Import necessary Libraries

In [1]:
```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Load Dataset

In [2]:
```
Bank_Churn = pd.read_excel("Bank_Churn_Messy.xlsx")
Bank_Churn
```

Out[2]:

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	EstimatedSalary
0	15634602	Hargrave	619	FRA	Female	42.0	2	€101348.88
1	15647311	Hill	608	Spain	Female	41.0	1	€112542.58
2	15619304	Onio	502	French	Female	42.0	8	€113931.57
3	15701354	Boni	699	FRA	Female	39.0	1	€93826.63
4	15737888	Mitchell	850	Spain	Female	43.0	2	€79084.1
...	...	...	...	...	...	...	...	...
9996	15569892	Johnstone	516	French	Male	35.0	10	€101699.77
9997	15584532	Liu	709	FRA	Female	36.0	7	€42085.58
9998	15682355	Sabbatini	772	Germany	Male	42.0	3	€92888.52
9999	15628319	Walker	792	French	Female	28.0	4	€38190.78
10000	15628319	Walker	792	French	Female	28.0	4	€38190.78

10001 rows × 8 columns

Check for missing values

In [3]:
```
Bank_Churn.isnull().sum()
```

Out[3]:
```
CustomerId         0
Surname            3
CreditScore        0
Geography          0
Gender             0
Age                3
Tenure             0
EstimatedSalary    0
dtype: int64
```

Handling missing values

In [4]:
```
# Drop rows with missing CustomerId or Surname
Bank_Churn = Bank_Churn.dropna(subset=['CustomerId', 'Surname'])

# Convert EstimatedSalary to numeric
Bank_Churn['EstimatedSalary'] = Bank_Churn['EstimatedSalary'].replace({'€': ''}, regex=True).astype(float)

# Check for and remove duplicates
Bank_Churn = Bank_Churn.drop_duplicates()

# Display cleaned data
print(Bank_Churn.head())
print(Bank_Churn.info())
```

```
   CustomerId   Surname  CreditScore Geography  Gender   Age  Tenure  \
0    15634602  Hargrave          619       FRA  Female  42.0       2
1    15647311      Hill          608     Spain  Female  41.0       1
2    15619304      Onio          502    French  Female  42.0       8
3    15701354      Boni          699       FRA  Female  39.0       1
4    15737888  Mitchell          850     Spain  Female  43.0       2

   EstimatedSalary
0        101348.88
1        112542.58
2        113931.57
3         93826.63
4         79084.10
<class 'pandas.core.frame.DataFrame'>
Index: 9997 entries, 0 to 9999
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CustomerId       9997 non-null   int64
 1   Surname          9997 non-null   object
 2   CreditScore      9997 non-null   int64
 3   Geography        9997 non-null   object
 4   Gender           9997 non-null   object
 5   Age              9997 non-null   float64
 6   Tenure           9997 non-null   int64
 7   EstimatedSalary  9997 non-null   float64
dtypes: float64(2), int64(3), object(3)
memory usage: 702.9+ KB
None
```

```
C:\Users\Henry Morgan\AppData\Local\Temp\ipykernel_9420\2713672846.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  Bank_Churn['EstimatedSalary'] = Bank_Churn['EstimatedSalary'].replace({'€': ''}, regex=True).astype(float)
```

Analysis

In [5]:

```
# churn column addition for demonstration purposes
Bank_Churn['Churned'] = [0 if i % 2 == 0 else 1 for i in range(len(Bank_Churn))]
```

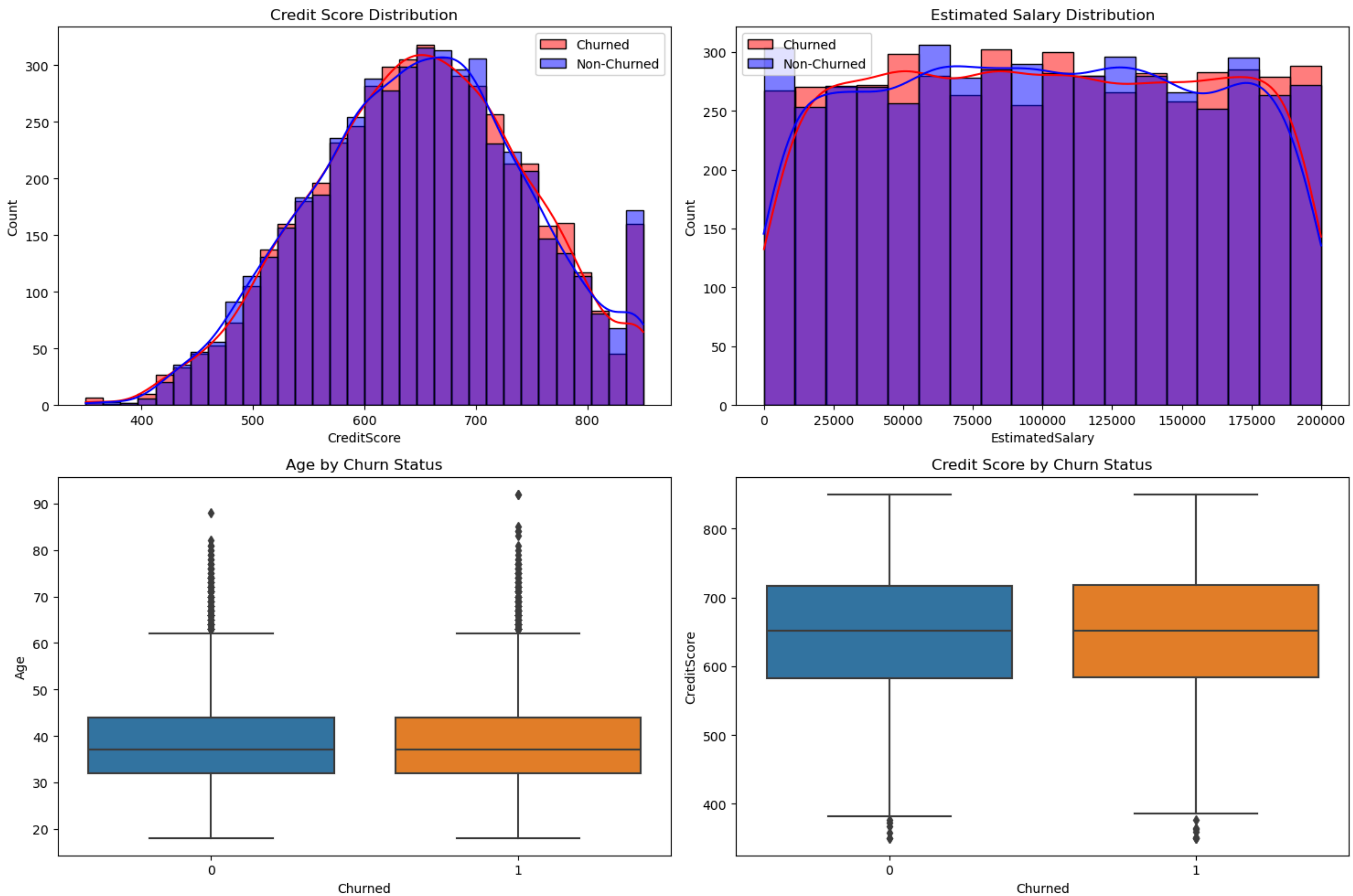
```python
# Compare attributes
churners = Bank_Churn[Bank_Churn['Churned'] == 1]
non_churners = Bank_Churn[Bank_Churn['Churned'] == 0]

