Customer Churn Prediction

Student's Full Name: Manglam Srivastav (Teammate- Jasraj Singh Khalsa)

Course Title: INFO 531: Data Warehousing and Analytics in the Cloud

Term name and year: Spring 2025

Submission Week: Week 16 - Final Project Report

Instructor's Name: Nayem Rahman

Date of Submission: May 12, 2025

Project Description

The objective of this project is to predict customer churn for a banking institution using machine learning techniques. Churn refers to customers discontinuing their relationship with the bank. By identifying at-risk customers using their demographics and transactional behavior, banks can proactively implement retention strategies to reduce attrition and improve customer lifetime value. Our solution leverages classification models to predict the likelihood of churn based on customer data and evaluates their effectiveness using standard performance metrics.

1. Data Description and Source

We used the publicly available Bank Customer Churn Prediction dataset from Kaggle. The dataset contains 10,002 rows and 14 columns, including customer demographics, account details, and the churn label (Exited).

Target Variable: Exited (1 = customer churned, 0 = customer retained)

Key features include:

- **Demographics:** Geography, Gender, Age
- **Financials:** CreditScore, Balance, EstimatedSalary
- **Banking Activity:** Tenure, NumOfProducts, HasCrCard, IsActiveMember

Dataset Link: https://www.kaggle.com/datasets/shubhammeshram579/bank-customer-churn-prediction

2. Data Preparation

Data preparation is a very important step to make sure that the dataset is clean, consistent, and also suitable for machine learning algorithms to process and interpret from. This phase is used to handle missing data, categorical variable encoding, feature scaling, and outlier handling. As part of data preparation, we did the following:

2.1. Data Loading and Cleaning

We imported essential libraries for data manipulation (pandas, numpy), visualization (matplotlib, seaborn), and model training and evaluation (sklearn).

Code:

```python
# ----------------------------------------
# 📦 1. Import Libraries
# ----------------------------------------
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    classification_report, confusion_matrix, accuracy_score,
    roc_auc_score, roc_curve
)

import warnings
warnings.filterwarnings('ignore')
```

✓ 12.0s

2.2. Load and Inspect Dataset

We loaded the dataset and reviewed the first few records. It contains 10,002 rows and 14 columns.

Code: We can see the head and data structure from the screenshot below.

```python
# ----------------------------------------
# 📁 2. Load and Inspect the Dataset
# ----------------------------------------
df = pd.read_csv('Churn_Modelling.csv')
print(df.shape)
df.head()
```

Python

(10002, 14)

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42.0	2	0.00	1	1.0	1.0	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41.0	1	83807.86	1	0.0	1.0	112542.58	0
2	3	15619304	Onio	502	France	Female	42.0	8	159660.80	3	1.0	0.0	113931.57	1
3	4	15701354	Boni	699	France	Female	39.0	1	0.00	2	0.0	0.0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43.0	2	125510.82	1	NaN	1.0	79084.10	0

2.3. Data Overview

We used these functions to understand data types, summary statistics, and missing values. Four rows contained missing values and were removed for simplicity.

Code:

```python
# ----------------------------------------
# 🔍 3. Data Overview
# ----------------------------------------
df.info()
df.describe()
df.isnull().sum()
```

✓ 0.0s

```
<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
```

```
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
```

2.4. Data Cleaning

We removed rows with null values and dropped non-informative columns like RowNumber, CustomerId, and Surname to reduce noise.

<u>Code:</u>

```python
# ----------------------------------------
# 🧹 4. Data Cleaning
# ----------------------------------------
df.dropna(inplace=True)  # Remove rows with missing values
df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1, inplace=True)  # Drop non-predictive columns
```

2.5. Encoding Categorical Features

Gender was label-encoded (Female = 0, Male = 1), and Geography was one-hot encoded to Geography_Germany and Geography_Spain.

<u>Code:</u>

```python
# ----------------------------------------
# 🔄 5. Encoding & Feature Engineering
# ----------------------------------------
# Encode Gender
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])

# One-hot encode Geography
df = pd.get_dummies(df, columns=['Geography'], drop_first=True)
```

2.6. Feature Scaling

Numeric columns were standardized to ensure equal weight in model learning.

