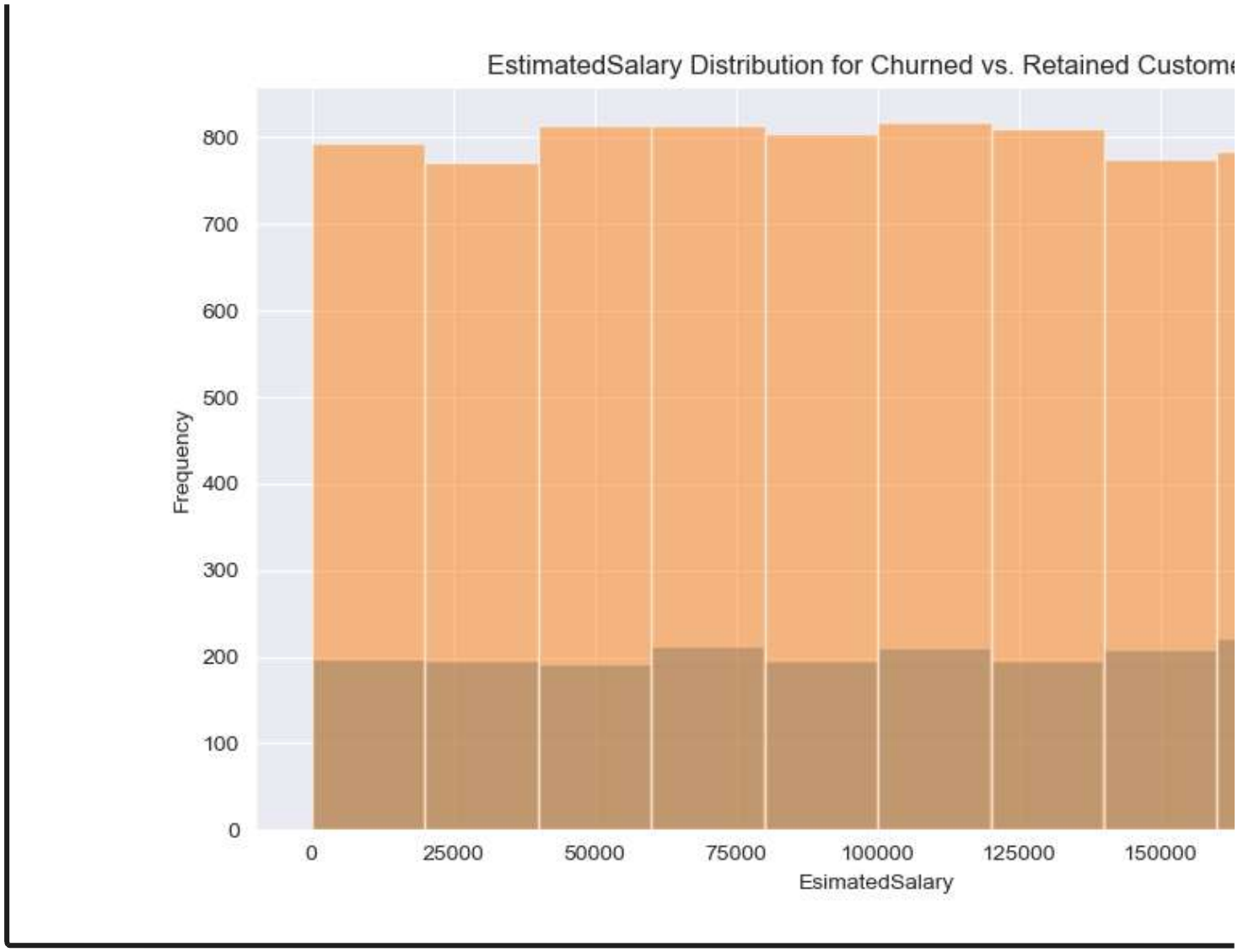
```
# Plotting
plt.figure(figsize=(10, 6))

# Plot age distribution for churned customers
plt.hist(df_churned['EstimatedSalary'], bins=10, alpha=0.5, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['EstimatedSalary'], bins=10, alpha=0.5, label='Retained')

plt.title('EstimatedSalary Distribution for Churned vs. Retained Customers')
plt.xlabel('EsimatedSalary')
plt.ylabel('Frequency')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```


EstimatedSalary Distribution for Churned vs. Retained Custome
Frequency
0
100
200
300
400
500
600
700
800
0
25000
50000
75000
100000
125000
150000
EsimatedSalary

In [26]:

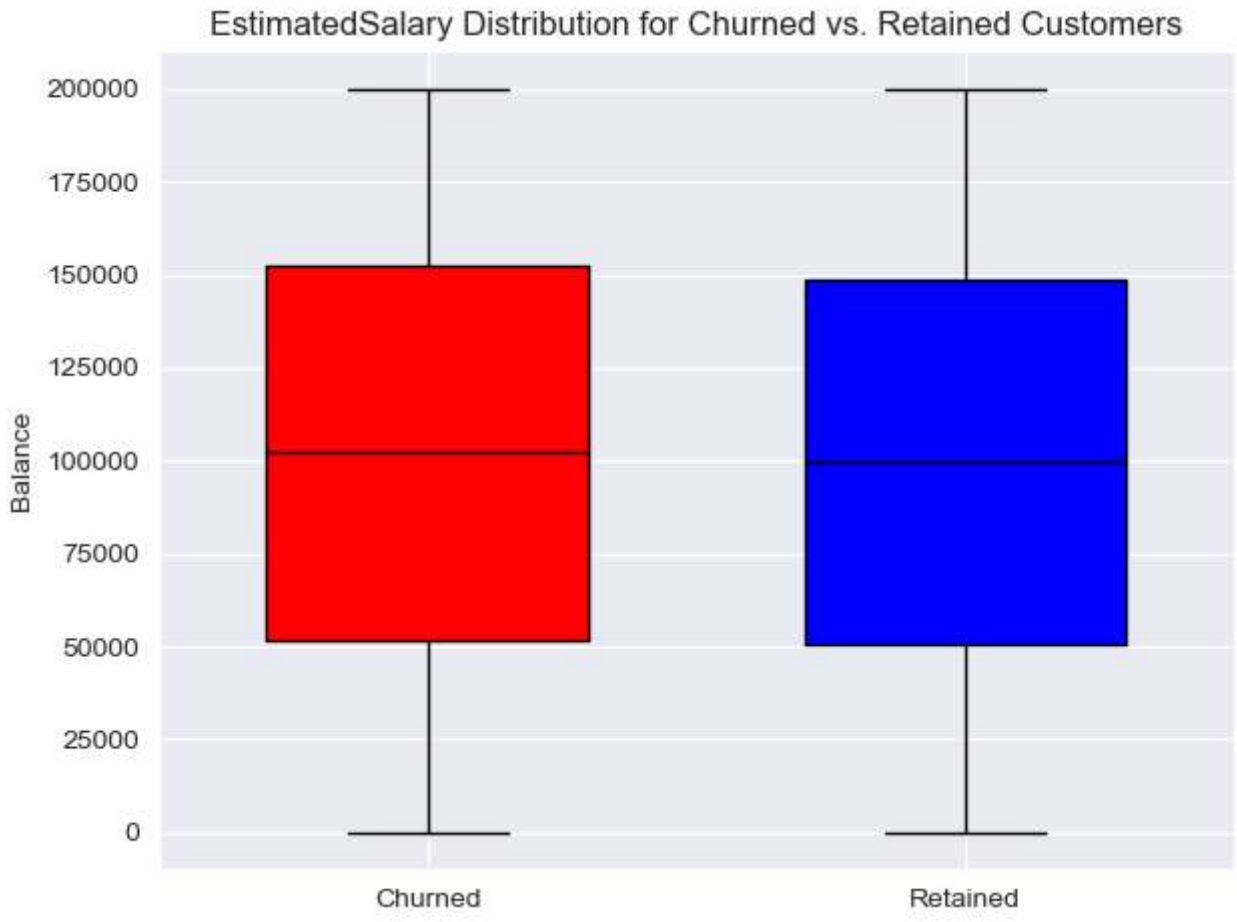
```
# Create boxplot for churned customers' credit scores
plt.boxplot(df_churned['EstimatedSalary'], positions=[1], widths=0.6, patch_arti

