Bank Customer Churn Prediction - EDA

Libraries dependence

In [1]:
```
import pandas as pd
import numpy as np

# Matplotlib for visualization
from matplotlib import pyplot as plt
# display plots in the notebook
%matplotlib inline

# Seaborn for easier visualization
import seaborn as sns
sns.set_style('darkgrid')

# store elements as dictionary keys and their counts as dictionary values
from collections import Counter

import warnings
```

Data Source

Kaggle - Churn Modelling Calssification Data Set

- This data set contains details of a bank's customers and the target variable is a binary variable reflecting the fact whether the customer left the bank (closed his account) or he continues to be a customer.
- It consists of 10,000 records with demographic and bank history information from customers from three countries, France, Germany and Spain.

In [2]:
```
#Exploratory Analysis

#Read the CSV and Perform Basic Data Cleaning
```

In [3]:
```
#Load the dataset
df = pd.read_csv("Bank Churn Modelling.csv")
print(f"Dataframe dimensions: {df.shape}")
df.head()
```

```
Dataframe dimensions: (10000, 13)
```

Out[3]:

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products
0	15634602	Hargrave	619	France	Female	42	2	0.00	1
1	15647311	Hill	608	Spain	Female	41	1	83807.86	1
2	15619304	Onio	502	France	Female	42	8	159660.80	3
3	15701354	Boni	699	France	Female	39	1	0.00	2
4	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1

In [4]:
```
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
```

In [5]:
```
# List number of unique customer IDs
df.CustomerId.nunique()
```

Out[5]:
```
10000
```

All Cusuomer IDs are unique --> that also means no duplicates

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
0
```

Unused Features

====>> To make dataframe easily readable we will drop features not needed for machine learning <<=====

- CustomerId

In [7]:
```
# Drop unused features
df.drop(['CustomerId', 'Surname'], axis=1, inplace=True)
print(f"Dataframe dimensions: {df.shape}")
df.head()
```

```
Dataframe dimensions: (10000, 11)
```

Out[7]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estir s
0	619	France	Female	42	2	0.00	1	1	1	1013
1	608	Spain	Female	41	1	83807.86	1	0	1	1125
2	502	France	Female	42	8	159660.80	3	1	0	1139
3	699	France	Female	39	1	0.00	2	0	0	938
4	850	Spain	Female	43	2	125510.82	1	1	1	790

In [8]:
```
df.isnull().sum()
```

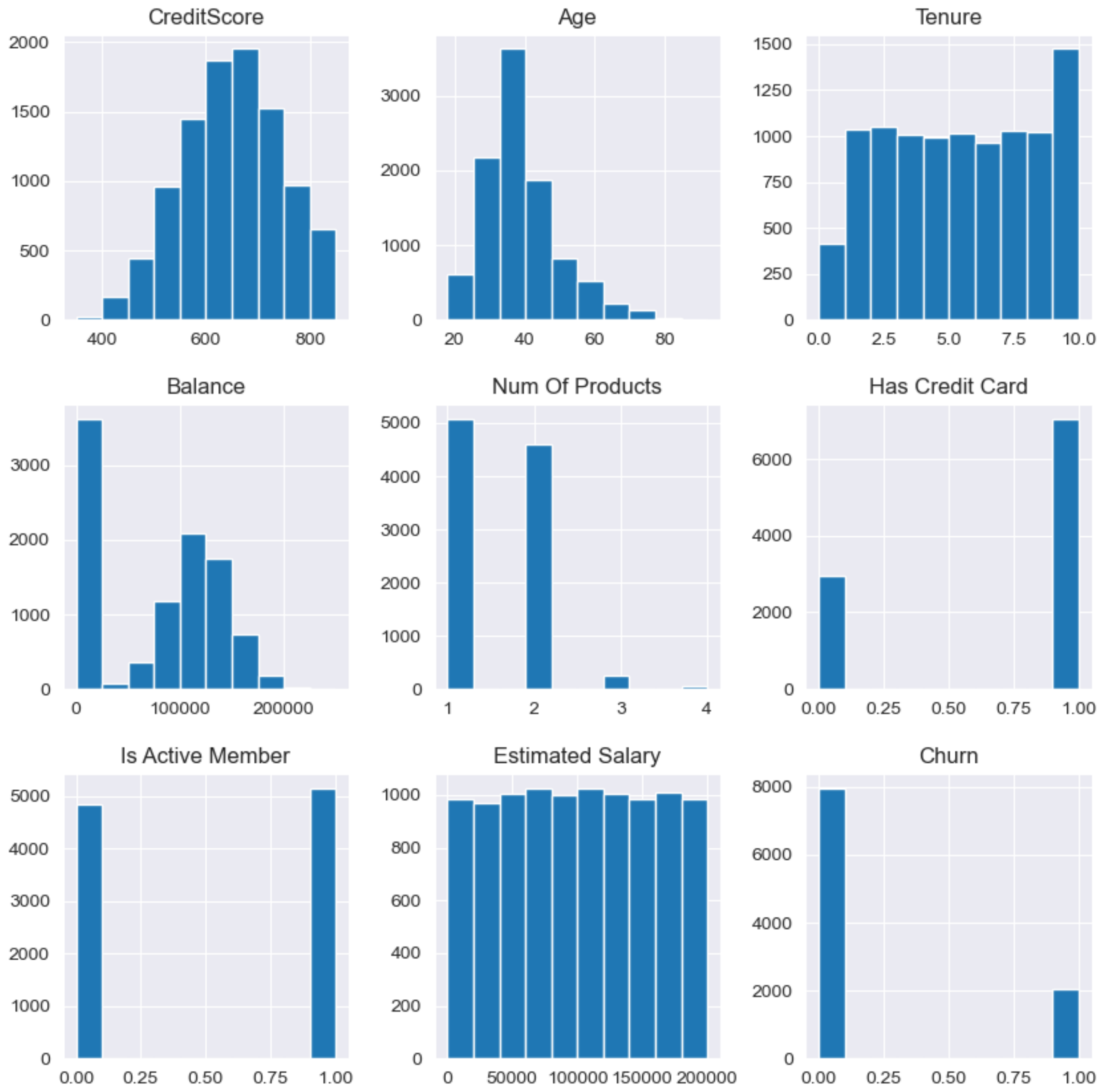
Out[8]:
```
CreditScore         0
Geography           0
Gender              0
Age                 0
Tenure              0
Balance             0
Num Of Products     0
Has Credit Card     0
Is Active Member    0
Estimated Salary    0
Churn               0
dtype: int64
```

Distributions of Numeric Features

Plotting Histogram grid

In [9]:

