# Customer Churn Prediction

**Objective: To develop a predictive model using the provided churn dataset that accurately identifies customers likely to exit**

**Members: Tyler, Humza, Angelina, Christine, Efaz, Nathan**

In [ ]:

```
# General Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as ms
%matplotlib inline

# Machine Learning Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_score, classification
_report
```

# Part 1: Data Understanding

## Exploratory Data Analysis (EDA)

**Let's take a look at the dataset and view only the first 10 rows**

In [ ]:

```
churn = pd.read_csv('Churn.csv')
print("Size of Churn Set: ", churn.shape)
churn.head(10)
```

```
Size of Churn Set:  (10000, 14)
```

Out[ ]:

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCar |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 |
| **1** | 2 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 |
| **2** | 3 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 |
| **3** | 4 | 15701354 | Boni | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 |
| **4** | 5 | 15737888 | Mitchell | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | 1 |
| **5** | 6 | 15574012 | Chu | 645 | Spain | Male | 44 | 8 | 113755.78 | 2 | 1 |
| **6** | 7 | 15592531 | Bartlett | 822 | France | Male | 50 | 7 | 0.00 | 2 | 1 |
| **7** | 8 | 15656148 | Obinna | 376 | Germany | Female | 29 | 4 | 115046.74 | 4 | 1 |
| **8** | 9 | 15792365 | He | 501 | France | Male | 44 | 4 | 142051.07 | 2 | 0 |
| **9** | 10 | 15592389 | H? | 684 | France | Male | 27 | 2 | 134603.88 | 1 | 1 |

**Our target variable seems to be the Exited column which has responses containing either 1 for yes and 0 for no. Therefore we are dealing with a Binary Classification problem**

**It seems the dataset has 14 columns, with two of them likely not being two important: RowNumber and CustomerId, so let's drop the two columns**

In [ ]:

```
# Dropping the first two columns as it doesn't offer much outside of being labels for eac
h customer
churn = churn.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
churn.head(5)
```

Out[ ]:

| | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 | 1 | 101348.88 |
| **1** | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 | 1 | 112542.58 |
| **2** | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 | 0 | 113931.57 |
| **3** | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 | 0 | 93826.63 |
| **4** | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | 1 | 1 | 79084.10 |

**Now let's take a look at the data types for each of the variables in the dataframe**

In [ ]:

```
data_types = churn.dtypes

print("Data Types and Additional Information:")
for column in churn.columns:
  print(f'Column: {column}')
  print(f' - Data Type: {data_types[column]}')
  print(f' - Number of Unique Values: {churn[column].nunique()}')
  print(f' - Sample Values: {churn[column].dropna().unique()[:5]}')
```

```
Data Types and Additional Information:
Column: CreditScore
 - Data Type: int64
 - Number of Unique Values: 460
 - Sample Values: [619 608 502 699 850]
Column: Geography
 - Data Type: object
 - Number of Unique Values: 3
 - Sample Values: ['France' 'Spain' 'Germany']
Column: Gender
 - Data Type: object
 - Number of Unique Values: 2
 - Sample Values: ['Female' 'Male']
Column: Age
 - Data Type: int64
 - Number of Unique Values: 70
 - Sample Values: [42 41 39 43 44]
Column: Tenure
 - Data Type: int64
 - Number of Unique Values: 11
 - Sample Values: [2 1 8 7 4]
Column: Balance
 - Data Type: float64
 - Number of Unique Values: 6382
 - Sample Values: [     0.     83807.86 159660.8   125510.82 113755.78]
Column: NumOfProducts
 - Data Type: int64
 - Number of Unique Values: 4
 - Sample Values: [1 3 2 4]
Column: HasCrCard
```

```
 - Data Type: int64
 - Number of Unique Values: 2
 - Sample Values: [1 0]
Column: IsActiveMember
 - Data Type: int64
 - Number of Unique Values: 2
 - Sample Values: [1 0]
Column: EstimatedSalary
 - Data Type: float64
 - Number of Unique Values: 9999
 - Sample Values: [101348.88 112542.58 113931.57  93826.63  79084.1 ]
Column: Exited
 - Data Type: int64
 - Number of Unique Values: 2
 - Sample Values: [1 0]
```

**Numerical Variables: CreditScore, Age, Tenure, Balance, NumOfProducts, HasCrCard,IsActiveMember,EstimatedSalary,Exited Categorical Variables: Surname, Geography, Gender**

In [ ]:

```
churn.describe()
```

Out[ ]:

| | CreditScore | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSal |
|---|---|---|---|---|---|---|---|---|
| **count** | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.00000 | 10000.000000 | 10000.0000 |
| **mean** | 650.528800 | 38.921800 | 5.012800 | 76485.889288 | 1.530200 | 0.70550 | 0.515100 | 100090.2398 |
| **std** | 96.653299 | 10.487806 | 2.892174 | 62397.405202 | 0.581654 | 0.45584 | 0.499797 | 57510.4928 |
| **min** | 350.000000 | 18.000000 | 0.000000 | 0.000000 | 1.000000 | 0.00000 | 0.000000 | 11.5800 |
| **25%** | 584.000000 | 32.000000 | 3.000000 | 0.000000 | 1.000000 | 0.00000 | 0.000000 | 51002.1100 |
| **50%** | 652.000000 | 37.000000 | 5.000000 | 97198.540000 | 1.000000 | 1.00000 | 1.000000 | 100193.9150 |
| **75%** | 718.000000 | 44.000000 | 7.000000 | 127644.240000 | 2.000000 | 1.00000 | 1.000000 | 149388.2475 |
| **max** | 850.000000 | 92.000000 | 10.000000 | 250898.090000 | 4.000000 | 1.00000 | 1.000000 | 199992.4800 |

**Seems that there might not be any missing data just by looking at counts, however, let's look into missing values anyways in a later section.**

**But, there likely is outliers if we look at the min and max values of the Balance and EstimatedSalary variables so we will deal with these later.**

**Now, let's look at the correlation between each variable and see if any pairs are particularly connected with each other.**

