Accesing the dataset from Data folder

In [198…
```
# importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
```

1.Load and explore dataset

In [199…
```
df = pd.read_csv('Data/Churn_Modelling.csv')
df.head()
```

Out[199…

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
0	1	15634602	Hargrave	619	France	Female	42.0	2	0.00	1	1.0
1	2	15647311	Hill	608	Spain	Female	41.0	1	83807.86	1	0.0
2	3	15619304	Onio	502	France	Female	42.0	8	159660.80	3	1.0
3	4	15701354	Boni	699	France	Female	39.0	1	0.00	2	0.0
4	5	15737888	Mitchell	850	Spain	Female	43.0	2	125510.82	1	NaN

In [200…
```
df.shape
```

Out[200…
```
(10002, 14)
```

In [201…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10002 entries, 0 to 10001
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10002 non-null  int64
 1   CustomerId       10002 non-null  int64
 2   Surname          10002 non-null  object
 3   CreditScore      10002 non-null  int64
 4   Geography        10001 non-null  object
 5   Gender           10002 non-null  object
 6   Age              10001 non-null  float64
 7   Tenure           10002 non-null  int64
 8   Balance          10002 non-null  float64
 9   NumOfProducts    10002 non-null  int64
 10  HasCrCard        10001 non-null  float64
 11  IsActiveMember   10001 non-null  float64
 12  EstimatedSalary  10002 non-null  float64
 13  Exited           10002 non-null  int64
dtypes: float64(5), int64(6), object(3)
memory usage: 1.1+ MB
```

In [202…

```
df.describe()
```

Out[202…

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	Is/
count	10002.000000	1.000200e+04	10002.000000	10001.000000	10002.000000	10002.000000	10002.000000	10001.000000	
mean	5001.499600	1.569093e+07	650.555089	38.922311	5.012498	76491.112875	1.530194	0.705529	
std	2887.472338	7.193177e+04	96.661615	10.487200	2.891973	62393.474144	0.581639	0.455827	
min	1.000000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.000000	
25%	2501.250000	1.562852e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.000000	
50%	5001.500000	1.569073e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.000000	
75%	7501.750000	1.575323e+07	718.000000	44.000000	7.000000	127647.840000	2.000000	1.000000	
max	10000.000000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.000000	

In [203…
```
df.isnull().sum()
```

Out[203…
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          1
Gender             0
Age                1
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          1
IsActiveMember     1
EstimatedSalary    0
Exited             0
dtype: int64
```

In [204…
```
df.duplicated().sum()
```

Out[204…
```
2
```

In [205…
```
df['Exited'].value_counts()
```

Out[205…
```
Exited
0    7964
1    2038
Name: count, dtype: int64
```

In [206…
```
df['Exited'].value_counts(normalize=True)*100
```

Out[206…
```
Exited
0    79.624075
1    20.375925
Name: proportion, dtype: float64
```

As observed, our dataset comprises two classes with a distribution ratio of 79,6% to 20,4%, indicating a significant class imbalance. To address this issue, techniques such as oversampling the minority class or undersampling the majority class will be applied to balance the dataset in subsequent steps. However, prior to implementing any balancing methods, we will focus on data cleaning.

This involves handling missing values, identifying and removing duplicates, and addressing potential outliers to ensure the dataset is consistent and suitable for modeling.

2. Data Cleaning

In [207…
```
# As first step, drop irrelevant columns: RowNumber, CustomerId, Surname
df = df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)
print(df.columns)
```

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

In [208…
```
# Filters rows with any missing values
df[df.isnull().any(axis=1)]
```

Out[208…

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Ex
4	850	Spain	Female	43.0	2	125510.82	1	NaN	1.0	79084.10	
6	822	NaN	Male	50.0	7	0.00	2	1.0	1.0	10062.80	
8	501	France	Male	44.0	4	142051.07	2	0.0	NaN	74940.50	
9	684	France	Male	NaN	2	134603.88	1	1.0	1.0	71725.73	

Considering the dataset contains 10,002 rows and specific NaN values in the filtered subset, it is acceptable to:

1. Delete rows with NaN values in the `Age` and `HasCrCard` columns.
2. Replace NaN values in the `Geography` column with most frequent value.
3. Set NaN values in the `IsActiveMember` column to `1.0` becouse this client have `Balance` greater than 0 and `Exited` = "0".

In [209…
```
# Step 1: Drop rows with NaN in 'Age' and 'HasCrCard'
df_cleaned = df.dropna(subset=['Age', 'HasCrCard'])
```

```
# Step 2: Fill missing values in 'Geography' using the mode
geography_mode = df_cleaned['Geography'].mode()[0]  # Use mode from 'df_cleaned'
df_cleaned['Geography'] = df_cleaned['Geography'].fillna(geography_mode)

# Step 3: Replace NaN in 'IsActiveMember' with '1.0'
df_cleaned['IsActiveMember'] = df_cleaned['IsActiveMember'].fillna("1.0")

# Step 4: Verify that all missing values have been handled
print("Check for Missing Values after cleaning:\n", df_cleaned.isnull().sum())
```

```
Check for Missing Values after cleaning:
 CreditScore        0
Geography           0
Gender              0
Age                 0
Tenure              0
Balance             0
NumOfProducts       0
HasCrCard           0
IsActiveMember      0
EstimatedSalary     0
Exited              0
dtype: int64
```

In [210…

```
# Drop duplicates
df_cleaned = df_cleaned.drop_duplicates()
print("Check for Duplicates after cleaning:\n", df_cleaned.duplicated().sum())
```

```
Check for Duplicates after cleaning:
 0
```

In [211…

```
df_cleaned.head()
```

Out[211…

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Ex
0	619	France	Female	42.0	2	0.00	1	1.0	1.0	101348.88	
1	608	Spain	Female	41.0	1	83807.86	1	0.0	1.0	112542.58	
2	502	France	Female	42.0	8	159660.80	3	1.0	0.0	113931.57	
3	699	France	Female	39.0	1	0.00	2	0.0	0.0	93826.63	
5	645	Spain	Male	44.0	8	113755.78	2	1.0	0.0	149756.71	

In [212…
```
df_cleaned.shape
```

Out[212…
```
(9998, 11)
```

3. Exploratory Data Analysis (EDA)

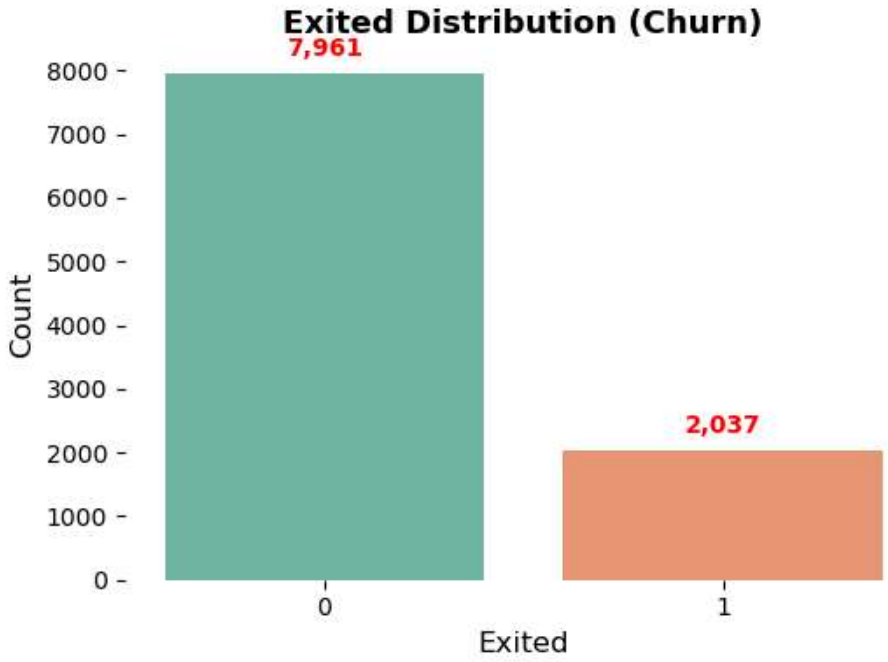
For simplicity in code we will use `df_cleaned` as `df`.

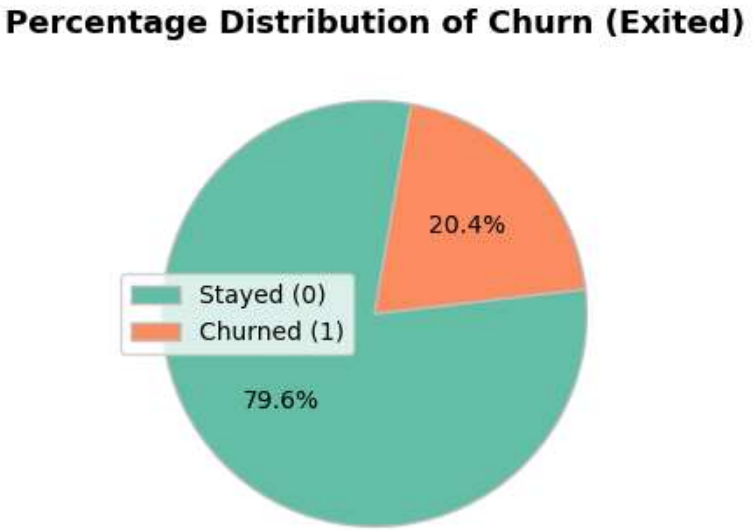
In [213…
```
df = df_cleaned
```

3.1 Univariate Analysis

In [214…
```
# Count and percentage distribution of 'Exited'

exited_counts = df['Exited'].value_counts()
exited_percentage = df['Exited'].value_counts(normalize=True) * 100
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

# Subplot 1: Bar plot for Exited count distribution
sns.barplot(x=exited_counts.index, y=exited_counts, ax=axes[0], palette='Set2')
axes[0].set_title('Exited Distribution (Churn)', fontsize=13, fontweight='bold')
axes[0].set_xlabel('Exited', fontsize=12)
axes[0].set_ylabel('Count', fontsize=12)
```

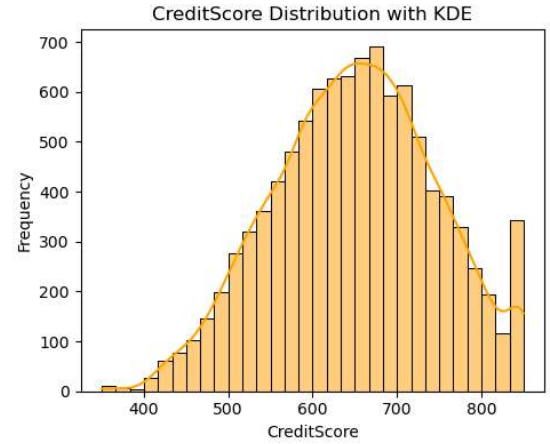
```python
# Annotate bars with exact counts
for bar in axes[0].patches:
    axes[0].annotate(f'{bar.get_height():,.0f}',  # Format count with thousands separator
                     (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                     ha='center', va='center', xytext=(0, 10), textcoords='offset points',
                     fontsize=10, color='red', weight='bold')
# Remove unnecessary spines for cleaner visuals
sns.despine(left=True, bottom=True)

# Subplot 2: Pie chart for percentage distribution
axes[1].pie(exited_percentage,
            autopct='%1.1f%%',
            colors=sns.color_palette('Set2'),
            startangle=80,
            wedgeprops={'edgecolor': 'silver'})  # Add black edge for better visibility
axes[1].set_title('Percentage Distribution of Churn (Exited)', fontsize=13, fontweight='bold')

plt.legend(['Stayed (0)', 'Churned (1)'])
plt.tight_layout()  # Ensure plots do not overlap
plt.show()
```

