رواد مصر الرقمية

وزارة الاتصالات وتكنولوجيا المعلومات

5 Stars

predicting tomorrow ,today

Card Churn Prediction

This project was implemented by:

Mahmoud Saad Mahmoud (Team Leader)

zeyad ahmed mostafa **Mohamed Badr Mohamed**

Alaa Ghalib Othman **Ahmed Mohamed Tawfik**

Table of Content

1.Abstract

Customer churn prediction is crucial in customer retention strategies across various industries. In this project, we aim to build a model that predicts customer churn based on transactional and demographic data from a bank. Using machine learning algorithms and exploratory data analysis (EDA), we analyze customer behaviors and develop predictive models to identify those at risk of churning. By implementing models like Random Forest, Gradient Boosting, and Logistic Regression, we provide actionable insights into customer retention. The results demonstrate that machine learning can significantly improve churn prediction accuracy, helping banks retain valuable customers.

2. Introduction

Overview of the Project

This project focuses on predicting customer churn for a credit card company. Customer churn refers to the phenomenon where customers discontinue their relationship with a business. Identifying customers at risk of churning enables organizations to implement targeted retention strategies, thereby minimizing potential revenue loss.

Purpose and Goals

The primary objective of this project is to develop a predictive model capable of accurately identifying customers at risk of churn. To achieve this, the project will encompass the following key steps:

- Collecting and preprocessing relevant data
- Conducting exploratory data analysis to extract meaningful insights
- Implementing a variety of machine learning models
- Evaluating and comparing the performance of these models
- Providing actionable recommendations based on the analysis

Relevance and Applications

Predictive analytics for customer churn is essential, particularly in competitive sectors such as finance and telecommunications. By effectively predicting churn, companies can:

- Enhance customer retention strategies
- Improve customer satisfaction and loyalty
- Boost overall revenue and profitability

Background

In highly competitive industries like banking, customer retention is crucial for long-term business sustainability. Customer churn when a client ceases to use a product or service can significantly affect revenue and growth. By predicting churn, businesses can implement proactive measures to improve customer retention rates and mitigate financial losses. This project specifically targets credit card churn prediction within a banking institution, leveraging customer data to construct predictive models utilizing machine learning techniques.

Problem Statement

Churn prediction involves identifying customers who are likely to leave the service based on historical data. This project seeks to answer the question: How can machine learning models be employed to predict cardholder churn with high accuracy?

3. Literature Review

Summary of Existing Methods and Models for Card Churn Prediction

Customer churn prediction has leveraged a variety of machine learning algorithms, including logistic regression, decision trees, random forests, and gradient boosting machines. Recent developments also incorporate deep learning techniques, such as artificial neural networks (ANNs) and convolutional neural networks (CNNs).

Comparison of Different Approaches

Comparative studies indicate that while simpler models like logistic regression offer greater interpretability, more complex models, such as random forests and gradient boosting, tend to deliver superior predictive performance. Additionally, deep learning methods are increasingly popular for their capacity to identify intricate patterns within data.

Identification of Gaps and Areas for Improvement

Despite these advancements, challenges remain in achieving a balance between model accuracy and interpretability. Furthermore, there is an ongoing need to refine data preprocessing techniques and feature engineering to enhance overall model performance.

Review of Relevant Literature

Customer churn prediction has been extensively examined across various industries, including telecommunications, banking, and subscription-based services. A wide array of machine learning algorithms has been employed, from traditional logistic regression to more advanced ensemble methods like random forests and gradient boosting.

- Kumar et al. (2020): This study applies random forests and decision trees for churn prediction in the telecom sector. The authors conclude that random forests outperform other algorithms in terms of prediction accuracy due to their effectiveness in managing non-linear relationships within the data.
- Chen et al. (2019): This research utilizes gradient boosting machines (GBM) for predicting customer churn in the banking industry. The model's high accuracy and interpretability render it a strong candidate for financial institutions.
- Yin et al. (2021): This paper investigates the application of neural networks for churn prediction in subscription services. While neural networks achieve higher accuracy, they are also more computationally intensive and less interpretable than simpler models like logistic regression and decision trees.

4. Data Collection and Preprocessing

Source

The dataset was obtained from Kaggle, a popular platform for sharing and exploring public datasets.

Link of dataset : **https://www.kaggle.com/datasets/anwarsan/credit-card-bank-churn**

Description of the Dataset

The dataset used in this project, credit_card_churn.csv, contains 10,127 rows and 21 features related to customer demographics, credit card usage, and engagement. The key target variable is the Attrition_Flag, which indicates whether a customer has churned or not.

Key Features:

- Customer_Age: The age of the customer.
- Gender: The gender of the customer (M or F).
- Dependent_count: The number of dependents that a customer has.
- Education_Level: The educational qualification of the customer.
- Marital_Status: The marital status of the customer (Married, Single, Divorced).
- Income_Category: The income category of the customer (e.g., Less than $40K, $40K-$60K, $60K-$80K).
- Card_Category: The type of credit card (Blue, Gold, Silver, Platinum).
- Months_on_book: The number of months the customer has had their account open.
- Total_Relationship_Count: The total number of products held by the customer.
- Months_Inactive_12_mon: The number of months the customer has been inactive in the last 12 months.
- Contacts_Count_12_mon: The number of contacts with the customer in the last 12 months.
- Credit_Limit: The credit limit on the customer’s card.
- Total_Revolving_Bal: The total revolving balance on the customer’s credit card.
- Avg_Open_To_Buy: The average open-to-buy credit available on the credit card.
- Total_Amt_Chng_Q4_Q1: The change in the total transaction amount between Q4 and Q1.
- Total_Trans_Amt: The total transaction amount in the last 12 months.
- Total_Trans_Ct: The total number of transactions in the last 12 months.
- Total_Ct_Chng_Q4_Q1: The change in the total transaction count between Q4 and Q1.
- Avg_Utilization_Ratio: The average credit card utilization ratio (percentage of the credit limit used).

Steps for Data Cleaning and Preprocessing

Preprocessing steps performed before modeling:

1. Handling Missing Values: Visualizing missing data using missingno and filling missing values accordingly.
2. Encoding Categorical Variables: Using OneHotEncoder and LabelEncoder to convert categorical features like Gender, Education_Level, Marital_Status, etc., into numerical format.
3. Scaling: Normalizing numerical features such as Credit_Limit and Avg_Utilization_Ratio using StandardScaler to ensure they are on the same scale.

Scaling Numerical Features

```
# Columns to scale
columns_to_scale = ['Customer_Age', 'Dependent_count', 'Months_on_book', 'Total_Relationship_Count',
                    'Months_Inactive_12_mon', 'Contacts_Count_12_mon', 'Credit_Limit',
                    'Total_Revolving_Bal', 'Avg_Open_To_Buy','Total_Amt_Chng_Q4_Q1',
                    'Total_Trans_Amt', 'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio']

# Initialize the StandardScaler
scaler = StandardScaler()

# Apply Standard Scaling to training and testing data
X_train[columns_to_scale] = scaler.fit_transform(X_train[columns_to_scale])
X_test[columns_to_scale] = scaler.transform(X_test[columns_to_scale])

# Save them for later use during deployment
joblib.dump(scaler, 'scaler.pkl')
```

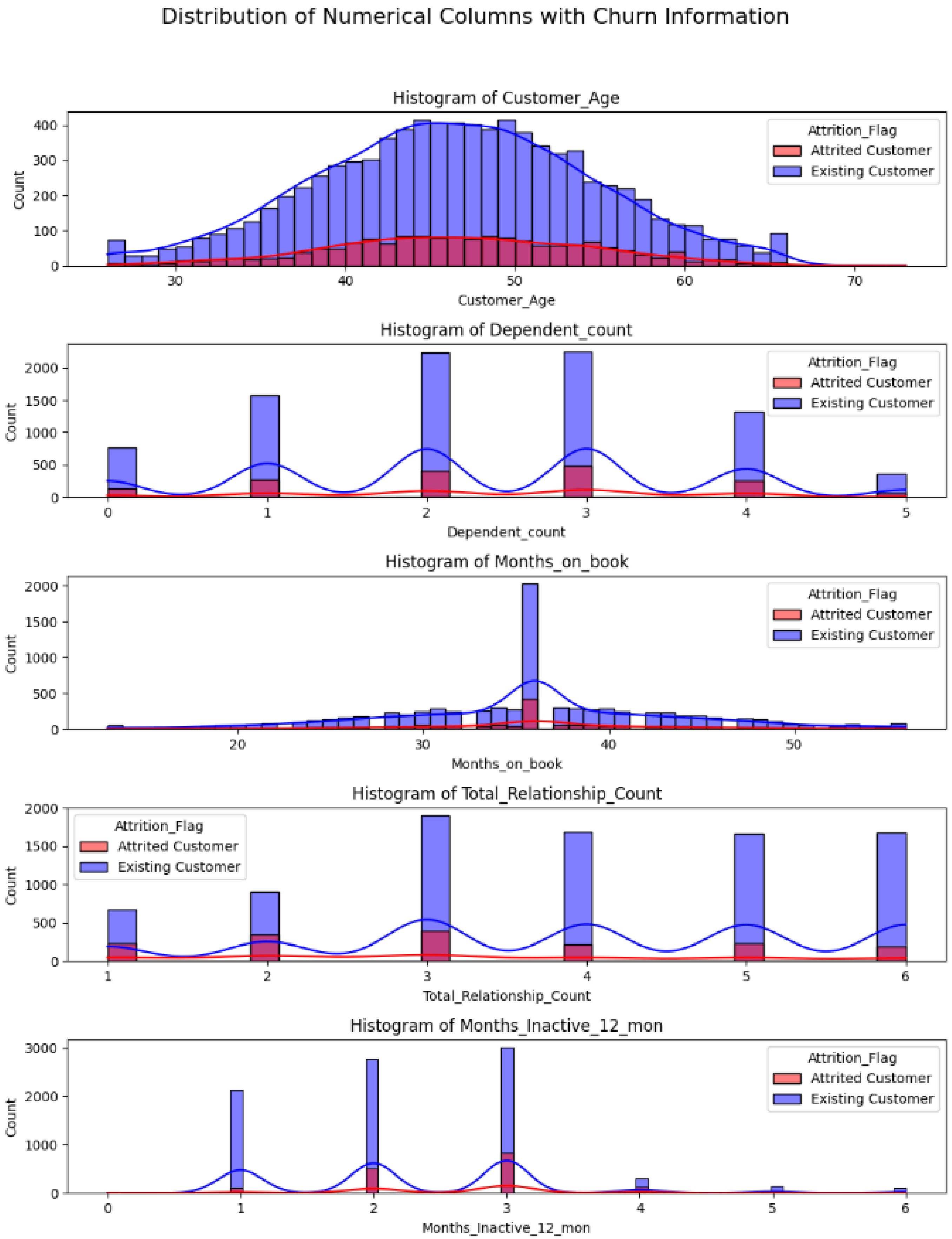
4. Handling Imbalance: Using SMOTEENN (Synthetic Minority Oversampling Technique and Edited Nearest Neighbors) to deal with the imbalanced target variable (Attrition_Flag), ensuring that the model doesn't favor the majority class.

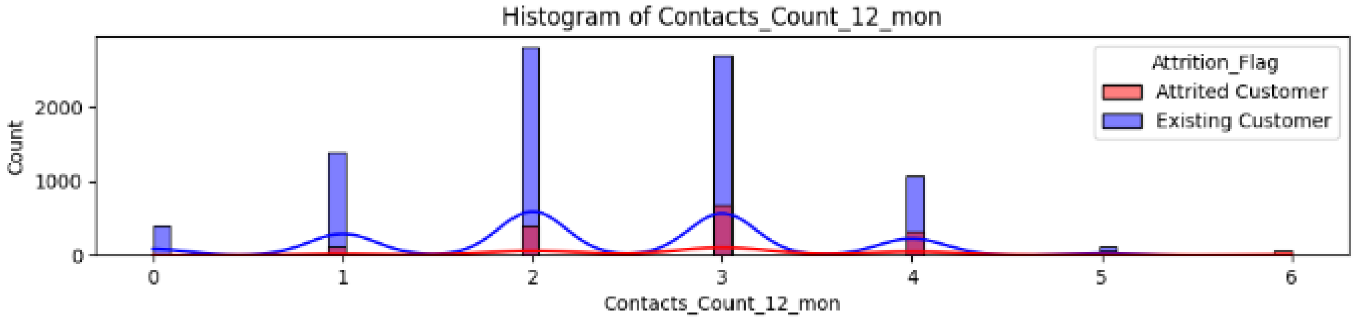
Handling Class Imbalance using SMOTE

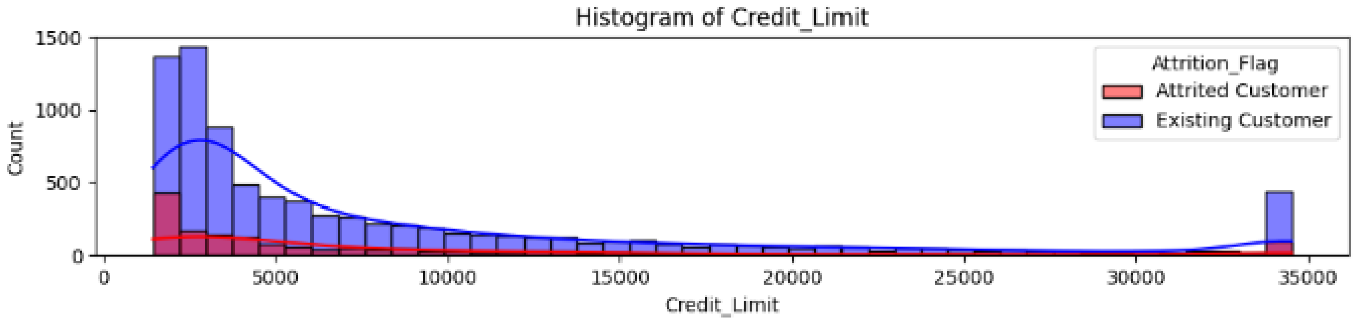
```
# Apply SMOTE to handle class imbalance
smote = SMOTEENN(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)
```

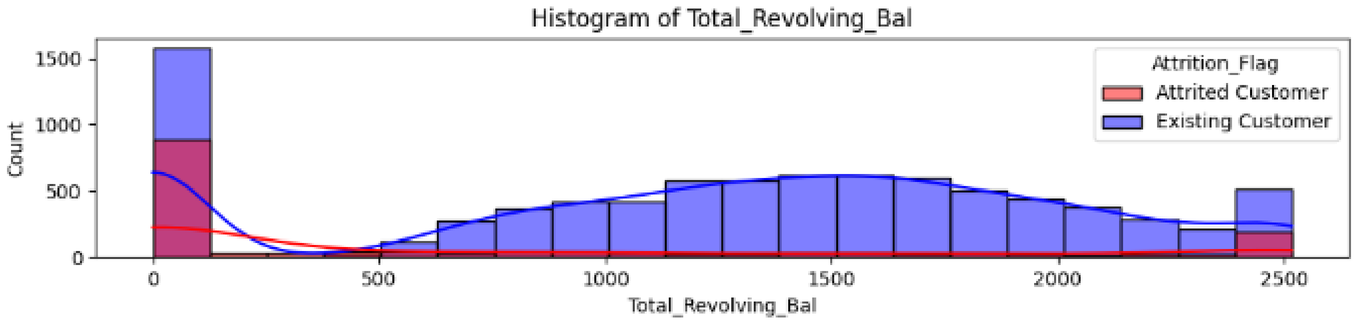
5. Exploratory Data Analysis (EDA)

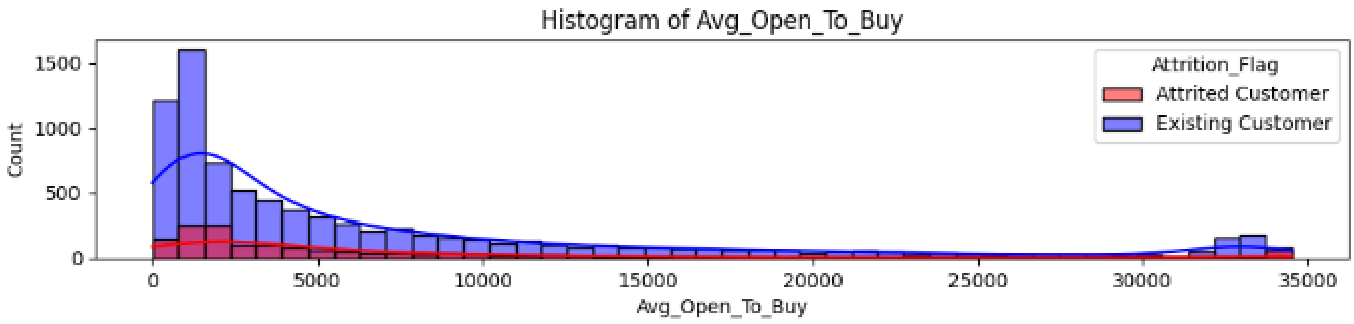
1- Data Distribution: Plotted histograms and box plots to understand the distribution of numerical variables.

