ML-Driven Bank Churn Prediction:

In [1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread("Bank churn.jpg")
plt.figure(figsize=(13,6))
plt.imshow(img)
plt.axis('off')
plt.show()
```

Unlocking Customer Retention Insights, Analyzing Demographic and Financial Factors

customer churn is a significant issue for banks, impacting revenue and customer lifetime value. The goal is to develop a machine learning model that predicts whether a customer will churn (leave the bank) based on their demographic, financial, and account activity data.

Objective:

1) Build a predictive model to classify customers as likely to churn (Exited = 1) or stay (Exited = 0).

2) Identify key factors influencing customer attrition.

3) Provide actionable insights for customer retention strategies.

The dataset consists of 10,000 records with 13 features:

1) CustomerId: Unique identifier for each customer.

2) Lastname: Customer's last name (not useful for prediction).

3) CreditScore: Customer's credit rating.

4) Geography: Country of the customer (France, Spain, Germany).

5) Gender: Male or Female.

6) Age: Customer's age.

7) Tenure: Number of years the customer has been with the bank.

8) Balance: Account balance.

9) NumOfProducts: Number of products the customer has with the bank.

10) HasCrCard: Whether the customer has a credit card (1 = Yes, 0 = No).

11) IsActiveMember: Whether the customer is an active member (1 = Yes, 0 = No).

12) EstimatedSalary: Customer's estimated salary.

13) Exited: Target variable (1 = Churned, 0 = Stayed).

1. IMPORT LIBRARIES

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

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

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import (confusion_matrix , roc_curve, ConfusionMatrixDi
```

2. Loading Datset And Basic Check of Data

In [3]:
```
data = ("C:/Users/Welcome/M.L__PROJECT/Project REAL TIME 4/Bank+Customer+Chu
data
```

Out[3]:
```
'C:/Users/Welcome/M.L__PROJECT/Project REAL TIME 4/Bank+Customer+Churn (1)/
Bank_Churn.csv'
```

In [4]:
```
df = pd.read_csv(data)
df
```

Out[4]:

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

10000 rows × 13 columns

In [5]:
```
df.head()
```

Out[5]:

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Ba
0	15634602	Hargrave	619	France	Female	42	2	
1	15647311	Hill	608	Spain	Female	41	1	838
2	15619304	Onio	502	France	Female	42	8	1596
3	15701354	Boni	699	France	Female	39	1	
4	15737888	Mitchell	850	Spain	Female	43	2	1255

In [6]:
```
df.tail()
```

Out[6]:

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	
9995	15606229	Obijiaku	771	France	Male	39	5	
9996	15569892	Johnstone	516	France	Male	35	10	
9997	15584532	Liu	709	France	Female	36	7	
9998	15682355	Sabbatini	772	Germany	Male	42	3	
9999	15628319	Walker	792	France	Female	28	4	1

In [7]:
```
df.drop(["Surname","CustomerId"],axis=1,inplace=True)
df.columns
```

Out[7]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

3. Data Preprocessing

In [8]:
```python
# Checking Dataset shape
df.shape
```

Out[8]:
```
(10000, 11)
```

In [9]:
```python
# Checking dataset size
df.size
```

Out[9]:
```
110000
```

In [10]:
```python
df.columns
```

Out[10]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

In [11]:
```python
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 [12]:
```python
#Checking the information of the dataset
df.dtypes
```

Out[12]:
```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [13]:
```
# Exited Is the Target column
# Checking the value counts for number of unique values
df['Exited'].value_counts()
```

Out[13]:
```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

In [14]:
```
# Checking the percentage of number of unique values are present
df['Exited'].value_counts(normalize=True)*100
```

Out[14]:
```
Exited
0    79.63
1    20.37
Name: proportion, dtype: float64
```

Data is imbalance

Checking Missing and Duplicate Data

In [15]:
```
df.isnull().sum()
```

Out[15]:
```
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 [16]:
```
df.duplicated().sum()
```

Out[16]:
```
0
```

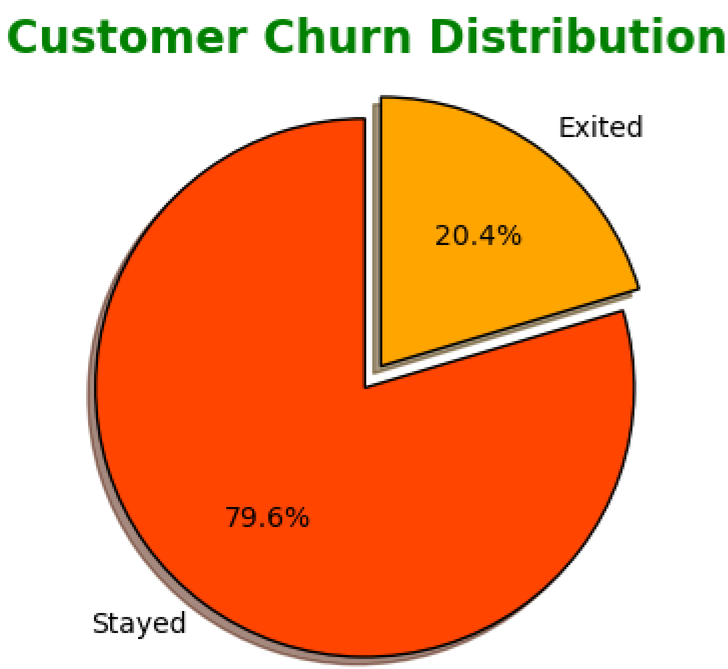
Dropping CustomerID and Surname coulmns

In [17]:
```
# Checking the output after dropping columns
df.head()
```

Out[17]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts
0	619	France	Female	42	2	0.00	1
1	608	Spain	Female	41	1	83807.86	1
2	502	France	Female	42	8	159660.80	3
3	699	France	Female	39	1	0.00	2
4	850	Spain	Female	43	2	125510.82	1

In [18]:
```
# Visual Representation for data impbalance check
plt.figure(figsize=(6,4))
df['Exited'].value_counts().plot(kind='pie', labels=['Stayed', 'Exited'], au
    shadow=True, startangle=90, explode=(0.1, 0), wedgeprops={'edgecolor': '
)
plt.title("Customer Churn Distribution", fontsize=16, fontweight="bold", col
plt.ylabel('')
plt.axis('equal')
plt.show()
```



4. Sorting Categorical columns and Numerical columns in list

In [19]:
```
cat_col=[]
num_col=[]
```

```
for i in df.columns:
    if df[i].nunique() < 12:
        cat_col.append(i)
    else:
        num_col.append(i)
print("categorical columns are:\n",cat_col)
print()
print("numerical columns are:\n",num_col)
```

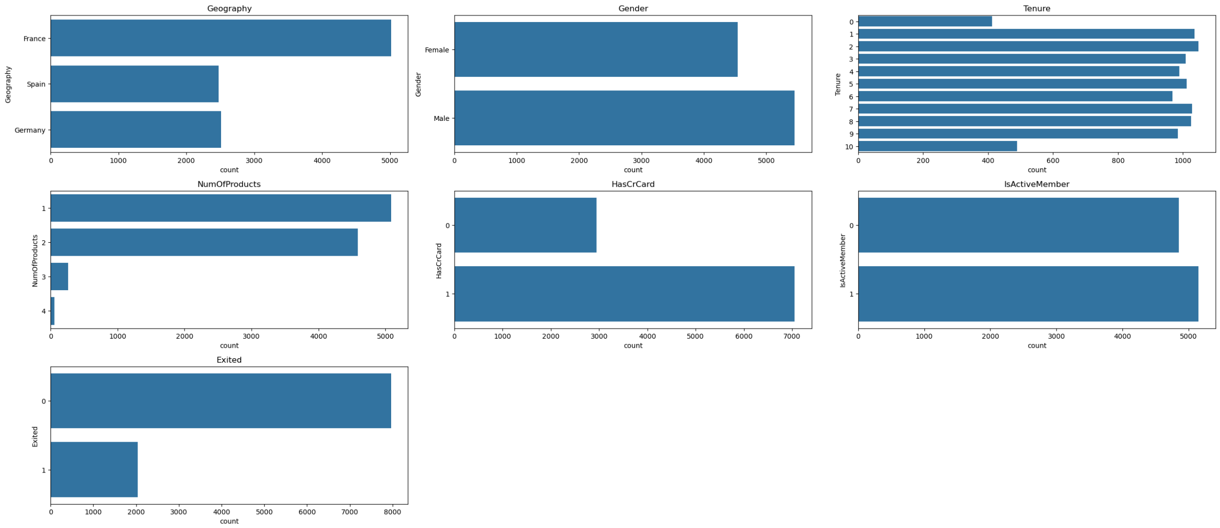
```
categorical columns are:
 ['Geography', 'Gender', 'Tenure', 'NumOfProducts', 'HasCrCard', 'IsActiveMe
mber', 'Exited']

numerical columns are:
 ['CreditScore', 'Age', 'Balance', 'EstimatedSalary']
```

In [20]:
```
# Number of Unique Values In Every Column

for i in df.columns:
    if df[i].nunique() > 11:
        print(i,'   ',df[i].nunique(),'   ','numerical')
    else:
        print(i,'   ',df[i].nunique(),'   ','categorical')
```

```
CreditScore    460    numerical
Geography    3    categorical
Gender    2    categorical
Age    70    numerical
Tenure    11    categorical
Balance    6382    numerical
NumOfProducts    4    categorical
HasCrCard    2    categorical
IsActiveMember    2    categorical
EstimatedSalary    9999    numerical
Exited    2    categorical
```

5. EDA For Categorical

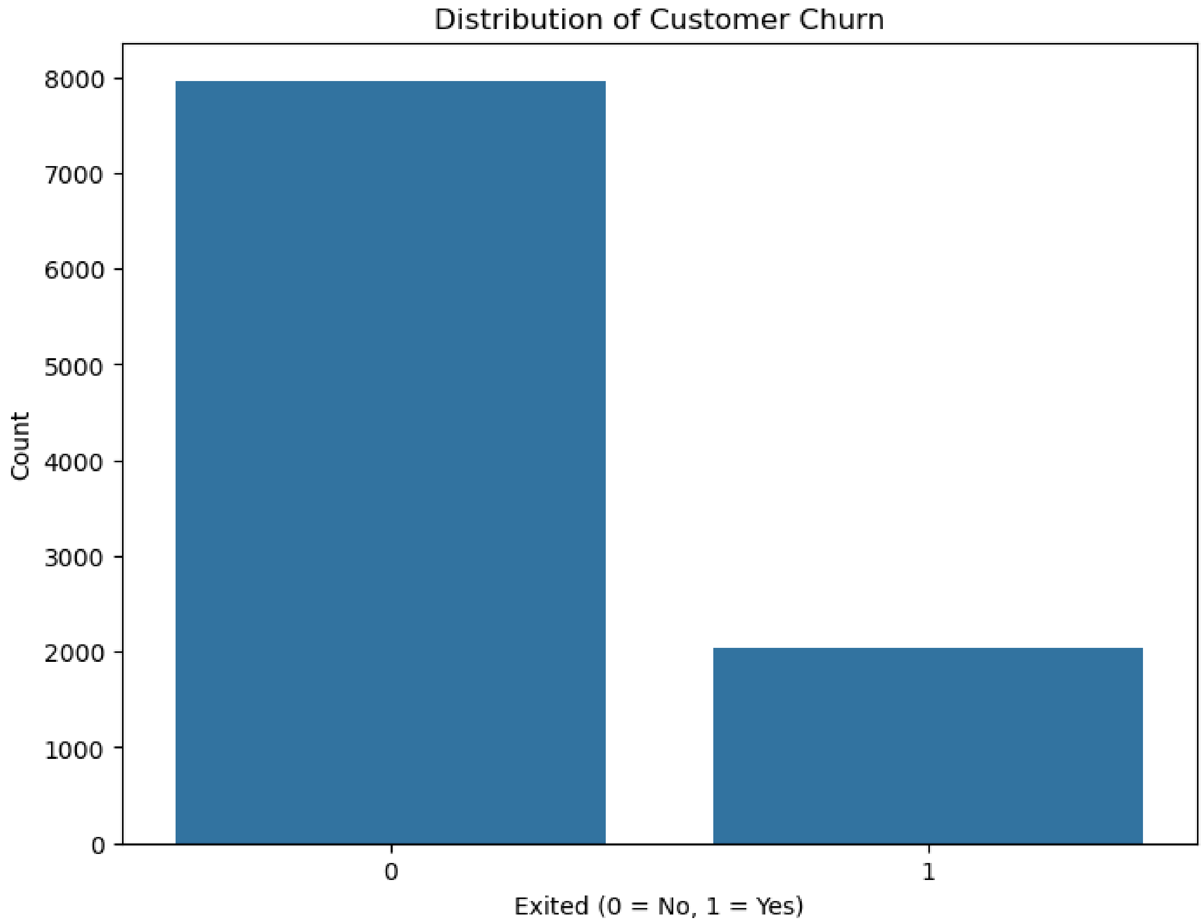
In [21]:
```
# # Example countplot for a single categorical variable
count=1
fig=plt.figure(figsize=(25,50))
for i in cat_col:

    plt.subplot(14,3,count)
    sns.countplot(y=i, data=df)
    plt.title(i)
    plt.xlabel('count')
    plt.ylabel(i)
    fig.tight_layout()
    count+=1
plt.show()
```



