Credit Card Customer Retention
An Ensemble Learning Analysis

Table of Contents

Executive Summary

Introduction

A business manager of a consumer credit card portfolio is facing the problem of customer attrition. They want the data analyzed in order to find the reasons for this and to better predict customers who are likely to leave.

In this project, we will work to find valuable information that the portfolio manager can use in order to retain customers. To achieve this objective, the underlying factors driving customer attrition will be investigated. This will involve employing various techniques, including data analysis and feature engineering. Subsequently, ensemble learning models will be used to model the data, proactively identifying potential churned customers while gaining insights into the reasons behind customer attrition. The primary question to be answered as a result of this project is:

- What are the characteristics of churning customers?

Methodology

Data Cleaning

The data wrangling process in this project was relatively brief. It included checking for nulls and duplicates, string manipulations, and removing a few unknown values. After the unknown values were removed, the class balance was identified for the singular dataset used throughout this document.

Data Analysis

The data analysis section began by generating summary statistics followed by a couple of grouped aggregate calculations to identify the measures of central tendency grouped by attrition for a couple of variables. This provided information regarding some potential characteristics that separate retained and attrited customers. Afterwards, data visualizations were created in order to visually identify some differences and similarities between the two customer groups.

Feature Engineering

A concise feature engineering section involved creating a feature for the average amount spent per transaction per customer as well as performing dummy encoding for the categorical response variable.

Data Modeling

In this project, the data modeling process involved creating two separate models using ensemble learning techniques. Both the random forest and gradient boosting machine were implemented using cross-validation in order to identify the best hyperparameters. After these models were trained, they were compared using four metrics: accuracy, recall, precision, and the F1 score. A champion model was chosen and used to predict on unseen data, and these results were used to evaluate the final model performance and to gain insights about the data.

Results

Model Performance

The gradient boosting machine performed better than the random forest in every metric. With XGBoost, the F1 score improved by approximately 3%, and the recall saw an increase of around 3.6% over the random forest. The cross-validation scores for the XGBoost model are as follows:

- Accuracy: 96.70%
- Recall: 0.9156
- Precision: 0.8706
- F1 Score: 0.8924

Feature Importance

The most influential features regarding customer attrition are the ones relevant to product engagement such as the total number of transactions that they made and their change in transaction count from one quarter to the next. The most important variable is how much they've spent in total as a credit card customer. Because of this, it has been concluded that customer retention may be improved by inspiring increased product engagement.

Strategic Recommendations

Through an in-depth analysis of the data, I was able to successfully obtain valuable information regarding credit card customer behavior and develop a predictive model. The most relevant insights to the business problem are that the customers who are leaving are the ones who:

- Make fewer total transactions
- Have fewer total transaction amounts
- Spend less per transaction
- Use their credit cards less over time

Based on this information, I have curated the following list of strategic recommendations:

1. Tiered Benefits
    - Introduce tiered benefits or loyalty rewards based on card usage. Offer additional rewards, perks, or higher cashback rates for reaching specific spending thresholds.
2. Spending Challenges
    - Develop a rewards system with gamified elements designed to encourage increased card usage. Offer challenges, levels, or surprise rewards tied to specific spending milestones, creating an engaging experience for cardholders.
3. Segmented Marketing
    - Focus marketing efforts towards customers displaying reduced spending or infrequent transactions. Create strategies to encourage card usage tailored specifically for these customers.

These recommendations are designed to improve credit card customer retention by increasing product engagement, utilizing data-driven insights derived from data analysis and machine learning models.

Data Description

- Back to Table of Contents

In this dataset, there are 10127 rows, 23 columns, and the following variables:

Variable	Description
CLIENTNUM	Client number - Unique identifier for the customer holding the account
Attrition_Flag	Denotes if the customer's account is closed
Customer_Age	Demographic variable - Customer's age in years
Gender	Demographic variable - Customer's gender (M=Male or F=Female)
Dependent_count	Demographic variable - Number of dependents
Education_Level	Demographic variable - Customer's level of education attained
Marital_Status	Demographic variable - Customer's marital status
Income_Category	Demographic variable - Annual income category of the account holder
Card_Category	Product variable - Type of card (Blue, Silver, Gold, Platinum)
Months_on_book	Period of relationship with bank
Total_Relationship_Count	Total number of products held by the customer
Months_Inactive_12_mon	Number of months inactive in the last 12 months
Contacts_Count_12_mon	Number of contacts in the last 12 months
Credit_Limit	Credit limit on the credit card
Total_Revolving_Bal	Total revolving balance on the credit card
Avg_Open_To_Buy	Open to buy credit line (average of the last 12 months)
Total_Amg_Chng_Q4_Q1	Change in transaction amount (Q4 over Q1)
Total_Trans_Amt	Total transaction amount (last 12 months)
Total_Trans_Ct	Total transaction count (last 12 months)
Total_Ct_Chng_Q4_Q1	Change in transaction count (Q4 over Q1)
Avg_Utilization_Ratio	Average card utilization ratio

Import Statements

- Back to Table of Contents

In the next few lines, the relevant libraries and dataset will be imported.