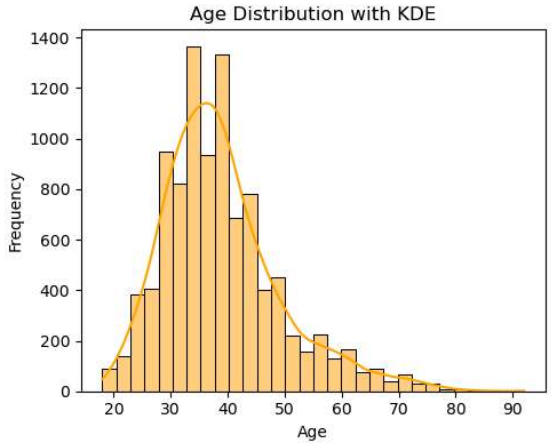


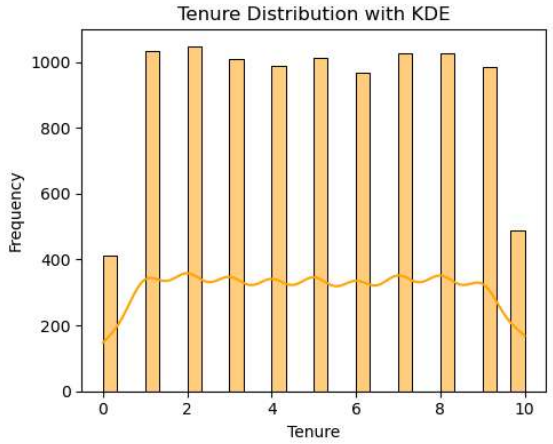


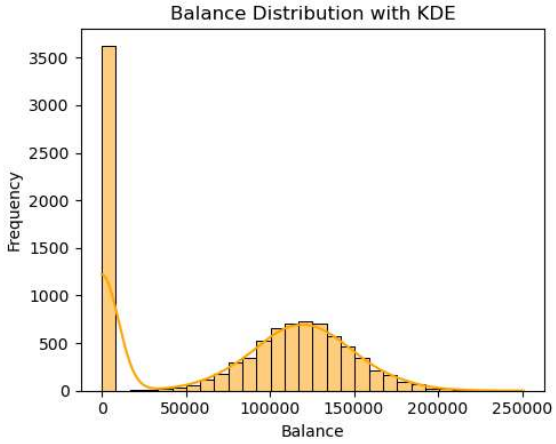
In [215…

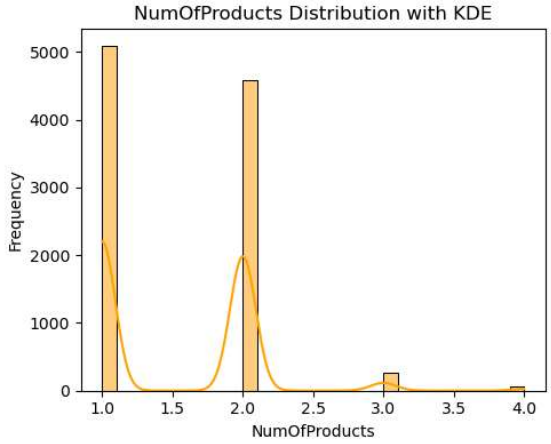
```python
# Function to perform univariate analysis for different numeric columns

def univariate_analysis(data, columns, grid_size=(3, 3), figsize=(15, 12)):
    fig, axes = plt.subplots(*grid_size, figsize=figsize)  # Create subplot grid
    axes = axes.flatten()  # Flatten axes for easy iteration

    for ax, column in zip(axes, columns):
        sns.histplot(data[column], kde=True, bins=30, color='orange' , ax=ax)
        ax.set_title(f'{column.replace("_", " ")} Distribution with KDE', fontsize=12)
        ax.set_xlabel(column.replace('_', ' '), fontsize=10)
        ax.set_ylabel('Frequency', fontsize=10)
    # Remove unused axes if columns < grid_size
    for ax in axes[len(columns):]:
        ax.axis('off')
    plt.tight_layout()
    plt.show()

# Columns to analyze
columns_to_analyze = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'Est

# Perform univariate analysis
univariate_analysis(df, columns_to_analyze)
```

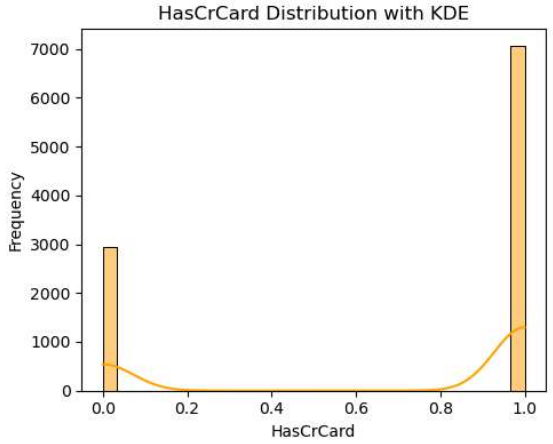

CreditScore Distribution with KDE
Frequency
CreditScore

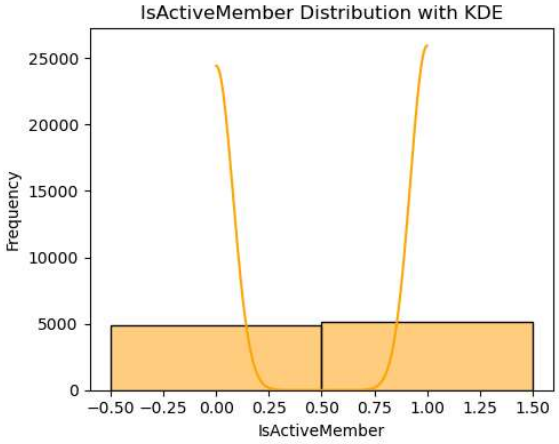

Age Distribution with KDE
Frequency
Age

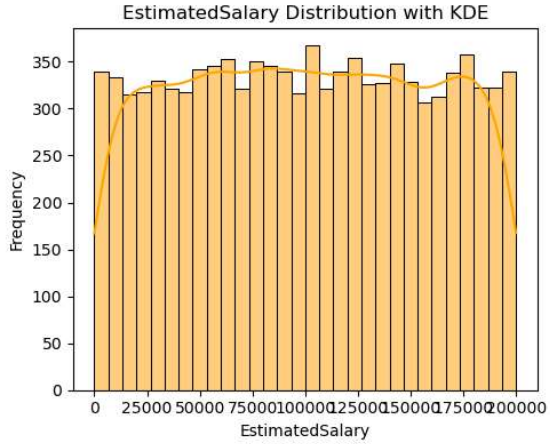

Tenure Distribution with KDE
Frequency
Tenure


Balance Distribution with KDE
Frequency
Balance


NumOfProducts Distribution with KDE
Frequency
NumOfProducts


HasCrCard Distribution with KDE
Frequency
HasCrCard


IsActiveMember Distribution with KDE
Frequency
IsActiveMember


EstimatedSalary Distribution with KDE
Frequency
EstimatedSalary

Feature Distributions with KDE

This chart visualizes the distributions of various features in the dataset using histograms with Kernel Density Estimates (KDE). Below is a description of each feature's distribution:

Features

1. **CreditScore**
   - Distribution: Approximately normal.
   - Observations: Most customers have a credit score between 500 and 800.
2. **Age**
   - Distribution: Right-skewed.
   - Observations: The majority of customers are between 30 and 50 years old.
3. **Tenure**
   - Distribution: Uniform.
   - Observations: Customers are evenly distributed across different tenure levels.
4. **Balance**
   - Distribution: Many customers have a balance of zero.
   - Observations: Balances range up to approximately 250,000, with a significant number clustered at zero.
5. **NumOfProducts**
   - Distribution: Peaks at 1 and 2 products.
   - Observations: Very few customers have 3 or 4 products.
6. **HasCrCard**
   - Distribution: Binary (0 or 1).
   - Observations: The majority of customers possess a credit card (`1`).
7. **IsActiveMember**
   - Distribution: Binary (0 or 1).

- Observations: Slightly more customers are active members (`1`) than inactive ones.

8. **EstimatedSalary**

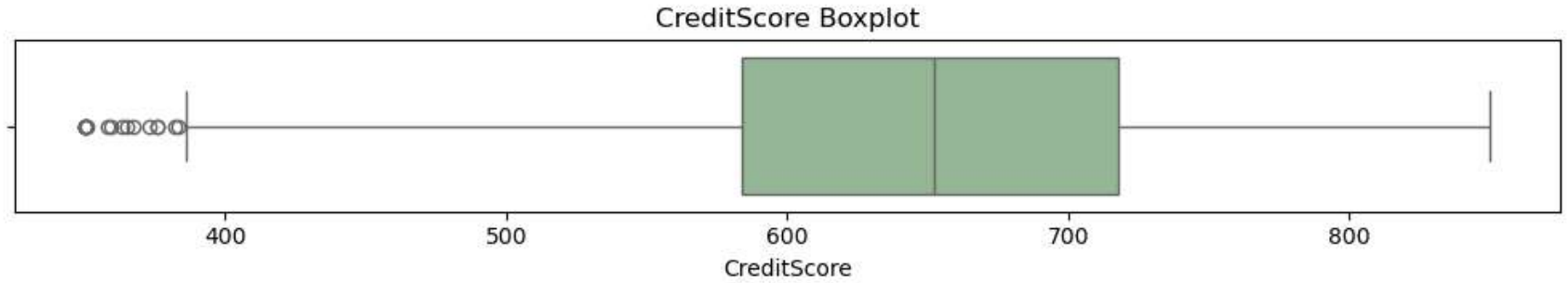
   - Distribution: Relatively uniform.
   - Observations: Salaries are spread evenly across the range, with a slight decrease at the higher end.

Insights

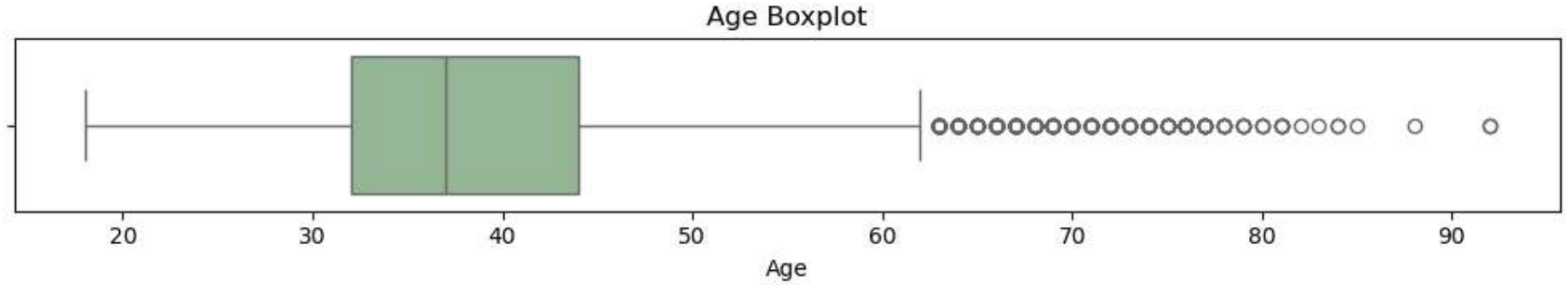
- **Balance**: A significant portion of customers have a zero balance, indicating a potential pattern or special group within the data.
- **NumOfProducts**: Most customers have limited products (1 or 2), which could influence customer retention strategies.
- **Age**: The dataset is skewed toward middle-aged customers.
- **Binary Features (HasCrCard, IsActiveMember)**: The binary distributions show a clear majority in one category.

In [216…

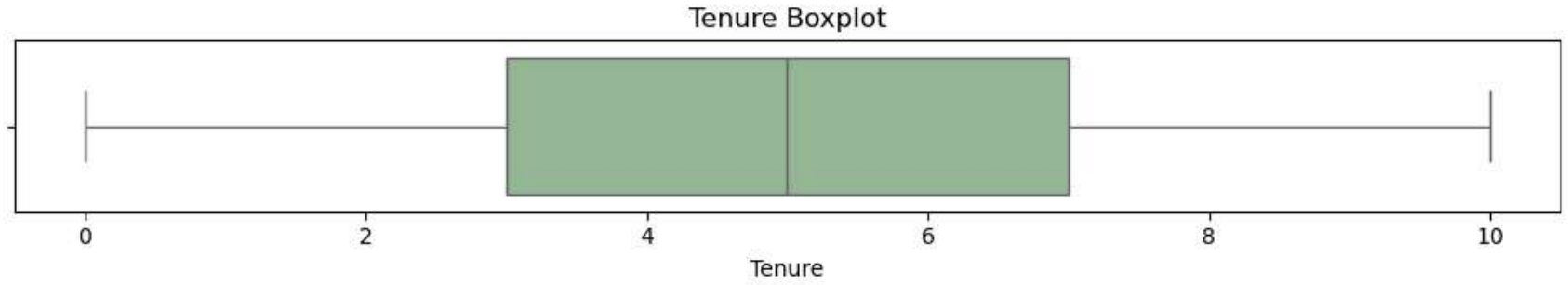
```
 # Function to perform univariate analysis for numeric columns
def univariate_analysis(data, column, title):
    plt.figure(figsize=(10, 2))

    sns.boxplot(x=data[column], color='darkseagreen')
    plt.title(f'{title} Boxplot')

    plt.tight_layout()
    plt.show()

    print(f'\nSummary Statistics for {title}:\n', data[column].describe())

columns_to_analyze = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'Est

for column in columns_to_analyze:
    univariate_analysis(df, column, column.replace('_', ' '))
```



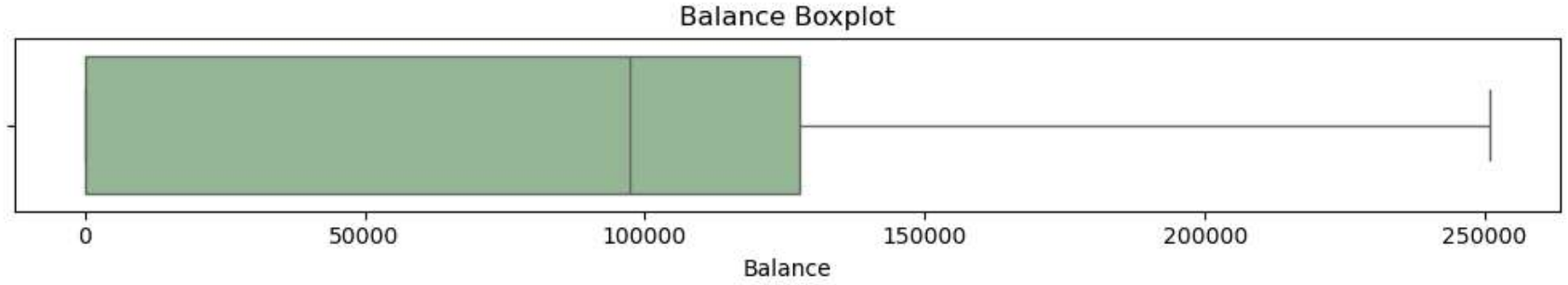

```
Summary Statistics for CreditScore:
 count    9998.000000
mean      650.505501
std        96.641794
min       350.000000
25%       584.000000
50%       652.000000
75%       717.750000
max       850.000000
Name: CreditScore, dtype: float64
```