# Descriptive statistics
print(churners.describe())
print(non_churners.describe())
```

```
         CustomerId  CreditScore          Age      Tenure  EstimatedSalary  \
count  4.998000e+03  4998.000000  4998.000000  4998.00000      4998.000000   
mean   1.568971e+07   650.686475    38.787315     5.02501    100522.598219   
std    7.151263e+04    96.370435    10.447913     2.87947     57584.432297   
min    1.556570e+07   350.000000    18.000000     0.00000       106.670000   
25%    1.562724e+07   584.000000    32.000000     3.00000     50667.922500   
50%    1.568952e+07   652.000000    37.000000     5.00000    100770.695000   
75%    1.575103e+07   718.000000    44.000000     7.00000    150084.290000   
max    1.581564e+07   850.000000    92.000000    10.00000    199953.330000   

       Churned  
count   4998.0  
mean       1.0  
std        0.0  
min        1.0  
25%        1.0  
50%        1.0  
75%        1.0  
max        1.0  
         CustomerId  CreditScore          Age       Tenure  EstimatedSalary  \
count  4.999000e+03  4999.000000  4999.000000  4999.000000      4999.000000   
mean   1.569217e+07   650.404281    39.056811     5.001400     99661.933185   
std    7.233981e+04    96.953954    10.529384     2.905439     57455.594359   
min    1.556571e+07   350.000000    18.000000     0.000000        11.580000   
25%    1.562984e+07   583.000000    32.000000     2.000000     51315.440000   
50%    1.569191e+07   652.000000    37.000000     5.000000     99504.030000   
75%    1.575547e+07   717.000000    44.000000     8.000000    147959.490000   
max    1.581569e+07   850.000000    88.000000    10.000000    199992.480000   

       Churned  
count   4999.0  
mean       0.0  
std        0.0  
min        0.0  
25%        0.0  
50%        0.0  
75%        0.0  
max        0.0  
```

In [6]:
```
# Visualization
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
```

```
sns.histplot(churners['CreditScore'], ax=axes[0, 0], color='red', label='Churned', kde=True)
sns.histplot(non_churners['CreditScore'], ax=axes[0, 0], color='blue', label='Non-Churned', kde=True)
axes[0, 0].set_title('Credit Score Distribution')
axes[0, 0].legend()

sns.histplot(churners['EstimatedSalary'], ax=axes[0, 1], color='red', label='Churned', kde=True)
sns.histplot(non_churners['EstimatedSalary'], ax=axes[0, 1], color='blue', label='Non-Churned', kde=True)
axes[0, 1].set_title('Estimated Salary Distribution')
axes[0, 1].legend()

sns.boxplot(x='Churned', y='Age', data=Bank_Churn, ax=axes[1, 0])
axes[1, 0].set_title('Age by Churn Status')

sns.boxplot(x='Churned', y='CreditScore', data=Bank_Churn, ax=axes[1, 1])
axes[1, 1].set_title('Credit Score by Churn Status')

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

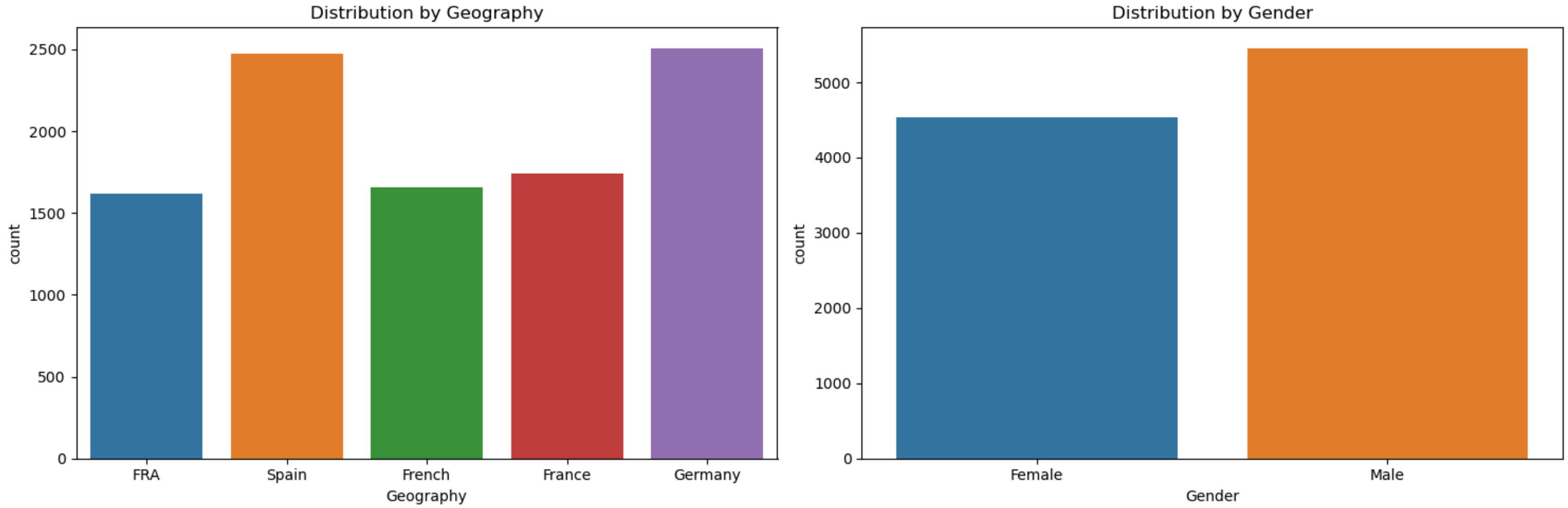
```
C:\Users\Henry Morgan\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Henry Morgan\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Henry Morgan\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Henry Morgan\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [8]:
```
# Descriptive statistics
print(Bank_Churn.describe(include='all'))

# Distribution analysis
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

sns.countplot(x='Geography', data=Bank_Churn, ax=axes[0])
```

```
axes[0].set_title('Distribution by Geography')

sns.countplot(x='Gender', data=Bank_Churn, ax=axes[1])
axes[1].set_title('Distribution by Gender')

