Hello! Welcome to my Notebook 💗

🔗 Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,recall_score ,a
```

📌 Reading Data

In [2]:
```
df = pd.read_csv('/kaggle/input/bank-customer-churn-dataset/Bank Customer Churn Pre
df.head()
```

Out[2]:

	customer_id	credit_score	country	gender	age	tenure	balance	products_number
0	15634602	619	France	Female	42	2	0.00	1
1	15647311	608	Spain	Female	41	1	83807.86	1
2	15619304	502	France	Female	42	8	159660.80	3
3	15701354	699	France	Female	39	1	0.00	2
4	15737888	850	Spain	Female	43	2	125510.82	1

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customer_id       10000 non-null  int64
 1   credit_score      10000 non-null  int64
 2   country           10000 non-null  object
 3   gender            10000 non-null  object
 4   age               10000 non-null  int64
 5   tenure            10000 non-null  int64
 6   balance           10000 non-null  float64
 7   products_number   10000 non-null  int64
 8   credit_card       10000 non-null  int64
 9   active_member     10000 non-null  int64
 10  estimated_salary  10000 non-null  float64
 11  churn             10000 non-null  int64
dtypes: float64(2), int64(8), object(2)
memory usage: 937.6+ KB
```

In [4]:

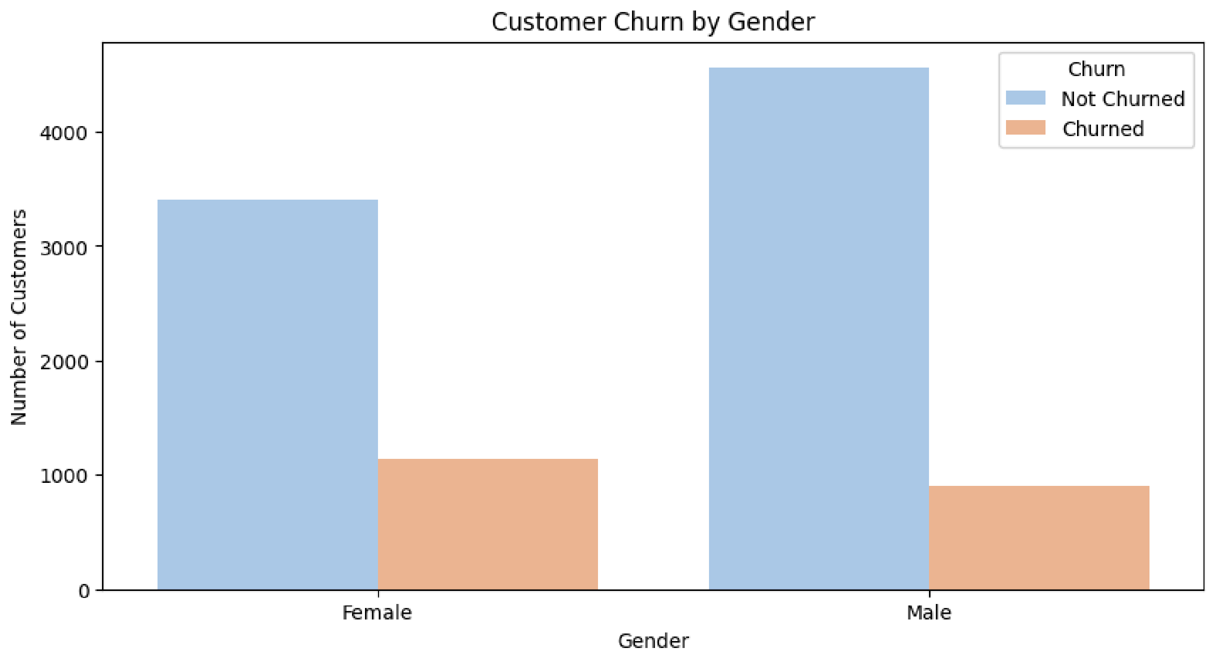
```
df.describe()
```

Out[4]:

	customer_id	credit_score	age	tenure	balance	products_nu
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.0
mean	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.5
std	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.5
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.0
25%	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.0
50%	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.0
75%	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.0
max	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.0