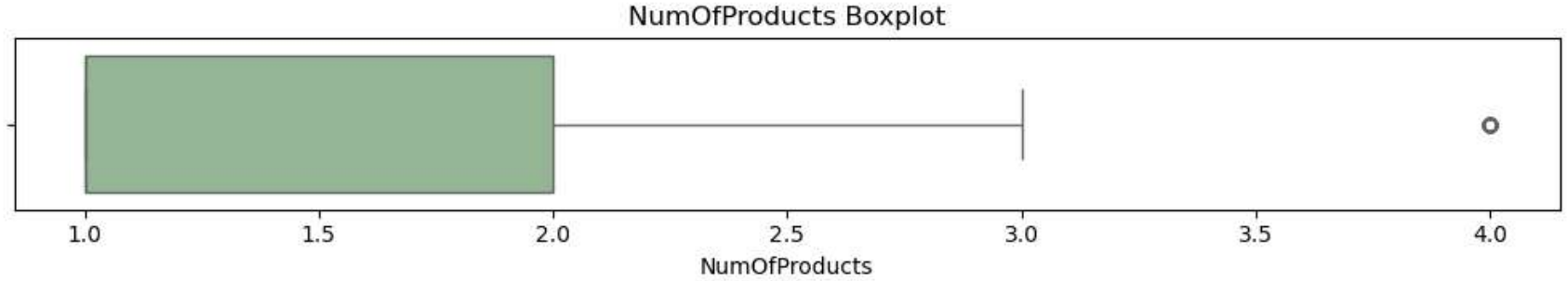

```
Summary Statistics for Age:
 count    9998.000000
mean       38.922688
std        10.488080
min        18.000000
25%        32.000000
50%        37.000000
75%        44.000000
max        92.000000
Name: Age, dtype: float64
```



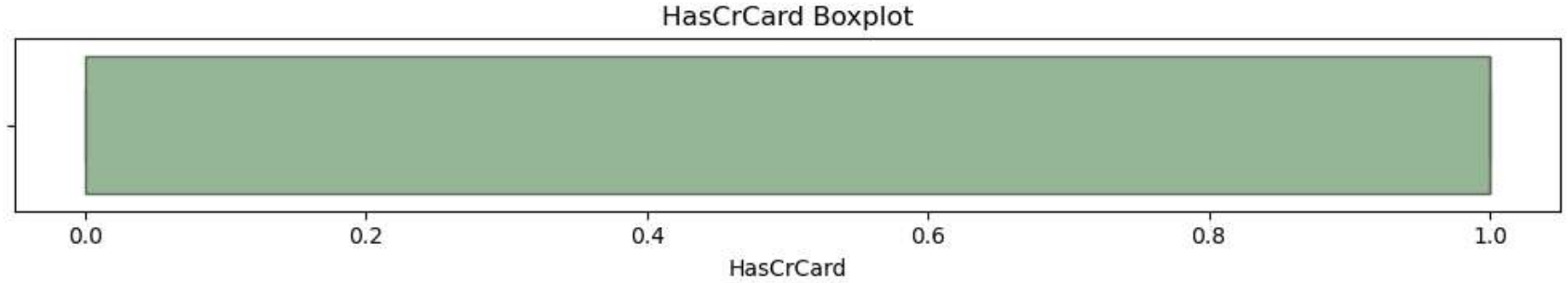

```
Summary Statistics for Tenure:
 count    9998.000000
mean        5.013403
std         2.892150
min         0.000000
25%         3.000000
50%         5.000000
75%         7.000000
max        10.000000
Name: Tenure, dtype: float64
```



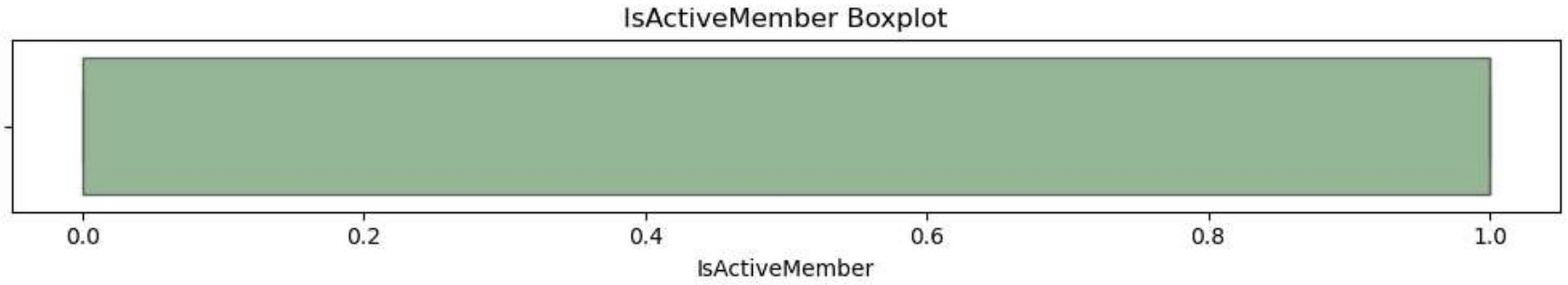

```
Summary Statistics for Balance:
 count      9998.000000
mean      76475.172853
std       62399.011964
min           0.000000
25%           0.000000
50%       97173.290000
75%      127641.417500
max      250898.090000
Name: Balance, dtype: float64
```



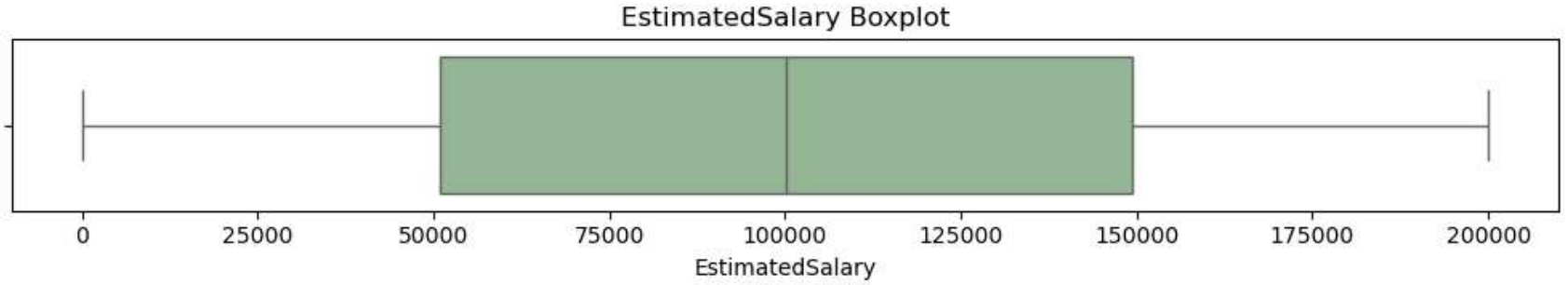

```
Summary Statistics for NumOfProducts:
 count    9998.000000
mean        1.530306
std         0.581664
min         1.000000
25%         1.000000
50%         1.000000
75%         2.000000
max         4.000000
Name: NumOfProducts, dtype: float64
```




```
Summary Statistics for HasCrCard:
 count    9998.000000
mean        0.705441
std         0.455867
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: HasCrCard, dtype: float64
```




```
Summary Statistics for IsActiveMember:
 count     9998.0
unique       3.0
top          1.0
freq      5148.0
Name: IsActiveMember, dtype: float64
```




```
Summary Statistics for EstimatedSalary:
 count      9998.000000
mean     100095.177934
std       57515.161757
min          11.580000
25%       50983.750000
50%      100218.210000
75%      149395.882500
max      199992.480000
Name: EstimatedSalary, dtype: float64
```

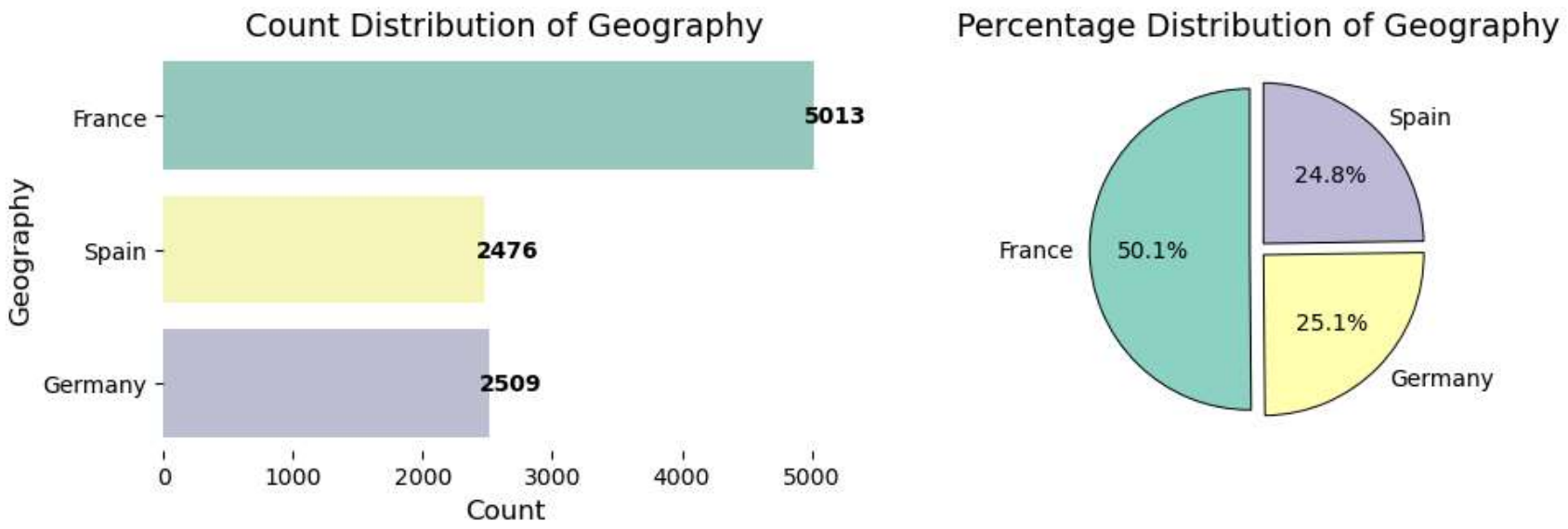
In [217…

```python
# Analyse categorical columns

def plot_categorical_distribution(column_name, data, palette='Set3'):
    unique_values = data[column_name].nunique()
```

```
    fig, axes = plt.subplots(1, 2, figsize=(10, 4))
    fig.suptitle(f"Analysis of '{column_name}'", fontsize=16, fontweight='bold')

    sns.countplot(y=column_name, data=data, palette=palette, ax=axes[0])
    axes[0].set_title(f"Count Distribution of {column_name}", fontsize=14)
    axes[0].set_xlabel('Count', fontsize=12)
    axes[0].set_ylabel(column_name, fontsize=12)

    for p in axes[0].patches:
        axes[0].annotate(f'{int(p.get_width())}',
                         (p.get_width(), p.get_y() + p.get_height() / 2),
                         ha='center', va='center',
                         xytext=(10, 0), textcoords='offset points',
                         fontsize=10, color='black', weight='bold')
    sns.despine(left=True, bottom=True)

    data[column_name].value_counts().plot.pie(
        autopct='%1.1f%%',
        colors=sns.color_palette(palette, unique_values),
        startangle=90,
        explode=[0.05] * unique_values,
        ax=axes[1],
        wedgeprops={'edgecolor': 'black', 'linewidth': 0.7}
    )
    axes[1].set_title(f"Percentage Distribution of {column_name}", fontsize=14)
    axes[1].set_ylabel('')

    plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout to fit the suptitle
    plt.show()
```

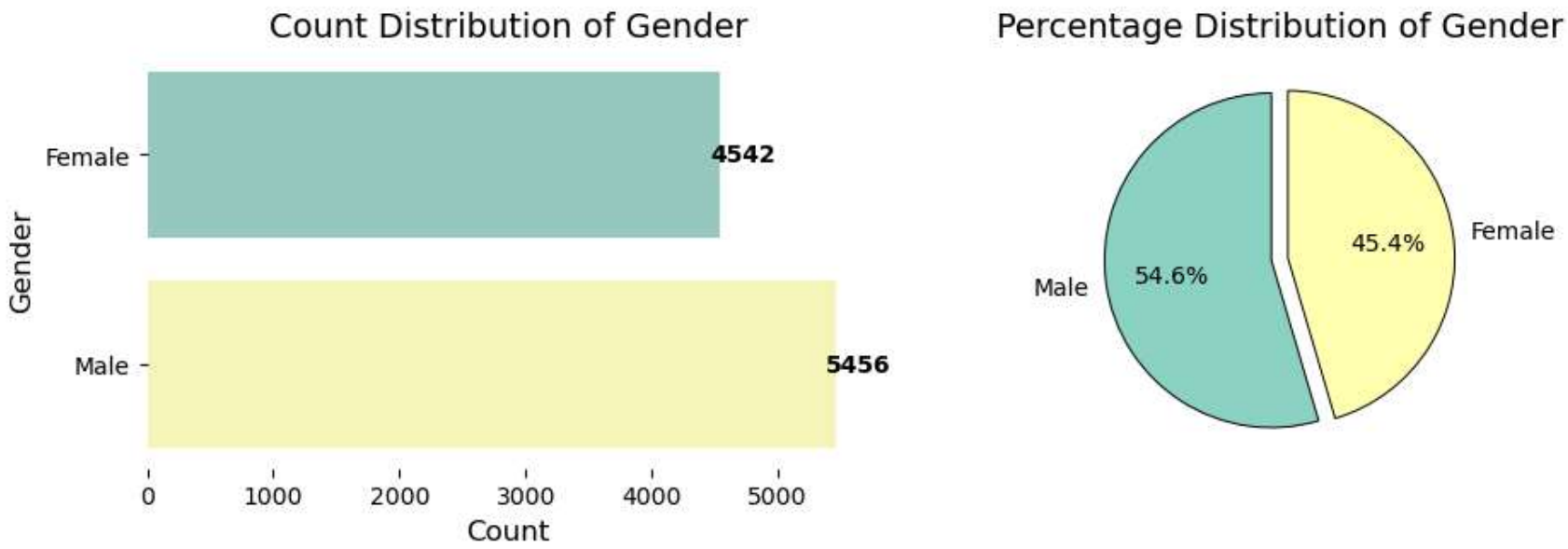
In [218…

```
plot_categorical_distribution(column_name='Geography', data=df)
```