# Create boxplot for retained customers' credit scores
plt.boxplot(df_retained['EstimatedSalary'], positions=[2], widths=0.6, patch_art

# Adjust x-axis labels
plt.xticks([1, 2], ['Churned', 'Retained'])

plt.title('EstimatedSalary Distribution for Churned vs. Retained Customers')
plt.ylabel('Balance')
plt.grid(True)

# Show the plot
plt.tight_layout()
plt.show()
```



## Categorical Variables

In [27]:

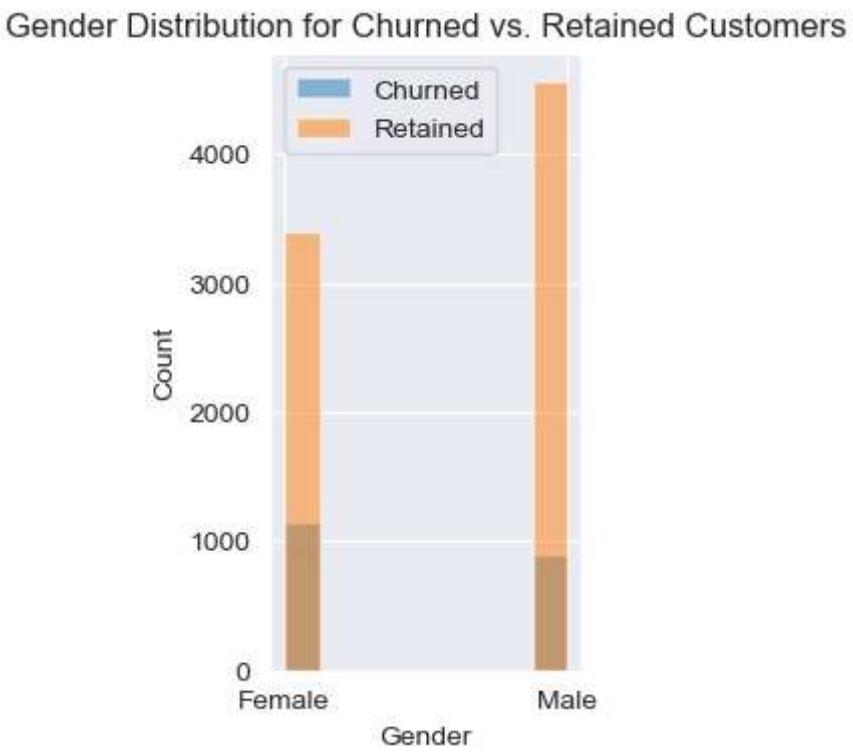
```
# Plotting
plt.figure(figsize=(2, 4))