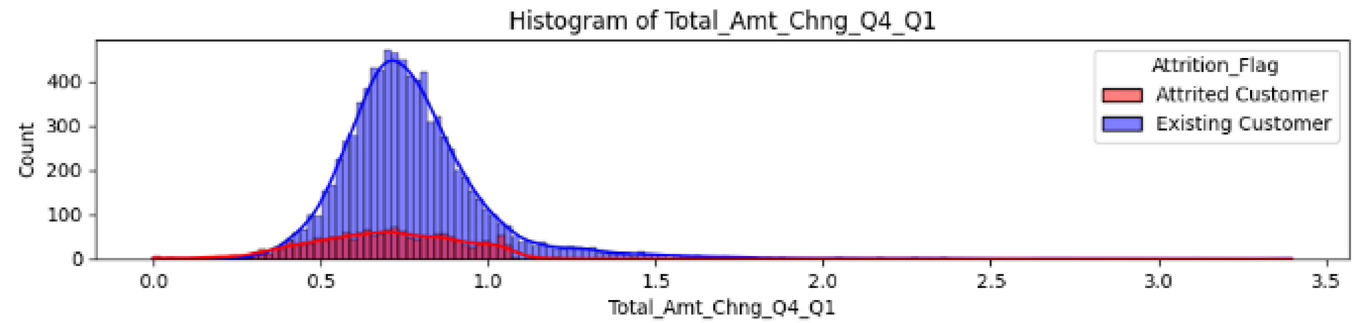


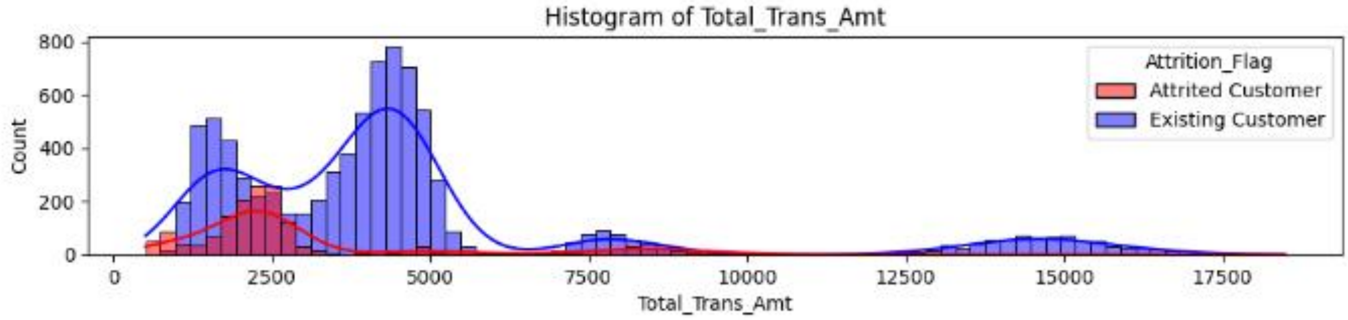

Histogram of Contacts_Count_12_mon
Attrition_Flag
Attrited Customer
Existing Customer
Count
0
1000
2000
0
1
2
3
4
5
6
Contacts_Count_12_mon

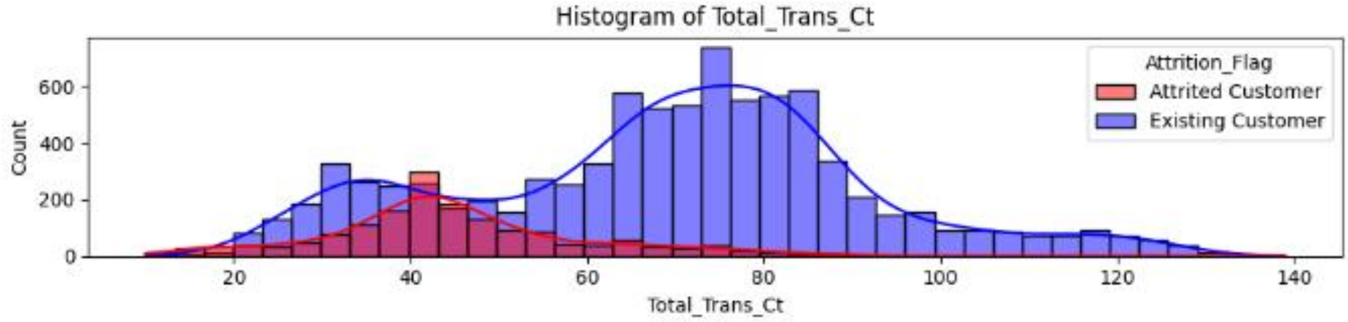

Histogram of Credit_Limit
Attrition_Flag
Attrited Customer
Existing Customer
Count
0
500
1000
1500
0
5000
10000
15000
20000
25000
30000
35000
Credit_Limit

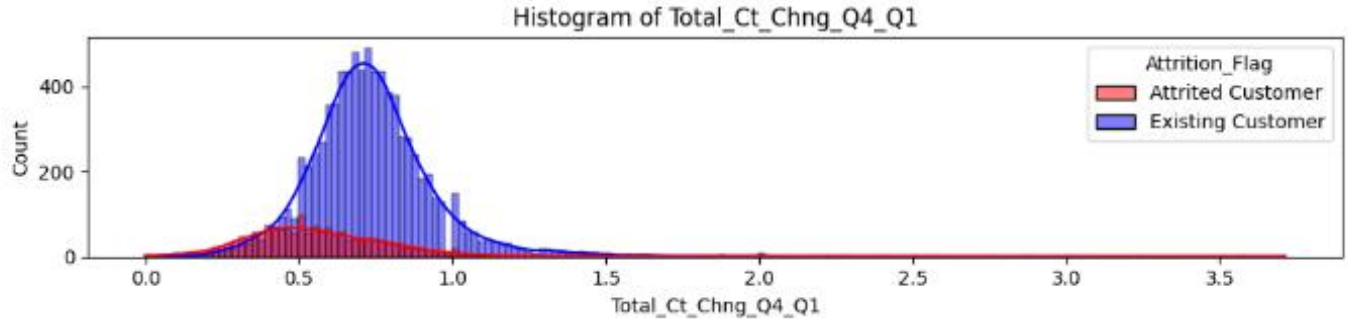

Histogram of Total_Revolving_Bal
Attrition_Flag
Attrited Customer
Existing Customer
Count
0
500
1000
1500
0
500
1000
1500
2000
2500
Total_Revolving_Bal

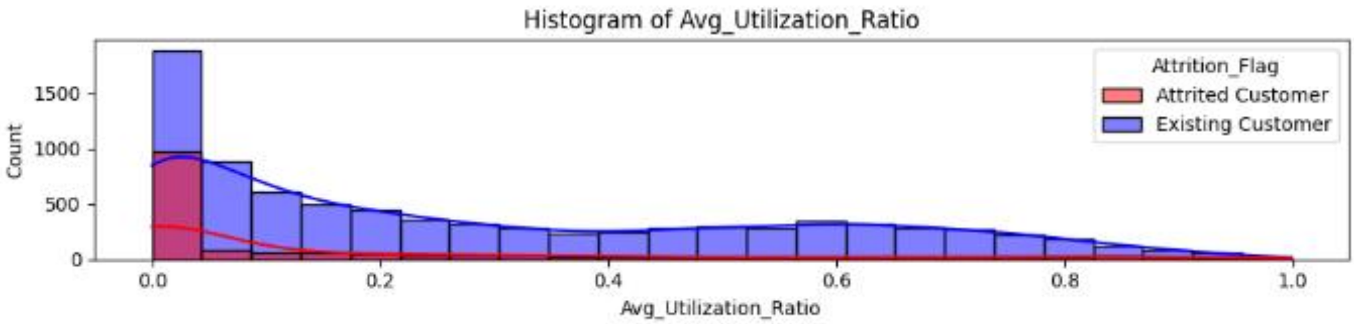

Histogram of Avg_Open_To_Buy
Attrition_Flag
Attrited Customer
Existing Customer
Count
0
500
1000
1500
0
5000
10000
15000
20000
25000
30000
35000
Avg_Open_To_Buy

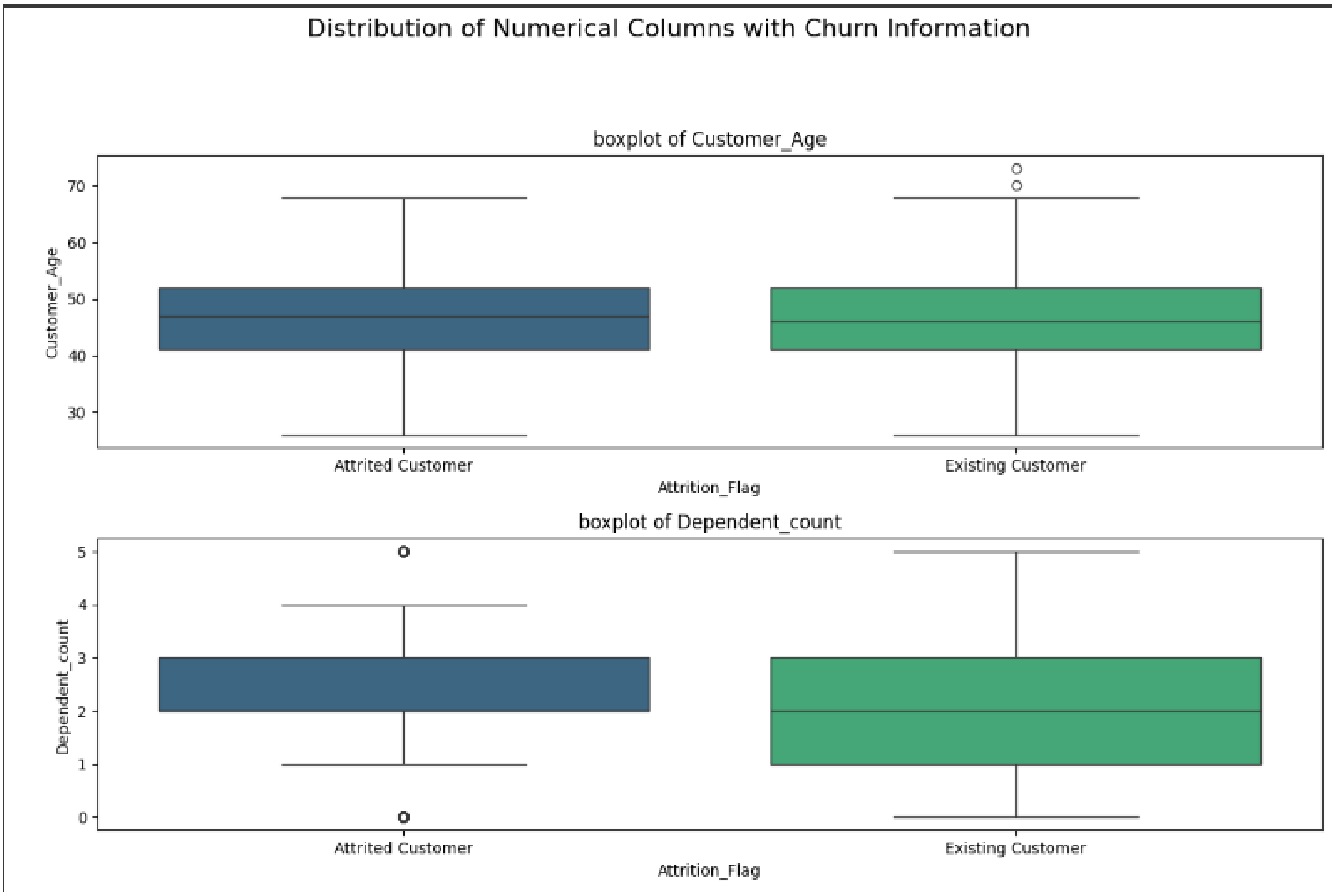

Histogram of Total_Amt_Chng_Q4_Q1
Attrition_Flag
Attrited Customer
Existing Customer
Count
0
100
200
300
400
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Total_Amt_Chng_Q4_Q1

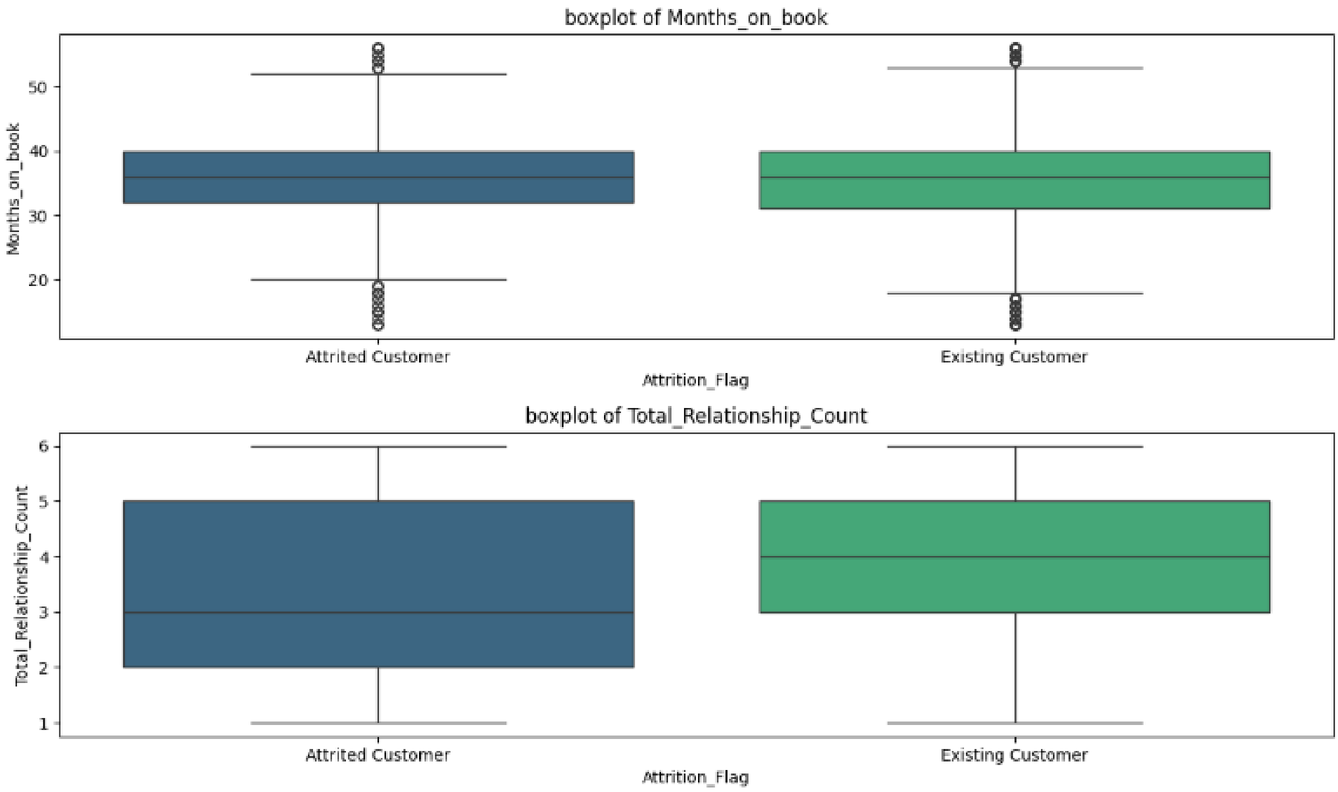

Histogram of Total_Trans_Amt
Count
Total_Trans_Amt
Attrition_Flag
Attrited Customer
Existing Customer