<u>Code:</u>

```python
# ----------------------------------------
# 📏 6. Feature Scaling
# ----------------------------------------
scaler = StandardScaler()
scale_cols = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']
df[scale_cols] = scaler.fit_transform(df[scale_cols])

print("Data after Scaled Features:")
df[scale_cols].head()
```
✓ 0.0s

From the executed code, we get the following output:

```
Data after Scaled Features:
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary
0	-0.326298	0.293657	-1.041838	-1.225860	-0.911570	0.021720
1	-0.440137	0.198305	-1.387619	0.117428	-0.911570	0.216366
2	-1.537125	0.293657	1.032846	1.333214	2.526981	0.240519
3	0.501618	0.007601	-1.387619	-1.225860	0.807705	-0.109083
5	-0.057226	0.484361	1.032846	0.597439	0.807705	0.863478

3. Predictor and Target Variables

- **Target Variable**: Exited (1 = churned, 0 = stayed)
- **Predictor Variables**:
    - **Demographics**: Gender, Age, Geography_Germany, Geography_Spain
    - **Financials**: CreditScore, Balance, EstimatedSalary
    - **Banking Behavior**: Tenure, NumOfProducts, HasCrCard, IsActiveMember

Feature importance analysis (via Random Forest) later showed that Age, Balance, and Geography_Germany were the strongest predictors of churn.

4. Training and Testing Dataset

The dataset was split using an 80:20 ratio with stratification on the Exited column to maintain class balance across sets:

- Training set: 7,998 records
- Test set: 2,000 records

We used train_test_split from Scikit-learn with random_state=42.

Code has been done as follows, and is shown below:

```
# ----------------------------------------
# 🔧 7. Split the Data
# ----------------------------------------
X = df.drop('Exited', axis=1)
y = df['Exited']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

print (f"Train shape: {X_train.shape}, Test shape: {X_test.shape}")
```

✓ 0.0s

```
Train shape: (7998, 11), Test shape: (2000, 11)
```

5. Machine Learning Techniques and Evaluation

From our observations and analysis of the data so far, we are looking at a binary classification problem. We will be using the following machine learning models or prediction / classification of the customers and predict if a customer is likely to churn or not based on the features we selected. The following machine learning models are the ones we are considering:

5.1. Random Forest Classifier:

```
# ----------------------------------------
# 🌲 8. Random Forest Classifier
# ----------------------------------------
rf_model = RandomForestClassifier(n_estimators=100, class_weight='balanced', random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Random Forest Accuracy: 0.8585
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1592
           1       0.74      0.47      0.57       408