📌 EAD

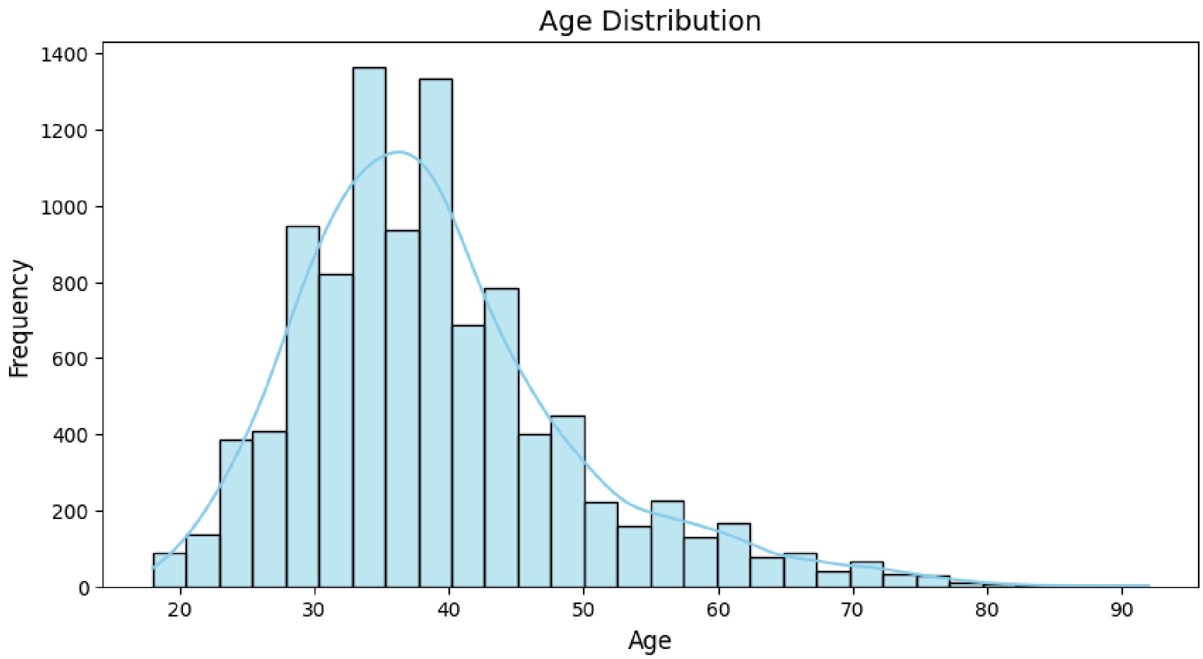
In [5]:
```
plt.figure(figsize=(10, 5))
sns.countplot(x='gender', hue='churn', data=df, palette='pastel')
plt.title('Customer Churn by Gender')
plt.xlabel('Gender')
plt.ylabel('Number of Customers')
plt.legend(title='Churn', labels=['Not Churned', 'Churned'])
plt.show()
```



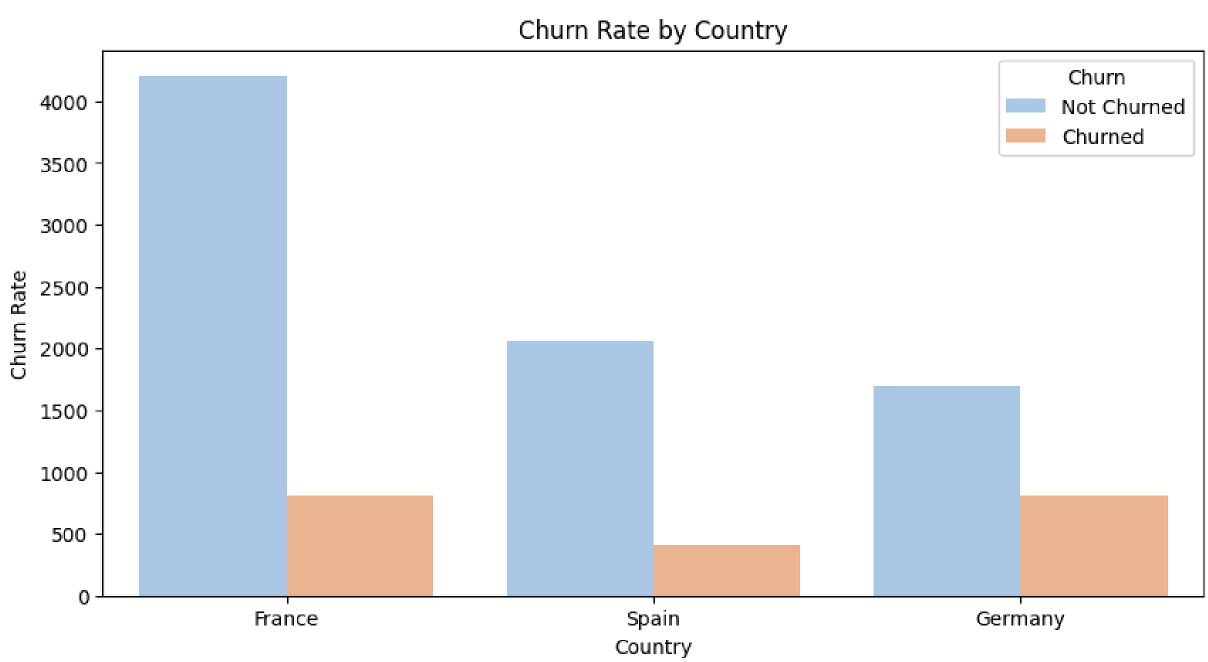
Females have a higher churn rate than males.

In [6]:
```
# Plotting the age distribution
plt.figure(figsize=(10, 5))
sns.histplot(df['age'], bins=30, kde=True, color='skyblue')
plt.title('Age Distribution', fontsize=14)
plt.xlabel('Age', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use
_inf_as_na option is deprecated 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):
```