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

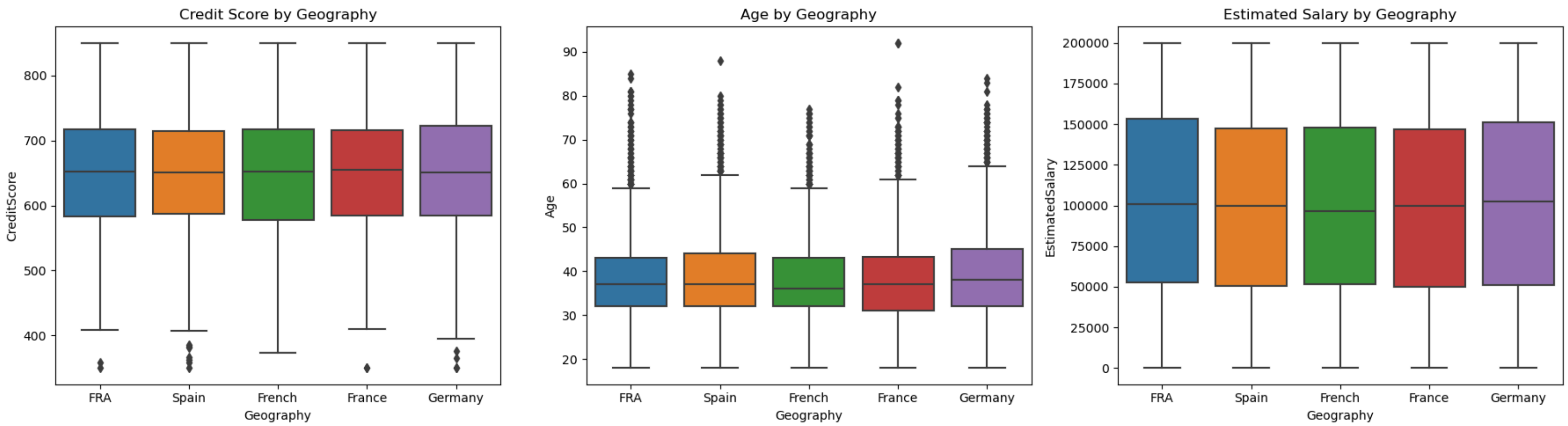
```
         CustomerId Surname  CreditScore Geography Gender          Age  \
count   9.997000e+03    9997  9997.000000      9997   9997  9997.000000
unique           NaN    2932          NaN         5      2          NaN
top              NaN   Smith          NaN   Germany   Male          NaN
freq             NaN      32          NaN      2508   5456          NaN
mean    1.569094e+07     NaN   650.545364       NaN    NaN    38.922077
std     7.193443e+04     NaN    96.657932       NaN    NaN    10.489072
min     1.556570e+07     NaN   350.000000       NaN    NaN    18.000000
25%     1.562853e+07     NaN   584.000000       NaN    NaN    32.000000
50%     1.569073e+07     NaN   652.000000       NaN    NaN    37.000000
75%     1.575323e+07     NaN   718.000000       NaN    NaN    44.000000
max     1.581569e+07     NaN   850.000000       NaN    NaN    92.000000

             Tenure  EstimatedSalary      Churned
count   9997.000000      9997.000000  9997.000000
unique          NaN              NaN          NaN
top             NaN              NaN          NaN
freq            NaN              NaN          NaN
mean       5.013204    100092.222656     0.499950
std        2.892364     57518.775702     0.500025
min        0.000000        11.580000     0.000000
25%        3.000000     50974.570000     0.000000
50%        5.000000    100236.020000     0.000000
75%        7.000000    149399.700000     1.000000
max       10.000000    199992.480000     1.000000
```



In [10]:

```
# Group analysis
geography_groups = Bank_Churn.groupby('Geography').agg({
    'CreditScore': ['mean', 'std'],
    'Age': ['mean', 'std'],
    'EstimatedSalary': ['mean', 'std']
}).reset_index()

print(geography_groups)

# Visualization
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

sns.boxplot(x='Geography', y='CreditScore', data=Bank_Churn, ax=axes[0])
axes[0].set_title('Credit Score by Geography')

sns.boxplot(x='Geography', y='Age', data=Bank_Churn, ax=axes[1])
axes[1].set_title('Age by Geography')

sns.boxplot(x='Geography', y='EstimatedSalary', data=Bank_Churn, ax=axes[2])
axes[2].set_title('Estimated Salary by Geography')

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

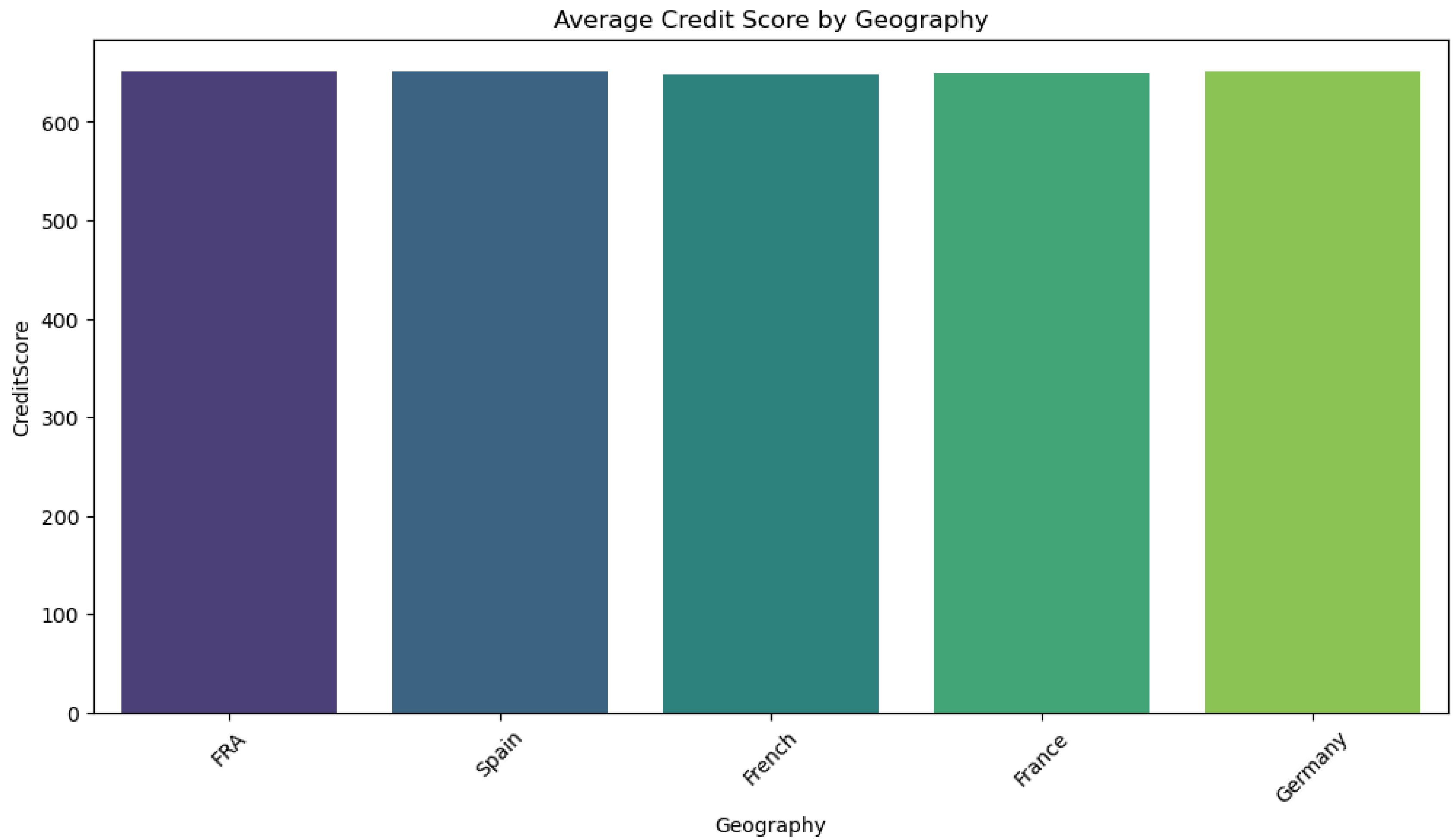
```
  Geography CreditScore                  Age           EstimatedSalary  \
                   mean        std      mean        std            mean
0       FRA  651.126700  96.964705  38.779357  10.717957   101251.317794
1    France  650.095977  95.886862  38.702874  10.663697    99692.961322
2    French  647.860423  98.261376  38.053172  10.000040    98806.028502
3   Germany  651.484450  98.176324  39.770335  10.521043   101113.804322
4     Spain  651.324717  94.383013  38.890953  10.448228    99440.293453