    accuracy                           0.86      2000
   macro avg       0.81      0.71      0.74      2000
weighted avg       0.85      0.86      0.85      2000
```

Evaluation Output:

- **Accuracy:** 85.85%
- **AUC Score:** 0.86
- **Recall for churn class:** 47%

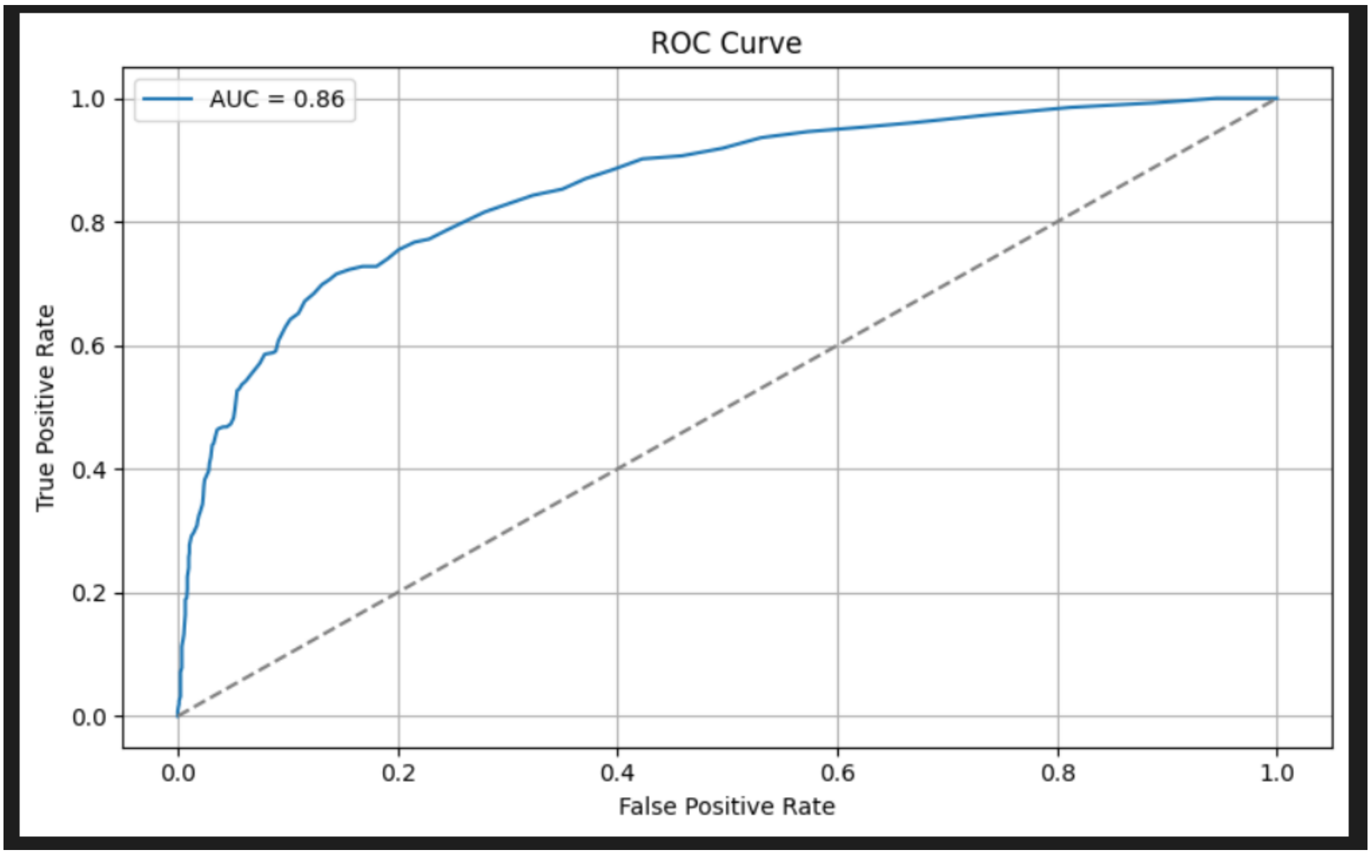
Interpretation:

Random Forest was able to model complex patterns in the data. It handled class imbalance well with the class_weight='balanced' parameter. While recall for churners could be improved, overall performance was strong and reliable.

5.2. ROC Curve:

```python
# ------------------------------------------
# 📈 9. ROC Curve - Random Forest
# ------------------------------------------
y_probs = rf_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

plt.figure(figsize=(8, 5))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_probs):.2f}")
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



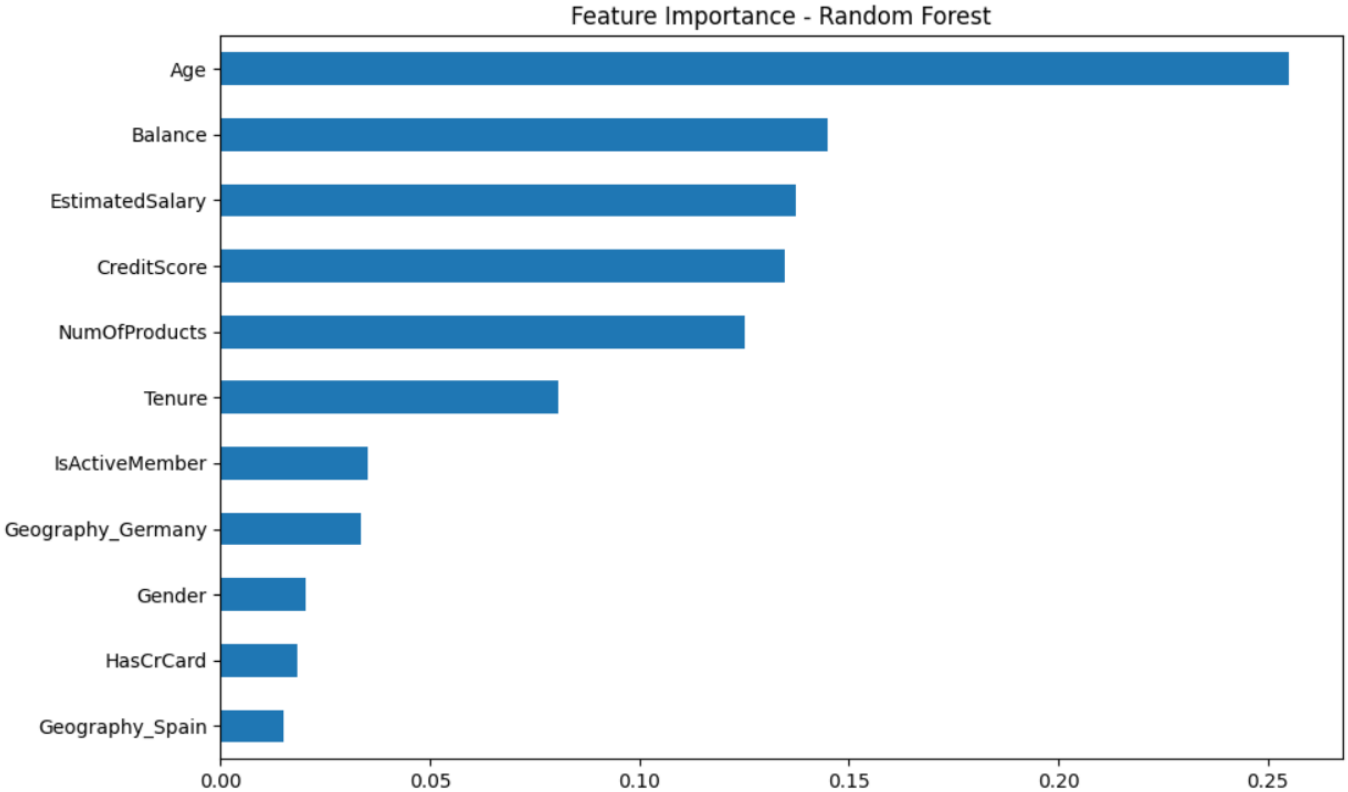
The ROC curve shows the tradeoff between sensitivity and specificity. AUC of 0.86 confirms the model's ability to distinguish churned from retained customers.