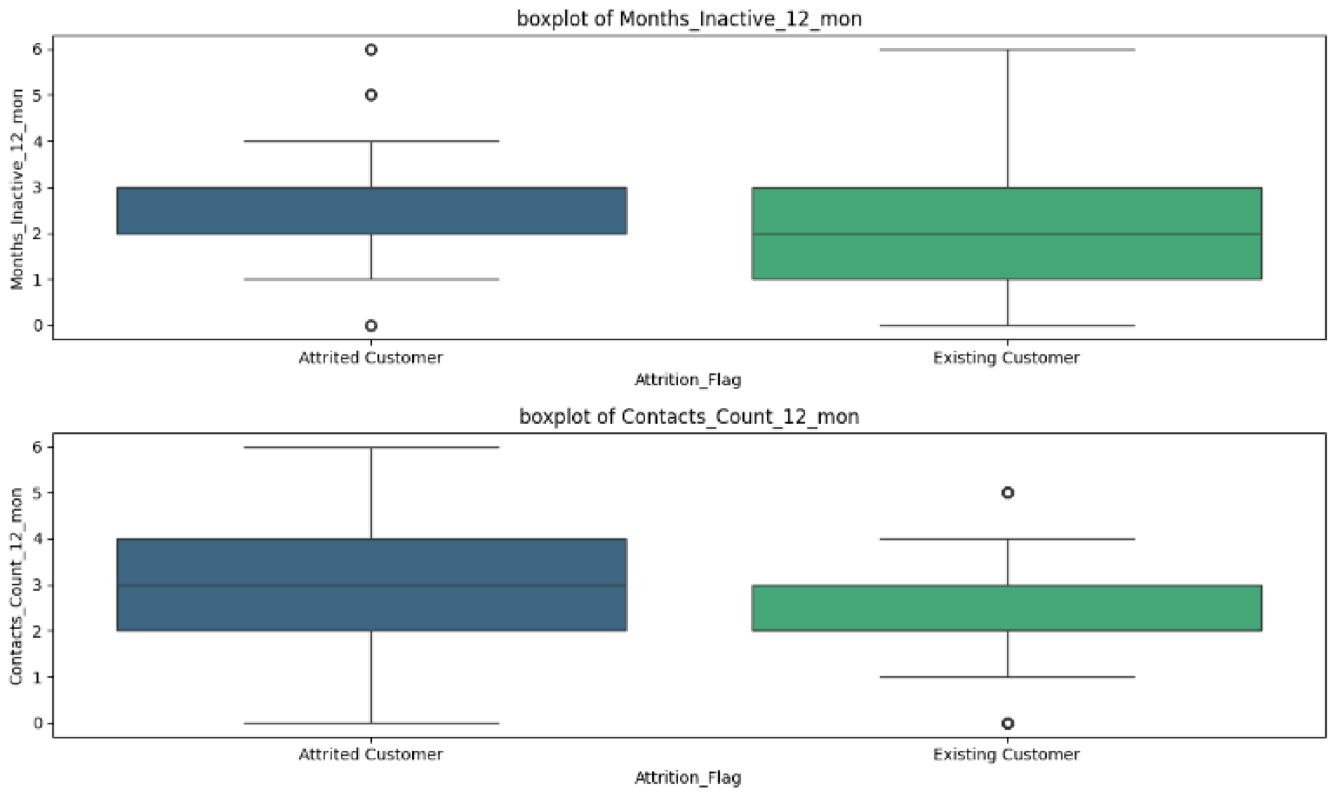

Histogram of Total_Trans_Ct
Count
Total_Trans_Ct
Attrition_Flag
Attrited Customer
Existing Customer

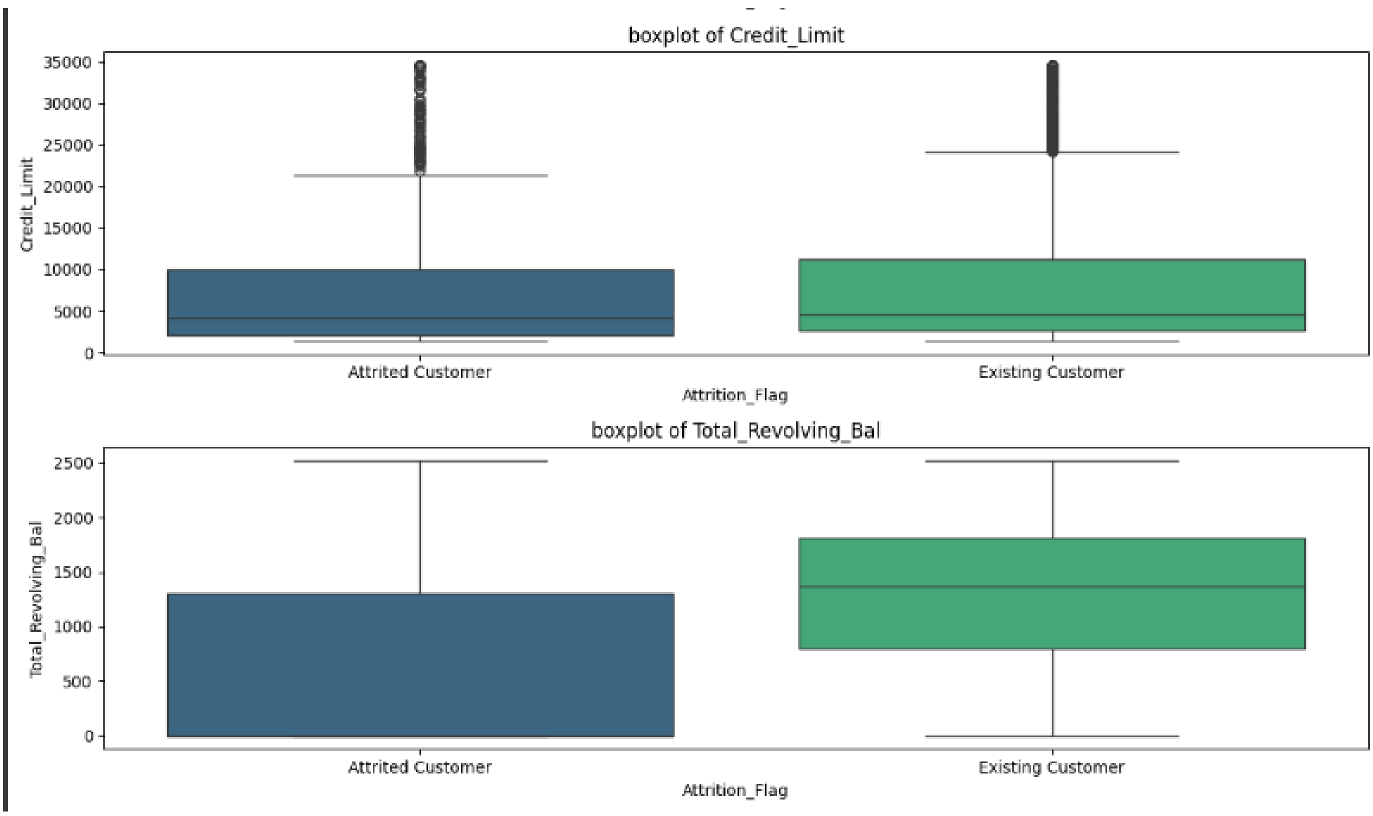

Histogram of Total_Ct_Chng_Q4_Q1
Count
Total_Ct_Chng_Q4_Q1
Attrition_Flag
Attrited Customer
Existing Customer

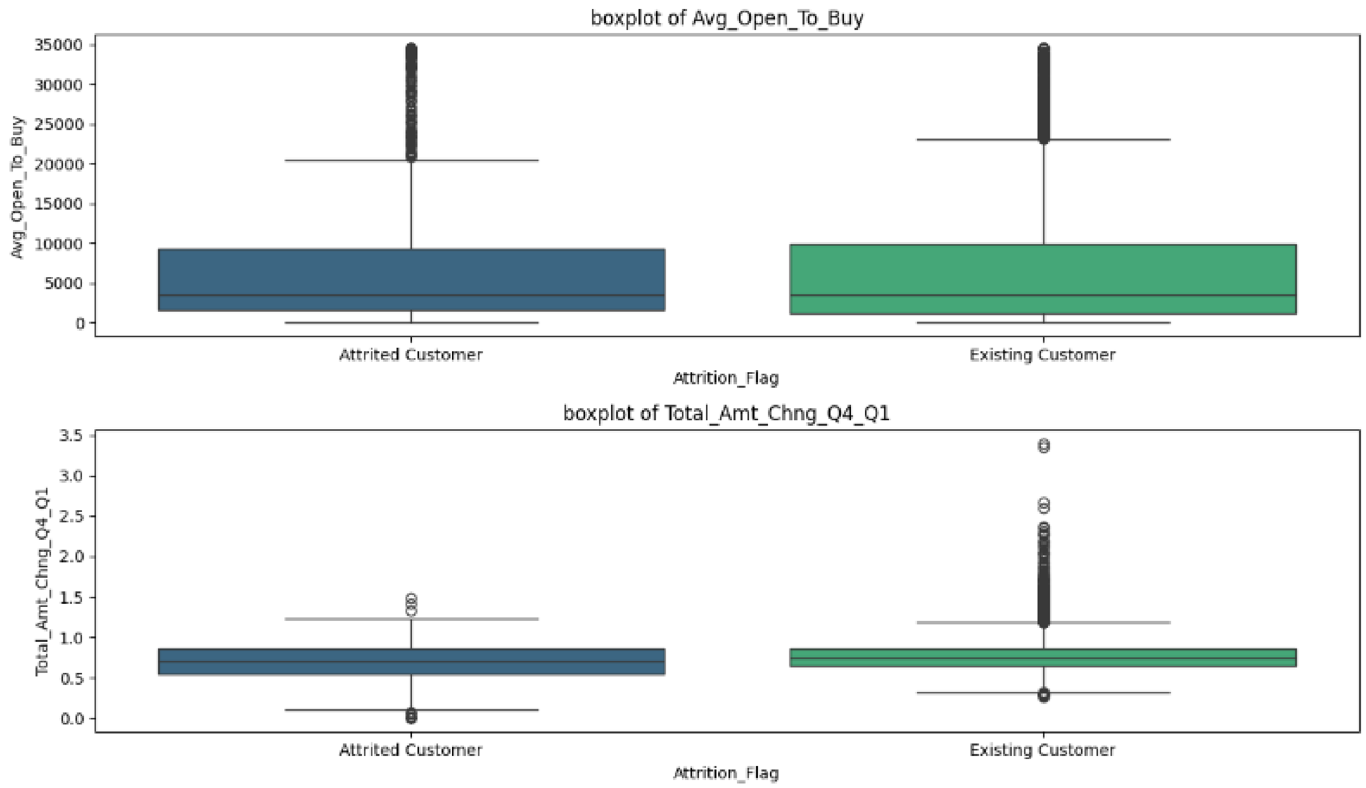

Histogram of Avg_Utilization_Ratio
Count
Avg_Utilization_Ratio
Attrition_Flag
Attrited Customer
Existing Customer

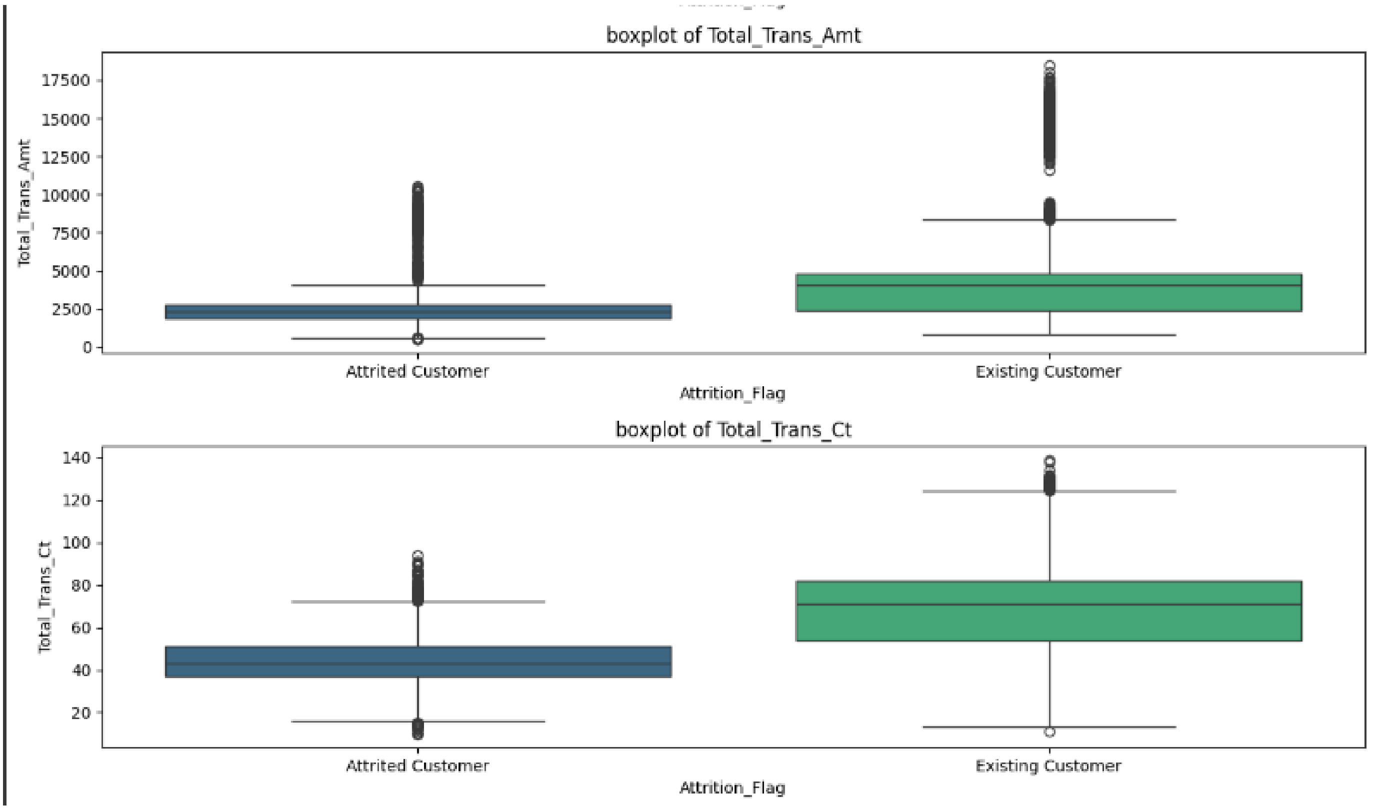

Distribution of Numerical Columns with Churn Information
boxplot of Customer_Age
Customer_Age
70
60
50
40
30
Attrited Customer
Existing Customer
Attrition_Flag
boxplot of Dependent_count
Dependent_count
5
4
3
2
1
0
Attrited Customer
Existing Customer
Attrition_Flag