# Plot age distribution for churned customers
plt.hist(df_churned['Gender'], bins=8, alpha=0.5, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['Gender'], bins=8, alpha=0.5, label='Retained')

plt.title('Gender Distribution for Churned vs. Retained Customers')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



In [28]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

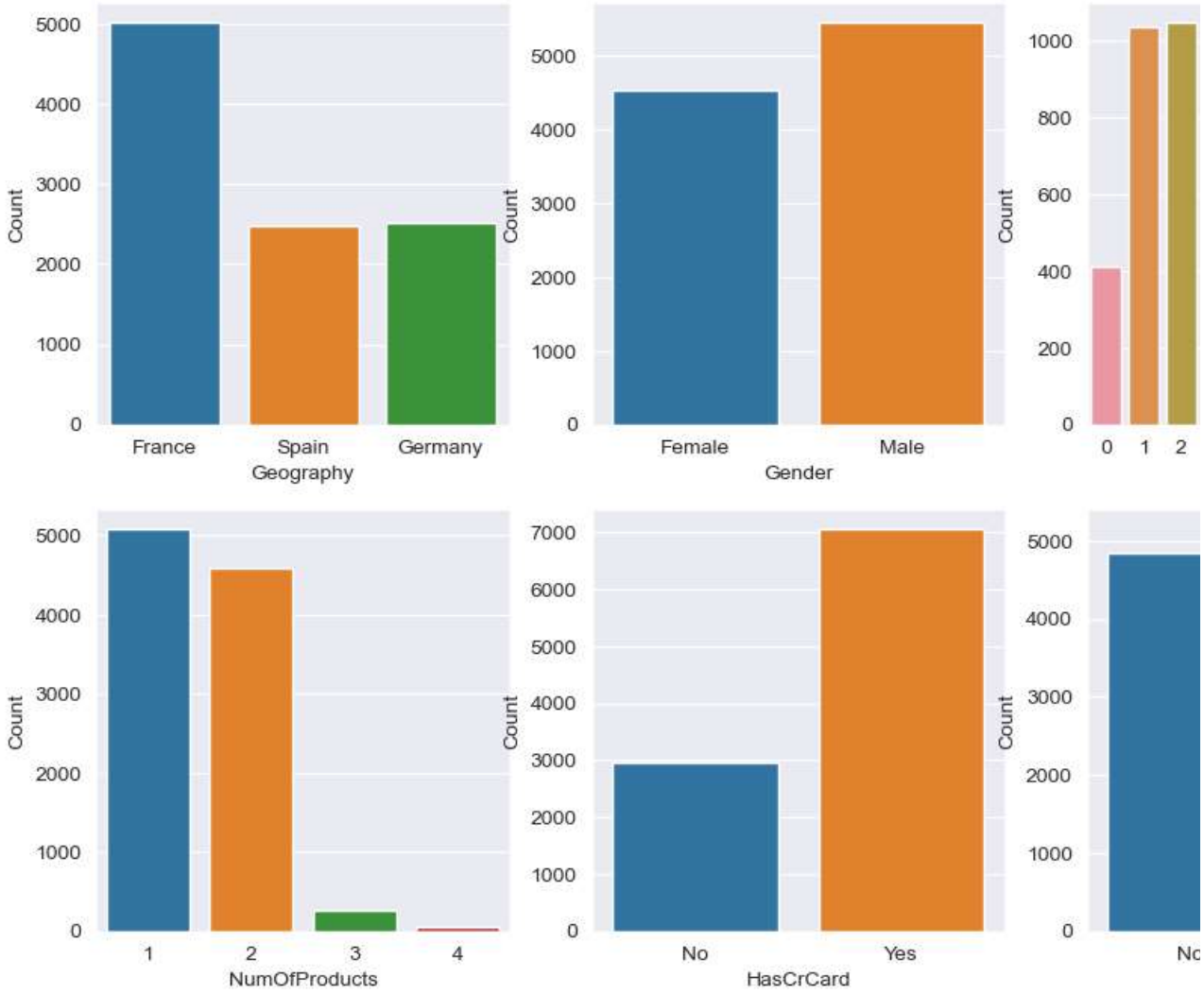
In [29]:
```
df_cat = df[categorical]

fig, ax = plt.subplots(2, 3, figsize=(12, 8))

for index, column in enumerate(df_cat.columns):

    plt.subplot(2, 3, index + 1)
    sns.countplot(x=column, data=df)

    plt.ylabel('Count')
    if (column == 'HasCrCard' or column == 'IsActiveMember'):
        plt.xticks([0, 1], ['No', 'Yes'])
```



In [30]:

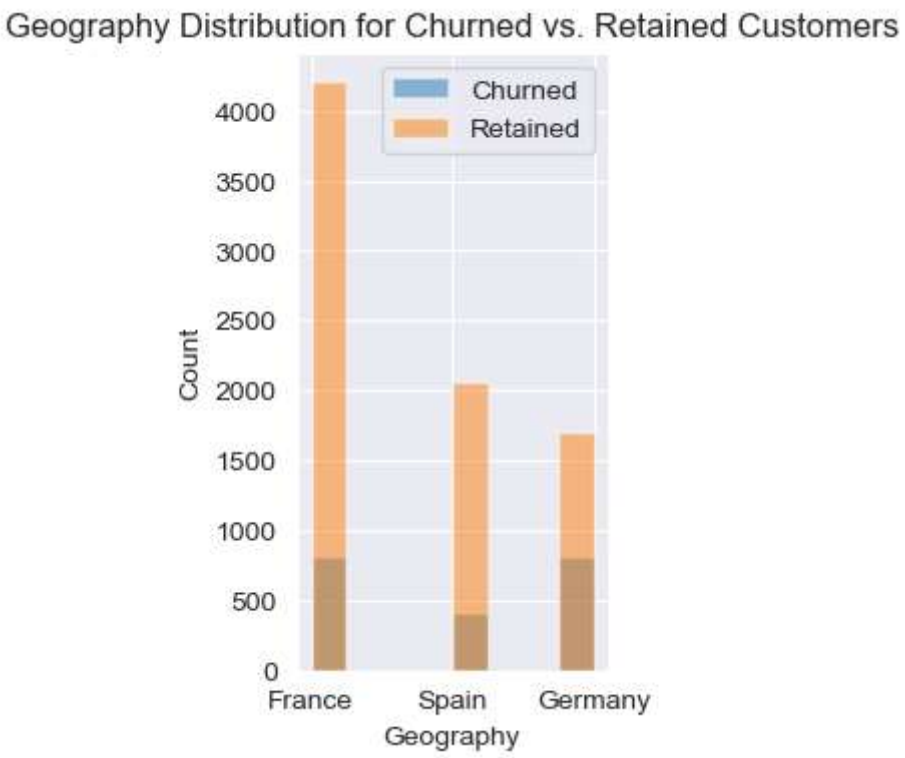
```
# Plotting
plt.figure(figsize=(2, 4))

# Plot age distribution for churned customers
plt.hist(df_churned['Geography'], bins=8, alpha=0.5, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['Geography'], bins=8, alpha=0.5, label='Retained')

plt.title('Geography Distribution for Churned vs. Retained Customers')
plt.xlabel('Geography')
plt.ylabel('Count')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



In [31]:

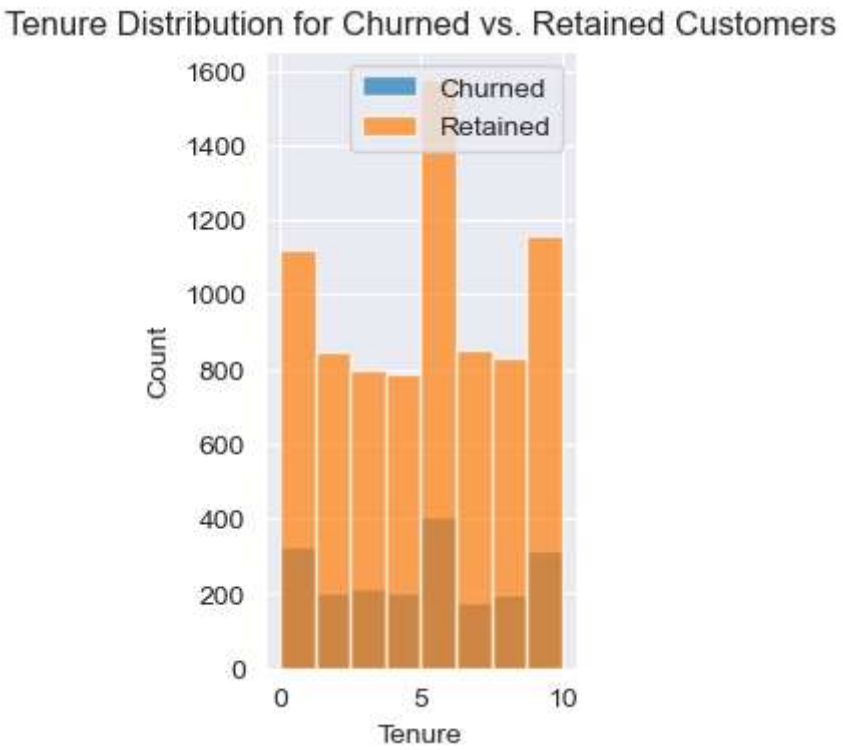
```
# Plotting
plt.figure(figsize=(2, 4))

# Plot age distribution for churned customers
plt.hist(df_churned['Tenure'], bins=8, alpha=0.7, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['Tenure'], bins=8, alpha=0.7, label='Retained')

plt.title('Tenure Distribution for Churned vs. Retained Customers')
plt.xlabel('Tenure')
plt.ylabel('Count')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