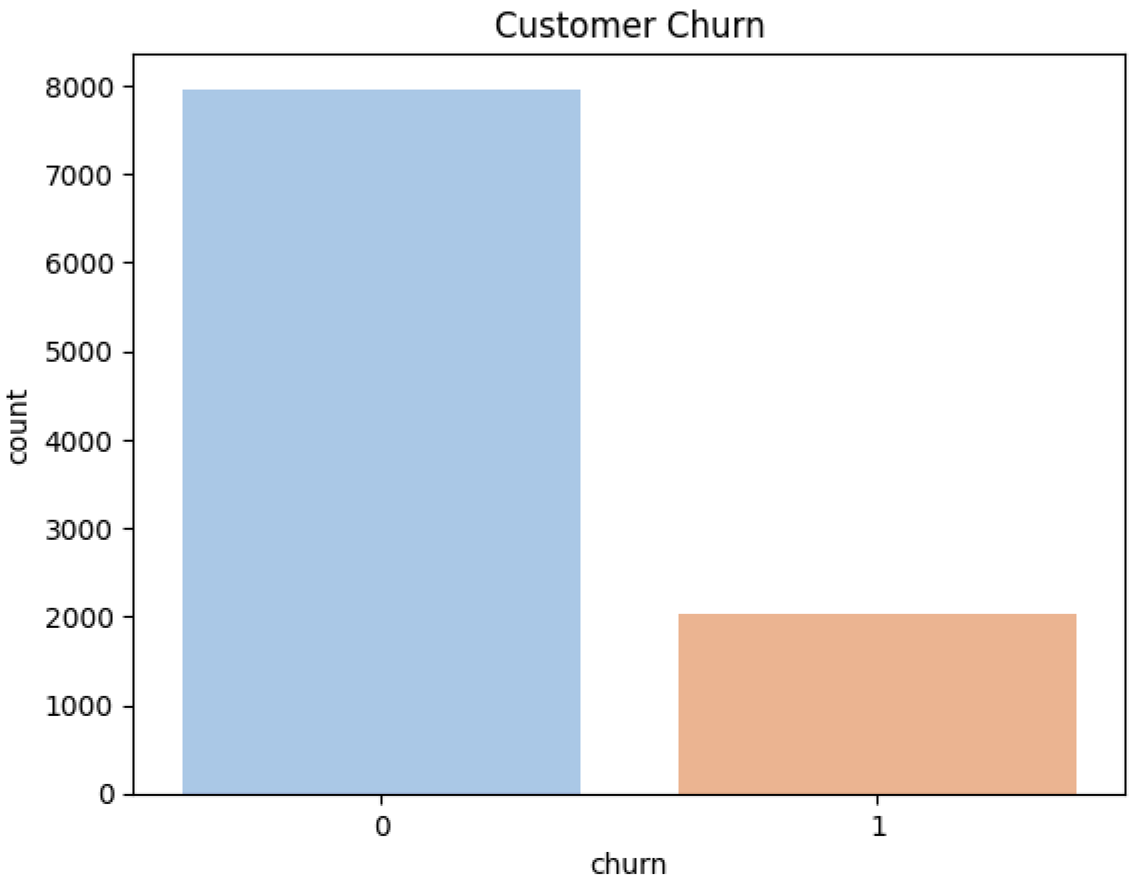
In [7]:
```
# Calculating churn rate by country
churn_rate_country = df.groupby('country')['churn'].mean().reset_index()
plt.figure(figsize=(10, 5))
sns.countplot(x='country', hue='churn', data=df, palette='pastel')
plt.title('Churn Rate by Country')
plt.xlabel('Country')
plt.ylabel('Churn Rate')
plt.legend(title='Churn', labels=['Not Churned', 'Churned'])
plt.show()
```



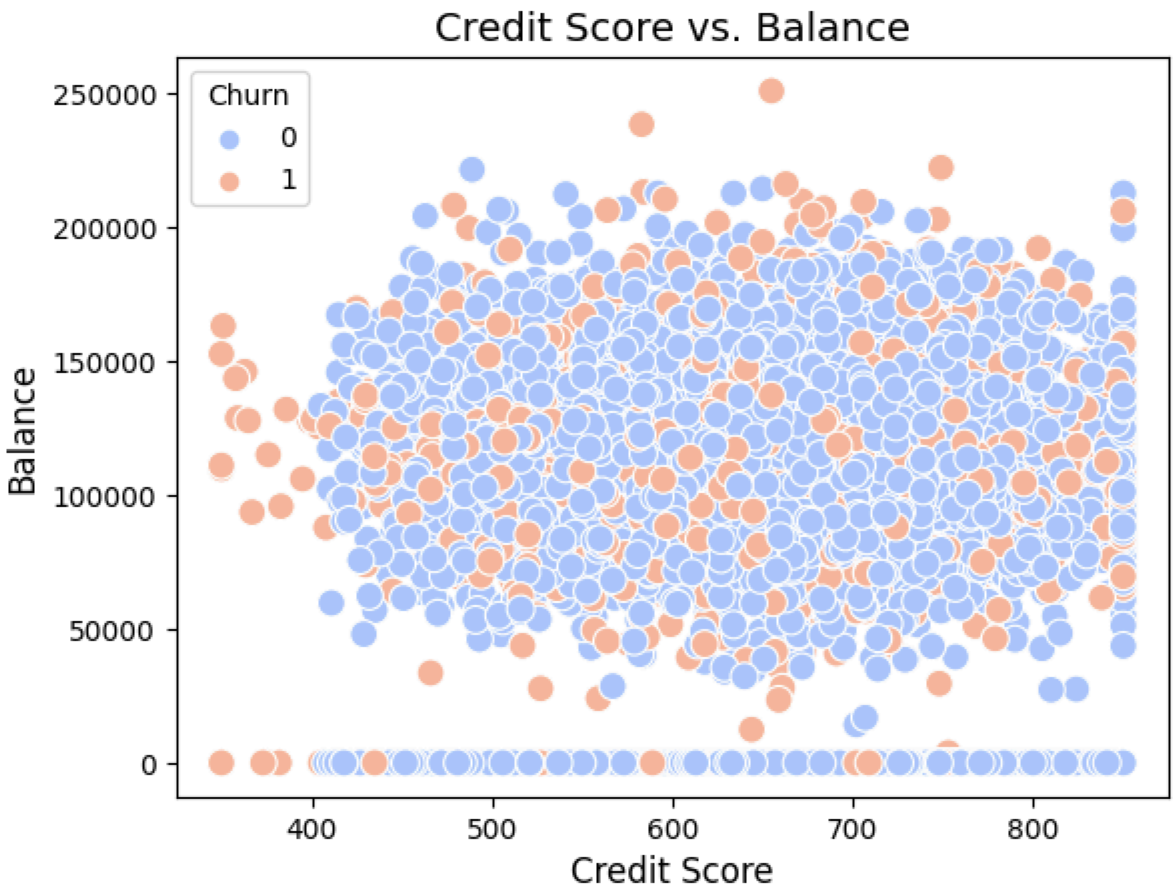
France and Germany have a higher churn rate.