In [ ]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import numpy as np
import pandas as pd

# Set Pandas to display all columns in output
pd.set_option('display.max_columns', None)

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import precision_score, recall_score, accuracy_score, f
confusion_matrix, ConfusionMatrixDisplay

from xgboost import XGBClassifier
from xgboost import plot_importance
```

In [ ]:
```
# Import data
df0 = pd.read_csv('BankChurners.csv')

# Preview dataframe
df0.head()
```

Out[ ]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level
0	768805383	Existing Customer	45	M	3	High School
1	818770008	Existing Customer	49	F	5	Graduate
2	713982108	Existing Customer	51	M	3	Graduate
3	769911858	Existing Customer	40	F	4	High School
4	709106358	Existing Customer	40	M	3	Uneducated

Data Cleaning

- Back to Table of Contents

The individual who posted this dataset mentioned that the final two columns were added by mistake, so I will begin by dropping these columns first.

In [ ]:

```
# Drop the final two columns
df0.drop(df0.columns[-2:], axis=1, inplace=True)
```

Next, the data cleaning process will be continued by checking for missing values and duplicated entries.

In [ ]:
```python
# Print the number of missing values for each column
df0.isna().sum()
```

Out[ ]:
```
CLIENTNUM                   0
Attrition_Flag              0
Customer_Age                0
Gender                      0
Dependent_count             0
Education_Level             0
Marital_Status              0
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
```

In [ ]:
```python
# Print the total number of duplicated entries
df0.duplicated().sum()
```

Out[ ]:
```
0
```

It is now confirmed that the dataset has neither any missing values nor any duplicates.

Continuing the data cleaning process, I will drop the `'CLIENTNUM'` column as it will not provide any useful information going forward. I will also exclude the `'Gender'` column due to ethical considerations, ensuring our model avoids making predictions influenced by gender.

In [ ]:
```python
# Drop the 'CLIENTNUM' and 'Gender' columns
df0.drop(['CLIENTNUM', 'Gender'], axis=1, inplace=True)
```

Next, I'll standardize the column names in the DataFrame to snake case before proceeding with further analysis. This ensures consistent, readable, and standard naming throughout the columns in the DataFrame. Because the dataset already includes underscores in the column names, I will only need to change the capital letters to lowercase letters.

In [ ]:
```python
# Change the uppercase letters to lowercase
df0.columns = df0.columns.str.lower()
df0.head()
```

Out[ ]:

	attrition_flag	customer_age	dependent_count	education_level	marital_status	income_ca
0	Existing Customer	45	3	High School	Married	60.
1	Existing Customer	49	5	Graduate	Single	Less tha
2	Existing Customer	51	3	Graduate	Married	80K
3	Existing Customer	40	4	High School	Unknown	Less tha
4	Existing Customer	40	3	Uneducated	Married	60.

While reviewing the data on Kaggle, I could see that some columns contain the value `'Unknown'`. Now, I'll confirm that these observations exist in the DataFrame.

In [ ]:
```
# Check 'education_level' for unknown values
print(df0['education_level'].unique())
```

```
['High School' 'Graduate' 'Uneducated' 'Unknown' 'College' 'Post-Graduate'
 'Doctorate']
```

The output of the cell above confirms that the dataset has the string `'Unknown'`. Next, I'll check all columns for this particular string value and output the column names which contain `'Unknown'`.

In [ ]:
```
# Check all columns for 'Unknown'
for column_name in df0.columns:
    has_unknown = (df0[column_name]=='Unknown').any()
    if has_unknown:
        print(f'{column_name} contains unknown values: {has_unknown}')
    else:
        pass
```

```
education_level contains unknown values: True
marital_status contains unknown values: True
income_category contains unknown values: True
```

At this point in the data cleaning process, I am considering dropping the observations which have unknown values so that `'Unknown'` is not considered as a category in the models. However, before dropping these rows, I'm going to ensure that doing this won't significantly affect the original class balance or drop too many data points.

To do this, I will first check the class balance ratio for the response variable: `'attrition_flag'` (recall that the column name has been changed to lowercase).

In [ ]:
```
# Get the percentage of existing vs attrited customers
df0['attrition_flag'].value_counts(normalize=True)
```

Out[ ]:
```
attrition_flag
Existing Customer    0.83934
Attrited Customer    0.16066
Name: proportion, dtype: float64
```

In [ ]:
```
# Get the number of existing and attrited customers
df0['attrition_flag'].value_counts()
```

Out[ ]:
```
attrition_flag
Existing Customer    8500
Attrited Customer    1627
Name: count, dtype: int64
```

As can be seen, this dataset has an imbalance between the classes, with a ratio of approximately 84% to 16% for the majority and minority classes respectively. Despite this imbalance, further examination reveals that the dataset consists of 8,500 observations in the majority class and 1,627 observations in the minority class.

While acknowledging the presence of a class imbalance, the minority class has 1,627 observations which offers a sizable quantity of data to draw insights from and to potentially develop strategies to address the imbalance.

Given the relatively substantial number of observations available for both classes, I'll continue to remove the observations in the dataset that contain the string `'Unknown'` and remain aware of the class imbalance's potential impact on modeling and evaluation. Now, I'll drop those observations from these three columns and save as a new DataFrame, and then re-examine the class balance and number of observations.

In [ ]:
```
# Assign to a variable a list of indices in the DataFrame where there is an
rows_to_drop = df0[df0[['education_level', 'marital_status', 'income_categor