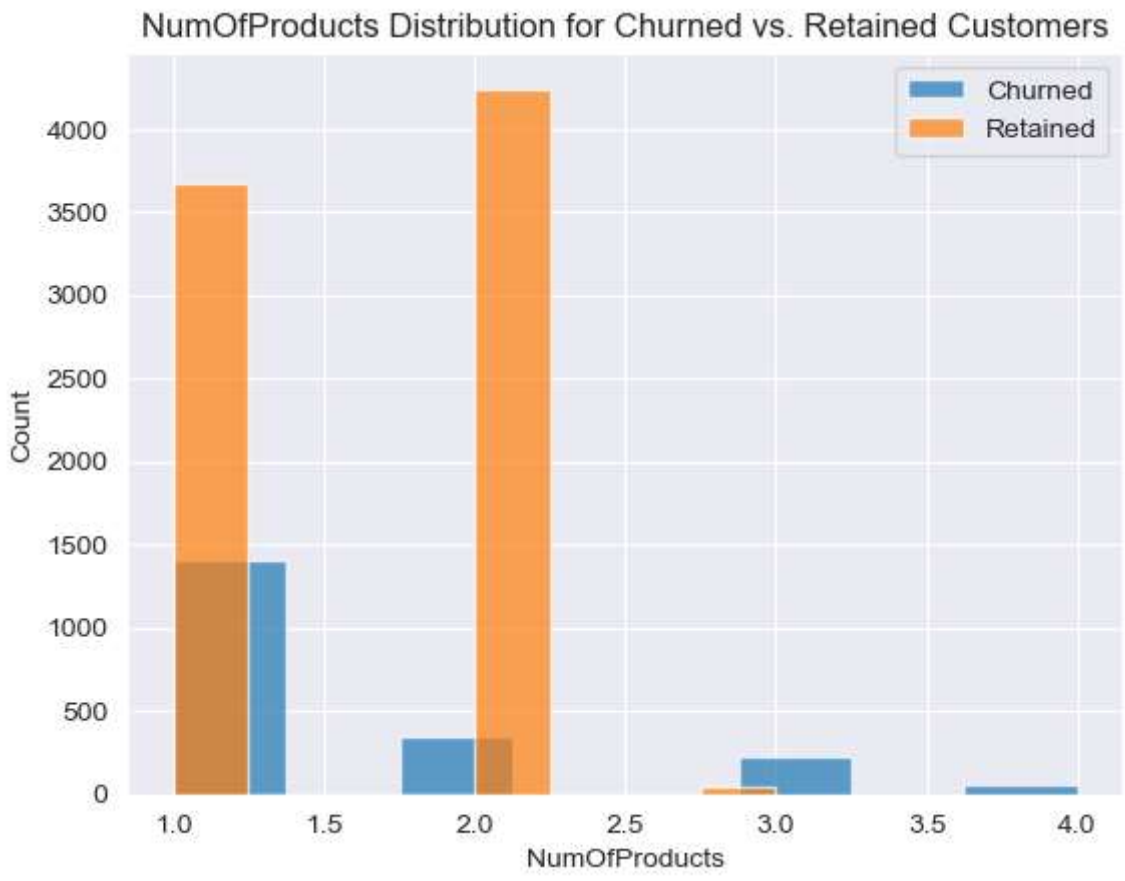
In [32]:
```
# Plotting
plt.figure()

# Plot age distribution for churned customers
plt.hist(df_churned['NumOfProducts'], bins=8, alpha=0.7, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['NumOfProducts'], bins=8, alpha=0.7, label='Retained')

plt.title('NumOfProducts Distribution for Churned vs. Retained Customers')
plt.xlabel('NumOfProducts')
plt.ylabel('Count')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



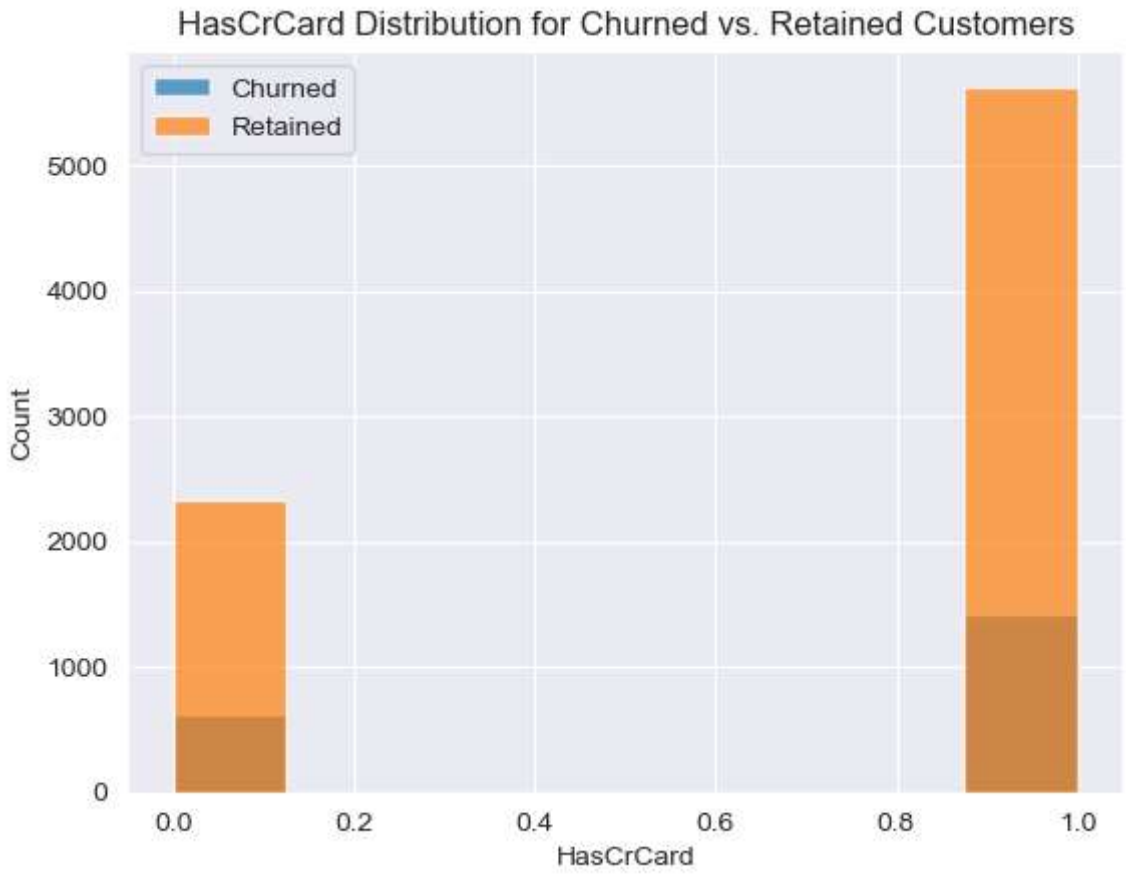
In [33]:

```
# Plotting
plt.figure()

# Plot age distribution for churned customers
plt.hist(df_churned['HasCrCard'], bins=8, alpha=0.7, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['HasCrCard'], bins=8, alpha=0.7, label='Retained')

plt.title('HasCrCard Distribution for Churned vs. Retained Customers')
plt.xlabel('HasCrCard')
plt.ylabel('Count')
plt.legend()
plt.grid(True)
```



In [34]:

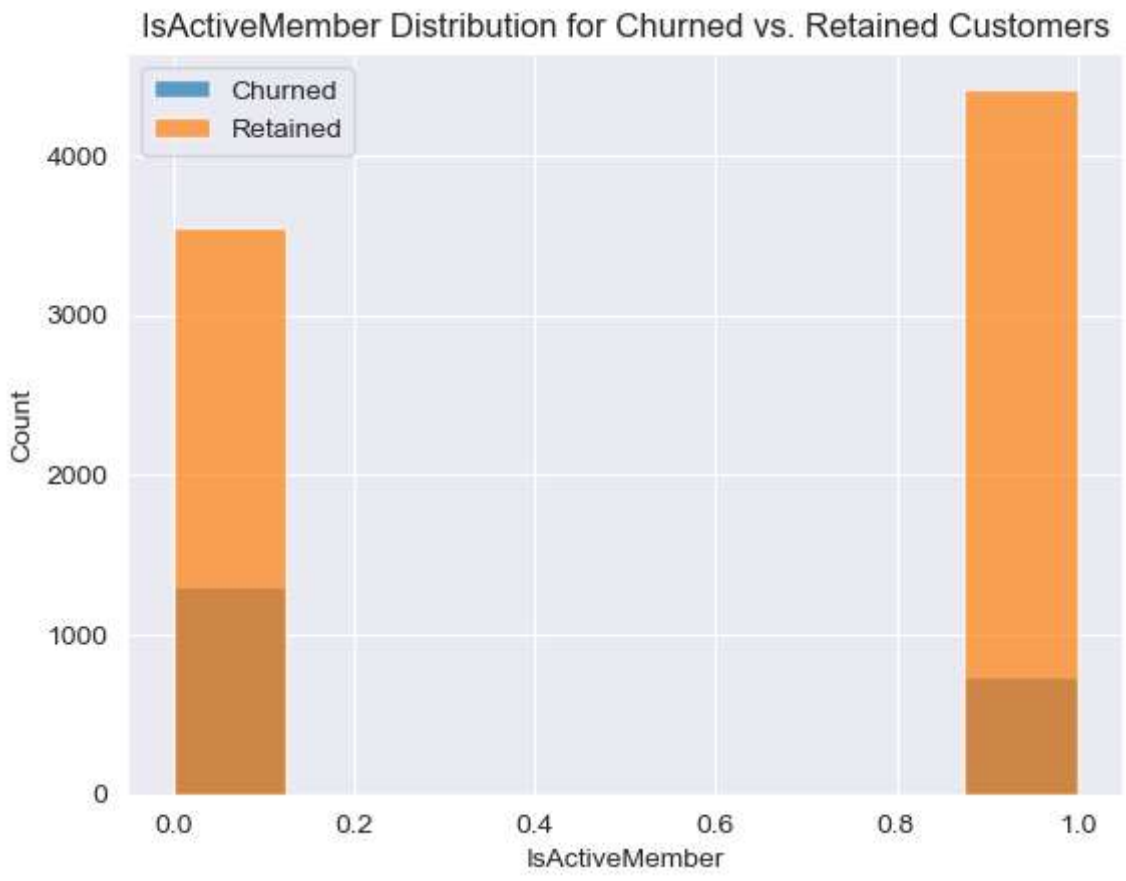
```
# Plotting
plt.figure()

# Plot age distribution for churned customers
plt.hist(df_churned['IsActiveMember'], bins=8, alpha=0.7, label='Churned')

# Plot age distribution for retained customers
plt.hist(df_retained['IsActiveMember'], bins=8, alpha=0.7, label='Retained')

plt.title('IsActiveMember Distribution for Churned vs. Retained Customers')
plt.xlabel('IsActiveMember')
plt.ylabel('Count')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



In [35]:
```
df.drop(['Tenure', 'HasCrCard', 'EstimatedSalary'], axis=1, inplace=True)
df.columns
```

```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Balance', 'NumOfProducts',
       'IsActiveMember', 'Exited'],
      dtype='object')
```