In [8]:
```
plt.figure()
sns.countplot(x='churn', data=df, palette='pastel')
```

```
plt.title('Customer Churn')
plt.show()
```

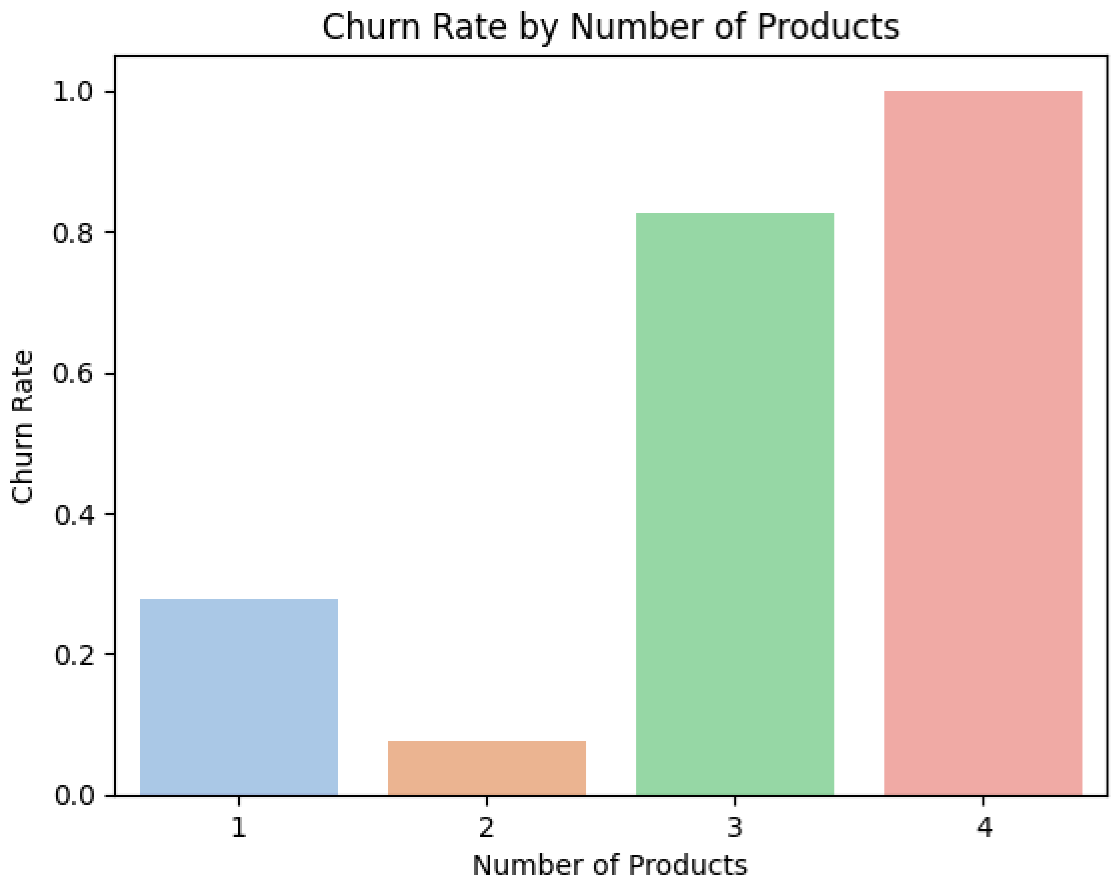


In [9]:
```
# Plotting the scatter plot for Credit Score vs. Balance
plt.figure()
sns.scatterplot(data=df, x='credit_score', y='balance', hue='churn', palette='coolw
plt.title('Credit Score vs. Balance', fontsize=14)
plt.xlabel('Credit Score', fontsize=12)
plt.ylabel('Balance', fontsize=12)
plt.legend(title='Churn')
plt.show()
```