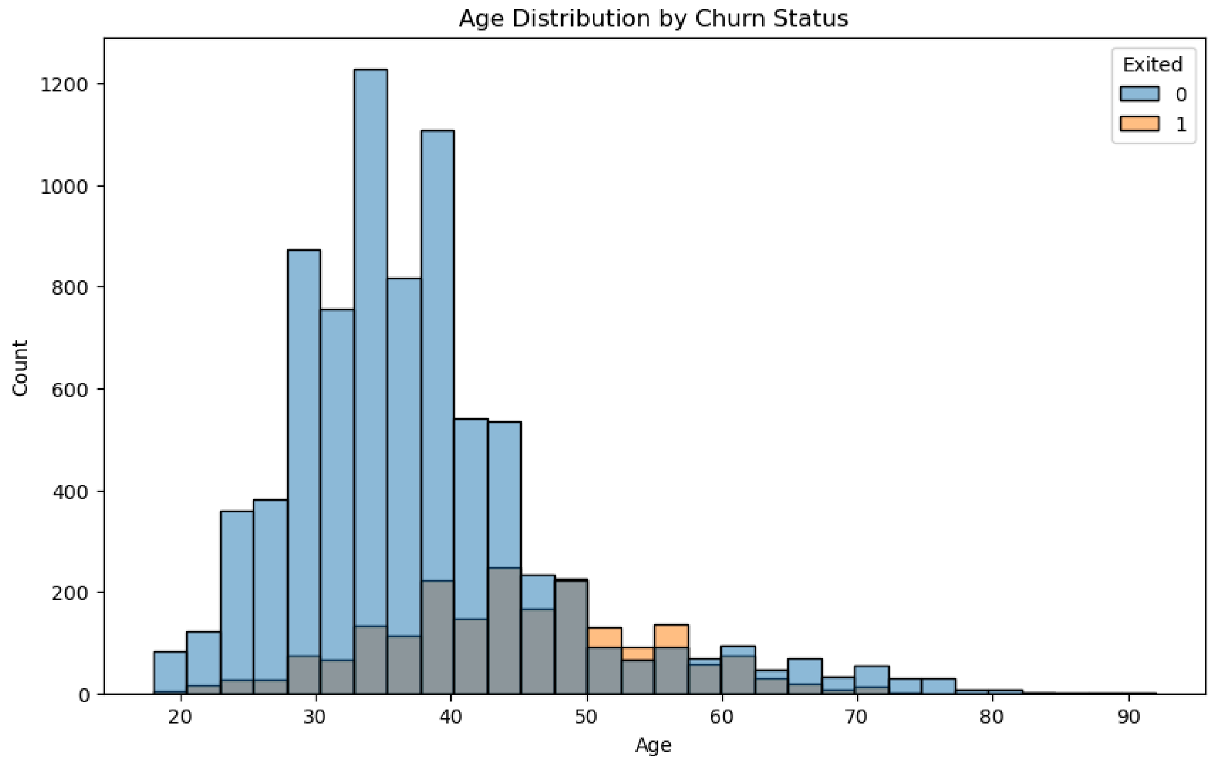
Step 5 A]: Exploratory Data Analysis (EDA) with Visualizations

In [22]:

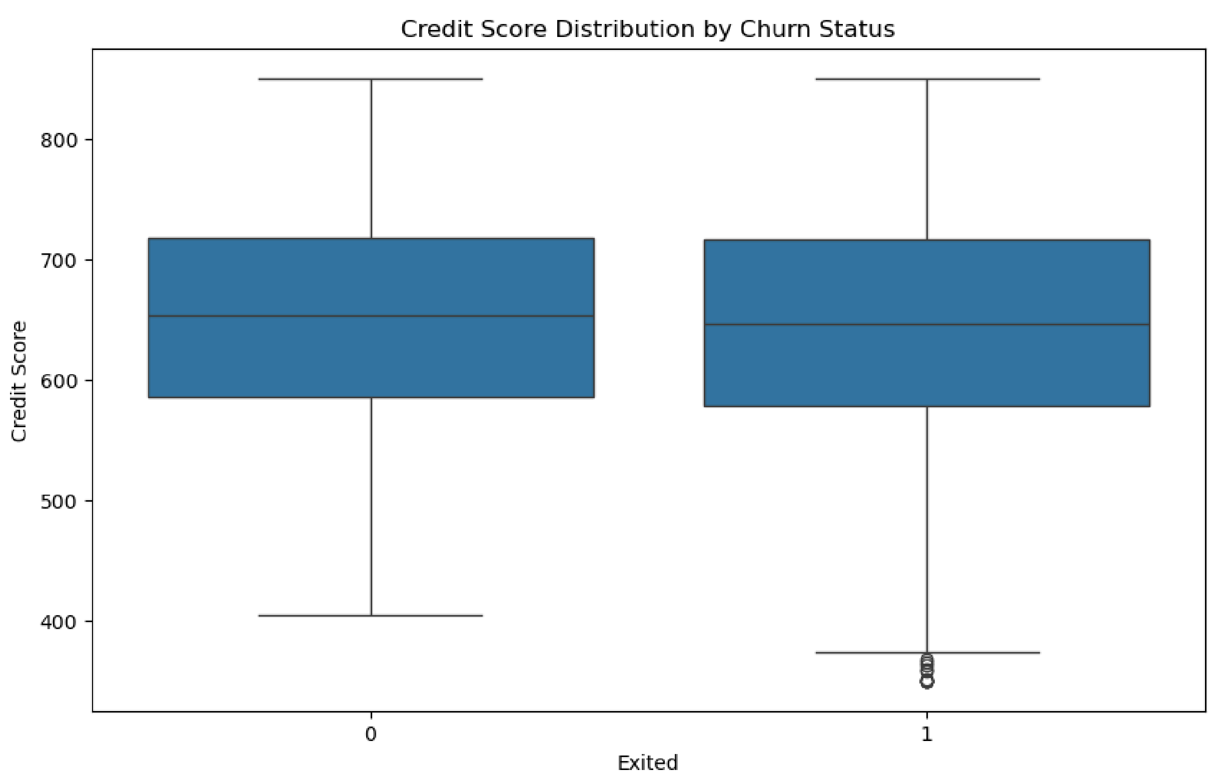
```
# 5.1: Distribution of Target Variable (Exited)
plt.figure(figsize=(8, 6))
sns.countplot(x='Exited', data=df)
plt.title('Distribution of Customer Churn')
plt.xlabel('Exited (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()
```