In [36]:
```
df.shape
```

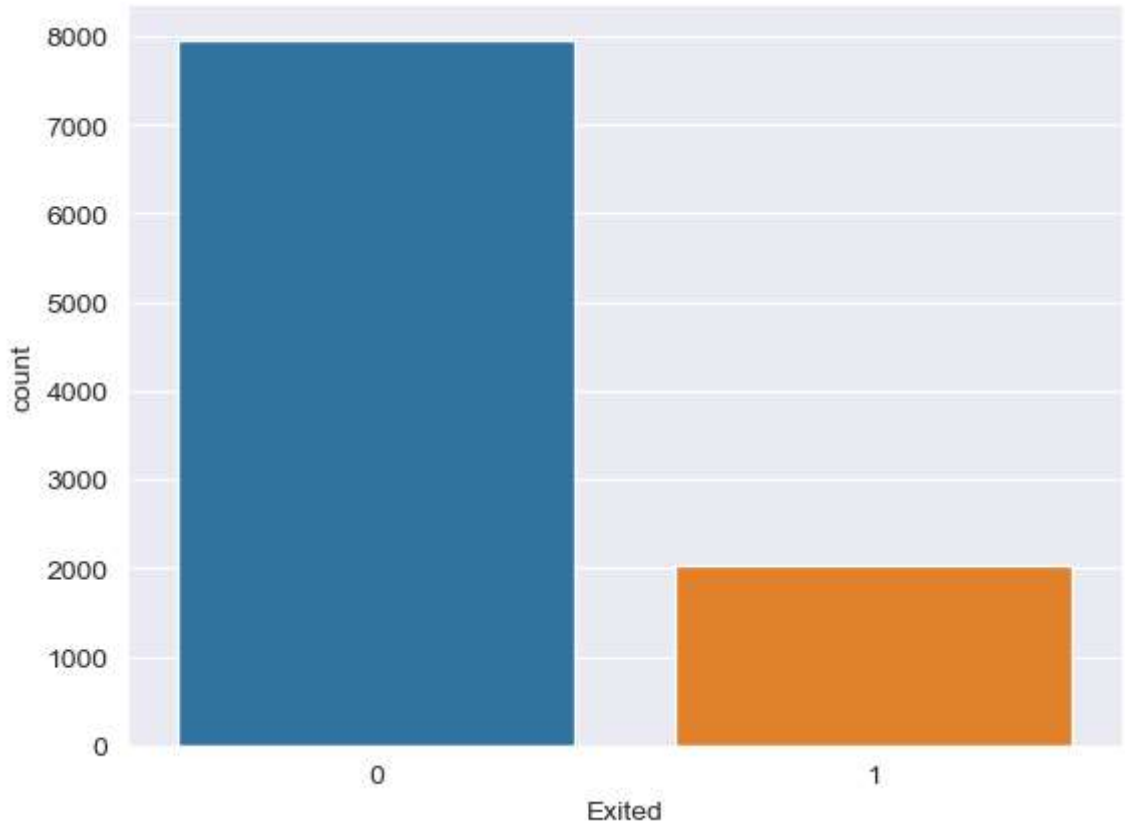
```
(10000, 8)
```

In [37]:
```
df['Exited'].value_counts()
```

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

In [38]:
```
# Assuming df is your DataFrame
sns.countplot(x='Exited', data=df)

# Optionally, show the plot
plt.show()
```




split the featurevariable and tergetvarible

In [39]:
```
X = df.iloc[:, 0:8].values
y = df.iloc[:, -1].values
```

In [40]:
```
X
```

```
array([[619, 'France', 'Female', ..., 1, 1, 1],
       [608, 'Spain', 'Female', ..., 1, 1, 0],
       [502, 'France', 'Female', ..., 3, 0, 1],
       ...,
       [709, 'France', 'Female', ..., 1, 1, 1],
       [772, 'Germany', 'Male', ..., 2, 0, 1],
       [792, 'France', 'Female', ..., 1, 0, 0]], dtype=object)
```

In [41]:
```
X.shape
```

```
(10000, 8)
```

In [42]:
```
y
```

```
array([1, 0, 1, ..., 1, 1, 0], dtype=int64)
```

In [43]:
```
y.shape
```

```
(10000,)
```

## Encoding categorical data

In [44]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

In [45]:
```
X
```

```
array([[619, 'France', 0, ..., 1, 1, 1],
       [608, 'Spain', 0, ..., 1, 1, 0],
       [502, 'France', 0, ..., 3, 0, 1],
       ...,
       [709, 'France', 0, ..., 1, 1, 1],
       [772, 'Germany', 1, ..., 2, 0, 1],
       [792, 'France', 0, ..., 1, 0, 0]], dtype=object)
```

In [46]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 1] = le.fit_transform(X[:, 1])
```

In [47]:
```
X
```

```
array([[619, 0, 0, ..., 1, 1, 1],
       [608, 2, 0, ..., 1, 1, 0],
       [502, 0, 0, ..., 3, 0, 1],
       ...,
       [709, 0, 0, ..., 1, 1, 1],
       [772, 1, 1, ..., 2, 0, 1],
       [792, 0, 0, ..., 1, 0, 0]], dtype=object)
```

In [48]:
```
y
```

```
array([1, 0, 1, ..., 1, 1, 0], dtype=int64)
```

In [49]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
#for this observation let me selcted as 100 observaion for test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,randc
```

In [50]:
```
X_train.shape
```

```
(8000, 8)
```

In [51]:
```
X_test.shape
```

```
(2000, 8)
```

In [52]:
```
y_train.shape
```

```
(8000,)
```

In [53]:
```
y_test.shape
```

```
(2000,)
```

In [54]:
```
X_train
```

```
array([[667, 2, 0, ..., 2, 0, 0],
       [427, 1, 1, ..., 1, 1, 0],
       [535, 0, 0, ..., 1, 0, 0],
       ...,
       [738, 0, 1, ..., 2, 0, 0],
       [590, 2, 0, ..., 2, 1, 0],
       [623, 1, 0, ..., 1, 0, 1]], dtype=object)
```