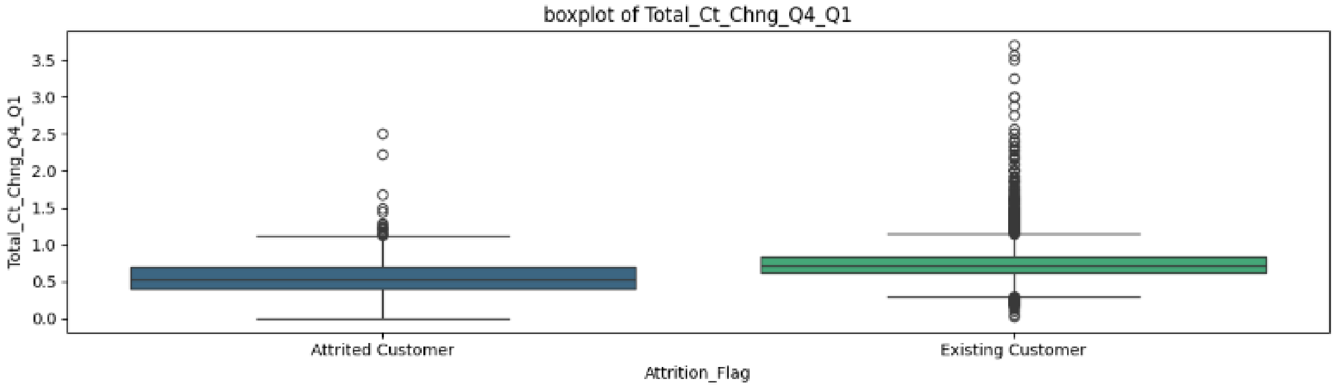

boxplot of Months_on_book
Months_on_book
50
40
30
20
Attrited Customer
Existing Customer
Attrition_Flag
boxplot of Total_Relationship_Count
Total_Relationship_Count
6
5
4
3
2
1
Attrited Customer
Existing Customer
Attrition_Flag

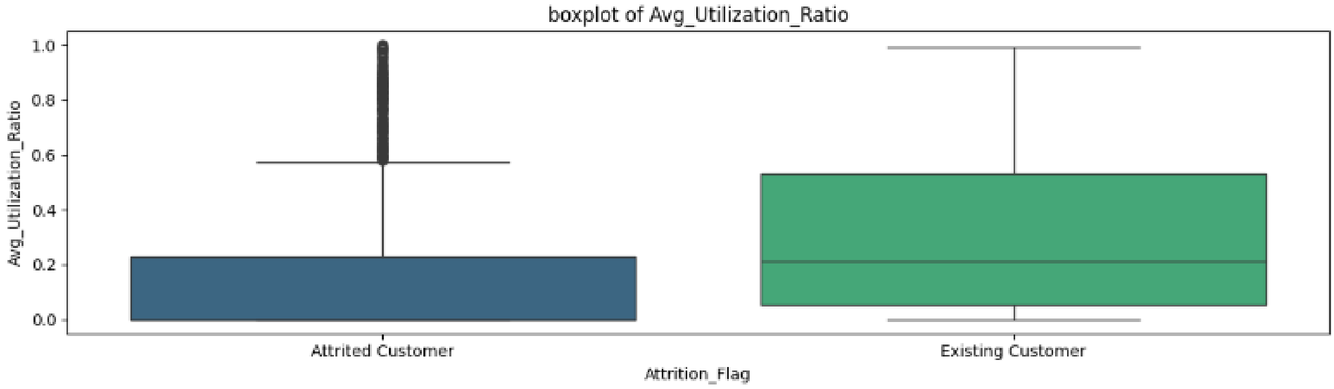

boxplot of Months_Inactive_12_mon
Months_Inactive_12_mon
0
1
2
3
4
5
6
Attrited Customer
Existing Customer
Attrition_Flag
boxplot of Contacts_Count_12_mon
Contacts_Count_12_mon
0
1
2
3
4
5
6
Attrited Customer
Existing Customer
Attrition_Flag

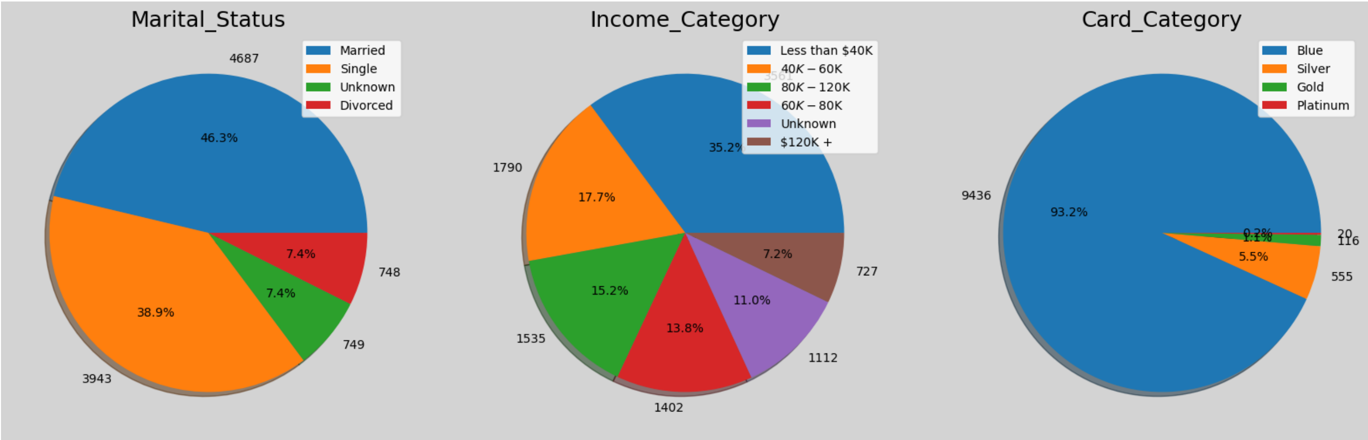

boxplot of Credit_Limit
Credit_Limit
0
5000
10000
15000
20000
25000
30000
35000
Attrited Customer
Existing Customer
Attrition_Flag
boxplot of Total_Revolving_Bal
Total_Revolving_Bal
0
500
1000
1500
2000
2500
Attrited Customer
Existing Customer
Attrition_Flag

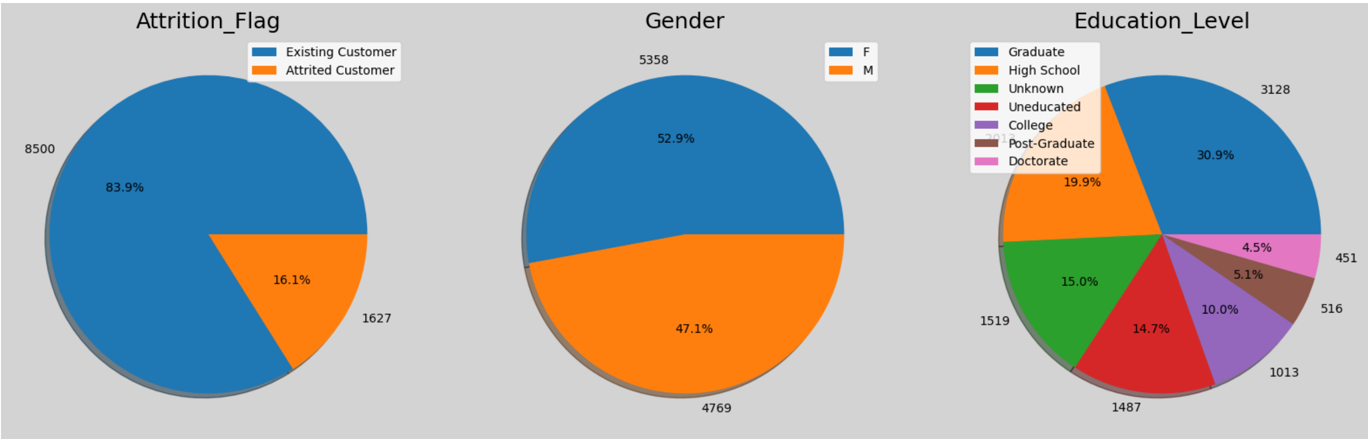

boxplot of Avg_Open_To_Buy
Avg_Open_To_Buy
35000
30000
25000
20000
15000
10000
5000
0
Attrited Customer
Existing Customer
Attrition_Flag
boxplot of Total_Amt_Chng_Q4_Q1
Total_Amt_Chng_Q4_Q1
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Attrited Customer
Existing Customer
Attrition_Flag

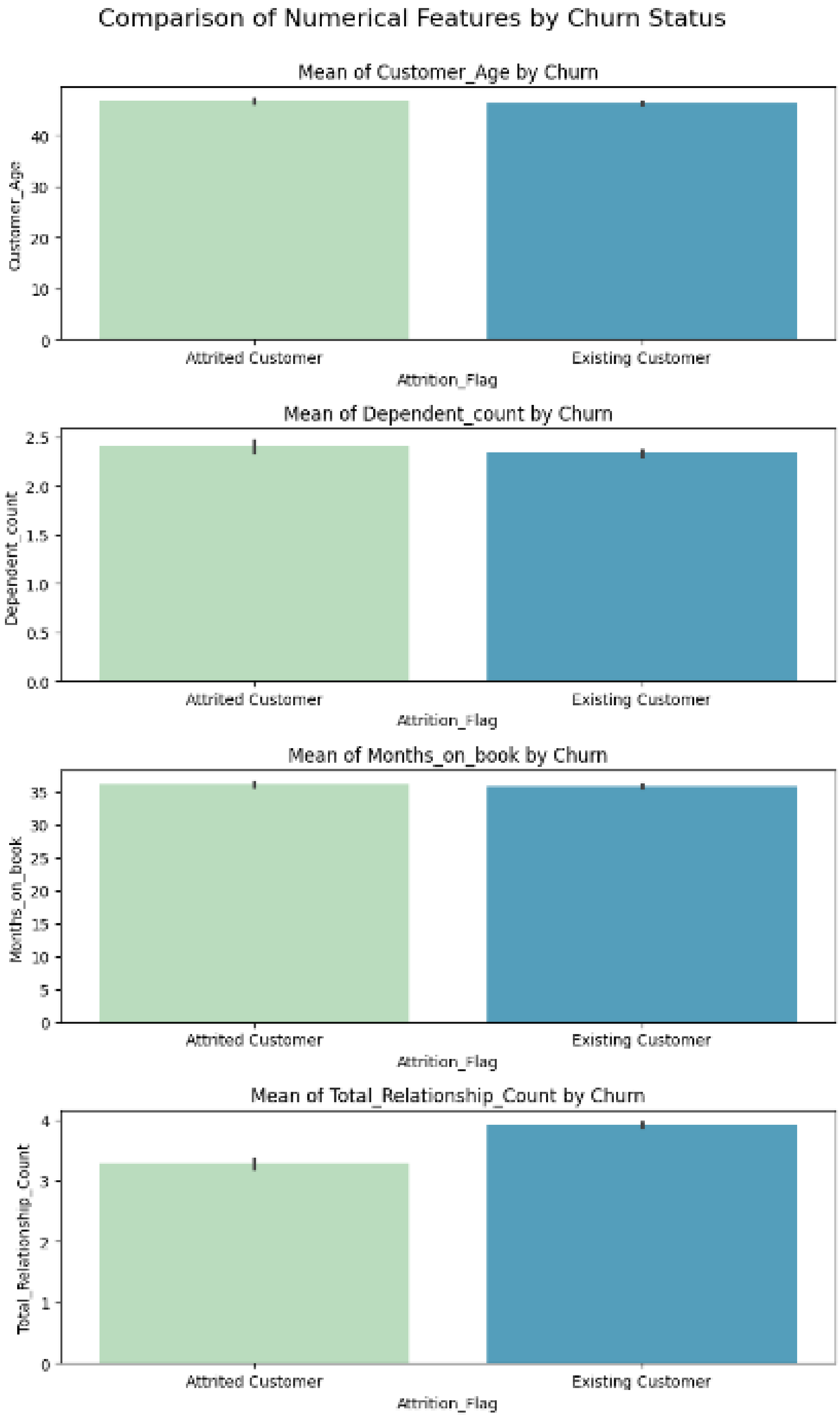

boxplot of Total_Trans_Amt
Total_Trans_Amt
17500
15000
12500
10000
7500
5000
2500
0
Attrited Customer
Existing Customer
Attrition_Flag
boxplot of Total_Trans_Ct
Total_Trans_Ct
140
120
100
80
60
40
20
Attrited Customer
Existing Customer
Attrition_Flag

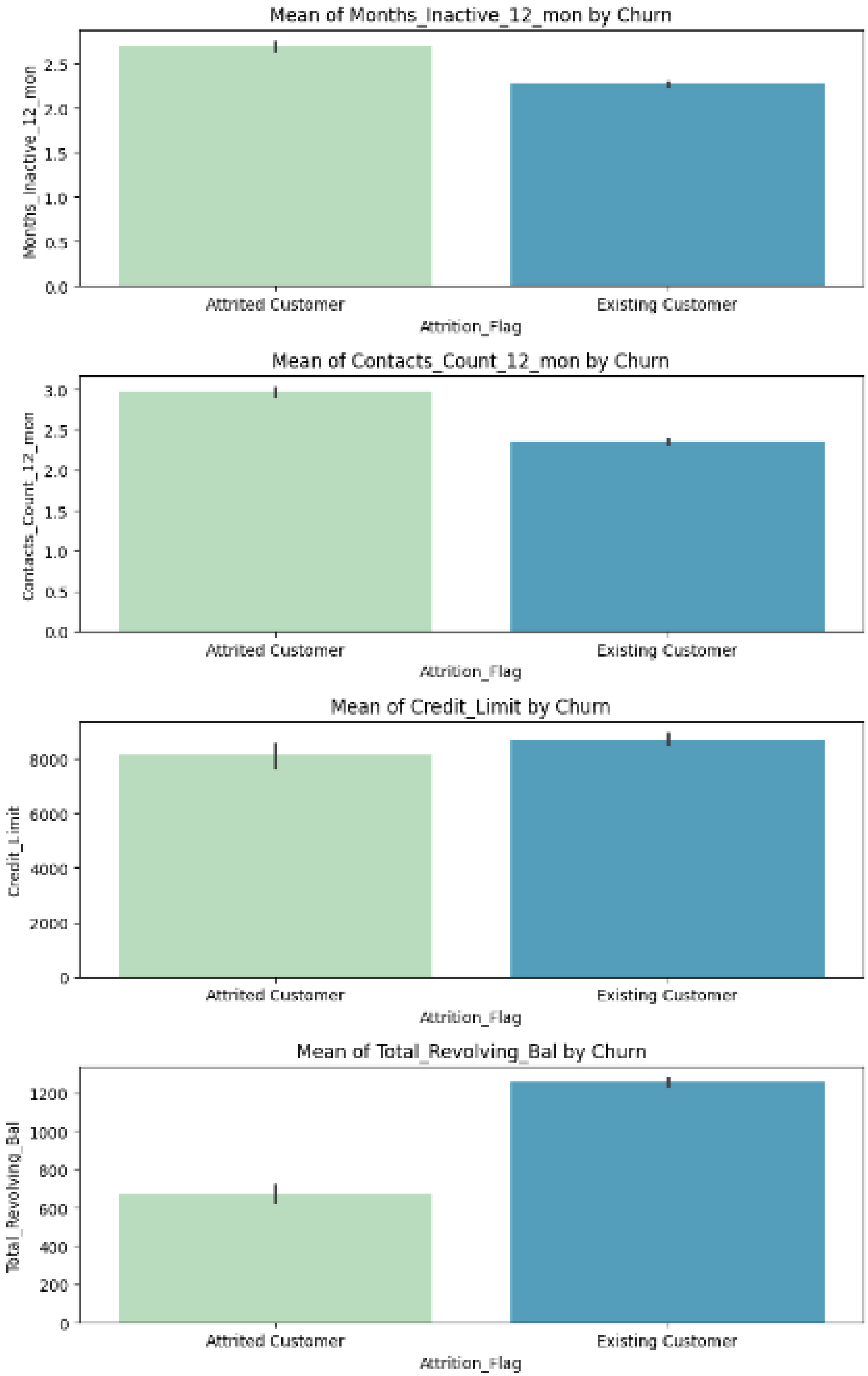

boxplot of Total_Ct_Chng_Q4_Q1
Total_Ct_Chng_Q4_Q1
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Attrited Customer
Existing Customer
Attrition_Flag


boxplot of Avg_Utilization_Ratio
Avg_Utilization_Ratio
1.0
0.8
0.6
0.4
0.2
0.0
Attrited Customer
Existing Customer
Attrition_Flag