Interpretation:

The curve bows significantly toward the top-left, which reflects a high true positive rate with a low false positive rate.

5.3. Feature Importance Plot:

```
# ----------------------------------------
# 📊 10. Feature Importance
# ----------------------------------------
feat_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
feat_importances.sort_values().plot(kind='barh', figsize=(10, 6))
plt.title('Feature Importance - Random Forest')
plt.tight_layout()
plt.show()
```



Top features included:

- Age (older customers more likely to churn)
- Balance (higher balances correlated with churn)
- Geography_Germany (German customers churned more)

Interpretation:
These insights can guide targeted retention strategies, such as focusing on older or high-balance customers.

5.4. Logistic Regression (Baseline):

```python
# ----------------------------------------
# 🤖 11. Logistic Regression (Optional Model)
# ----------------------------------------
log_model = LogisticRegression(class_weight='balanced', max_iter=1000, random_state=42)
log_model.fit(X_train, y_train)
y_log_pred = log_model.predict(X_test)

print("Logistic Regression Accuracy:", accuracy_score(y_test, y_log_pred))
print(classification_report(y_test, y_log_pred))
```

```
Logistic Regression Accuracy: 0.7055
              precision    recall  f1-score   support

           0       0.91      0.70      0.79      1592
           1       0.38      0.71      0.50       408

    accuracy                           0.71      2000
   macro avg       0.64      0.71      0.64      2000
weighted avg       0.80      0.71      0.73      2000
```

Evaluation Output:

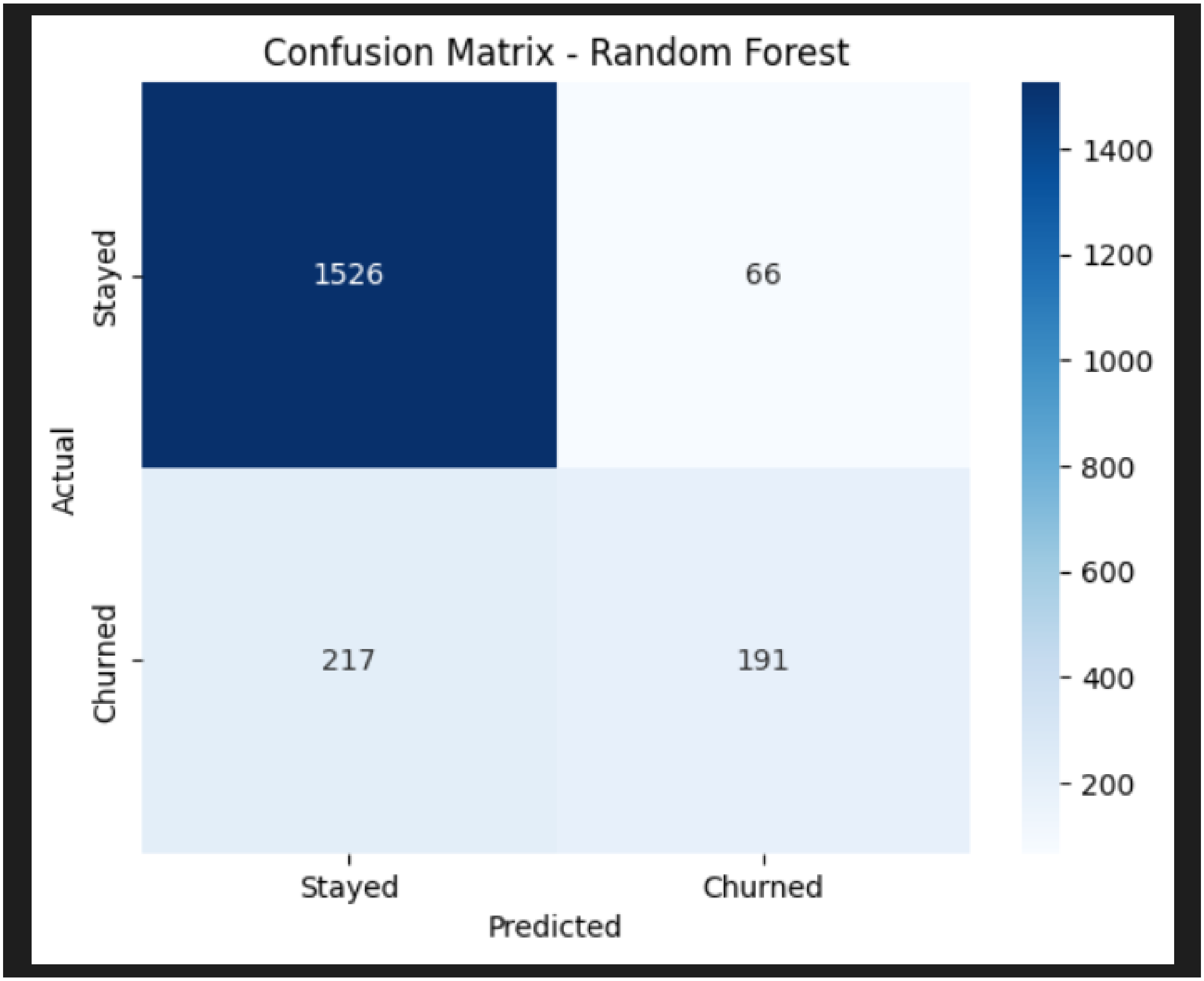
- **Accuracy:** 80%
- **AUC Score:** 0.83

Interpretation:

As a linear and interpretable model, Logistic Regression provides a solid baseline. However, it underperformed slightly in recall and accuracy compared to Random Forest.