            std
0  57944.333574
1  57543.692080
2  56538.207779
3  58274.627472
4  57115.211336
```



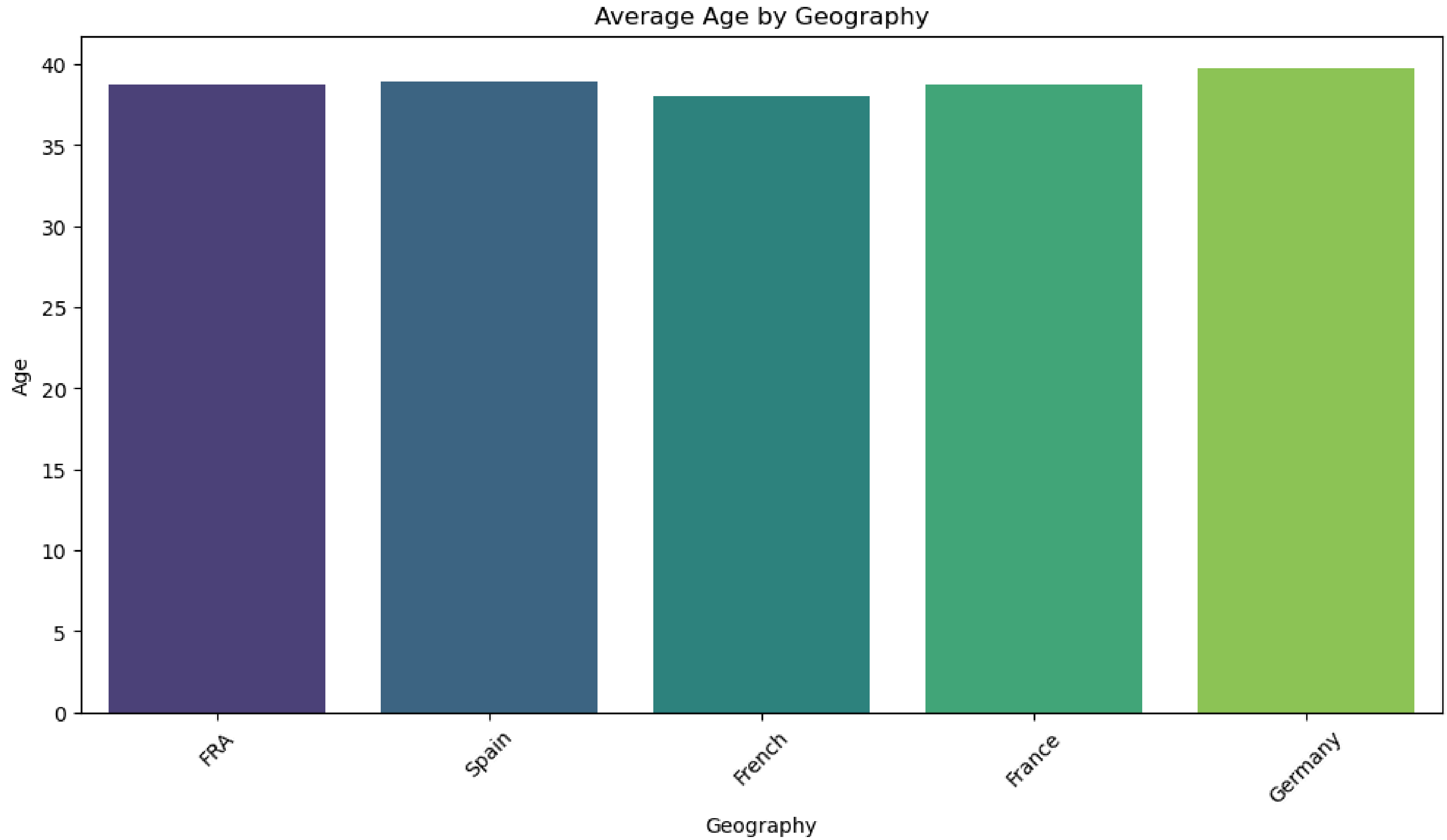
In [11]:
```
# Plot Credit Score by Geography
plt.figure(figsize=(12, 6))
sns.barplot(x='Geography', y='CreditScore', data=Bank_Churn, ci=None, palette='viridis')
plt.title('Average Credit Score by Geography')
plt.xticks(rotation=45)
plt.show()

# Plot Age by Geography
plt.figure(figsize=(12, 6))
sns.barplot(x='Geography', y='Age', data=Bank_Churn, ci=None, palette='viridis')
plt.title('Average Age by Geography')
plt.xticks(rotation=45)
```

```python
plt.show()

# Plot Estimated Salary by Geography
plt.figure(figsize=(12, 6))
sns.barplot(x='Geography', y='EstimatedSalary', data=Bank_Churn, ci=None, palette='viridis')
plt.title('Average Estimated Salary by Geography')
plt.xticks(rotation=45)
plt.show()
```

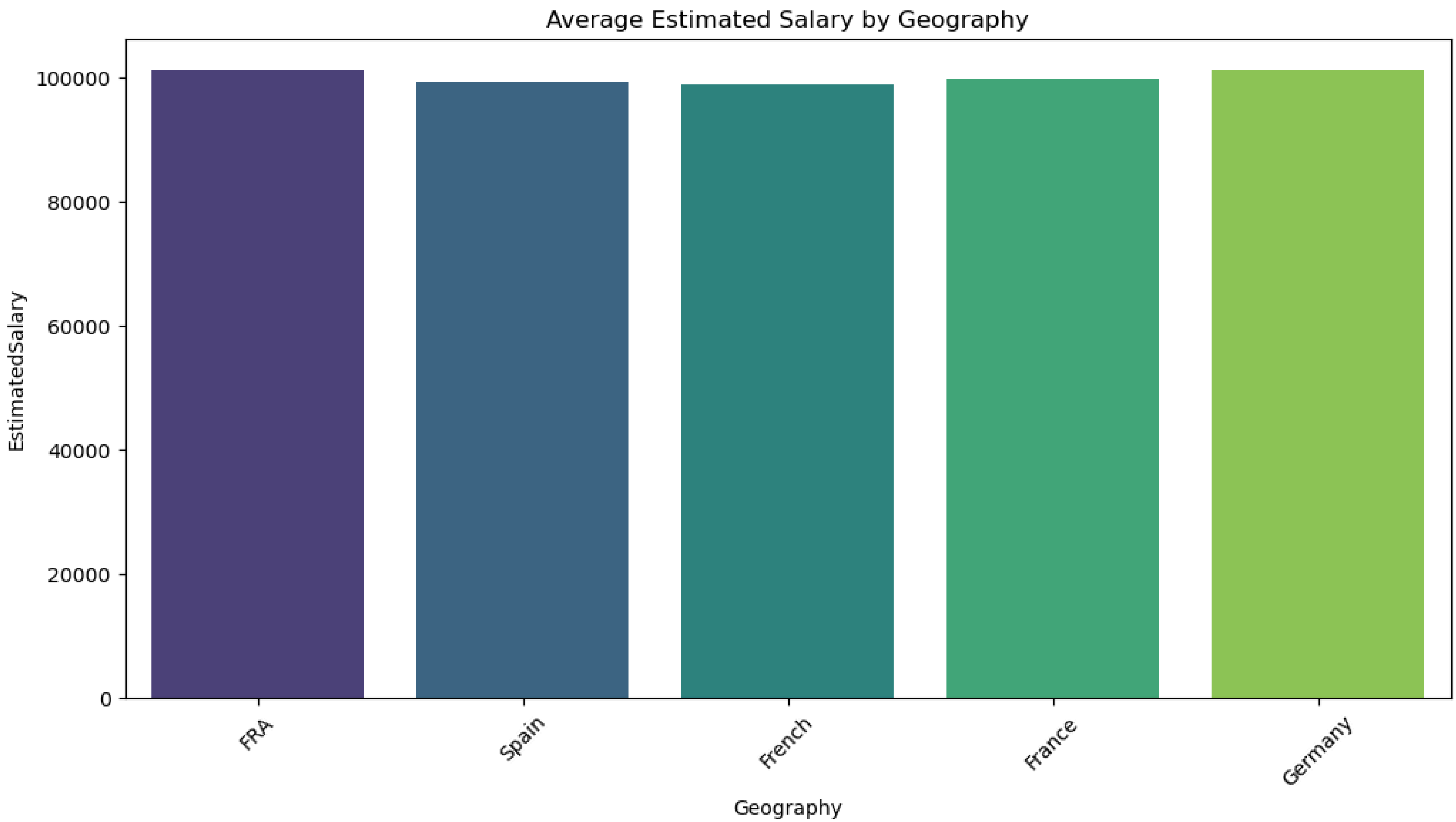
```
C:\Users\Henry Morgan\AppData\Local\Temp\ipykernel_9420\1582143337.py:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='Geography', y='CreditScore', data=Bank_Churn, ci=None, palette='viridis')
```