```
# Plot histogram grid
df.hist(figsize=(10,10))
plt.show()
```



Summary statistics for the numeric features

In [10]:
```
# Summarize numerical features
df.describe()
```

Out[10]:

	CreditScore	Age	Tenure	Balance	Num Of Products	Has Credit Card
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000

From the summary statistics and the histograms we can conclude that all features look OK. We do not see any extreme values for any feature.

Distributions of Categorical Features

In [11]:
```
# Summarize categorical features
df.describe(include=["object"])
```

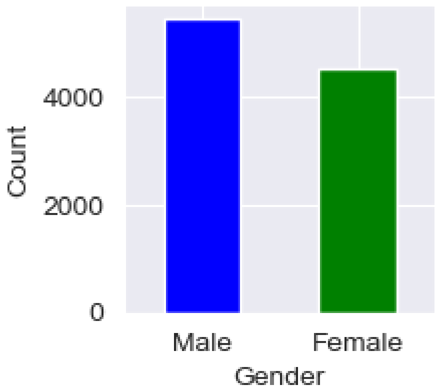
Out[11]:

	Geography	Gender
count	10000	10000
unique	3	2
top	France	Male
freq	5014	5457

This shows us the number of unique classes for each feature. For example, there are more males (5457) than females. And France is most common of 3 geographies in our dataframe. There are no sparse classes.

Let's visualize this information

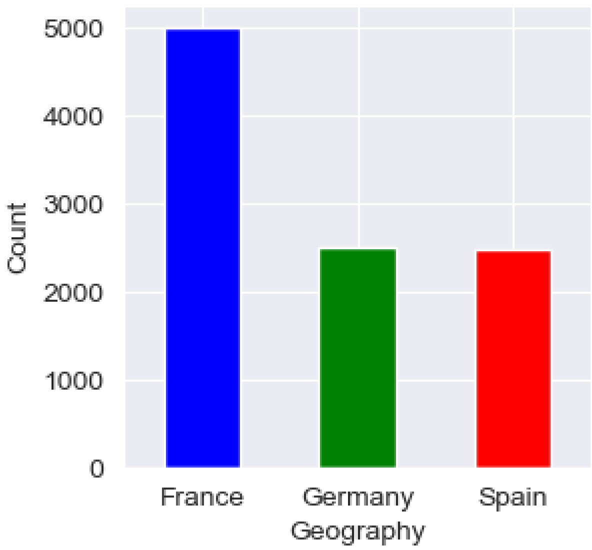
In [12]:
```
# Bar plot for "Gender"
plt.figure(figsize=(2,2))
df['Gender'].value_counts().plot.bar(color=['b','g'])
plt.ylabel('Count')
plt.xlabel('Gender')
plt.xticks(rotation=0)
plt.show()

# Display count of each class
Counter(df.Gender)
```



Out[12]: `Counter({'Female': 4543, 'Male': 5457})`

In our data sample there are more males than females.

In [13]:
```
# Bar plot for "Geography"
plt.figure(figsize=(3,3))
df['Geography'].value_counts().plot.bar(color=['b','g','r'])
plt.ylabel('Count')
plt.xlabel('Geography')
plt.xticks(rotation=0)
plt.show()

# Display count of each class
Counter(df.Geography)
```



Out[13]: `Counter({'France': 5014, 'Spain': 2477, 'Germany': 2509})`

Majority of customers are from France, about 50%, and from Germany and Spain around 25% each

Churn Segmentation by Gender

In [14]:
```
# Segment "Churn" by gender and display the frequency and percentage within
grouped = df.groupby('Gender')['Churn'].agg(Count='value_counts')
grouped
```

Out[14]:

Gender	Churn	Count
Female	0	3404
	1	1139
Male	0	4559
	1	898

In [15]:
```
# Reorganize dataframe for plotting count
dfgc = grouped
dfgc = dfgc.pivot_table(values = 'Count', index = 'Gender', columns = ['Chur
dfgc
```

Out[15]:

Churn / Gender	0	1
Female	3404	1139
Male	4559	898

In [16]:

```python
# Calculate percentage within each class
dfgp = grouped.groupby(level=[0]).apply(lambda g: round(g * 100 / g.sum(), 2
dfgp.rename(columns={'Count' : 'Percentage'}, inplace = True)
dfgp
```