In [219…

```
plot_categorical_distribution(column_name='Gender', data=df)
```



3.2 Bivariate Analysis

In [220…

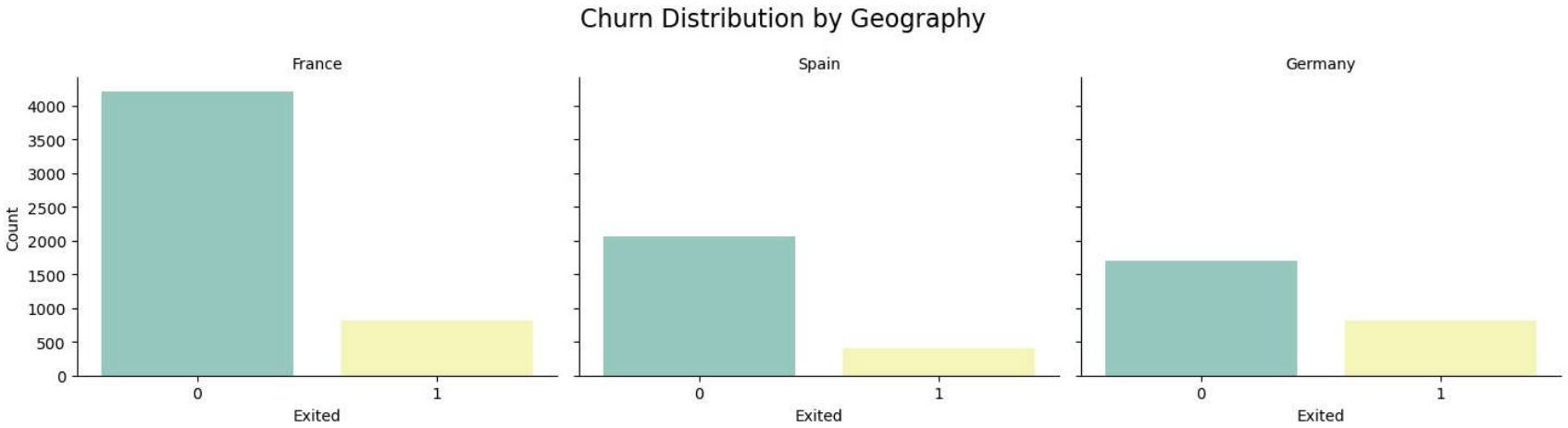
```
# Create a crosstab of Geography and Exited
contingency_table = pd.crosstab(df['Geography'], df['Exited'])

# Display the crosstab
print(contingency_table)
```

```
Exited        0    1
Geography
France     4203  810
Germany    1695  814
Spain      2063  413
```

In [221…

```
# Create a FacetGrid to visualize Geography by Exited
g = sns.FacetGrid(df, col='Geography', height=4, aspect=1.2)

g.map(sns.countplot, 'Exited', palette='Set3')
```

```
g.set_titles("{col_name}")
g.set_axis_labels("Exited", "Count")
g.fig.suptitle('Churn Distribution by Geography', fontsize=16)
plt.tight_layout()
plt.show()
```



Churn Distribution by Geography:

Observations:

France and Spain have relatively low churn rates, indicating better customer retention in these regions.

Germany appears to have a higher churn rate, suggesting that targeted strategies may be needed to address customer retention challenges in this region.

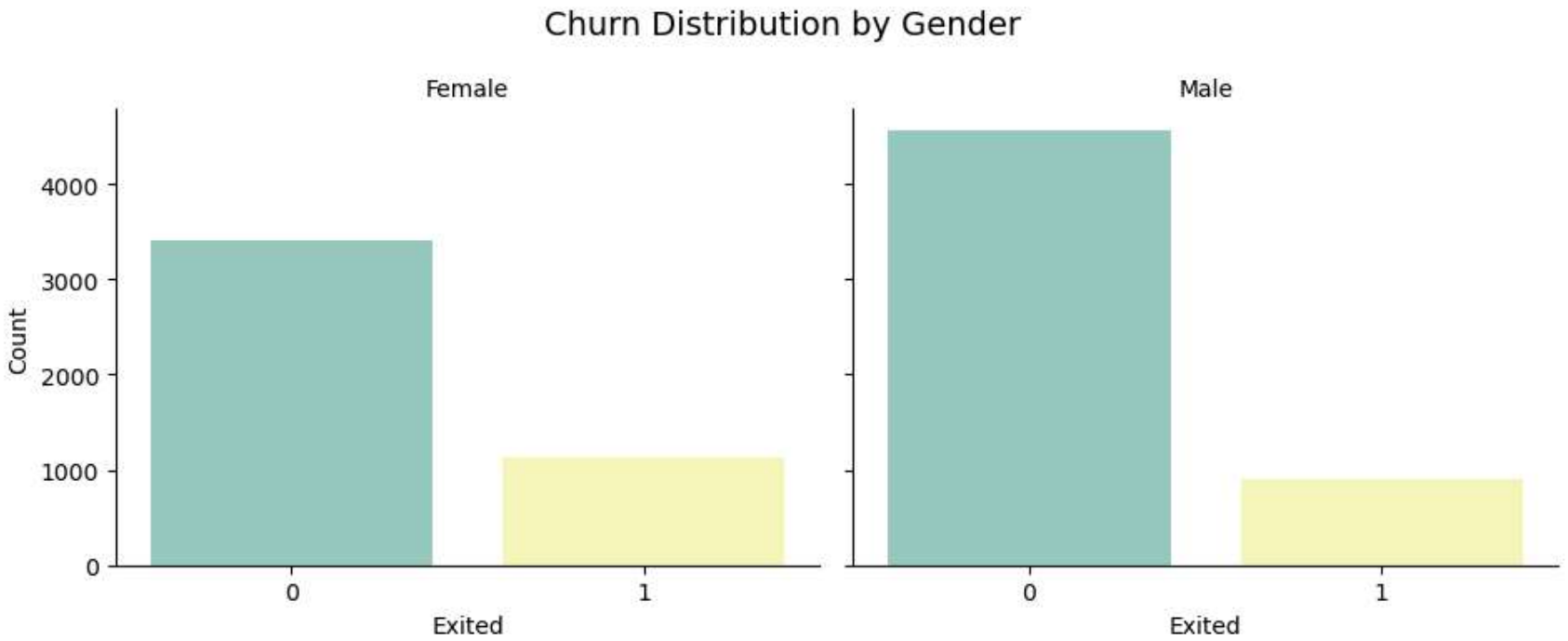
In [222…

```
# Create a crosstab of Gender and Exited
contingency_table = pd.crosstab(df['Gender'], df['Exited'])

# Display the crosstab
print(contingency_table)
```

```
Exited     0     1
Gender
Female  3403  1139
Male    4558   898
```

In [223…

```
# Create a FacetGrid to visualize Gender by Exited
g = sns.FacetGrid(df, col='Gender', height=4, aspect=1.2)

g.map(sns.countplot, 'Exited', palette='Set3')
g.set_titles("{col_name}")
g.set_axis_labels("Exited", "Count")
g.fig.suptitle('Churn Distribution by Gender', fontsize=14)
plt.tight_layout()
plt.show()
```



Churn Distribution by Gender

Observations:

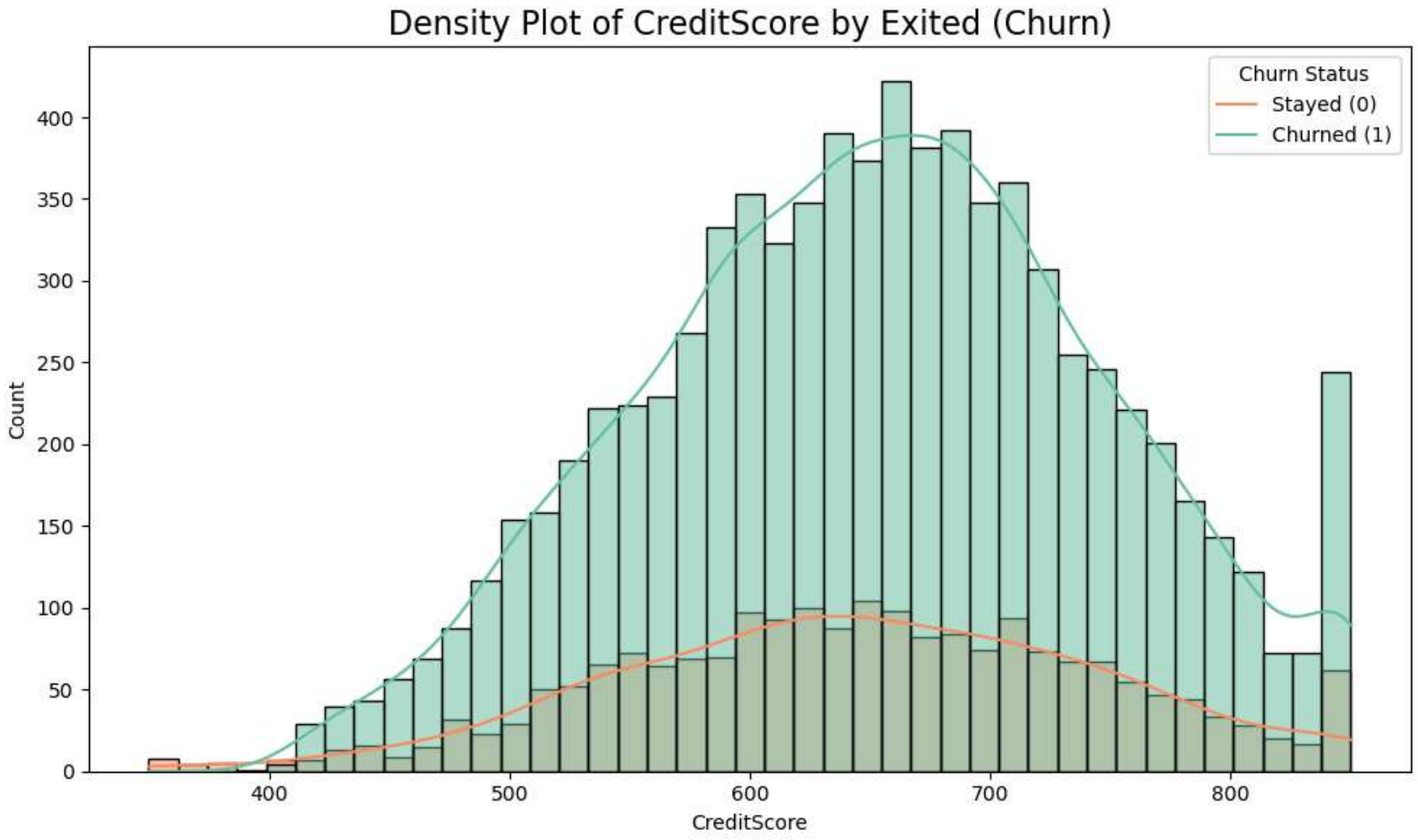
1. **Female Customers**: 25% churned (1,139 out of 4,542), while 75% did not churn.
2. **Male Customers**: 16% churned (898 out of 5,456), while 84% did not churn.

Insights:

- Female customers have a higher churn rate compared to male customers.
- Retention strategies should focus more on female customers to reduce churn.