# Drop rows with 'Unknown'
df1 = df0.drop(index=rows_to_drop).reset_index(drop=True)
```

Checking the class balance and number of observations in the new dataset.

In [ ]:
```
# Get the percentage of existing vs attrited customers
print(df1['attrition_flag'].value_counts(normalize=True))

# Get the number of existing and attrited customers
print(df1['attrition_flag'].value_counts())
```

```
attrition_flag
Existing Customer    0.842819
Attrited Customer    0.157181
Name: proportion, dtype: float64
attrition_flag
Existing Customer    5968
Attrited Customer    1113
Name: count, dtype: int64
```

The size of the majority class has been reduced by 2,532 observations and the minority class by 514. Despite this change, the balance between the classes has stayed fairly consistent, and there is still ample data for analysis and modeling purposes.

Since a random forest and gradient boosting machine will be used for predictive modeling, I'll forego checking for and dealing with outliers in this project.

Data Analysis

- Back to Table of Contents

In [ ]:
```
# Display data summary
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7081 entries, 0 to 7080
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   attrition_flag            7081 non-null   object 
 1   customer_age              7081 non-null   int64  
 2   dependent_count           7081 non-null   int64  
 3   education_level           7081 non-null   object 
 4   marital_status            7081 non-null   object 
 5   income_category           7081 non-null   object 
 6   card_category             7081 non-null   object 
 7   months_on_book            7081 non-null   int64  
 8   total_relationship_count  7081 non-null   int64  
 9   months_inactive_12_mon    7081 non-null   int64  
 10  contacts_count_12_mon     7081 non-null   int64  
 11  credit_limit              7081 non-null   float64
 12  total_revolving_bal       7081 non-null   int64  
 13  avg_open_to_buy           7081 non-null   float64
 14  total_amt_chng_q4_q1      7081 non-null   float64
 15  total_trans_amt           7081 non-null   int64  
 16  total_trans_ct            7081 non-null   int64  
 17  total_ct_chng_q4_q1       7081 non-null   float64
 18  avg_utilization_ratio     7081 non-null   float64
dtypes: float64(5), int64(9), object(5)
memory usage: 1.0+ MB
```

After data cleaning, the dataset has decreased from 10,127 rows and 23 columns to 7,081 rows and 19 columns. Additionally, it can be seen that the response variable as well as many of the demographic variables are categorical.

In [ ]:
```
# Generate descriptive statistics for quantitative variables
df1.describe()
```

Out[ ]:

	customer_age	dependent_count	months_on_book	total_relationship_count	months_i
count	7081.000000	7081.000000	7081.000000	7081.000000	
mean	46.347691	2.337805	35.981359	3.819376	
std	8.041225	1.291649	8.002609	1.544444	
min	26.000000	0.000000	13.000000	1.000000	
25%	41.000000	1.000000	31.000000	3.000000	
50%	46.000000	2.000000	36.000000	4.000000	
75%	52.000000	3.000000	40.000000	5.000000	
max	73.000000	5.000000	56.000000	6.000000	

I would like to view some of the data grouped by `'attrition_flag'` in order to better understand the drivers behind customer churn. So now, I'll continue working with these quantitative variables to explore whether or not there are significant differences between the two types of customers.

In [ ]:
```
# Display total revolving balance grouped by customer attrition
grouped_df0 = df1.groupby('attrition_flag')['total_revolving_bal'].agg(['mea
grouped_df0
```

Out[ ]:

	mean	median
attrition_flag		
Attrited Customer	668.353998	0.0
Existing Customer	1260.589980	1365.0

In [ ]:
```
# Display average utilization ratio grouped by customer attrition
grouped_df1 = df1.groupby('attrition_flag')['avg_utilization_ratio'].agg(['m
grouped_df1
```

Out[ ]:

	mean	median
attrition_flag		
Attrited Customer	0.163571	0.0000
Existing Customer	0.304458	0.2235

From the two lines above, I'm starting to get the idea that the customers who are leaving are ones who are less engaged with the credit card. For now, I'll refrain from drawing any conclusions at this point and continue to perform further analysis on the data.

Now, I'll create some visualizations to better understand the dataset.

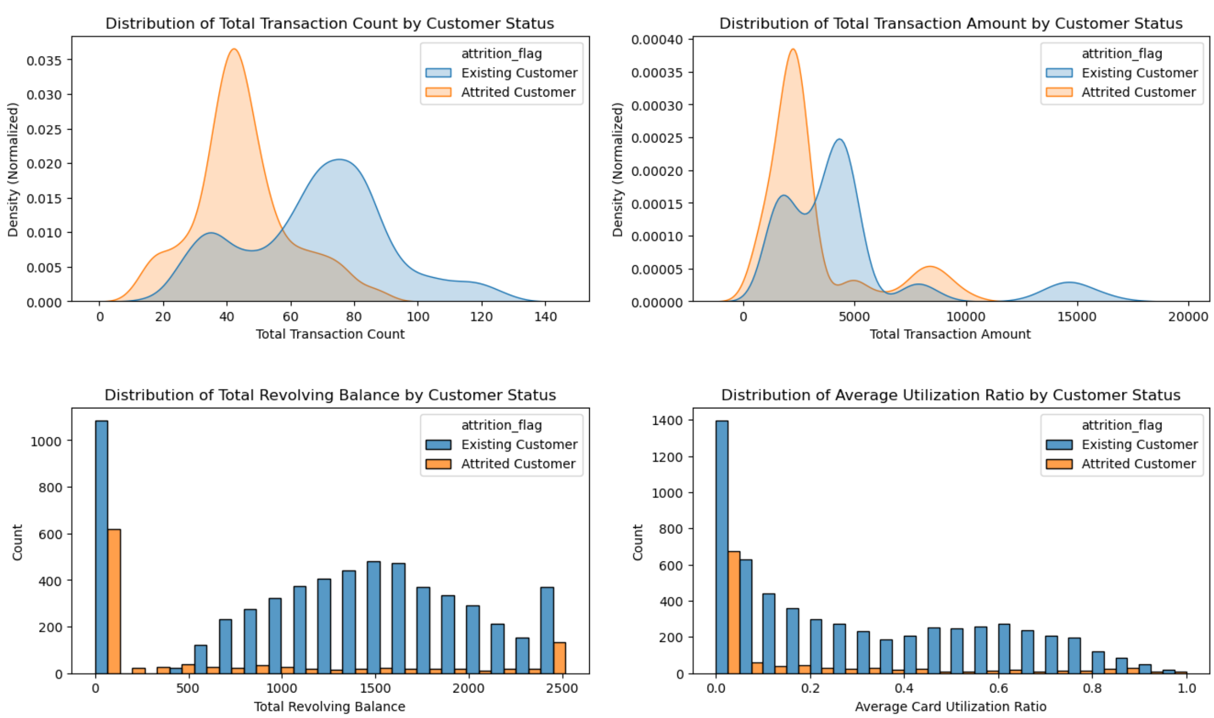
Data Visualizations

In this section, I'll aim to explore the factors relevant to product engagement by focusing on the quantitative variables in the dataset. I'll begin by visualizing the distributions the following variables grouped by customer attrition: `'total_trans_ct'`, `'total_trans_amt'`, `'total_revolving_bal'`, and `'avg_utilization_ratio'`.

Important note: This dataset exhibits a class imbalance (existing vs. attrited customers). In the kernel density estimate (KDE) plots below, this imbalance has been normalized by using `common_norm=False`. However, the histograms are not normalized, so, in them, it's shown that there are less observations in total for attrited customers than existing customers.

In [ ]:
```
# Generate a figure with four subplots
fig, axs = plt.subplots(2, 2, figsize=(16, 9))
```