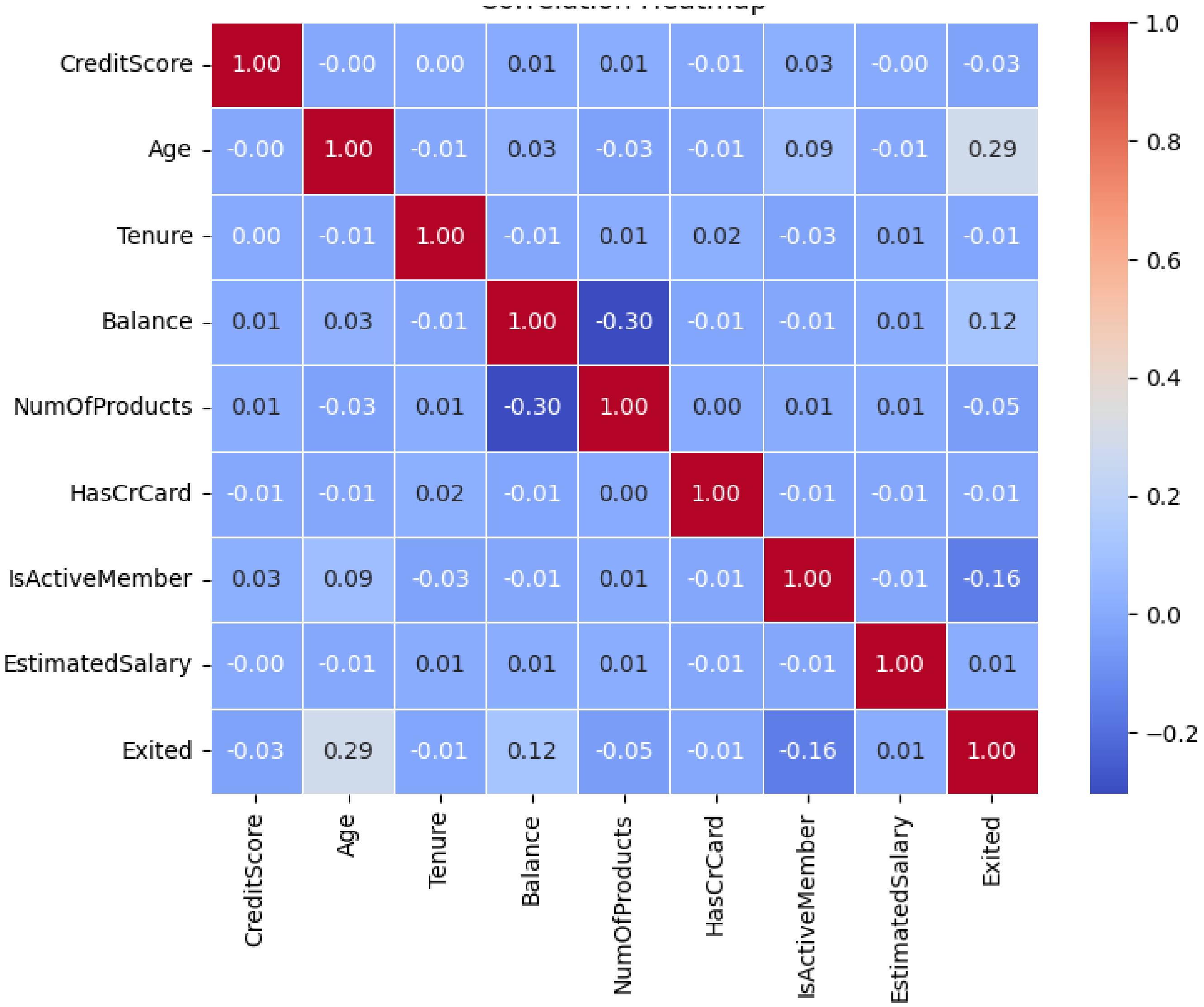
In [ ]:

```
# Creating a correlation matrix to see what variables are highly/not correlated with each
other
correlation_matrix = churn.corr()

# Creating a correlation heatmap to visualize the matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```

```
<ipython-input-6-c61476b88c26>:2: FutureWarning: The default value of numeric_only in Dat
aFrame.corr is deprecated. In a future version, it will default to False. Select only val
id columns or specify the value of numeric_only to silence this warning.
  correlation_matrix = churn.corr()
```

Correlation Heatmap




There seems to be some sort of negative correlation with Balance and NumOfProducts, Age and Exited, IsActiveMember and Exited.

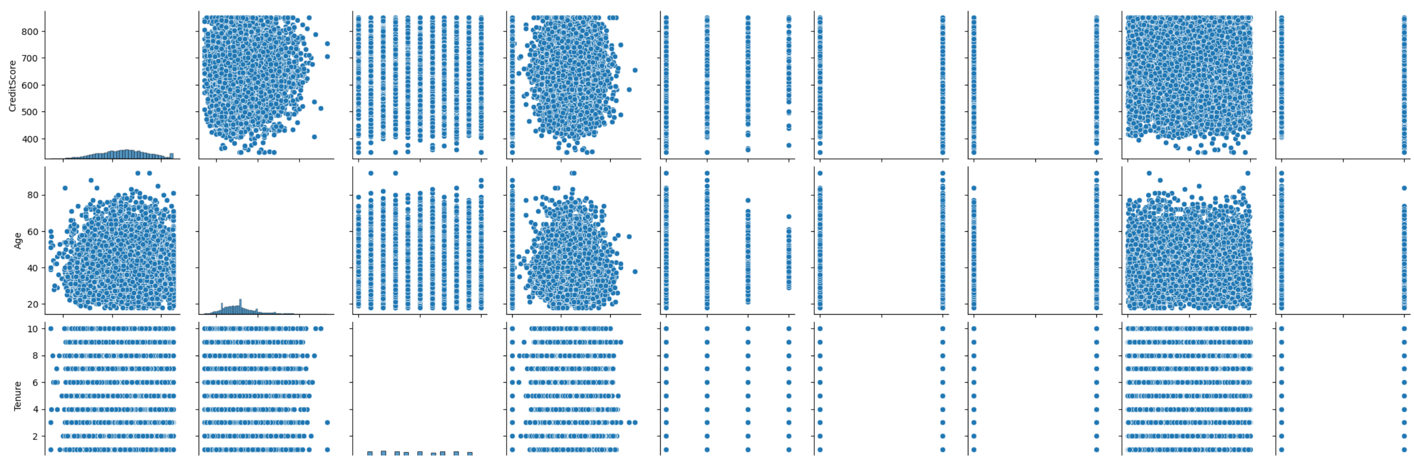
# Visualization

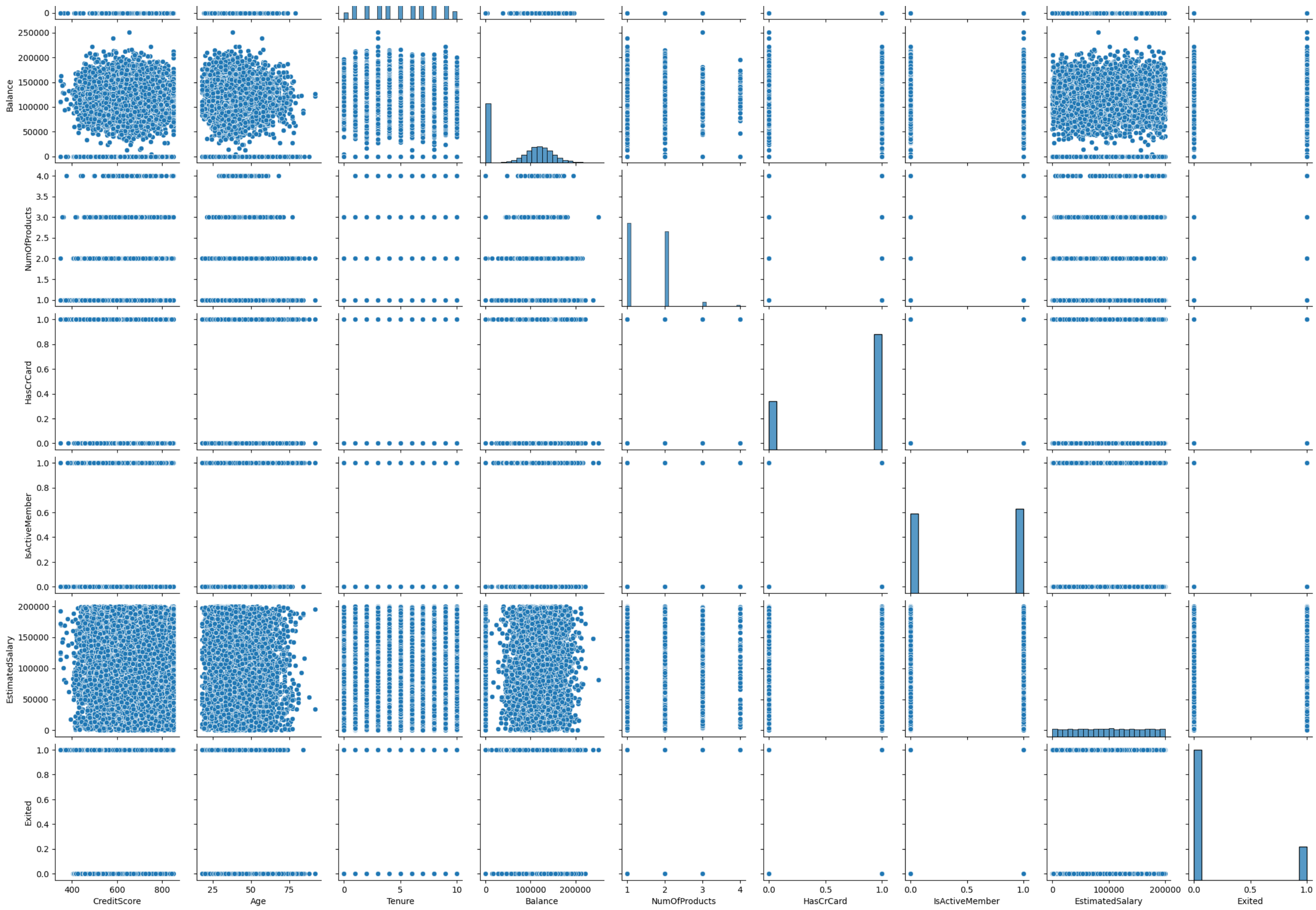
**Let's check the distribution of the numerical variables in this dataset**

In [ ]:

```
numerical_vars = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard'
, 'IsActiveMember', 'EstimatedSalary', 'Exited']
numerical_df = churn[numerical_vars]

sns.pairplot(numerical_df) #Pairs each variable with one another to see if there's any sp
ecific patterns
plt.show()
```






# Part 2: Data Preprocessing

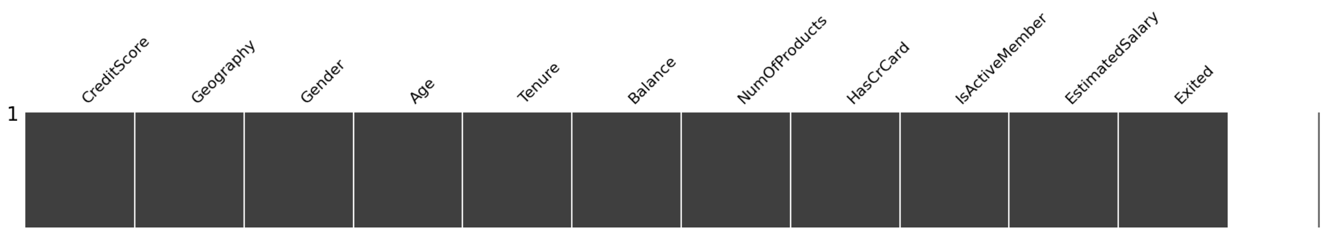
## Cleaning

### Let's see if there are missing values and duplicated rows in our dataset

In [ ]:

```
#Checking for Missing Values
missing_values = churn.isnull().sum()
print("Missing values:\n", missing_values)
ms.matrix(churn)
plt.show()
```

```
Missing values:
 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 [ ]:

```
# Checking duplicated records
duplicates = churn.duplicated().sum()
print('Number of Duplicated Entries: ',duplicates)
```

```
Number of Duplicated Entries:  0
```

**Looks like we don't have any null values nor duplicated rows! Cool, we can now move on to the next stage and transform our data as fit**

## Transformation

**The Balance column has some zeroes in each cell which really doesn't make sense, so let's replace them with the mean of the overall column**

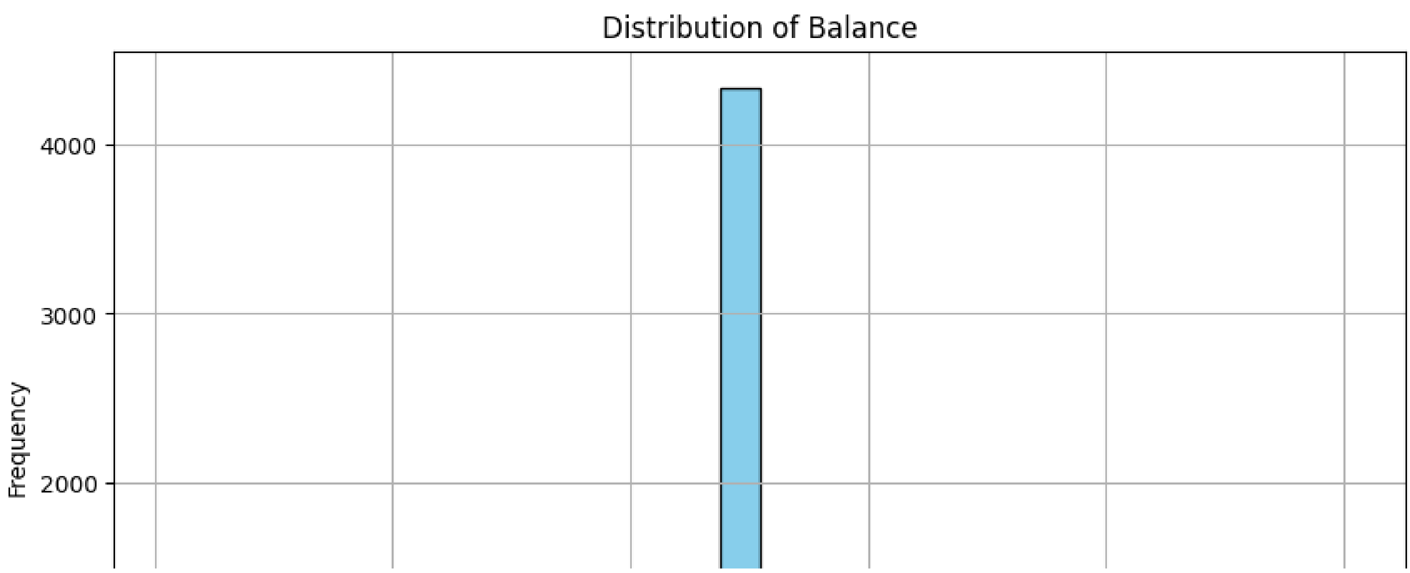
In [ ]:

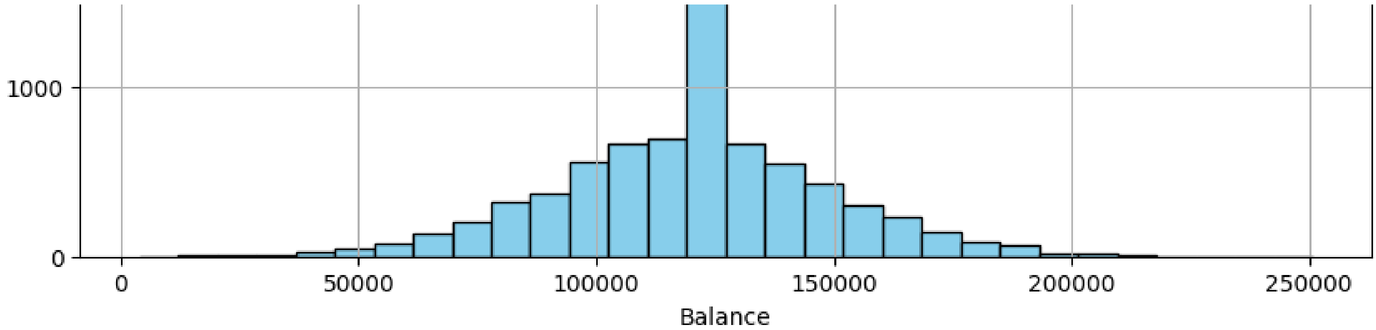
```
# Calculate the mean of non-zero values in the 'Balance' column
balance_mean = churn[churn['Balance'] != 0]['Balance'].mean()

# Replace zeros with the mean value
churn['Balance'] = churn['Balance'].replace(0, balance_mean)
```

In [ ]:

```
# Plot a histogram of the "Balance" column
plt.figure(figsize=(10, 6))
plt.hist(churn['Balance'], bins=30, color='skyblue', edgecolor='black')
plt.title('Distribution of Balance')
plt.xlabel('Balance')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```






# Part 3: Model Development and Evaluation

## Model Selection

In [ ]:

```
# Model Libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc

# Pipeline Procedures
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

In [ ]:

```
X =  churn.drop(columns=['Exited']) # setting up our X set that uses every column except Exited
y =  churn['Exited']# sets up the y set that only contains the Exited variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
) # splits the X and y sets into train/test containing 20% of the dataset randomly
```

## Training and Validation

### What Metrics are being considered?

- **Accuracy and CV Accuracy:**
  - **Accuracy measures the proprtion of correct predictions among total predictions**
  - **Cross-Validation Accuracy is a measure of how well the model performs across multiple subsets of the data using cross-validation**
- **Precision:**
  - **Measures the proportion of true positive prediction among all positive predictions**
- **Recall:**
  - **Measures the true positive predictions among all actual positive instances**
- **F1-Score:**
  - **Harmonic mean of precision and recall**
  - **Provides a balance between precision and recall and is especially useful when distribution is imbalanced**
- **AUC-ROC Curve:**
  - **ROC: plot of the true positive rate against the false positive rate**
  - **AUC: quantifies the overall performance of the model across all possible thresholds**

### K-Nearest Neighbor

In [ ]:

```
from sklearn.model_selection import cross_val_score
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Estimate
dSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle miss
ing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical vari
ables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Create a pipeline with preprocessing and KNN classifier
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', KNeighborsClassifier())
])

# Step 3: Perform cross-validation
cv_scores = cross_val_score(pipeline, X, y, cv=5)  # Perform 5-fold cross-validation

# Step 4: Display cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.8375 0.84   0.829  0.83   0.829 ]
Mean CV accuracy: 0.8331
```

#### KNN Model

In [ ]:

```
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Estimate
dSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle miss
ing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical vari
ables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

```
# Step 2: Define the KNN classifier
k = 5
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Step 3: Create the pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', knn_classifier)
])

# Step 4: Train the model
pipeline.fit(X_train, y_train)

# Step 5: Evaluate the model
y_pred = pipeline.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.841
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      1607
           1       0.63      0.46      0.53       393

    accuracy                           0.84      2000
   macro avg       0.75      0.70      0.72      2000
weighted avg       0.83      0.84      0.83      2000
```

### ROC Curve

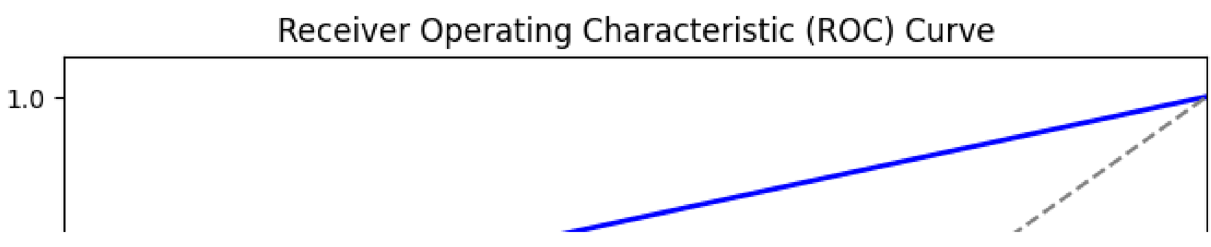
In [ ]:

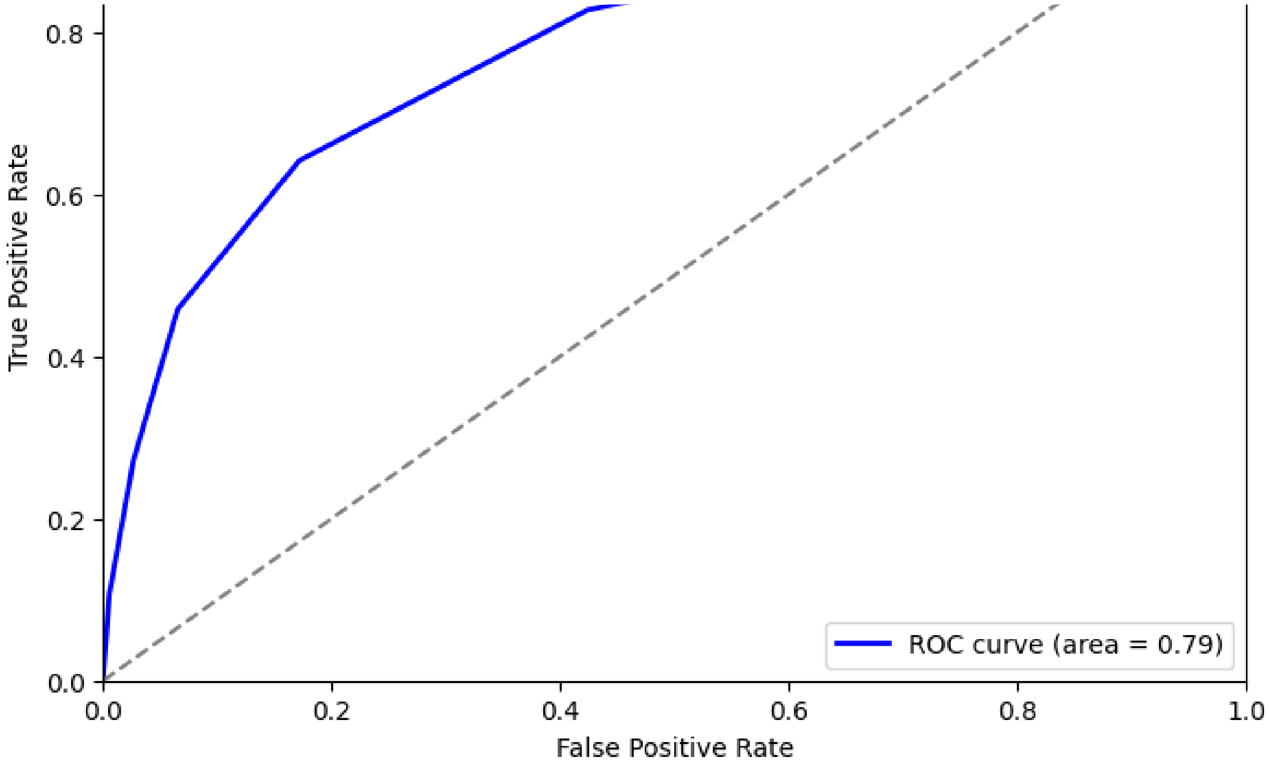
```
# Step 1: Predict probabilities for the testing data
y_pred_proba = pipeline.predict_proba(X_test)[:, 1]