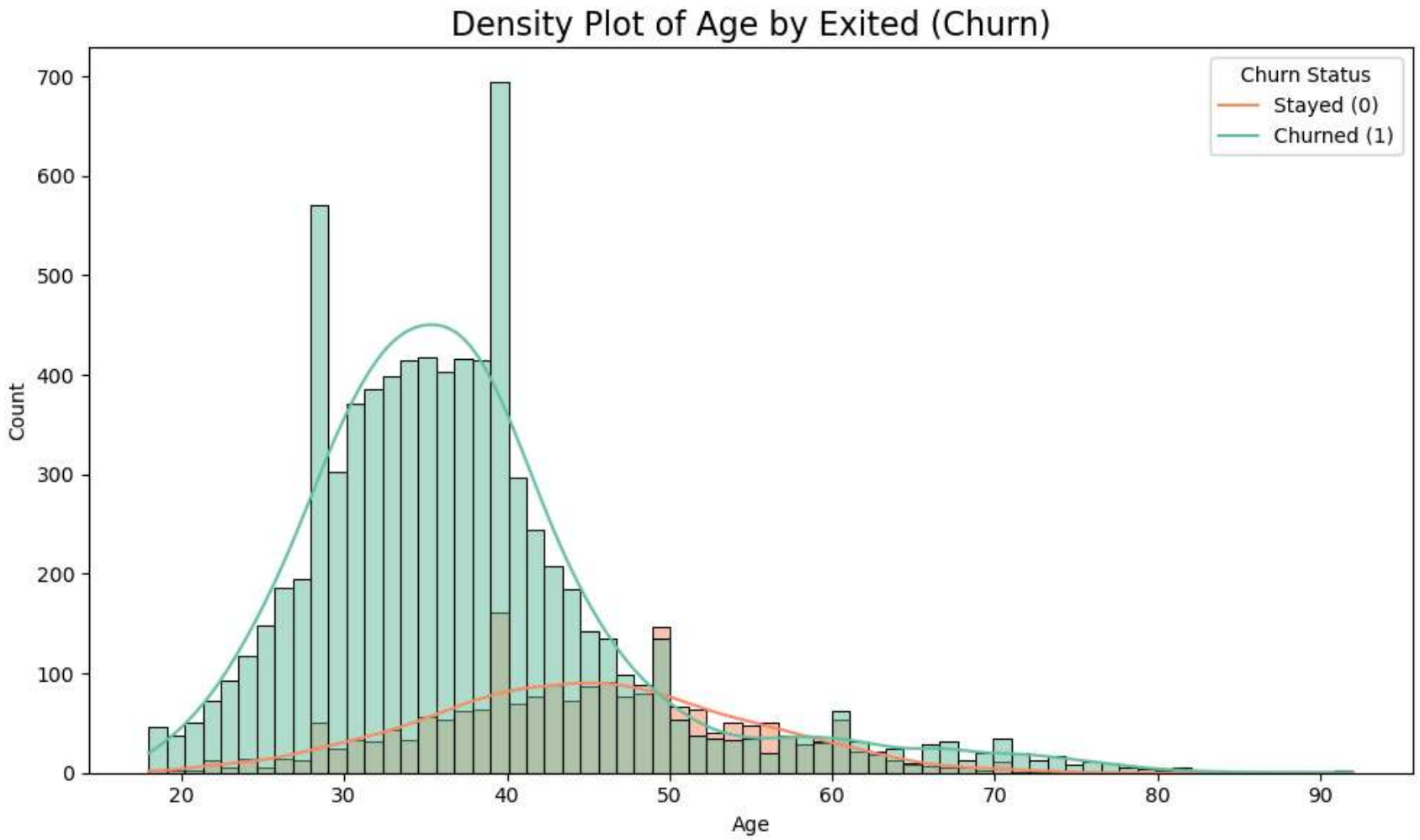
In [224…

```
# Create a histplot for CreditScore by Exited
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='CreditScore', hue='Exited', kde=True, fill=True, palette='Set2')
plt.title('Density Plot of CreditScore by Exited (Churn)', fontsize=16)
plt.xlabel('CreditScore')
plt.ylabel('Count')
plt.legend(title='Churn Status', labels=['Stayed (0)', 'Churned (1)'])
plt.tight_layout()
plt.show()
```



In [225…

```
# Plotting Age by Exited
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Age', hue='Exited', kde=True, fill=True, palette='Set2')
plt.title('Density Plot of Age by Exited (Churn)', fontsize=16)
plt.xlabel('Age')
plt.ylabel('Count')
plt.legend(title='Churn Status', labels=['Stayed (0)', 'Churned (1)'])
plt.tight_layout()
plt.show()
```



Density Plot of Age by Exited (Churn)

Summary:

This density plot visualizes the age distribution of customers, separated by churn status (`Exited`).

- **Churned Customers (Exited = 1)**:
  - Higher density in the age range of **40 to 60**, with a peak around the late 40s.

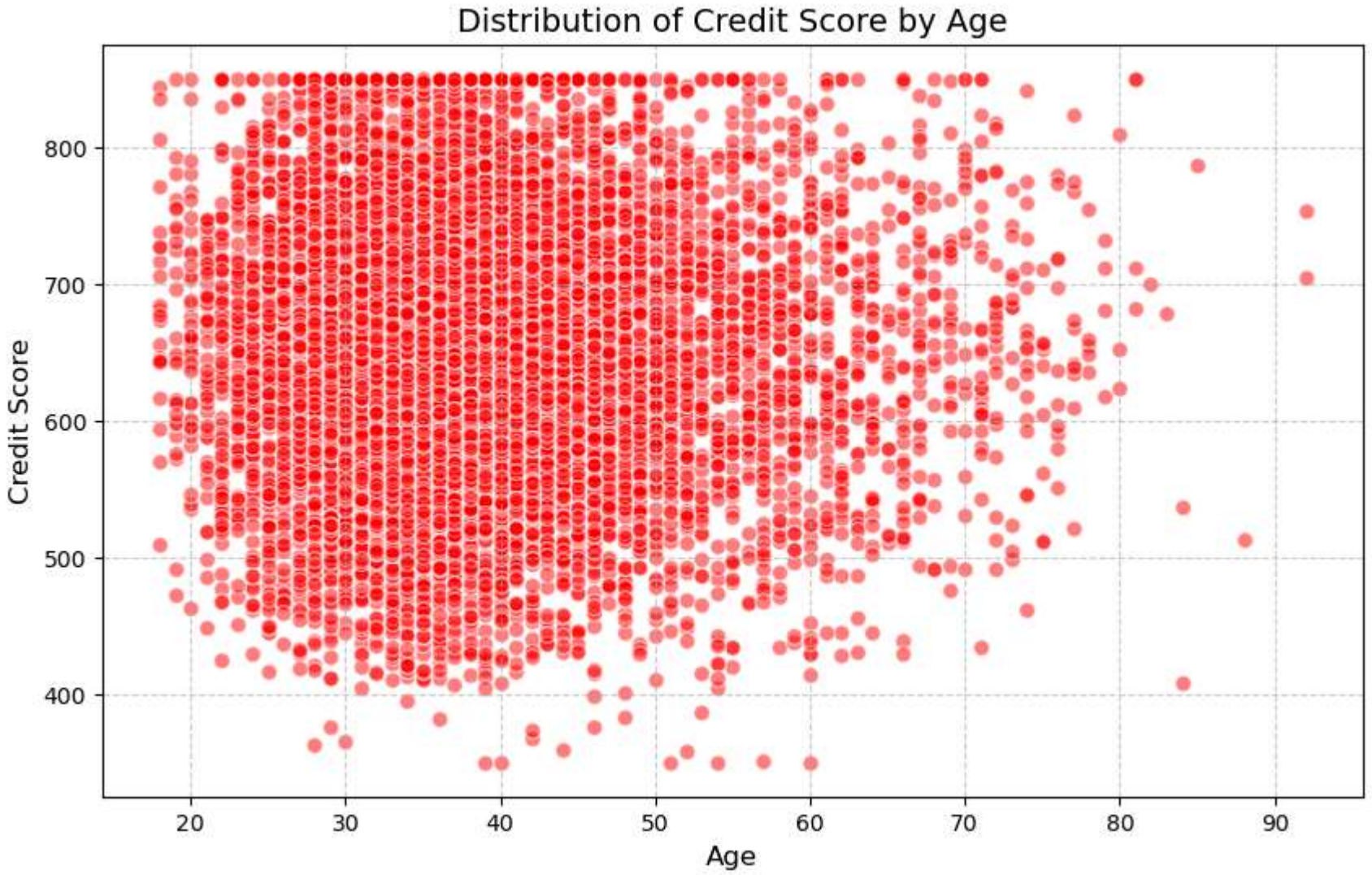
    - Suggests older customers are more likely to churn.
- **Non-Churned Customers (Exited = 0)**:
    - Distributed more evenly, with a peak around the **30s** and gradual decline thereafter.
    - Younger customers tend to stay longer.

Key Insight:

The likelihood of churn increases with age, especially for customers in the 40–60 age range. Retention strategies should focus on addressing the needs of this age group.

In [226…

```
# Plotting CreditScore by Age distribution
plt.figure(figsize=(10, 6))
plt.scatter(df['Age'], df['CreditScore'], alpha=0.5, c='red', edgecolors='w', s=50)
plt.title("Distribution of Credit Score by Age", fontsize=14)
plt.xlabel("Age", fontsize=12)
plt.ylabel("Credit Score", fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



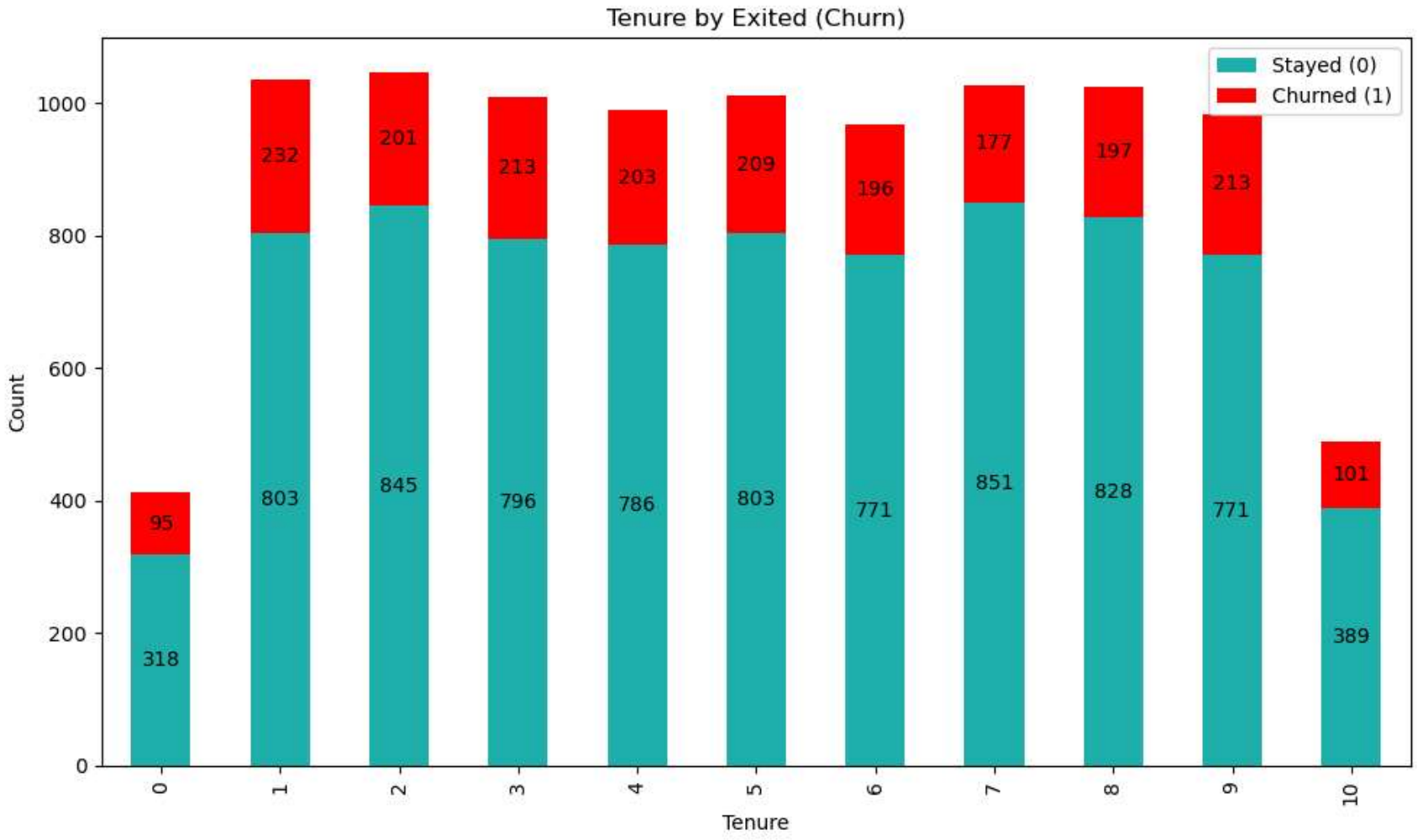
The distribution shows that credit scores are relatively independent of age, though the population skews younger.

Retention or credit improvement strategies may not require age-specific segmentation for credit scores.

In [227…

```
# Group the data by Tenure and Exited to get counts
tenure_exited_counts = df.groupby(['Tenure', 'Exited']).size().unstack()
ax = tenure_exited_counts.plot(kind='bar', stacked=True, figsize=(10, 6), color=['lightseagreen', 'red'])
```