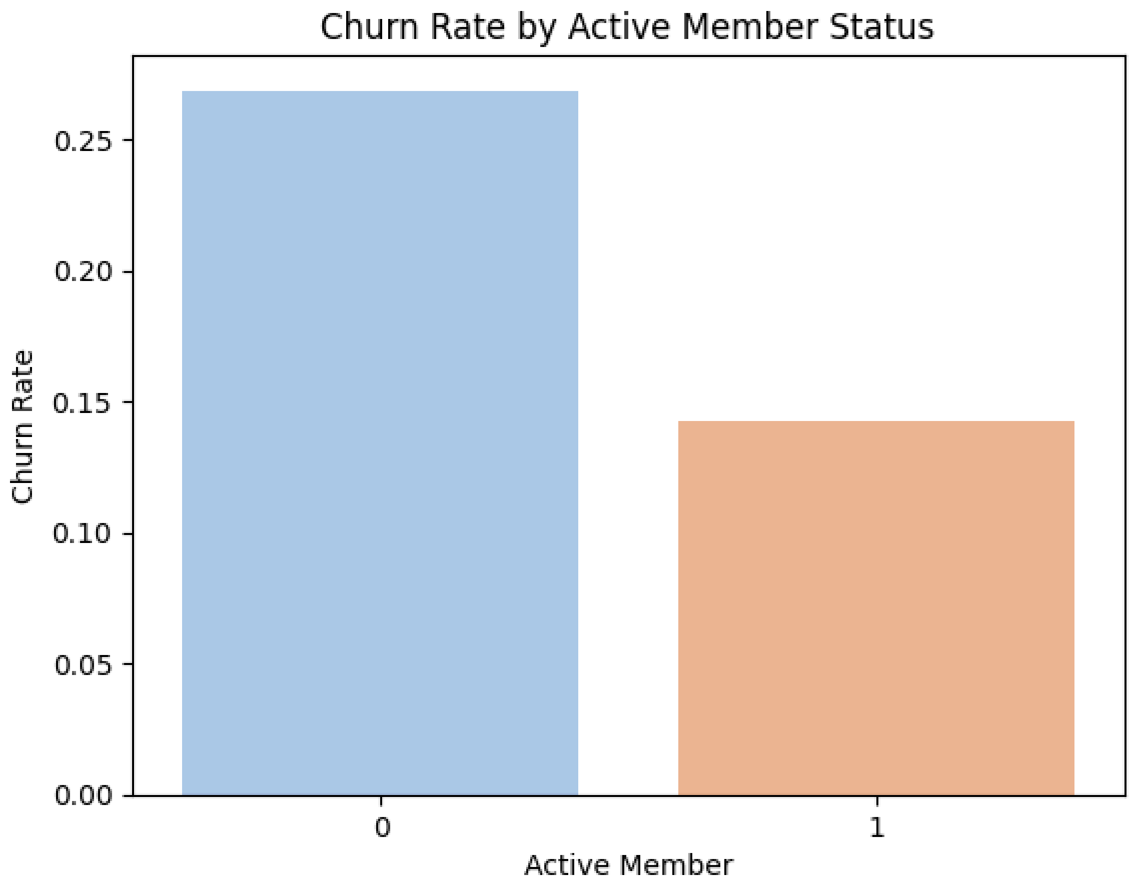


In [10]:

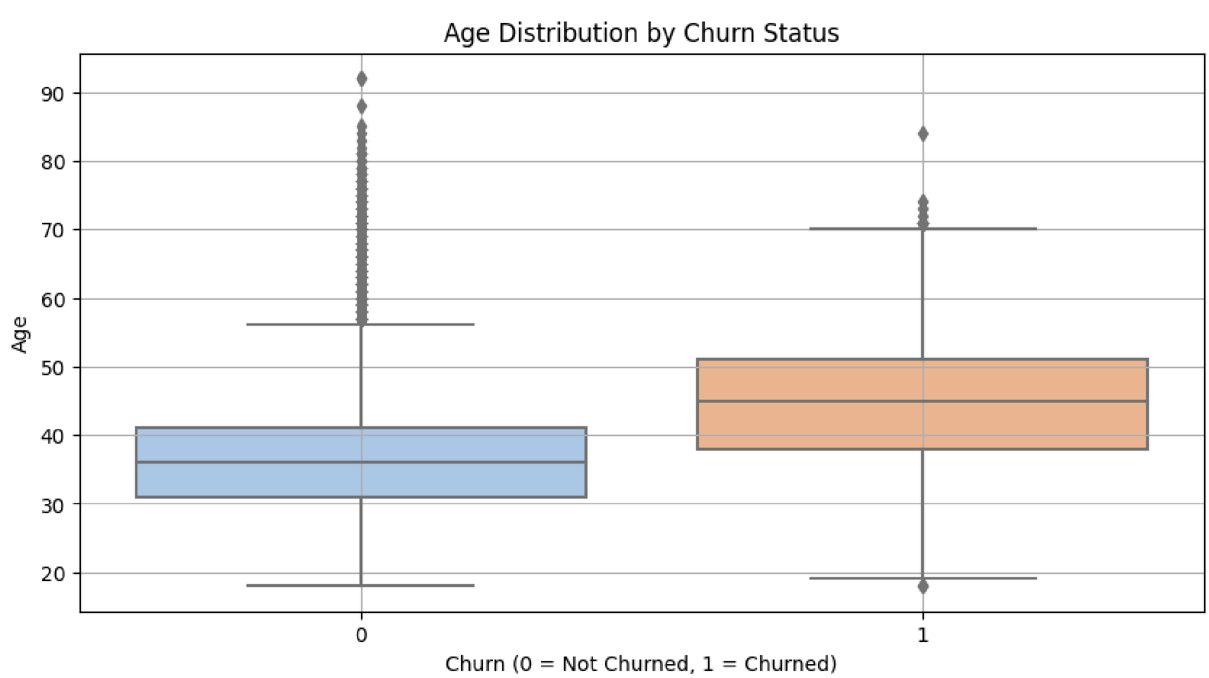
```
churn_rate_by_products = df.groupby('products_number')['churn'].mean().reset_index(

# Create the bar plot
plt.figure()
sns.barplot(x='products_number', y='churn', data=churn_rate_by_products, palette='p

# Set plot labels and title
plt.xlabel('Number of Products')
plt.ylabel('Churn Rate')
plt.title('Churn Rate by Number of Products')

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



In [11]:
```
churn_rate = df.groupby('active_member')['churn'].mean().reset_index()