```
C:\Users\baps\AppData\Local\Temp\ipykernel_21152\4234812768.py:2: FutureWa
rning: Not prepending group keys to the result index of transform-like app
ly. In the future, the group keys will be included in the index, regardles
s of whether the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  dfgp = grouped.groupby(level=[0]).apply(lambda g: round(g * 100 / g.sum
(), 2))
```

Out[16]:

		Percentage
Gender	**Churn**	
Female	**0**	74.93
	1	25.07
Male	**0**	83.54
	1	16.46

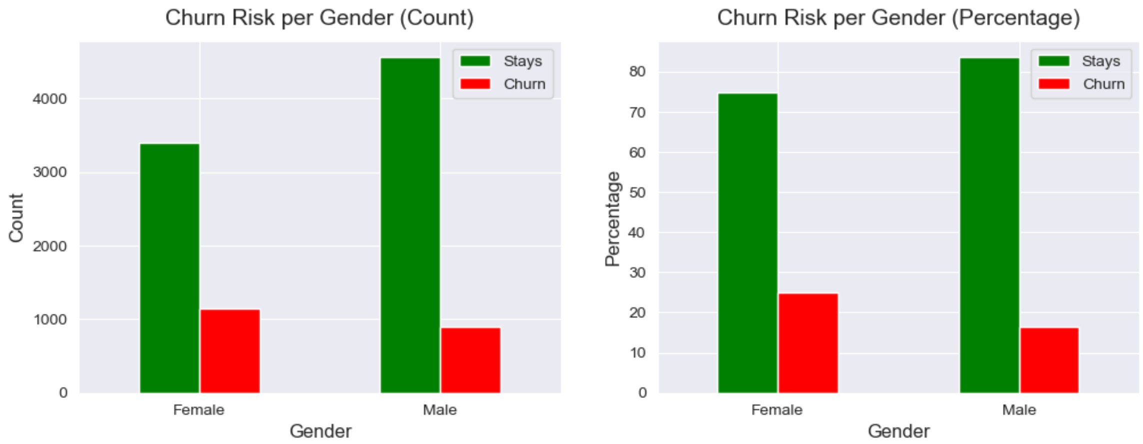
In [17]:

```python
# Reorganize dataframe for plotting percentage
dfgp = dfgp.pivot_table(values='Percentage', index='Gender', columns=['Churn
dfgp
```

Out[17]:

Churn	**0**	**1**
Gender		
Female	74.93	25.07
Male	83.54	16.46

In [18]:
```
# Churn distribution by gender, count + percentage

labels= ['Stays', 'Churn']

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

dfgc.plot(kind='bar',
          color=['g', 'r'],
          rot=0,
          ax=ax1)
ax1.legend(labels)
ax1.set_title('Churn Risk per Gender (Count)', fontsize=14, pad=10)
ax1.set_ylabel('Count',size=12)
ax1.set_xlabel('Gender', size=12)


dfgp.plot(kind='bar',
          color=['g', 'r'],
          rot=0,
          ax=ax2)
ax2.legend(labels)
ax2.set_title('Churn Risk per Gender (Percentage)', fontsize=14, pad=10)
ax2.set_ylabel('Percentage',size=12)
ax2.set_xlabel('Gender', size=12)

plt.show()
```



In percentage females are more likely to leave the bank; 25% comparing to males, 16%.

Churn Segmentation by Geography

In [19]:
```
# Segment "Exited" by geography and display the frequency and percentage wit
grouped = df.groupby('Geography')['Churn'].agg(Count='value_counts')
grouped
```

Out[19]:

		Count
Geography	**Churn**	
France	**0**	4204
	1	810
Germany	**0**	1695
	1	814
Spain	**0**	2064
	1	413

In [20]:
```
# Reorganize dataframe for plotting count
dfgeoc = grouped
dfgeoc = dfgeoc.pivot_table(values='Count', index='Geography', columns=['Chu
dfgeoc
```

Out[20]:

Churn	0	1
Geography		
France	4204	810
Germany	1695	814
Spain	2064	413

In [21]:
```
# Calculate percentage within each class
dfgeop = grouped.groupby(level=[0]).apply(lambda g: round(g * 100 / g.sum(),
dfgeop.rename(columns={'Count': 'Percentage'}, inplace=True)
dfgeop
```

```
C:\Users\baps\AppData\Local\Temp\ipykernel_21152\11712713.py:2: FutureWarn
ing: Not prepending group keys to the result index of transform-like appl
y. In the future, the group keys will be included in the index, regardless
of whether the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  dfgeop = grouped.groupby(level=[0]).apply(lambda g: round(g * 100 / g.su
m(), 2))
```

Out[21]:

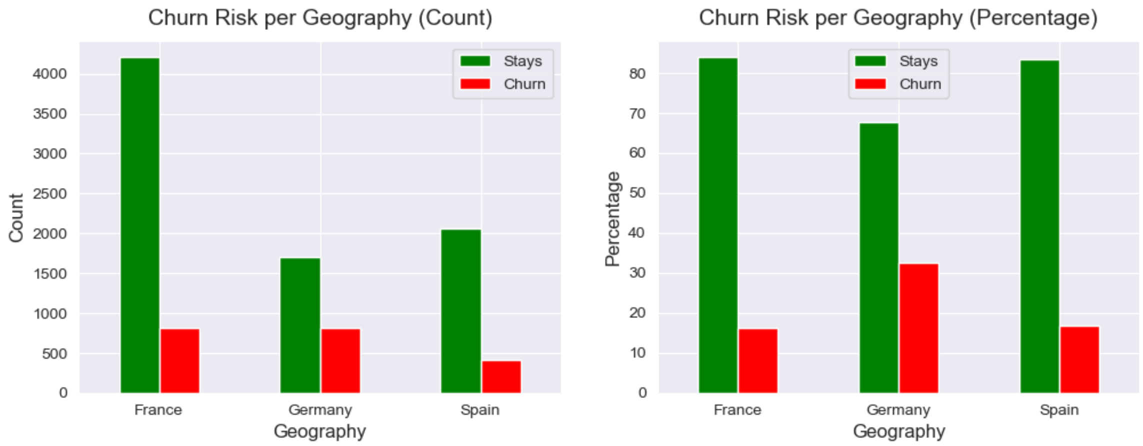
		Percentage
Geography	**Churn**	
France	**0**	83.85
	1	16.15
Germany	**0**	67.56
	1	32.44
Spain	**0**	83.33
	1	16.67

In [22]:
```
# Reorganize dataframe for plotting percentage
dfgeop = dfgeop.pivot_table(values='Percentage', index='Geography', columns=
dfgeop
```

Out[22]:

Churn	**0**	**1**
Geography		
France	83.85	16.15
Germany	67.56	32.44
Spain	83.33	16.67

In [23]:
```
# Churn distribution by geography, count + percentage

labels= ['Stays', 'Churn']