```
plt.title('Tenure by Exited (Churn)')
plt.xlabel('Tenure')
plt.ylabel('Count')
plt.legend(['Stayed (0)', 'Churned (1)'])
plt.tight_layout()

for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    if height > 0:  # Avoid labels for zero heights
        x = p.get_x() + width / 2
        y = p.get_y() + height / 2
        ax.annotate(f'{int(height)}', (x, y), ha='center', va='center')

plt.show()
```



Tenure by Exited (Churn)

Summary:

This stacked bar chart visualizes the churn (`Exited`) distribution across different tenure levels.

- **Key Observations**:

    1. **Tenure = 0**:

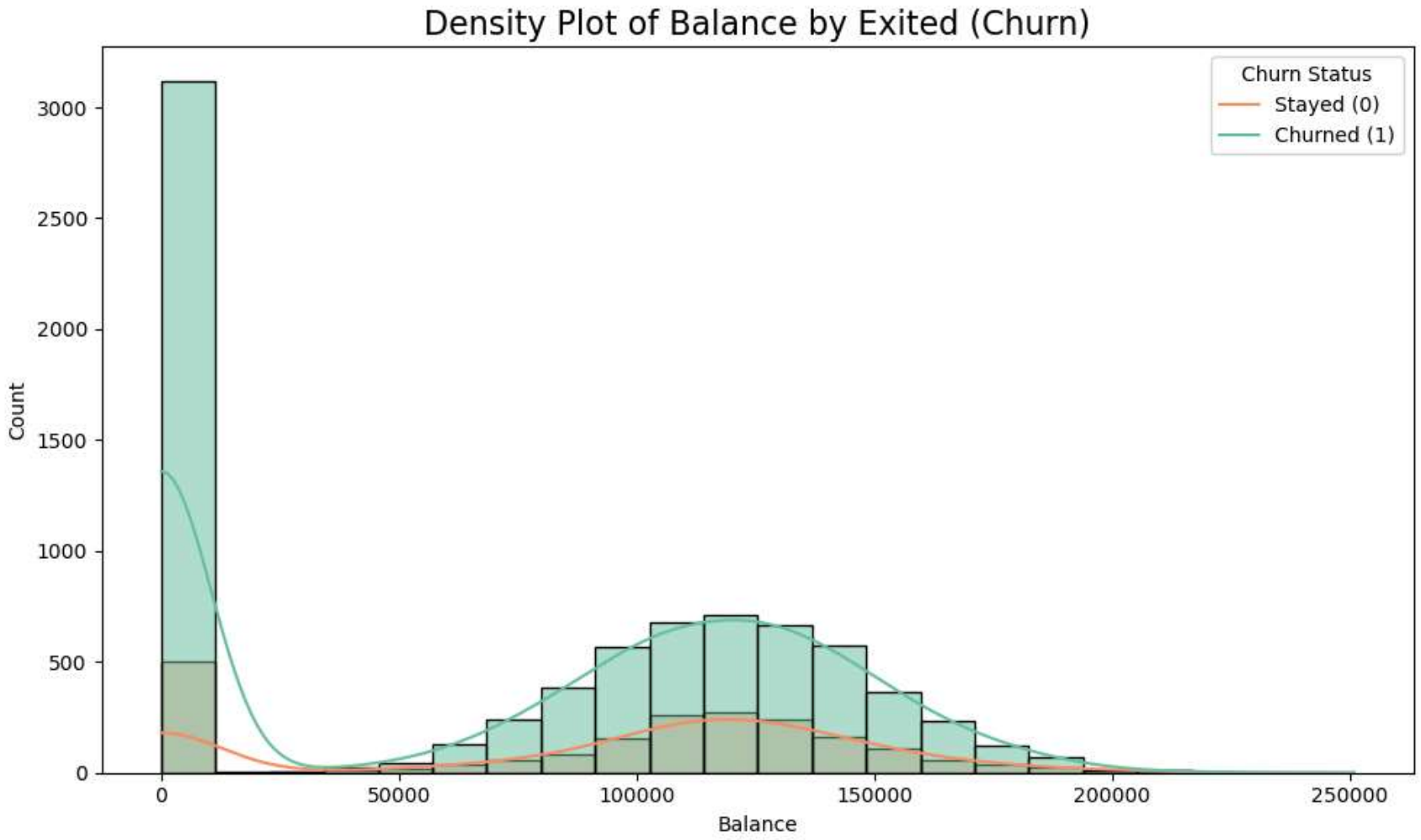
    - High churn proportion with **95 churned customers** compared to **318 who stayed.**
  2. **Tenure = 1 to 9**:
    - The churn proportion is relatively stable, with churned customers ranging from **177 to 232** across these tenure levels.
  3. **Tenure = 10**:
    - Similar to tenure = 0, higher churn proportion with **101 churned customers** compared to **389 who stayed.**
- **General Trend**:
  - Customers with tenure of **0 or 10 years** exhibit higher churn proportions compared to those with mid-range tenure levels.

Insights:

- **Retention efforts** should prioritize customers with very low (`Tenure = 0`) or very high tenure (`Tenure = 10`), as these groups are more likely to churn.
- Mid-tenure customers are relatively stable and show lower churn rates.

In [228…

```
# Create a histplot for Balance by Exited
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Balance', hue='Exited', kde=True, fill=True, palette='Set2')
plt.title('Density Plot of Balance by Exited (Churn)', fontsize=16)
plt.xlabel('Balance')
plt.ylabel('Count')
plt.legend(title='Churn Status', labels=['Stayed (0)', 'Churned (1)'])
plt.tight_layout()
plt.show()
```



Density Plot of Balance by Exited (Churn)

Summary:

This density plot illustrates the distribution of customer account balances, separated by churn status (`Exited`).

- **Key Observations**:
    1. **Zero Balance**:

- A significant number of customers have a balance of **0**, with a higher proportion of non-churned customers (`Stayed = 0`).

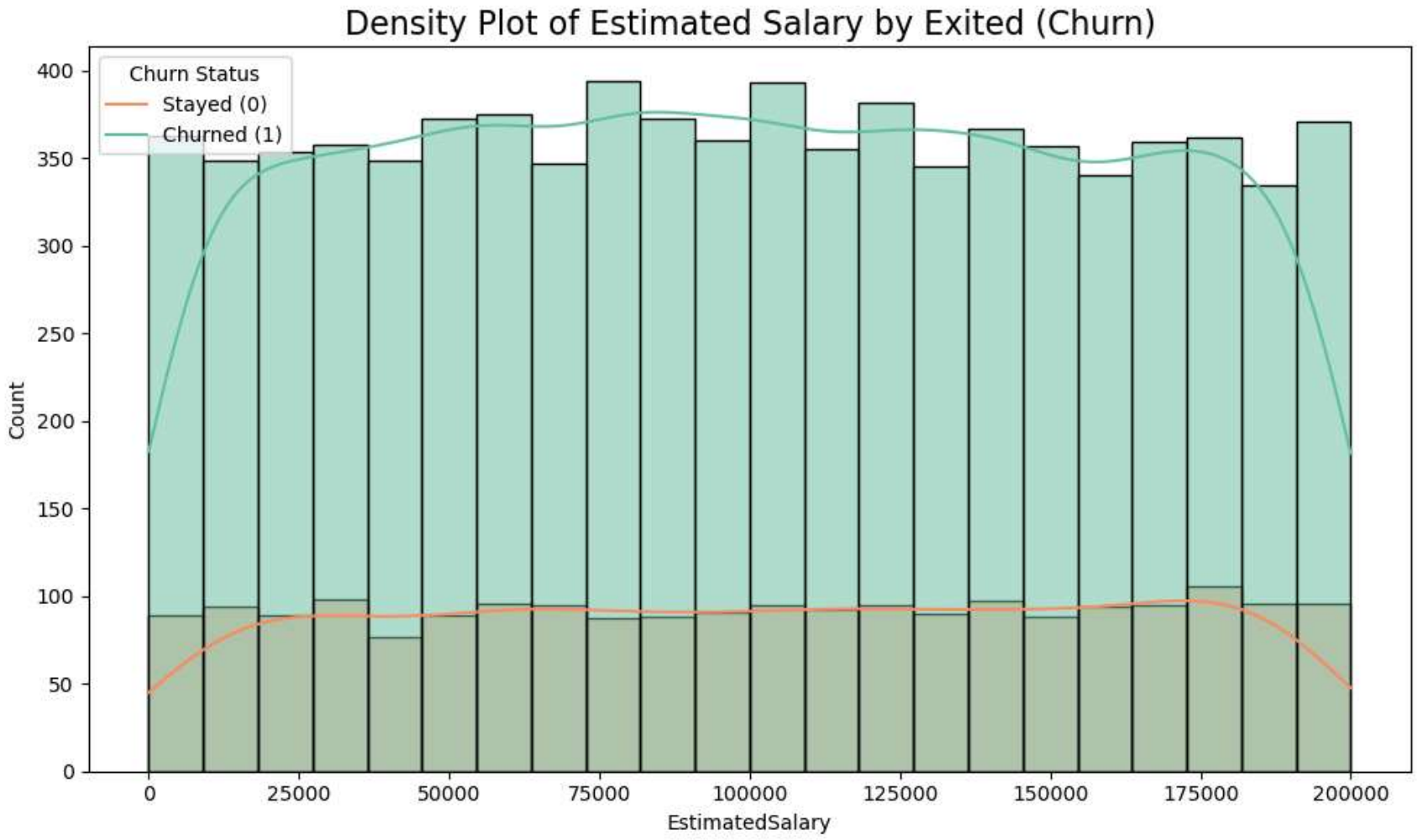
2. **Mid-Range Balances (50,000 - 150,000)**:
   - Both churned (`Exited = 1`) and non-churned customers are fairly distributed in this range.
3. **High Balances (>150,000)**:
   - Few customers exist in this range, and they are more likely to remain (`Stayed = 0`).

Insights:

- Customers with zero balance are less likely to churn, suggesting that these may represent inactive accounts.
- Churn is more evenly distributed among customers with mid-range balances.
- Retention efforts might focus on mid-balance customers, where churn rates are more evenly distributed, to identify and address potential issues.

In [229…

```
# Create a histplot for EstimatedSalary by Exited
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='EstimatedSalary', hue='Exited', kde=True, fill=True, palette='Set2')
plt.title('Density Plot of Estimated Salary by Exited (Churn)', fontsize=16)
plt.xlabel('EstimatedSalary')
plt.ylabel('Count')
plt.legend(title='Churn Status', labels=['Stayed (0)', 'Churned (1)'])
plt.tight_layout()
plt.show()
```



Density Plot of Estimated Salary by Exited (Churn)

Summary:

This density plot shows the distribution of estimated salaries among customers, separated by churn status (`Exited`).

- **Key Observations**:
    1. **Uniform Distribution**:

- Salaries are evenly distributed across the range, with no significant peaks or valleys for either churned (`Exited = 1`) or non-churned customers (`Exited = 0`).

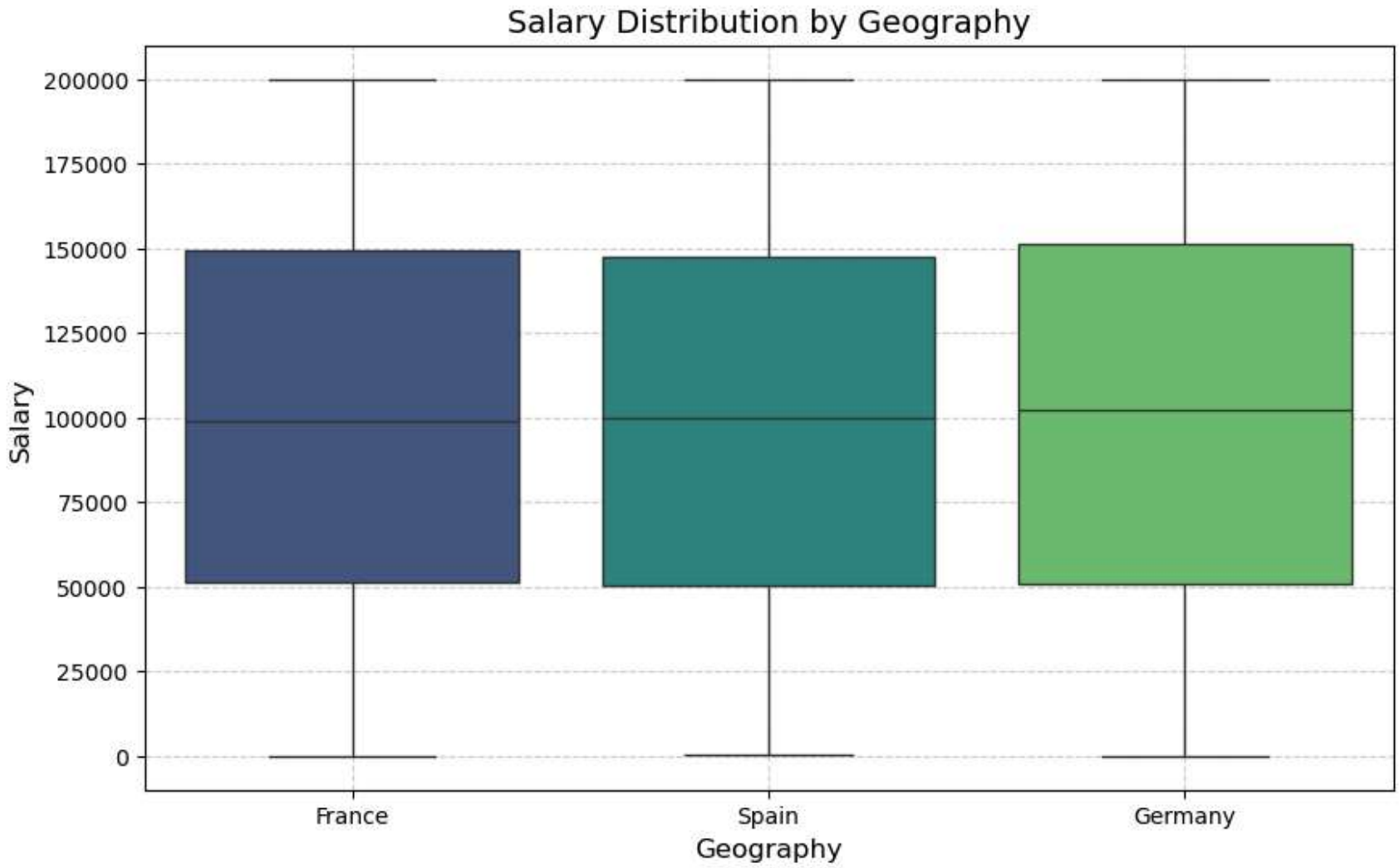
2. **Churn Proportion**:
   - The churn proportion remains consistent across all salary levels, suggesting no strong correlation between estimated salary and churn behavior.

Insights:

- Salary appears to have little impact on customer churn, as the distribution is uniform and the churn rate does not vary significantly across the salary range.
- Retention strategies should focus on other factors influencing churn, as salary alone is not a differentiating variable.

In [230…

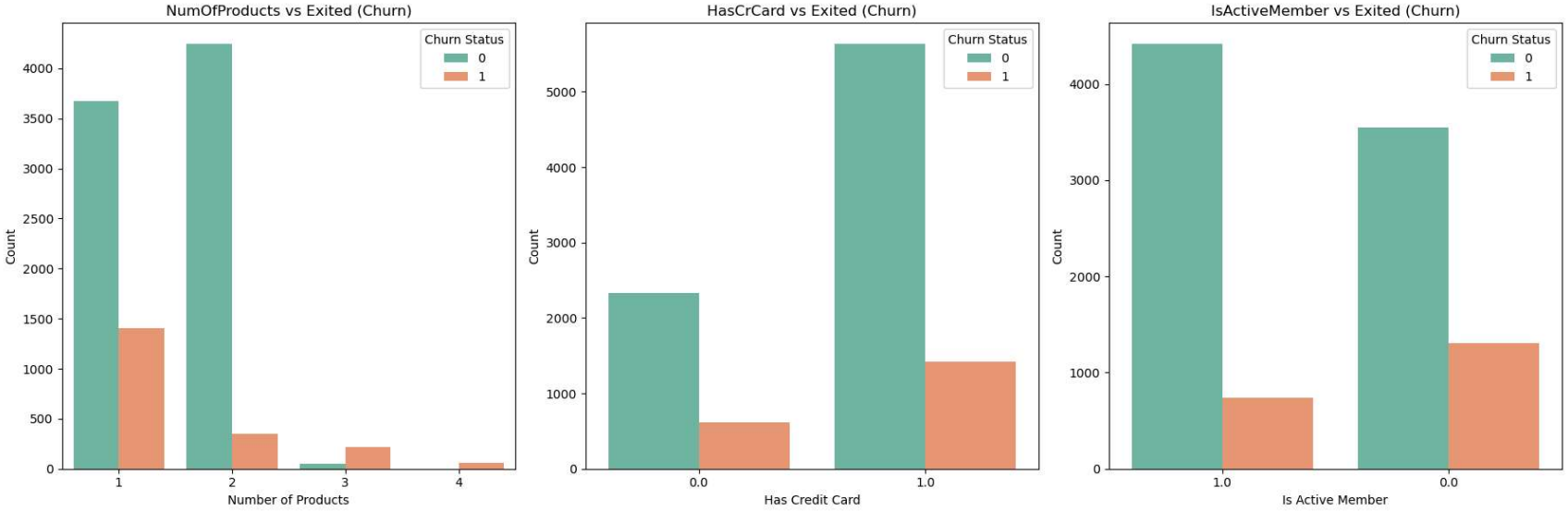
```
# Plotting salary distribution by geography
plt.figure(figsize=(10, 6))
sns.boxplot(x="Geography", y="EstimatedSalary", data=df, palette="viridis")
plt.title("Salary Distribution by Geography", fontsize=14)
plt.xlabel("Geography", fontsize=12)
plt.ylabel("Salary", fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```