# Create a bar plot using seaborn
sns.barplot(x='active_member', y='churn', data=churn_rate,palette='pastel')
plt.xlabel('Active Member')
plt.ylabel('Churn Rate')
plt.title('Churn Rate by Active Member Status')
plt.show()
```

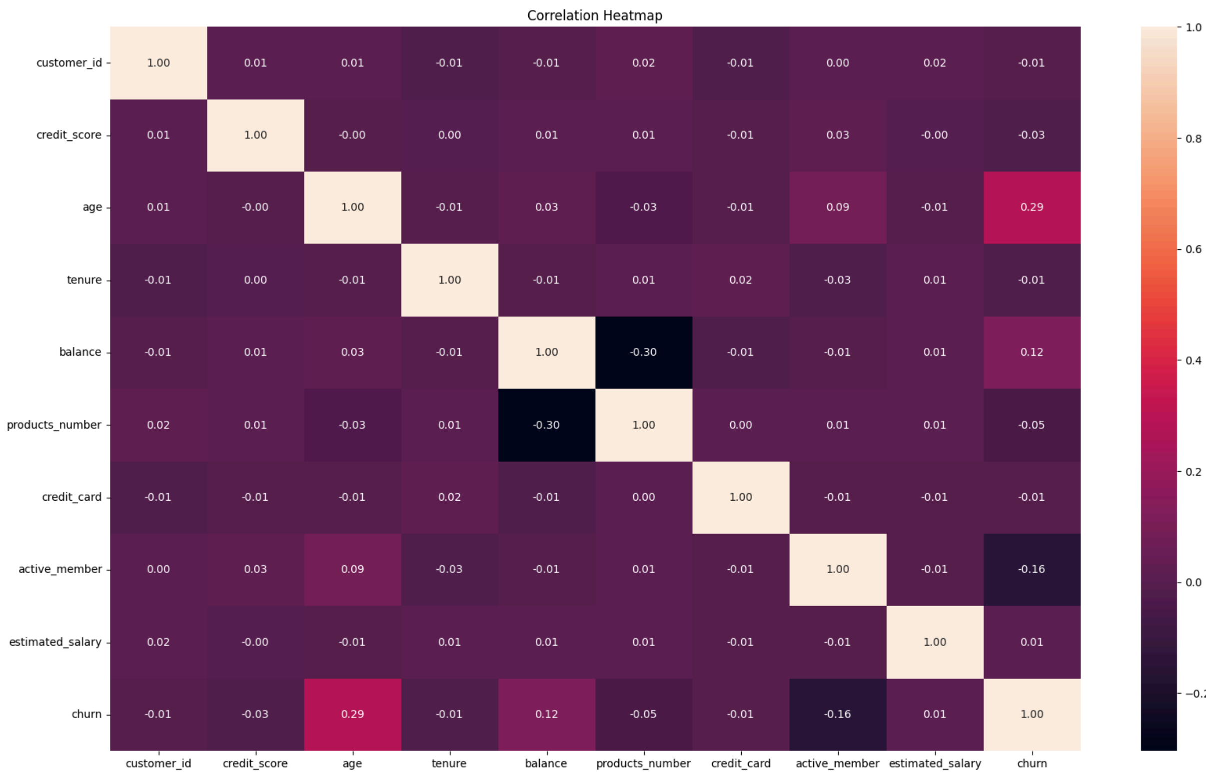


In [12]:
```
plt.figure(figsize=(10, 5))
sns.boxplot(x='churn', y='age', data=df, palette='pastel')
plt.title('Age Distribution by Churn Status')
plt.xlabel('Churn (0 = Not Churned, 1 = Churned)')
plt.ylabel('Age')
plt.grid()
plt.show()
```



Correlation between Features

In [13]:
```python
numeric_df = df.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()
plt.figure(figsize=(20, 12))
# Create a heatmap using seaborn
sns.heatmap(correlation_matrix, annot=True, fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



📌 Data Preprocessing

Encoding categorical features

In [14]:
```python
label_encoder = LabelEncoder()
df['country'] = label_encoder.fit_transform(df['country'])
df['gender'] = label_encoder.fit_transform(df['gender'])
```

Removing the customer_id column because it has no impact on the data.

In [15]:
```python
df.drop(columns=['customer_id'], inplace=True)
df.columns
```

Out[15]:
```
Index(['credit_score', 'country', 'gender', 'age', 'tenure', 'balance',
       'products_number', 'credit_card', 'active_member', 'estimated_salary',
       'churn'],
      dtype='object')
```

📌 Model Building and Training

In [16]:
```
X = df.drop('churn', axis=1)
y = df['churn']
```

Splitting our data

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [18]:
```
print(f'X_train shape: {X_train.shape}')
print(f'y_train shape: {y_train.shape}')
print(f'X_test shape: {X_test.shape}')
print(f'y_test shape: {y_test.shape}')
```

```
X_train shape: (8000, 10)
y_train shape: (8000,)
X_test shape: (2000, 10)
y_test shape: (2000,)
```

Scaling the data using Standard Scaler

In [19]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [20]:
```
X_train
```

Out[20]:
```
array([[ 0.35649971, -0.9055496 ,  0.91324755, ...,  0.64920267,
         0.97481699,  1.36766974],
       [-0.20389777,  0.30164867,  0.91324755, ...,  0.64920267,
         0.97481699,  1.6612541 ],
       [-0.96147213,  1.50884694,  0.91324755, ...,  0.64920267,
        -1.02583358, -0.25280688],
       ...,
       [ 0.86500853, -0.9055496 , -1.09499335, ..., -1.54035103,
        -1.02583358, -0.1427649 ],
       [ 0.15932282, -0.9055496 ,  0.91324755, ...,  0.64920267,
        -1.02583358, -0.05082558],
       [ 0.47065475,  0.30164867,  0.91324755, ...,  0.64920267,
         0.97481699, -0.81456811]])
```