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

dfgeoc.plot(kind='bar',
            color=['g', 'r'],
            rot=0,
            ax=ax1)
ax1.legend(labels)
ax1.set_title('Churn Risk per Geography (Count)', fontsize=14, pad=10)
ax1.set_ylabel('Count',size=12)
ax1.set_xlabel('Geography', size=12)


dfgeop.plot(kind='bar',
            color=['g', 'r'],
            rot=0,
            ax=ax2)
ax2.legend(labels)
ax2.set_title('Churn Risk per Geography (Percentage)', fontsize=14, pad=10)
ax2.set_ylabel('Percentage',size=12)
ax2.set_xlabel('Geography', size=12)

plt.show()
```



The smallest number of customers are from Germany but it looks that they are most likely to leave the bank. Almost one third of German customers in our sample left the bank

Correlations

In [24]:
```
# Calculate correlations between numeric features
correlations = df.corr()

# sort features in order of their correlation with "Exited"
sort_corr_cols = correlations.Churn.sort_values(ascending=False).keys()
sort_corr = correlations.loc[sort_corr_cols,sort_corr_cols]
sort_corr
```

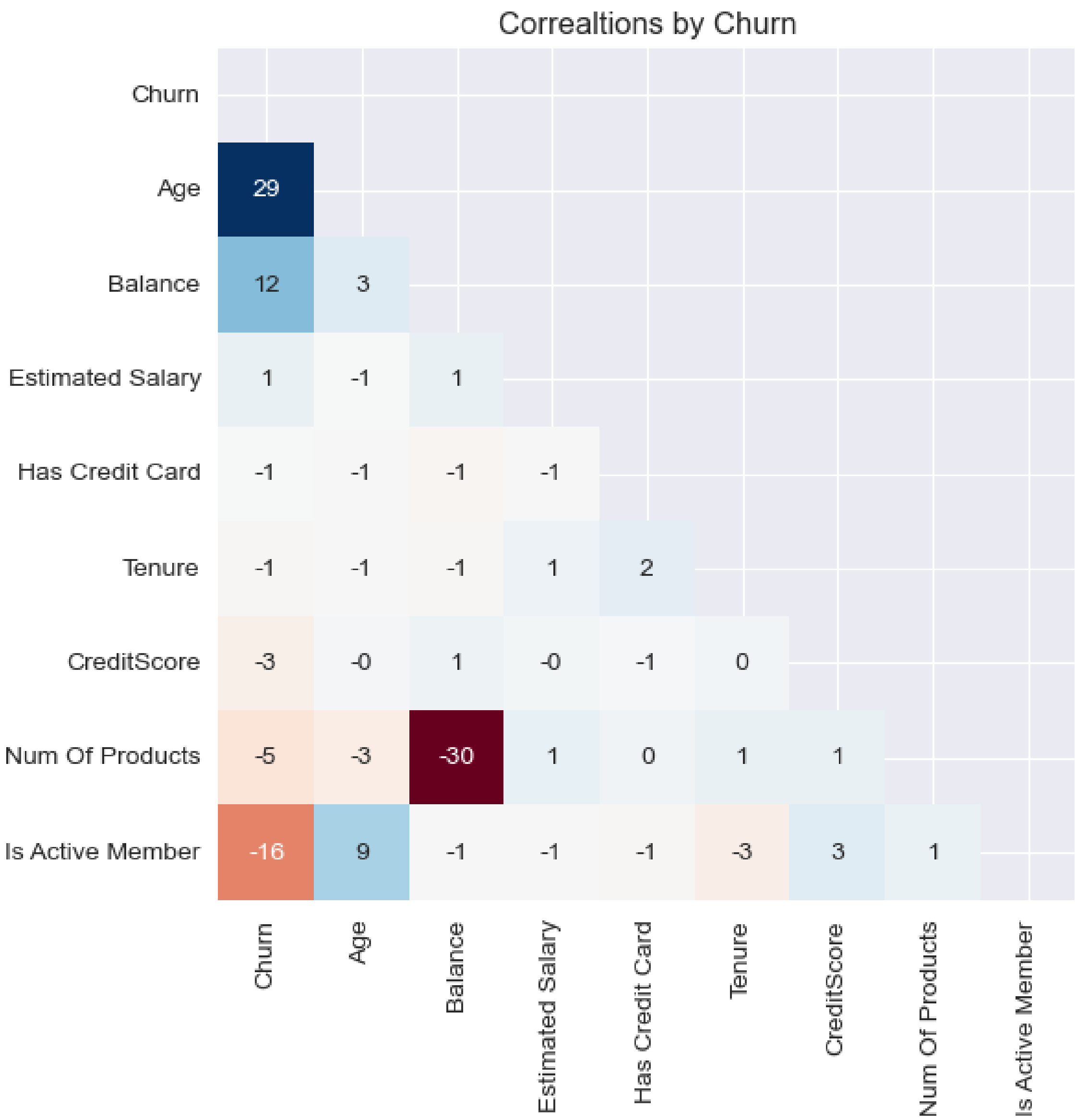
```
C:\Users\baps\AppData\Local\Temp\ipykernel_21152\749194107.py:2: FutureWar
ning: The default value of numeric_only in DataFrame.corr is deprecated. I