In [231…

```
plt.figure(figsize=(18, 6))

# Countplot for NumOfProducts vs Exited
plt.subplot(1, 3, 1)
sns.countplot(data=df, x='NumOfProducts', hue='Exited', palette='Set2')
plt.title('NumOfProducts vs Exited (Churn)')
plt.xlabel('Number of Products')
plt.ylabel('Count')
plt.legend(title='Churn Status')
```

```
# Countplot for HasCrCard vs Exited
plt.subplot(1, 3, 2)
sns.countplot(data=df, x='HasCrCard', hue='Exited', palette='Set2')
plt.title('HasCrCard vs Exited (Churn)')
plt.xlabel('Has Credit Card')
plt.ylabel('Count')
plt.legend(title='Churn Status')

# Countplot for IsActiveMember vs Exited
plt.subplot(1, 3, 3)
sns.countplot(data=df, x='IsActiveMember', hue='Exited', palette='Set2')
plt.title('IsActiveMember vs Exited (Churn)')
plt.xlabel('Is Active Member')
plt.ylabel('Count')
plt.legend(title='Churn Status')

plt.tight_layout()
plt.show()
```



Categorical Feature Analysis by Exited (Churn)

1. NumOfProducts vs Exited (Churn)

- Customers with **1 or 2 products** dominate the dataset.
- Churn is significantly higher for customers with **2 products** compared to 1 product.
- Few customers have 3 or 4 products, but churn rates are proportionally high in these groups.

2. HasCrCard vs Exited (Churn)

- Most customers have a credit card (`HasCrCard = 1`), and churn is higher among them compared to those without a credit card.
- Customers without a credit card (`HasCrCard = 0`) exhibit lower churn.

3. IsActiveMember vs Exited (Churn)

- Active members (`IsActiveMember = 1`) churn less frequently than inactive members.
- Churn is significantly higher among inactive members (`IsActiveMember = 0`), indicating strong retention challenges in this group.

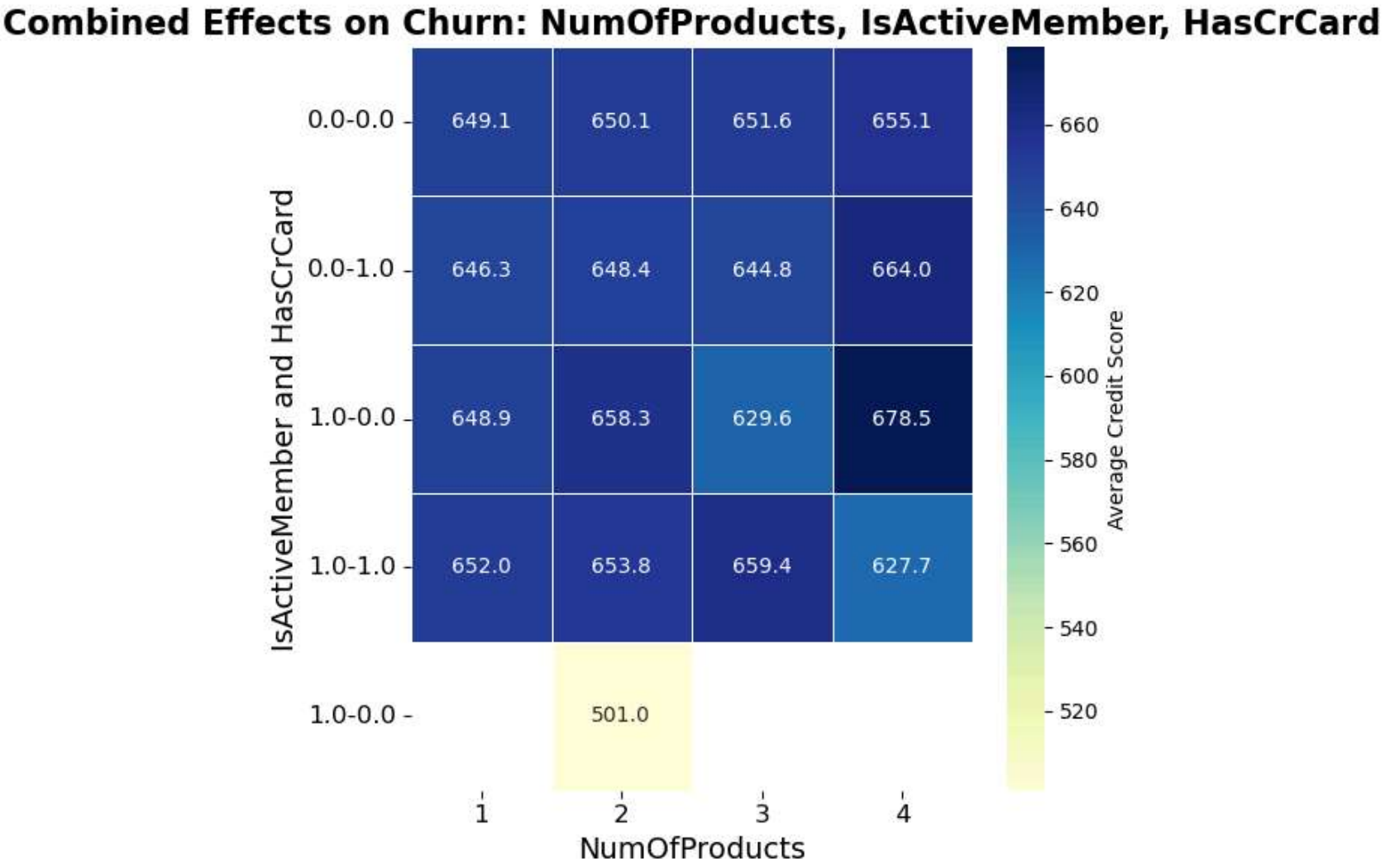
Insights:

1. Customers with **2 products** and inactive memberships probably are more likely to churn.
2. Efforts should be targeted at retaining customers with multiple products and converting inactive members into active ones.
3. The impact of having a credit card on churn requires further analysis, as churn is relatively higher among those with credit cards. Probably this clients look for discounts and offers in another banks.

In [232…

```
# Visualizing the combined effects of NumOfProducts, IsActiveMember, and HasCrCard on Churn
plt.figure(figsize=(6, 6))

sns.heatmap(
    df.groupby(['NumOfProducts', 'IsActiveMember', 'HasCrCard'])['CreditScore']
    .mean()
    .unstack(level=0),
    annot=True,
    fmt=".1f",
    cmap="YlGnBu",
    linewidths=0.5,
    cbar_kws={'label': 'Average Credit Score'}  # Adds a descriptive color bar label
)
```

```
# Titles and Labels
plt.title("Combined Effects on Churn: NumOfProducts, IsActiveMember, HasCrCard", fontsize=16, fontweight='bold')
plt.xlabel("NumOfProducts", fontsize=14)
plt.ylabel("IsActiveMember and HasCrCard", fontsize=14)

# Adjusting tick labels for better readability
plt.xticks(fontsize=12)
plt.yticks(fontsize=12, rotation=0)

plt.tight_layout()  # Adjusts layout to prevent clipping
plt.show()
```



Analysis of Combined Effects on Churn

Observations:

1. **Low Credit Scores for 2 Products**:
   - Customers with **2 products** and inactive membership (`IsActiveMember = 0`) have the **lowest average credit score (501)**, indicating potential financial stress in this group.

2. **High Credit Scores for Active Members**:
    - Active members (`IsActiveMember = 1`) generally have **higher credit scores**, particularly those with **3 or more products**.
3. **Churn Drivers**:
    - High product ownership (3–4 products) and active membership are correlated with higher credit scores, potentially reducing churn risk.
    - The lowest scores among inactive members with fewer products suggest they are more likely to churn.

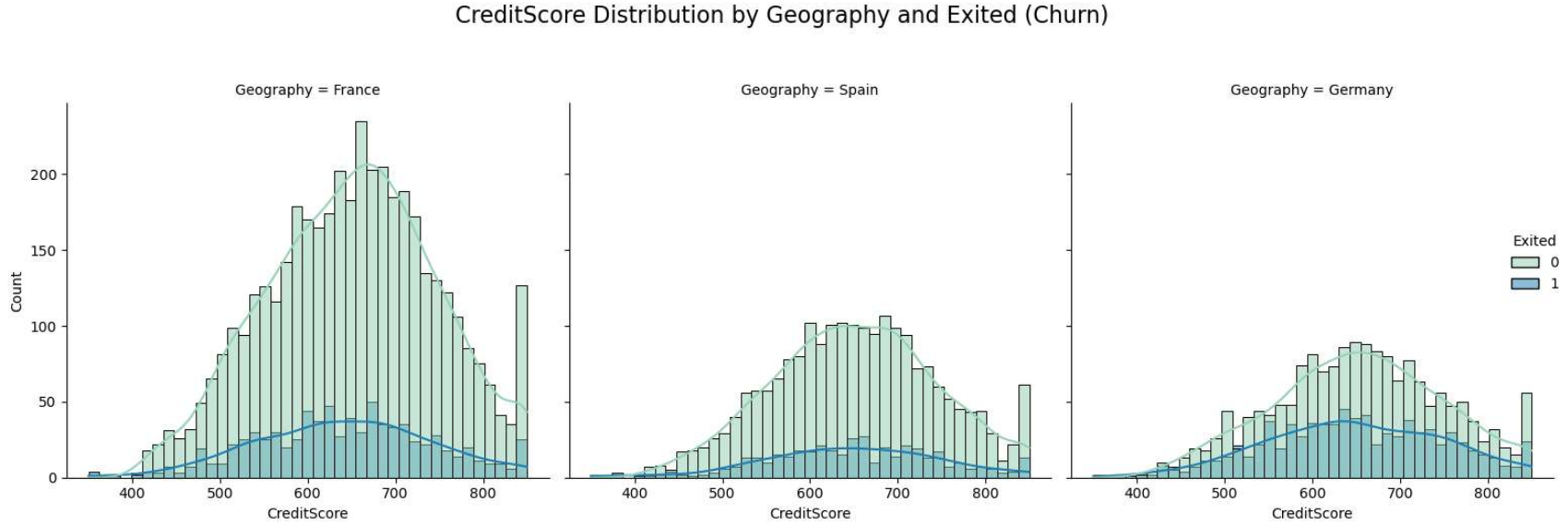
Recommendations:

- **Retention Focus**:
    - Prioritize inactive members with 1–2 products for churn prevention campaigns.
    - Provide personalized financial solutions to improve engagement in this segment.
- **Encourage Product Diversification**:
    - Promote additional product adoption among active members to strengthen retention and improve financial stability.