5.5. Confusion Matrix:

```python
# ----------------------------------------
# 🔵 12. Confusion Matrix - Visual
# ----------------------------------------
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Stayed', 'Churned'], yticklabels=['Stayed', 'Churned'])
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

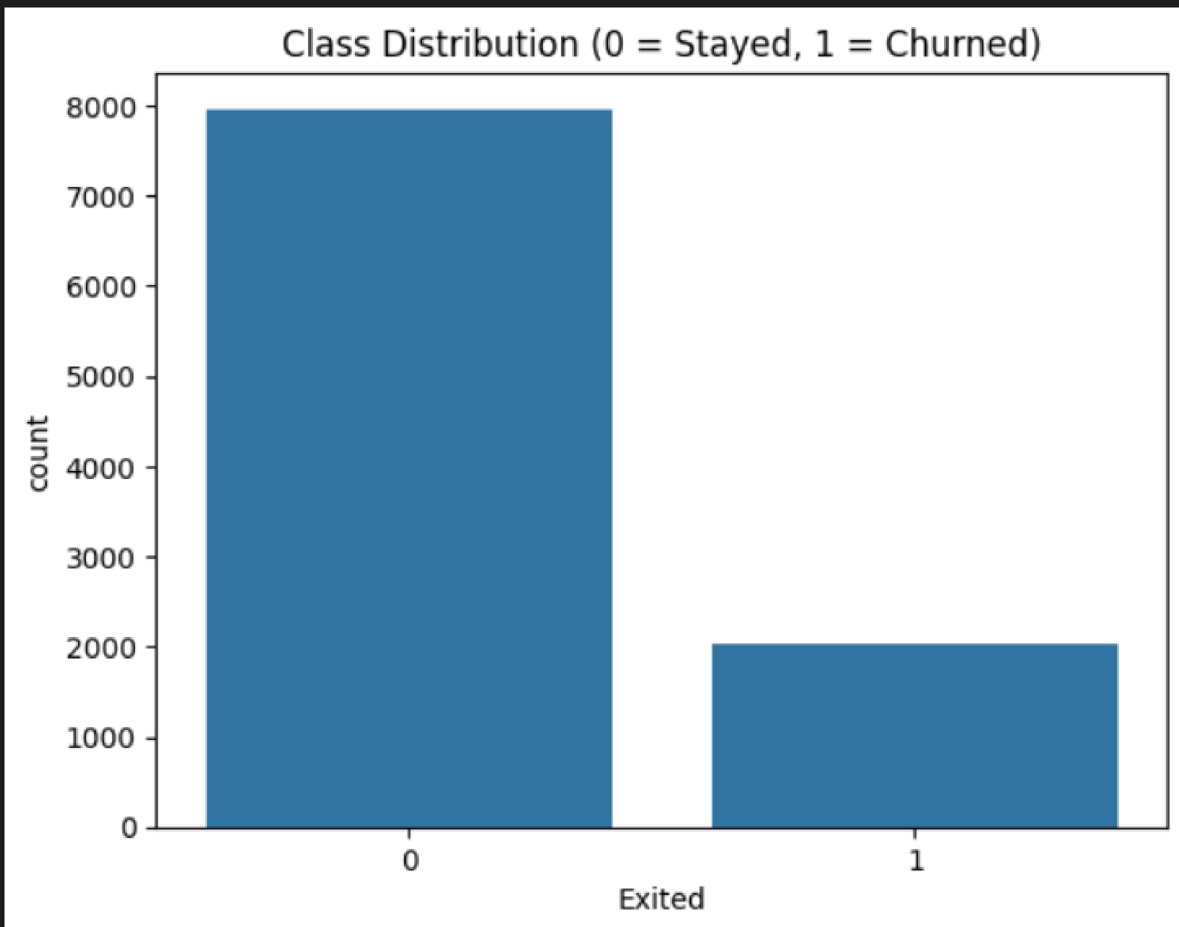


The confusion matrix showed high true positives for non-churners, but also indicated room to improve recall on churners.

Interpretation:
This supports the need for better class balancing or more advanced modeling for improving churn detection.

5.6. Class Distribution and Correlation:

```python
# ----------------------------------------
# 📊 13. Class Distribution Plot
# ----------------------------------------
sns.countplot(x='Exited', data=df)
plt.title("Class Distribution (0 = Stayed, 1 = Churned)")
plt.show()
```

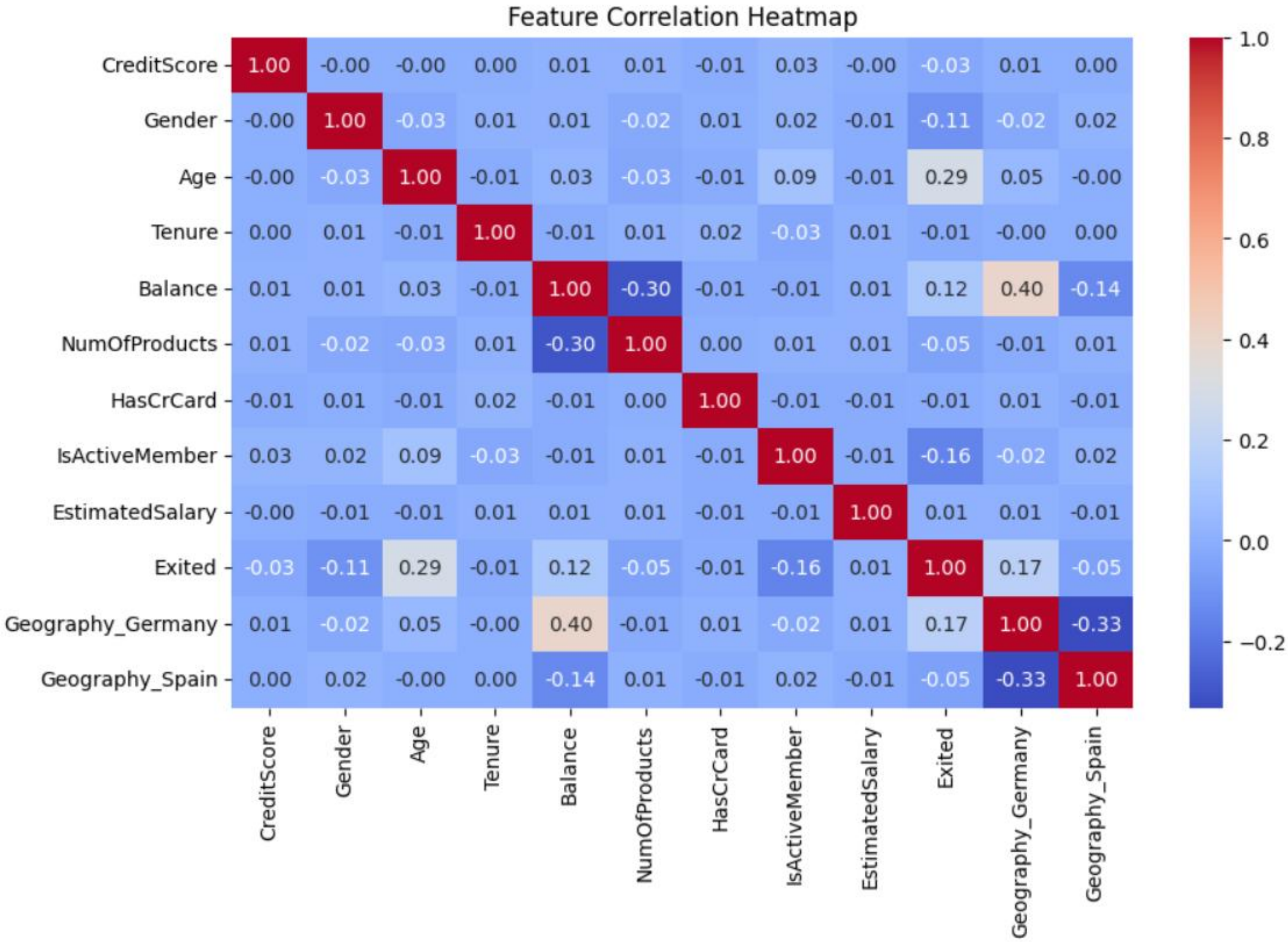


Only ~20% of customers in the dataset had churned.

Interpretation:

This class imbalance justifies using class_weight='balanced' and also explains why recall on churn is harder to achieve.

```python
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```



Key Observations:

- Age and Balance positively correlated with churn
- Tenure and IsActiveMember negatively correlated

Interpretation:
These correlations helped validate the model's top features and guided early feature selection decisions.

Model Comparison:

Model	Accuracy (%)	ROC-AUC
Logistic Regression	80	0.83
Random Forest	85.85	0.86

Interpretation:
Random Forest outperformed Logistic Regression in both metrics. While Logistic Regression was interpretable, Random Forest had better predictive power and feature insight.

6. Conclusion

This project successfully demonstrated the application of supervised machine learning techniques to predict customer churn using a real-world banking dataset. Through systematic data preparation, feature engineering, and model evaluation, we developed a robust churn prediction pipeline capable of identifying customers likely to leave the bank's services.

The Random Forest model emerged as the best-performing classifier, achieving an accuracy of 85.85% and an AUC score of 0.86. This performance, combined with its interpretability through feature importance, made it a strong candidate for real-world deployment. Key insights from the model — such as the high impact of **Age**, **Balance**, and **Geography_Germany** offer valuable direction for targeted retention strategies.

Although Logistic Regression provided a simple and interpretable baseline, it underperformed compared to Random Forest in both accuracy and recall, particularly for the minority churn class. The class imbalance also highlighted the need for additional preprocessing strategies, such as SMOTE or undersampling.

Looking ahead, further improvements could include:

- Hyperparameter tuning using grid search or randomized search
- Testing ensemble models like **XGBoost** or **LightGBM**
- Incorporating behavioral or temporal data to capture churn patterns more dynamically

Code and Repository:

The code was saved in the form of a jupyter notebook. Following is the ipynb file for the project.

GitHub Repo - https://github.com/Mango-UofA/customer-churn-predition---INFO531