```
C:\Users\Henry Morgan\AppData\Local\Temp\ipykernel_9420\1582143337.py:10: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='Geography', y='Age', data=Bank_Churn, ci=None, palette='viridis')
```



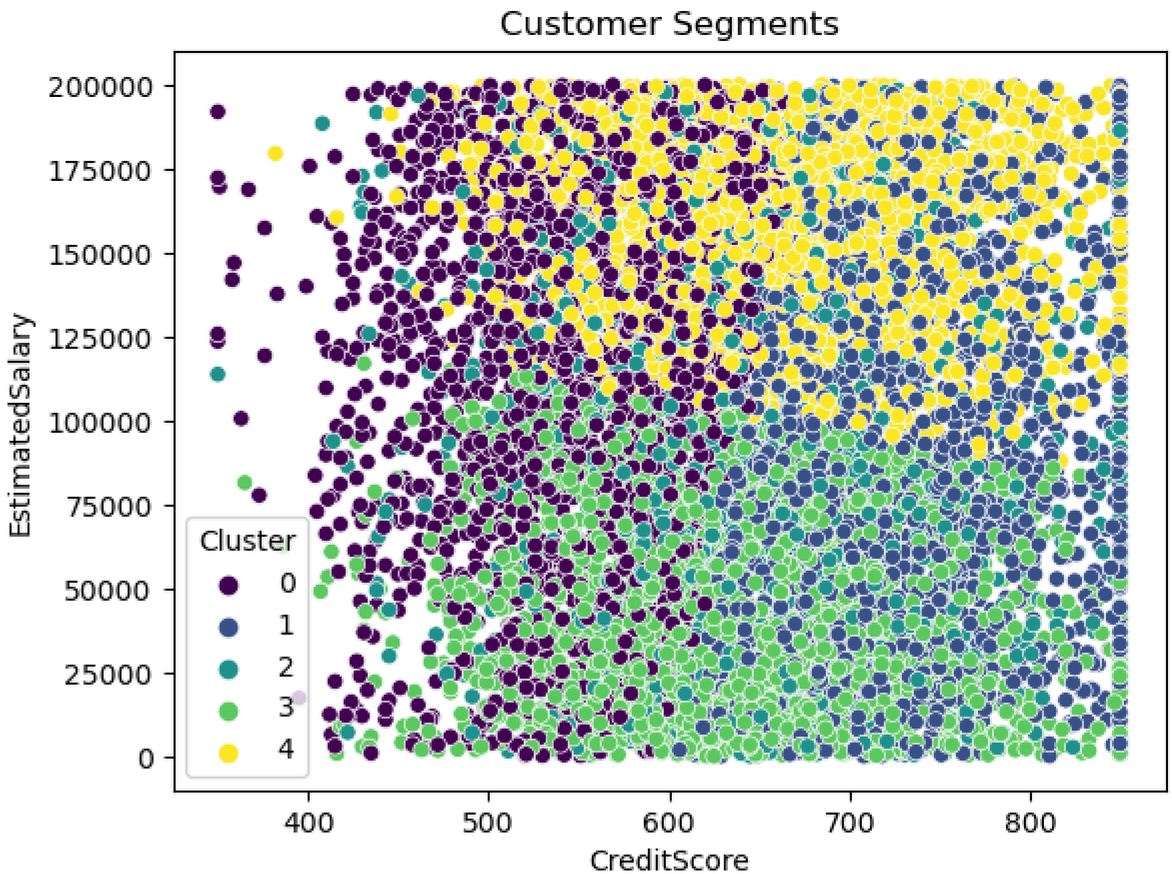

```
C:\Users\Henry Morgan\AppData\Local\Temp\ipykernel_9420\1582143337.py:17: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.barplot(x='Geography', y='EstimatedSalary', data=Bank_Churn, ci=None, palette='viridis')
```



In [12]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Select features for clustering
features = Bank_Churn[['CreditScore', 'Age', 'EstimatedSalary', 'Tenure']]

# Normalize the data
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=5, random_state=0)
```