n a future version, it will default to False. Select only valid columns or
specify the value of numeric_only to silence this warning.
  correlations = df.corr()
```

Out[24]:

	Churn	Age	Balance	Estimated Salary	Has Credit Card	Tenure	CreditScore	N Pr
Churn	1.000000	0.285323	0.118533	0.012097	-0.007138	-0.014001	-0.027094	-0.(
Age	0.285323	1.000000	0.028308	-0.007201	-0.011721	-0.009997	-0.003965	-0.(
Balance	0.118533	0.028308	1.000000	0.012797	-0.014858	-0.012254	0.006268	-0.3
Estimated Salary	0.012097	-0.007201	0.012797	1.000000	-0.009933	0.007784	-0.001384	0.(
Has Credit Card	-0.007138	-0.011721	-0.014858	-0.009933	1.000000	0.022583	-0.005458	0.(
Tenure	-0.014001	-0.009997	-0.012254	0.007784	0.022583	1.000000	0.000842	0.(
CreditScore	-0.027094	-0.003965	0.006268	-0.001384	-0.005458	0.000842	1.000000	0.(
Num Of Products	-0.047820	-0.030680	-0.304180	0.014204	0.003183	0.013444	0.012238	1.(
Is Active Member	-0.156128	0.085472	-0.010084	-0.011421	-0.011866	-0.028362	0.025651	0.(

Let's use Seaborn's .heatmap() function to visualize the correlation grid.

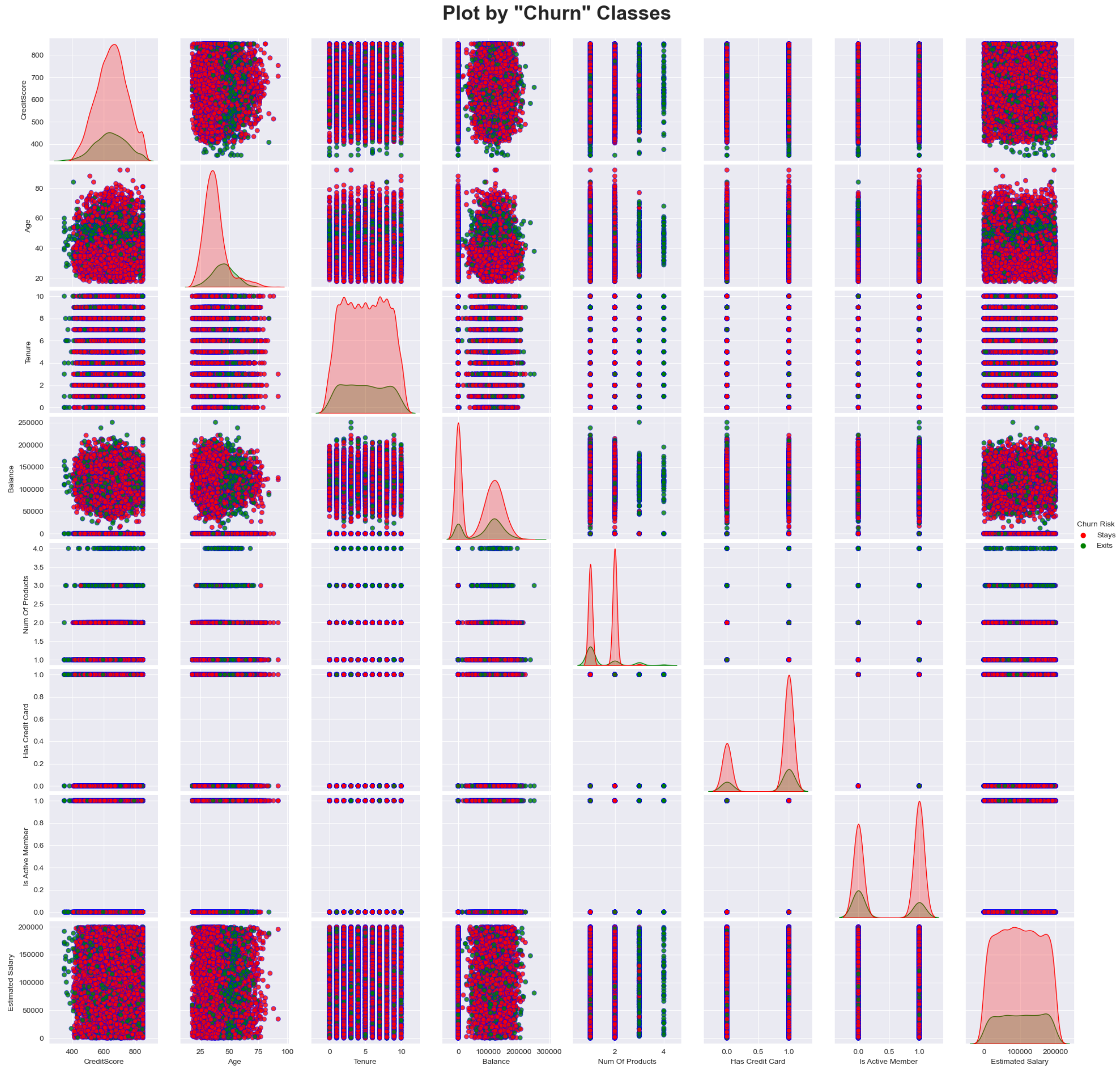
In [25]:
```python
# Generate a mask for the upper triangle
corr_mask = np.zeros_like(correlations)
corr_mask[np.triu_indices_from(corr_mask)] = 1