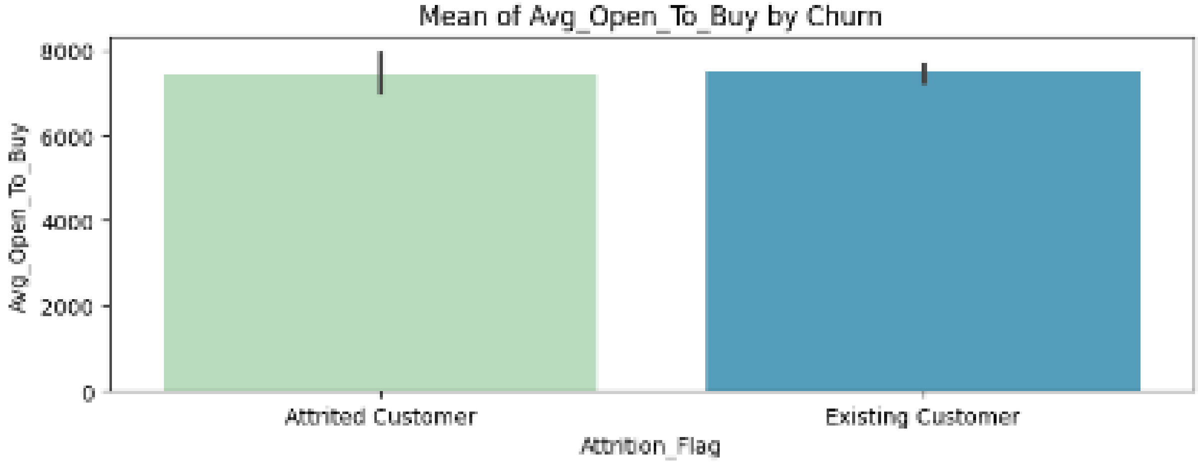

Marital_Status
Married
Single
Unknown
Divorced
4687
46.3%
7.4%
748
38.9%
7.4%
749
3943
Income_Category
Less than $40K
$40K - $60K$
$80K - $120K$
$60K - $80K$
Unknown
$120K +
1790
35.2%
17.7%
7.2%
727
15.2%
11.0%
1535
13.8%
1112
1402
Card_Category
Blue
Silver
Gold
Platinum
9436
93.2%
0.2%
1.1%
20
116
5.5%
555

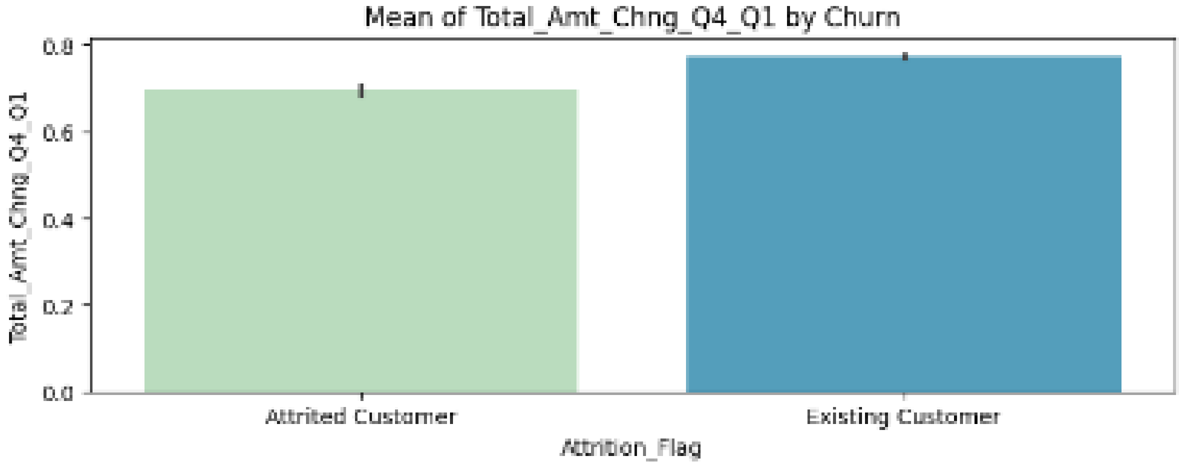

Attrition_Flag
Existing Customer
Attrited Customer
8500
83.9%
16.1%
1627
Gender
F
M
5358
52.9%
47.1%
4769
Education_Level
Graduate
High School
Unknown
Uneducated
College
Post-Graduate
Doctorate
3128
30.9%
19.9%
4.5%
451
5.1%
516
15.0%
10.0%
1519
14.7%
1013
1487

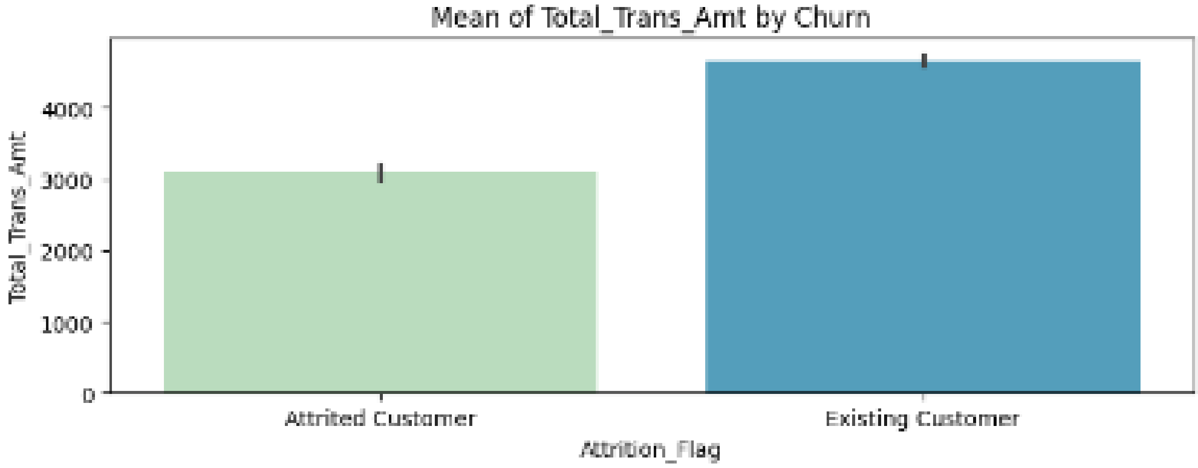

Comparison of Numerical Features by Churn Status
Mean of Customer_Age by Churn
Customer_Age
0
10
20
30
40
Attrited Customer
Existing Customer
Attrition_Flag
Mean of Dependent_count by Churn
Dependent_count
0.0
0.5
1.0
1.5
2.0
2.5
Attrited Customer
Existing Customer
Attrition_Flag
Mean of Months_on_book by Churn
Months_on_book
0
5
10
15
20
25
30
35
Attrited Customer
Existing Customer
Attrition_Flag
Mean of Total_Relationship_Count by Churn
Total_Relationship_Count
0
1
2
3
4
Attrited Customer
Existing Customer
Attrition_Flag

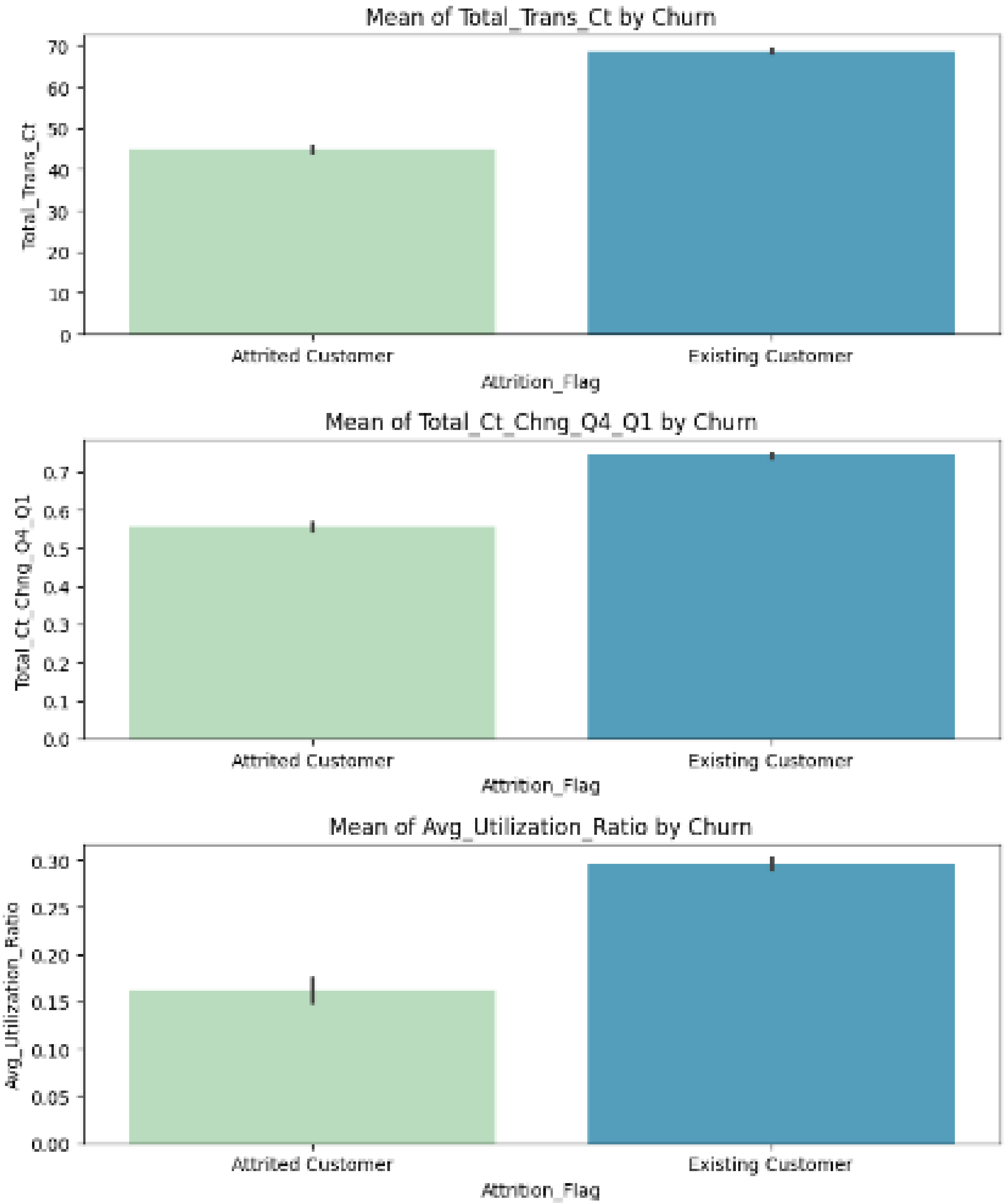


Mean of Months_Inactive_12_mon by Churn
Months_Inactive_12_mon
0.0
0.5
1.0
1.5
2.0
2.5
Attrited Customer
Existing Customer
Attrition_Flag
Mean of Contacts_Count_12_mon by Churn
Contacts_Count_12_mon
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Attrited Customer
Existing Customer
Attrition_Flag
Mean of Credit_Limit by Churn
Credit_Limit
0
2000
4000
6000
8000
Attrited Customer
Existing Customer
Attrition_Flag
Mean of Total_Revolving_Bal by Churn
Total_Revolving_Bal
0
200
400
600
800
1000
1200
Attrited Customer
Existing Customer
Attrition_Flag

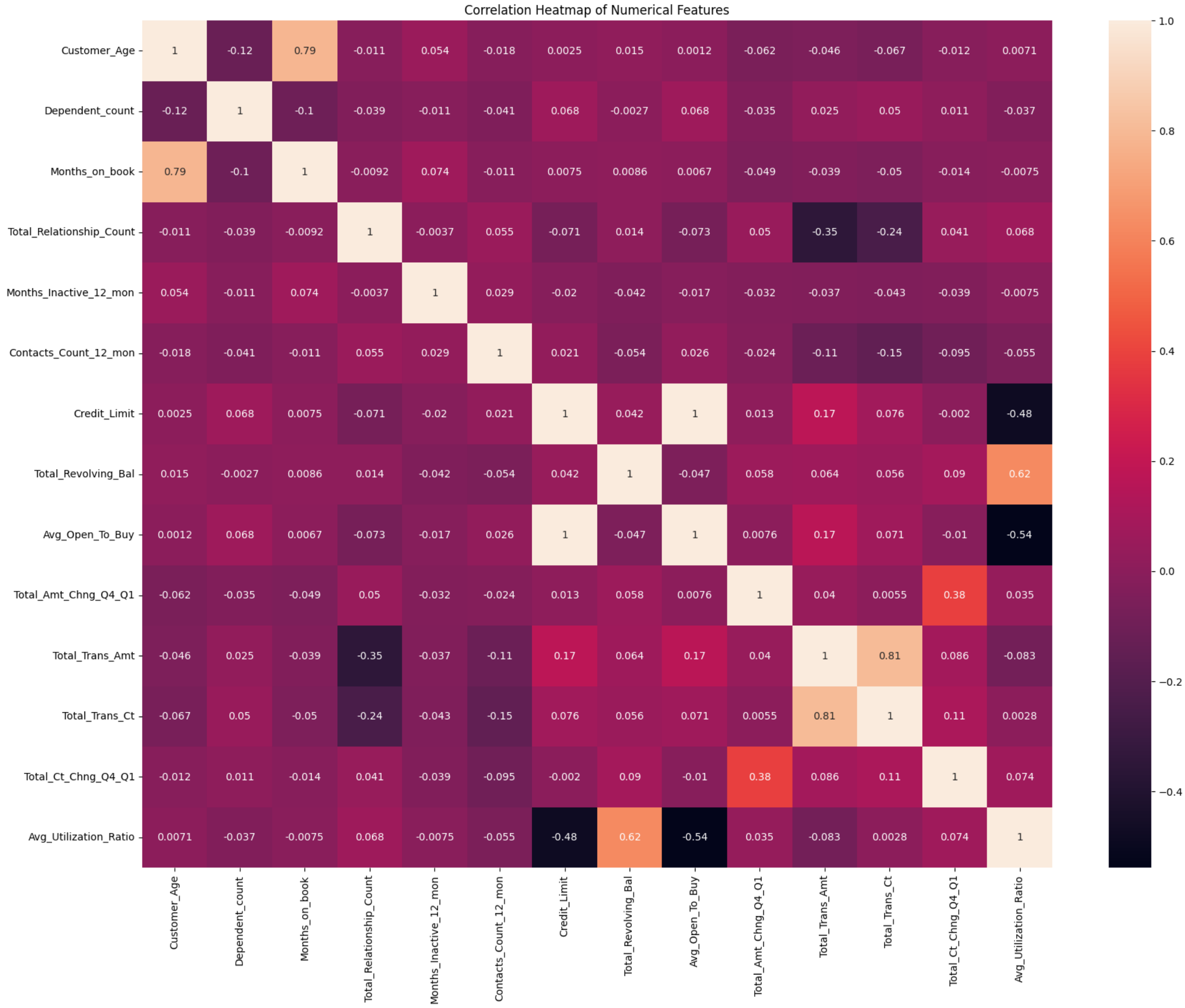

Mean of Avg_Open_To_Buy by Churn
8000
6000
4000
2000
0
Avg_Open_To_Buy
Attrited Customer
Existing Customer
Attrition_Flag