```
# Add vertical space between subplots for better readability
plt.subplots_adjust(hspace=0.4)

# Generate a density plot of total_trans_ct grouped by attrition_flag
sns.kdeplot(data=df1, x='total_trans_ct', hue='attrition_flag',
            fill=True, common_norm=False, ax=axs[0, 0])
axs[0, 0].set_title('Distribution of Total Transaction Count by Customer Sta
axs[0, 0].set_xlabel('Total Transaction Count')
axs[0, 0].set_ylabel('Density (Normalized)')

# Generate a density plot of total_trans_amt grouped by attrition_flag
sns.kdeplot(data=df1, x='total_trans_amt', hue='attrition_flag',
             fill=True, common_norm=False, ax=axs[0, 1])
axs[0, 1].set_title('Distribution of Total Transaction Amount by Customer St
axs[0, 1].set_xlabel('Total Transaction Amount')
axs[0, 1].set_ylabel('Density (Normalized)')

# Generate a histogram of total_revolving_bal grouped by attrition_flag
sns.histplot(data=df1, x='total_revolving_bal', hue='attrition_flag',
             multiple='dodge', ax=axs[1, 0])
axs[1, 0].set_title('Distribution of Total Revolving Balance by Customer Sta
axs[1, 0].set_xlabel('Total Revolving Balance')

# Generate a histogram of avg_utilization_ratio grouped by attrition_flag
sns.histplot(data=df1, x='avg_utilization_ratio', hue='attrition_flag',
             multiple='dodge', ax=axs[1, 1])
axs[1, 1].set_title('Distribution of Average Utilization Ratio by Customer S
axs[1, 1].set_xlabel('Average Card Utilization Ratio')