Make List to append the result of each model in it

In [21]:
```
final=[]
```

Evaluation Function

In [22]:
```
def evaluate_model(y_test, y_pred):
    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
    print(f"Precision: {precision_score(y_test, y_pred)}")
    print(f"Recall: {recall_score(y_test, y_pred)}")
    cm = confusion_matrix(y_test, y_pred)
    sensitivity = cm[1, 1] / (cm[1, 1] + cm[1, 0])
    specificity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
    print(f"Sensitivity: {sensitivity}")
    print(f"Specificity: {specificity}")
    print(f"ROC AUC: {roc_auc_score(y_test, y_pred)}")
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.show()
```

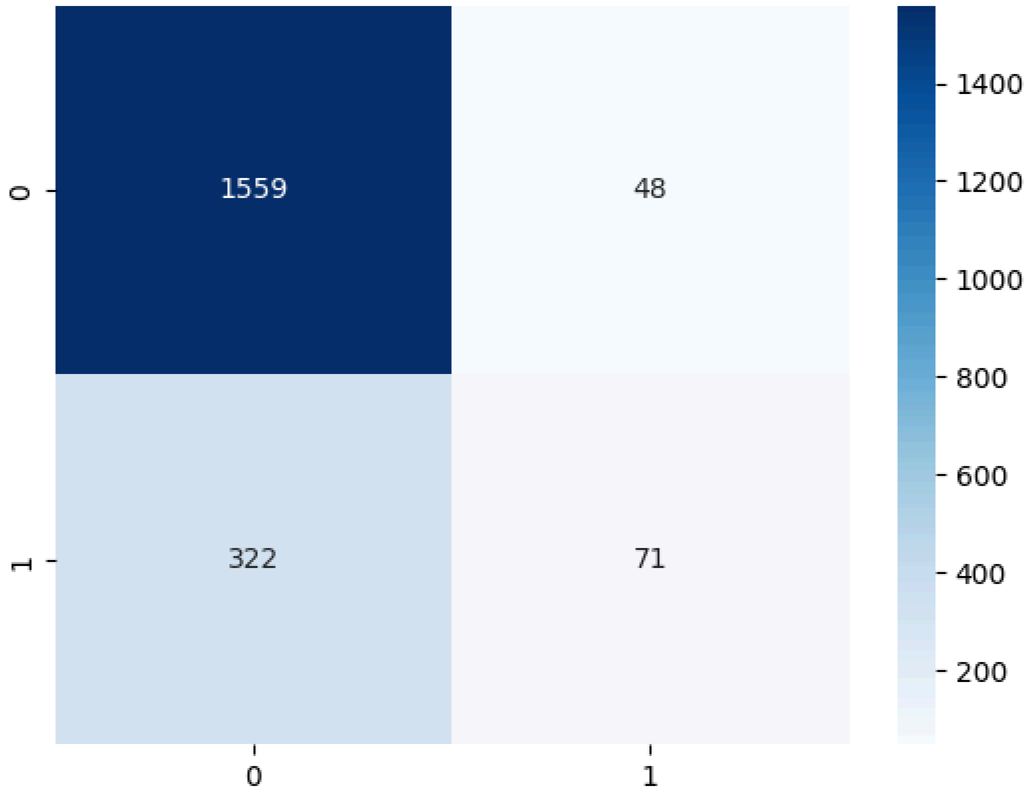
📌 Logistic Regression

In [23]:
```
from sklearn.linear_model import LogisticRegression
LR= LogisticRegression()
```

In [24]:
```
# Train the model
LR.fit(X_train, y_train)
# Make predictions
y_pred_LR = LR.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_LR)
final.append(accuracy)
```

In [25]:
```
evaluate_model(y_test,y_pred_LR)
```

```
Accuracy: 0.815
Precision: 0.5966386554621849
Recall: 0.1806615776081425
Sensitivity: 0.1806615776081425
Specificity: 0.970130678282514
ROC AUC: 0.5753961279453282
```