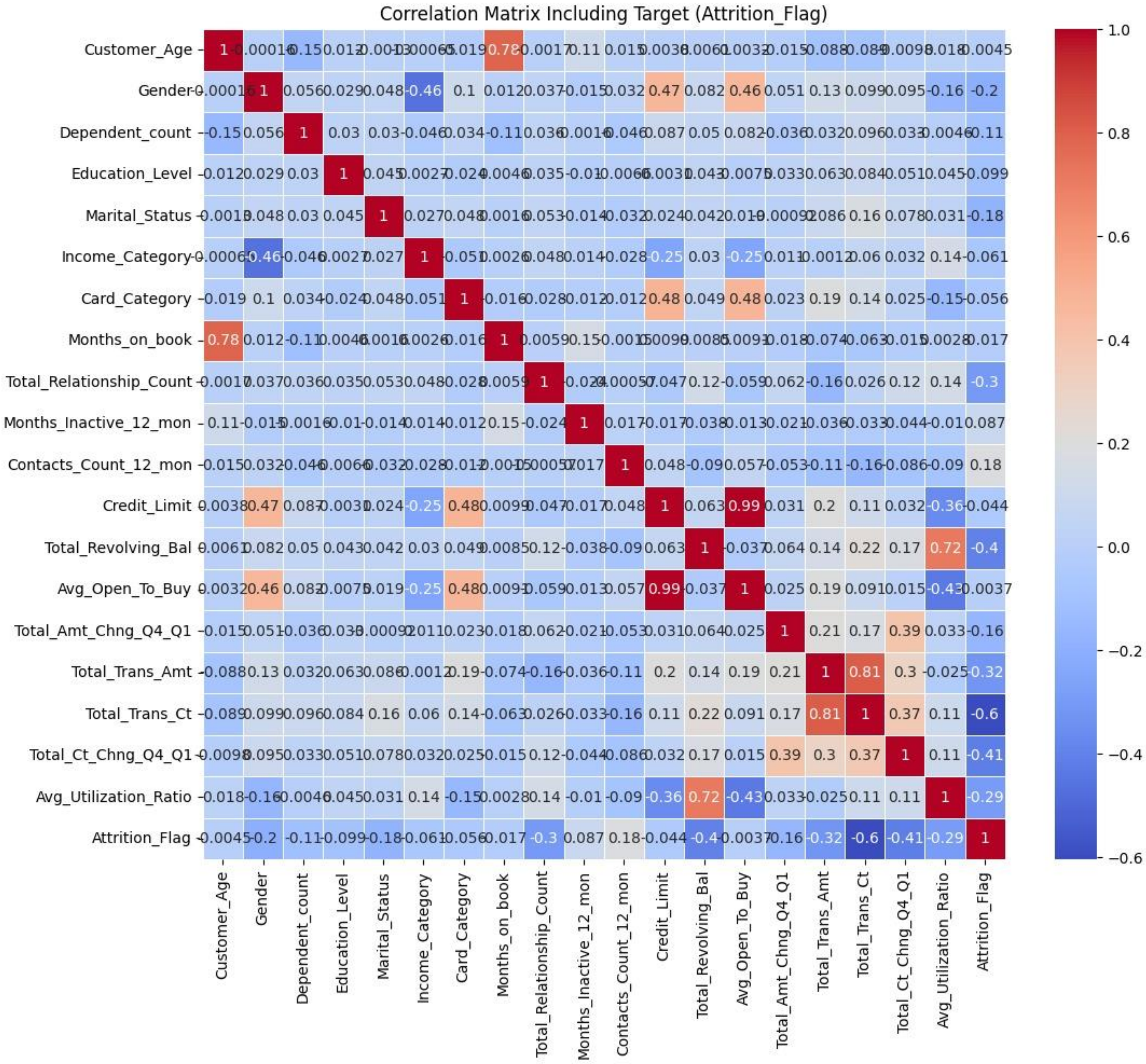

Mean of Total_Amt_Chng_Q4_Q1 by Churn
0.8
0.6
0.4
0.2
0.0
Total_Amt_Chng_Q4_Q1
Attrited Customer
Existing Customer
Attrition_Flag


Mean of Total_Trans_Amt by Churn
4000
3000
2000
1000
0
Total_Trans_Amt
Attrited Customer
Existing Customer
Attrition_Flag


Mean of Total_Trans_Ct by Churn
Total_Trans_Ct
0
10
20
30
40
50
60
70
Attrited Customer
Existing Customer
Attrition_Flag
Mean of Total_Ct_Chng_Q4_Q1 by Churn
Total_Ct_Chng_Q4_Q1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Attrited Customer
Existing Customer
Attrition_Flag
Mean of Avg_Utilization_Ratio by Churn
Avg_Utilization_Ratio
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Attrited Customer
Existing Customer
Attrition_Flag

5. 2 Correlation Analysis: Created a heatmap to visualize correlations between different features.




Correlation Matrix Including Target (Attrition_Flag)
Customer_Age
Gender
Dependent_count
Education_Level
Marital_Status
Income_Category
Card_Category
Months_on_book
Total_Relationship_Count
Months_Inactive_12_mon
Contacts_Count_12_mon
Credit_Limit
Total_Revolving_Bal
Avg_Open_To_Buy
Total_Amt_Chng_Q4_Q1
Total_Trans_Amt
Total_Trans_Ct
Total_Ct_Chng_Q4_Q1
Avg_Utilization_Ratio
Attrition_Flag
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

	Attrition_flag	Count	Percentage (%)	min_Customer_Age	max_Customer_Age	avg_Customer_Age
0	Existing Customer	8500	83.93%	26.00	73.00	46.26
1	Attrited Customer	1627	16.07%	26.00	68.00	46.66

	Attrition_flag	Count	Percentage (%)	min_Dependent_count	max_Dependent_count	avg_Dependent_count
0	Existing Customer	8500	83.93%	0.00	5.00	2.34
1	Attrited Customer	1627	16.07%	0.00	5.00	2.40

	Attrition_flag	Count	Percentage (%)	min_Months_on_book	max_Months_on_book	avg_Months_on_book
0	Existing Customer	8500	83.93%	13.00	56.00	35.88
1	Attrited Customer	1627	16.07%	13.00	56.00	36.18

	Attrition_flag	Count	Percentage (%)	min_Total_Relationship_Count	max_Total_Relationship_Count	avg_Total_Relationship_Count
0	Existing Customer	8500	83.93%	1.00	6.00	3.91
1	Attrited Customer	1627	16.07%	1.00	6.00	3.28

	Attrition_flag	Count	Percentage (%)	min_Months_Inactive_12_mon	max_Months_Inactive_12_mon	avg_Months_Inactive_12_mon
0	Existing Customer	8500	83.93%	0.00	6.00	2.27
1	Attrited Customer	1627	16.07%	0.00	6.00	2.69

	Attrition_flag	Count	Percentage (%)	min_Contacts_Count_12_mon	max_Contacts_Count_12_mon	avg_Contacts_Count_12_mon
0	Existing Customer	8500	83.93%	0.00	5.00	2.36
1	Attrited Customer	1627	16.07%	0.00	6.00	2.97

	Attrition_flag	Count	Percentage (%)	min_Credit_Limit	max_Credit_Limit	avg_Credit_Limit
0	Existing Customer	8500	83.93%	1438.30	34516.00	8726.88
1	Attrited Customer	1627	16.07%	1438.30	34516.00	8136.04

	Attrition_flag	Count	Percentage (%)	min_Total_Revolving_Bal	max_Total_Revolving_Bal	avg_Total_Revolving_Bal
0	Existing Customer	8500	83.93%	0.00	2517.00	1256.60
1	Attrited Customer	1627	16.07%	0.00	2517.00	672.82

	Attrition_flag	Count	Percentage (%)	min_Avg_Open_To_Buy	max_Avg_Open_To_Buy	avg_Avg_Open_To_Buy
0	Existing Customer	8500	83.93%	15.00	34516.00	7470.27
1	Attrited Customer	1627	16.07%	3.00	34516.00	7463.22

	Attrition_flag	Count	Percentage (%)	min_Total_Amt_Chng_Q4_Q1	max_Total_Amt_Chng_Q4_Q1	avg_Total_Amt_Chng_Q4_Q1
0	Existing Customer	8500	83.93%	0.26	3.40	0.77
1	Attrited Customer	1627	16.07%	0.00	1.49	0.69

	Attrition_flag	Count	Percentage (%)	min_Total_Trans_Amt	max_Total_Trans_Amt	avg_Total_Trans_Amt
0	Existing Customer	8500	83.93%	816.00	18484.00	4654.66
1	Attrited Customer	1627	16.07%	510.00	10583.00	3095.03

	Attrition_flag	Count	Percentage (%)	min_Total_Trans_Ct	max_Total_Trans_Ct	avg_Total_Trans_Ct
0	Existing Customer	8500	83.93%	11.00	139.00	68.67
1	Attrited Customer	1627	16.07%	10.00	94.00	44.93

	Attrition_flag	Count	Percentage (%)	min_Total_Ct_Chng_Q4_Q1	max_Total_Ct_Chng_Q4_Q1	avg_Total_Ct_Chng_Q4_Q1
0	Existing Customer	8500	83.93%	0.03	3.71	0.74
1	Attrited Customer	1627	16.07%	0.00	2.50	0.55

	Attrition_flag	Count	Percentage (%)	min_Avg_Utilization_Ratio	max_Avg_Utilization_Ratio	avg_Avg_Utilization_Ratio
0	Existing Customer	8500	83.93%	0.00	0.99	0.30
1	Attrited Customer	1627	16.07%	0.00	1.00	0.16

Insights from EDA

Targeted Credit Limit Management Key Insight: The credit limits vary significantly across income categories. Higher income groups (e.g., 120K+)haveanaveragecreditlimitof 19,717, while lower income groups like those earning Less than 40Khavemuchlowercreditlimits(3,754).

Actionable Strategy: we can enhance customer satisfaction and loyalty by offering personalized credit limits based on spending habits, not just income. Customers in lower income categories might appreciate higher limits if they demonstrate good credit behavior, improving retention.

Optimized Utilization of Revolving Balance Key Insight: The Average Utilization Ratio is much higher for lower income categories (e.g., 0.38 for Less than 40K)comparedtohigherincomegroups (0.12for 120K+). This suggests lower-income customers rely more on their revolving credit balances.

Actionable Strategy: Implement a reward program or financial coaching aimed at helping lower-income customers manage their credit utilization better. This could reduce credit risk and foster trust, while also encouraging higher card usage in a controlled way, benefiting both the customer and the business.

6. Methodology

This project utilizes several Python libraries for data manipulation, machine learning, and visualization:

- Pandas: Data manipulation and analysis.
- Numpy: Numerical computations.
- Matplotlib & Seaborn: Data visualization and plotting.
- Scikit-learn: Machine learning algorithms and model evaluation tools.
- Imbalanced-learn: For handling imbalanced datasets with SMOTEENN (a combination of oversampling and undersampling).
- XGBoost & LightGBM: Gradient boosting algorithms for classification.
- TQDM: Progress bar for tracking lengthy operations.
- Joblib & Pickle: Model serialization for saving and loading trained models.
- Streamlit: Web framework for deploying the model as an interactive app.

For this project, several machine learning models were explored, including:

- Logistic Regression: A baseline linear model.
- K-Nearest Neighbors (KNN): A non-parametric method that classifies based on proximity.
- Decision Tree: A model that splits data recursively to create decision rules.
- Random Forest: An ensemble of decision trees to improve performance.
- AdaBoost: An adaptive boosting technique for classification.
- Gradient Boosting: Sequentially builds models to minimize errors.
- XGBoost: An efficient and scalable implementation of gradient boosting.
- Naive Bayes: Based on Bayes' theorem for probabilistic classification.
- LightGBM: A gradient boosting framework focusing on large datasets with better performance.

Steps and Procedures Followed

1. Feature Selection: Identified and selected the most important features influencing customer churn.
2. Feature Engineering: Created new features that could potentially improve the model's performance.
3. Model Training: Trained each model using a training dataset and validated their performance on a test set.
4. Hyperparameter Tuning: Optimized the model parameters to enhance their performance.
5. Model Evaluation: Used various evaluation metrics like accuracy, precision, recall, and F1-score to compare the models.

Steps to Implement

Week 1: Data Understanding and Preprocessing

- Load and clean the dataset.
- Handle missing values, encode categorical variables, and scale numerical features.

Week 2: Exploratory Data Analysis and Feature Engineering

- Perform exploratory data analysis to identify trends and relationships.
- Engineer new features to enhance model performance.

Week 3: Model Development and Evaluation

- Train and evaluate models such as logistic regression, random forest, and gradient boosting.
- Fine-tune model hyperparameters for optimal performance.

Week 4: Model Optimization and Deployment

- Perform hyperparameter tuning using GridSearchCV.
- Document the results and prepare the final report.

7. Model Training and Evaluation

1 . XGBoost Model Training and Evaluation:

- Initializes the XGBoost classifier (`XGBClassifier()`), trains it on the training data (`X_train`), and evaluates it on the test set (`X_test`).
- Displays the classification report, accuracy score, confusion matrix, and feature importance for the XGBoost model.

```
# Initializing the XGBoost classifier
model = XGBClassifier()

# Training the model
model.fit(X_train, y_train)

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

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Displaying the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Plotting feature importance
plt.figure(figsize=(12, 8))
plot_importance(model, importance_type='weight')
plt.title('Feature Importance')
plt.xlabel('F score')
plt.ylabel('Features')
plt.show()
```

Explanation:

- **Model Initialization**: The XGBClassifier is a gradient boosting framework known for its predictive power.
- **Training & Predictions**: The model is trained on `X_train`, and predictions are made on the `X_test` set.
- **Evaluation**: Accuracy is calculated, followed by a classification report and confusion matrix for performance analysis. Feature importance is plotted to highlight impactful features.