plt.show();
```



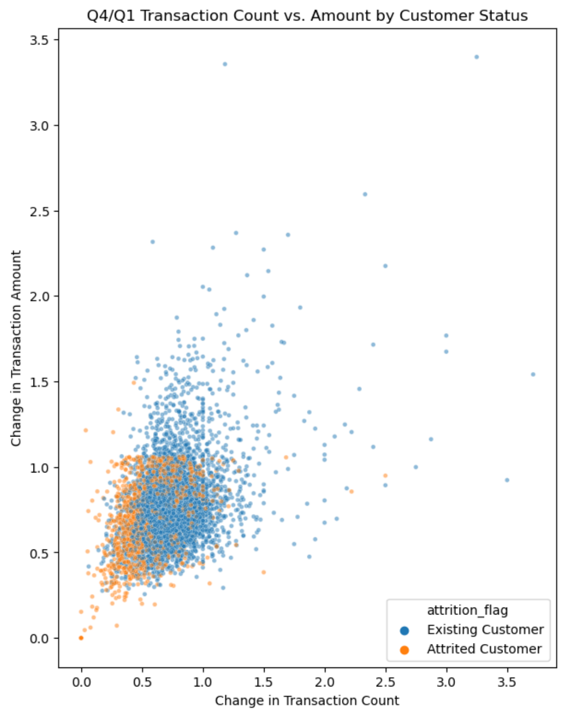
The figure above creates a visual representation of the distributions of a few of the variables in the dataset in conjunction with the differences between existing and attrited customers.

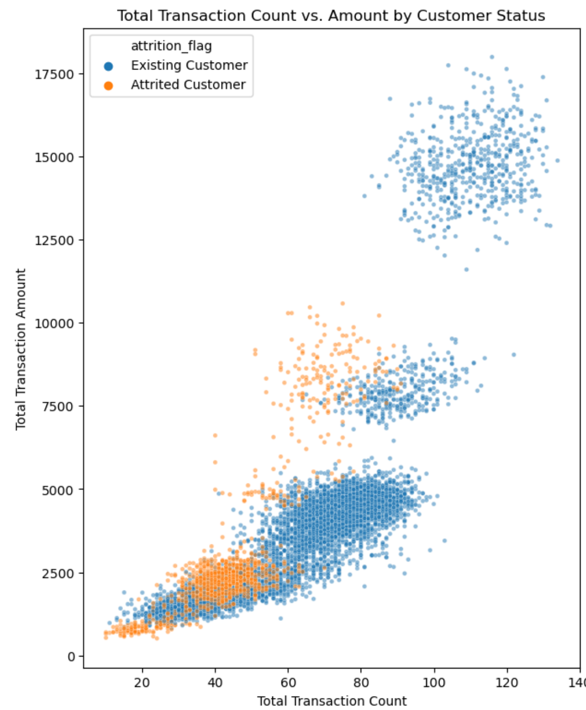
While some similarities can be observed between the two types of customers, more differences are also beginning to appear. Most notably that there are no churned customers with relatively high total transaction counts or total transaction amounts. This seems to suggest that which I began to suspect during exploratory data analysis: that the customers who are leaving are the ones who are less engaged with the credit card.

Next, I'll continue to explore the transaction counts and transaction amounts through visualizations. I will break these down into two scatterplots. There will be one for the *change* in transaction amounts vs. counts, and there will be another for the *total* transaction amounts vs. counts. Both of these scatterplots will use different colors depending on customer status.

In [ ]:

```
# Generate a figure with two subplots
fig, ax = plt.subplots(1, 2, figsize=(16, 9))

# Add horizontal space between subplots for better readability
plt.subplots_adjust(wspace=0.3)

# Scatterplot of the change in transaction amount by the change in transacti
sns.scatterplot(data=df1, x='total_ct_chng_q4_q1', y='total_amt_chng_q4_q1',
                s=12, alpha=0.5, ax=ax[0])
ax[0].set_title('Q4/Q1 Transaction Count vs. Amount by Customer Status')
ax[0].set_xlabel('Change in Transaction Count')
ax[0].set_ylabel('Change in Transaction Amount')

# Scatterplot of the total transaction amount by the total transaction count
sns.scatterplot(data=df1, x='total_trans_ct', y='total_trans_amt', hue='attr
                s=12, alpha=0.5, ax=ax[1])
ax[1].set_title('Total Transaction Count vs. Amount by Customer Status')
ax[1].set_xlabel('Total Transaction Count')
ax[1].set_ylabel('Total Transaction Amount')

plt.show();
```


Q4/Q1 Transaction Count vs. Amount by Customer Status
Change in Transaction Amount
Change in Transaction Count
attrition_flag
Existing Customer
Attrited Customer
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


Total Transaction Count vs. Amount by Customer Status
Total Transaction Amount
Total Transaction Count
attrition_flag
Existing Customer
Attrited Customer
0
2500
5000
7500
10000
12500
15000
17500
20
40
60
80
100
120
140

While the different distributions for existing and attrited customers with regard to their transaction behaviors are noticeable, it can also be seen that the customers can't be easily separated into two distinct groups based on these metrics alone. In order to draw more precise conclusions and gather more insights about the features in the dataset, let's continue the analysis.

Feature Engineering

- Back to Table of Contents

Before beginning the modeling process, the data needs to be prepared. First, I'll create a new feature that may be useful to the process called `'avg_amt_per_trans'`. This feature will be equal to `'total_trans_amt'` divided by `'total_trans_ct`.

In [ ]:
```
# Create a new column 'avg_amt_per_trans'
df1['avg_amt_per_trans'] = df1['total_trans_amt'] / df1['total_trans_ct']
```

In order to use the data for modeling, the categorical features must be encoded. For this purpose, I'll use the `get_dummies()` method in the pandas library. In the `get_dummies()` method, I'll set `drop_first=True` to reduce redundancy.

In [ ]:
```
# Dummy encode categorical features
df2 = pd.get_dummies(df1, dtype=int, drop_first=True)
```

In [ ]:
```
df2.rename(columns={'attrition_flag_Existing Customer': 'attrition_flag'}, i
```

Next, I'll ensure that, in the `'attrition_flag'` column, 0 is for an existing customer and

1 is for an attrited customer. Since it can be seen in an earlier DataFrame that the first index is that of an existing customer, I'll use a simple "if" statement to switch these two integers if they are in the incorrect place.

In [ ]:
```
if df2['attrition_flag'].iloc[0] == 1:
    df2['attrition_flag'] = df2['attrition_flag'].replace({0: 1, 1: 0})
else:
    pass