3.3 Multivariate analysis

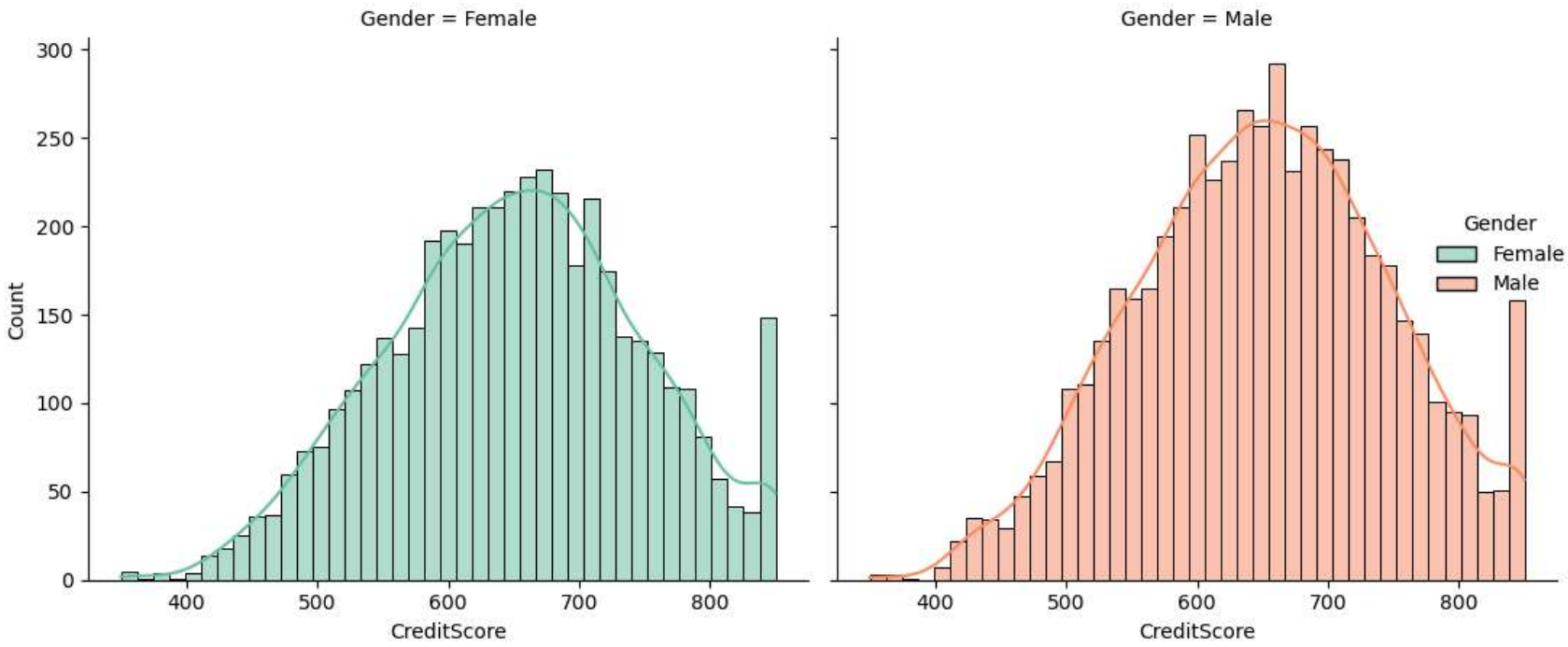
In [233…

```
# Plotting distribution of credit score by Geography and Exited
g = sns.displot(data=df, x="CreditScore", col="Geography", hue="Exited", kde=True, palette='YlGnBu')
g.fig.suptitle('CreditScore Distribution by Geography and Exited (Churn)', fontsize=16, y=1.05)
plt.tight_layout()
plt.show()
```



In [234…

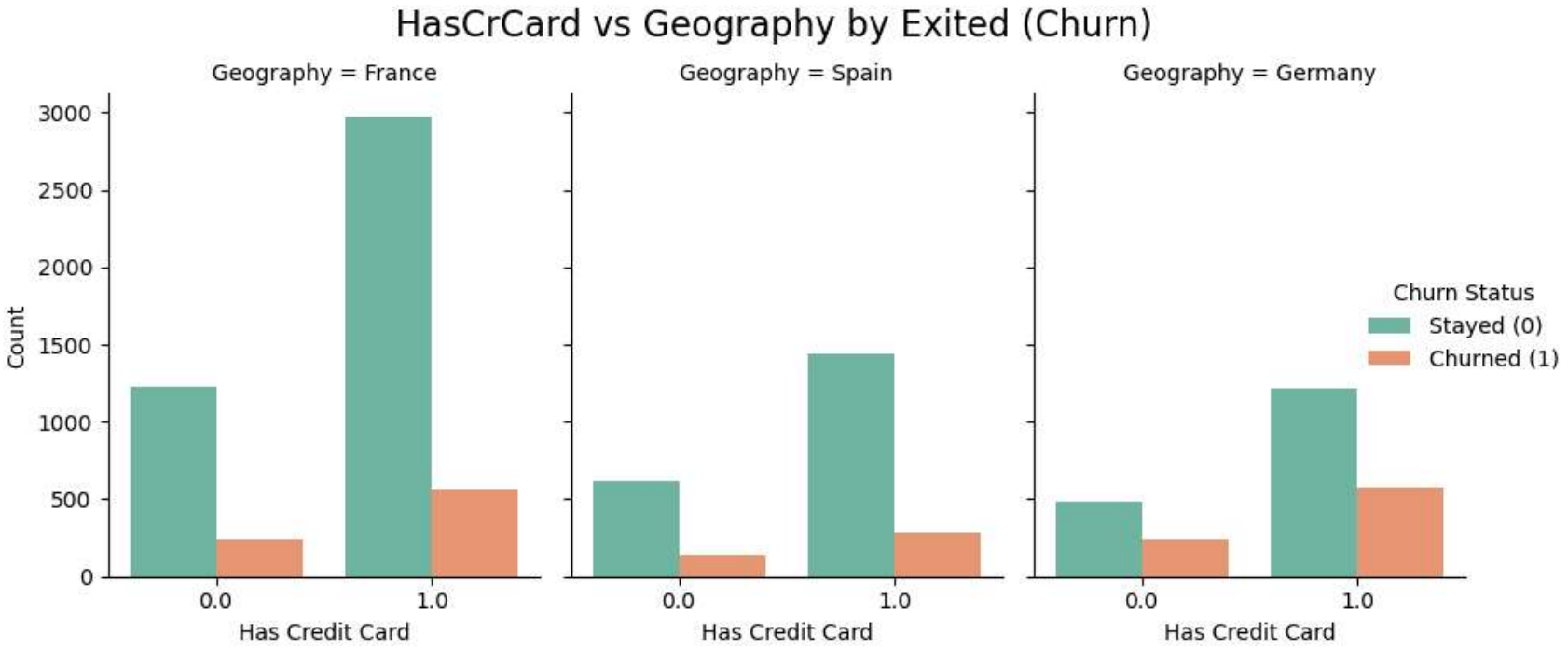
```
# Create the displot
g = sns.displot(data=df, x="CreditScore", col="Gender", hue="Gender", kde=True, palette='Set2')
g.fig.suptitle('CreditScore Distribution by Gender', fontsize=14, y=1.10)
plt.tight_layout()
plt.show()
```



In [235…

```
# Create a catplot to visualize HasCrCard vs Geography by Exited
g = sns.catplot(
    data=df,
    x='HasCrCard',
    hue='Exited',
    col='Geography',
    kind='count',
    palette='Set2',
    height=4,
    aspect=0.8
)

g.fig.suptitle('HasCrCard vs Geography by Exited (Churn)', fontsize=16, y=1.05)
g.set_axis_labels('Has Credit Card', 'Count')
g._legend.set_title('Churn Status')
```

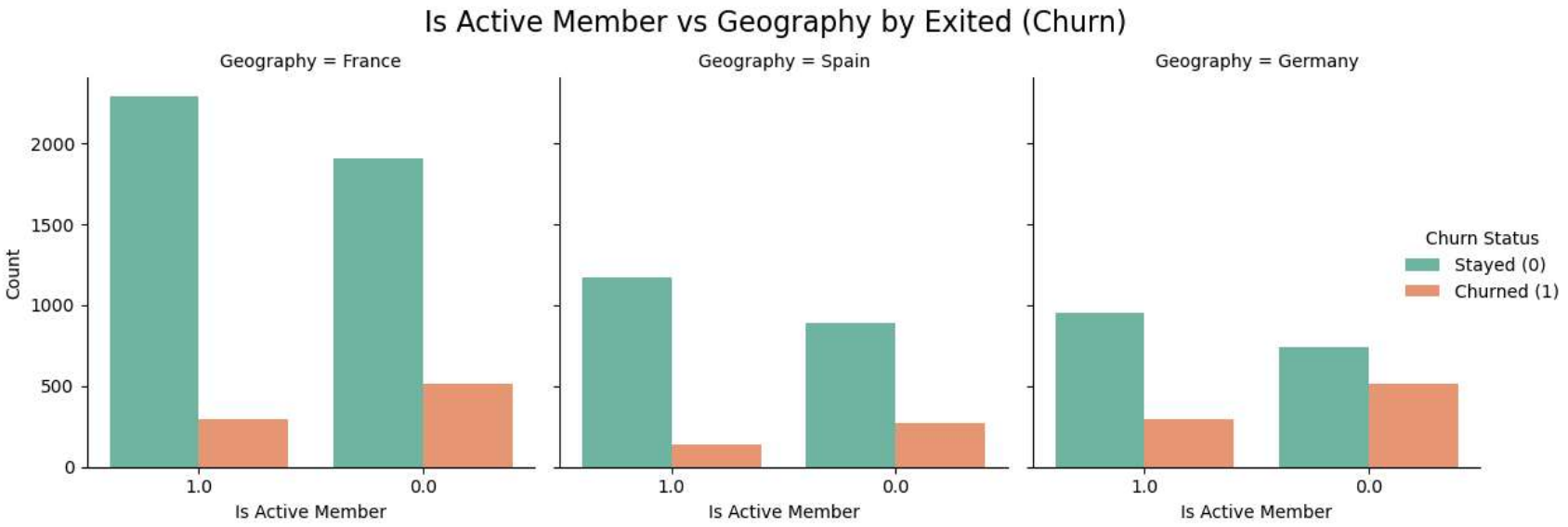
```
new_labels = ['Stayed (0)', 'Churned (1)']
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)
plt.show()
```



In [236…

```
# Create a catplot to visualize IsActiveMember vs Geography by Exited
g = sns.catplot(
    data=df,
    x='IsActiveMember',
    hue='Exited',
    col='Geography',
    kind='count',
    palette='Set2',
    height=4,
    aspect=1
)

g.fig.suptitle('Is Active Member vs Geography by Exited (Churn)', fontsize=16, y=1.05)
g.set_axis_labels('Is Active Member', 'Count')
g._legend.set_title('Churn Status')
new_labels = ['Stayed (0)', 'Churned (1)']
```

```
for t, l in zip(g._legend.texts, new_labels): t.set_text(l)
plt.show()
```



4. Data Preprocessing

Preparing accesible for modeling process

In [237…

```
# Initializing LabelEncoder
label_encoder = LabelEncoder()

# Encoding 'Geography'
df['Geography'] = label_encoder.fit_transform(df['Geography'])

# Encoding 'Gender'
df['Gender'] = label_encoder.fit_transform(df['Gender'])

print("Transformed Dataset:")
print(df.head())
```

```
Transformed Dataset:
   CreditScore  Geography  Gender   Age  Tenure    Balance  NumOfProducts  \
0          619          0       0  42.0       2       0.00              1
1          608          2       0  41.0       1   83807.86              1
2          502          0       0  42.0       8  159660.80              3
3          699          0       0  39.0       1       0.00              2
5          645          2       1  44.0       8  113755.78              2

   HasCrCard IsActiveMember  EstimatedSalary  Exited
0        1.0            1.0        101348.88       1
1        0.0            1.0        112542.58       0
2        1.0            0.0        113931.57       1
3        0.0            0.0         93826.63       0
5        1.0            0.0        149756.71       1
```

In [238…

```python
# Save the  preprocessed DataFrame to a CSV file for further modeling process
df.to_csv('data/ModelingDataSet.csv', index=False)

print("DataFrame saved to 'data/ModelingDataSet.csv'")
```

```
DataFrame saved to 'data/ModelingDataSet.csv'
```

Now we are ready to start the modeling process.

For modeling we will use `ModelingDataSet.csv` in separate file.