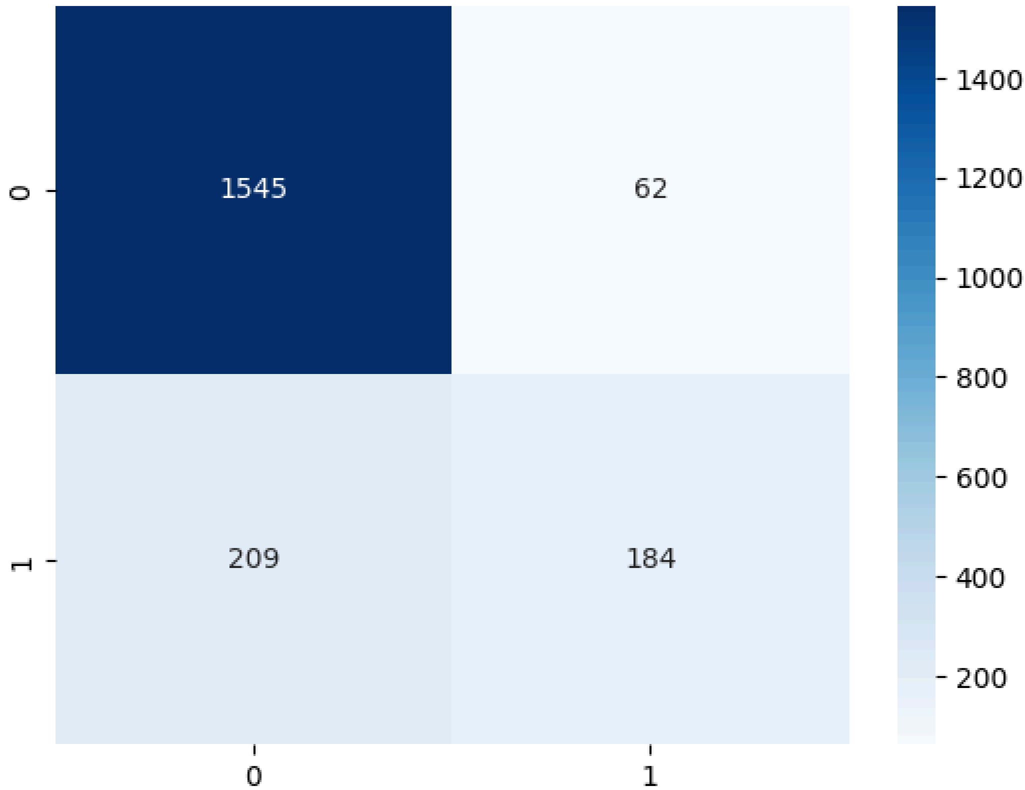
📌 Random Forest

In [26]:
```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [27]:
```
# Train the model
RF.fit(X_train, y_train)
# Make predictions
y_pred_rf = RF.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_rf)
final.append(accuracy)
```

In [28]:
```
evaluate_model(y_test,y_pred_rf)
```

```
Accuracy: 0.8645
Precision: 0.7479674796747967
Recall: 0.4681933842239186
Sensitivity: 0.4681933842239186
Specificity: 0.9614187927815806
ROC AUC: 0.7148060885027495
```



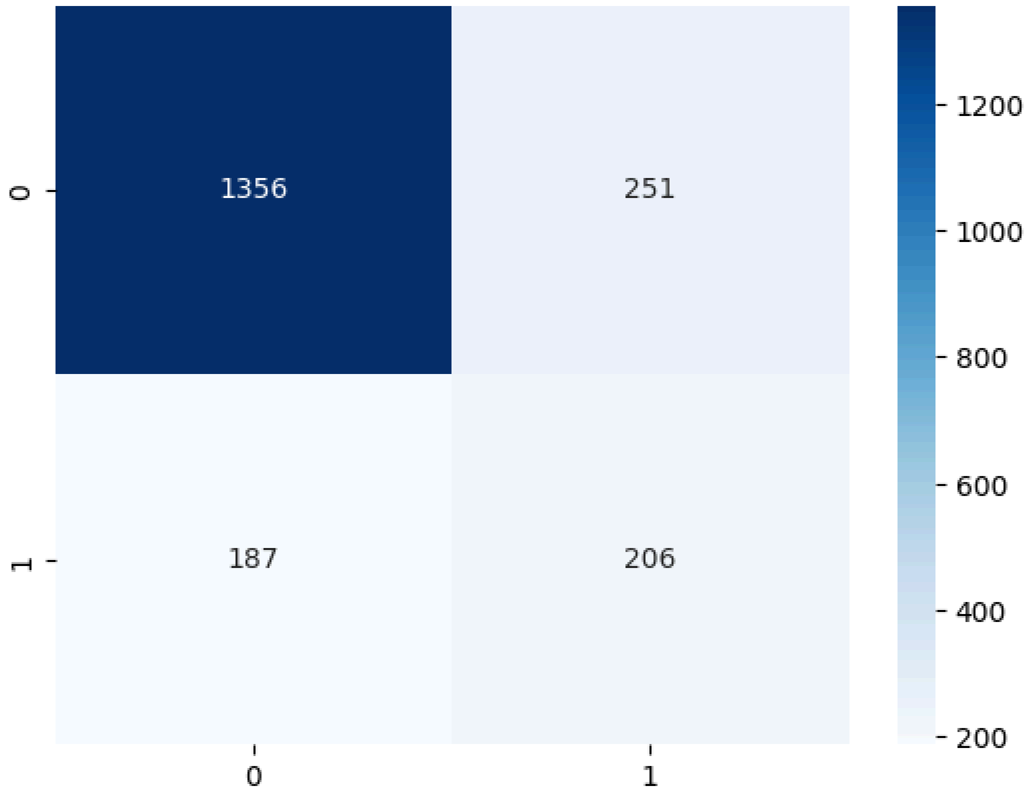
📌 Decision Tree

In [29]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=42)
```

In [30]:
```
# Train the model
dt.fit(X_train, y_train)
# Make predictions
y_pred_dt = dt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_dt)
final.append(accuracy)
```

In [31]:
```
evaluate_model(y_test,y_pred_dt)
```

```
Accuracy: 0.781
Precision: 0.45076586433260396
Recall: 0.5241730279898219
Sensitivity: 0.5241730279898219
Specificity: 0.8438083385189795
ROC AUC: 0.6839906832544006
```



In [32]:

```
final=np.array(final)
result=final.reshape(3,1)
columns=['Accuracy']
index=['Logistic Regression','Random Forest', 'Decision Tree']
final_result=pd.DataFrame(result,index=index,columns=columns)
```

In [33]:

```
final_result
```

Out[33]:

	Accuracy
Logistic Regression	0.8150
Random Forest	0.8645
Decision Tree	0.7810