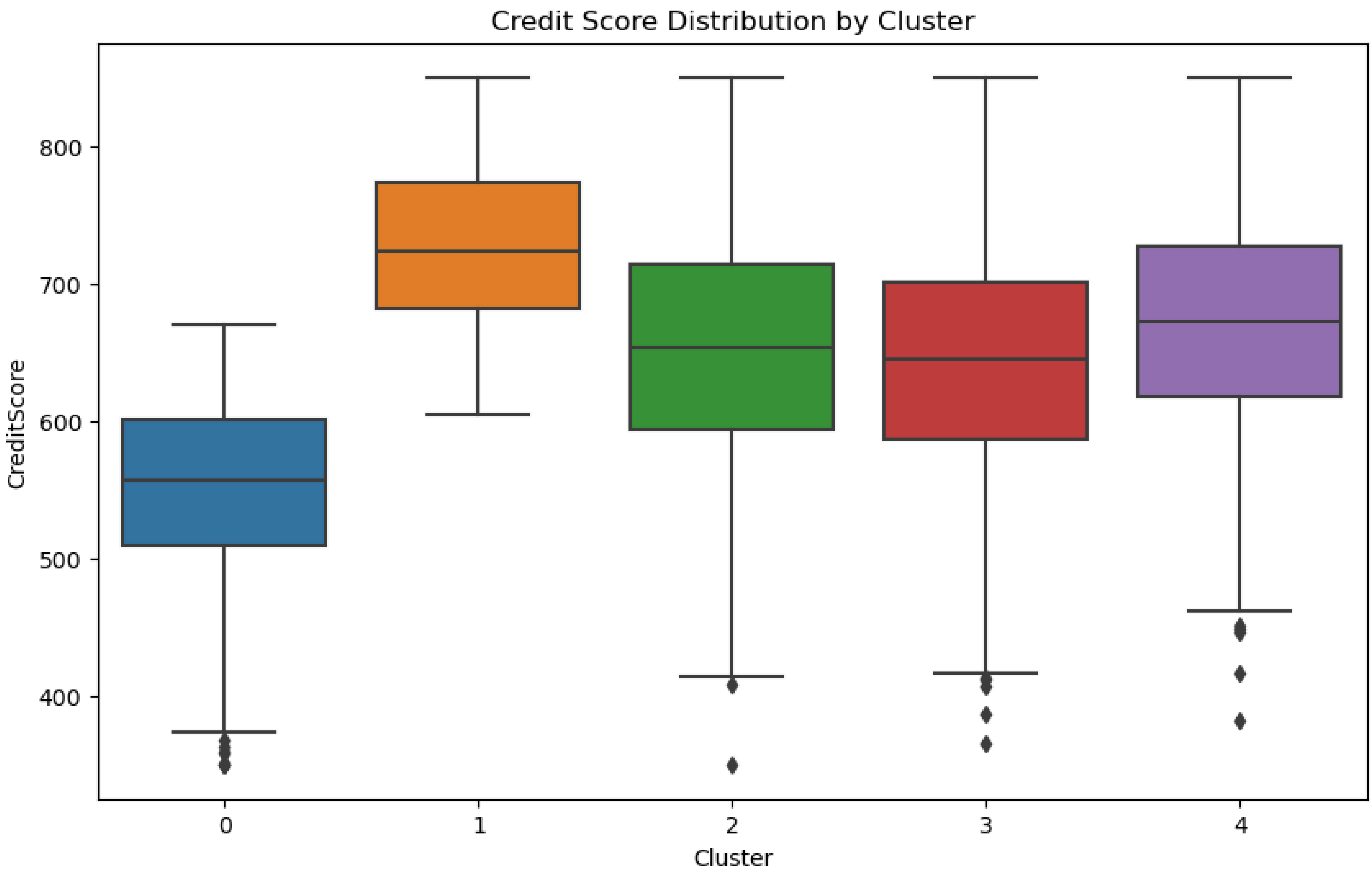
```python
Bank_Churn['Cluster'] = kmeans.fit_predict(scaled_features)

# Cluster analysis
cluster_summary = Bank_Churn.groupby('Cluster').agg({
    'CreditScore': ['mean', 'std'],
    'Age': ['mean', 'std'],
    'EstimatedSalary': ['mean', 'std'],
    'Tenure': ['mean', 'std']
}).reset_index()

print(cluster_summary)

# Visualization
sns.scatterplot(x='CreditScore', y='EstimatedSalary', hue='Cluster', data=Bank_Churn, palette='viridis')
plt.title('Customer Segments')
plt.show()
```

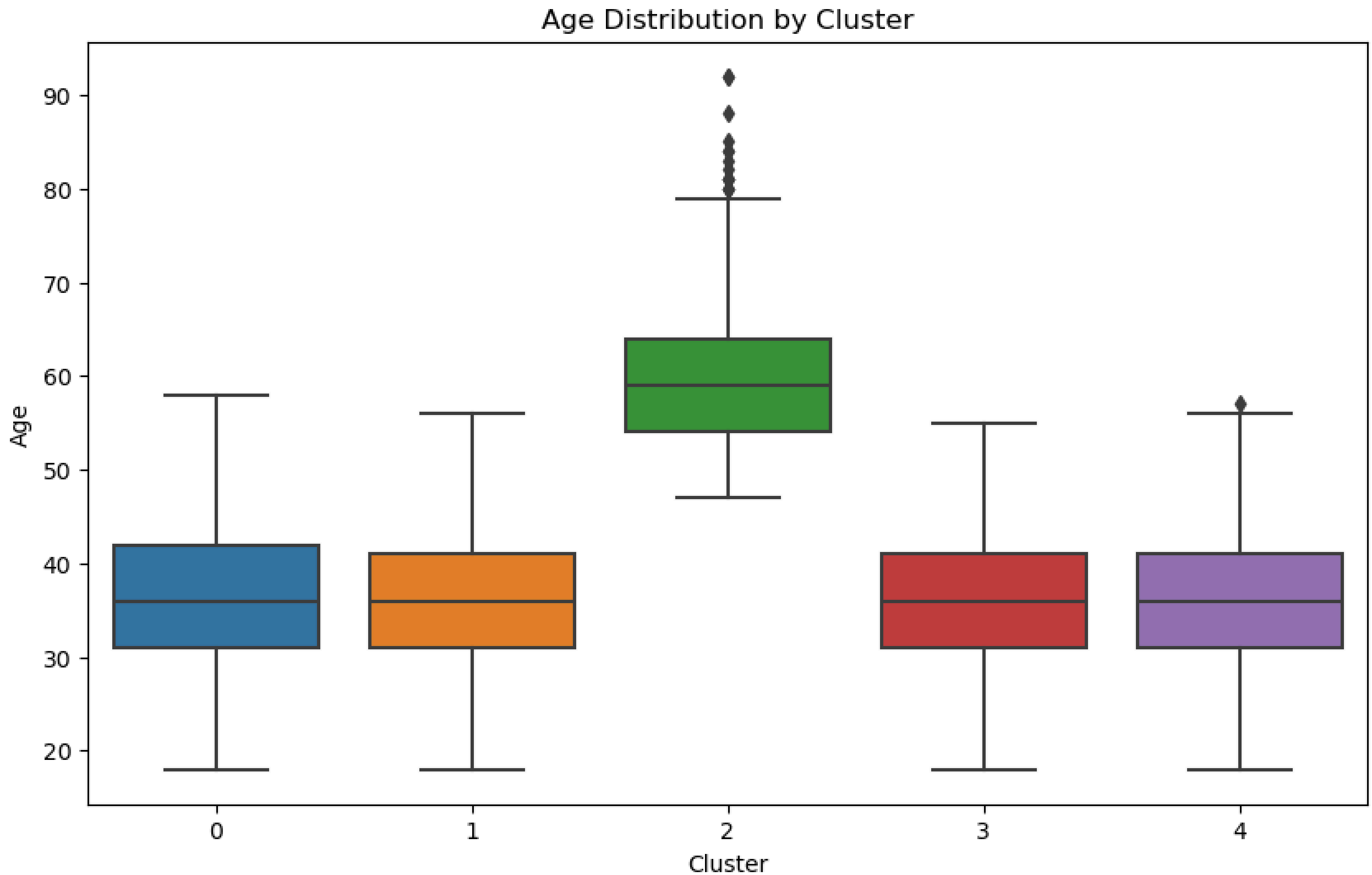
```
  Cluster CreditScore                  Age           EstimatedSalary  \
                 mean        std       mean       std            mean   
0       0  551.782629  61.336961  36.545540  7.321705   113911.952610   
1       1  729.851301  61.434557  35.799257  7.047467    90481.335804   
2       2  653.533220  91.656510  59.866269  7.597168    95463.164242   
3       3  644.706237  85.071329  36.056774  7.049020    45742.469114   
4       4  673.003159  82.484811  36.149368  7.094892   155617.626656   

                     Tenure           
            std        mean       std  
0  51744.931177  7.130986  1.906265  
1  51920.732820  7.693773  1.549313  
2  54151.240933  4.949744  2.725591  
3  29325.306187  2.744516  1.745444  
4  28530.136487  2.788357  1.774377  
```