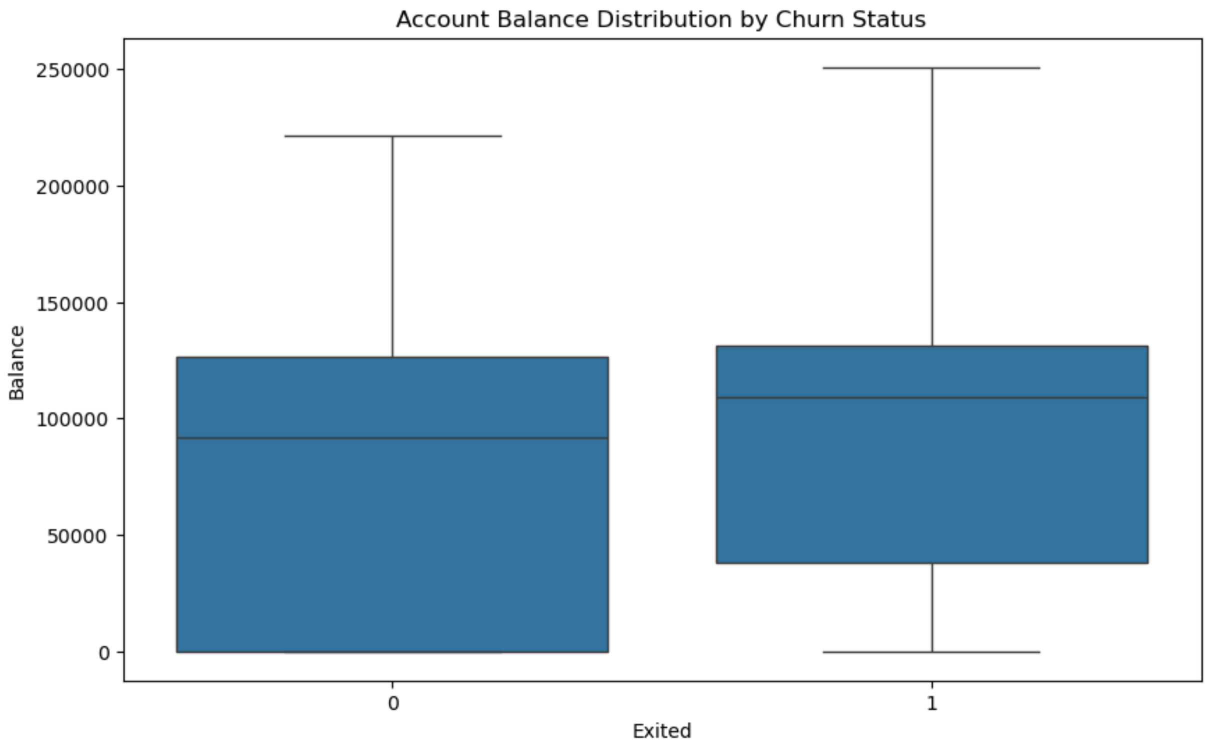
In [23]:
```
# 5.2: Age Distribution by Churn Status
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Age', hue='Exited', bins=30)
plt.title('Age Distribution by Churn Status')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```



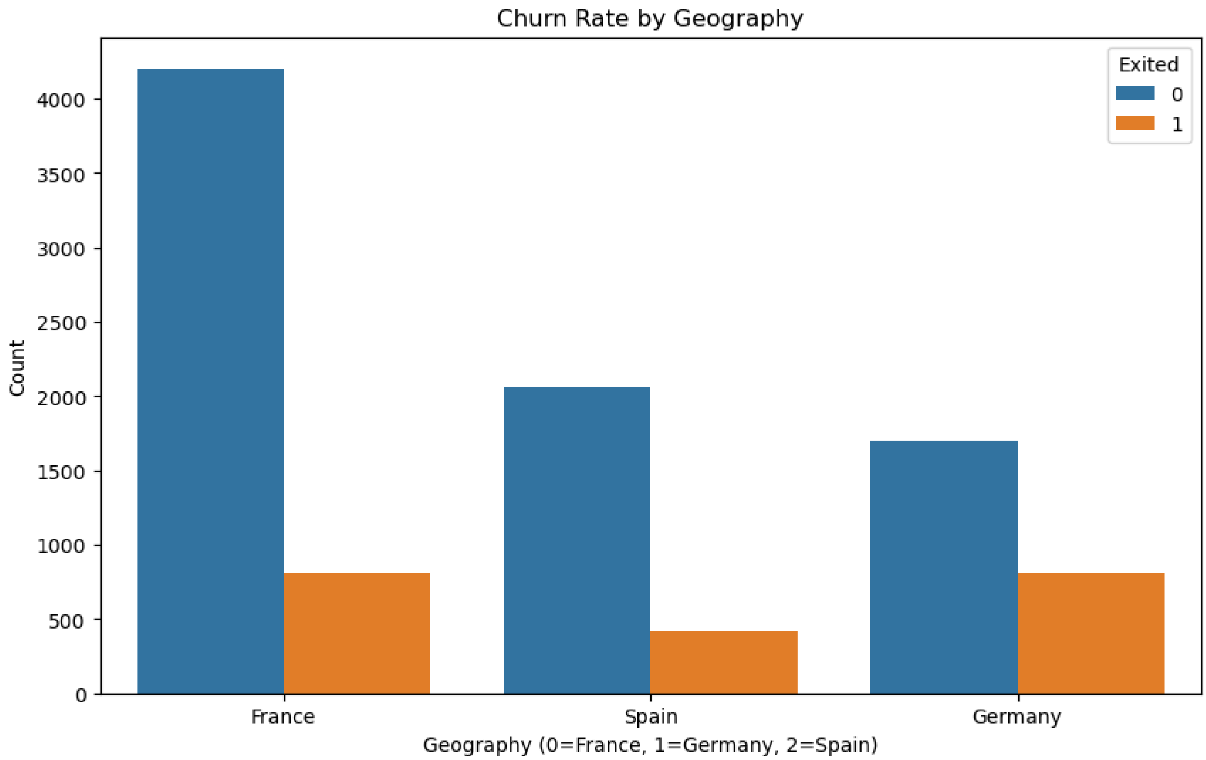
In [24]:
```
# 5.3: Credit Score Distribution
plt.figure(figsize=(10, 6))
sns.boxplot(x='Exited', y='CreditScore', data=df)
plt.title('Credit Score Distribution by Churn Status')
plt.xlabel('Exited')
plt.ylabel('Credit Score')
plt.show()
```



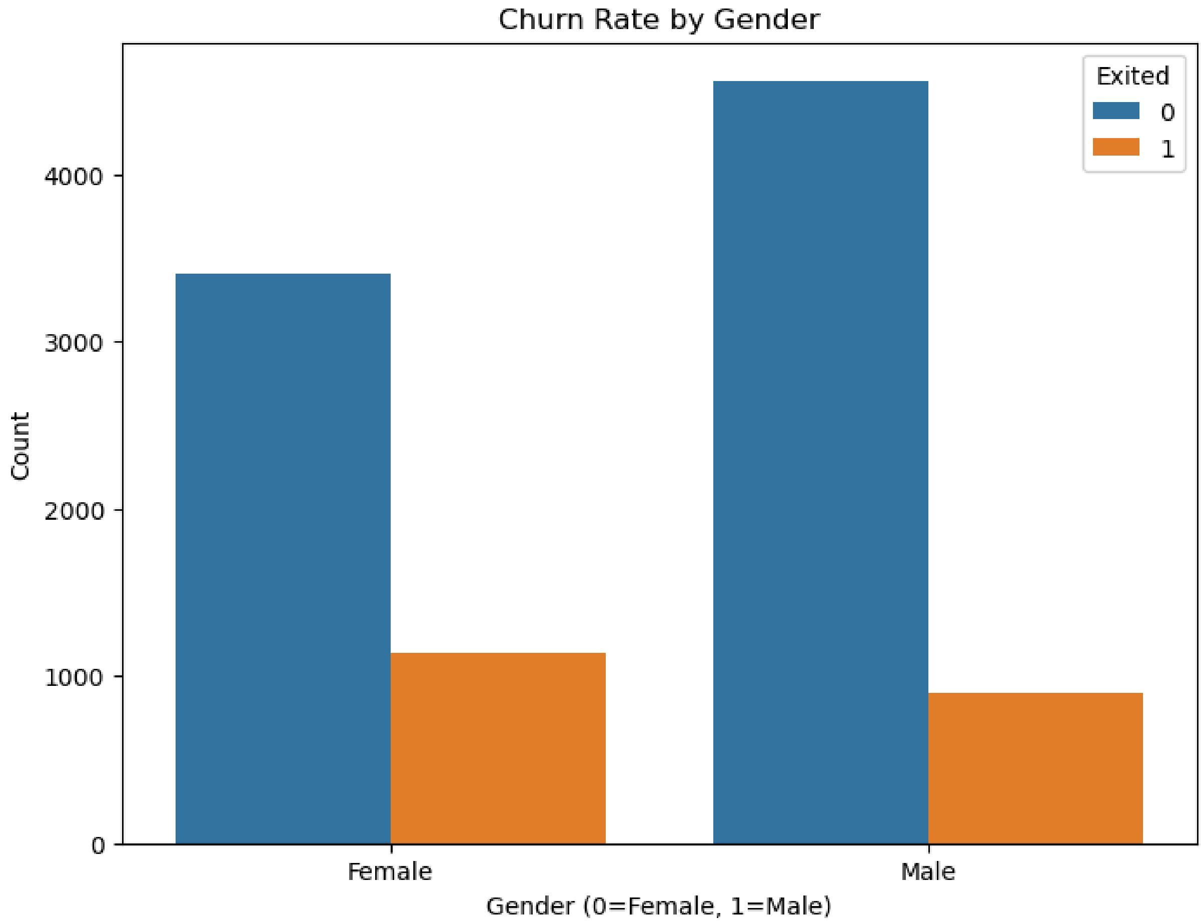
In [25]:
```
# 5.4: Balance vs Exited
plt.figure(figsize=(10, 6))
sns.boxplot(x='Exited', y='Balance', data=df)
plt.title('Account Balance Distribution by Churn Status')
plt.xlabel('Exited')
plt.ylabel('Balance')
plt.show()
```



In [26]:
```
# 5.5: Churn Rate by Geography
plt.figure(figsize=(10, 6))
sns.countplot(x='Geography', hue='Exited', data=df)
plt.title('Churn Rate by Geography')
plt.xlabel('Geography (0=France, 1=Germany, 2=Spain)')
plt.ylabel('Count')
plt.show()
```

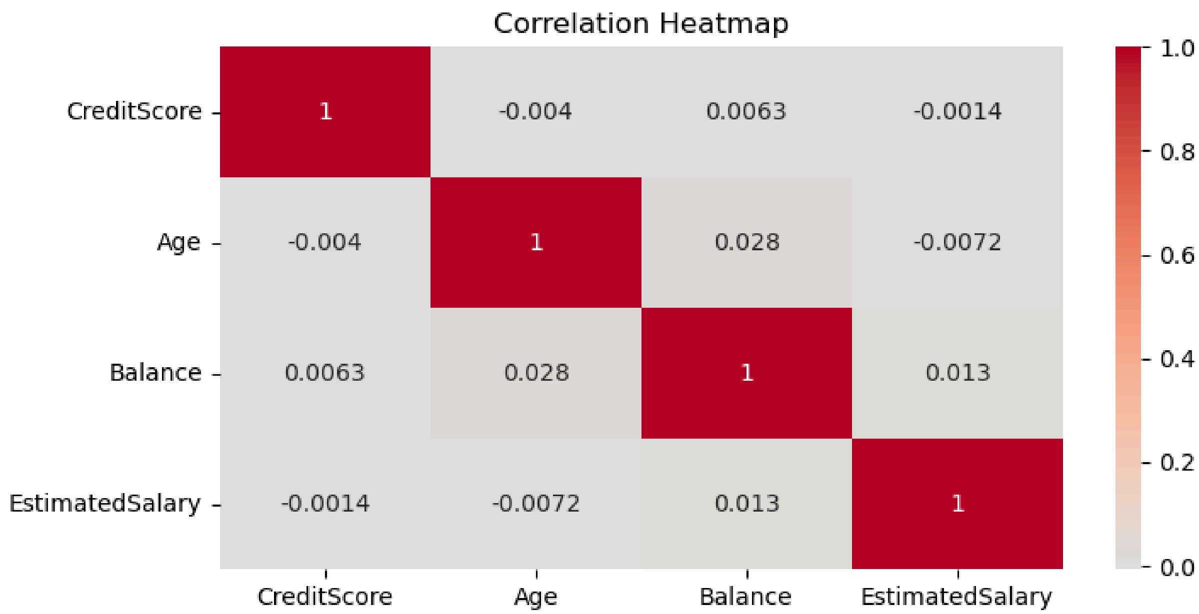


In [27]:
```
# 5.6: Churn Rate by Gender
plt.figure(figsize=(8, 6))
sns.countplot(x='Gender', hue='Exited', data=df)