# Ensure that the first few rows of 'attrition_flag' are 0s (1 means attrite
df2.head()
```

Out[ ]:

	customer_age	dependent_count	months_on_book	total_relationship_count	months_inact
0	45	3	39	5	
1	49	5	44	6	
2	51	3	36	4	
3	40	3	21	5	
4	44	2	36	3	

Data Modeling

- Back to Table of Contents

In this data modeling section, I will create two predictive models using ensemble learning methods. 75% of the available data will be used cross-validate the models using Scikit-learn's `GridSearchCV` class. After the models are fit to the training data, they will be scored based on their cross-validation results using four metrics: accuracy, precision, recall, and the F1 score. Then, they will be compared, and a champion model will be used to predict on the unseen test data. The feature importances reported by the final model are the ones which will be used in order to make strategic, data-driven recommendations.

Contents: Data Modeling

Data Modeling I: Random Forest

- Back to Data Modeling Contents

Splitting the Data

Now that the data is cleaned and the categorical variables are encoded, the data will be split into training and testing sets. I'll set aside 25% of the data for the test set. Note that the `stratify=y` parameter is included in scikit-learn's `train_test_split()` method in order to maintain the imbalanced class ratio in both the training and test sets.

In [ ]:
```
# Define the target variable
y = df2['attrition_flag']

# Define the predictor variables
X = df2.copy()
X = X.drop('attrition_flag', axis=1)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    stratify=y, random_state
```

Model Instantiation and Hyperparameter Tuning

First, I'll instantiate the random forest classifier and assign it to the variable `rf`. Subsequently, a dictionary of hyperparameters will be defined (`cv_params`) for hyperparameter tuning. Accuracy, precision, recall, and F1 scores will be used to evaluate the cross-validation models. I'll be using the F1 score to refit the model, aiming to strike a balance between precision and recall.

In [ ]:

```
# Instantiate the model
rf = RandomForestClassifier(random_state=0)

# Specify hyperparameters
cv_params = {'max_depth': [10, 15, None],
             'max_features': ['sqrt', None],
             'min_samples_leaf': [1, 2, 3],
             'min_samples_split': [2, 3, 4],
             'n_estimators': [75, 100, 125, 150, 200]
             }

# Define scoring metrics for evaluation
scoring = ['accuracy', 'precision', 'recall', 'f1']

# Hyperparameter tuning with cross-validation
rf_cv = GridSearchCV(rf, cv_params, scoring=scoring, n_jobs=-1, cv=5, refit=
```

Model Fitting and Scoring

CPU: Ryzen 9 5950x

In [ ]:

```
%%time

# Fit the model using training data
rf_cv.fit(X_train, y_train)
```

```
CPU times: user 5.42 s, sys: 1.05 s, total: 6.48 s
Wall time: 1min 28s
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [ ]:

```
# Display best parameters and corresponding F1 score
print(f'Best Parameters: {rf_cv.best_params_} \nBest F1 Score: {rf_cv.best_s
```

```
Best Parameters: {'max_depth': None, 'max_features': None, 'min_samples_leaf
': 1, 'min_samples_split': 3, 'n_estimators': 125}
Best F1 Score: 0.8625993676603432
```

In the upcoming step, I'll generate a Pandas DataFrame to examine the performance metrics of the best estimator obtained from a `GridSearchCV` object. To do this, I'll create a function called `cv_scores`. This function will extract scoring metrics from the best estimator's results, convert them into a DataFrame, and return the DataFrame containing the model's performance metrics.

In [ ]:

```
def cv_scores(model_name: str, model_object):
    '''
    Accepts as arguments a custom model name (string) and a fit GridSearchCV

    Returns a pandas df with the F1, recall, precision, and accuracy scores
    model with the best mean F1 score across all validation folds.
```

```
    '''

    # Convert cross-validation results to a dataframe
    cv_results = pd.DataFrame(model_object.cv_results_ )

    # Select the best estimator based on the highest mean test F1 score
    best_estimator = cv_results.iloc[cv_results['mean_test_f1'].idxmax(), :]

    # Create variables for the scoring values to be used when constructing t
    accuracy = best_estimator.mean_test_accuracy
    precision = best_estimator.mean_test_precision
    recall = best_estimator.mean_test_recall
    f1 = best_estimator.mean_test_f1

    # Create a dataframe with cross-validation performance metrics
    table = pd.DataFrame({'Model': [model_name],
                          'Accuracy': [accuracy],
                          'Precision': [precision],
                          'Recall': [recall],
                          'F1': [f1]
                          })

    return table
```

In [ ]:
```
# Generate a dataframe with random forest cross-validation results
rf_cv_results = cv_scores('Random Forest CV', rf_cv)
rf_cv_results
```

Out[ ]:

	Model	Accuracy	Precision	Recall	F1
0	Random Forest CV	0.958192	0.892561	0.834731	0.862599

These metrics suggest that the model demonstrates high accuracy in identifying churn cases. Because the model is fit to an unbalanced dataset, I'll primarily focus our attention on the other scoring metrics. The precision and recall scores indicate that the model is doing well at predicting positive churn instances as well as correctly reporting actual churn instances. Additionally, the F1 score of 86.26% confirms that the model is performing well across both precision and recall measurements.

Next, I'm going to compare the cross-validation performance of a gradient boosting model to the random forest model.