# Step 2: Compute the False Positive Rate (FPR) and True Positive Rate (TPR) at various t
hresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Step 3: Calculate the Area Under the ROC Curve (AUC-ROC)
auc_roc = auc(fpr, tpr)

# Step 4: Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'.format(auc_roc)
)
plt.plot([0, 1], [0, 1], color='gray', 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 (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```






## DecisionTreeClassifier

In [ ]:

```
from sklearn.model_selection import cross_val_score
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Estimate
dSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle miss
ing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical vari
ables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Create a pipeline with preprocessing and DecisionTreeClassifier
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier())
])

# Step 3: Perform cross-validation
cv_scores = cross_val_score(pipeline, X, y, cv=5)  # Perform 5-fold cross-validation

# Step 4: Display cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.7895 0.786  0.7835 0.781  0.7775]
Mean CV accuracy: 0.7835
```

In [ ]:

```
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Estimate
dSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle miss
ing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical vari
ables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Define the DecisionTreeClassifier
decision_tree_classifier = DecisionTreeClassifier(random_state=42)

# Step 3: Create the pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', decision_tree_classifier)
])

# Step 4: Train the model
pipeline.fit(X_train, y_train)

# Step 5: Evaluate the model
y_pred = pipeline.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.776
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.85      0.86      1607
           1       0.43      0.46      0.45       393

    accuracy                           0.78      2000
   macro avg       0.65      0.66      0.65      2000
weighted avg       0.78      0.78      0.78      2000
```

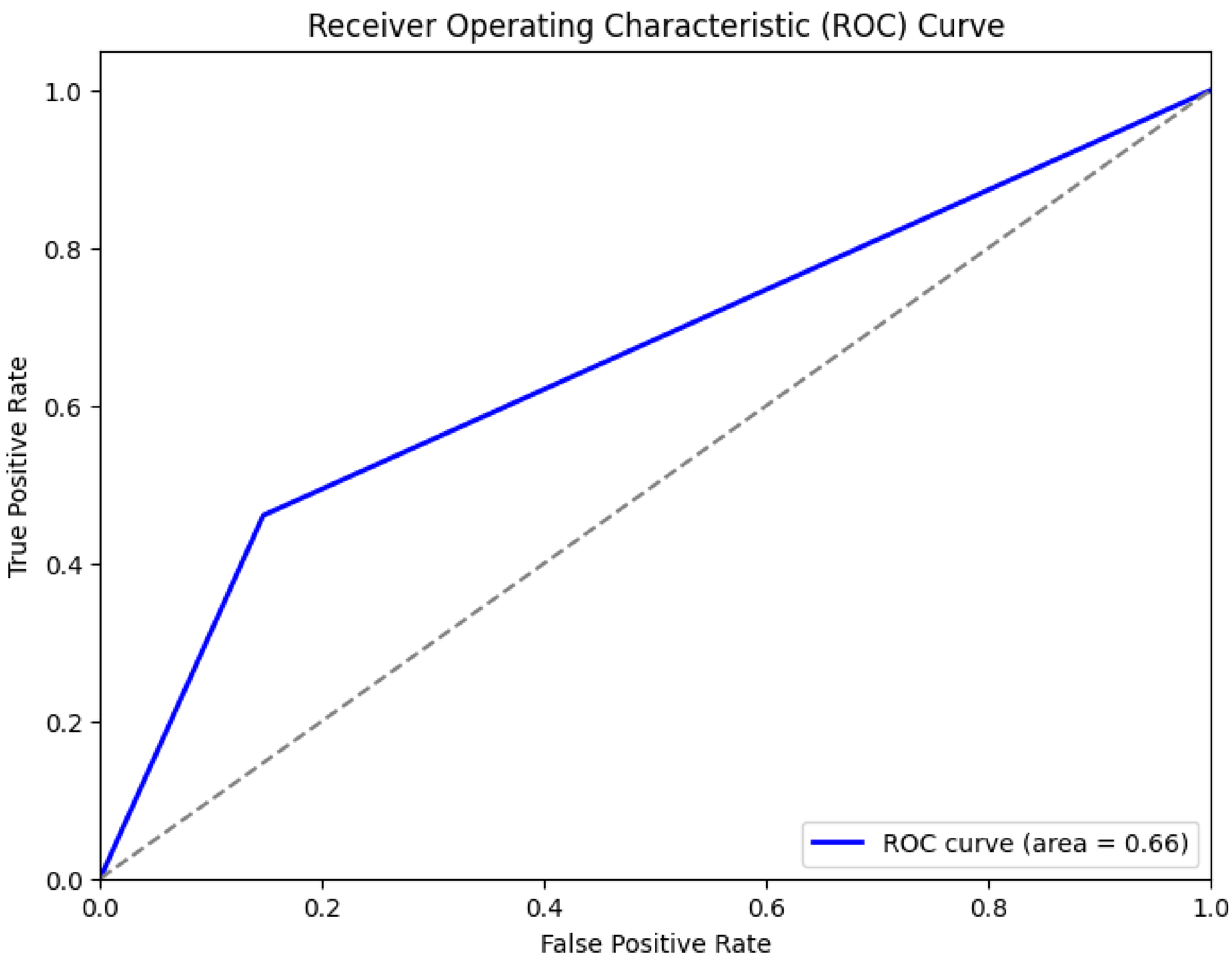
In [ ]:

```
# Step 1: Predict probabilities for the testing data
y_pred_proba = pipeline.predict_proba(X_test)[:, 1]

# Step 2: Compute the False Positive Rate (FPR) and True Positive Rate (TPR) at various t
hresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Step 3: Calculate the Area Under the ROC Curve (AUC-ROC)
auc_roc = auc(fpr, tpr)
```

```
# Step 4: Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'.format(auc_roc)
)
plt.plot([0, 1], [0, 1], color='gray', 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 (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




## Naives Bayes

In [ ]:

```
from sklearn.model_selection import cross_val_score
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Estimate
dSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle miss
ing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical vari
ables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
```

```
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Create a pipeline with preprocessing and Naive Bayes classifier
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', naive_bayes_classifier)
])

# Step 3: Perform cross-validation
cv_scores = cross_val_score(pipeline, X, y, cv=5)  # Perform 5-fold cross-validation

# Step 4: Display cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.797  0.801  0.808  0.807  0.7955]
Mean CV accuracy: 0.8017
```

In [ ]:

```
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle missing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical variables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Define the Gaussian Naive Bayes classifier
naive_bayes_classifier = GaussianNB()