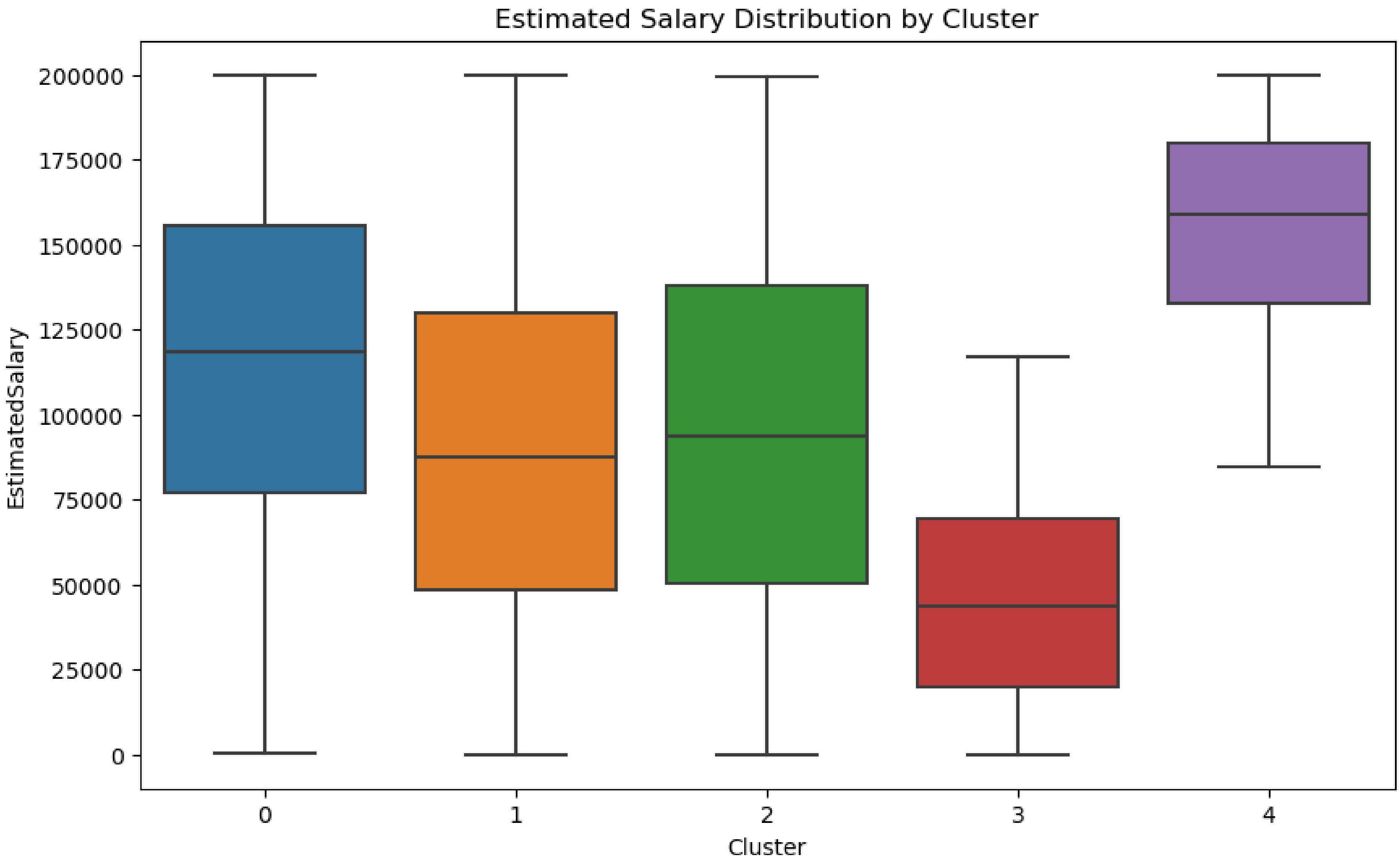


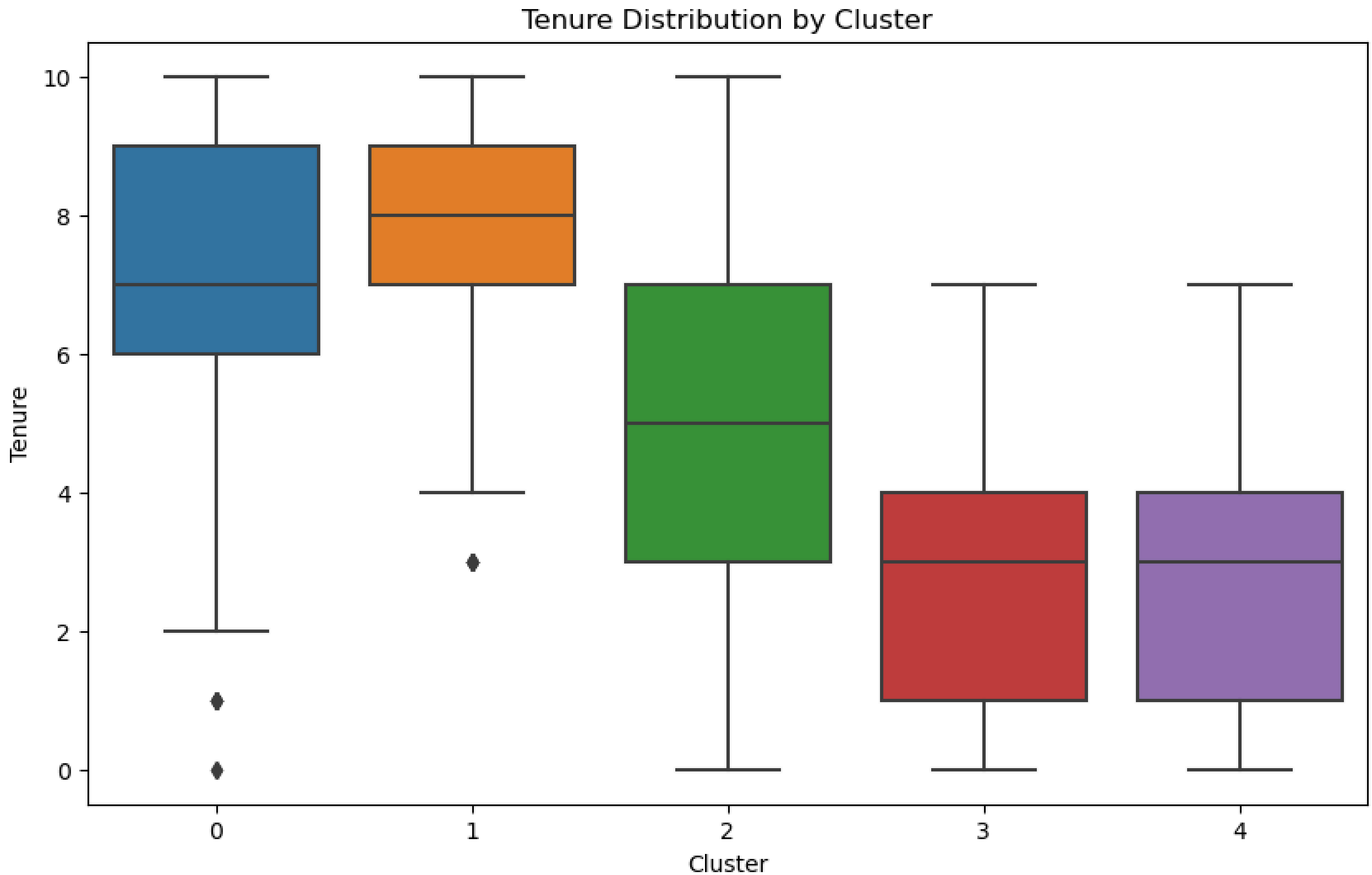
In [13]:
```
# Plotting CreditScore by Cluster
plt.figure(figsize=(10, 6))
sns.boxplot(x='Cluster', y='CreditScore', data=Bank_Churn)
plt.title('Credit Score Distribution by Cluster')
plt.show()

# Plotting Age by Cluster
plt.figure(figsize=(10, 6))
sns.boxplot(x='Cluster', y='Age', data=Bank_Churn)
plt.title('Age Distribution by Cluster')
plt.show()

# Plotting Estimated Salary by Cluster
plt.figure(figsize=(10, 6))
```

```
sns.boxplot(x='Cluster', y='EstimatedSalary', data=Bank_Churn)
plt.title('Estimated Salary Distribution by Cluster')
plt.show()

# Plotting Tenure by Cluster
plt.figure(figsize=(10, 6))
sns.boxplot(x='Cluster', y='Tenure', data=Bank_Churn)
plt.title('Tenure Distribution by Cluster')
plt.show()
```