# Make the figsize 6X6
plt.figure(figsize = (6,6))

# Plot heatmap of annotated correlations; change background to white
##with sns.axes_style('white'):
sns.heatmap(sort_corr*100,
            cmap='RdBu',
            annot=True,
            fmt='.0f',
            mask=corr_mask,
            cbar=False)

plt.title('Correaltions by Churn', fontsize=12)
plt.yticks(rotation=0)
plt.show()
```



Very weak correlations in general. Only weak positive correlation with age, very weak positive correlation with balance, and very weak negative correlations with number of products and membership

Pairplot

In [26]:
```
# Plot Seaborn's pairplot
g = sns.pairplot(df, hue='Churn',
                 palette={1 : 'green',
                          0 : 'red'},
                 plot_kws={'alpha' : 0.8, 'edgecolor' : 'b', 'linewidth' : 0

fig = g.fig
fig.subplots_adjust(top=0.95, wspace=0.2)
fig.suptitle('Plot by "Churn" Classes',
             fontsize=26,
             fontweight='bold')


# Update the legend
new_title = 'Churn Risk'
g._legend.set_title(new_title)
# replace labels
new_labels = ['Stays', 'Exits']
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)

plt.show()
```

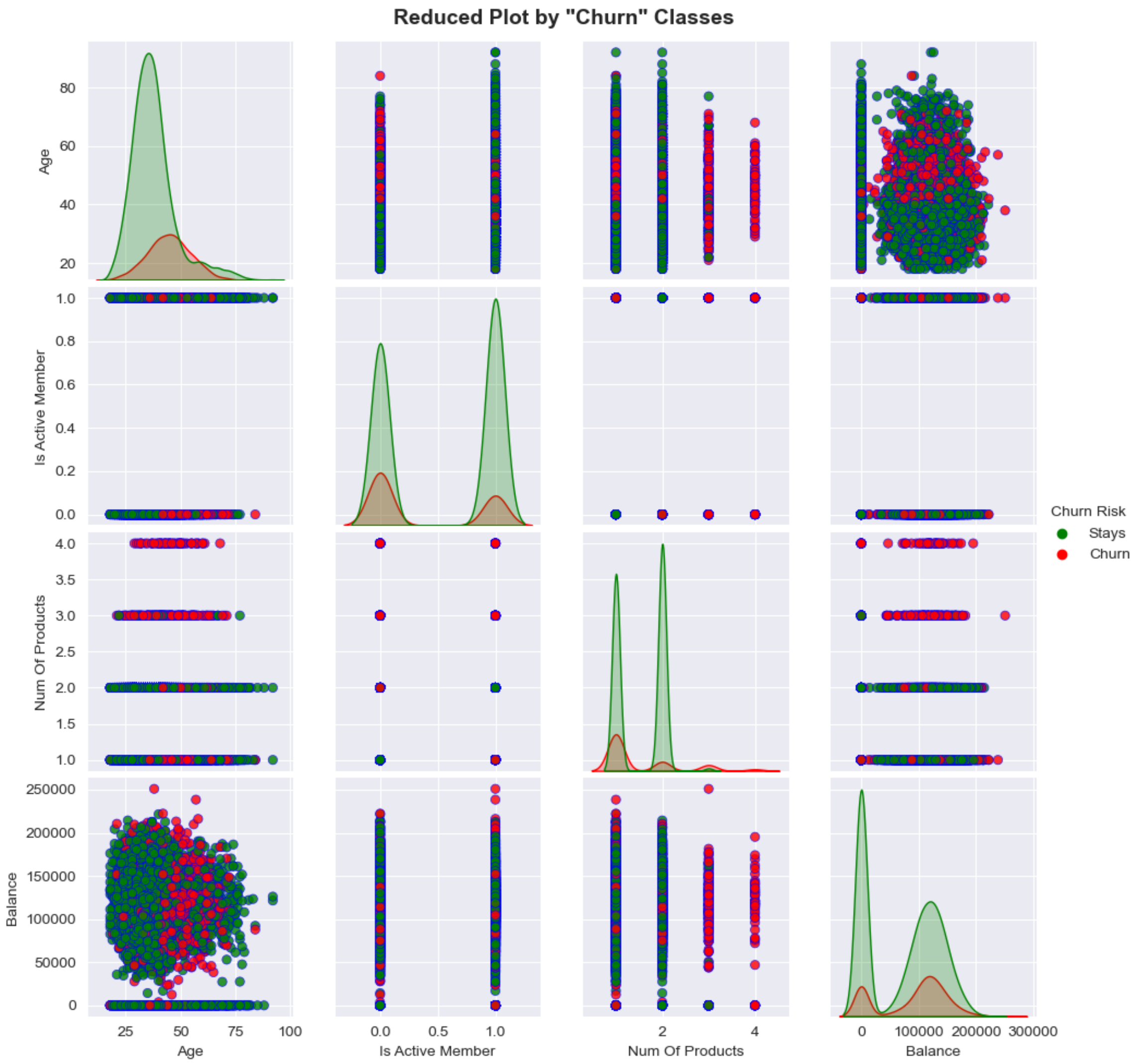


The density plots on the diagonal make it easier to compare these distributions. We can notice that only few features have slightly different distributions. For example, from the density plot for Age, it could be seen that older people have slightly higher tendecy to leave the bank.

Let’s reduce the clutter by plotting only four features:

- Age,
- Is Active Member,
- Num Of Products
- Balance

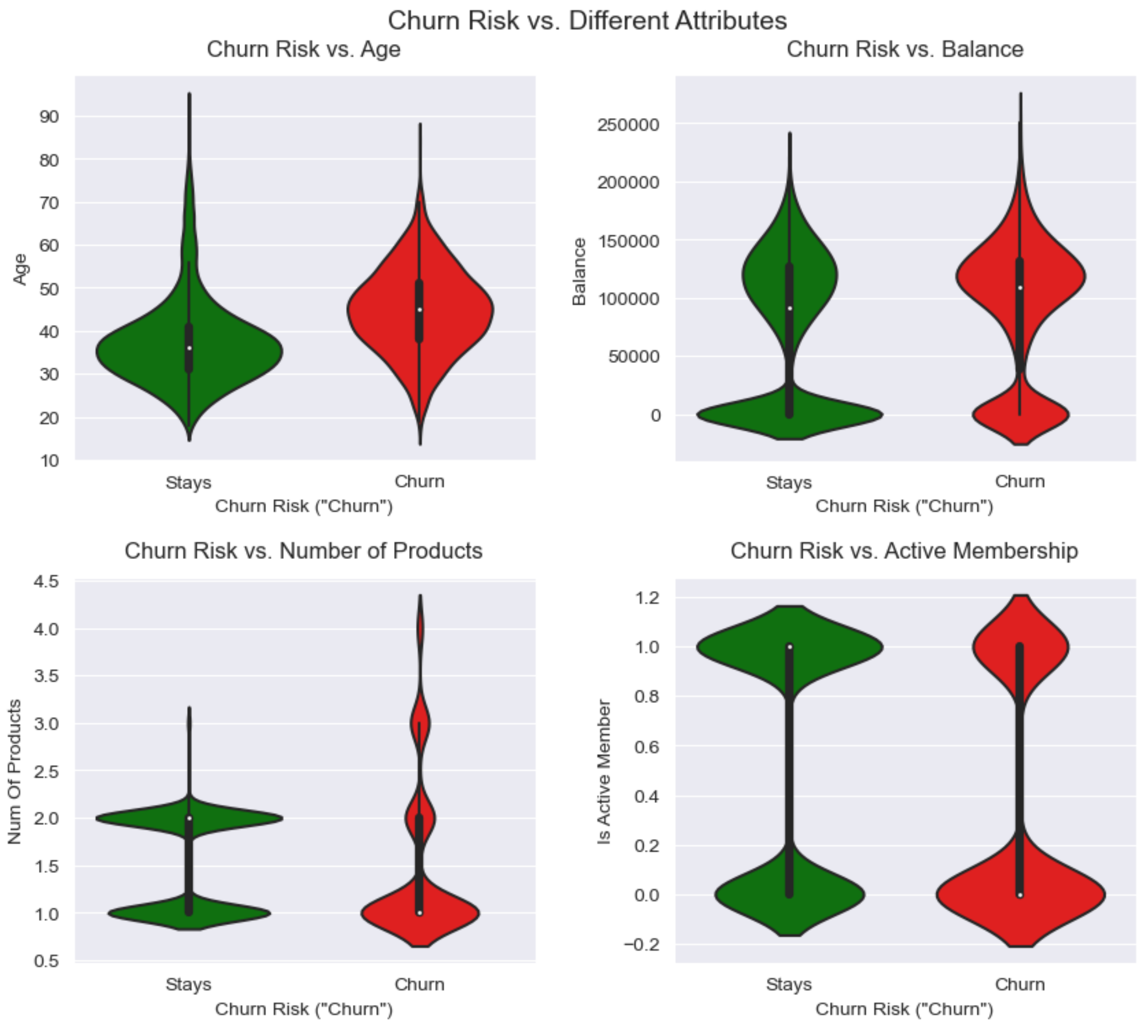
In [27]:
```
# Plot Seaborn's pairplot
g = sns.pairplot(df, hue='Churn',
                 vars=['Age', 'Is Active Member', 'Num Of Products', 'Balanc
                 palette={0 : 'green',
                          1 : 'red'},
                 plot_kws={'alpha' : 0.8, 'edgecolor' : 'b', 'linewidth' : 0