Data Modeling II: XGBoost

- Back to Data Modeling Contents

Model Instantiation and Hyperparameter Tuning

Similar to what was done with the random forest classifier, I'll initiate the classifier and assign it to a variable. Then, I'll define a new dictionary of hyperparameters. The same scoring metrics will be used as in the random forest model, and this one will also be cross-validated using scikit-learn's `'GridSearchCV'` class.

In [ ]:
```
# Instantiate the model
xgb = XGBClassifier(objective='binary:logistic', random_state=42)

# Specify hyperparameters
cv_params = {'max_depth': [4, 5, 6, 7, 8, None],
             'min_child_weight': [1, 2, 3, 4, 5],
             'learning_rate': [0.1, 0.2, 0.3],
             'n_estimators': [75, 100, 125, 150, 200]
             }

# Define scoring metrics for evaluation
scoring = ['accuracy', 'precision', 'recall', 'f1']

# Hyperparamter tuning with cross-validation
xgb_cv = GridSearchCV(xgb, cv_params, scoring=scoring, cv=5, n_jobs=-1, refi
```

Model Fitting and Scoring

CPU: Ryzen 9 5950x

In [ ]:
```
%%time

# Fit the model using training data
xgb_cv.fit(X_train, y_train)
```

```
CPU times: user 8.21 s, sys: 1.27 s, total: 9.48 s
Wall time: 1min 20s
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: XGBClassifier
  ▸ XGBClassifier
```

In [ ]:
```
# Display best parameters and corresponding F1 score
print(f'Best Parameters: {xgb_cv.best_params_} \nBest F1 Score: {xgb_cv.best
```

```
Best Parameters: {'learning_rate': 0.3, 'max_depth': 5, 'min_child_weight':
1, 'n_estimators': 75}
Best F1 Score: 0.8924947081986311
```

In [ ]:
```
# Generate a dataframe with xgboost cross-validation results
xgb_cv_results = cv_scores('XGBoost CV', xgb_cv)
xgb_cv_results
```

Out[ ]:

	Model	Accuracy	Precision	Recall	F1
0	XGBoost CV	0.967043	0.915613	0.870659	0.892495

Like the random forest model above, this model is performing acceptably in all metrics. Because the dataset is imbalanced, less of an importance will be placed on accuracy. Additionally, this model seems to be performing better than the random forest model. I'll expand upon this in the next section.

Model Comparison and Performance Evaluation

- Back to Data Modeling Contents

To compare the performances of the two models, I'll create a table of the results.

In [ ]:
```
# Concatenate xgboost cv results to random forest cv results
full_results = pd.concat([rf_cv_results, xgb_cv_results], axis=0).sort_value
full_results
```

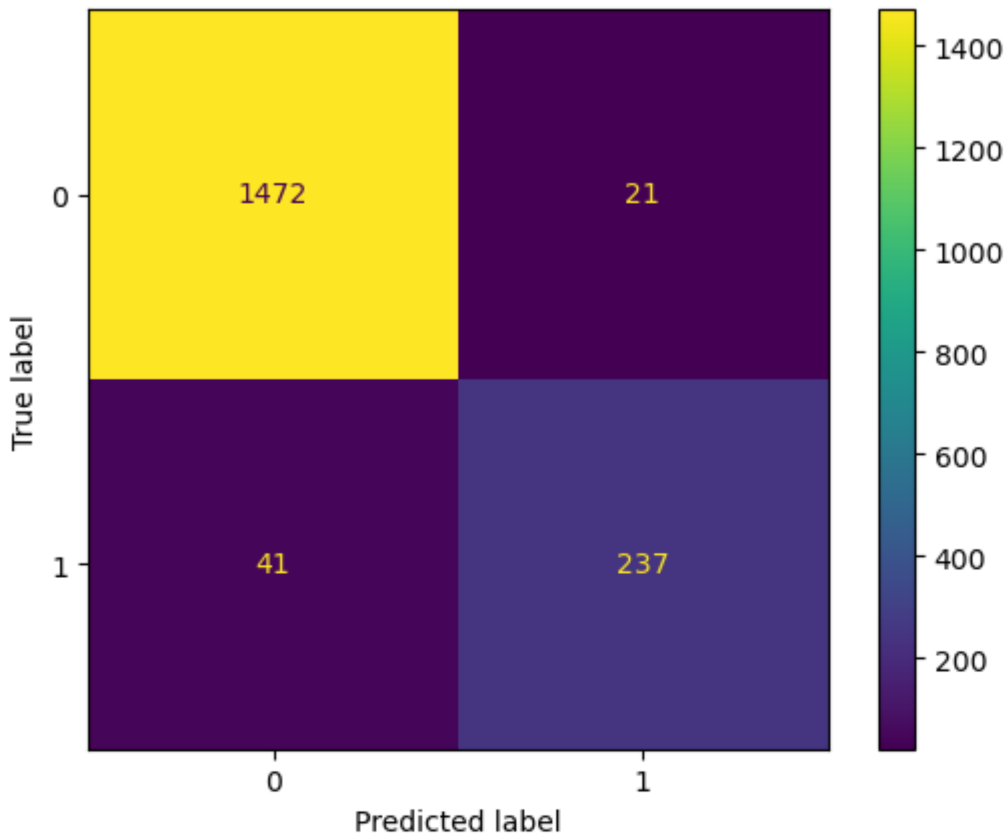
Out[ ]:

	Model	Accuracy	Precision	Recall	F1
0	XGBoost CV	0.967043	0.915613	0.870659	0.892495
0	Random Forest CV	0.958192	0.892561	0.834731	0.862599

The gradient boosting model outperformed the random forest in every metric. The F1 score improved by approximately 3%, and the recall saw an increase of around 3.6%. This suggests a noteworthy decrease in incorrectly predicting that a customer would not churn.