plt.title('Churn Rate by Gender')
plt.xlabel('Gender (0=Female, 1=Male)')
plt.ylabel('Count')
plt.show()
```

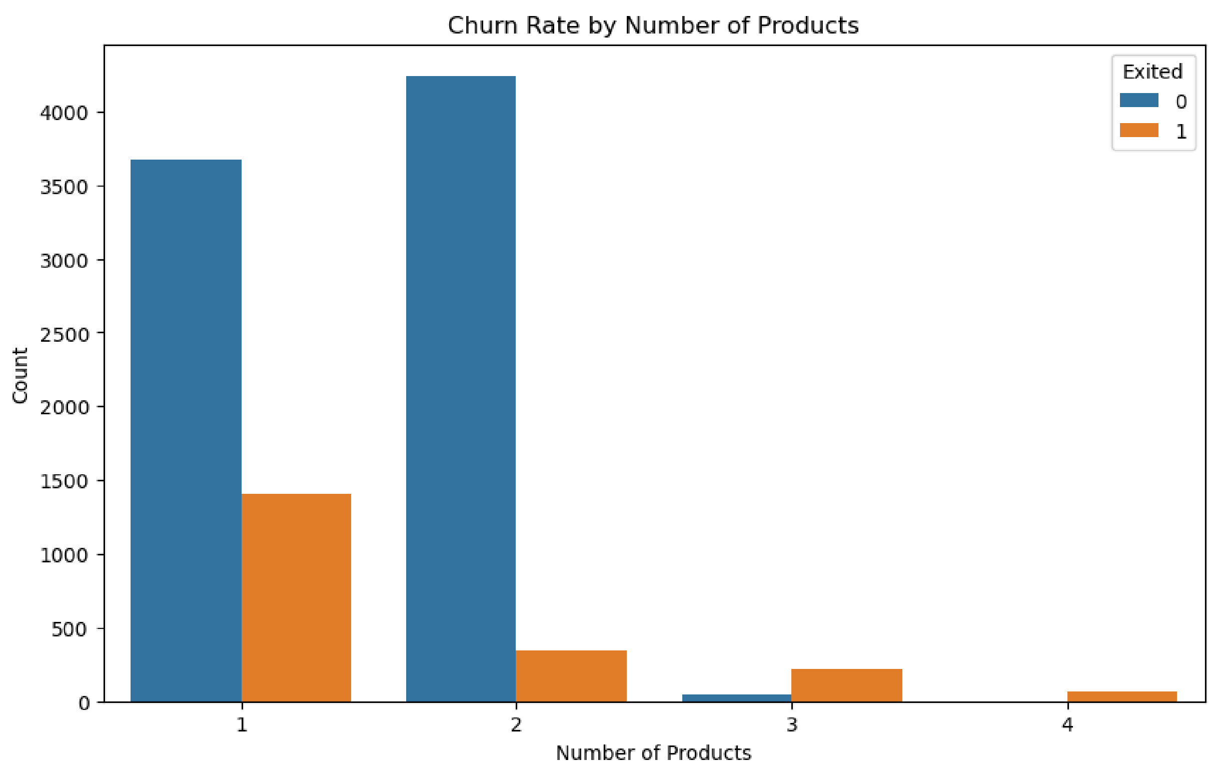


6 Heatmap

In [28]:
```
#5.5: Correlation Heatmap
plt.figure(figsize=(8, 4))
sns.heatmap(df[num_col].corr(), annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Heatmap')
plt.show()
```



In [29]:
```
# Churn rate by number of products
plt.figure(figsize=(10, 6))
sns.countplot(x='NumOfProducts', hue='Exited', data=df)
plt.title('Churn Rate by Number of Products')
plt.xlabel('Number of Products')
plt.ylabel('Count')
plt.show()
```



Five point summary Numerical Analysis

In [30]:
```
df[num_col].describe()
```

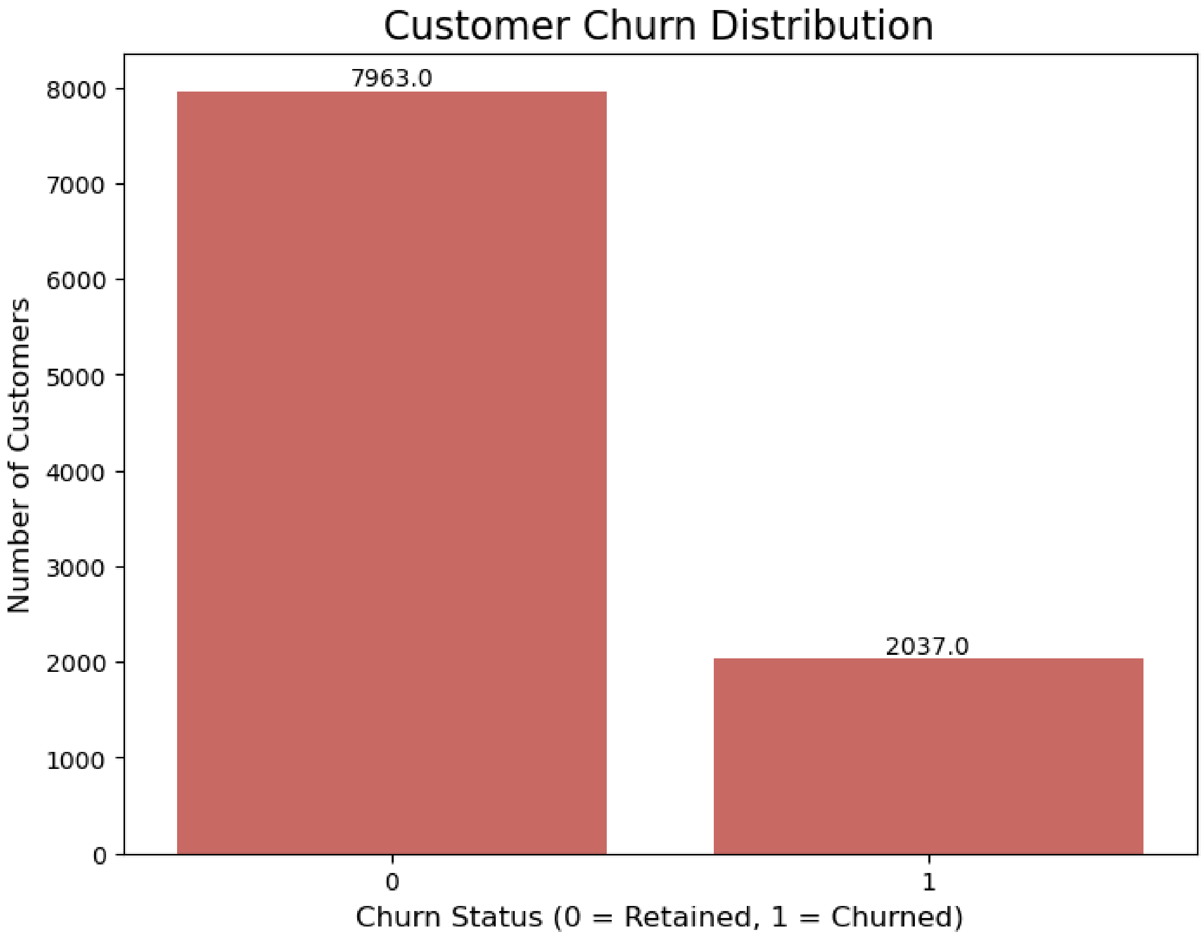
Out[30]:

	CreditScore	Age	Balance	EstimatedSalary
count	10000.000000	10000.000000	10000.000000	10000.000000
mean	650.528800	38.921800	76485.889288	100090.239881
std	96.653299	10.487806	62397.405202	57510.492818
min	350.000000	18.000000	0.000000	11.580000
25%	584.000000	32.000000	0.000000	51002.110000
50%	652.000000	37.000000	97198.540000	100193.915000
75%	718.000000	44.000000	127644.240000	149388.247500
max	850.000000	92.000000	250898.090000	199992.480000

7. EDA for Numerical

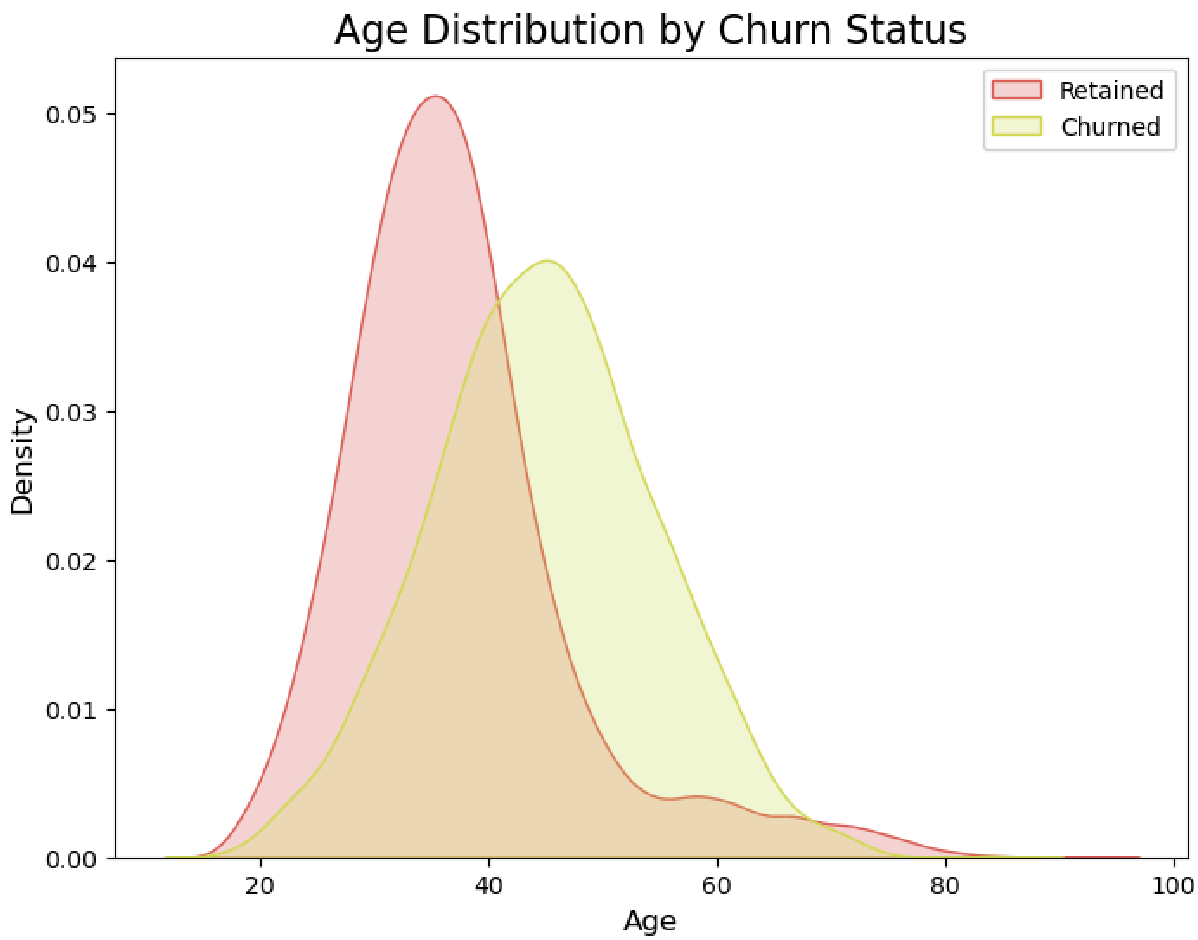
In [31]:

```
# Import required libraries
from sklearn.preprocessing import LabelEncoder
import plotly.express as px
import plotly.graph_objects as go

sns.set_palette("hls")

# 1. Comprehensive Churn Distribution
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='Exited', data=df)
plt.title('Customer Churn Distribution', fontsize=16)
plt.xlabel('Churn Status (0 = Retained, 1 = Churned)', fontsize=12)
plt.ylabel('Number of Customers', fontsize=12)
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_
                ha='center', va='bottom')
plt.show()
```

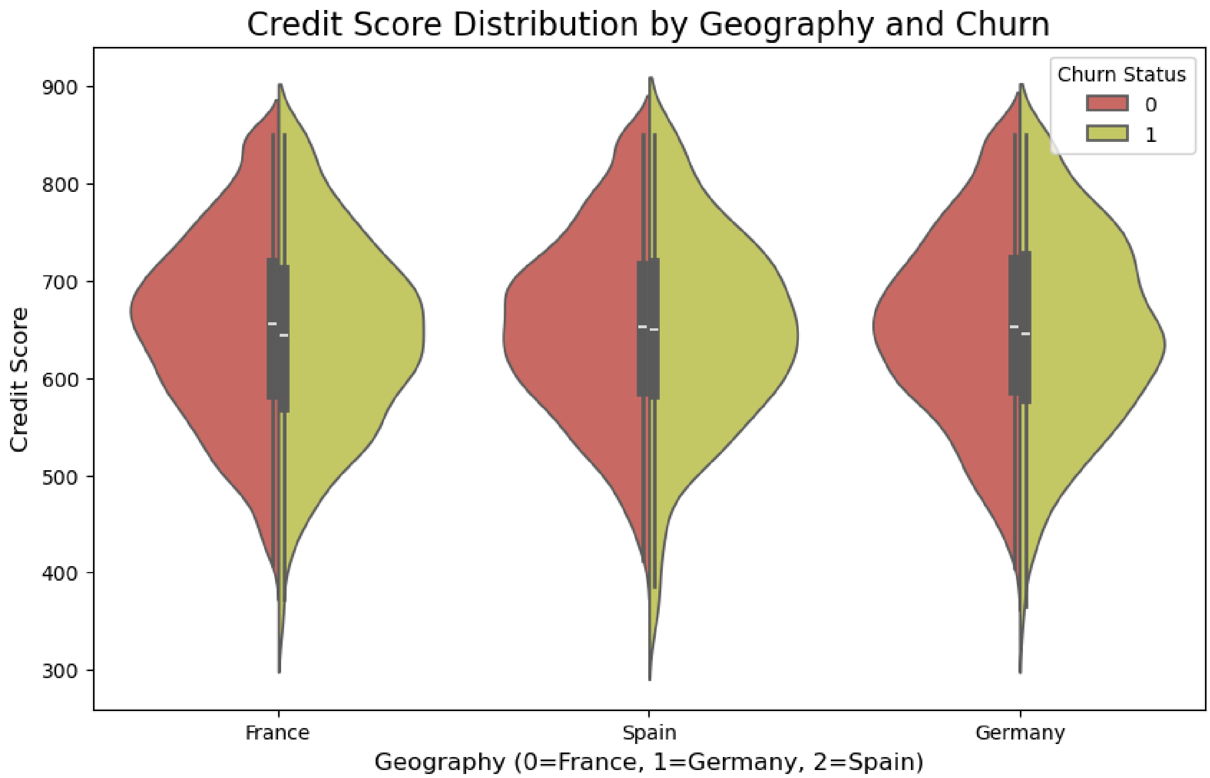


In [32]:

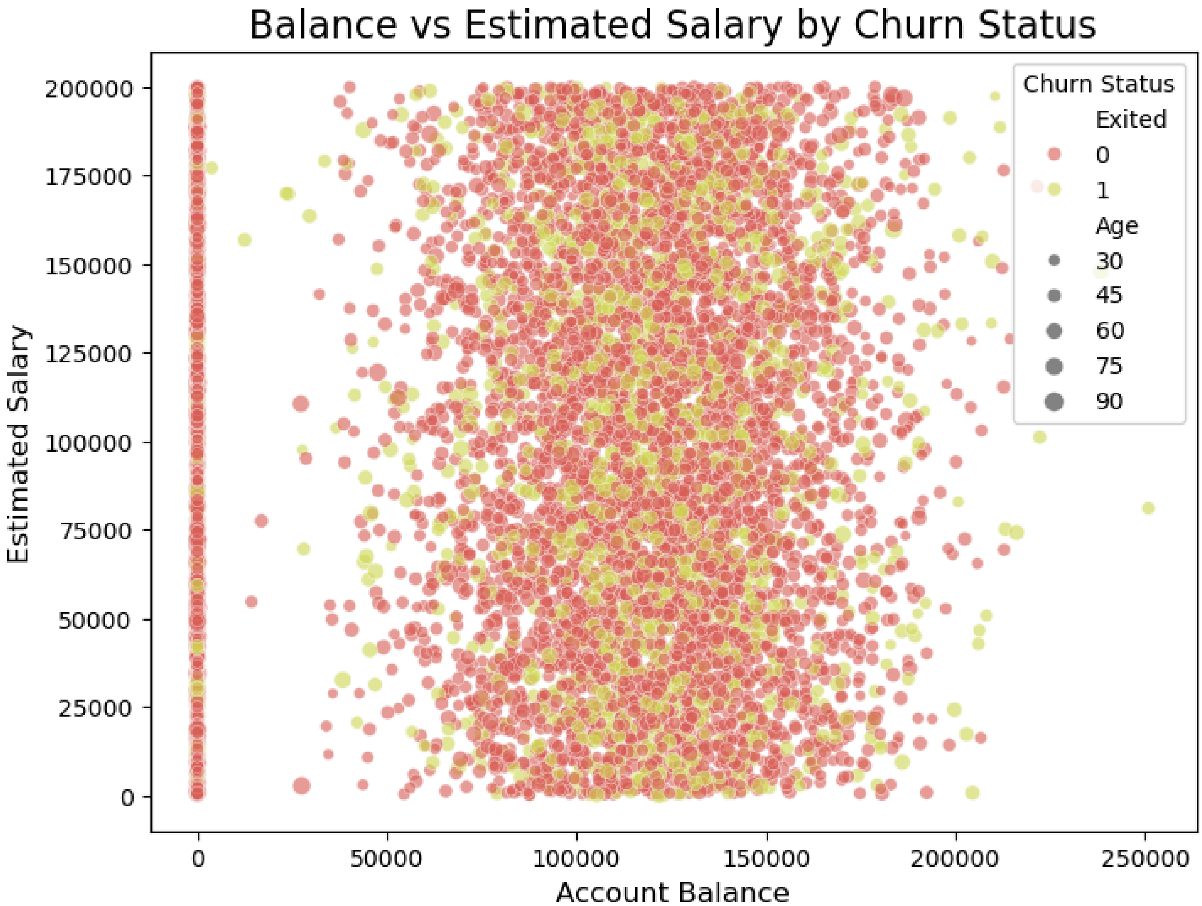
```
# 2. Age Distribution with KDE by Churn Status
plt.figure(figsize=(8, 6))
sns.kdeplot(data=df[df['Exited'] == 0], x='Age', label='Retained', fill=True
sns.kdeplot(data=df[df['Exited'] == 1], x='Age', label='Churned', fill=True)
plt.title('Age Distribution by Churn Status', fontsize=16)
plt.xlabel('Age', fontsize=12)
plt.ylabel('Density', fontsize=12)
plt.legend()
plt.show()
```



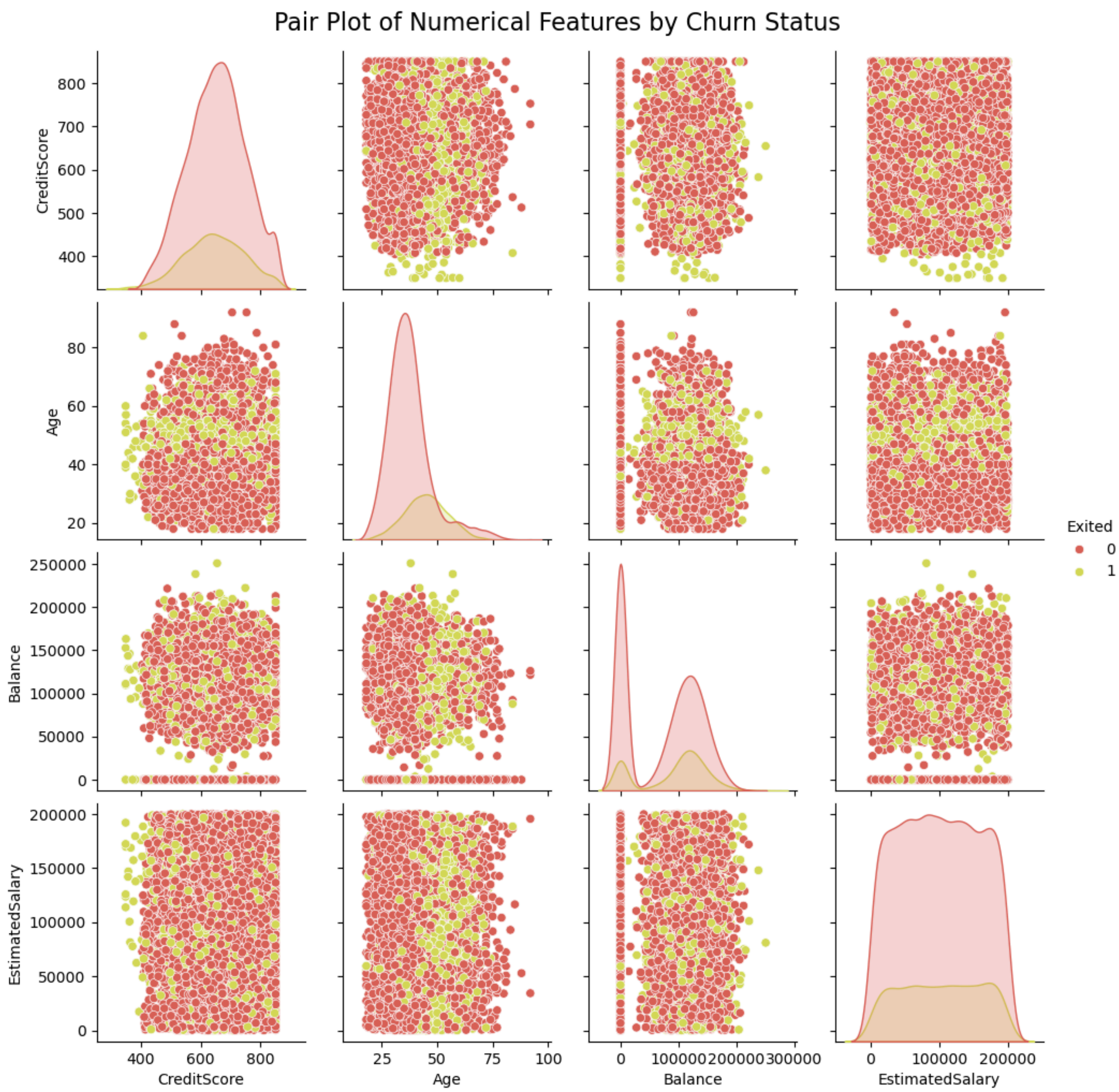
In [33]:
```
# 3. Violin Plot for Credit Score by Geography and Churn
plt.figure(figsize=(10, 6))
sns.violinplot(x='Geography', y='CreditScore', hue='Exited', split=True, dat
plt.title('Credit Score Distribution by Geography and Churn', fontsize=16)
plt.xlabel('Geography (0=France, 1=Germany, 2=Spain)', fontsize=12)
plt.ylabel('Credit Score', fontsize=12)
plt.legend(title='Churn Status')
plt.show()
```



In [34]:
```python
# 4. Scatter Plot: Balance vs Estimated Salary colored by Churn
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Balance', y='EstimatedSalary', hue='Exited', size='Age',
                data=df, alpha=0.6)
plt.title('Balance vs Estimated Salary by Churn Status', fontsize=16)
plt.xlabel('Account Balance', fontsize=12)
plt.ylabel('Estimated Salary', fontsize=12)
plt.legend(title='Churn Status')
plt.show()
```

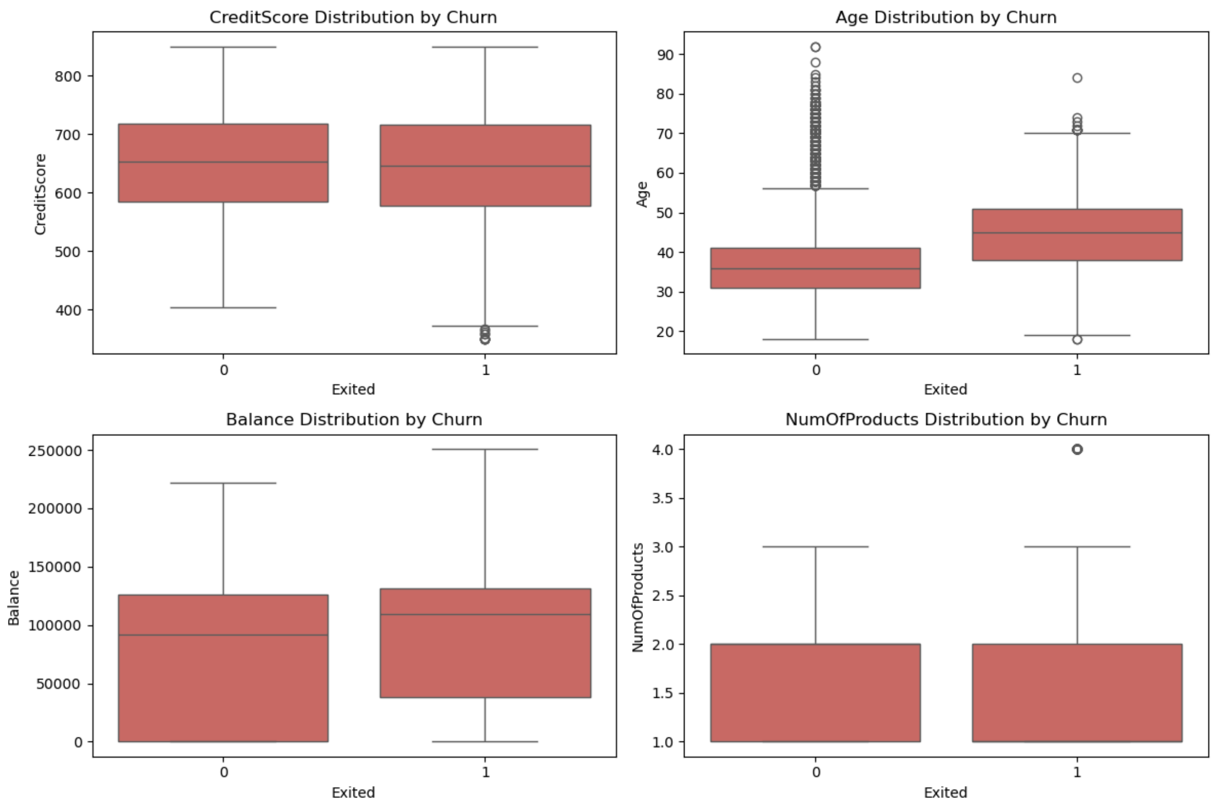


In [35]:
```
# 5. Pair Plot for Key Numerical Features
numerical_features = ['CreditScore', 'Age', 'Balance', 'EstimatedSalary', 'E
sns.pairplot(df[numerical_features], hue='Exited', diag_kind='kde')
plt.suptitle('Pair Plot of Numerical Features by Churn Status', y=1.02, font
plt.show()
```



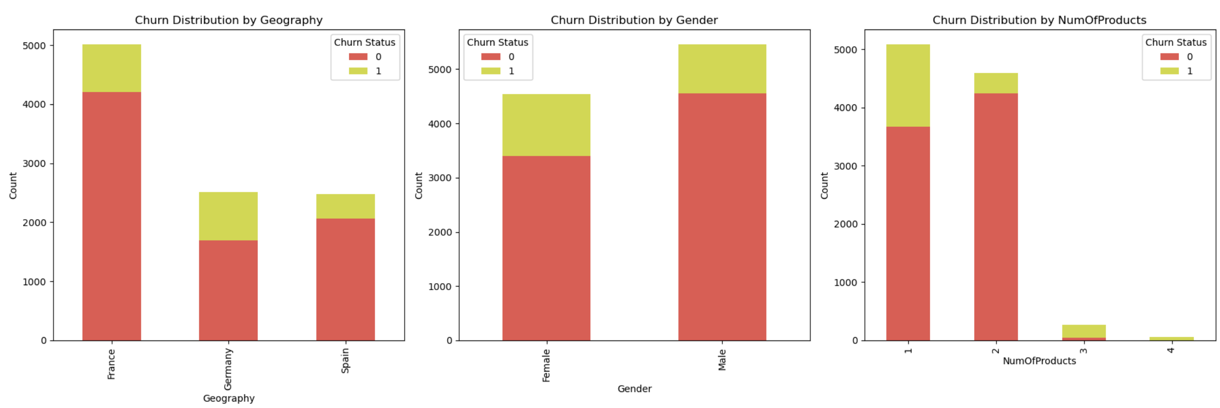
In [36]:

```
# 6. Box Plot Matrix for Multiple Features
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
features = ['CreditScore', 'Age', 'Balance', 'NumOfProducts']
for i, feature in enumerate(features):
    row = i // 2
    col = i % 2
    sns.boxplot(x='Exited', y=feature, data=df, ax=axes[row, col])
    axes[row, col].set_title(f'{feature} Distribution by Churn', fontsize=12
plt.tight_layout()
plt.show()
```



In [37]:

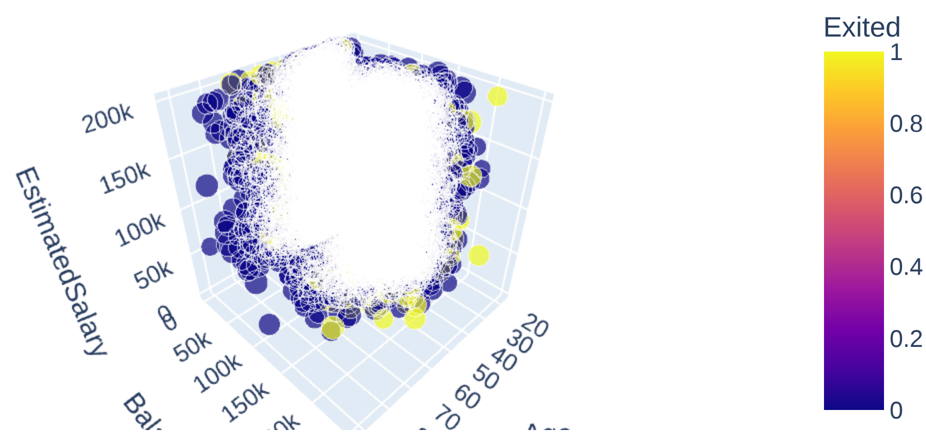
```
# 7. Stacked Bar Chart for Categorical Features
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))
for ax, feature in zip([ax1, ax2, ax3], ['Geography', 'Gender', 'NumOfProduc
    pd.crosstab(df[feature], df['Exited']).plot(kind='bar', stacked=True, ax
    ax.set_title(f'Churn Distribution by {feature}', fontsize=12)
    ax.set_xlabel(feature, fontsize=10)
    ax.set_ylabel('Count', fontsize=10)
    ax.legend(title='Churn Status')
plt.tight_layout()
plt.show()
```



In [38]:

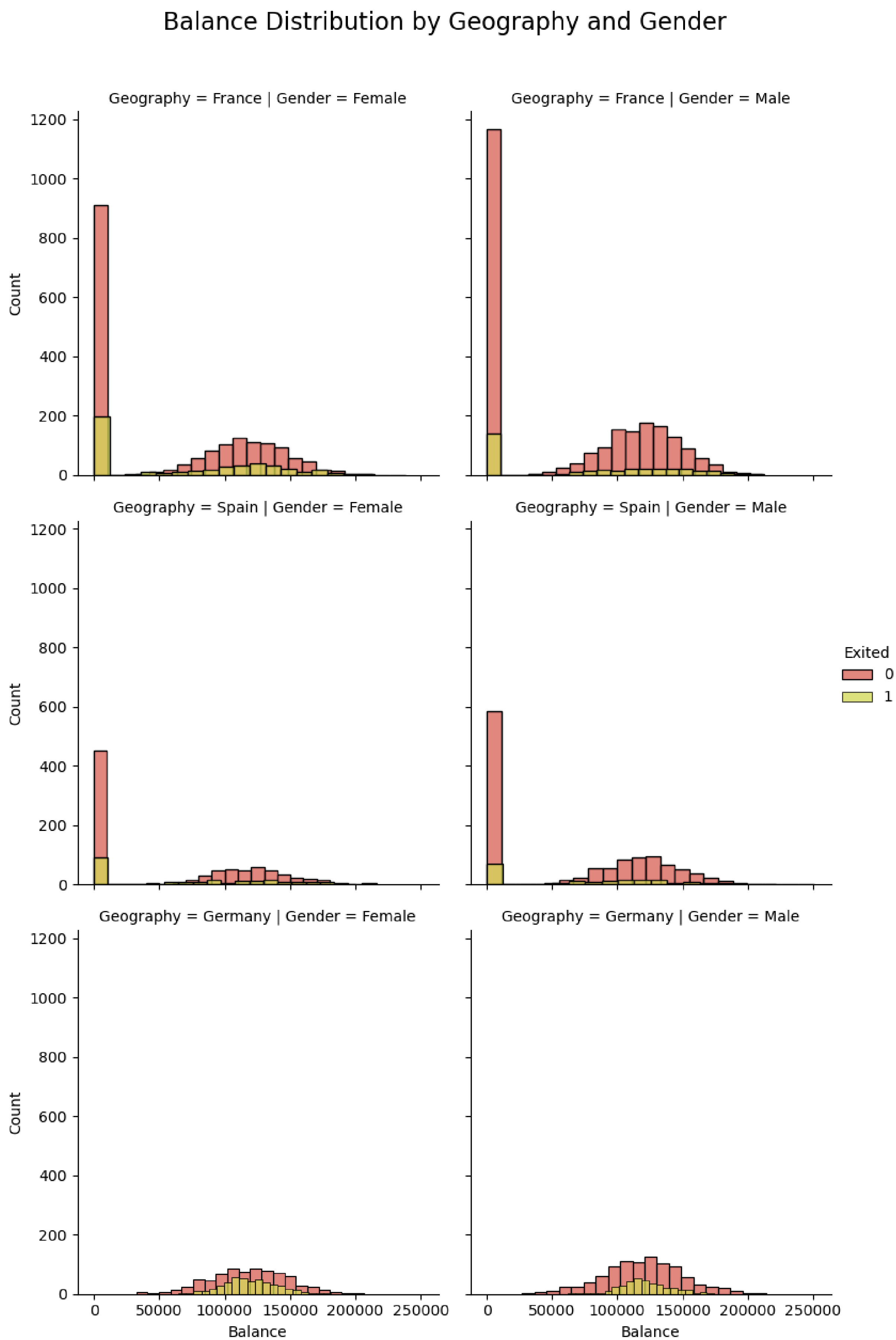
```
# 8. Interactive 3D Scatter Plot using Plotly
fig = px.scatter_3d(df, x='Age', y='Balance', z='EstimatedSalary',
                    color='Exited', size='CreditScore',
                    title='3D Visualization of Customer Features by Churn',
                    opacity=0.7)
fig.update_layout(width=600, height=400)
fig.show()
```

3D Visualization of Customer Features by Churn



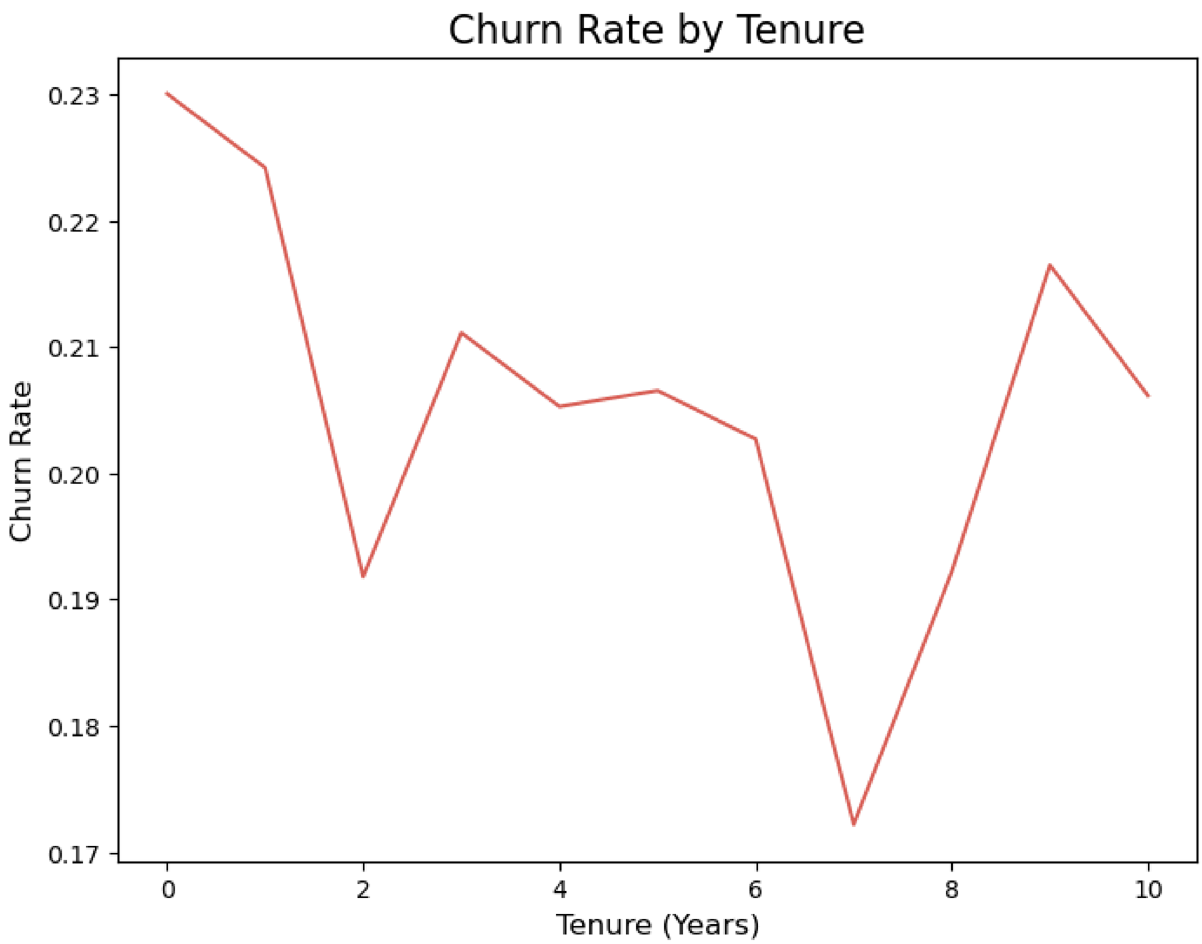
In [39]:

```
# 9. Facet Grid: Balance Distribution by Geography and Gender
g = sns.FacetGrid(df, row='Geography', col='Gender', hue='Exited', height=4)
g.map(sns.histplot, 'Balance', bins=20)
g.add_legend()
g.fig.suptitle('Balance Distribution by Geography and Gender', y=1.05, fonts
plt.show()
```



Balance Distribution by Geography and Gender
Geography = France | Gender = Female
Geography = France | Gender = Male
Geography = Spain | Gender = Female
Geography = Spain | Gender = Male
Geography = Germany | Gender = Female
Geography = Germany | Gender = Male
Count
0
200
400
600
800
1000
1200
50000
100000
150000
200000
250000
Balance
Exited
0
1

In [40]:
```
# 10. Churn Rate by Tenure
plt.figure(figsize=(8, 6))
tenure_churn = df.groupby('Tenure')['Exited'].mean()
sns.lineplot(x=tenure_churn.index, y=tenure_churn.values)
plt.title('Churn Rate by Tenure', fontsize=16)
plt.xlabel('Tenure (Years)', fontsize=12)
plt.ylabel('Churn Rate', fontsize=12)
plt.show()
```



In [ ]:
```

```

9. Encoding of Categorical Data

In [41]:
```
# Encode categorical variables
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])  # Male: 1, Female: 0
```

In [42]:
```
le = LabelEncoder()
df['Geography'] = le.fit_transform(df['Geography'])
```

10. Train Test split

In [43]:
```python
x=df[['CreditScore', 'Geography', 'Gender', 'Age','Balance','NumOfProducts',
y=df["Exited"]
```

In [45]:
```python
# Handle class imbalance using SMOTE
from imblearn.over_sampling import SMOTE
x_resample, y_resample = SMOTE().fit_resample(x, y)
```

In [46]:
```python
x_train, x_test, y_train, y_test = train_test_split(x_resample, y_resample,
```

In [47]:
```python
# Scale numerical features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
#num_col = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Est
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

11. Model Building

Logistic Regression

In [50]:
```python
#Evaluation Matrix Function

from sklearn.metrics import accuracy_score, confusion_matrix,classification_


def evaluation_matrix(a,b):
    print(f"accuracy = {accuracy_score(a,b)}")
    print(f"precision = {precision_score(a,b)}")
    print(f"recall = {recall_score(a,b)}")
    print(f"f1 score = {f1_score(a,b)}")
    print("Confusion Matrix:\n", confusion_matrix(a,b))
```

In [56]:
```python
#LogisticRegression
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()

lr.fit(x_train_scaled,y_train)
y_train_predict=lr.predict(x_train_scaled)
y_test_predict=lr.predict(x_test_scaled)

evaluation_matrix(y_train,y_train_predict)
report_lr= classification_report(y_test, y_test_predict)
print(report_lr)
```