fig = g.fig
fig.subplots_adjust(top=0.95, wspace=0.2)
fig.suptitle('Reduced Plot by "Churn" Classes',
             fontsize=14,
             fontweight='bold')

# Update the legend
new_title = 'Churn Risk'
g._legend.set_title(new_title)
# replace labels
new_labels = ['Stays', 'Churn']
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)

plt.show()
```



From density plots we can see that older customers and customer with more products more often leaving the bank.

Violin Plots

In [28]:
```
# Segment age by Churn and plot distributions
#  “categorical” variable Churn is a numeric
#  for plotting purposes only we will change it to real categorical variable

# Define palette
my_pal = {'Stays': 'green', 'Churn': 'red'}
# Convert to categorical
hr = {0: 'Stays', 1: 'Churn'}
churn = df['Churn'].map(hr)

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(10, 8))
fig.suptitle('Churn Risk vs. Different Attributes', fontsize=14)
fig.subplots_adjust(top=0.92, wspace=0.3, hspace=0.3)

sns.violinplot(x=churn,
               y=df['Age'],
               order=['Stays', 'Churn'],
               palette=my_pal,
               ax=ax1)

ax1.set_title('Churn Risk vs. Age', fontsize=12, pad=10)
ax1.set_ylabel('Age',size=10)
ax1.set_xlabel('Churn Risk ("Churn")', size=10)

sns.violinplot(x=churn,
               y=df['Balance'],
               order=['Stays', 'Churn'],
               palette=my_pal,
               ax=ax2)

ax2.set_title('Churn Risk vs. Balance', fontsize=12, pad=10)
ax2.set_ylabel('Balance',size=10)
ax2.set_xlabel('Churn Risk ("Churn")', size=10)

sns.violinplot(x=churn,
               y=df['Num Of Products'],
               order=['Stays', 'Churn'],
               palette=my_pal,
               ax=ax3)

ax3.set_title('Churn Risk vs. Number of Products', fontsize=12, pad=10)
ax3.set_ylabel('Num Of Products',size=10)
ax3.set_xlabel('Churn Risk ("Churn")', size=10)

sns.violinplot(x=churn,
               y=df['Is Active Member'],
               order=['Stays', 'Churn'],
               palette=my_pal,
               ax=ax4)