Due to its superior performance with the dataset, I'll select the XGBoost model to be the champion model. It will be used to predict on the test data and generate a confusion matrix so that can be used to visualize the results.

In [ ]:
```
# Use XGBoost to model to predict on test data
preds = xgb_cv.best_estimator_.predict(X_test)

# Create confusion matrix using predicted and actual values
cm = confusion_matrix(y_test, preds, labels=xgb_cv.classes_)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=xgb_cv.cla
disp.plot(values_format='');
```



The model generates almost twice as many false negatives as false positives. It is likely that this is primarily due to the imbalance in the data, namely that there were much more existing customers than attrited customers. Despite this, it is an improvement over the random forest, and we can also conclude that it is a well-performing model for our purposes.

In the next section, I'll focus on the insights that can be derived from the model, and then move on to identifying actionable solutions that can be recommended to the portfolio manager.

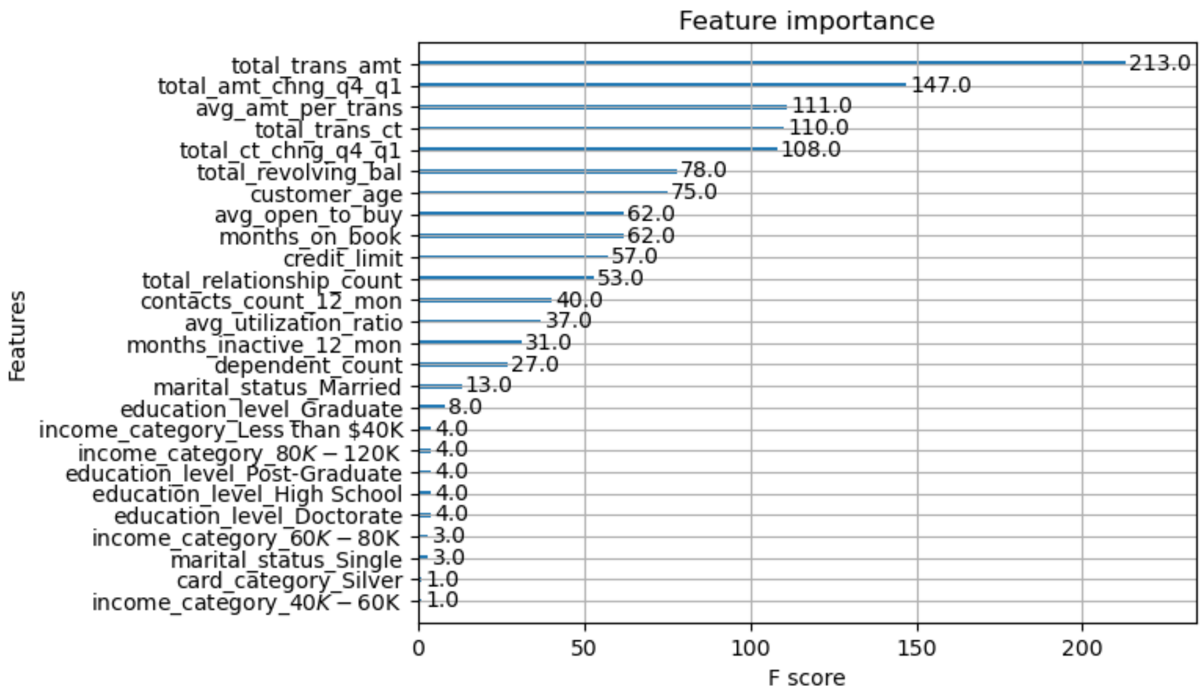
Feature Importance

- Back to Data Modeling Contents

At this point, I'll get the feature importances for the best-performing model.

In [ ]:
```
# Plot feature importances for XGBoost
plot_importance(xgb_cv.best_estimator_)
```

Out[ ]:
```
<Axes: title={'center': 'Feature importance'}, xlabel='F score', ylabel='Fe
atures'>
```



As suspected from the Data Analysis section, the most influential features are the ones relevant to product engagement such as `'total_trans_amt'` and `'total_trans_ct'`. Despite seeing a notable difference in the average and median `'avg_utilization_ratio'` between existing and attrited customers, it doesn't seem to have as much of an effect as might have been suggested during EDA. According to the cross-validated XGBoost model, the most important demographic variable by far is `'customer_age'`, so it may be beneficial to investigate this further in order to make specific recommendations regarding age-based strategies.

At this point, we can take the top 5 features by importance to definitively conclude that that the customers who are leaving are the ones who are spending less and making fewer transactions with the credit card. Additionally, decreasing credit card purchasing habits from one quarter to the next are also indicative of a potential future attrited customer.

This marks the end of this project. A presentation of the results as well as data-driven recommendations are in the Executive Summary at the beginning of this document.