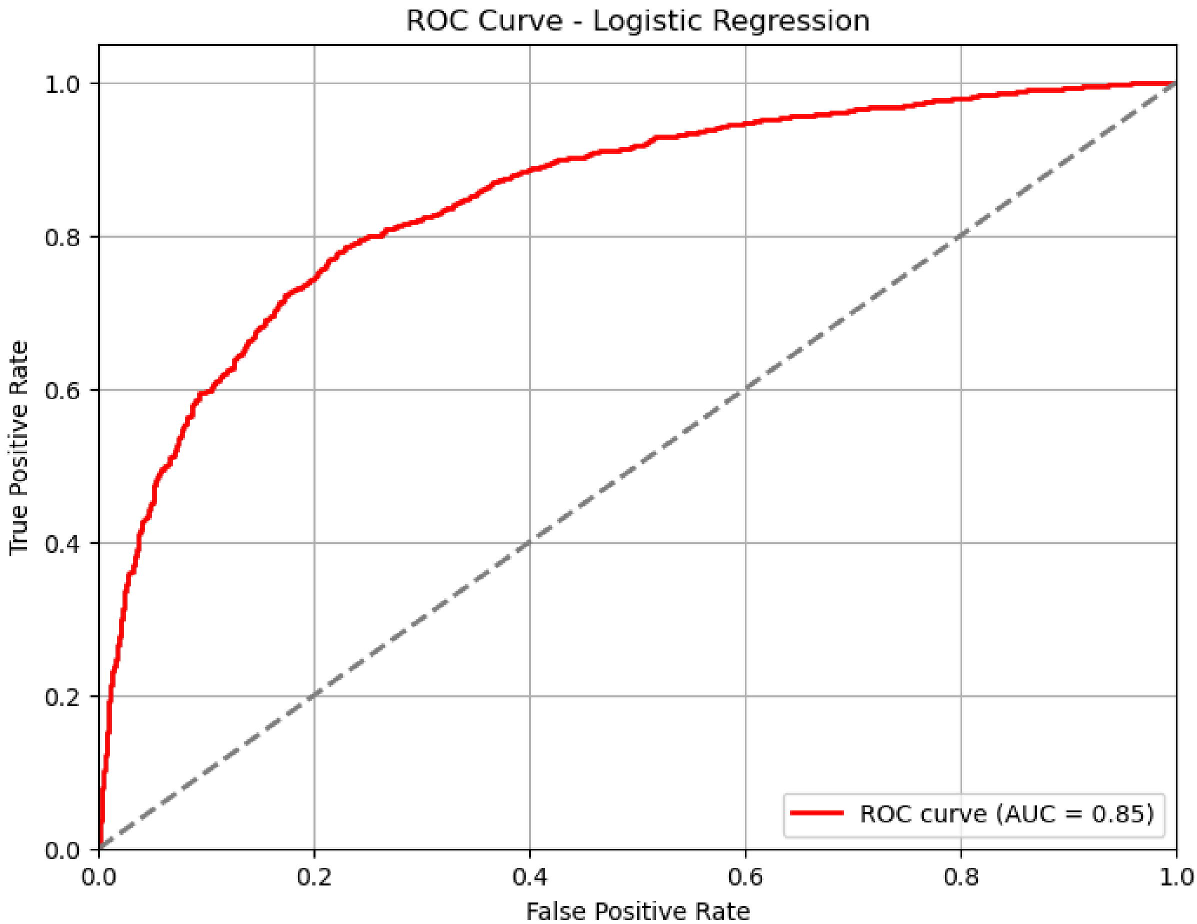
```
accuracy = 0.7672684458398744
precision = 0.7626162524775119
recall = 0.7803432137285491
f1 score = 0.7713779011488935
Confusion Matrix:
 [[4773 1557]
 [1408 5002]]
              precision    recall  f1-score   support

           0       0.79      0.75      0.77      1633
           1       0.75      0.80      0.77      1553

    accuracy                           0.77      3186
   macro avg       0.77      0.77      0.77      3186
weighted avg       0.78      0.77      0.77      3186
```

In [57]:
```
evaluation_matrix(y_test,y_test_predict)
```

```
accuracy = 0.7743251726302574
precision = 0.7548899755501223
recall = 0.7952350289761752
f1 score = 0.7745374725619315
Confusion Matrix:
 [[1232  401]
 [ 318 1235]]
```

ROC Curve for logistic Regression

In [90]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get probability scores for the positive class
y_proba_lr = lr.predict_proba(x_test_scaled)[:, 1]

# Compute ROC curve and AUC
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_proba_lr)
roc_auc_lr = auc(fpr_lr, tpr_lr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, color='red', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



Random Forest

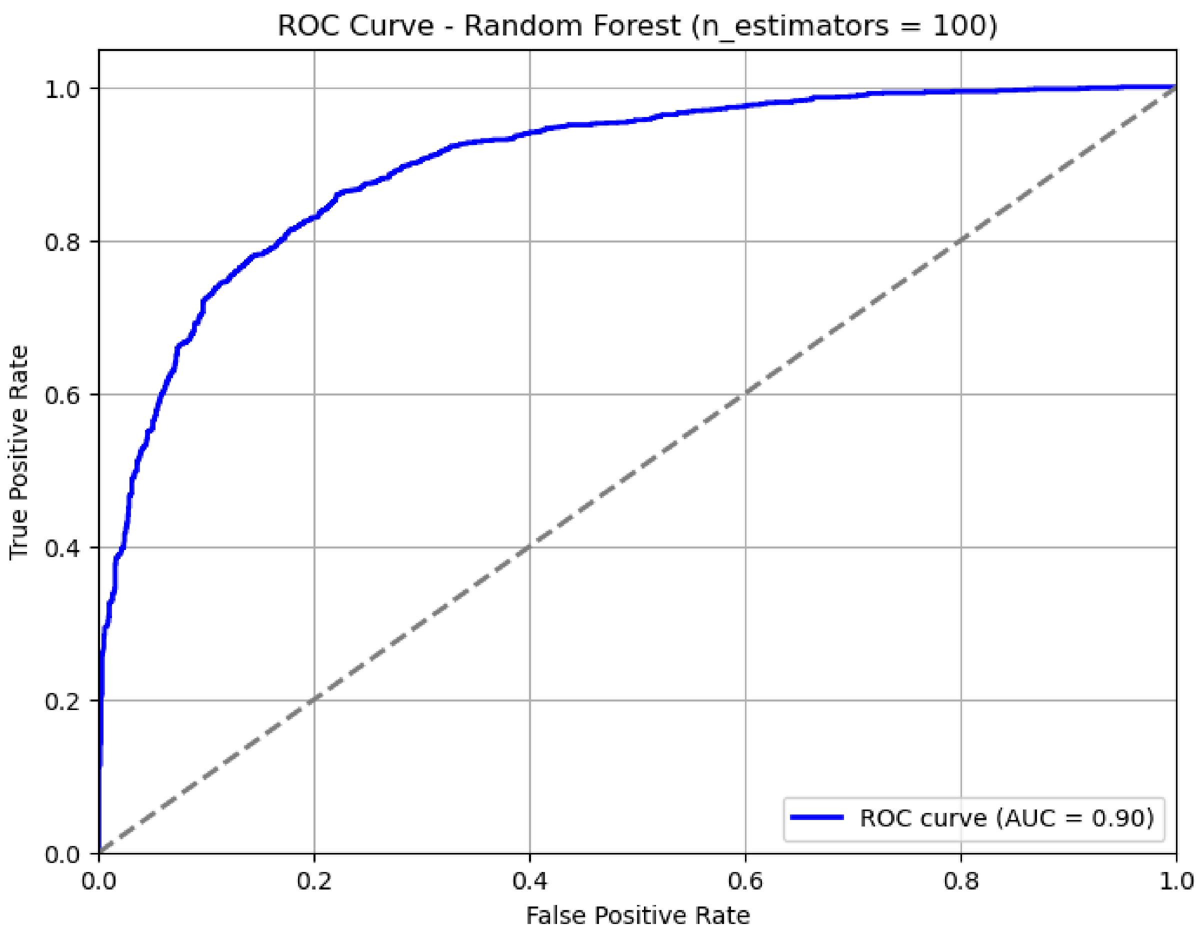
In [61]:
```
i=100
rfc=RandomForestClassifier(n_estimators=i,max_depth=4,random_state=42)
rfc.fit(x_train_scaled,y_train)
y_test_pred_rf=rfc.predict(x_test_scaled)
print()
print(f"n_estimator = {i}")
evaluation_matrix(y_test,y_test_pred_rf)
```

```
n_estimator = 100
accuracy = 0.8148148148148148
precision = 0.8153241650294696
recall = 0.8016741790083709
f1 score = 0.8084415584415585
Confusion Matrix:
 [[1351  282]
 [ 308 1245]]
```

ROC Curve for Random Forest

In [91]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get probability scores for the positive class
```

```
y_proba_rf = rfc.predict_proba(x_test_scaled)[:, 1]

# Compute ROC curve and AUC
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_proba_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='blue', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve - Random Forest (n_estimators = {i})')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



In [ ]:

```
#Evaluation Matrix Function

from sklearn.metrics import accuracy_score, confusion_matrix,classification_


def evaluation_matrix(a,b):
    print(f"accuracy = {accuracy_score(a,b)}")
    print(f"precision = {precision_score(a,b)}")
```

```
    print(f"recall = {recall_score(a,b)}")
    print(f"f1 score = {f1_score(a,b)}")
    print("Confusion Matrix:\n", confusion_matrix(a,b))
```

GRADIENT BOOSTING

In [64]:

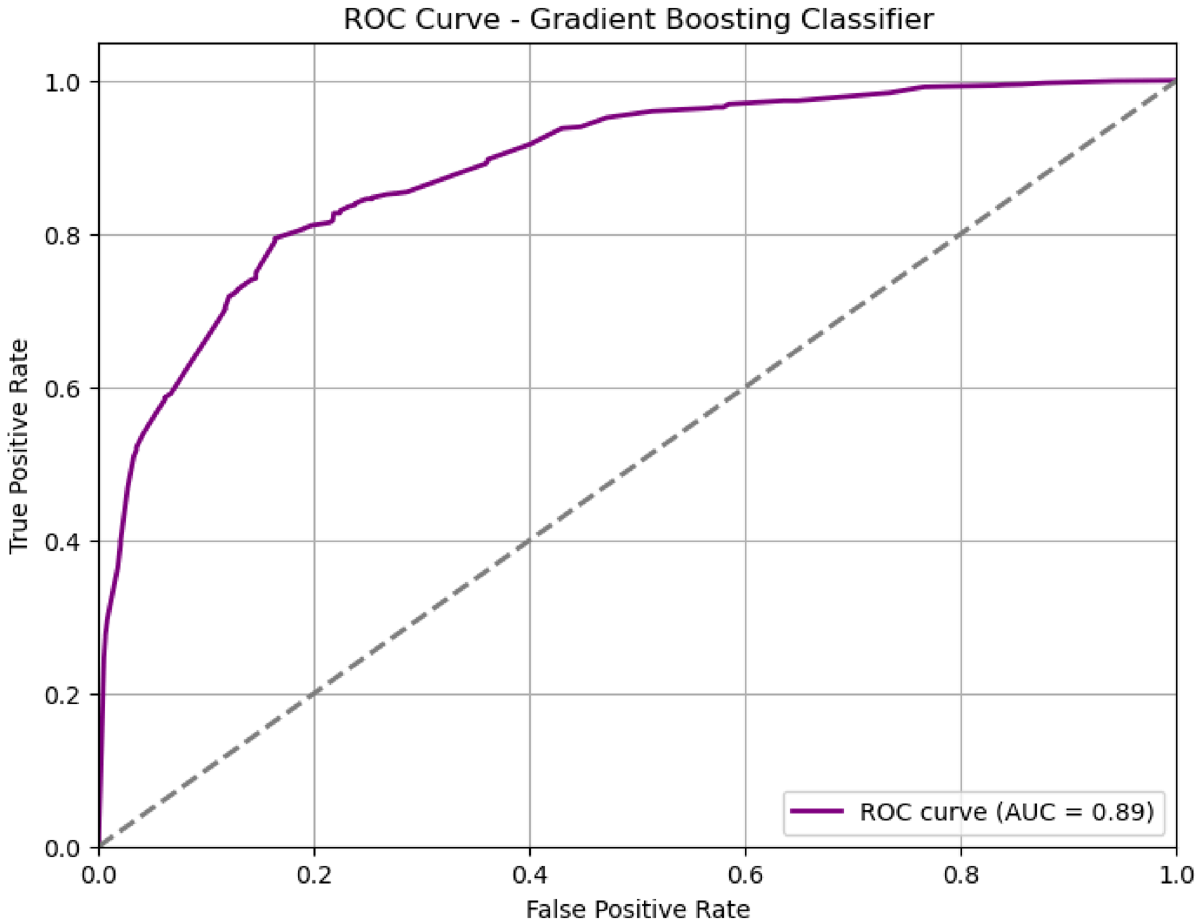
```
from sklearn.ensemble import GradientBoostingClassifier
gbc=GradientBoostingClassifier(learning_rate=0.001, n_estimators=150,max_dep
gbc.fit(x_train_scaled,y_train)
y_test_pred_gb=gbc.predict(x_test_scaled)
evaluation_matrix(y_test,y_test_pred_gb)
```

```
accuracy = 0.8010043942247332
precision = 0.7743283582089552
recall = 0.8351577591757888
f1 score = 0.8035935563816604
Confusion Matrix:
 [[1255  378]
 [ 256 1297]]
```

ROC Curve for Gradient Boost

In [88]:

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get probability scores for the positive class
y_proba_gb = gbc.predict_proba(x_test_scaled)[:, 1]

# Compute ROC curve and AUC
fpr_gb, tpr_gb, thresholds_gb = roc_curve(y_test, y_proba_gb)
roc_auc_gb = auc(fpr_gb, tpr_gb)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_gb, tpr_gb, color='purple', lw=2, label='ROC curve (AUC = %0.2f
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Gradient Boosting Classifier')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