ax4.set_title('Churn Risk vs. Active Membership', fontsize=12, pad=10)
ax4.set_ylabel('Is Active Member',size=10)
ax4.set_xlabel('Churn Risk ("Churn")', size=10)
plt.show()
```



Violin plots are confirming the earlier statement that older customers and customer with more products are more likely to leave the bank.

In [29]:
```
# Define our target variable
y = df.Churn
```

In [30]:
```
y.shape
```

Out[30]:
```
(10000,)
```

Let's define a small helper funtcion which displays count and percentage per class of the target feature.

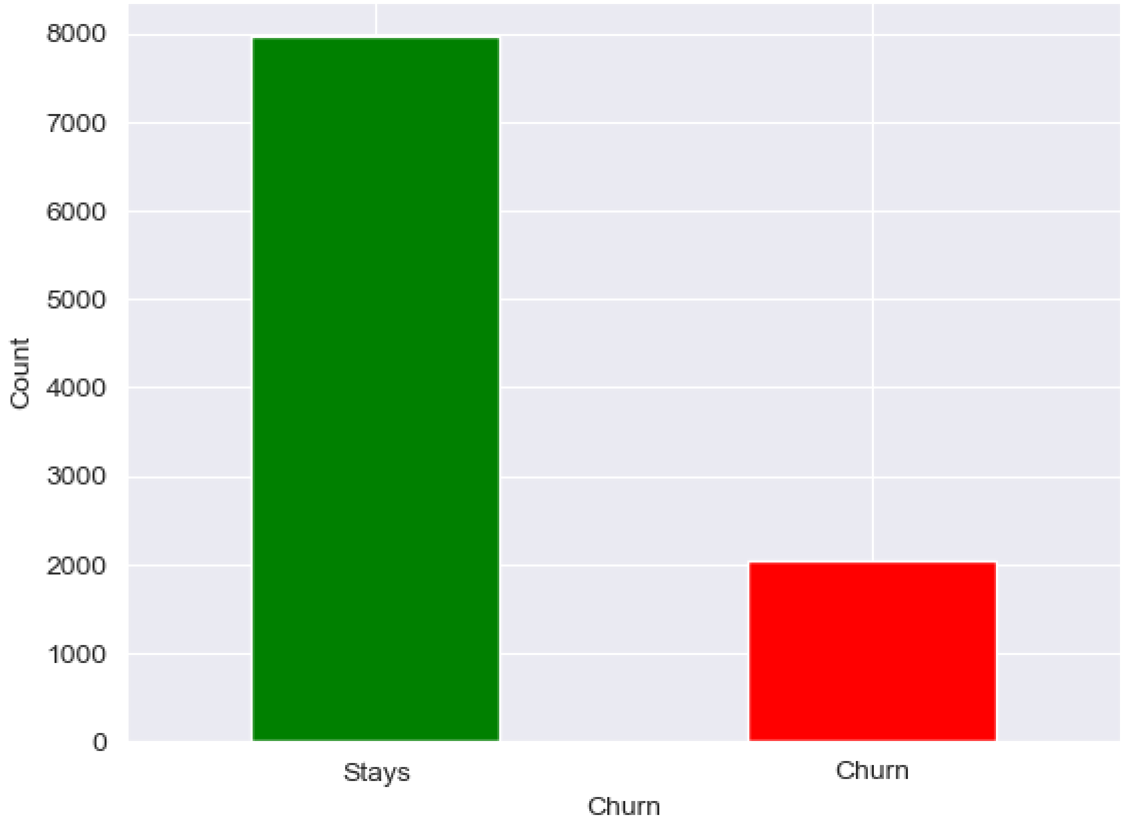
In [31]:
```
# Function to display count and percentage per class of target feature
def class_count(a):
    counter=Counter(a)
    kv=[list(counter.keys()),list(counter.values())]
    dff = pd.DataFrame(np.array(kv).T, columns=['Churn','Count'])
    dff['Count'] = dff['Count'].astype('int64')
    dff['%'] = round(dff['Count'] / a.shape[0] * 100, 2)
    return dff.sort_values('Count',ascending=False)
```

In [32]:
```python
# Let's use the function
dfcc = class_count(y)
dfcc
```

Out[32]:

	Churn	Count	%
1	0	7963	79.63
0	1	2037	20.37

In [33]:
```python
# Plot distribution of target variable, Exited column

labels=['Stays', 'Churn']
dfcc.plot.bar(x='Churn', y='Count', color=['g', 'r'], legend=False)
plt.xticks(dfcc['Churn'], labels, rotation=0)
plt.ylabel('Count')
plt.show()
```



We can see that our dataset is imbalanced. The majority class, "Stays" (0), has around 80% data points and the minority class, "Churn" (1), has around 20% datapoints.

To address this, in our machine learning algorithms we will use SMOTE (Synthetic Minority Over-sampling Technique).

Finalizing the Dataframe

In [34]:
```
df.head()
```

Out[34]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estir S
0	619	France	Female	42	2	0.00	1	1	1	1013
1	608	Spain	Female	41	1	83807.86	1	0	1	1125
2	502	France	Female	42	8	159660.80	3	1	0	1139
3	699	France	Female	39	1	0.00	2	0	0	938
4	850	Spain	Female	43	2	125510.82	1	1	1	790

In [35]:
```
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   Num Of Products    10000 non-null  int64
 7   Has Credit Card    10000 non-null  int64
 8   Is Active Member   10000 non-null  int64
 9   Estimated Salary   10000 non-null  float64
 10  Churn              10000 non-null  int64
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

Our dataframe looks good and it is ready to be saved.

Save the dataframe as the analytical base table

In [36]:
```
# Save analytical base table
### df.to_csv('ELCOT\Desktop\csv file_base_table.csv', index=None)
```