In [55]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
```

```
array([[ 0.16958176,  1.51919821, -1.09168714, ...,  0.8095029 ,
        -1.03227043, -0.50624244],
       [-2.30455945,  0.3131264 ,  0.91601335, ..., -0.92159124,
         0.9687384 , -0.50624244],
       [-1.19119591, -0.89294542, -1.09168714, ..., -0.92159124,
        -1.03227043, -0.50624244],
       ...,
       [ 0.9015152 , -0.89294542,  0.91601335, ...,  0.8095029 ,
        -1.03227043, -0.50624244],
       [-0.62420521,  1.51919821, -1.09168714, ...,  0.8095029 ,
         0.9687384 , -0.50624244],
       [-0.28401079,  0.3131264 , -1.09168714, ..., -0.92159124,
        -1.03227043,  1.97533814]])
```

In [56]:
```
X_test
```

```
array([[-0.55204276,  0.3131264 , -1.09168714, ..., -0.92159124,
         0.9687384 , -0.50624244],
       [-1.31490297, -0.89294542, -1.09168714, ..., -0.92159124,
        -1.03227043,  1.97533814],
       [ 0.57162971,  1.51919821, -1.09168714, ..., -0.92159124,
         0.9687384 , -0.50624244],
       ...,
       [-0.74791227,  1.51919821,  0.91601335, ...,  0.8095029 ,
        -1.03227043, -0.50624244],
       [-0.00566991,  0.3131264 ,  0.91601335, ..., -0.92159124,
         0.9687384 , -0.50624244],
       [-0.79945688,  0.3131264 ,  0.91601335, ..., -0.92159124,
        -1.03227043, -0.50624244]])
```

## Training XGBoost on the Training set

In [57]:
```
from xgboost import XGBClassifier
classifier = XGBClassifier()
classifier.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

## Predicting the Test set results

In [58]:
```
y_pred = classifier.predict(X_test)
y_pred
```

```
array([0, 1, 0, ..., 0, 0, 0])
```

In [59]:
```
## Making the Confusion Matrix
```

In [60]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1595    0]
 [   0  405]]
```

In [61]:
```
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
print(ac)
```

```
1.0
```

In [62]:
```
bias = classifier.score(X_train,y_train)
bias
```

```
1.0
```

In [63]:
```
variance = classifier.score(X_test,y_test)
variance
```

```
1.0
```

## Applying k-Fold Cross Validation

In [64]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, c
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 100.00 %
Standard Deviation: 0.00 %
```

In [65]:
```
# Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier()
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.
```

In [66]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred
```

```
array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
```

In [67]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1595    0]
 [   0  405]]
```

In [68]:
```
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
print(ac)
```

```
1.0
```

In [69]:
```
bias = classifier.score(X_train,y_train)
bias
```

```
1.0
```

In [70]:
```
variance = classifier.score(X_test,y_test)
variance
```

```
1.0
```

In [71]:
```
# train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB


# instantiate the model
gnb = GaussianNB()


# fit the model
gnb.fit(X_train, y_train)
```

```
GaussianNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [72]:
```
y_pred = gnb.predict(X_test)

y_pred
```

```
array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
```

In [73]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 1.0000
```

In [74]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 1.0000
```

In [75]:
```
bias = classifier.score(X_train,y_train)
bias
```

```
1.0
```

In [76]:
```
variance = classifier.score(X_test,y_test)
variance
```

```
1.0
```

In [77]:
```
# Training the SVM model on the Training set
from sklearn.svm import SVC
classifier = SVC()
classifier.fit(X_train, y_train)
```

```
SVC()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [78]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred
```

```
array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
```

In [79]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1595    0]
 [   0  405]]
```

In [80]:
```
# This is to get the Models Accuracy
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
print(ac)
```

```
1.0
```

In [81]:
```
bias = classifier.score(X_train,y_train)
bias
```

```
1.0
```

In [82]:
```
variance = classifier.score(X_test,y_test)
variance
```

```
1.0
```

In [83]:
```
# This is to get the Classification Report
from sklearn.metrics import classification_report
cr = classification_report(y_test, y_pred)
cr
```

```
'              precision    recall  f1-score   support\n\n           0       1.00      1.00      1.00
1.00      1.00       405\n\n    accuracy                           1.00      2000\n   macro avg
weighted avg       1.00      1.00      1.00      2000\n'
```

In [84]:
```
# Training the Logistic Regression model on the Training set

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
```

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [85]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred
```

```
array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
```

In [86]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1595    0]
 [   0  405]]
```

In [87]:
```
# This is to get the Models Accuracy
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
print(ac)
```

```
1.0
```

In [88]:
```
# This is to get the Classification Report
from sklearn.metrics import classification_report
cr = classification_report(y_test, y_pred)
cr
```

```
'              precision    recall  f1-score   support\n\n           0       1.00      1.00      1.00
1.00      1.00       405\n\n    accuracy                           1.00      2000\n   macro avg
weighted avg       1.00      1.00      1.00      2000\n'
```

In [89]:
```
bias = classifier.score(X_train, y_train)
bias
```

```
1.0
```

In [90]:
```
variance = classifier.score(X_test, y_test)
variance
```

```
1.0
```

In [91]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 1.0000
```

In [ ]:
```
```