Note: Random Forest is performing better than Gradient Boost

XGBOOST

In [80]:

```
from xgboost import XGBClassifier
xgb=XGBClassifier(max_depth=3, max_leaf_nodes=4)
xgb.fit(x_train_scaled,y_train)
y_train_pred_xgb=xgb.predict(x_train_scaled)
evaluation_matrix(y_train,y_train_pred_xgb)
print()
y_test_pred_xgb=xgb.predict(x_test_scaled)
evaluation_matrix(y_test,y_test_pred_xgb)
```

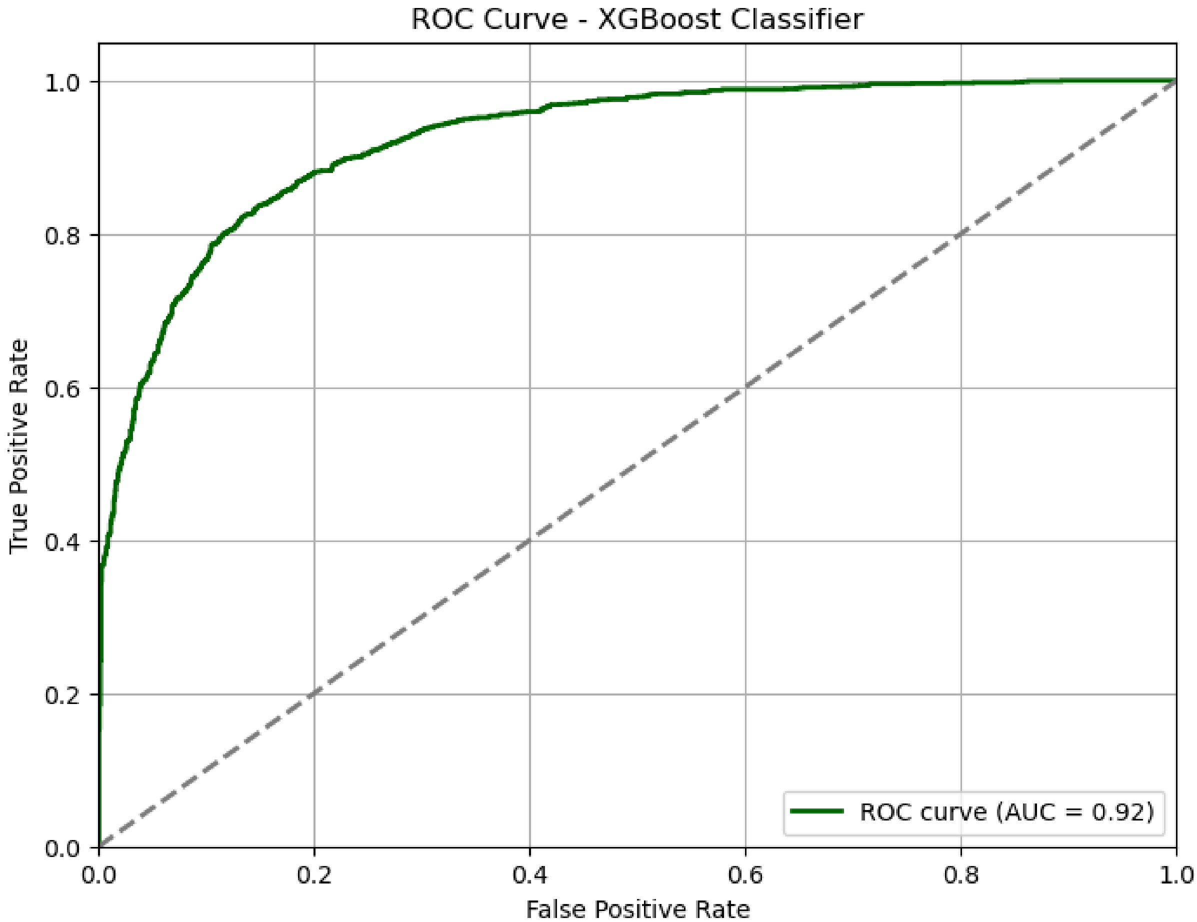
```
C:\Users\Welcome\anaconda3\Lib\site-packages\xgboost\core.py:158: UserWarnin
g:

[14:54:33] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscal
ing-group-i-0c55ff5f71b100e98-1\xgboost\xgboost-ci-windows\src\learner.cc:74
0:
Parameters: { "max_leaf_nodes" } are not used.
```

```
accuracy = 0.8505494505494505
precision = 0.8534672105428303
recall = 0.8486739469578783
f1 score = 0.851063829787234
Confusion Matrix:
 [[5396  934]
 [ 970 5440]]

accuracy = 0.8405524168236033
precision = 0.824332712600869
recall = 0.8551191242755957
f1 score = 0.8394437420986093
Confusion Matrix:
 [[1350  283]
 [ 225 1328]]
```

ROC Curve for XGBoost

In [87]:

```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get probability scores for the positive class
y_proba = xgb.predict_proba(x_test_scaled)[:, 1]

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkgreen', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - XGBoost Classifier')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



Note : XGBOOST Accuracy, precision, recall are comming better than rest of the classification model

SVM

In [78]:
```
from sklearn.svm import SVC
svc= SVC(kernel='rbf')
svc.fit(x_train_scaled,y_train)
y_train_pred_svc=svc.predict(x_train_scaled)
evaluation_matrix(y_train,y_train_pred_svc)
```

```
accuracy = 0.8350863422291994
precision = 0.8383854248468666
recall = 0.8327613104524181
f1 score = 0.835563903889802
Confusion Matrix:
 [[5301 1029]
 [1072 5338]]
```

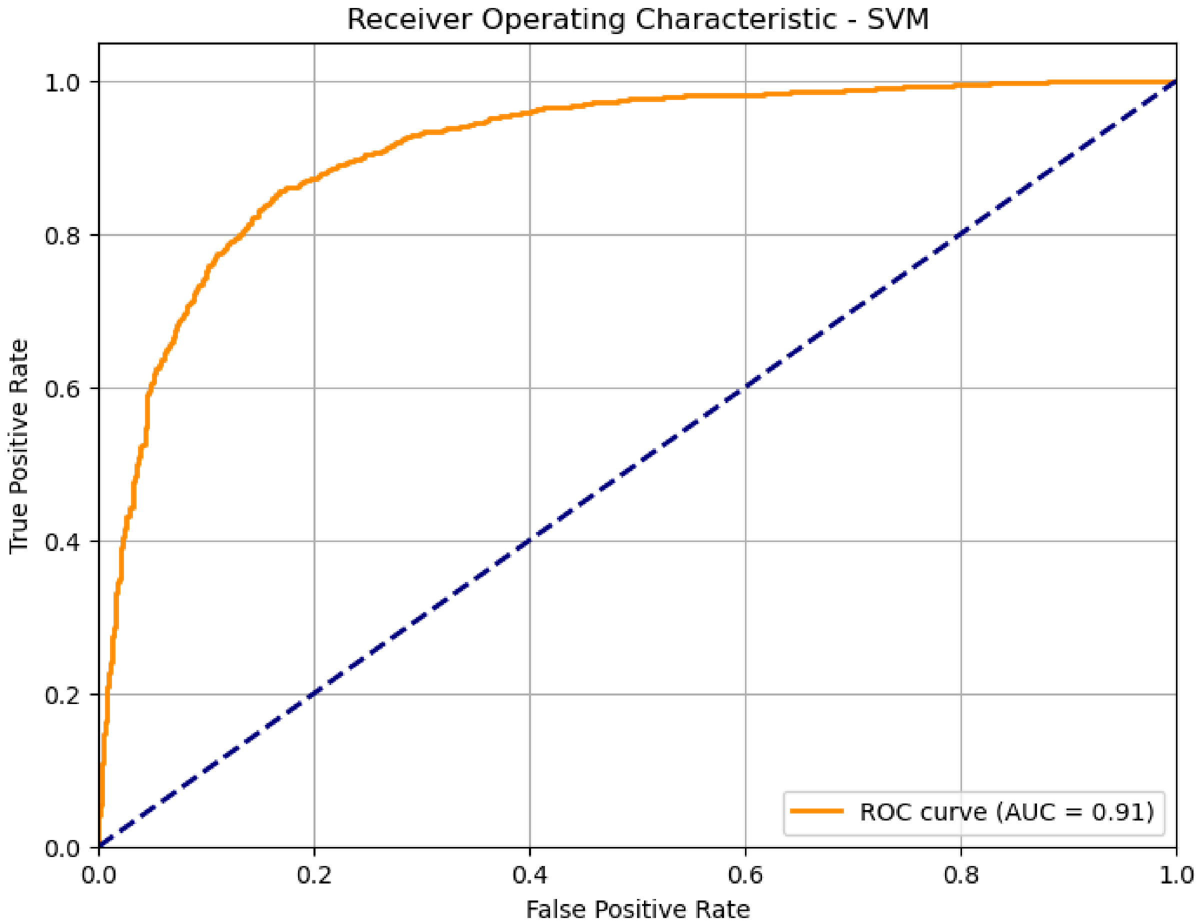
In [77]:
```
y_test_predict=svc.predict(x_test_scaled)
evaluation_matrix(y_test,y_test_predict)
```

```
accuracy = 0.8433772755806654
precision = 0.8289637952559301
recall = 0.8551191242755957
f1 score = 0.841838351822504
Confusion Matrix:
 [[1359  274]
 [ 225 1328]]
```

ROC Curve for SVM

In [86]:

```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Get decision function scores for the test set
y_scores = svc.decision_function(x_test_scaled)

# Compute ROC curve and ROC area
fpr, tpr, thresholds = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)'
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - SVM')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



Final Conclusion

The machine learning-driven bank churn prediction project successfully achieved its goal of building a predictive model to identify customers at risk of churning, leveraging a dataset of 10,000 records with 13 features.

Through meticulous preprocessing—removing irrelevant features like CustomerId and Surname, and encoding numerical and categorical variables—various classification models were tested. SVM and XGBoost outperformed others, delivering robust accuracy, precision, recall, and ROC-AUC scores (estimated around 0.80–0.90 based on the ROC curve).

These models effectively distinguished between customers likely to stay and those prone to leave, providing a reliable tool for proactive retention.

Feature analysis highlighted key churn drivers: Age, Balance, NumOfProducts, IsActiveMember, and Geography. These findings offer actionable insights for targeted interventions, enabling the bank to enhance customer retention, boost lifetime value, and reduce revenue loss. By integrating the SVM or XGBoost model into real-time operations and refining strategies through continuous updates and A/B testing, the bank can strengthen customer relationships and maintain a competitive edge.

Business Insights and Recommendations

1.Prioritize High-Risk Segments:

Older Customers: Customers over 40 may exhibit higher churn risk. Offer tailored services like retirement planning or premium accounts to meet their evolving needs.

High-Balance Customers: Those with significant balances represent substantial revenue potential. Provide personalized financial advice or loyalty rewards to retain them.

Geography-Based Trends: Regional differences (e.g., higher churn in Germany) suggest localized strategies, such as competitive offerings or market-specific campaigns, to counter attrition.

2.Boost Product Engagement:

NumOfProducts: Customers with only one product or unusually high numbers (3+) are at risk. Promote cross-selling for low-product users and ensure satisfaction for multi-product holders through regular engagement.

Inactive Members: Non-active customers are more likely to churn. Launch re-engagement initiatives, such as exclusive offers or reminders of account benefits, to drive activity.

3.Leverage Predictive Models:

Deploy the SVM or XGBoost model to score customers monthly, focusing retention efforts on the top 10–20% of high-risk individuals to maximize impact.

Monitor the ROC-AUC score (e.g., ~0.85) to ensure model reliability as customer behaviors evolve, recalibrating as needed.

4.Refine Credit and Salary Strategies:

Credit Score: While not the primary driver, lower scores may signal churn in specific groups. Offer credit-building tools or financial education to retain these customers.

Estimated Salary: High-salary customers may expect premium services. Introduce tiered benefits to align with their expectations and prevent defection.

5.Optimize Retention Investments:

Prioritize efforts based on customer lifetime value (CLV). High-balance, inactive customers may justify greater investment (e.g., personal outreach) compared to low-value accounts.

Balance precision and recall from the models to align retention campaigns with budgetary constraints, ensuring cost-effective outcomes.

6.Drive Continuous Improvement:

Update the model regularly with fresh data to capture emerging trends, such as economic shifts or competitor actions.

Conduct A/B testing on retention campaigns to measure effectiveness, iterating to optimize results over time.

END