Output of XGBoost Model Training and Evaluation:-

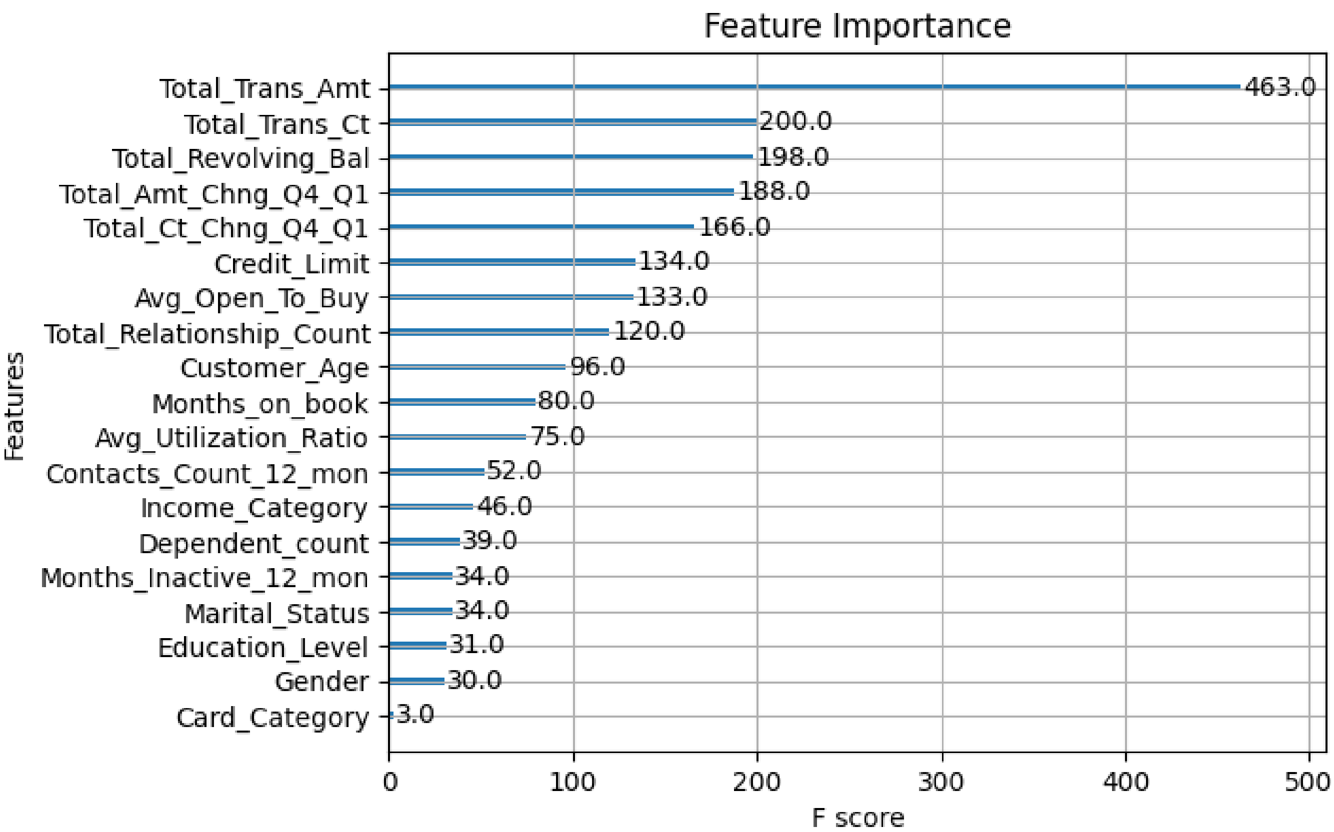
```
Accuracy: 0.947680157946693

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97      1699
           1       0.79      0.93      0.85       327

    accuracy                           0.95      2026
   macro avg       0.89      0.94      0.91      2026
weighted avg       0.95      0.95      0.95      2026


Confusion Matrix:
[[1617   82]
 [  24  303]]
```



Model Hyperparameter Tuning with Grid Search:

- A dictionary (`models_params`) defines different machine learning models (Logistic Regression, Naive Bayes, K-Nearest Neighbors, Decision Trees, Random Forest, AdaBoost, Gradient Boosting, XGBoost, and LightGBM) and their hyperparameter grids.
- Grid search with 3-fold cross-validation is used to find the best hyperparameters for each model, optimizing for recall. The results include the best parameters, confusion matrix, and classification report.

```python
# Dictionary of models and their parameter grids
models_params = {
    "Logistic Regression": (LogisticRegression(solver='liblinear'), {
        'C': [0.001, 0.01, 0.1, 1, 10, 100],
        'penalty': ['l1', 'l2'],  # L1 and L2 penalties
        'max_iter': [100, 200, 300],
    }),

    "Naive Bayes": (GaussianNB(), {
        # GaussianNB doesn't have many hyperparameters to tune
    }),

    "K-Nearest Neighbors": (KNeighborsClassifier(), {
        'n_neighbors': [3, 5, 7, 10],
        'weights': ['uniform', 'distance'],
        'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    }),

    "Decision Tree Classifier": (DecisionTreeClassifier(), {
        'max_depth': [None, 5, 10, 20],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'criterion': ['gini', 'entropy'],
    }),

    "Random Forest": (RandomForestClassifier(), {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'criterion': ['gini', 'entropy'],
    }),

    "AdaBoost": (AdaBoostClassifier(), {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 1, 10],
    }),

    "Gradient Boosting": (GradientBoostingClassifier(), {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [3, 5, 7],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
    }),

    "XGBoost": (XGBClassifier(use_label_encoder=False, eval_metric='logloss'), {
        'n_estimators': [50, 100, 200],
        'max_depth': [3, 5, 7],
        'learning_rate': [0.01, 0.1, 0.2],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
    }),

    "LightGBM": (LGBMClassifier(), {
        'n_estimators': [50, 100, 200],
        'learning_rate': [0.01, 0.1, 0.2],
        'max_depth': [-1, 5, 10],
        'num_leaves': [31, 50, 100],
        'min_child_samples': [20, 50, 100],
    })
}
```

```python
# Model training with tqdm and GridSearchCV
for name, (model, params) in tqdm(models_params.items(), desc="Training models", total=len(models_params)):

    # Perform Grid Search with Recall as the scoring metric
    grid_search = GridSearchCV(estimator=model, param_grid=params, scoring='recall', cv=3, n_jobs=-1)
    grid_search.fit(X_train, y_train)

    # Best model and its parameters
    best_model = grid_search.best_estimator_
    best_params = grid_search.best_params_

    # Model prediction
    model_pred = best_model.predict(X_test)

    # Displaying model results
    print(f"\nModel: {name}")
    print(f"Best Parameters: {best_params}")
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, model_pred))
    print("\nClassification Report:")
    print(classification_report(y_test, model_pred))
```

Explanation:

- Grid Search Setup: A dictionary of models and parameter grids is used to perform hyperparameter tuning. GridSearchCV helps in selecting the best combination of parameters for each model, optimizing for recall.
- Best Model Selection: The best models are chosen based on the grid search results, and the evaluation metrics (confusion matrix, classification report) help compare their performance.

```
Training models:  22%|██        | 2/9 [00:12<00:35,  5.04s/it]

Model: Logistic Regression
Best Parameters: {'C': 10, 'max_iter': 100, 'penalty': 'l1'}
Confusion Matrix:
[[1452  247]
 [  76  251]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.85      0.90      1699
           1       0.50      0.77      0.61       327

    accuracy                           0.84      2026
   macro avg       0.73      0.81      0.75      2026
weighted avg       0.88      0.84      0.85      2026
```

```
Model: AdaBoost
Best Parameters: {'learning_rate': 10, 'n_estimators': 50}
Confusion Matrix:
[[ 198 1501]
 [   1  326]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.12      0.21      1699
           1       0.18      1.00      0.30       327

    accuracy                           0.26      2026
   macro avg       0.59      0.56      0.26      2026
weighted avg       0.86      0.26      0.22      2026


Training models:  78%|███████▊  | 7/9 [59:31<34:37, 1038.51s/it]
```

```
Model: Random Forest
Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Confusion Matrix:
[[1609   90]
 [  32  295]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96      1699
           1       0.77      0.90      0.83       327

    accuracy                           0.94      2026
   macro avg       0.87      0.92      0.90      2026
weighted avg       0.95      0.94      0.94      2026


Training models:  67%|██████▋   | 6/9 [12:29<08:37, 172.58s/it]
```

```
Model: K-Nearest Neighbors
Best Parameters: {'algorithm': 'auto', 'n_neighbors': 3, 'weights': 'distance'}
Confusion Matrix:
[[1445  254]
 [  71  256]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.85      0.90      1699
           1       0.50      0.78      0.61       327

    accuracy                           0.84      2026
   macro avg       0.73      0.82      0.76      2026
weighted avg       0.88      0.84      0.85      2026


Training models:  44%|████     | 4/9 [01:22<01:53, 22.77s/it]
```

```
Model: Naive Bayes
Best Parameters: {}
Confusion Matrix:
[[1335  364]
 [  99  228]]

Classification Report:
...
    accuracy                           0.77      2026
   macro avg       0.66      0.74      0.67      2026
weighted avg       0.84      0.77      0.79      2026
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
Training models:  33%|███      | 3/9 [01:09<02:53, 28.89s/it]
```

```
Model: Decision Tree Classifier
Best Parameters: {'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 4, 'min_samples_split': 2}
Confusion Matrix:
[[1600   99]
 [  47  280]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.96      1699
           1       0.74      0.86      0.79       327

    accuracy                           0.93      2026
   macro avg       0.86      0.90      0.87      2026
weighted avg       0.93      0.93      0.93      2026


Training models:  56%|█████    | 5/9 [11:48<16:00, 240.24s/it]
```

```
Model: Gradient Boosting
Best Parameters: {'learning_rate': 0.2, 'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 200}
Confusion Matrix:
[[1616   83]
 [  19  308]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97      1699
           1       0.79      0.94      0.86       327

    accuracy                           0.95      2026
   macro avg       0.89      0.95      0.91      2026
weighted avg       0.96      0.95      0.95      2026


Training models:  89%|████████▉ | 8/9 [1:02:32<12:45, 765.50s/it]
```

```
Model: XGBoost
Best Parameters: {'colsample_bytree': 0.6, 'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 100, 'subsample': 0.8}
Confusion Matrix:
[[1620   79]
 [  30  297]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.97      1699
           1       0.79      0.91      0.84       327

    accuracy                           0.95      2026
   macro avg       0.89      0.93      0.91      2026
weighted avg       0.95      0.95      0.95      2026

[LightGBM] [Info] Number of positive: 5777, number of negative: 5427
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001165 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1795
[LightGBM] [Info] Number of data points in the train set: 11204, number of used features: 18
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.515619 -> initscore=0.062498
[LightGBM] [Info] Start training from score 0.062498
Training models: 100%|██████████| 9/9 [1:06:20<00:00, 442.29s/it]
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
Model: LightGBM
Best Parameters: {'learning_rate': 0.2, 'max_depth': 10, 'min_child_samples': 100, 'n_estimators': 200, 'num_leaves': 31}
Confusion Matrix:
[[1619   80]
 [  26  301]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.97      1699
           1       0.79      0.92      0.85       327

    accuracy                           0.95      2026
   macro avg       0.89      0.94      0.91      2026
weighted avg       0.95      0.95      0.95      2026
```

Model Training Without Grid Search:

- After grid search, the best parameters are used to define the final models.
- These models are trained directly on the training data and saved as `.pkl` files using Python's `pickle` module for future use.
- After training, the results (confusion matrix and classification report) are displayed for each model.

```
# Dictionary of models with best parameters from previous grid search results
models_params = {
    "Logistic Regression": LogisticRegression(C=10, max_iter=100, penalty='l1',
solver='liblinear'),

    "Naive Bayes": GaussianNB(),

    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=3, weights='distance',
algorithm='auto'),

    "Decision Tree Classifier": DecisionTreeClassifier(criterion='entropy',
max_depth=20, min_samples_leaf=4, min_samples_split=2),

    "Random Forest": RandomForestClassifier(criterion='entropy', max_depth=None,
min_samples_leaf=1, min_samples_split=2, n_estimators=200),

    "AdaBoost": AdaBoostClassifier(learning_rate=10, n_estimators=50),

    "Gradient Boosting": GradientBoostingClassifier(learning_rate=0.2, max_depth=5,
min_samples_leaf=4, min_samples_split=5, n_estimators=2000),

    "XGBoost": XGBClassifier(colsample_bytree=0.6, learning_rate=0.2, max_depth=7,
n_estimators=100, subsample=0.8, use_label_encoder=False, eval_metric='logloss'),
    "LightGBM": LGBMClassifier(learning_rate=0.2, max_depth=10, num_leaves=31,
min_child_samples=100, n_estimators=200)
}
trained_models = {}
# Model training without grid search
for name, model in tqdm(models_params.items(), desc="Training models",
total=len(models_params)):
    # Fit the model
    model.fit(X_train, y_train)
    # Model prediction
    model_pred = model.predict(X_test)
    # Save the trained model to the dictionary
    trained_models[name] = model
    # Save the model to a PKL file
    with open(f'{name}.pkl', 'wb') as f:
        pickle.dump(model, f)
    # Displaying model results
    print(f"\nModel: {name}")
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, model_pred))
    print("\nClassification Report:")
    print(classification_report(y_test, model_pred))
```

```
Training models:  11%|█         | 1/9 [00:00<00:01,  5.00it/s]

Model: Logistic Regression
Confusion Matrix:
[[1452  247]
 [  76  251]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.85      0.90      1699
           1       0.50      0.77      0.61       327

    accuracy                           0.84      2026
   macro avg       0.73      0.81      0.75      2026
weighted avg       0.88      0.84      0.85      2026
```

```
Model: K-Nearest Neighbors
Confusion Matrix:
[[1443  256]
 [  71  256]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.85      0.90      1699
           1       0.50      0.78      0.61       327

    accuracy                           0.84      2026
   macro avg       0.73      0.82      0.75      2026
weighted avg       0.88      0.84      0.85      2026


Training models:  44%|████▍     | 4/9 [00:00<00:00,  5.02it/s]
```

```
Model: Naive Bayes
Confusion Matrix:
[[1336  363]
 [  99  228]]

Classification Report:
              precision    recall  f1-score   support
...
    accuracy                           0.77      2026
   macro avg       0.66      0.74      0.67      2026
weighted avg       0.84      0.77      0.80      2026
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
Training models:  33%|███▎      | 3/9 [00:00<00:01,  5.11it/s]
```

```
Model: Random Forest
Confusion Matrix:
[[1611   88]
 [  32  295]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96      1699
           1       0.77      0.90      0.83       327

    accuracy                           0.94      2026
   macro avg       0.88      0.93      0.90      2026
weighted avg       0.95      0.94      0.94      2026


Training models:  67%|██████▋   | 6/9 [00:07<00:05,  1.77s/it]
```

```
Model: Decision Tree Classifier
Confusion Matrix:
[[1600   99]
 [  43  284]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.96      1699
           1       0.74      0.87      0.80       327

    accuracy                           0.93      2026
   macro avg       0.86      0.91      0.88      2026
weighted avg       0.94      0.93      0.93      2026


Training models:  56%|█████▌    | 5/9 [00:06<00:08,  2.15s/it]
```

```
Model: AdaBoost
Confusion Matrix:
[[ 198 1501]
 [   1  326]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.12      0.21      1699
           1       0.18      1.00      0.30       327

    accuracy                           0.26      2026
   macro avg       0.59      0.56      0.26      2026
weighted avg       0.86      0.26      0.22      2026


Training models:  78%|███████▊  | 7/9 [01:04<00:38, 19.09s/it]
```

```
Model: Gradient Boosting
Confusion Matrix:
[[1620   79]
 [  22  305]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.95      0.97      1699
           1       0.79      0.93      0.86       327

    accuracy                           0.95      2026
   macro avg       0.89      0.94      0.91      2026
weighted avg       0.96      0.95      0.95      2026


Training models:  89%|████████  | 8/9 [01:05<00:13, 13.28s/it]
```

```
Model: LightGBM
Confusion Matrix:
[[1618   81]
 [  26  301]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.97      1699
           1       0.79      0.92      0.85       327

    accuracy                           0.95      2026
   macro avg       0.89      0.94      0.91      2026
weighted avg       0.95      0.95      0.95      2026
```

```
Model: XGBoost
Confusion Matrix:
[[1620   79]
 [  28  299]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.97      1699
           1       0.79      0.91      0.85       327

    accuracy                           0.95      2026
   macro avg       0.89      0.93      0.91      2026
weighted avg       0.95      0.95      0.95      2026

[LightGBM] [Info] Number of positive: 5777, number of negative: 5427
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001313 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2050
[LightGBM] [Info] Number of data points in the train set: 11204, number of used features: 19
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.515619 -> initscore=0.062498
[LightGBM] [Info] Start training from score 0.062498
Training models: 100%|██████████| 9/9 [01:05<00:00,  7.30s/it]
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Evaluation Metrics Used

- Accuracy: The ratio of correctly predicted instances to the total instances.
- Precision: The ratio of correctly predicted positive observations to the total predicted positives.
- Recall (Sensitivity): The ratio of correctly predicted positive observations to all the observations in the actual class.
- F1-Score: The weighted average of Precision and Recall.

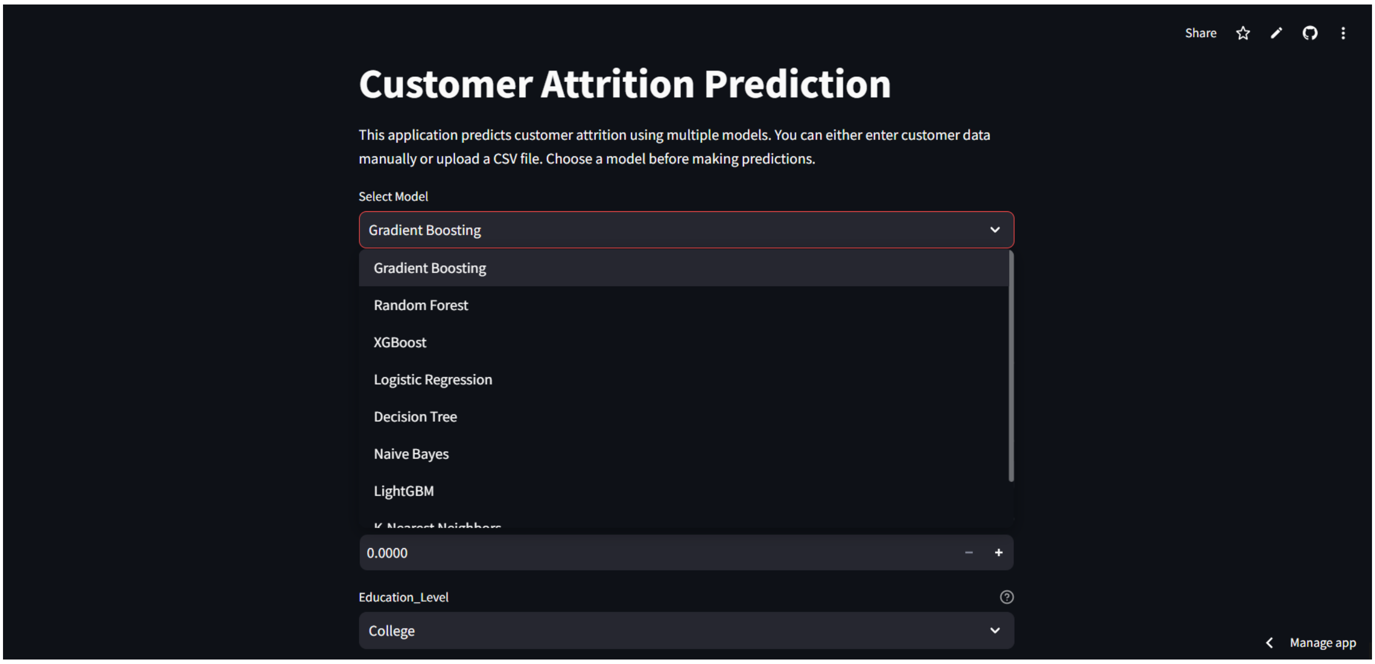
Comparison of Model Performance

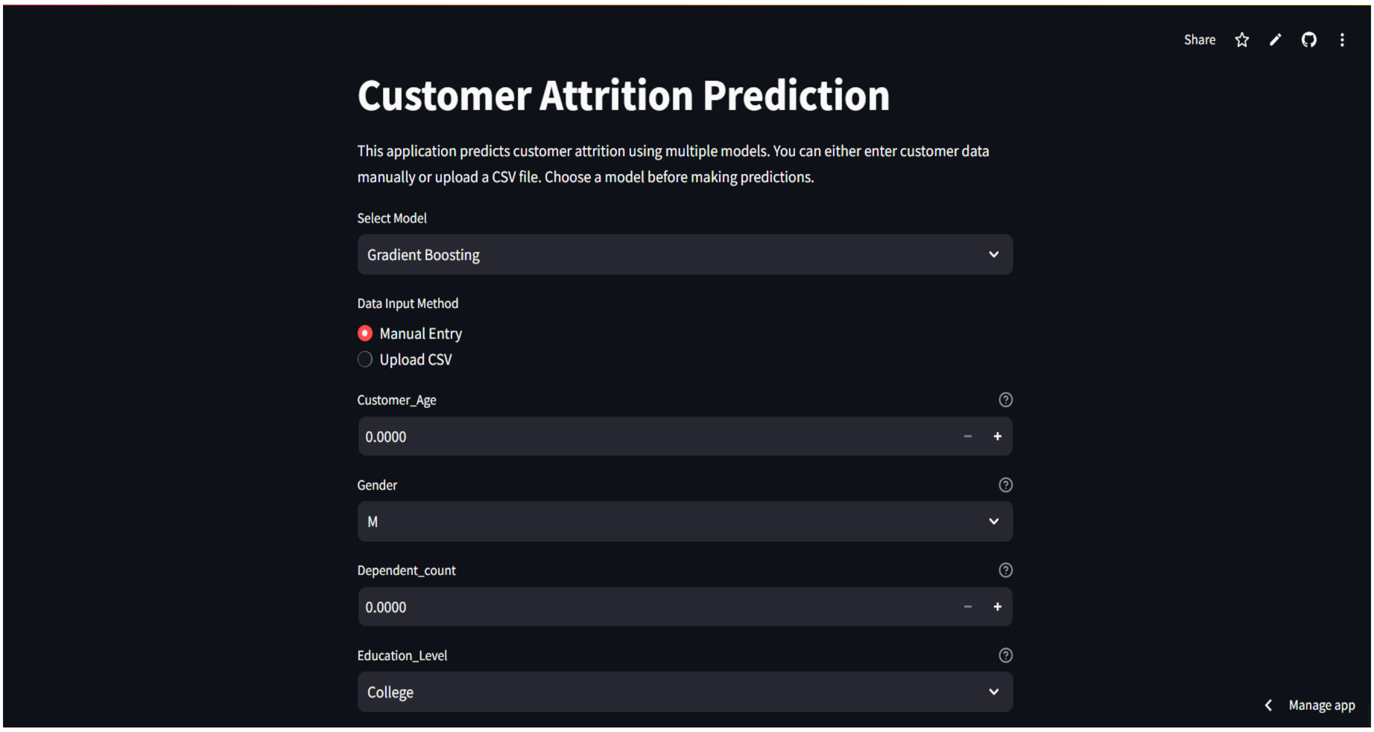
Model	Accuracy	Precision (Class 1)	Recall (Class 1)	F1-Score (Class 1)
Logistic Regression	0.84	0.50	0.77	0.61
Naive Bayes	0.77	0.39	0.70	0.50
K-Nearest Neighbors	0.84	0.50	0.78	0.61
Decision Tree	0.93	0.74	0.87	0.80
Random Forest	0.94	0.77	0.90	0.83
AdaBoost	0.26	0.18	1.00	0.30
Gradient Boosting	0.95	0.79	0.93	0.86
XGBoost	0.95	0.79	0.91	0.85
LightGBM	0.95	0.79	0.92	0.85

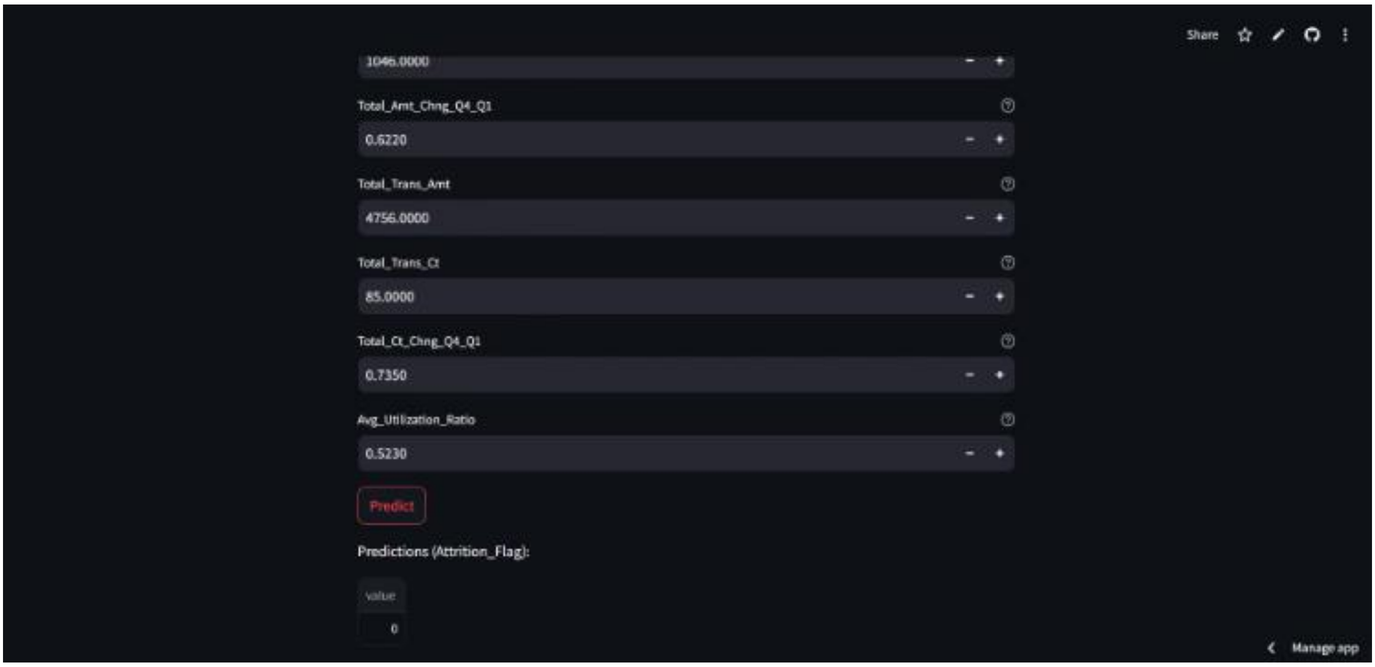
8. Results

After evaluating multiple models, XGBoost and Random Forest demonstrated the highest accuracy and recall. These models are most suitable for predicting customer churn due to their robustness and ability to handle non-linear relationships in the data.

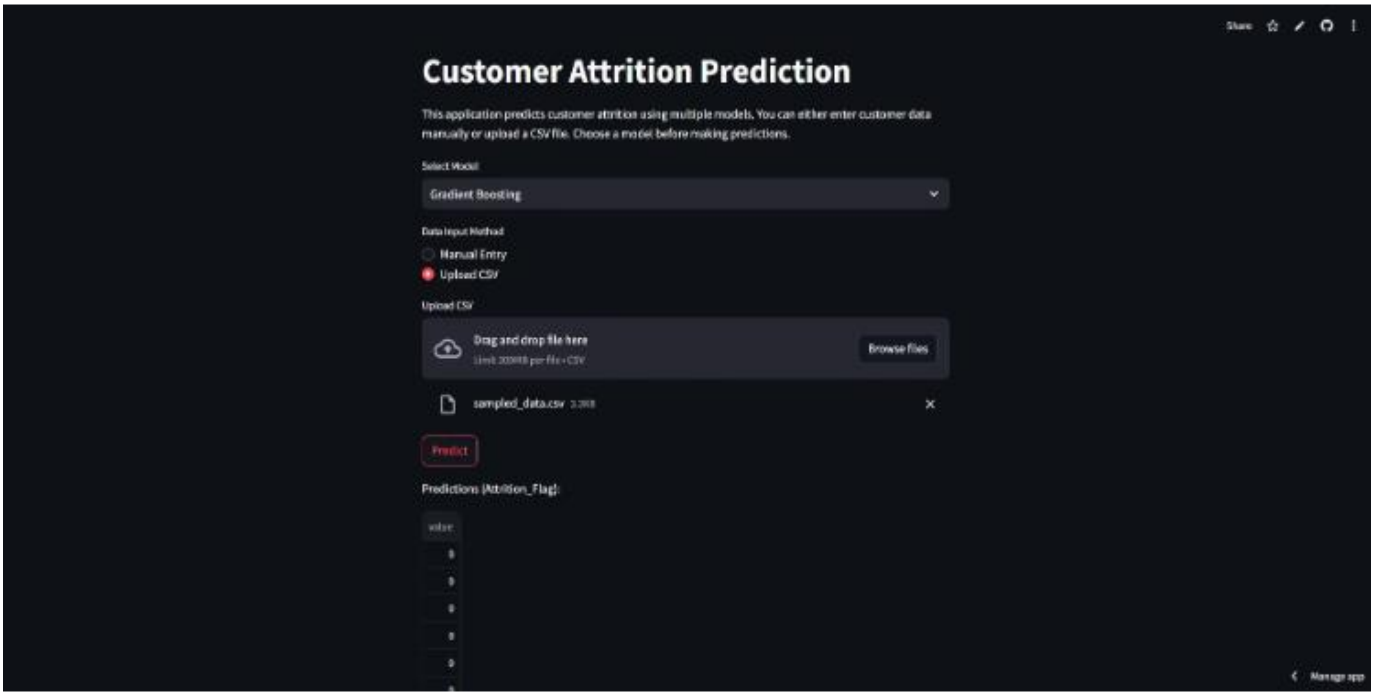
App Interface







Entering Manually



Upload a CSV file

9. Conclusion

Recap of the Project and Its Outcomes

The Card Churn Prediction project successfully developed a high-accuracy predictive model for customer churn within a credit card company. After evaluating various models, the Random Forest model demonstrated the best performance, followed closely by Gradient Boosting, both of which outperformed logistic regression in accuracy and precision. These advanced models offer banks valuable insights into customer behavior, empowering them to implement proactive strategies to retain their most valuable clients. Future enhancements could involve integrating real-time transaction data to further refine the model's predictive capabilities.

Future Work and Recommendations

- Implementing real-time churn prediction systems for timely interventions.
- Exploring advanced models like deep learning and ensemble techniques.
- Continuously updating the model with new data to maintain accuracy.

10. References

1. "Machine Learning Yearning" by Andrew Ng - This book offers insights into the process of building machine learning systems and can provide a great foundation for understanding the steps taken in this project.
2. "Pattern Recognition and Machine Learning" by Christopher M. Bishop - This book covers various machine learning algorithms and techniques, including those applied in this project.
3. "Data Mining: Practical Machine Learning Tools and Techniques" by Ian H. Witten, Eibe Frank, and Mark A. Hall - A comprehensive guide to data mining and machine learning, which includes practical examples similar to those used in this project.
4. "An Introduction to Statistical Learning" by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani - Provides an accessible overview of many important machine learning techniques.
5. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron - A practical guide to implementing machine learning models, including code snippets similar to those used in this project.
6. Verbraken, T., Verbeke, W., Baesens, B., & Bravo, C. (2013). Comparative analysis of churn prediction techniques. Expert Systems with Applications, 39(18), 13171-13179.
7. Keramati, A., Jafari-Marandi, R., Aliannejadi, M., et al. (2014). Improved churn prediction in telecommunication industry using data mining techniques. Applied Soft Computing, 24, 994-1012.
8. Fader, P. S., Hardie, B. G., & Lee, K. L. (2015). Counting your customers the easy way: An alternative to the Pareto/NBD model. Marketing Science, 24(2), 275-284.

Research Papers:-

- Predicting Customer Churn: A Case Study in the Telecom Industry" by Ahmad et al.
- A Comparative Study of Customer Churn Prediction Techniques and Applications" by Vafeiadis et al.
- Customer Churn Analysis: A Study on Customer Behavior and Factors" by Wong etal.