# Step 3: Create the pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', naive_bayes_classifier)
])

# Step 4: Train the model
pipeline.fit(X_train, y_train)

# Step 5: Evaluate the model
y_pred = pipeline.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8125
Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.85      0.93      0.89      1607
           1       0.54      0.35      0.42       393

    accuracy                           0.81      2000
   macro avg       0.69      0.64      0.66      2000
weighted avg       0.79      0.81      0.80      2000
```

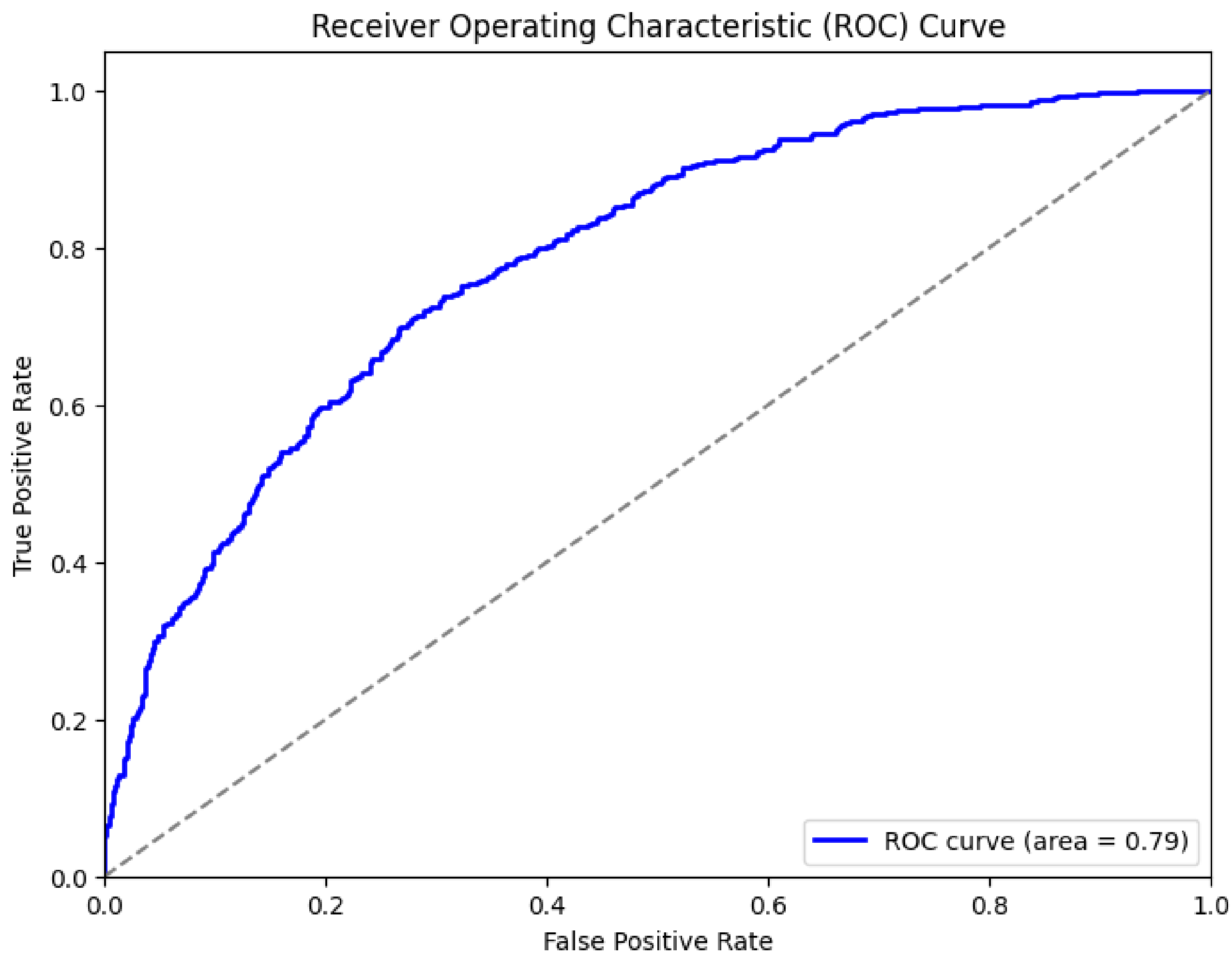
In [ ]:

```
# Step 1: Predict probabilities for the testing data
y_pred_proba = pipeline.predict_proba(X_test)[:, 1]

# Step 2: Compute the False Positive Rate (FPR) and True Positive Rate (TPR) at various t
hresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Step 3: Calculate the Area Under the ROC Curve (AUC-ROC)
auc_roc = auc(fpr, tpr)

# Step 4: Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'.format(auc_roc)
)
plt.plot([0, 1], [0, 1], color='gray', 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 (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




## RandomForestClassifier

In [ ]:

```python
from sklearn.model_selection import cross_val_score
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle missing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical variables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Create a pipeline with preprocessing and RandomForest classifier
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', random_forest_classifier)])

# Step 3: Perform cross-validation
cv_scores = cross_val_score(pipeline, X, y, cv=5)  # Perform 5-fold cross-validation

# Step 4: Display cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.8475 0.848  0.8485 0.8465 0.855 ]
Mean CV accuracy: 0.8491
```

In [ ]:

```python
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle missing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical variables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Define the RandomForestClassifier
random_forest_classifier = RandomForestClassifier(random_state=42)

# Step 3: Create the pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', random_forest_classifier)
])
```

```python
# Step 4: Train the model
pipeline.fit(X_train, y_train)

# Step 5: Evaluate the model
y_pred = pipeline.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8525
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      1607
           1       0.69      0.45      0.54       393

    accuracy                           0.85      2000
   macro avg       0.78      0.70      0.73      2000
weighted avg       0.84      0.85      0.84      2000
```

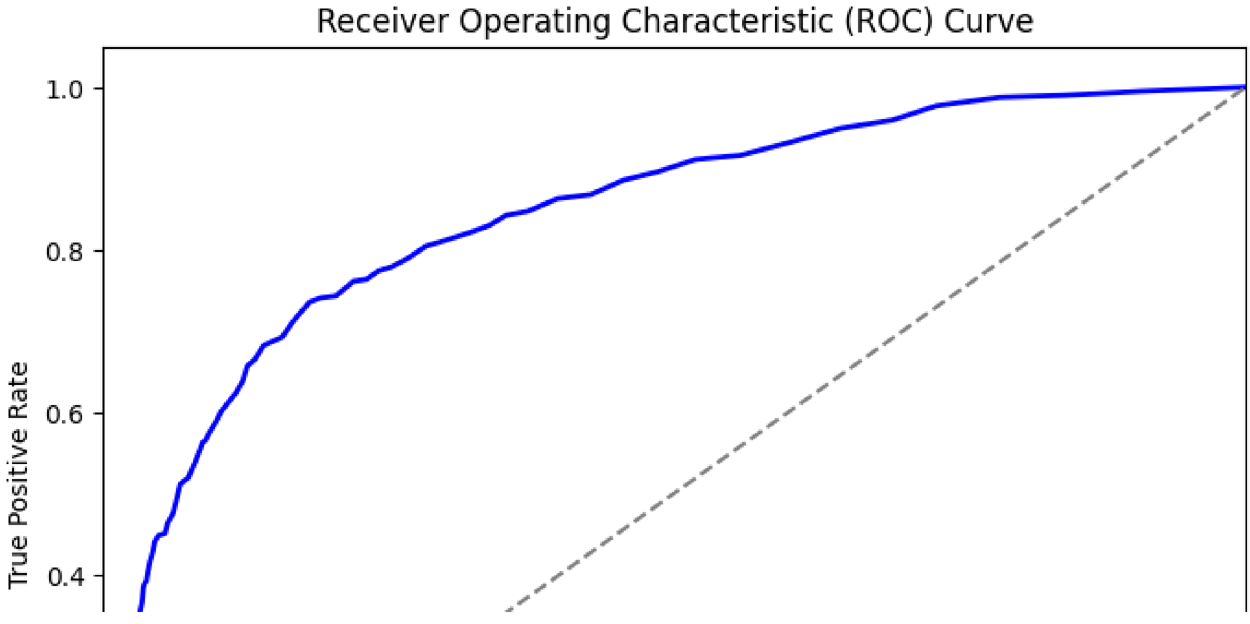
In [ ]:

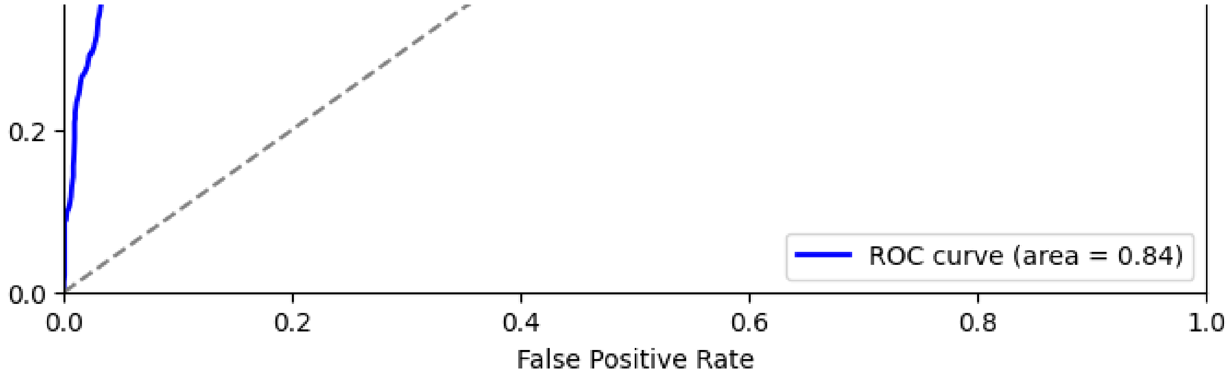
```python
# Step 1: Predict probabilities for the testing data
y_pred_proba = pipeline.predict_proba(X_test)[:, 1]

# Step 2: Compute the False Positive Rate (FPR) and True Positive Rate (TPR) at various t
hresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Step 3: Calculate the Area Under the ROC Curve (AUC-ROC)
auc_roc = auc(fpr, tpr)

# Step 4: Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'.format(auc_roc)
)
plt.plot([0, 1], [0, 1], color='gray', 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 (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```






### Stochastic Gradiant Descent

In [ ]:

```
from sklearn.model_selection import cross_val_score
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Estimate
dSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle miss
ing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical vari
ables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Create a pipeline with preprocessing and KNN classifier
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', SGDClassifier())
])

# Step 3: Perform cross-validation
cv_scores = cross_val_score(pipeline, X, y, cv=5)  # Perform 5-fold cross-validation

# Step 4: Display cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.796  0.796  0.7965 0.791  0.7905]
Mean CV accuracy: 0.7939999999999999
```

In [ ]:

```
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Estimate
dSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle miss
```

```
ing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical vari
ables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Define the SGDClassifier
sgd_classifier = SGDClassifier(random_state=42)

# Step 3: Create the pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', sgd_classifier)
])

# Step 4: Train the model
pipeline.fit(X_train, y_train)

# Step 5: Evaluate the model
y_pred = pipeline.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred, zero_division=0))
```

```
Accuracy: 0.8035
Classification Report:
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1607
           1       0.00      0.00      0.00       393

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.45      2000
weighted avg       0.65      0.80      0.72      2000
```

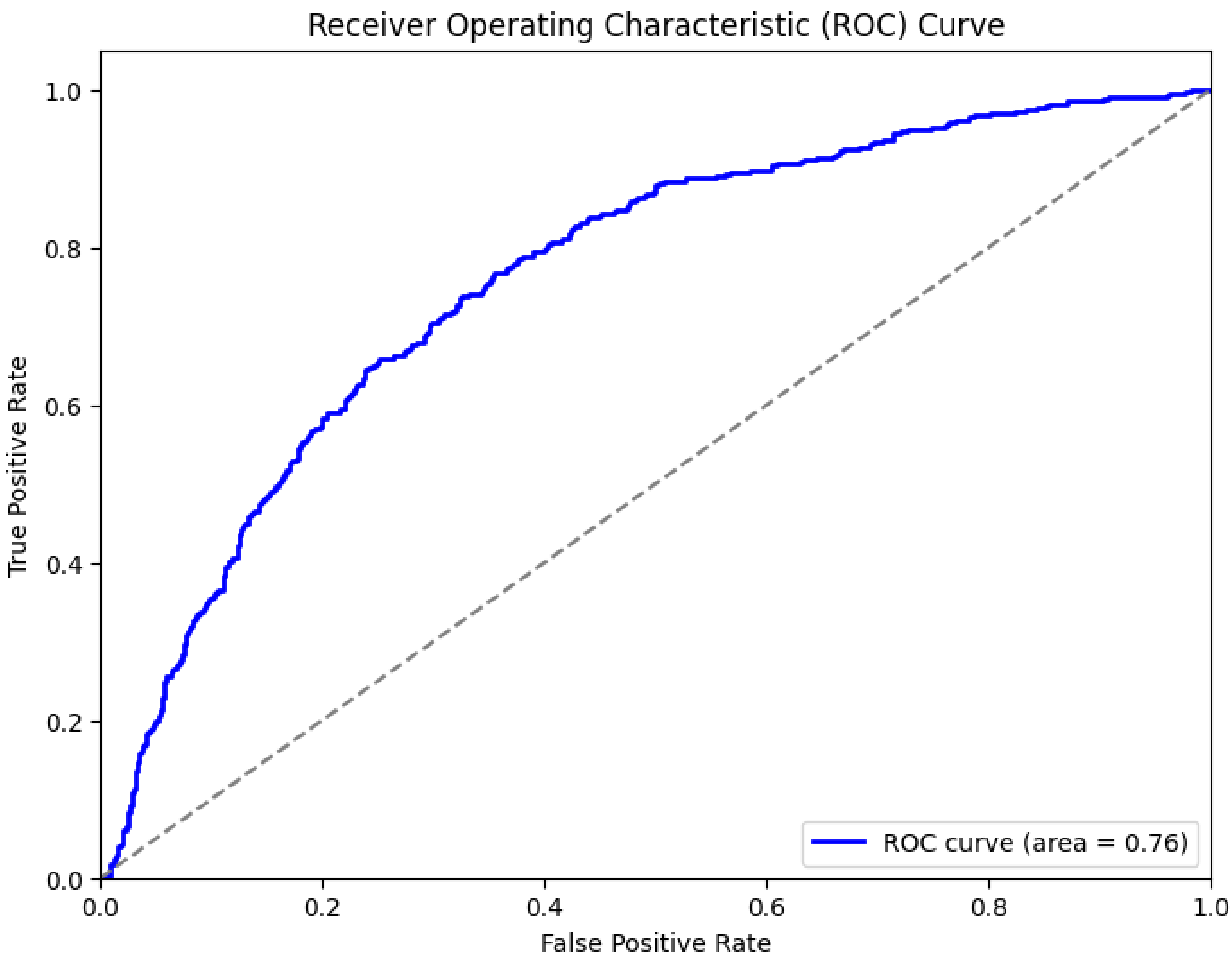
In [ ]:

```
# Step 1: Predict probabilities for the testing data
y_pred_proba = pipeline.predict_proba(X_test)[:, 1]

# Step 2: Compute the False Positive Rate (FPR) and True Positive Rate (TPR) at various t
hresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Step 3: Calculate the Area Under the ROC Curve (AUC-ROC)
auc_roc = auc(fpr, tpr)

# Step 4: Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'.format(auc_roc)
)
plt.plot([0, 1], [0, 1], color='gray', 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 (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




### LogisticRegression

In [ ]:

```
from sklearn.model_selection import cross_val_score
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Estimate
dSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle miss
ing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical vari
ables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Create a pipeline with preprocessing and KNN classifier
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier',logistic_regression)
])

# Step 3: Perform cross-validation
cv_scores = cross_val_score(pipeline, X, y, cv=5)  # Perform 5-fold cross-validation
```

```
# Step 4: Display cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV accuracy:", cv_scores.mean())
```

```
Cross-validation scores: [0.792  0.792  0.793  0.7945 0.798 ]
Mean CV accuracy: 0.7939
```

In [ ]:

```
# Step 1: Define preprocessing steps for numerical and categorical columns
numeric_features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'Estimate
dSalary']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),  # Handle missing values with median
    ('scaler', StandardScaler())  # Scale features
])

categorical_features = ['Geography', 'Gender']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  # Handle miss
ing values with a constant value
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode categorical vari
ables
])

# Combine preprocessing steps for numerical and categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 2: Define the Logistic Regression model
logistic_regression = LogisticRegression(random_state=42)

# Step 3: Create the pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', logistic_regression)
])

# Step 4: Train the model
pipeline.fit(X_train, y_train)

# Step 5: Evaluate the model
y_pred = pipeline.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8045
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.97      0.89      1607
           1       0.51      0.14      0.22       393

    accuracy                           0.80      2000
   macro avg       0.67      0.55      0.56      2000
weighted avg       0.76      0.80      0.76      2000
```

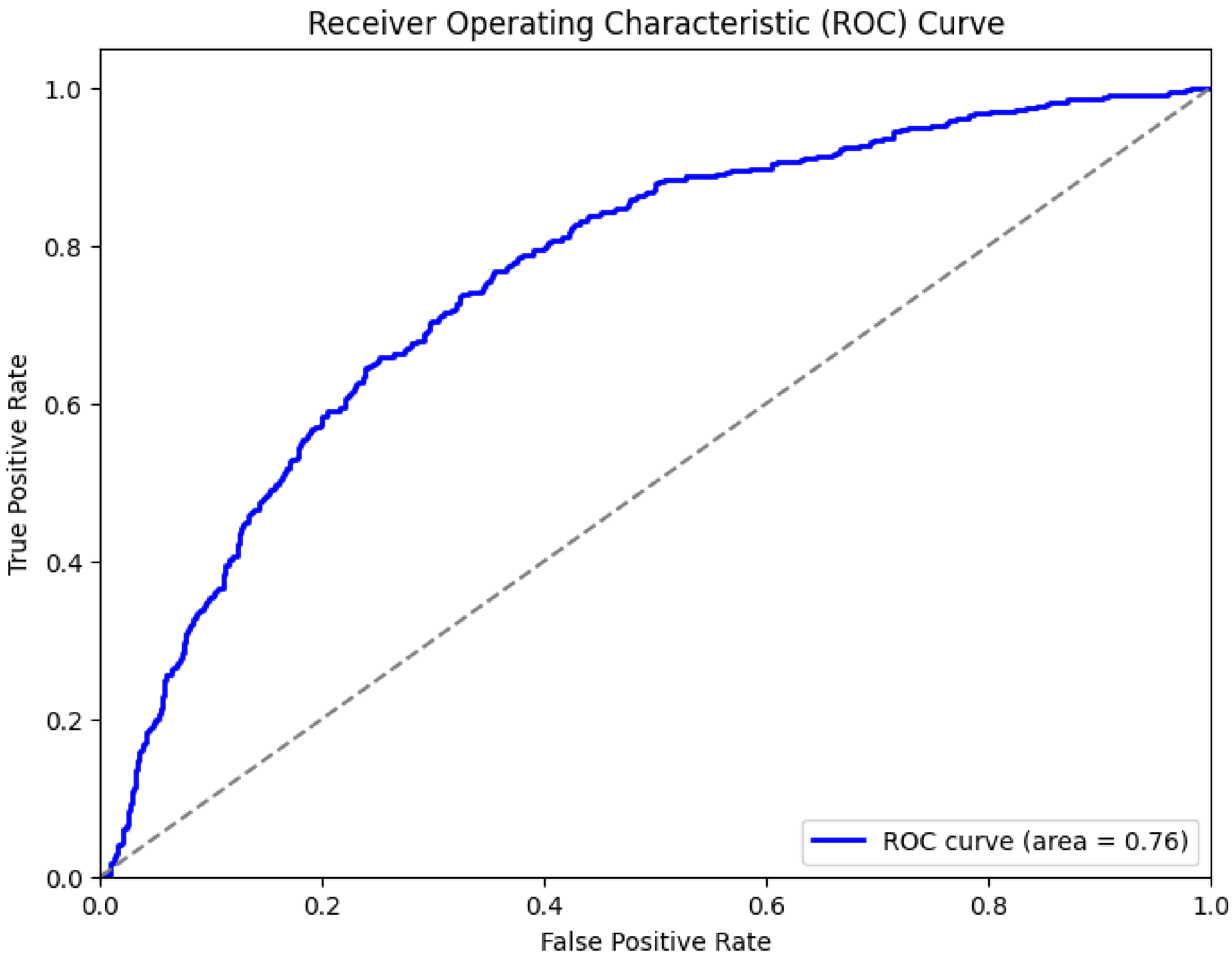
In [ ]:

```
# Step 1: Predict probabilities for the testing data
y_pred_proba = pipeline.predict_proba(X_test)[:, 1]
```

```
# Step 2: Compute the False Positive Rate (FPR) and True Positive Rate (TPR) at various t
hresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Step 3: Calculate the Area Under the ROC Curve (AUC-ROC)
auc_roc = auc(fpr, tpr)

# Step 4: Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = {:.2f})'.format(auc_roc)
)
plt.plot([0, 1], [0, 1], color='gray', 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 (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




# Which Model Is Best?

**Cross-Validation Scores**

- **Best Mean Cross-Validation Score: RandomForest**

**Metric Performance**

- **Accuracy Score**
  - **Highest Accuracy Score: RandomForest**
- **Precision**
  - **Highest Weighted Average: RandomForest**
- **Recall**
  - **Highest Weighted Average: RandomForest**
- **F1-Score**

    - **Highest Weighted Average: RandomForest**

**ROC Curve Areas**

- **Best Area Under Curve: RandomForest**

**Overall, it seems that the best model to use for the Churn dataset would be the RandomForestClassifier model**