Age Distribution by Cluster
Age
90
80
70
60
50
40
30
20
0
1
2
3
4
Cluster


Estimated Salary Distribution by Cluster
EstimatedSalary
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
2
3
4
Cluster



In [16]:
```
# Convert 'EstimatedSalary' to numeric and remove currency symbols
Bank_Churn['EstimatedSalary'] = Bank_Churn['EstimatedSalary'].replace({'€': ''}, regex=True).astype(float)

# Summary statistics for churned vs. non-churned customers
churned = Bank_Churn[Bank_Churn['Churned'] == 1]
non_churned = Bank_Churn[Bank_Churn['Churned'] == 0]

summary_churned = churned[['CreditScore', 'Age', 'Tenure', 'EstimatedSalary']].describe()
summary_non_churned = non_churned[['CreditScore', 'Age', 'Tenure', 'EstimatedSalary']].describe()
```

```
print("Churned Customers Summary:")
print(summary_churned)
print("\nNon-Churned Customers Summary:")
print(summary_non_churned)
```

```
Churned Customers Summary:
       CreditScore          Age      Tenure  EstimatedSalary
count  4998.000000  4998.000000  4998.00000      4998.000000
mean    650.686475    38.787315     5.02501    100522.598219
std      96.370435    10.447913     2.87947     57584.432297
min     350.000000    18.000000     0.00000       106.670000
25%     584.000000    32.000000     3.00000     50667.922500
50%     652.000000    37.000000     5.00000    100770.695000
75%     718.000000    44.000000     7.00000    150084.290000
max     850.000000    92.000000    10.00000    199953.330000

Non-Churned Customers Summary:
       CreditScore          Age       Tenure  EstimatedSalary
count  4999.000000  4999.000000  4999.000000      4999.000000
mean    650.404281    39.056811     5.001400     99661.933185
std      96.953954    10.529384     2.905439     57455.594359
min     350.000000    18.000000     0.000000        11.580000
25%     583.000000    32.000000     2.000000     51315.440000
50%     652.000000    37.000000     5.000000     99504.030000
75%     717.000000    44.000000     8.000000    147959.490000
max     850.000000    88.000000    10.000000    199992.480000
```

In [17]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Prepare features and target variable
X = Bank_Churn[['CreditScore', 'Age', 'Tenure', 'EstimatedSalary']]
y = Bank_Churn['Churned']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and fit the model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[859 618]
 [882 641]]
              precision    recall  f1-score   support

           0       0.49      0.58      0.53      1477
           1       0.51      0.42      0.46      1523

    accuracy                           0.50      3000
   macro avg       0.50      0.50      0.50      3000
weighted avg       0.50      0.50      0.50      3000
```

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler

# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Initialize GridSearchCV with RandomForestClassifier
grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=param_grid, cv=5)

# Fit the model
grid_search.fit(X_train, y_train)

# Print the best parameters found
print("Best parameters found: ", grid_search.best_params_)
```

```
Best parameters found:  {'max_depth': None, 'min_samples_split': 10, 'n_estimators': 200}
```

In [22]:
```
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Define parameter distributions for RandomizedSearchCV
param_distributions = {
    'n_estimators': randint(50, 200),
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': randint(2, 10)
}

# Initialize RandomizedSearchCV with RandomForestClassifier
random_search = RandomizedSearchCV(
    estimator=RandomForestClassifier(),
    param_distributions=param_distributions,
    n_iter=100,
    cv=5
)

# Fit the model
random_search.fit(X_train, y_train)

# Print the best parameters found
print("Best parameters found: ", random_search.best_params_)
```

```
Best parameters found:  {'max_depth': None, 'min_samples_split': 6, 'n_estimators': 95}
```

In [23]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Choose the best parameters from your runs (for example, the first run's parameters)
best_params = {'max_depth': None, 'min_samples_split': 10, 'n_estimators': 200}

# Train the model with the best parameters
model = RandomForestClassifier(**best_params)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
```

```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[767 710]
 [783 740]]

Classification Report:
               precision    recall  f1-score   support

           0       0.49      0.52      0.51      1477
           1       0.51      0.49      0.50      1523

    accuracy                           0.50      3000
   macro avg       0.50      0.50      0.50      3000
weighted avg       0.50      0.50      0.50      3000
```

In [26]:

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

# Choose the best parameters from your runs (for example, the first run's parameters)
best_params = {'max_depth': None, 'min_samples_split': 6, 'n_estimators': 95}

# Train the model with the best parameters
model = RandomForestClassifier(**best_params)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[767 710]
 [755 768]]

Classification Report:
               precision    recall  f1-score   support

           0       0.50      0.52      0.51      1477
           1       0.52      0.50      0.51      1523

    accuracy                           0.51      3000
   macro avg       0.51      0.51      0.51      3000
weighted avg       0.51      0.51      0.51      3000
```

In [27]:

```
# Save the cleaned dataset to a new Excel file
Bank_Churn.to_excel("Bank_Churn_Cleaned.xlsx", index=False)
```

In [28]:

```
Bank_Churn
```

Out[28]:

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	EstimatedSalary	Churned	Cluster
0	15634602	Hargrave	619	FRA	Female	42.0	2	101348.88	0	3
1	15647311	Hill	608	Spain	Female	41.0	1	112542.58	1	4
2	15619304	Onio	502	French	Female	42.0	8	113931.57	0	0
3	15701354	Boni	699	FRA	Female	39.0	1	93826.63	1	3
4	15737888	Mitchell	850	Spain	Female	43.0	2	79084.10	0	3
...	...	...	...	...	...	...	...	...	...	...
9995	15606229	Obijiaku	771	France	Male	39.0	5	96270.64	0	1
9996	15569892	Johnstone	516	French	Male	35.0	10	101699.77	1	0
9997	15584532	Liu	709	FRA	Female	36.0	7	42085.58	0	1
9998	15682355	Sabbatini	772	Germany	Male	42.0	3	92888.52	1	4
9999	15628319	Walker	792	French	Female	28.0	4	38190.78	0	3

9997 rows × 10 columns

In [ ]: