Project 3 : AIML – UT, Austin

Advanced Machine Learning: Credit Card Users Churn Prediction

Date : November 20, 2024

Contents / Agenda

- Executive Summary
- Business Problem Overview and Solution Approach
- EDA Results
- Data Preprocessing
- Model performance summary for hyperparameter tuning.
- Appendix

Executive Summary

Total Transaction Count, **Total Revolving Balance**, **Total Relationship Count**, **Total Transaction Amount**, and **Total Count Change from Q4 to Q1** are identified as the top five reasons for customer attrition at Thera Bank, here are tailored recommendations to address each factor and reduce the likelihood of customer churn:

Targeted Promotions and Rewards, Seasonal or Limited-Time Offers, Balance Transfer Offers, Customer Engagement, Loyalty Programs, Exclusive Services for Multi-Product Customers, Increase Engagement During the Holiday Season, Customer Segmentation are some of the recommendations, we can think of to increase Thera Bank's credit card customers.

Business Problem Overview and Solution Approach

- **Business Problem:**

Thera Bank has recently observed a significant decline in the number of credit card users, which could lead to a loss of revenue. Credit cards are a critical source of income for the bank due to various fees charged, including annual fees, balance transfer fees, late payment fees, and others. Customers discontinuing their credit card services could significantly impact the bank's profitability.

The objective is to predict which customers are at risk of leaving (attriting) the credit card services, identify the reasons for attrition, and enable the bank to take proactive actions to improve customer retention.

- **Key Challenge:**

To develop a **classification model** that will accurately predict whether a customer will leave the bank’s credit card services ("Attrited Customer") or remain an active user ("Existing Customer"). By understanding the key drivers of attrition, the bank can tailor its services, offerings, and communication strategies to reduce customer churn and retain valuable customers.

EDA Results

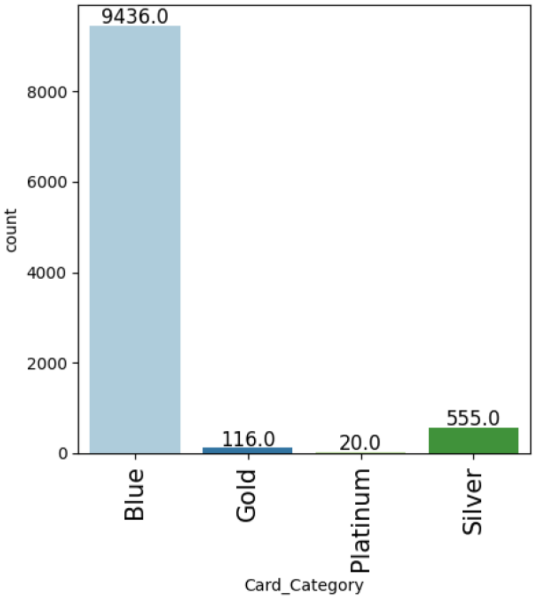
- The majority of customers fall within the age range of 30 to 60 years.
- On average, customers have been using the bank's services for 36 months (3 years).
- The average credit limit for customers is $8,600, and they spend an average of $4,400 monthly, with an average of 64 transactions per month.

	count	mean	std	min	25%	50%	75%	max
CLIENTNUM	10127.000	739177606.334	36903783.450	708082083.000	713036770.500	717926358.000	773143533.000	828343083.000
Customer_Age	10127.000	46.326	8.017	26.000	41.000	46.000	52.000	73.000
Dependent_count	10127.000	2.346	1.299	0.000	1.000	2.000	3.000	5.000
Months_on_book	10127.000	35.928	7.986	13.000	31.000	36.000	40.000	56.000
Total_Relationship_Count	10127.000	3.813	1.554	1.000	3.000	4.000	5.000	6.000
Months_Inactive_12_mon	10127.000	2.341	1.011	0.000	2.000	2.000	3.000	6.000
Contacts_Count_12_mon	10127.000	2.455	1.106	0.000	2.000	2.000	3.000	6.000
Credit_Limit	10127.000	8631.954	9088.777	1438.300	2555.000	4549.000	11067.500	34516.000
Total_Revolving_Bal	10127.000	1162.814	814.987	0.000	359.000	1276.000	1784.000	2517.000
Avg_Open_To_Buy	10127.000	7469.140	9090.685	3.000	1324.500	3474.000	9859.000	34516.000
Total_Amt_Chng_Q4_Q1	10127.000	0.760	0.219	0.000	0.631	0.736	0.859	3.397
Total_Trans_Amt	10127.000	4404.086	3397.129	510.000	2155.500	3899.000	4741.000	18484.000
Total_Trans_Ct	10127.000	64.859	23.473	10.000	45.000	67.000	81.000	139.000
Total_Ct_Chng_Q4_Q1	10127.000	0.712	0.238	0.000	0.582	0.702	0.818	3.714
Avg_Utilization_Ratio	10127.000	0.275	0.276	0.000	0.023	0.176	0.503	0.999

Link to Appendix slide on data background check

EDA Results

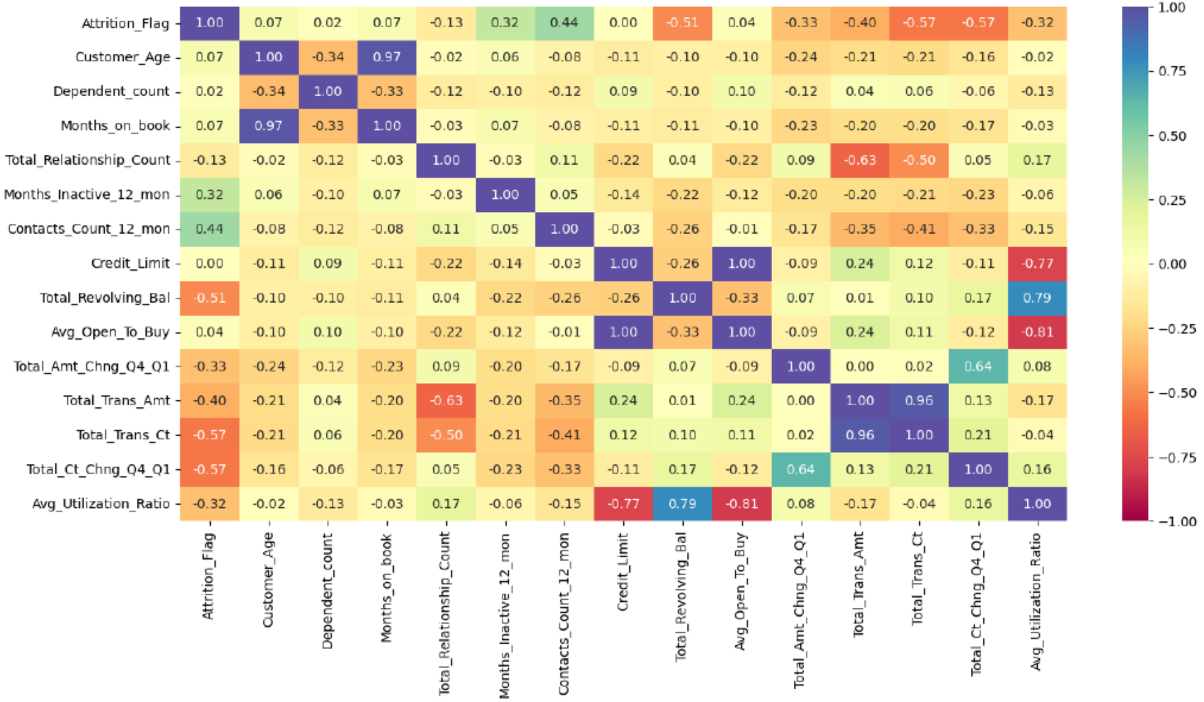
The majority of customers hold a blue credit card. We should conduct a survey to understand why customers prefer not to choose gold, silver or platinum credit cards.



Link to Appendix slide on data background check

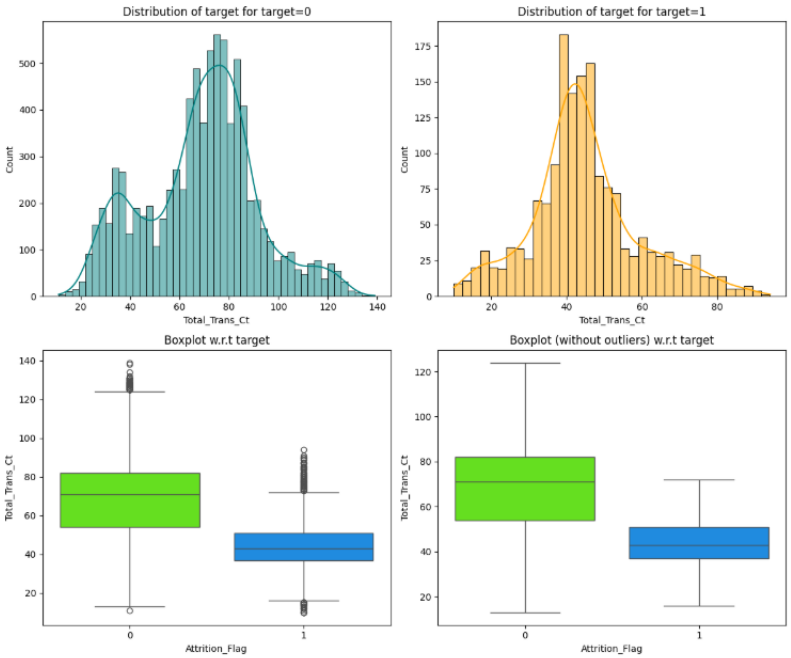
EDA Results

- Total_Amt_Chng_04_01 Vs. Attrition Flag : A moderate negative correlation indicates that customers who significantly increased their spending from Q4 to Q1 are less likely to churn.
- Total_Trans_Ct Vs. Attrition Flag: A strong negative correlation suggests that customers who make more transactions are less likely to churn.
- Total_Ct_Chng_04_01 Vs. Attrition Flag : A strong negative correlation suggests that customers who significantly increased their transaction frequency from Q4 to Q1 are less likely to churn.
- Based on the other stacked plots, it appears that marital status, education level, and dependent count have a minimal effect on the attrition rate.



EDA Results

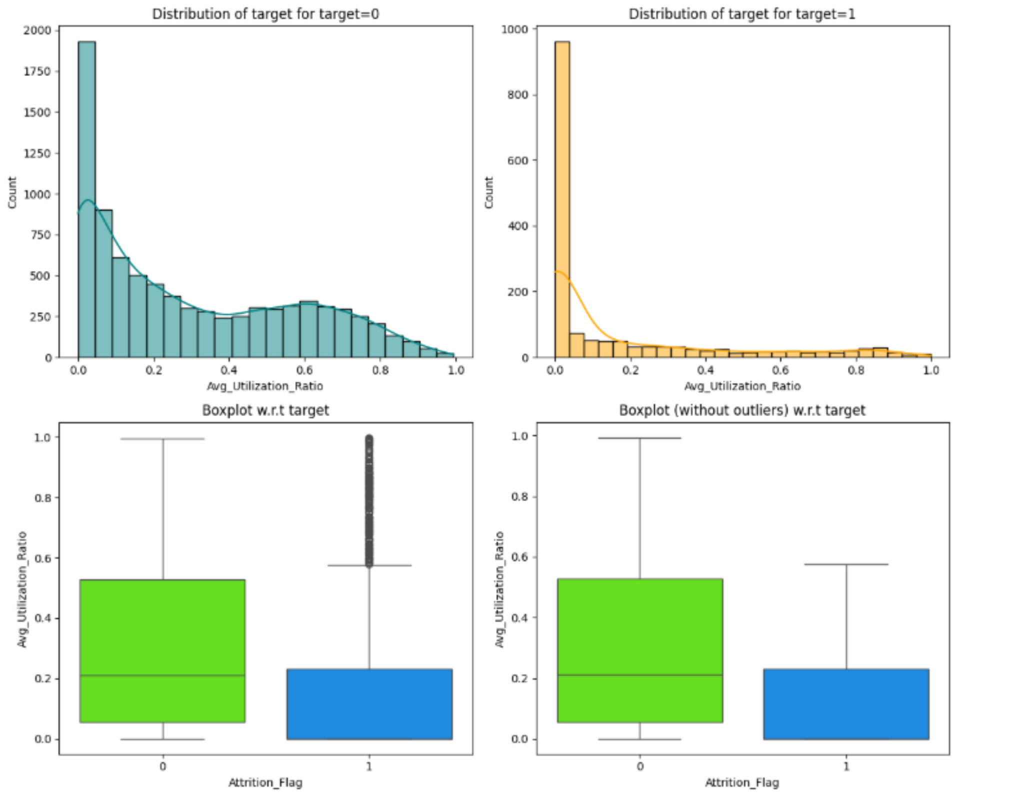
- The total transaction count has an impact on the attrition rate.



Link to Appendix slide on data background check

EDA Results

- Customers with more credit card utilization are less likely to leave the bank's services.



Data Preprocessing

- Duplicate value check – There are no duplicate values in the dataset.
- Missing value treatment – There are missing values in Education level, Income categoty and Martitial status category. With pandas dumties, those categories are encoded. In income category, the abc value is replaced with NA.
- Outlier check (treatment if needed) - Some individuals have very high credit limits, and their transactions involve large amounts. No outlier treatment is required for these cases.
- Feature engineering – We are dividing the data into 80:20 – 80% goes for training the model and 20% goes for testing.
- Data preparation for modeling – it is a classification problem. We need to predict correctly the customers which are likely to attrite the service. Also, we need to make sure that the model reduce False Negative.

Note: *You can use more than one slide if needed*

Model Performance Summary

- Summary of performance metrics for training and validation data in tabular format for comparison for tuned models

Training performance comparison:

	Gradient boosting trained with Undersampled data	Gradient boosting trained with Original data	AdaBoost trained with Undersampled data	AdaBoost trained with Original data	XGBoost trained with Undersampled data	XGBoost trained with Original data
Accuracy	0.976	0.928	0.973	0.985	0.774	0.977
Recall	0.980	0.862	0.978	0.934	1.000	0.992
Precision	0.972	0.992	0.969	0.969	0.689	0.880
F1	0.976	0.922	0.974	0.951	0.816	0.933

Validation performance comparison:

	Gradient boosting validated with Undersampled data	Gradient boosting validated with Original data	AdaBoost validated with Undersampled data	AdaBoost validated with Original data	XGBBoost validated with Undersample data	XGBBoost validated with Original data
Accuracy	0.943	0.959	0.945	0.966	0.542	0.949
Recall	0.946	0.784	0.959	0.892	1.000	0.946
Precision	0.737	0.921	0.740	0.880	0.242	0.761
F1	0.828	0.847	0.835	0.886	0.389	0.843

Model Performance Summary

- The XGBoost model demonstrated strong performance on both the training and validation datasets, achieving a recall rate of over 94%, which is a critical metric for banks to retain potential customers. The GBoost model performed well in terms of recall on the training dataset but showed slightly weaker results on the validation dataset. Meanwhile, the AdaBoost model performed well on both the training and validation datasets.

- Overall, the XGBoost model with tuned hyperparameters emerged as the best option.

- To further enhance the recall metric of the XGBoost model, we can apply oversampling techniques, which can boost the recall rate to nearly 100%.

[Link to Appendix slide on model assumptions]

APPENDIX

Model Performance Summary (original data)

Observations:

Training: All three models exhibit near-perfect performance on the training data, with Random Forest and XGBoost achieving a recall score of 1.0.

Validation:
XGBoost outperforms both Bagging and Random Forest on the validation set, demonstrating a higher ability to generalize to unseen data.
Bagging shows a significant drop in performance from training to validation, suggesting potential overfitting on the training data.
Random Forest exhibits a similar trend to Bagging, with a noticeable decline in performance on the validation set.

Conclusion:
Based on the validation performance, XGBoost appears to be the most robust model among the three, as it demonstrates the best ability to generalize to unseen data.

Training Performance		
Bagging	**Random Forest**	**XGBoost**
0.98	1	1
Validation Performance		
Bagging	**Random Forest**	**XGBoost**
0.85	0.76	0.93

```
Training Performance:

Bagging: 0.98
Random forest: 1.0
XGBoost: 1.0

Validation Performance:

Bagging: 0.8513513513513513
Random forest: 0.7567567567567568
XGBoost: 0.9324324324324325
```

Link to Appendix slide on model assumptions

Model Performance Summary (oversampled data)

- **Observations:**

Training: All three models exhibit similar performance on the training data, with Bagging having the highest recall score.
Validation: The performance of all three models remains consistent with their training performance, indicating that they are not overfitting.

- **Conclusion:**

Based on the validation performance, XGB appears to be the most robust model among the three, as it demonstrates the highest recall score on both training and validation sets.

Training Performance		
Bagging	**Random Forest**	**XGBoost**
0.9	0.83	0.92
Validation Performance		
Bagging	**Random Forest**	**XGBoost**
0.9	0.83	0.92

```
Training Performance:

Bagging: 0.9054054054054054
Random forest: 0.8378378378378378
XGBoost: 0.918918918918919

Validation Performance:

Bagging: 0.9054054054054054
Random forest: 0.8378378378378378
XGBoost: 0.918918918918919
```

Link to Appendix slide on model assumptions

Model Performance Summary (undersampled data)

- **Observations:**

Training: All three models exhibit similar performance on the training data, with XGBoost having the highest recall score.
Validation: The performance of all three models remains consistent with their training performance, indicating that they are not overfitting.

- **Conclusion:**

Based on the validation performance, XGBoost appears to be the most robust model among the three, as it demonstrates the highest recall score on both training and validation sets.

Training Performance		
Bagging	Random Forest	XGBoost
0.91	0.94	0.97
Validation Performance		
Bagging	Random Forest	XGBoost
0.91	0.94	0.97

```
Training Performance:

Bagging: 0.918918918918919
Random forest: 0.9459459459459459
XGBoost: 0.972972972972973

Validation Performance:

Bagging: 0.918918918918919
Random forest: 0.9459459459459459
XGBoost: 0.972972972972973
```

Link to Appendix slide on model assumptions

Happy Learning !

Problem Statement

Business Context

The Thera bank recently saw a steep decline in the number of users of their credit card, credit cards are a good source of income for banks because of different kinds of fees charged by the banks like annual fees, balance transfer fees, and cash advance fees, late payment fees, foreign transaction fees, and others. Some fees are charged to every user irrespective of usage, while others are charged under specified circumstances.

Customers' leaving credit cards services would lead bank to loss, so the bank wants to analyze the data of customers and identify the customers who will leave their credit card services and reason for same – so that bank could improve upon those areas

You as a Data scientist at Thera bank need to come up with a classification model that will help the bank improve its services so that customers do not renounce their credit cards

Data Description

- **CLIENTNUM: Client number. Unique identifier for the customer holding the account**
- **Attrition_Flag: Internal event (customer activity) variable - if the account is closed then "Attrited Customer" else "Existing Customer"**
- **Customer_Age: Age in Years**
- **Gender: Gender of the account holder**
- **Dependent_count: Number of dependents**
- **Education_Level: Educational Qualification of the account holder - Graduate, High School, Unknown, Uneducated, College(refers to college student), Post-Graduate, Doctorate**
- **Marital_Status: Marital Status of the account holder**
- **Income_Category: Annual Income Category of the account holder**
- **Card_Category: Type of Card**
- **Months_on_book: Period of relationship with the bank (in months)**
- **Total_Relationship_Count: Total no. of products held by the customer**
- **Months_Inactive_12_mon: No. of months inactive in the last 12 months**
- **Contacts_Count_12_mon: No. 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 last 12 months)**
- **Total_Amt_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**

What Is a Revolving Balance?

- **If we don't pay the balance of the revolving credit account in full every month, the unpaid portion carries over to the next month. That's called a revolving balance**

What is the Average Open to buy?

- **'Open to Buy' means the amount left on your credit card to use. Now, this column represents the average of this value for the last 12 months.**

What is the Average utilization Ratio?

- **The Avg_Utilization_Ratio represents how much of the available credit the customer spent. This is useful for calculating credit scores.**

Relation b/w Avg_Open_To_Buy, Credit_Limit and Avg_Utilization_Ratio:

- **(Avg_Open_To_Buy / Credit_Limit) + Avg_Utilization_Ratio = 1**

Please read the instructions carefully before starting the project.

This is a commented Jupyter IPython Notebook file in which all the instructions and tasks to be performed are mentioned.

- **Blanks '___' are provided in the notebook that needs to be filled with an appropriate code to get the correct result. With every '___' blank, there is a comment that briefly describes what needs to be filled in the blank space.**
- **Identify the task to be performed correctly, and only then proceed to write the required code.**
- **Fill the code wherever asked by the commented lines like "# write your code here" or "# complete the code". Running incomplete code may throw error.**
- **Please run the codes in a sequential manner from the beginning to avoid any unnecessary errors.**
- **Add the results/observations (wherever mentioned) derived from the analysis in the presentation and submit the same.**

Importing necessary libraries

In [270]:

```
!pip install scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 pandas=
=1.5.3 imbalanced-learn==0.10.1 xgboost==2.0.3 -q --user
```

In [271]:

```
# Installing the libraries with the specified version.
# uncomment and run the following line if Google Colab is being used
!pip install scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 pandas=
=1.5.3 imbalanced-learn==0.10.1 xgboost==2.0.3 -q --user
```

In [272]:

```
# Installing the libraries with the specified version.
# uncomment and run the following lines if Jupyter Notebook is being used
!pip install scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 pandas=
=1.5.3 imblearn==0.12.0 xgboost==2.0.3 -q --user
# !pip install --upgrade -q threadpoolctl
```

```
ERROR: Could not find a version that satisfies the requirement imblearn==0.12.0 (from ver
sions: 0.0)
ERROR: No matching distribution found for imblearn==0.12.0
```

Note: ***After running the above cell, kindly restart the notebook kernel and run all cells sequentially from the start again.***

In [273]:

```
# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# To suppress scientific notations
pd.set_option("display.float_format", lambda x: "%.3f" % x)

# Libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# To tune model, get different metric scores, and split data
from sklearn import metrics
from sklearn.metrics import (
```

```
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
)
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score

# To be used for data scaling and one hot encoding
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder

# To impute missing values
from sklearn.impute import SimpleImputer

# To oversample and undersample data
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

# To do hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV

# To define maximum number of columns to be displayed in a dataframe
pd.set_option("display.max_columns", None)

# To supress scientific notations for a dataframe
pd.set_option("display.float_format", lambda x: "%.3f" % x)

# To help with model building
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (
    AdaBoostClassifier,
    GradientBoostingClassifier,
    RandomForestClassifier,
    BaggingClassifier,
)
from xgboost import XGBClassifier

# To supress warnings
import warnings
warnings.filterwarnings("ignore")
```

Loading the dataset

In [274]:

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.moun
t("/content/drive/", force_remount=True).
```

In [275]:

```
import os as os
folder_path = "/content/drive/MyDrive/AIML_Project3/"
print(os.listdir(folder_path))
```

```
['BankChurners.csv', 'ETMT_Project_Business_Presentation_Template+%281%29.pptx', 'AML_Pro
ject_LearnerNotebook_LowCode.ipynb', 'AML_Project_LearnerNotebook_LowCode_Final.pdf', 'AM
L_Project_LearnerNotebook_LowCode.pdf', 'AML_Project_LearnerNotebook_FullCode.ipynb', 'AM
L_Project_LearnerNotebook_LowCode_Final.ipynb', 'AML_Project_LearnerNotebook_LowCode (2).
ipynb']
```

In [276]:

```
churn = pd.read_csv("/content/drive/MyDrive/AIML_Project3/BankChurners.csv")
```

Data Overview

Data Overview

The initial steps to get an overview of any dataset is to:

- **observe the first few rows of the dataset, to check whether the dataset has been loaded properly or not**
- **get information about the number of rows and columns in the dataset**
- **find out the data types of the columns to ensure that data is stored in the preferred format and the value of each property is as expected.**
- **check the statistical summary of the dataset to get an overview of the numerical columns of the data**

Checking the shape of the dataset

In [277]:

```
# Checking the number of rows and columns in the training data
churn.shape ##  Complete the code to view dimensions of the train data
```

Out[277]:

```
(10127, 21)
```

In [278]:

```
# let's create a copy of the data
data = churn.copy()
```

Displaying the first few rows of the dataset

In [279]:

```
# let's view the first 5 rows of the data
data.head(5)   ##  Complete the code to view top 5 rows of the data
```

Out[279]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category
0	768805383	Existing Customer	45	M	3	High School	Married	$60K-$80K
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K-$120K
3	769911858	Existing Customer	40	F	4	High School	NaN	Less than $40K
4	709106358	Existing Customer	40	M	3	Uneducated	Married	$60K-$80K

In [280]:

```
# let's view the last 5 rows of the data
data.tail(5) ##  Complete the code to view last 5 rows of the data
```

Out[280]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Catego
10122	772366833	Existing Customer	50	M	2	Graduate	Single	$40K-$6
10123	710638233	Attrited Customer	41	M	2	NaN	Divorced	$40K-$6
10124	716506083	Attrited Customer	44	F	1	High School	Married	Less than $4

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Catego
10125	717406983	Attrited Customer	30	M	2	Graduate	NaN	$40K - 60$
10126	714337233	Attrited Customer	43	F	2	Graduate	Married	Less than $4

Checking the data types of the columns for the dataset

In [281]:

```
# let's check the data types of the columns in the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   CLIENTNUM                 10127 non-null  int64
 1   Attrition_Flag            10127 non-null  object
 2   Customer_Age              10127 non-null  int64
 3   Gender                    10127 non-null  object
 4   Dependent_count           10127 non-null  int64
 5   Education_Level           8608 non-null   object
 6   Marital_Status            9378 non-null   object
 7   Income_Category           10127 non-null  object
 8   Card_Category             10127 non-null  object
 9   Months_on_book            10127 non-null  int64
 10  Total_Relationship_Count  10127 non-null  int64
 11  Months_Inactive_12_mon    10127 non-null  int64
 12  Contacts_Count_12_mon     10127 non-null  int64
 13  Credit_Limit              10127 non-null  float64
 14  Total_Revolving_Bal       10127 non-null  int64
 15  Avg_Open_To_Buy           10127 non-null  float64
 16  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 17  Total_Trans_Amt           10127 non-null  int64
 18  Total_Trans_Ct            10127 non-null  int64
 19  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 20  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(5), int64(10), object(6)
memory usage: 1.6+ MB
```

Checking for duplicate values

In [282]:

```
# let's check for duplicate values in the data
data.duplicated().sum() ##  Complete the code to check duplicate entries in the data
```

Out[282]:

```
0
```

Checking for missing values

In [283]:

```
# let's check for missing values in the data
data.isnull().sum()  ##  Complete the code to check missing entries in the train data
```

Out[283]:

	0
CLIENTNUM	0
Attrition_Flag	0

Customer_Age	0
Gender	0
Dependent_count	0
Education_Level	1519
Marital_Status	749
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

Statistical summary of the dataset

In [284]:

```
# let's view the statistical summary of the numerical columns in the data
data.describe().T ##  Complete the code to print the statitical summary of the train data
```

Out[284]:

	count	mean	std	min	25%	50%	75
CLIENTNUM	10127.000	739177606.334	36903783.450	708082083.000	713036770.500	717926358.000	773143533.00
Customer_Age	10127.000	46.326	8.017	26.000	41.000	46.000	52.00
Dependent_count	10127.000	2.346	1.299	0.000	1.000	2.000	3.00
Months_on_book	10127.000	35.928	7.986	13.000	31.000	36.000	40.00
Total_Relationship_Count	10127.000	3.813	1.554	1.000	3.000	4.000	5.00
Months_Inactive_12_mon	10127.000	2.341	1.011	0.000	2.000	2.000	3.00
Contacts_Count_12_mon	10127.000	2.455	1.106	0.000	2.000	2.000	3.00
Credit_Limit	10127.000	8631.954	9088.777	1438.300	2555.000	4549.000	11067.50
Total_Revolving_Bal	10127.000	1162.814	814.987	0.000	359.000	1276.000	1784.00
Avg_Open_To_Buy	10127.000	7469.140	9090.685	3.000	1324.500	3474.000	9859.00
Total_Amt_Chng_Q4_Q1	10127.000	0.760	0.219	0.000	0.631	0.736	0.85
Total_Trans_Amt	10127.000	4404.086	3397.129	510.000	2155.500	3899.000	4741.00
Total_Trans_Ct	10127.000	64.859	23.473	10.000	45.000	67.000	81.00
Total_Ct_Chng_Q4_Q1	10127.000	0.712	0.238	0.000	0.582	0.702	0.81
Avg_Utilization_Ratio	10127.000	0.275	0.276	0.000	0.023	0.176	0.50

In [285]:

```
data.describe(include=["object"]).T
```

Out[285]:

	count	unique	top	freq
Attrition_Flag	10127	2	Existing Customer	8500
Gender	10127	2	F	5358
Education_Level	8608	6	Graduate	3128
Marital_Status	9378	3	Married	4687
Income_Category	10127	6	Less than $40K	3561
Card_Category	10127	4	Blue	9436

In [286]:

```
for i in data.describe(include=["object"]).columns:
    print("Unique values in", i, "are :")
    print(data[i].value_counts())
    print("*" * 50)
```

```
Unique values in Attrition_Flag are :
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
**************************************************
Unique values in Gender are :
F    5358
M    4769
Name: Gender, dtype: int64
**************************************************
Unique values in Education_Level are :
Graduate         3128
High School      2013
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64
**************************************************
Unique values in Marital_Status are :
Married     4687
Single      3943
Divorced     748
Name: Marital_Status, dtype: int64
**************************************************
Unique values in Income_Category are :
Less than $40K    3561
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
abc               1112
$120K +            727
Name: Income_Category, dtype: int64
**************************************************
Unique values in Card_Category are :
Blue        9436
Silver       555
Gold         116
Platinum      20
Name: Card_Category, dtype: int64
**************************************************
```

In [287]:

```
data.head(2)
```

Out[287]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category
0	768805383	Existing Customer	45	M	3	High School	Married	$60K-$80K
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K

In [288]:

```
# CLIENTNUM consists of uniques ID for clients and hence will not add value to the modeli
ng
data.drop(["CLIENTNUM"], axis=1, inplace=True)
```

In [289]:

```
## Encoding Existing and Attrited customers to 0 and 1 respectively, for analysis.
data["Attrition_Flag"].replace("Existing Customer", 0, inplace=True)
data["Attrition_Flag"].replace("Attrited Customer", 1, inplace=True)
```

Exploratory Data Analysis

The below functions need to be defined to carry out the Exploratory Data Analysis.

In [290]:

```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a triangle will indicate the mean value of the colum
n
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [291]:

```
# function to create labeled barplots
```

```python
def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

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

In [292]:

```python
# function to plot stacked bar chart

def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_values(
        by=sorter, ascending=False
```

```
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 1, 5))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

In [293]:

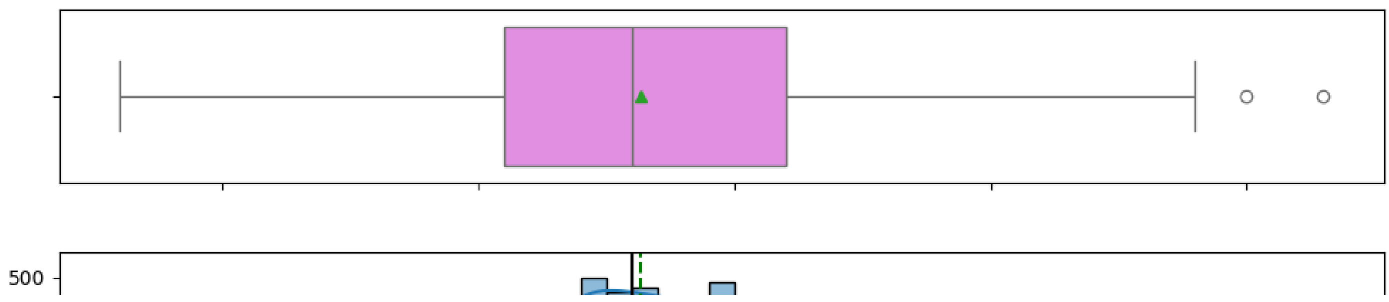
```
### Function to plot distributions

def distribution_plot_wrt_target(data, predictor, target):

    fig, axs = plt.subplots(2, 2, figsize=(12, 10))

    target_uniq = data[target].unique()

    axs[0, 0].set_title("Distribution of target for target=" + str(target_uniq[0]))
    sns.histplot(
        data=data[data[target] == target_uniq[0]],
        x=predictor,
        kde=True,
        ax=axs[0, 0],
        color="teal",
    )

    axs[0, 1].set_title("Distribution of target for target=" + str(target_uniq[1]))
    sns.histplot(
        data=data[data[target] == target_uniq[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
        color="orange",
    )

    axs[1, 0].set_title("Boxplot w.r.t target")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette="gist_rainbow")

    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(
        data=data,
        x=target,
        y=predictor,
        ax=axs[1, 1],
        showfliers=False,
        palette="gist_rainbow",
    )

    plt.tight_layout()
    plt.show()
```

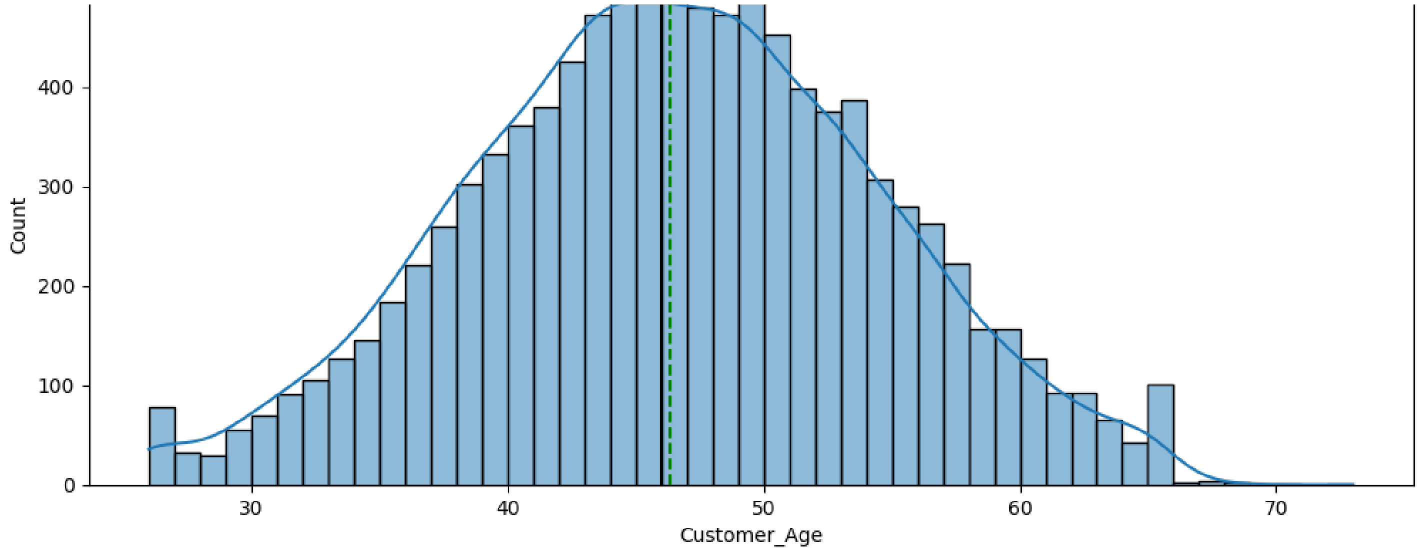
Univariate analysis

`Customer_Age`

In [294]:

```
histogram_boxplot(data, "Customer_Age", kde=True)
```

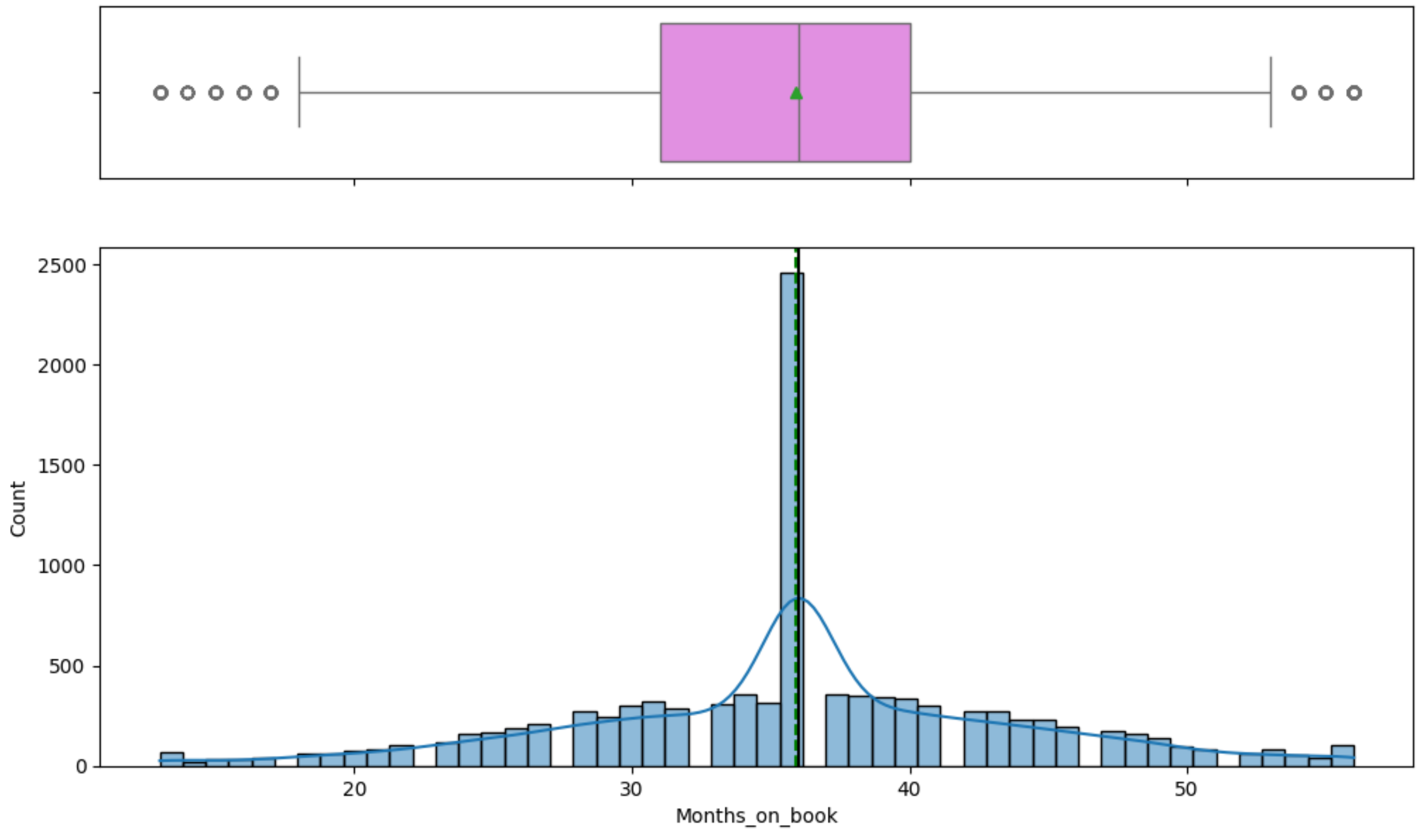




`Months_on_book`

In [295]:

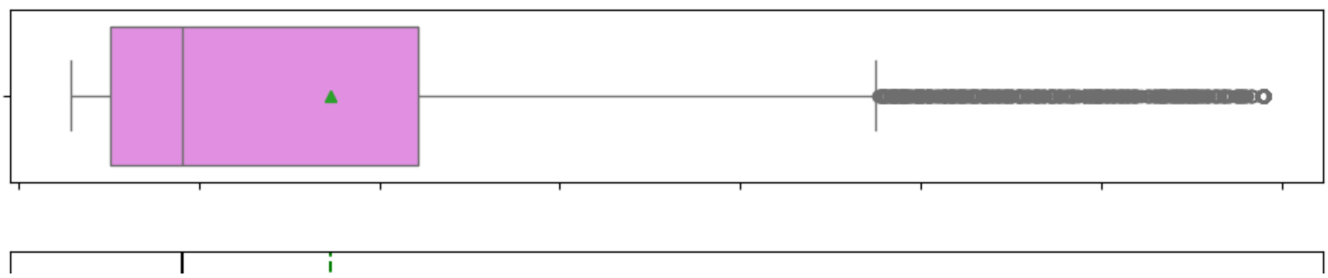
```
histogram_boxplot(data, "Months_on_book", kde=True)  ## Complete the code to create histogram_boxplot for 'New_Price'
```

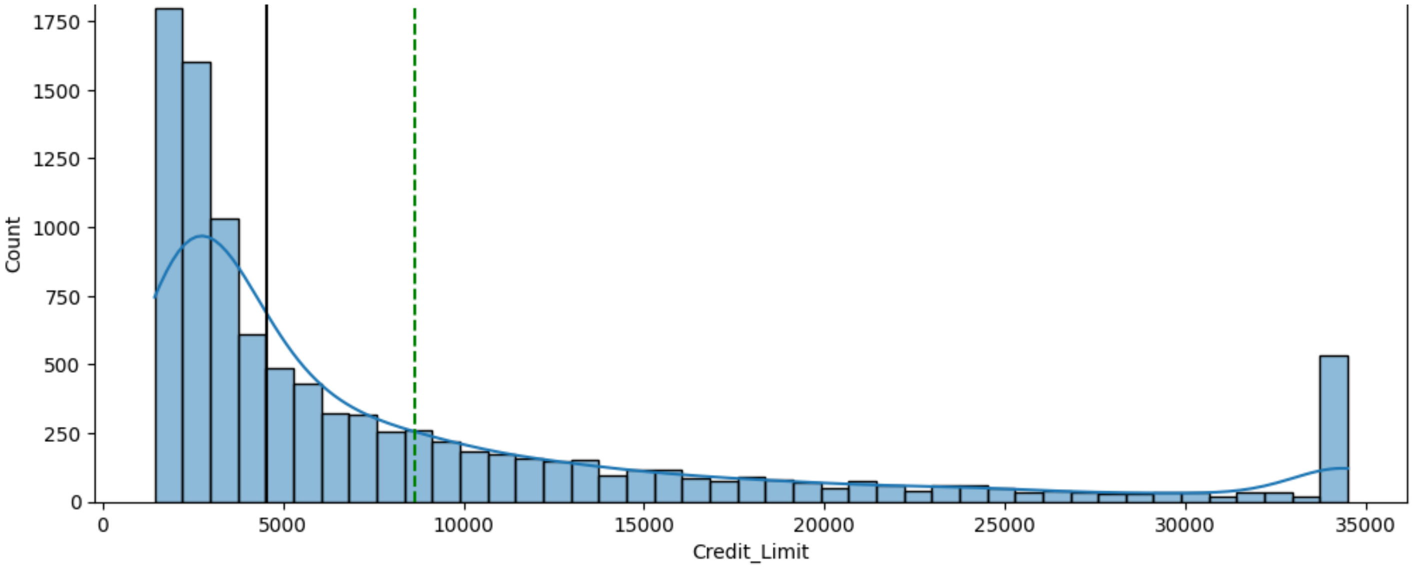


`Credit_Limit`

In [296]:

```
histogram_boxplot(data, "Credit_Limit", kde=True)  ## Complete the code to create histogram_boxplot for 'New_Price'
```

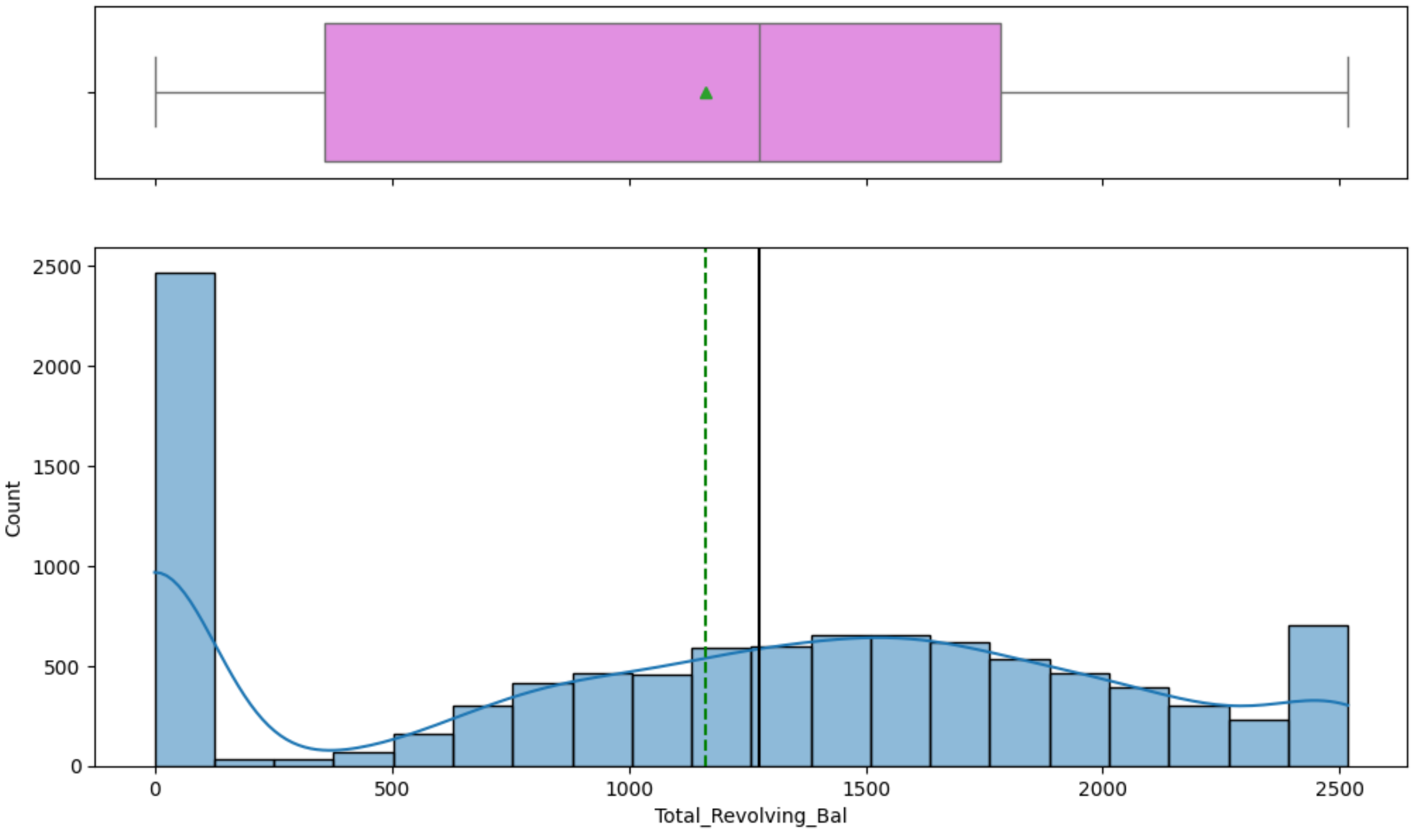




`Total_Revolving_Bal`

In [297]:

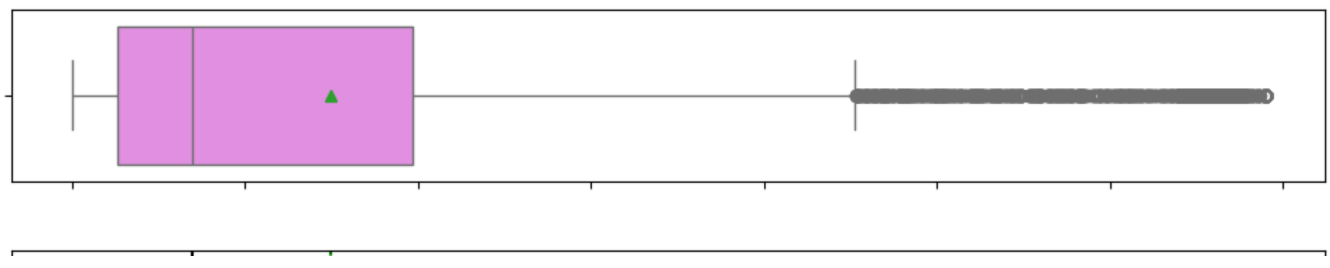
```
histogram_boxplot(data, "Total_Revolving_Bal", kde=True)  ## Complete the code to create histogram_boxplot for 'New_Price'
```

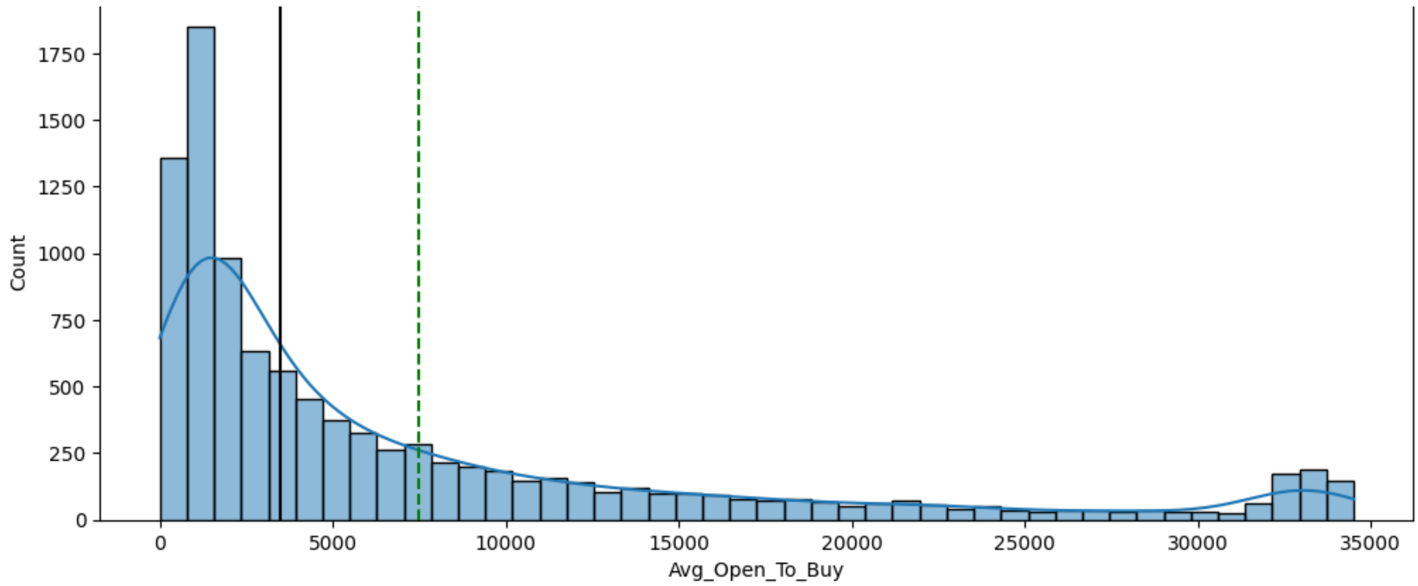


`Avg_Open_To_Buy`

In [298]:

```
histogram_boxplot(data, "Avg_Open_To_Buy", kde=True)  ## Complete the code to create histogram_boxplot for 'New_Price'
```

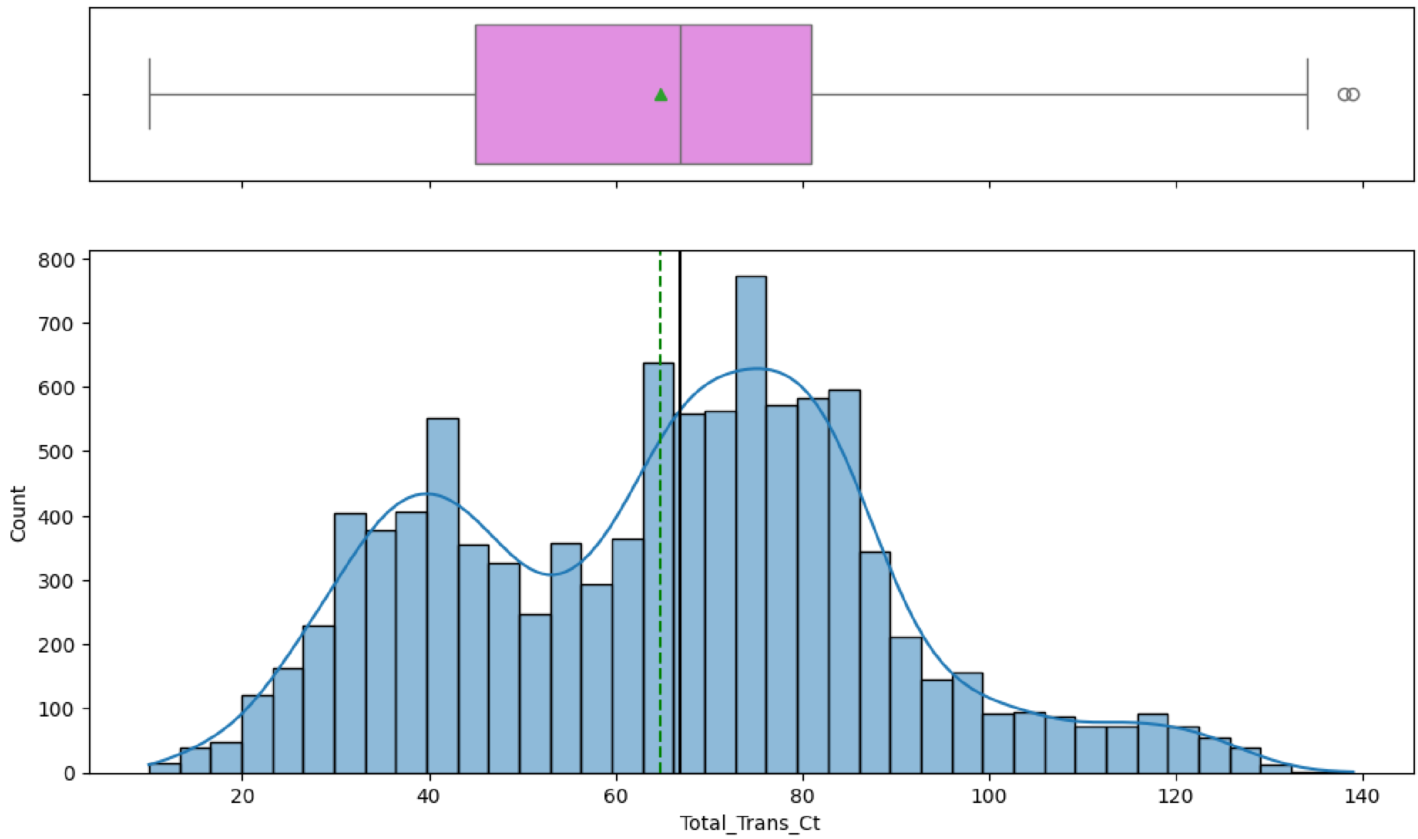




`Total_Trans_Ct`

In [299]:

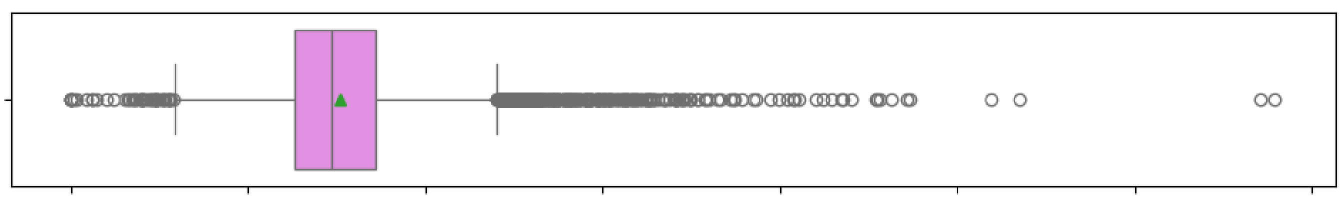
```
histogram_boxplot(data, "Total_Trans_Ct", kde=True)   ## Complete the code to create histo
gram_boxplot for 'New_Price'
```

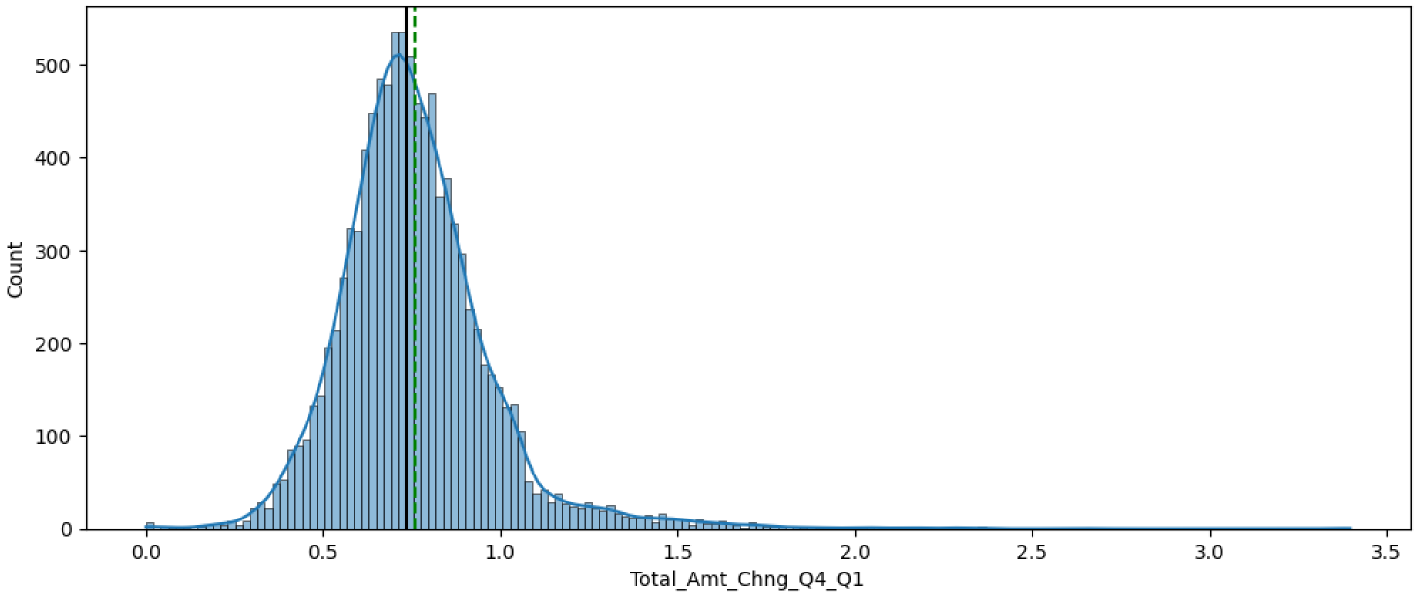


`Total_Amt_Chng_Q4_Q1`

In [300]:

```
histogram_boxplot(data, "Total_Amt_Chng_Q4_Q1", kde=True)   ## Complete the code to create
histogram_boxplot for 'New_Price'
```



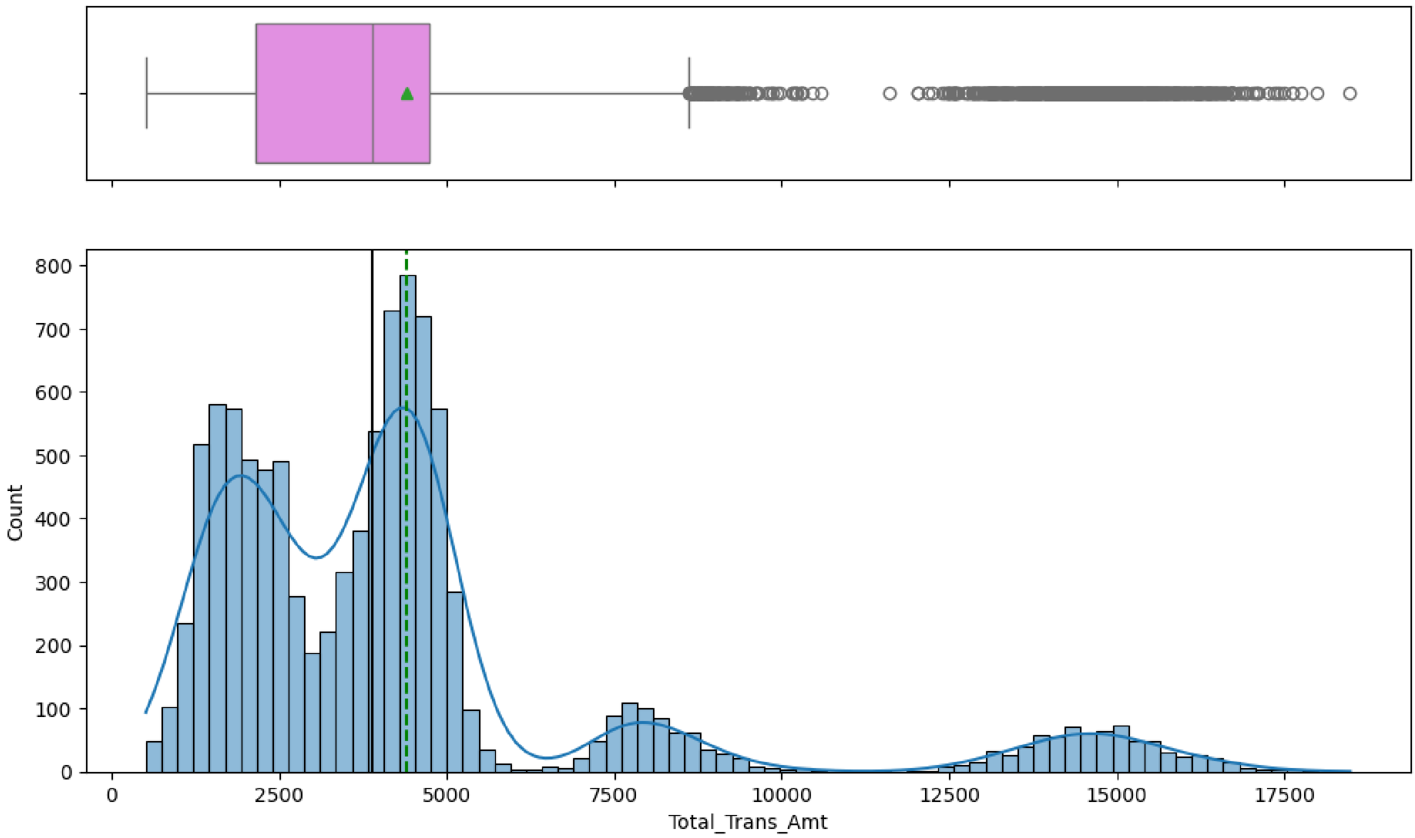


Let's see total transaction amount distributed

`Total_Trans_Amt`

In [301]:

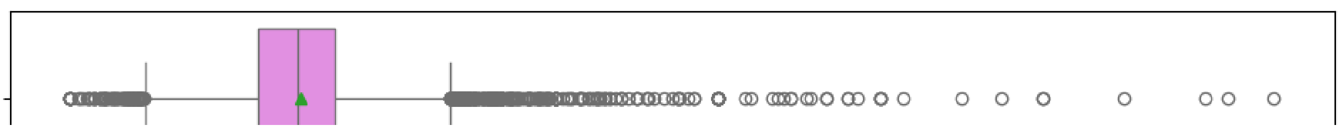
```
histogram_boxplot(data, "Total_Trans_Amt", kde=True)  ## Complete the code to create hist
ogram_boxplot for 'New_Price'
```

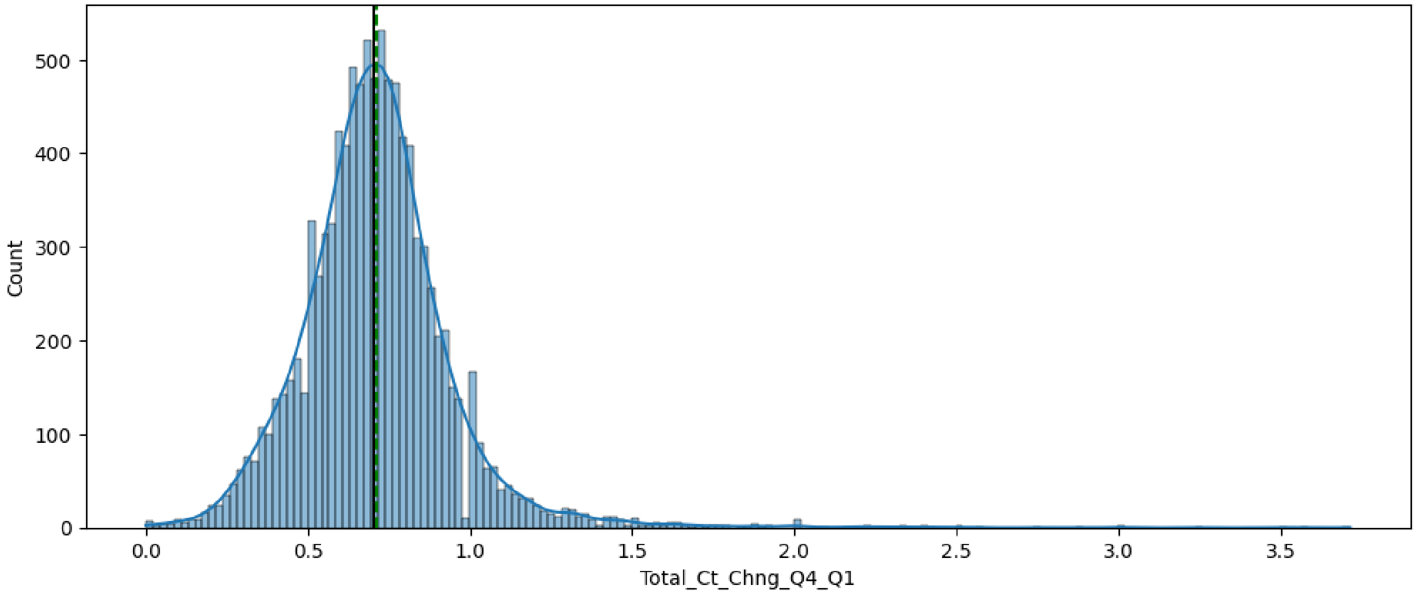


`Total_Ct_Chng_Q4_Q1`

In [302]:

```
histogram_boxplot(data, 'Total_Ct_Chng_Q4_Q1', kde=True)  ## Complete the code to create
histogram_boxplot for 'New_Price'
```

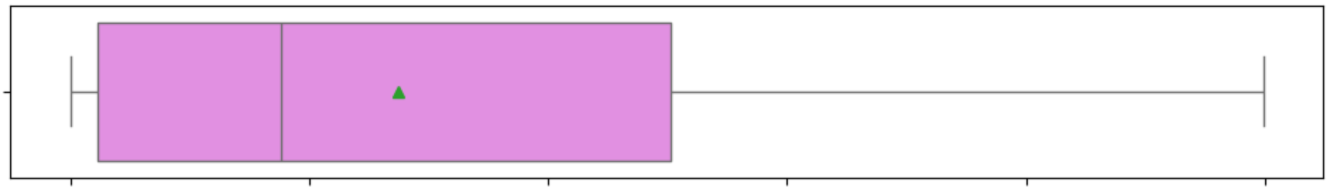


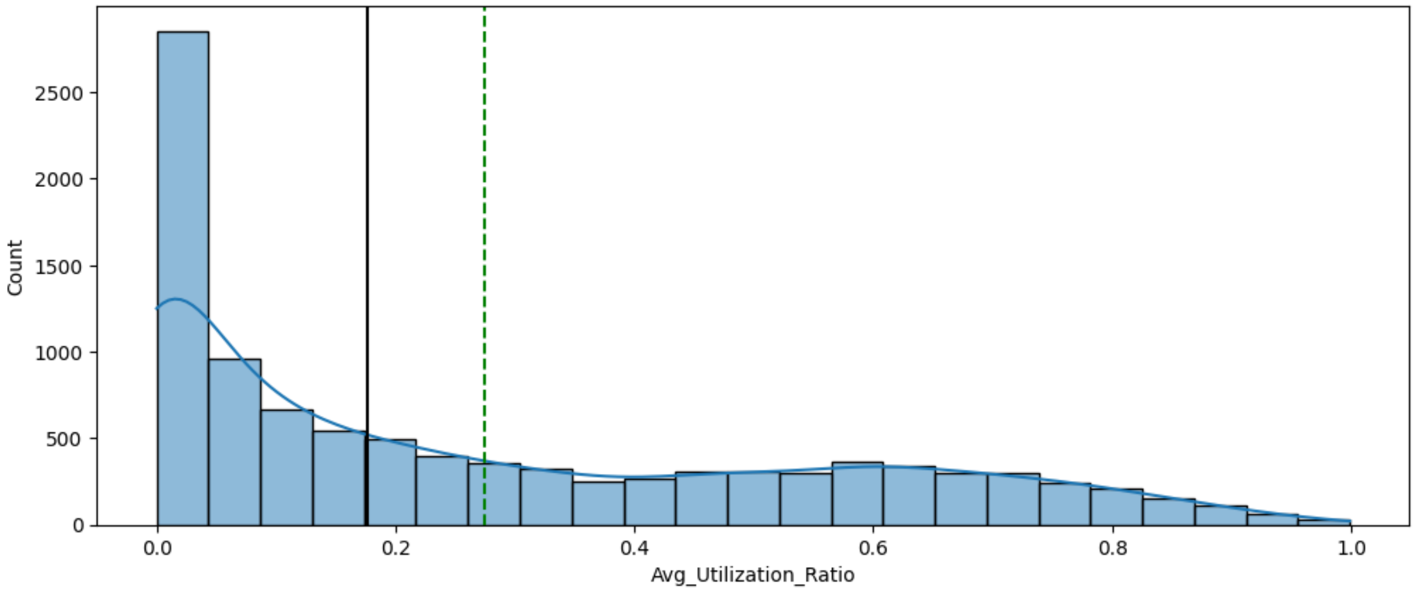


`Avg_Utilization_Ratio`

In [303]:

```
histogram_boxplot(data, 'Avg_Utilization_Ratio', kde=True)    ## Complete the code to crea
te histogram_boxplot for 'New_Price'
```

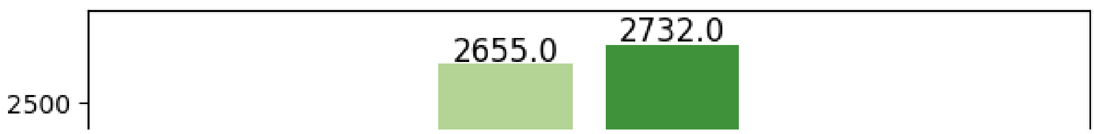


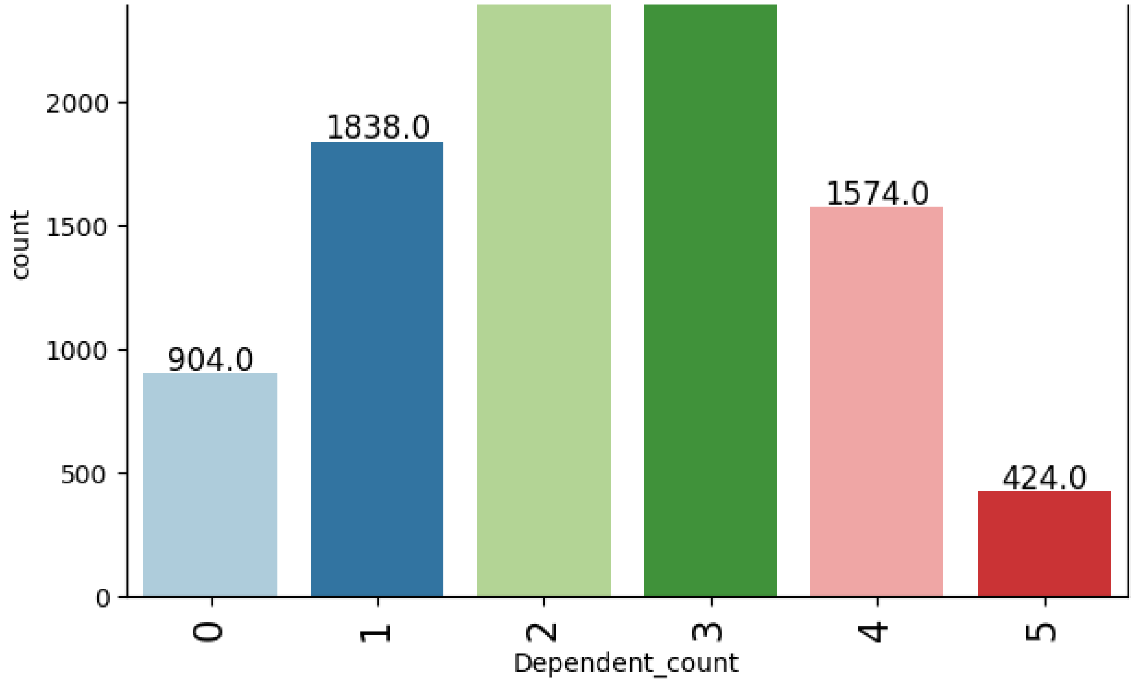


`Dependent_count`

In [304]:

```
labeled_barplot(data, "Dependent_count")
```

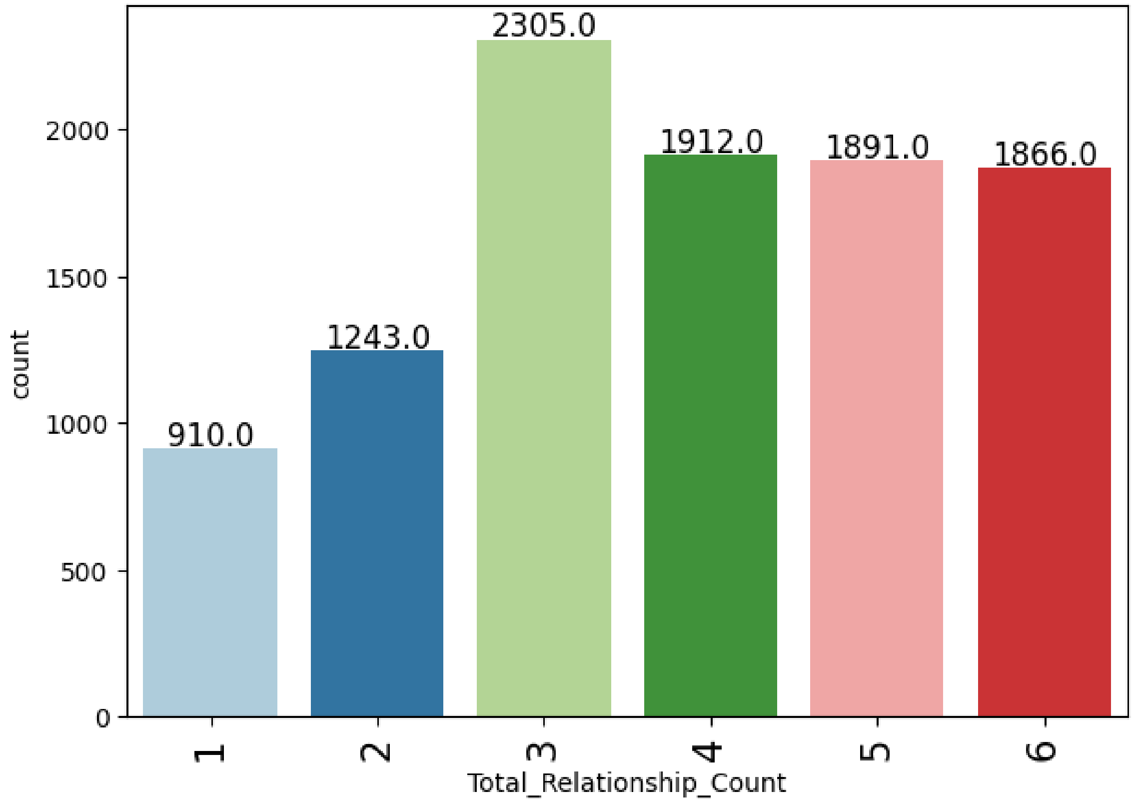




`Total_Relationship_Count`

In [305]:

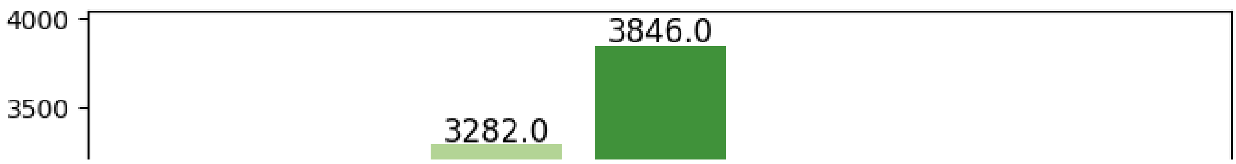
```
labeled_barplot(data,'Total_Relationship_Count') ## Complete the code to create labeled_barplot for 'Total_Relationship_Count'
```

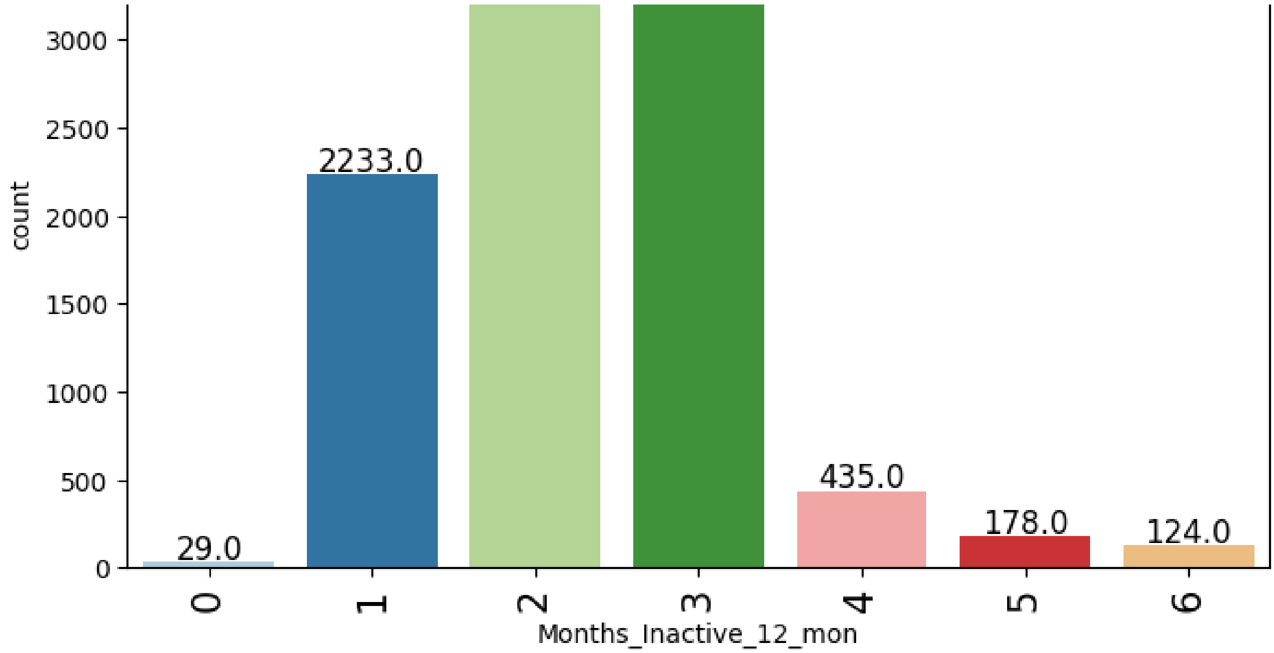


`Months_Inactive_12_mon`

In [306]:

```
labeled_barplot(data,'Months_Inactive_12_mon') ## Complete the code to create labeled_barplot for 'Months_Inactive_12_mon'
```

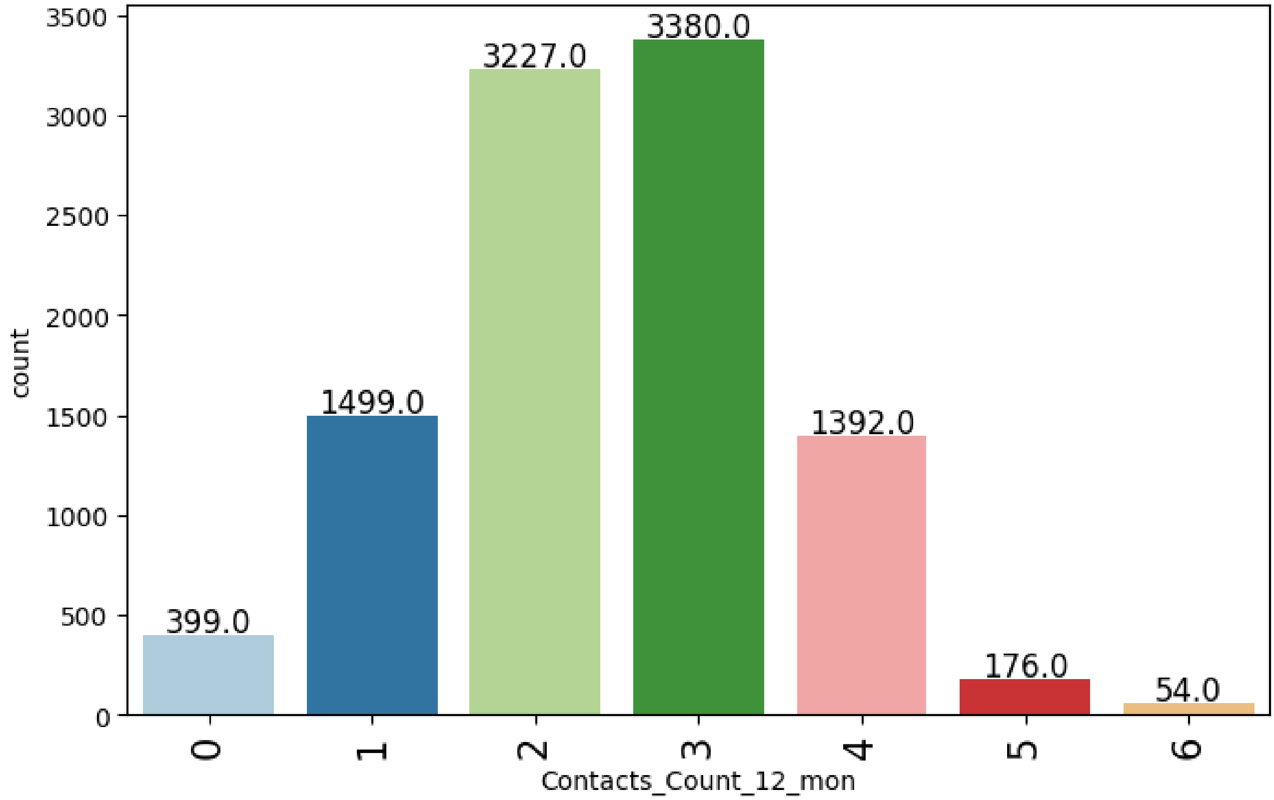




`Contacts_Count_12_mon`

In [307]:

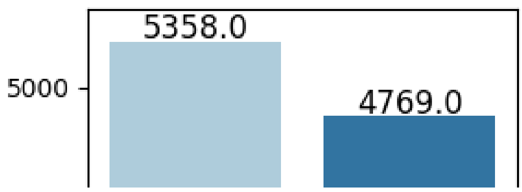
```
labeled_barplot(data,'Contacts_Count_12_mon') ## Complete the code to create labeled_barp
lot for 'Contacts_Count_12_mon'
```

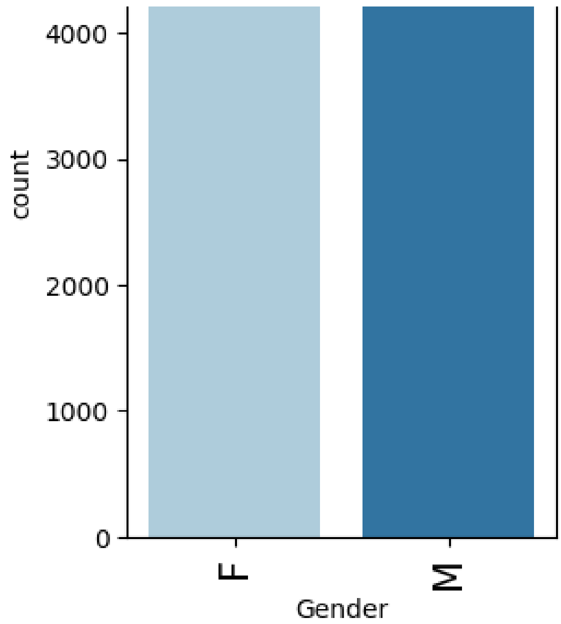


`Gender`

In [308]:

```
labeled_barplot(data,'Gender') ## Complete the code to create labeled_barplot for 'Gender
'
```



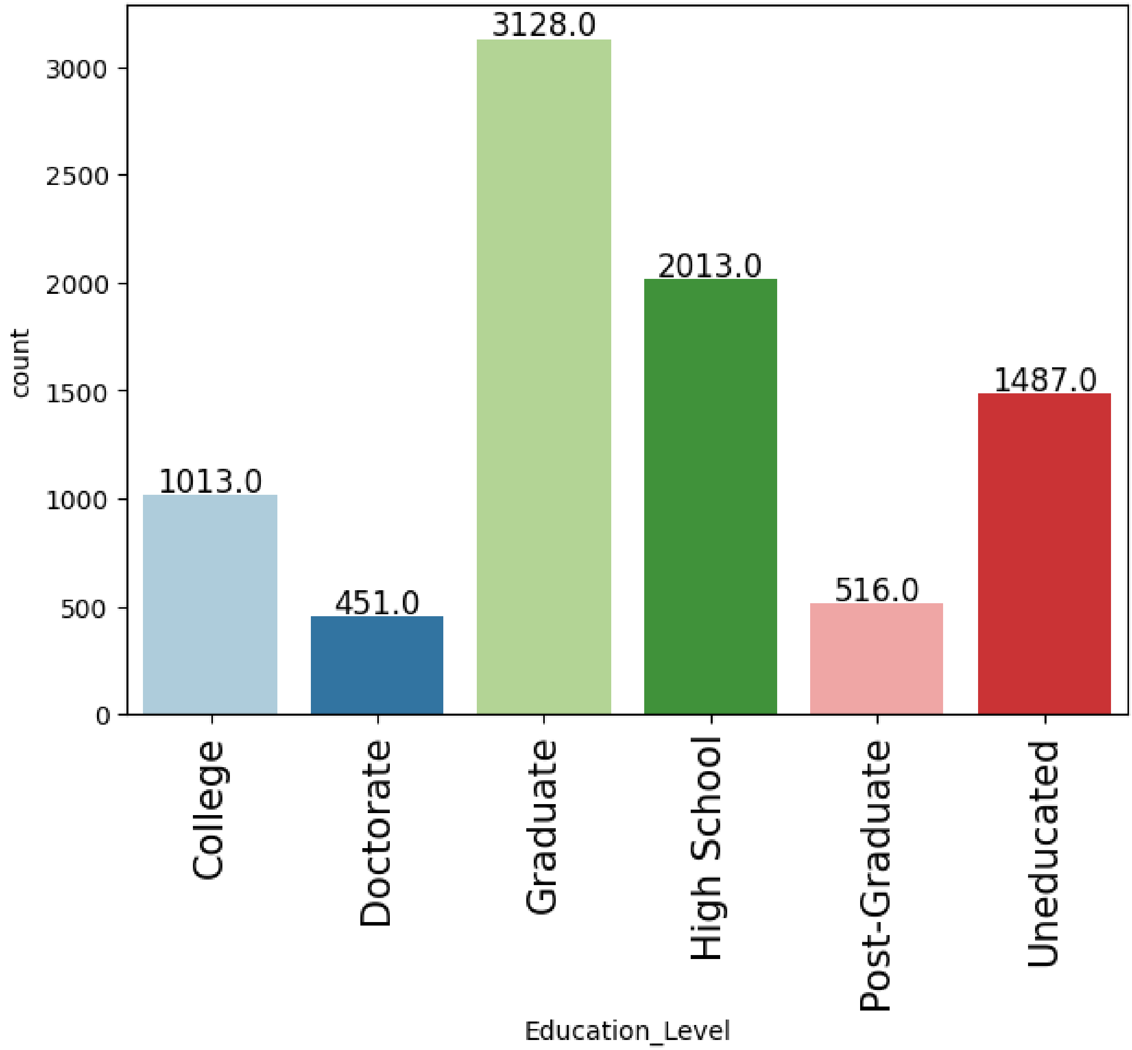


Let's see the distribution of the level of education of customers

`Education_Level`

In [309]:

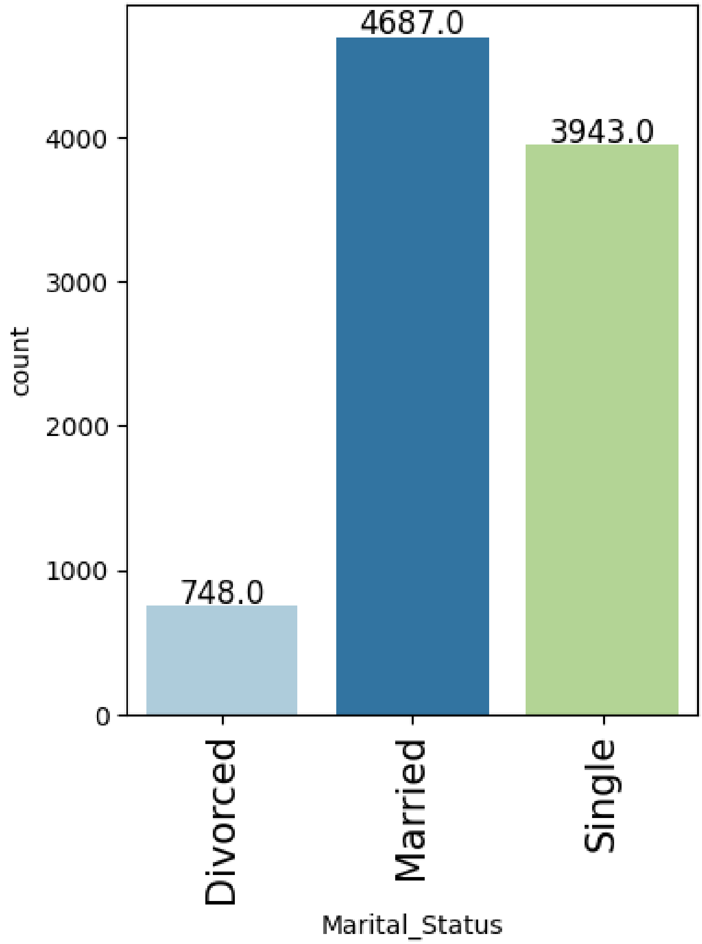
```
labeled_barplot(data,'Education_Level') ## Complete the code to create labeled_barplot fo
r 'Education_Level'
```



`Marital_Status`

In [310]:

```
labeled_barplot(data,'Marital_Status') ## Complete the code to create labeled_barplot for 'Marital_Status'
```

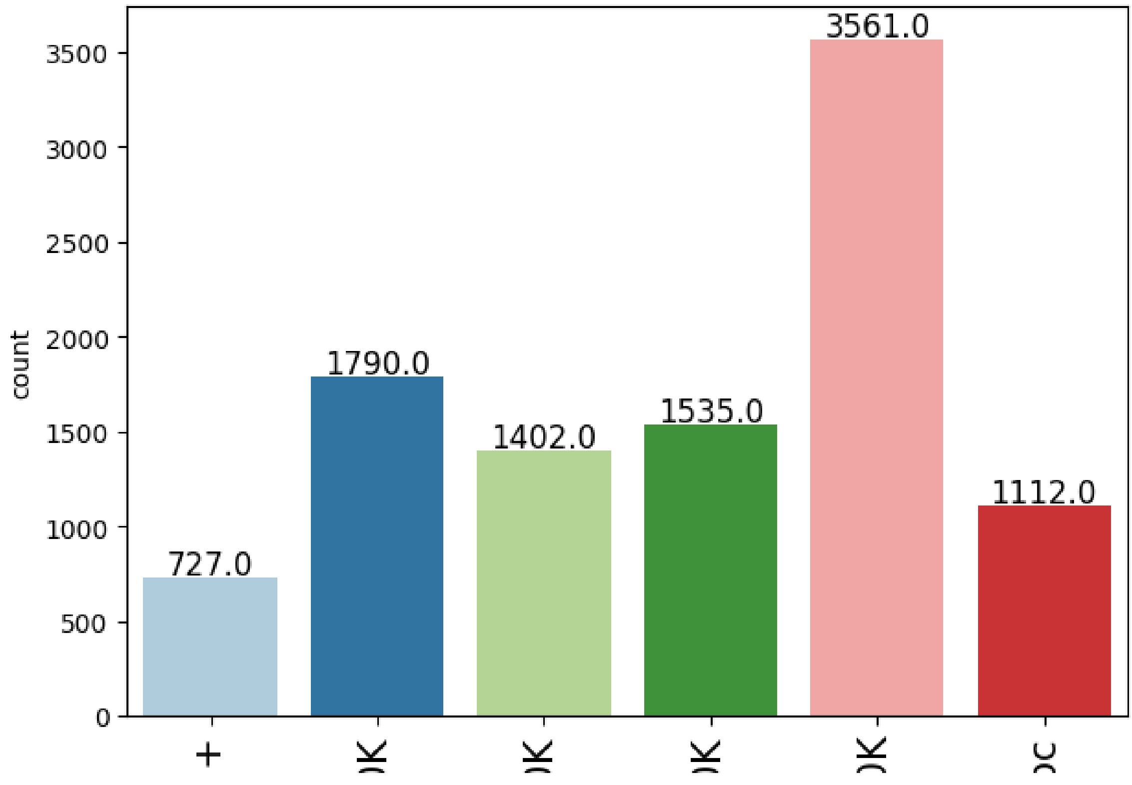


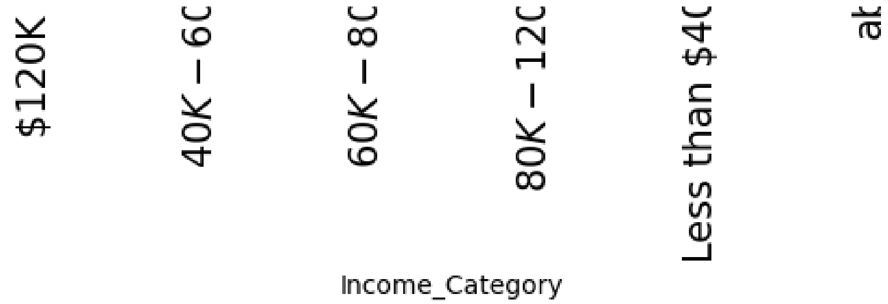
Let's see the distribution of the level of income of customers

`Income_Category`

In [311]:

```
labeled_barplot(data,'Income_Category') ## Complete the code to create labeled_barplot for 'Income_Category'
```

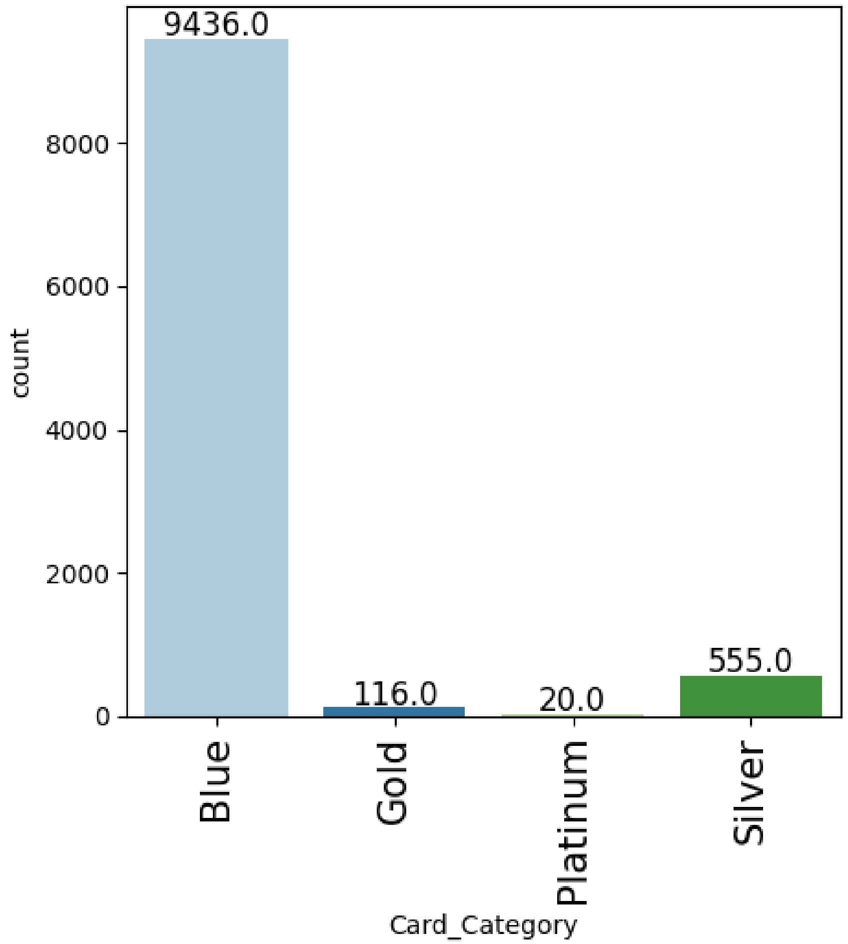




`Card_Category`

In [312]:

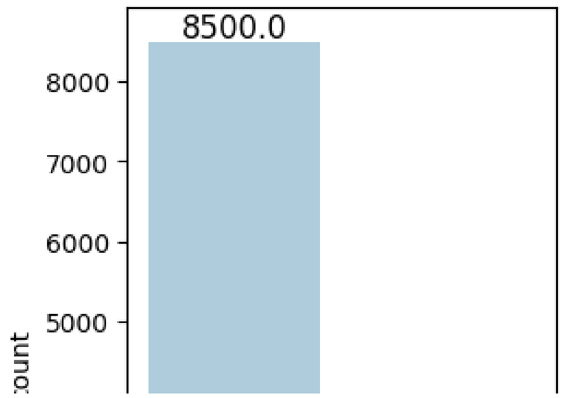
```
labeled_barplot(data,'Card_Category') ## Complete the code to create labeled_barplot for
'Card_Category'
```

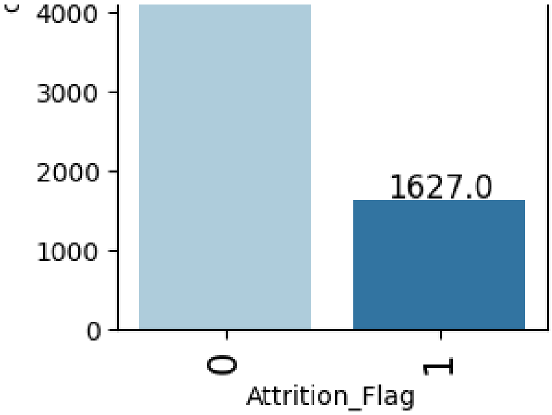


`Attrition_Flag`

In [313]:

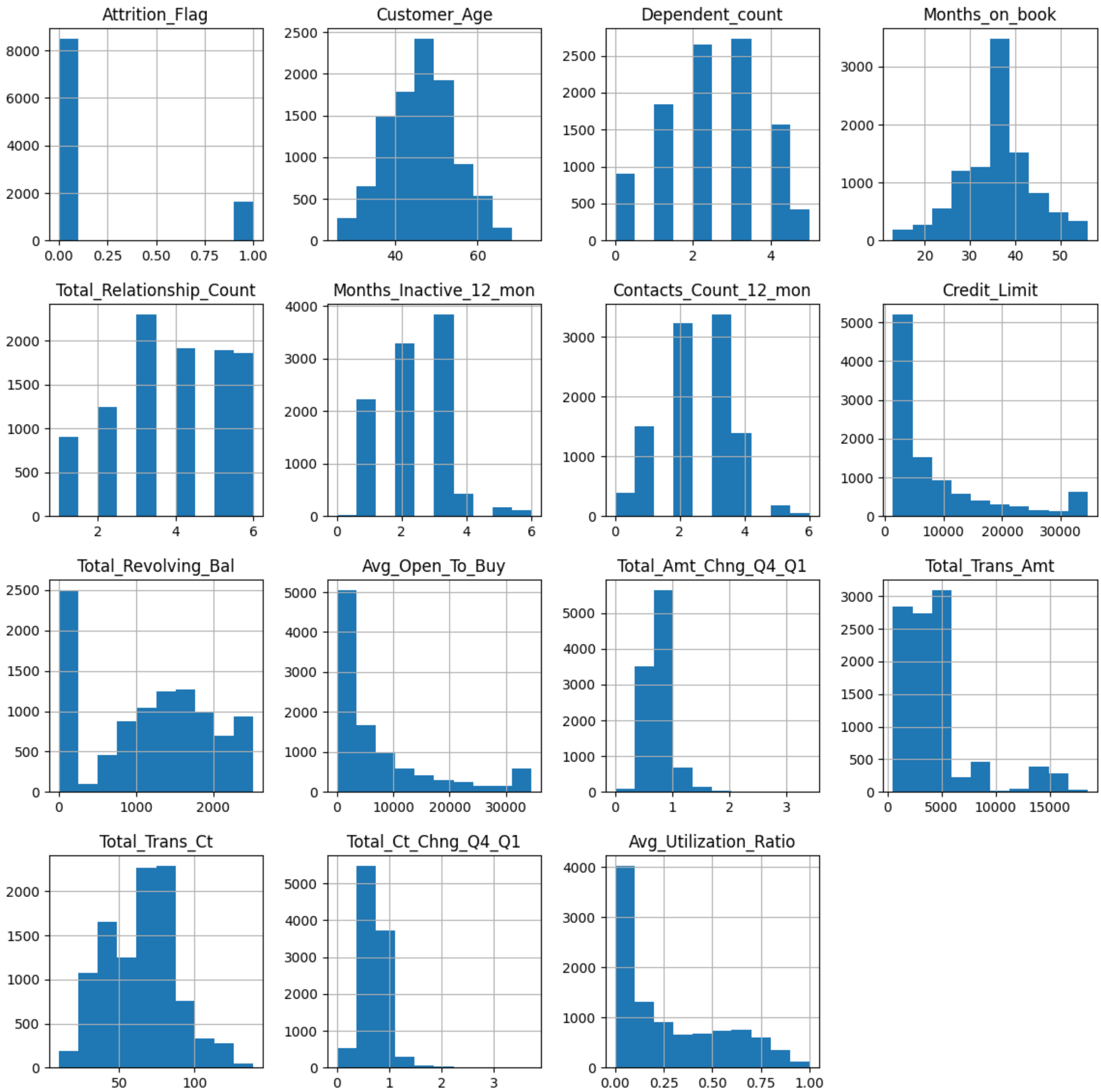
```
labeled_barplot(data,'Attrition_Flag') ## Complete the code to create labeled_barplot for
'Attrition_Flag'
```





In [314]:

```
# creating histograms
data.hist(figsize=(14, 14))
plt.show()
```



Bivariate Distributions

Let's see the attributes that have a strong correlation with each other

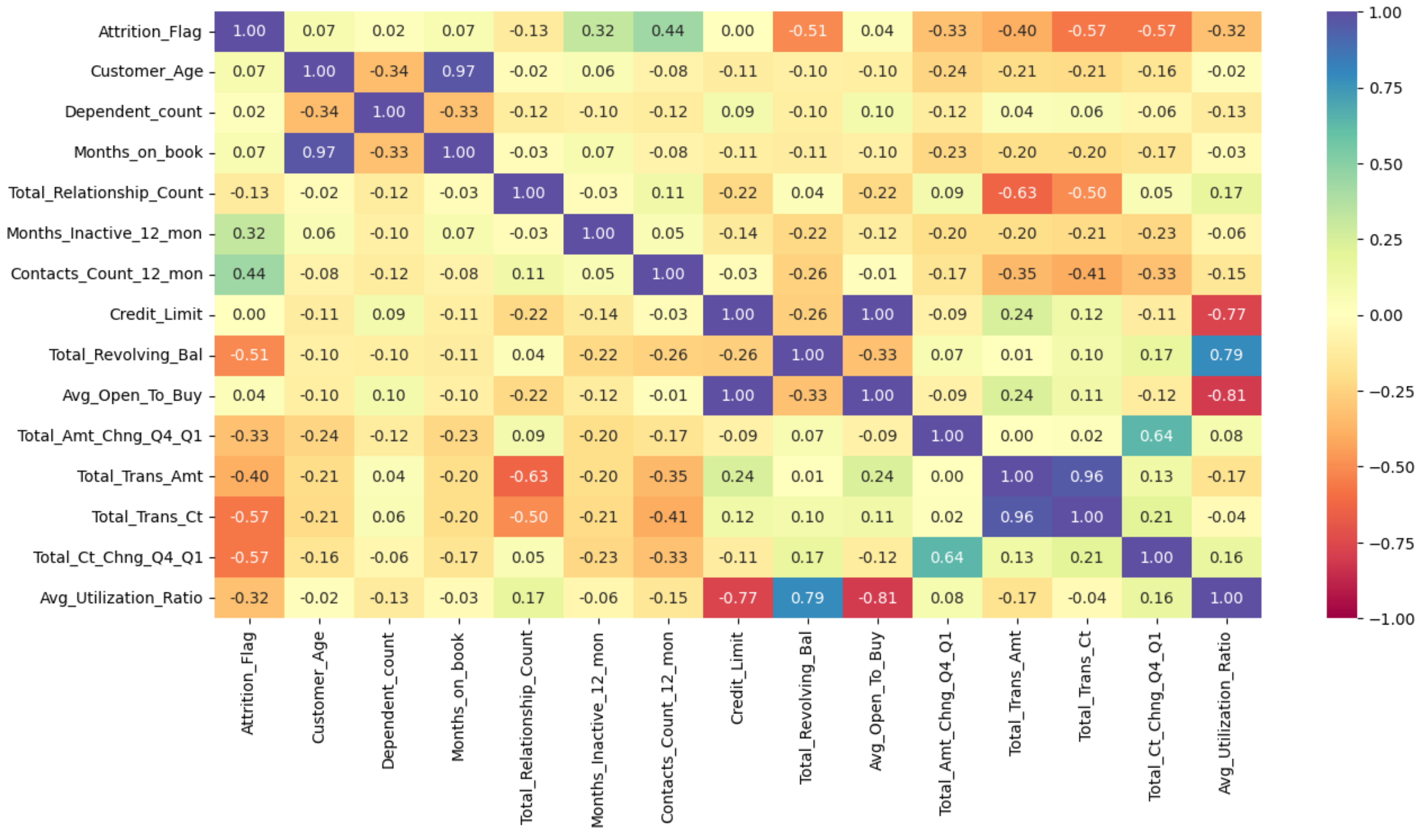
Correlation Check

In [315]:

```
# Select only numeric columns from the DataFrame
numeric_data = data.select_dtypes(include=['int64','float64'])

# Generate the correlation matrix for numeric columns only
correlation_matrix = numeric_data.corr()
```

In [316]:

```
plt.figure(figsize=(15, 7))
sns.heatmap(correlation_matrix.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spec
tral")
plt.show()
```

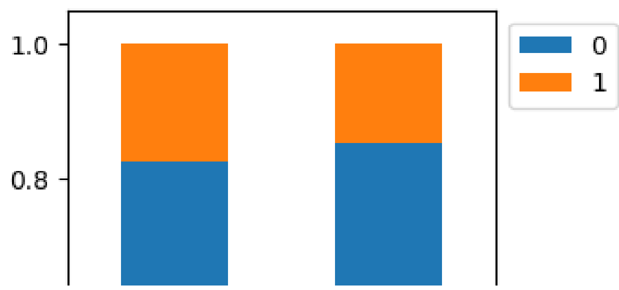


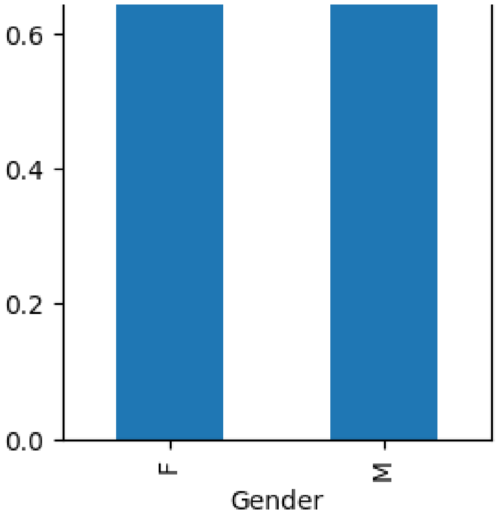
`Attrition_Flag vs Gender`

In [317]:

```
stacked_barplot(data, "Gender", "Attrition_Flag")
```

```
Attrition_Flag      0     1    All
Gender
All              8500  1627  10127
F                4428   930   5358
M                4072   697   4769
------------------------------------------------------------------------------------------------------------------------
```



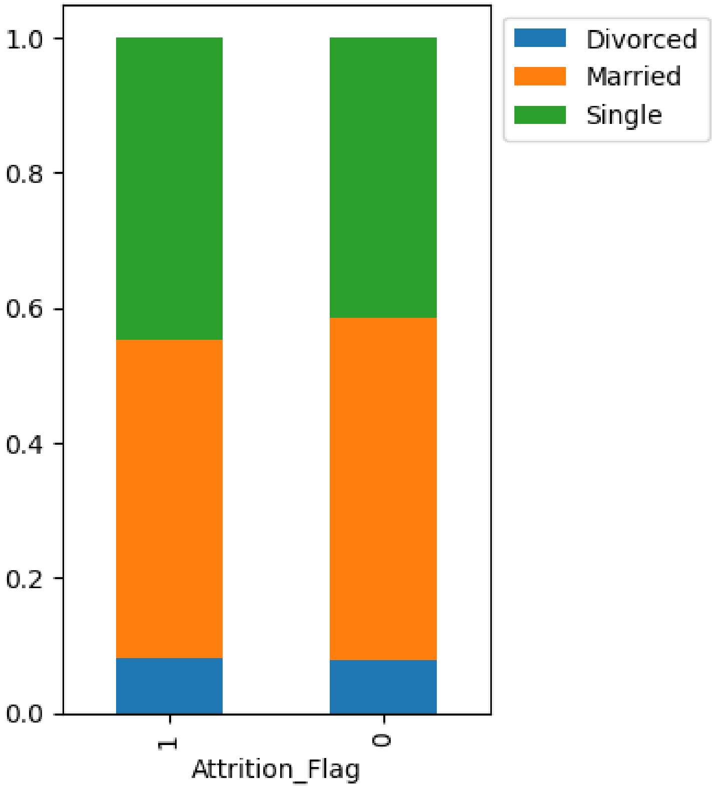


`Attrition_Flag vs Marital_Status`

In [318]:

```
stacked_barplot(data,"Attrition_Flag", "Marital_Status") ## Complete the code to create d
istribution_plot for Attrition_Flag vs Marital_Status
```

```
Marital_Status  Divorced  Married  Single   All
Attrition_Flag
All                  748     4687    3943  9378
0                    627     3978    3275  7880
1                    121      709     668  1498
------------------------------------------------------------------------------------------
----------------------------------
```



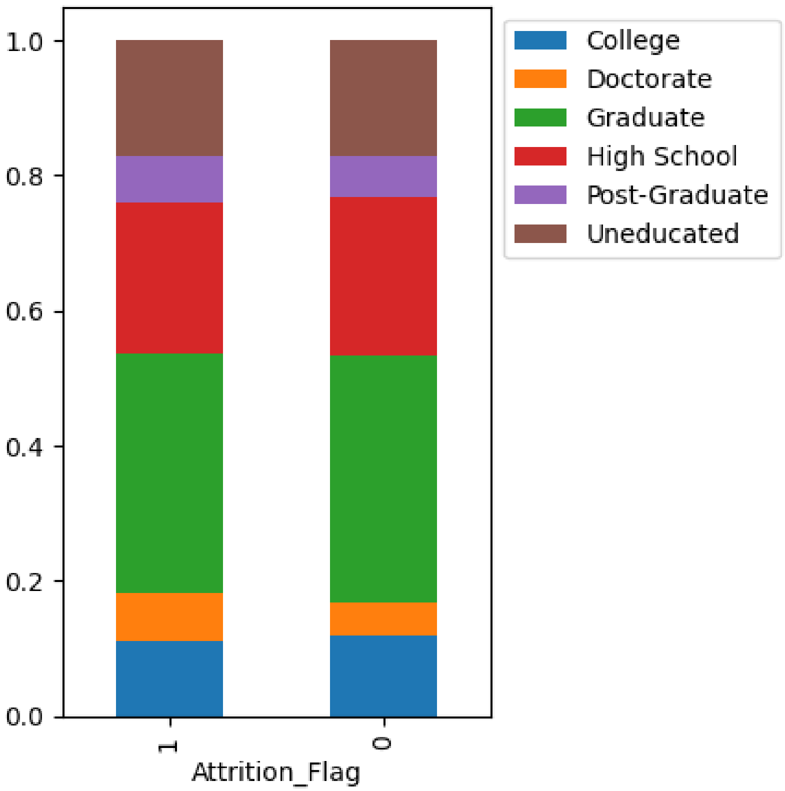
`Attrition_Flag vs Education_Level`

In [319]:

```
stacked_barplot(data,"Attrition_Flag", "Education_Level") ## Complete the code to create
distribution_plot for Attrition_Flag vs Education_Level
```

```
Education_Level  College  Doctorate  Graduate  High School  Post-Graduate  \
Attrition_Flag
All                 1013        451      3128         2013            516
```

```
0                        859         356       2641          1707              424
1                        154          95        487           306               92

Education_Level  Uneducated   All
Attrition_Flag
All                    1487  8608
0                      1250  7237
1                       237  1371
------------------------------------------------------------------------------------------
---------------------------------
```

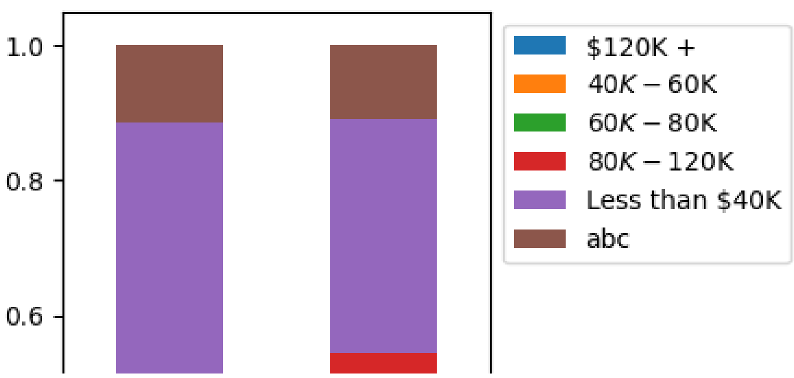


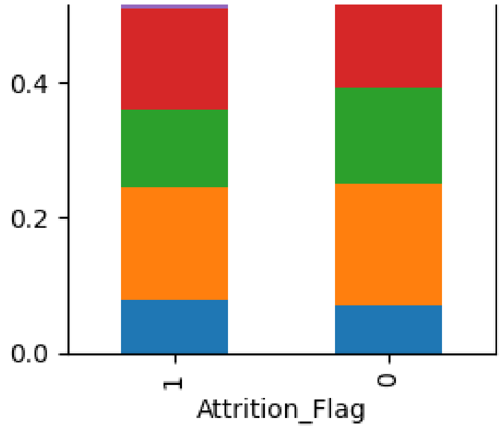
Attrition_Flag vs Income_Category

In [320]:

```
stacked_barplot(data,"Attrition_Flag", "Income_Category") ## Complete the code to create distribution_plot for Attrition_Flag vs Income_Category
```

```
Income_Category  $120K +  $40K - $60K  $60K - $80K  $80K - $120K  \
Attrition_Flag
All                  727         1790         1402          1535
0                    601         1519         1213          1293
1                    126          271          189           242

Income_Category  Less than $40K   abc    All
Attrition_Flag
All                        3561  1112  10127
0                          2949   925   8500
1                           612   187   1627
------------------------------------------------------------------------------------------
---------------------------------
```



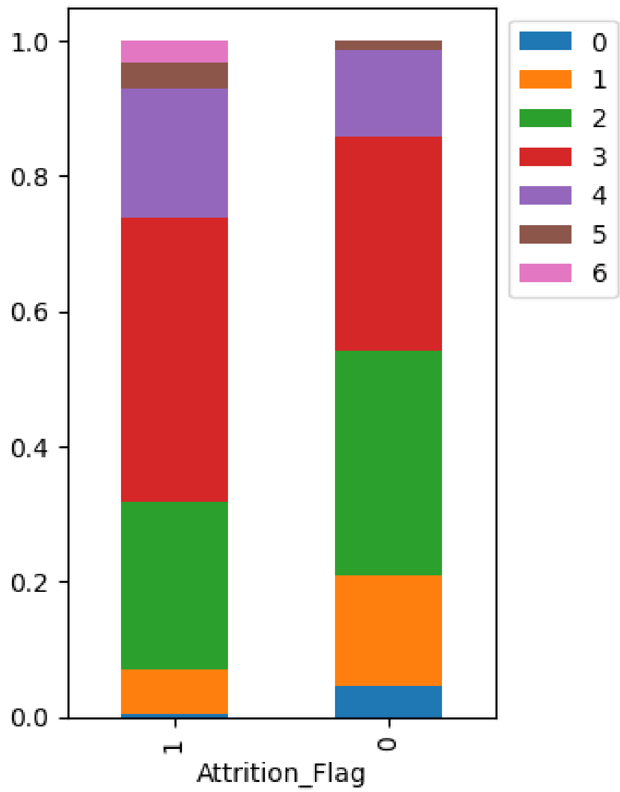


`Attrition_Flag vs Contacts_Count_12_mon`

In [321]:

```
stacked_barplot(data,"Attrition_Flag", "Contacts_Count_12_mon") ## Complete the code to c
reate distribution_plot for Attrition_Flag vs Income_Category
```

```
Contacts_Count_12_mon    0     1     2     3     4    5   6    All
Attrition_Flag
1                        7   108   403   681   315   59  54   1627
All                    399  1499  3227  3380  1392  176  54  10127
0                      392  1391  2824  2699  1077  117   0   8500
------------------------------------------------------------------------------------------
---------------------------------
```



Let's see the number of months a customer was inactive in the last 12 months (Months_Inactive_12_mon) vary by the customer's account status (Attrition_Flag)

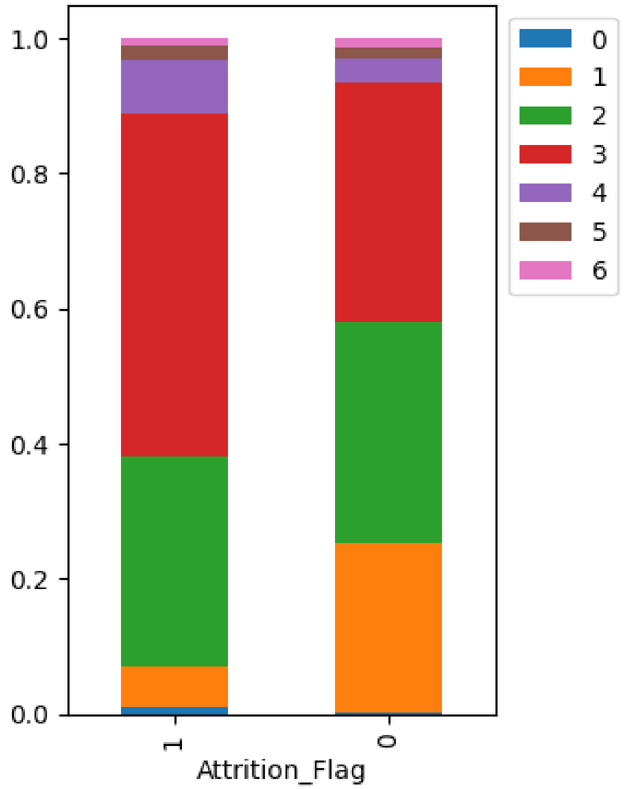
`Attrition_Flag vs Months_Inactive_12_mon`

In [322]:

```
stacked_barplot(data,"Attrition_Flag", "Months_Inactive_12_mon") ## Complete the code to
create distribution_plot for Attrition_Flag vs Months_Inactive_12_mon
```

```
Months_Inactive_12_mon    0     1     2     3    4    5    6    All
Attrition_Flag
```

```
All                     29  2233  3282  3846  435  178  124  10127
1                       15   100   505   826  130   32   19   1627
0                       14  2133  2777  3020  305  146  105   8500
------------------------------------------------------------------------------------------------------------------------
```

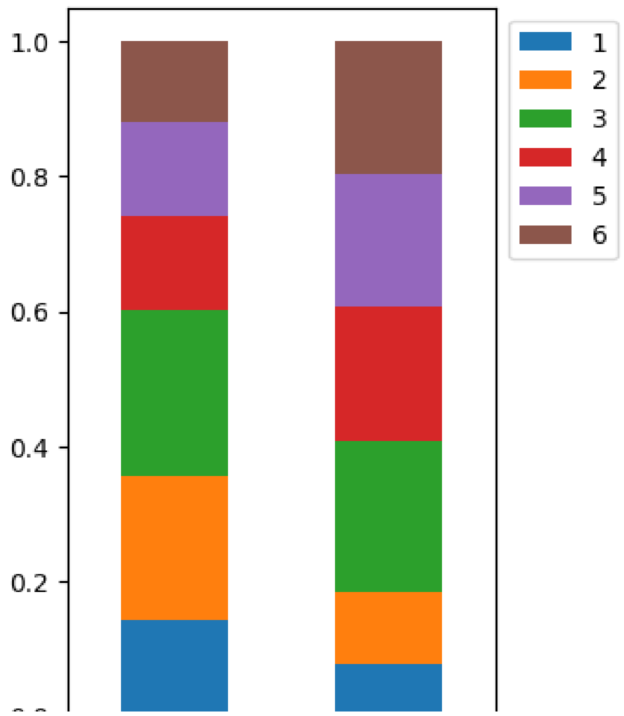



```
Attrition_Flag vs Total_Relationship_Count
```

In [323]:

```python
stacked_barplot(data,"Attrition_Flag", "Total_Relationship_Count") ## Complete the code to create distribution_plot for Attrition_Flag vs Total_Relationship_Count
```

```
Total_Relationship_Count    1     2     3     4     5     6    All
Attrition_Flag
All                       910  1243  2305  1912  1891  1866  10127
0                         677   897  1905  1687  1664  1670   8500
1                         233   346   400   225   227   196   1627
------------------------------------------------------------------------------------------------------------------------
```

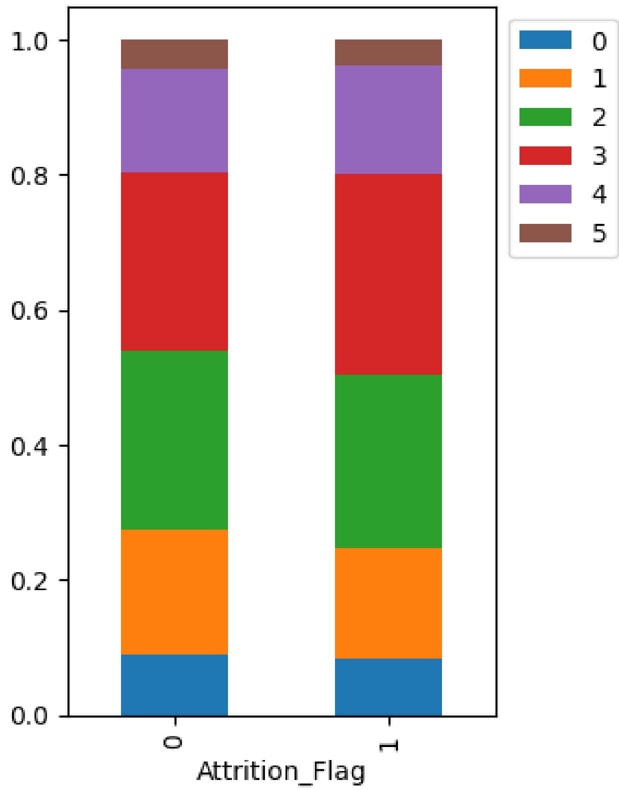


`Attrition_Flag vs Dependent_count`

In [324]:

```
stacked_barplot(data,"Attrition_Flag", "Dependent_count") ## Complete the code to create distribution_plot for Attrition_Flag vs Dependent_count
```

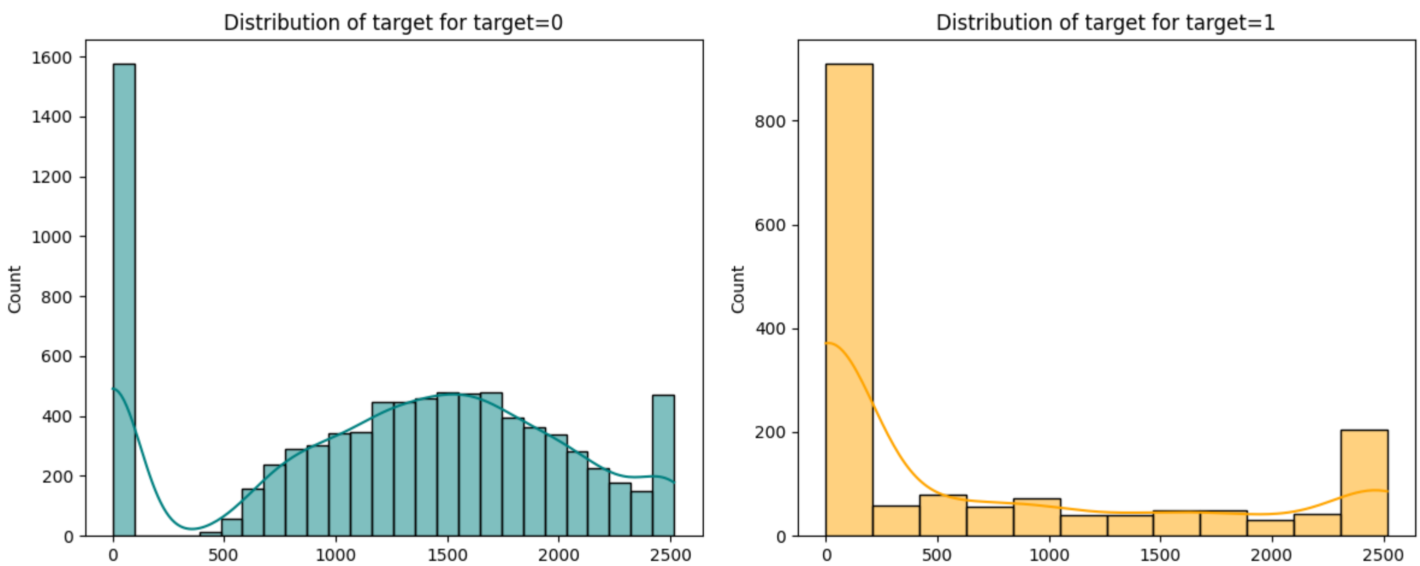
```
Dependent_count    0     1     2     3     4    5    All
Attrition_Flag
All              904  1838  2655  2732  1574  424  10127
0                769  1569  2238  2250  1314  360   8500
1                135   269   417   482   260   64   1627
------------------------------------------------------------------------------------------
----------------------------------
```

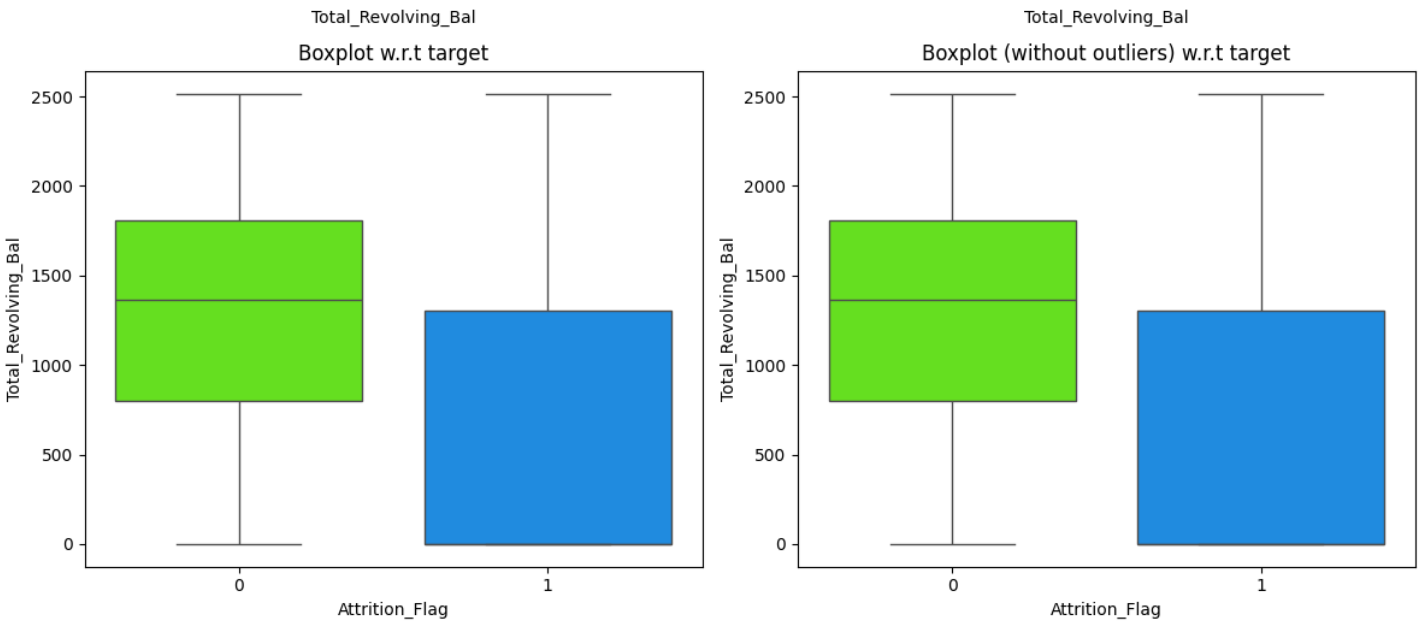


`Total_Revolving_Bal` **vs** `Attrition_Flag`

In [325]:

```
distribution_plot_wrt_target(data, "Total_Revolving_Bal", "Attrition_Flag")
```

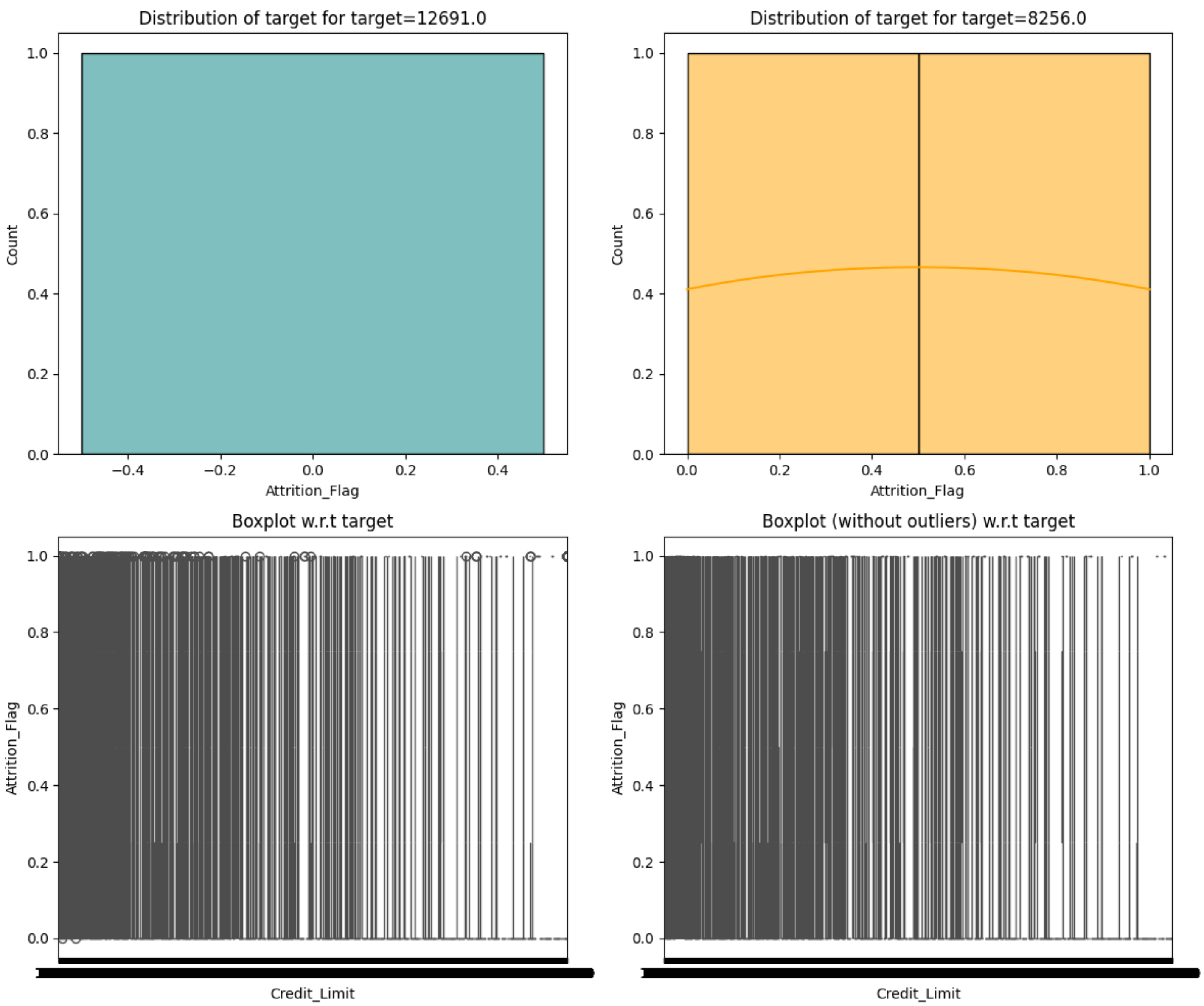




`Attrition_Flag vs Credit_Limit`

In [326]:

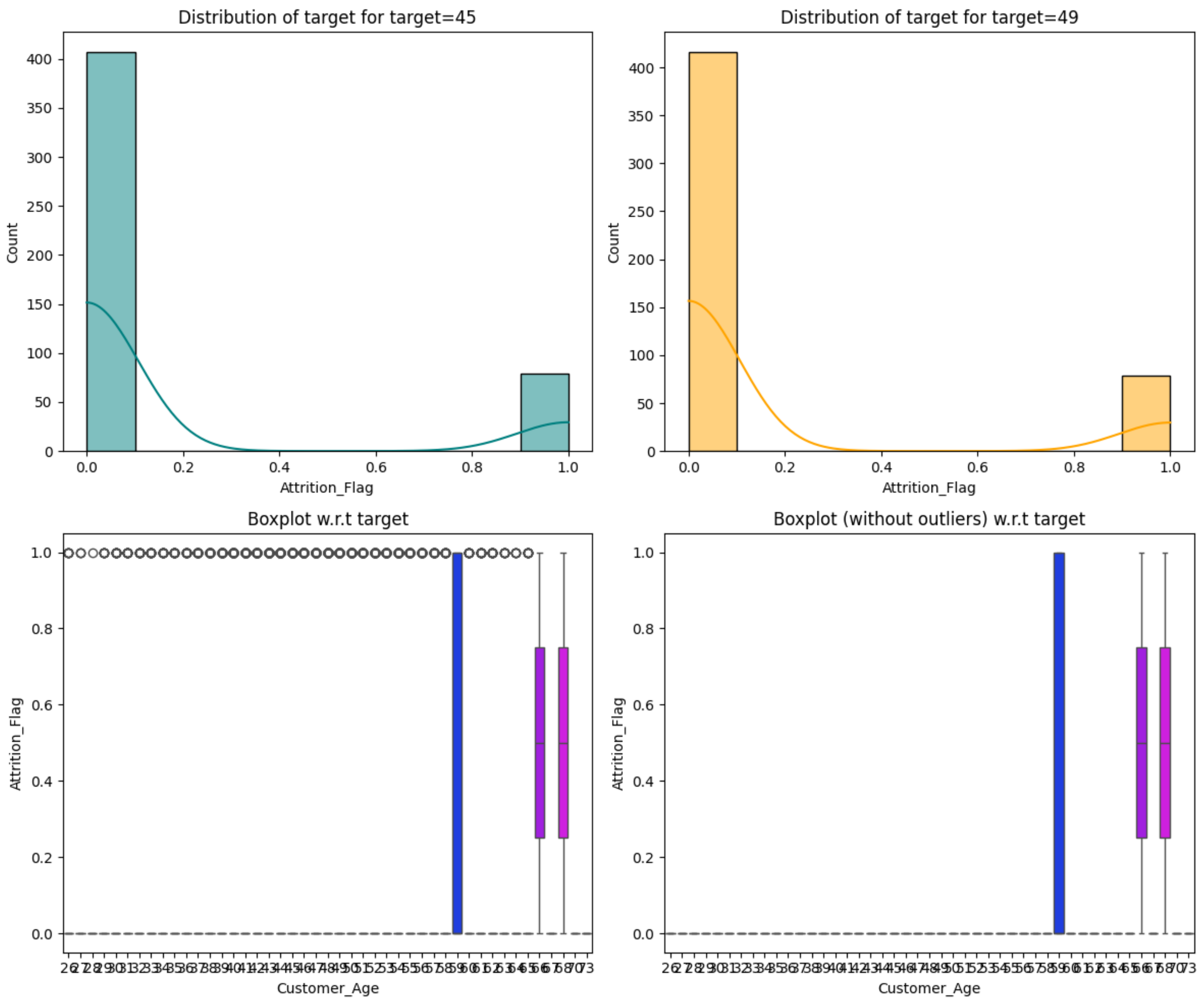
```
distribution_plot_wrt_target(data, "Attrition_Flag", "Credit_Limit") ## Complete the code to create distribution_plot for Attrition_Flag vs Credit_Limit
```



`Attrition_Flag vs Customer_Age`

In [327]:

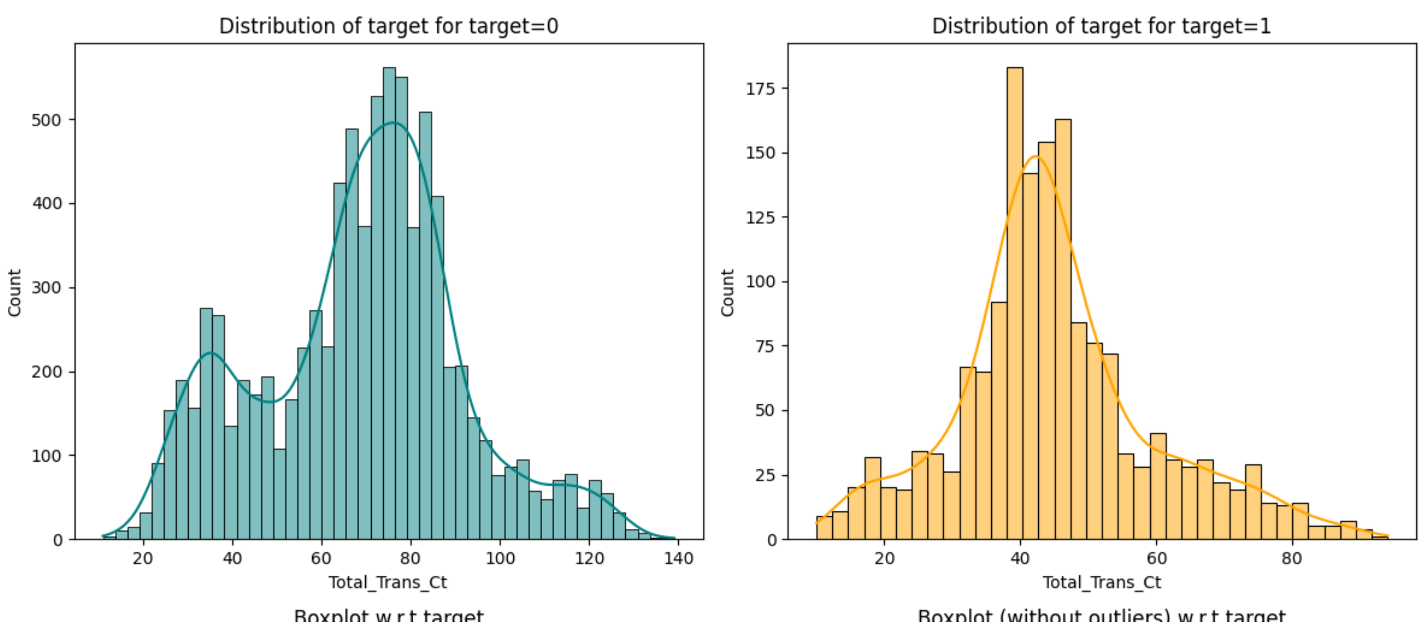
```
distribution_plot_wrt_target(data, "Attrition_Flag", "Customer_Age") ## Complete the code to create distribution_plot for Attrition_Flag vs Customer_Age
```

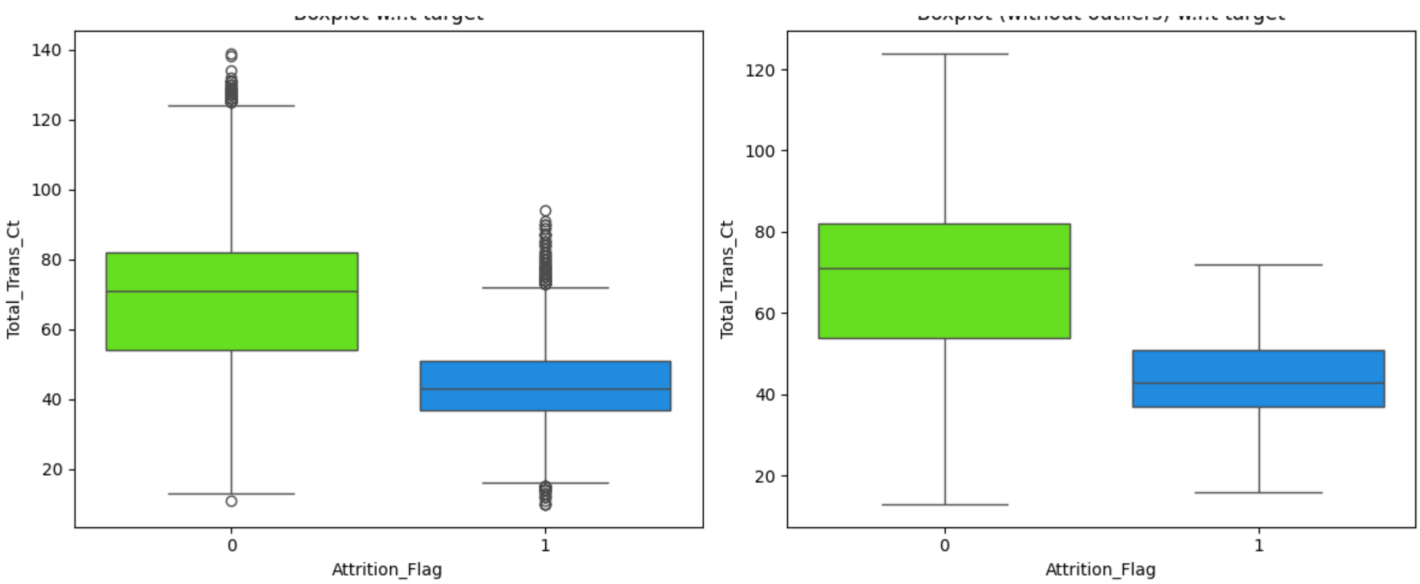


`Total_Trans_Ct` **vs** `Attrition_Flag`

In [328]:

```
distribution_plot_wrt_target(data, "Total_Trans_Ct", "Attrition_Flag") ## Complete the code to create distribution_plot for Total_Ct_Chng_Q4_Q1 vs Attrition_Flag
```

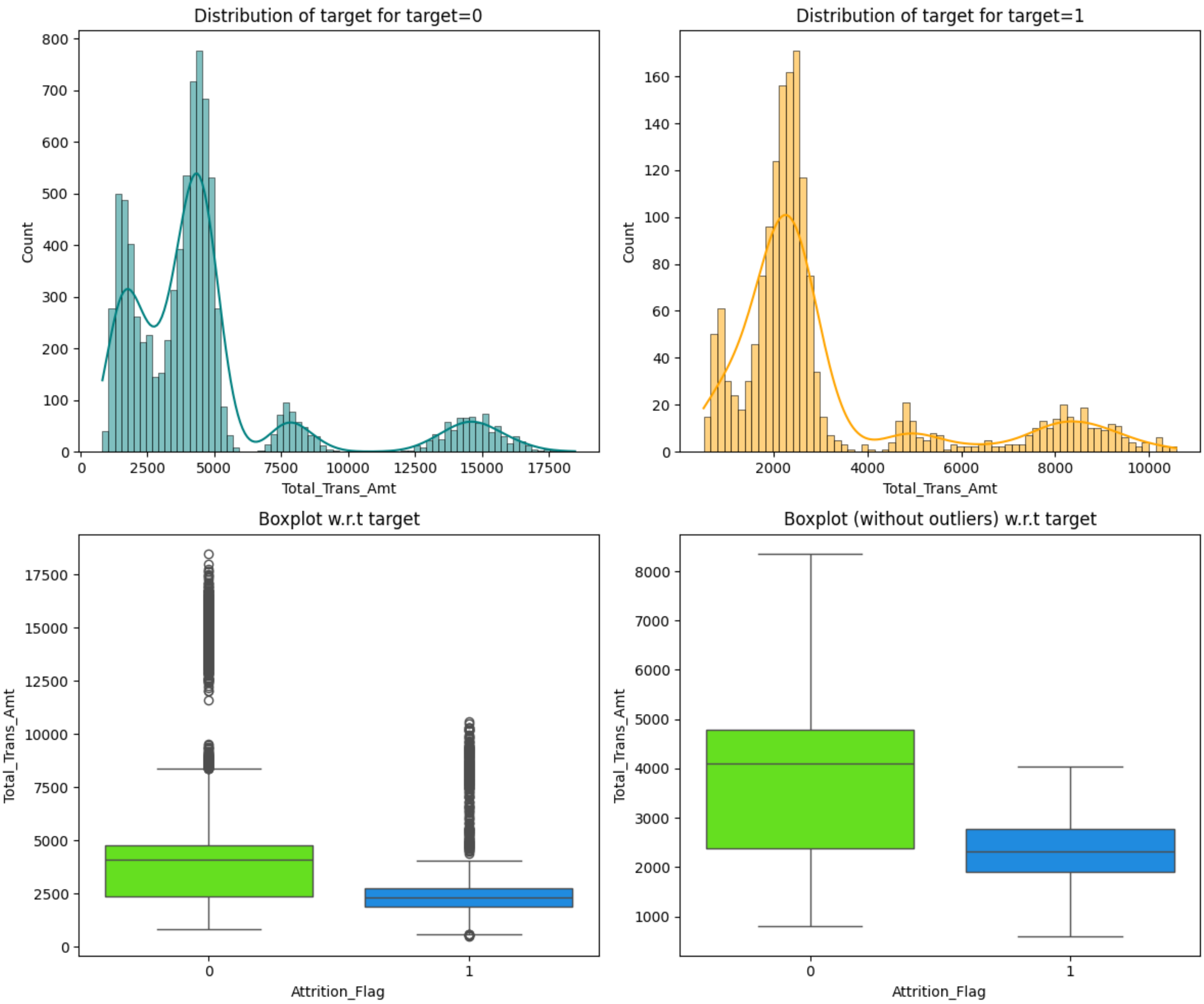




`Total_Trans_Amt` **vs** `Attrition_Flag`

In [329]:

```
distribution_plot_wrt_target(data, "Total_Trans_Amt", "Attrition_Flag") ## Complete the c
ode to create distribution_plot for Total_Trans_Amt vs Attrition_Flag
```



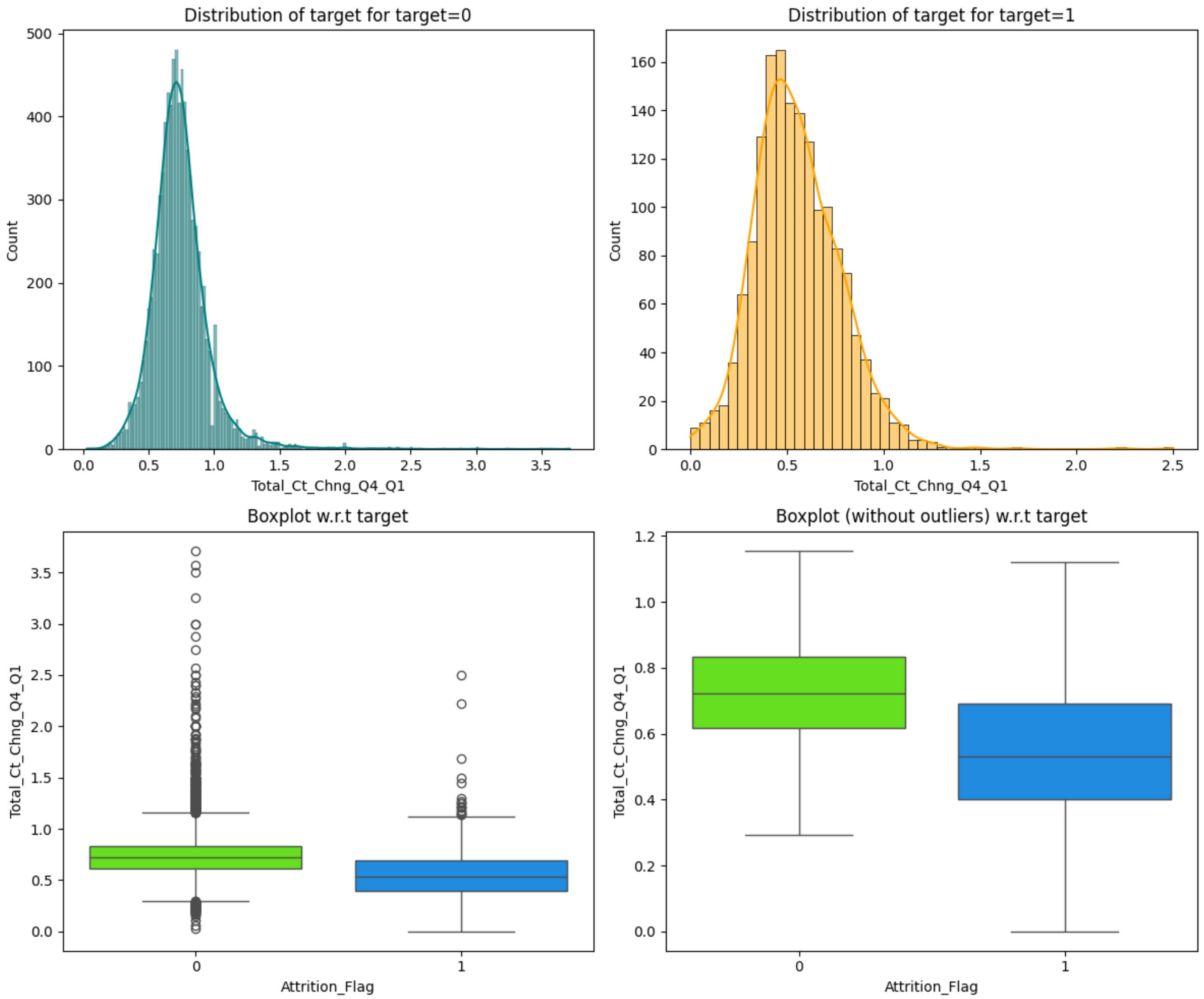
Let's see the change in transaction amount between Q4 and Q1 (total_ct_change_Q4_Q1) vary by the customer's account status (Attrition_Flag)

`Total_Ct_Chng_Q4_Q1` **vs** `Attrition_Flag`

`Total_Ct_Chng_Q4_Q1` **vs** `Attrition_Flag`

In [330]:

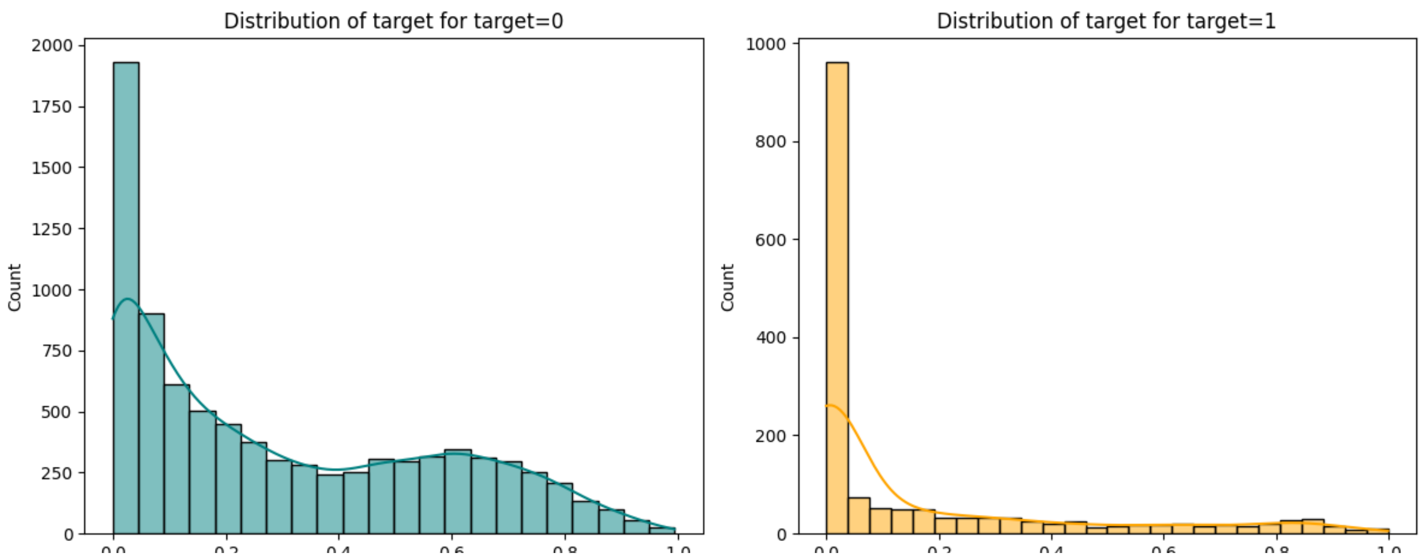
```
distribution_plot_wrt_target(data, "Total_Ct_Chng_Q4_Q1", "Attrition_Flag") ## Complete t
he code to create distribution_plot for Total_Ct_Chng_Q4_Q1 vs Attrition_Flag
```

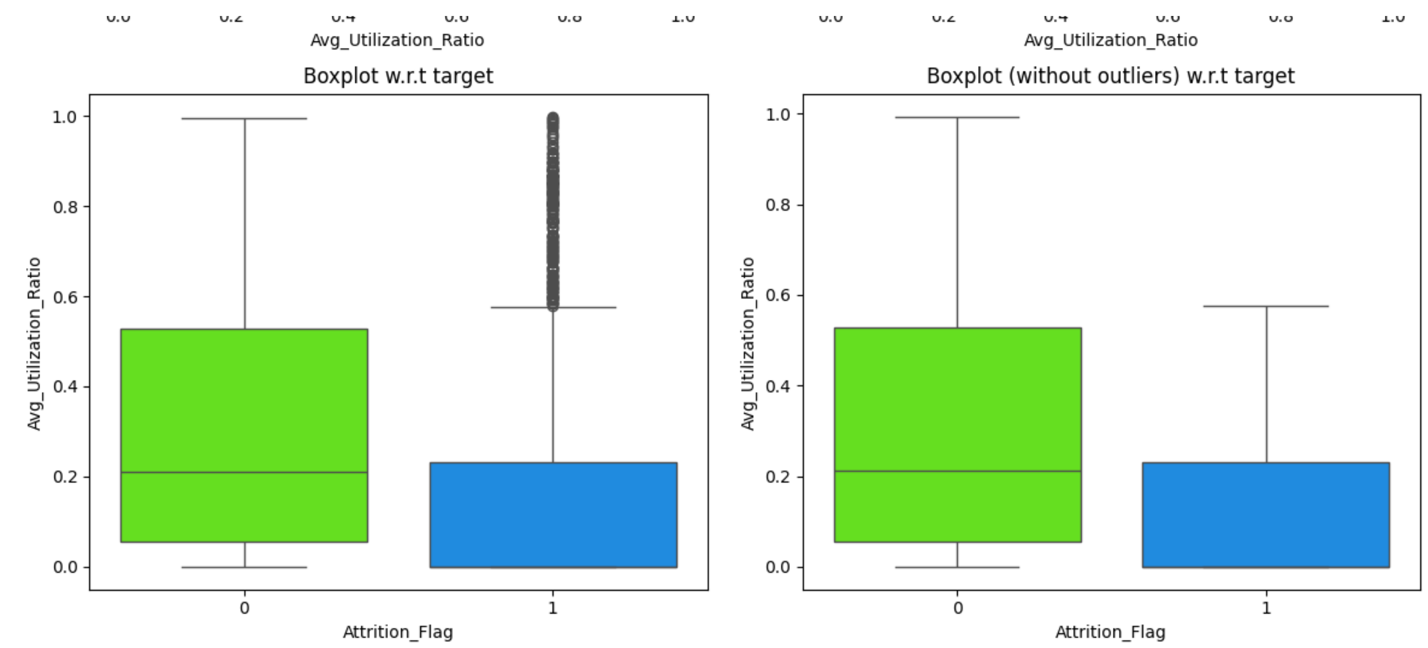


`Avg_Utilization_Ratio` **vs** `Attrition_Flag`

In [331]:

```
distribution_plot_wrt_target(data, "Avg_Utilization_Ratio", "Attrition_Flag") ## Complete
the code to create distribution_plot for Avg_Utilization_Ratio vs Attrition_Flag
```

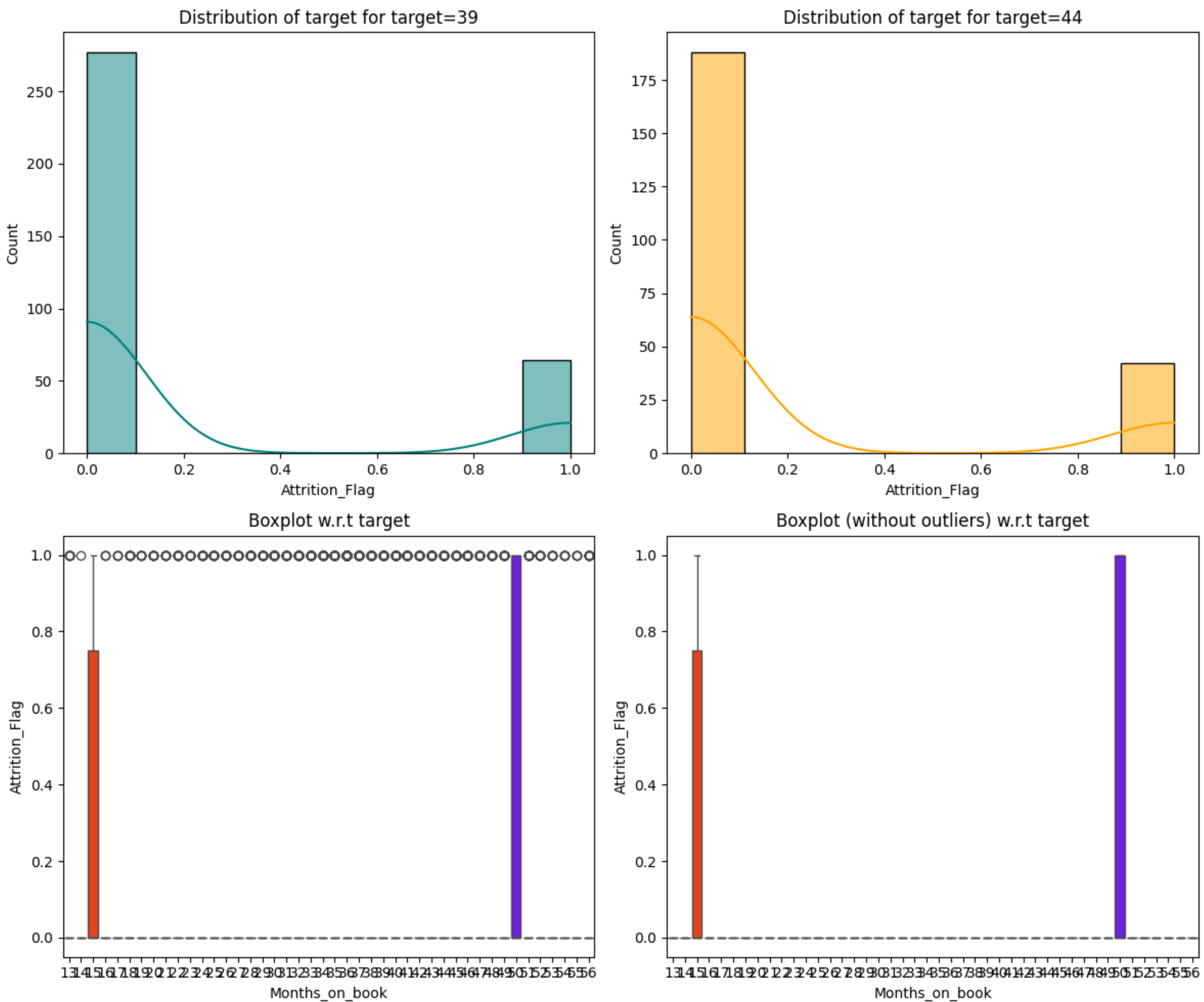




`Attrition_Flag vs Months_on_book`

In [332]:

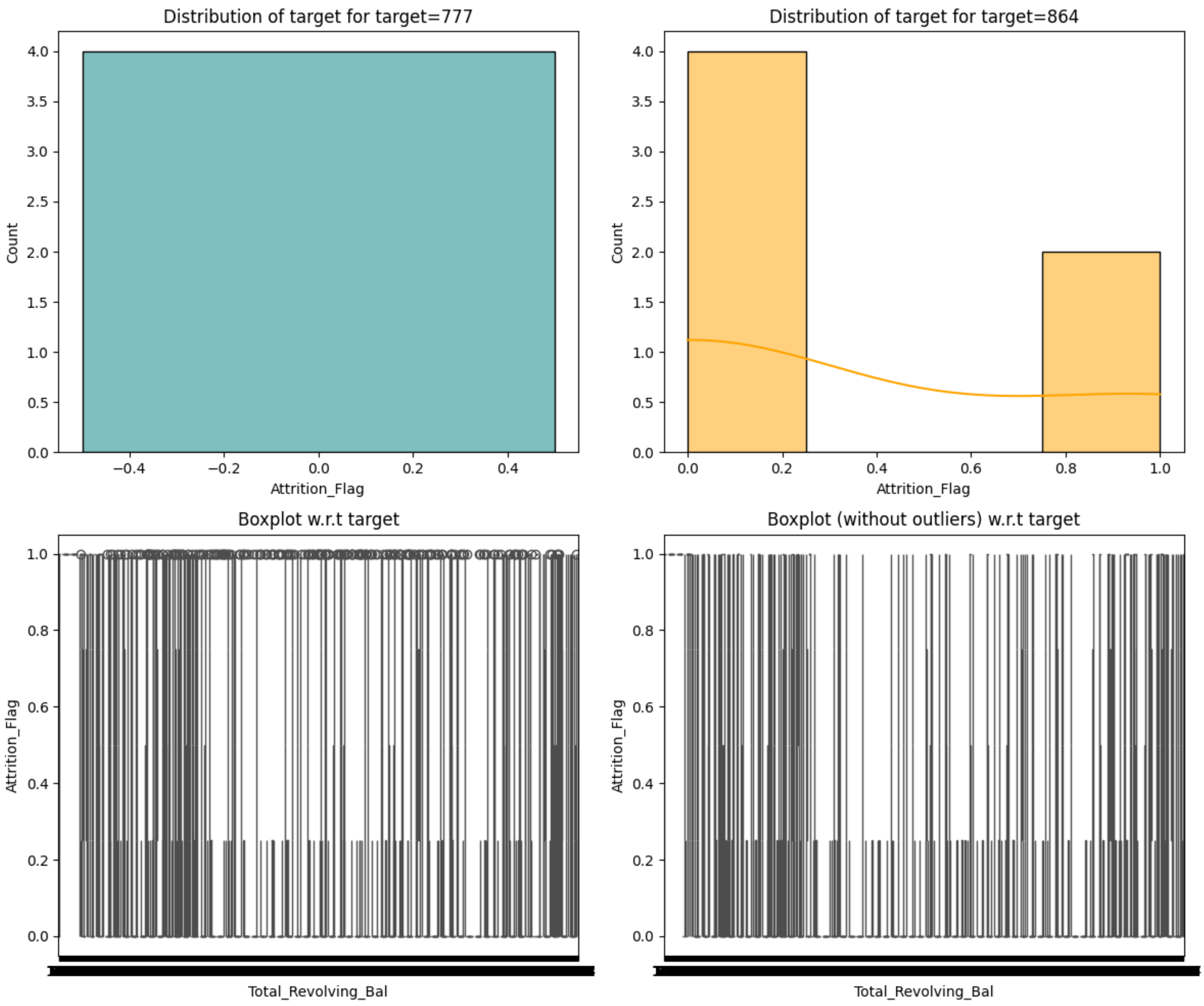
```
distribution_plot_wrt_target(data, "Attrition_Flag", "Months_on_book") ## Complete the co
de to create distribution_plot for Attrition_Flag vs Months_on_book
```



`Attrition_Flag vs Total_Revolving_Bal`

In [333]:

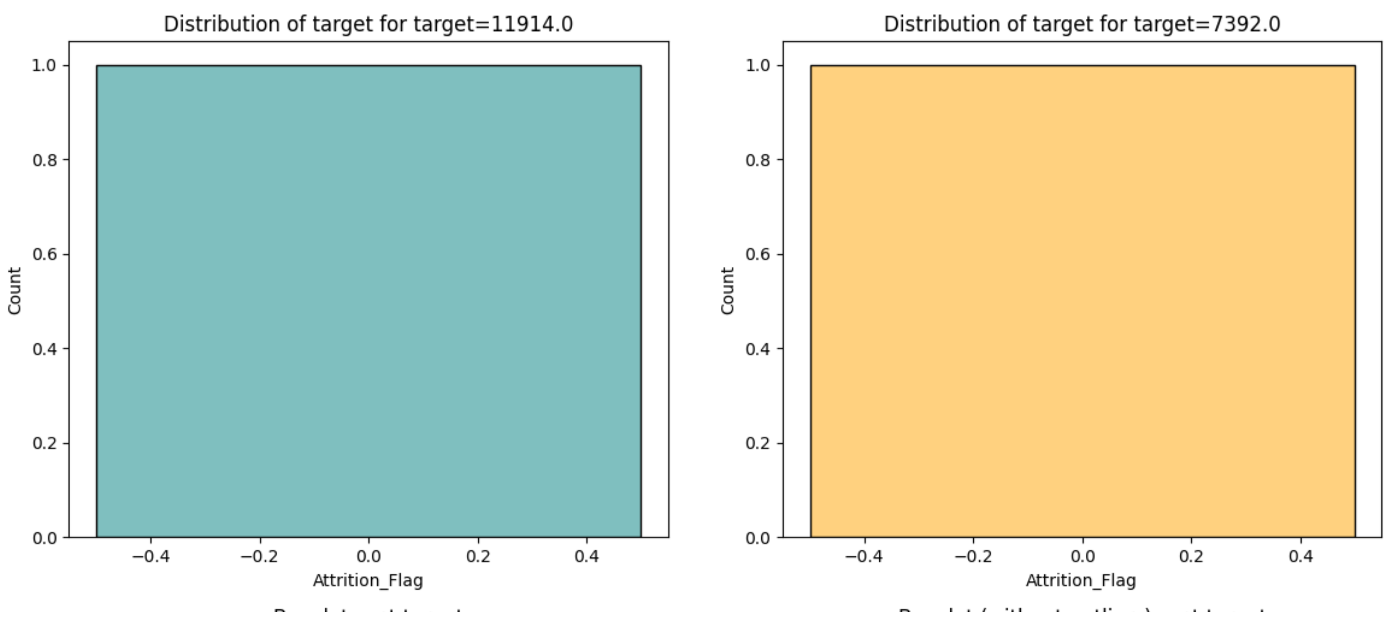
```
distribution_plot_wrt_target(data, "Attrition_Flag", "Total_Revolving_Bal") ## Complete t
he code to create distribution_plot for Attrition_Flag vs Total_Revolving_Bal
```

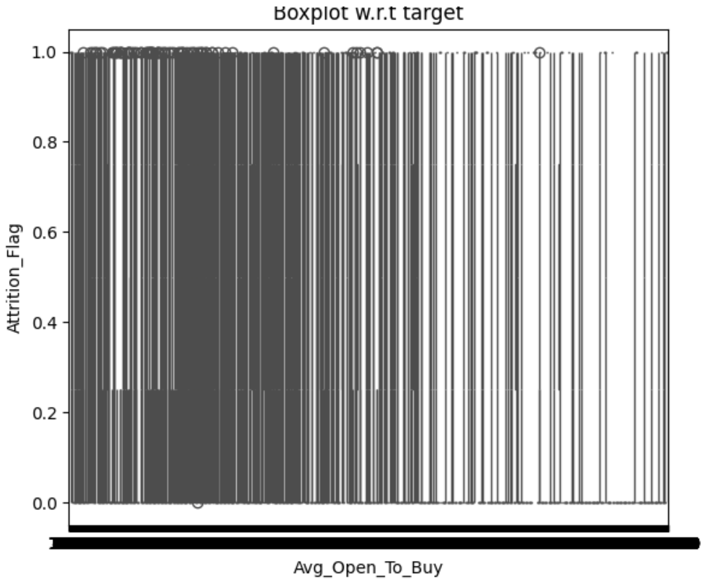


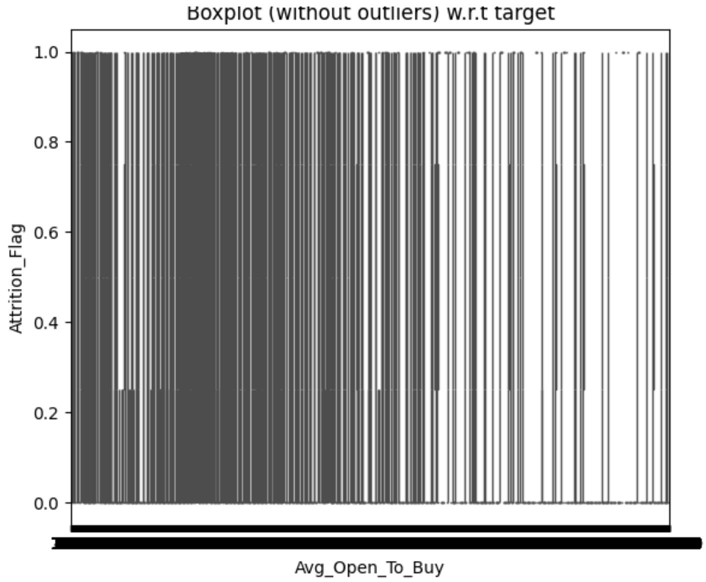
`Attrition_Flag vs Avg_Open_To_Buy`

In [334]:

```
distribution_plot_wrt_target(data, "Attrition_Flag", "Avg_Open_To_Buy") ## Complete the c
ode to create distribution_plot for Attrition_Flag vs Avg_Open_To_Buy
```







Data Preprocessing

Outlier Detection

In [335]:

```
Q1 = data.select_dtypes(include=["float64", "int64"]).quantile(0.25)  # To find the 25th
percentile
Q3 = data.select_dtypes(include=["float64", "int64"]).quantile(0.75)  # To find the 75th
percentile

IQR = Q3 - Q1  # Inter Quantile Range (75th perentile - 25th percentile)

# Finding lower and upper bounds for all values. All values outside these bounds are outl
iers
lower = (Q1 - 1.5 * IQR)
upper = (Q3 + 1.5 * IQR)
```

In [336]:

```
# checking the % outliers
((data.select_dtypes(include=["float64", "int64"]) < lower) | (data.select_dtypes(includ
e=["float64", "int64"]) > upper)).sum() / len(data) * 100
```

Out[336]:

	0
Attrition_Flag	16.066
Customer_Age	0.020
Dependent_count	0.000
Months_on_book	3.812
Total_Relationship_Count	0.000
Months_Inactive_12_mon	3.268
Contacts_Count_12_mon	6.211
Credit_Limit	9.717
Total_Revolving_Bal	0.000
Avg_Open_To_Buy	9.509
Total_Amt_Chng_Q4_Q1	3.910
Total_Trans_Amt	8.848
Total_Trans_Ct	0.020

	0
Total_Ct_Chng_Q4_Q1	3.891
Avg_Utilization_Ratio	0.000

dtype: float64

Train-Test Split

In [337]:

```
# creating the copy of the dataframe
data1 = data.copy()
```

In [338]:

```
data1["Income_Category"].replace("abc", np.nan, inplace=True) ### complete the code to re
place the anomalous values with NaN
```

In [339]:

```
data1.isna().sum()
```

Out[339]:

	0
Attrition_Flag	0
Customer_Age	0
Gender	0
Dependent_count	0
Education_Level	1519
Marital_Status	749
Income_Category	1112
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 [340]:

```
# creating an instace of the imputer to be used
imputer = SimpleImputer(strategy="most_frequent")
```

In [341]:

```
# Dividing train data into X and y
```

```
X = data1.drop(["Attrition_Flag"], axis=1)
y = data1["Attrition_Flag"]
```

In [342]:

```
# Splitting data into training and validation set:

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42
) ## Complete the code to split the data into train test in the ratio 80:20

X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.25, random_s
tate=42) ## Complete the code to split the data into train test in the ratio 75:25

print(X_train.shape, X_val.shape, X_test.shape)
```

```
(8101, 19) (507, 19) (1519, 19)
```

Missing value imputation

In [343]:

```
reqd_col_for_impute = ["Education_Level", "Marital_Status", "Income_Category"]
```

In [344]:

```
# Fit and transform the train data
X_train[reqd_col_for_impute] = imputer.fit_transform(X_train[reqd_col_for_impute])

# Transform the validation data
X_val[reqd_col_for_impute]  = imputer.transform(X_val[reqd_col_for_impute]) ## Complete
the code to impute missing values in X_val

# Transform the test data
X_test[reqd_col_for_impute] = imputer.transform(X_test[reqd_col_for_impute])## Complete
the code to impute missing values in X_test
```

In [345]:

```
data1.sample(5)
```

Out[345]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Categ
4451	1	50	F	1	High School	Single	Less than $40K	E
2277	0	34	M	3	Uneducated	Divorced	$120K +	E
911	0	46	M	3	High School	Married	60K−80K	E
5003	1	49	M	4	NaN	Single	$120K +	E
4280	0	54	M	1	Uneducated	Single	$120K +	E

In [346]:

```
X_train.head(10)
```

Out[346]:

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_o
9066	54	F	1	Graduate	Single	Less than $40K	Blue	
5814	58	F	4	High School	Married	Less than $40K	Blue	
792	45	F	4	Graduate	Single	Less than $40K	Gold	
1791	34	F	2	Graduate	Single	Less than $40K	Blue	

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_o
5011	49	F	2	High School	Married	40K−60K	Blue	
2260	60	F	0	Doctorate	Married	Less than $40K	Blue	
8794	43	F	4	Graduate	Single	Less than $40K	Blue	
4292	52	F	2	Graduate	Single	40K−60K	Blue	
1817	30	M	0	Graduate	Married	Less than $40K	Blue	
6025	33	F	3	Graduate	Single	Less than $40K	Blue	

In [347]:

```
data1.isna().sum()
```

Out[347]:

	0
Attrition_Flag	0
Customer_Age	0
Gender	0
Dependent_count	0
Education_Level	1519
Marital_Status	749
Income_Category	1112
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 [348]:

```
# Checking that no column has missing values in train or test sets
print(X_train.isna().sum())
print("-" * 30)
print(X_val.isna().sum())
print("-" * 30)
print(X_test.isna().sum())
```

```
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
------------------------------
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
------------------------------
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 [349]:

```
cols = X_train.select_dtypes(include=["object", "category"])
for i in cols.columns:
    print(X_train[i].value_counts())
    print("*" * 30)
```

```
F    4279
M    3822
Name: Gender, dtype: int64
******************************
Graduate         3733
High School      1619
Uneducated       1171
College           816
Post-Graduate     407
```

```
Doctorate          355
Name: Education_Level, dtype: int64
******************************
Married     4346
Single      3144
Divorced     611
Name: Marital_Status, dtype: int64
******************************
Less than $40K    3701
$40K - $60K       1453
$80K - $120K      1237
$60K - $80K       1122
$120K +            588
Name: Income_Category, dtype: int64
******************************
Blue        7557
Silver       436
Gold          93
Platinum      15
Name: Card_Category, dtype: int64
******************************
```

In [350]:

```
cols = X_val.select_dtypes(include=["object", "category"])
for i in cols.columns:
    print(X_val[i].value_counts())
    print("*" * 30)
```

```
F    266
M    241
Name: Gender, dtype: int64
******************************
Graduate         237
High School       94
Uneducated        84
College           49
Doctorate         24
Post-Graduate     19
Name: Education_Level, dtype: int64
******************************
Married     272
Single      193
Divorced     42
Name: Marital_Status, dtype: int64
******************************
Less than $40K    236
$40K - $60K        88
$60K - $80K        74
$80K - $120K       71
$120K +            38
Name: Income_Category, dtype: int64
******************************
Blue        465
Silver       37
Gold          3
Platinum      2
Name: Card_Category, dtype: int64
******************************
```

In [351]:

```
cols = X_test.select_dtypes(include=["object", "category"])
for i in cols.columns:
    print(X_train[i].value_counts())
    print("*" * 30)
```

```
F    4279
M    3822
Name: Gender, dtype: int64
******************************
Graduate          3733
```

```
High School      1619
Uneducated       1171
College           816
Post-Graduate     407
Doctorate         355
Name: Education_Level, dtype: int64
******************************
Married     4346
Single      3144
Divorced     611
Name: Marital_Status, dtype: int64
******************************
Less than $40K    3701
$40K - $60K       1453
$80K - $120K      1237
$60K - $80K       1122
$120K +            588
Name: Income_Category, dtype: int64
******************************
Blue        7557
Silver       436
Gold          93
Platinum      15
Name: Card_Category, dtype: int64
******************************
```

Encoding categorical variables

In [352]:

```
X_train = pd.get_dummies(X_train, drop_first=True)
X_val = pd.get_dummies(X_val, drop_first=True)   ## Complete the code to impute missing va
lues in X_val
X_test = pd.get_dummies(X_test, drop_first=True) ## Complete the code to impute missing
values in X_val
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(8101, 29) (507, 29) (1519, 29)
```

- **After encoding there are 29 columns.**

In [353]:

```
# check the top 5 rows from the train dataset
X_train.head()
```

Out[353]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_
9066	54	1	36	1	3	
5814	58	4	48	1	4	
792	45	4	36	6	1	
1791	34	2	36	4	3	
5011	49	2	39	5	3	

Model Building

Model evaluation criterion

Model can make wrong predictions as:

Model can make wrong predictions as:

- **Predicting a customer will attrite and the customer doesn't attrite**
- **Predicting a customer will not attrite and the customer attrites**

Which case is more important?

- **Predicting that customer will not attrite but he attrites i.e. losing on a valuable customer or asset.**

How to reduce this loss i.e need to reduce False Negatives??

- **Bank would want Recall to be maximized, greater the Recall higher the chances of minimizing false negatives. Hence, the focus should be on increasing Recall or minimizing the false negatives or in other words identifying the true positives(i.e. Class 1) so that the bank can retain their valuable customers by identifying the customers who are at risk of attrition.**

Let's define a function to output different metrics (including recall) on the train and test set and a function to show confusion matrix so that we do not have to use the same code repetitively while evaluating models.

In [354]:

```
# defining a function to compute different metrics to check performance of a classificati
on model built using sklearn
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
        index=[0],
    )

    return df_perf
```

In [355]:

```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
```

```
    plt.xlabel("Predicted label")
```

Model Building - Original Data

In [356]:

```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1)))
models.append(("XGBoost", XGBClassifier(random_state=1))) ## Complete the code to append remaining 3 models in the list models

print("\n" "Training Performance:" "\n")
for name, model in models:
    model.fit(X_train, y_train)
    scores = recall_score(y_train, model.predict(X_train))
    print("{}: {}".format(name, scores))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train, y_train)
    scores_val = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores_val))
```

```
Training Performance:

Bagging: 0.98
Random forest: 1.0
XGBoost: 1.0

Validation Performance:

Bagging: 0.8513513513513513
Random forest: 0.7567567567567568
XGBoost: 0.9324324324324325
```

Model Building - Oversampled Data

In [357]:

```
print("Before Oversampling, counts of label 'Yes': {}".format(sum(y_train == 1)))
print("Before Oversampling, counts of label 'No': {} \n".format(sum(y_train == 0)))

sm = SMOTE(
    sampling_strategy=1, k_neighbors=5, random_state=1
)  # Synthetic Minority Over Sampling Technique
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)


print("After Oversampling, counts of label 'Yes': {}".format(sum(y_train_over == 1)))
print("After Oversampling, counts of label 'No': {} \n".format(sum(y_train_over == 0)))


print("After Oversampling, the shape of train_X: {}".format(X_train_over.shape))
print("After Oversampling, the shape of train_y: {} \n".format(y_train_over.shape))
```

```
Before Oversampling, counts of label 'Yes': 1300
Before Oversampling, counts of label 'No': 6801

After Oversampling, counts of label 'Yes': 6801
After Oversampling, counts of label 'No': 6801

After Oversampling, the shape of train_X: (13602, 29)
After Oversampling, the shape of train_y: (13602,)
```

In [358]:

```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1)))
models.append(("XGBoost", XGBClassifier(random_state=1))) ## Complete the code to append remaining 3 models in the list models


print("\n" "Training Performance:" "\n")
for name, model in models:
    model.fit(X_train_over, y_train_over)
    scores = recall_score(y_val, model.predict(X_val))  ## Complete the code to
    print("{}: {}".format(name, scores))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train_over, y_train_over)
    scores = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores))
```

```
Training Performance:

Bagging: 0.9054054054054054
Random forest: 0.8378378378378378
XGBoost: 0.918918918918919

Validation Performance:

Bagging: 0.9054054054054054
Random forest: 0.8378378378378378
XGBoost: 0.918918918918919
```

Model Building - Undersampled Data

In [359]:

```
rus = RandomUnderSampler(random_state=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

In [360]:

```
print("Before Under Sampling, counts of label 'Yes': {}".format(sum(y_train == 1)))
print("Before Under Sampling, counts of label 'No': {} \n".format(sum(y_train == 0)))

print("After Under Sampling, counts of label 'Yes': {}".format(sum(y_train_un == 1)))
print("After Under Sampling, counts of label 'No': {} \n".format(sum(y_train_un == 0)))

print("After Under Sampling, the shape of train_X: {}".format(X_train_un.shape))
print("After Under Sampling, the shape of train_y: {} \n".format(y_train_un.shape))
```

```
Before Under Sampling, counts of label 'Yes': 1300
Before Under Sampling, counts of label 'No': 6801

After Under Sampling, counts of label 'Yes': 1300
After Under Sampling, counts of label 'No': 1300

After Under Sampling, the shape of train_X: (2600, 29)
After Under Sampling, the shape of train_y: (2600,)
```

In [361]:

```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=1)))
```

```
models.append(("Random forest", RandomForestClassifier(random_state=1)))
models.append(("XGBoost", XGBClassifier(random_state=1))) ## Complete the code to append remaining 3 models in the list models


print("\n" "Training Performance:" "\n")
for name, model in models:
    model.fit(X_train_un, y_train_un)
    scores = recall_score(y_val, model.predict(X_val))  ## Complete the code to build models on undersampled data
    print("{}: {}".format(name, scores))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train_un, y_train_un)
    scores = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores))
```

```
Training Performance:

Bagging: 0.918918918918919
Random forest: 0.9459459459459459
XGBoost: 0.972972972972973

Validation Performance:

Bagging: 0.918918918918919
Random forest: 0.9459459459459459
XGBoost: 0.972972972972973
```

Hyperparameter Tuning

Note

1. **Sample parameter grids have been provided to do necessary hyperparameter tuning. These sample grids are expected to provide a balance between model performance improvement and execution time. One can extend/reduce the parameter grid based on execution time and system configuration.**
    - **Please note that if the parameter grid is extended to improve the model performance further, the execution time will increase**
2. **The models chosen in this notebook are based on test runs. One can update the best models as obtained upon code execution and tune them for best performance.**

Tuning AdaBoost using original data

In [362]:

```
%%time

# defining model
Model = AdaBoostClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "n_estimators": np.arange(50,110,25),
    "learning_rate": [0.01,0.1,0.05],
    "base_estimator": [
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
```

```
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_job
s = -1, n_iter=50, scoring=scorer, cv=5, random_state=1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train, y_train) ## Complete the code to fit the model on original dat
a

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
omized_cv.best_score_))
```

```
Best parameters are {'n_estimators': 100, 'learning_rate': 0.1, 'base_estimator': Decisio
nTreeClassifier(max_depth=3, random_state=1)} with CV score=0.8692307692307691:
CPU times: user 4.76 s, sys: 441 ms, total: 5.2 s
Wall time: 2min 20s
```

In [363]:

```
tuned_adb = AdaBoostClassifier( random_state=1, # random_state set to 1 for reproducibili
ty
    n_estimators= 100, learning_rate= 0.1, base_estimator= DecisionTreeClassifier(max_de
pth=3, random_state=1)
) ## Complete the code with the best parameters obtained from tuning

tuned_adb.fit(X_train, y_train)
```

Out[363]:

```
▸ AdaBoostClassifier
▸ base_estimator: DecisionTreeClassifier
▸ DecisionTreeClassifier
```

In [364]:

```
adb_train = model_performance_classification_sklearn(tuned_adb, X_train, y_train) ## Com
plete the code to check the performance on training set
adb_train
```

Out[364]:

	Accuracy	Recall	Precision	F1
0	0.985	0.934	0.969	0.951

In [365]:

```
adb_val = model_performance_classification_sklearn(tuned_adb, X_val, y_val) ## Complete
the code to check the performance on validation set
adb_val
```

Out[365]:

	Accuracy	Recall	Precision	F1
0	0.966	0.892	0.880	0.886

Tuning Ada Boost using undersampled data

In [366]:

```
# Creating new pipeline with best parameters
tuned_ada2 = AdaBoostClassifier( random_state=1,
    n_estimators= 50, learning_rate= 0.1, base_estimator= DecisionTreeClassifier(max_dep
th=3, random_state=1)
) ## Complete the code with the best parameters obtained from tuning

tuned_ada2.fit(X_train_un, y_train_un)  ## Complete the code to fit the model on undersam
```

```
pled data
```

Out[366]:

```
▸          AdaBoostClassifier
▸ base_estimator: DecisionTreeClassifier
  ▸        DecisionTreeClassifier
```

In [367]:

```
adb2_train = model_performance_classification_sklearn(tuned_ada2, X_train_un, y_train_un)
## Complete the code to check the performance on training set
adb2_train
```

Out[367]:

	Accuracy	Recall	Precision	F1
0	0.973	0.978	0.969	0.974

In [368]:

```
adb2_val = model_performance_classification_sklearn(tuned_ada2, X_val, y_val) ## Complete
the code to check the performance on validation set
adb2_val
```

Out[368]:

	Accuracy	Recall	Precision	F1
0	0.945	0.959	0.740	0.835

Tuning Gradient Boosting using undersampled data

In [369]:

```
%%time

#Creating pipeline
Model = GradientBoostingClassifier(random_state=1)

#Parameter grid to pass in RandomSearchCV
param_grid = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random_state=1)],
    "n_estimators": np.arange(50,110,25),
    "learning_rate": [0.01,0.1,0.05],
    "subsample":[0.7,0.9],
    "max_features":[0.5,0.7,1],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_ite
r=50, scoring=scorer, cv=5, random_state=1, n_jobs = -1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_un, y_train_un) ## Complete the code to fit the model on under
sampled data


print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
omized_cv.best_score_))
```

```
Best parameters are {'subsample': 0.9, 'n_estimators': 100, 'max_features': 0.5, 'learnin
g_rate': 0.1, 'init': AdaBoostClassifier(random_state=1)} with CV score=0.955384615384615
```

```
4:
CPU times: user 2.7 s, sys: 213 ms, total: 2.91 s
Wall time: 1min 26s
```

In [370]:

```
# Creating new pipeline with best parameters
tuned_gbm1 = GradientBoostingClassifier(
    max_features=0.5,
    init=AdaBoostClassifier(random_state=1),
    random_state=1,
    learning_rate=0.1,
    n_estimators=100,
    subsample=0.9,
)## Complete the code with the best parameters obtained from tuning

tuned_gbm1.fit(X_train_un, y_train_un)
```

Out[370]:

```
▸ GradientBoostingClassifier
▸ init: AdaBoostClassifier
  ▸ AdaBoostClassifier
```

In [371]:

```
gbm1_train = model_performance_classification_sklearn(tuned_gbm1, X_train_un, y_train_un)
## Complete the code to check the performance on undersampled train set
gbm1_train
```

Out[371]:

	Accuracy	Recall	Precision	F1
0	0.976	0.980	0.972	0.976

In [372]:

```
gbm1_val = model_performance_classification_sklearn(tuned_gbm1, X_val, y_val) ## Complete the code to check the performance on validation set
gbm1_val
```

Out[372]:

	Accuracy	Recall	Precision	F1
0	0.943	0.946	0.737	0.828

Tuning Gradient Boosting using original data

In [373]:

```
%%time

#defining model
Model = GradientBoostingClassifier(random_state=1)

#Parameter grid to pass in RandomSearchCV
param_grid = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random_state=1)],
    "n_estimators": np.arange(50,110,25),
    "learning_rate": [0.01,0.1,0.05],
    "subsample":[0.7,0.9],
    "max_features":[0.5,0.7,1],
}
```

```
# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_iter=50, scoring=scorer, cv=5, random_state=1, n_jobs = -1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train, y_train) ## Complete the code to fit the model on original data


print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,randomized_cv.best_score_))
```

```
Best parameters are {'subsample': 0.9, 'n_estimators': 100, 'max_features': 0.5, 'learning_rate': 0.1, 'init': AdaBoostClassifier(random_state=1)} with CV score=0.8400000000000001:
CPU times: user 5.88 s, sys: 478 ms, total: 6.36 s
Wall time: 3min 32s
```

In [374]:

```
# Creating new pipeline with best parameters
tuned_gbm2 = GradientBoostingClassifier(
    max_features=0.5,
    init=AdaBoostClassifier(random_state=1),
    random_state=1,
    learning_rate=0.1,
    n_estimators=100,
    subsample=0.9,
)## Complete the code with the best parameters obtained from tuning

tuned_gbm2.fit(X_train, y_train)
```

Out[374]:

```
▸ GradientBoostingClassifier
▸ init: AdaBoostClassifier
▸ AdaBoostClassifier
```

Tuning Gradient Boosting using over sampled data

In [375]:

```
gbm2_train = model_performance_classification_sklearn(tuned_gbm2, X_train_over, y_train_over) ## Complete the code to check the performance on oversampled train set
gbm2_train
```

Out[375]:

	Accuracy	Recall	Precision	F1
0	0.928	0.862	0.992	0.922

In [376]:

```
gbm2_val = model_performance_classification_sklearn(tuned_gbm2, X_val, y_val) ## Complete the code to check the performance on validation set
gbm2_val
```

Out[376]:

	Accuracy	Recall	Precision	F1
0	0.959	0.784	0.921	0.847

Tuning XGBoost Model with Original data

Note: This section is optional. You can choose not to build XGBoost if you are facing issues with installation or if it is taking more time to execute.

In [377]:

```
%%time

# defining model
Model = XGBClassifier(random_state=1,eval_metric='logloss')

#Parameter grid to pass in RandomSearchCV
param_grid={'n_estimators':np.arange(50,110,25),
            'scale_pos_weight':[1,2,5],
            'learning_rate':[0.01,0.1,0.05],
            'gamma':[1,3],
            'subsample':[0.7,0.9]
           }
from sklearn import metrics

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_ite
r=50, n_jobs = -1, scoring=scorer, cv=5, random_state=1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train, y_train) ## Complete the code to fit the model on original dat
a

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
omized_cv.best_score_))
```

```
Best parameters are {'subsample': 0.7, 'scale_pos_weight': 5, 'n_estimators': 75, 'learni
ng_rate': 0.05, 'gamma': 3} with CV score=0.9353846153846155:
CPU times: user 2.39 s, sys: 292 ms, total: 2.69 s
Wall time: 1min 10s
```

In [378]:

```
tuned_xgb = XGBClassifier(
    random_state=1,
    eval_metric="logloss",
    subsample=0.7,
    scale_pos_weight=5,
    n_estimators=75,
    learning_rate=0.05,
    gamma=1,
)## Complete the code with the best parameters obtained from tuning

tuned_xgb.fit(X_train, y_train)
```

Out[378]:

▾ XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None
,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=1, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.05, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
```

In [381]:

```
xgb_train = model_performance_classification_sklearn(tuned_xgb, X_train, y_train) ## Com
plete the code to check the performance on original train set
xgb_train
```

Out[381]:

	Accuracy	Recall	Precision	F1
0	0.977	0.992	0.880	0.933

In [383]:

```
xgb_val = model_performance_classification_sklearn(tuned_xgb, X_val, y_val) ## Complete
the code to check the performance on validation set
xgb_val
```

Out[383]:

	Accuracy	Recall	Precision	F1
0	0.949	0.946	0.761	0.843

Model Comparison and Final Model Selection

Note: If you want to include XGBoost model for final model selection, you need to add xgb_train.T in the training performance comparison list and xgb_val.T in the validation performance comparison list below.

In [400]:

```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        gbm1_train.T,
        gbm2_train.T,
        adb2_train.T,
        adb_train.T,
        xgb_un_train.T,
        xgb_train.T,

    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Gradient boosting trained with Undersampled data",
    "Gradient boosting trained with Original data",
    "AdaBoost trained with Undersampled data",
    "AdaBoost trained with Original data",
    "XGBoost trained with Undersampled data",
    "XGBoost trained with Original data",
]
print("Training performance comparison:")
models_train_comp_df
```

Training performance comparison:

Out[400]:

	Gradient boosting trained with Undersampled data	Gradient boosting trained with Original data	AdaBoost trained with Undersampled data	AdaBoost trained with Original data	XGBoost trained with Undersampled data	XGBoost trained with Original data
Accuracy	0.976	0.928	0.973	0.985	0.774	0.977
Recall	0.980	0.862	0.978	0.934	1.000	0.992

	Gradient boosting trained with Undersampled data	Gradient boosting trained with Original data	AdaBoost trained with Undersampled data	AdaBoost trained with Original data	XGBoost trained with Undersampled data	XGBoost trained with Original data
Precision	0.972	0.992	0.969	0.969	0.689	0.880
F1	0.976	0.922	0.974	0.951	0.816	0.933

In [401]:

```
# validation performance comparison

 ## Write the code to compare the performance on validation set

# training performance comparison

models_val_comp_df = pd.concat(
    [
        gbm1_val.T,
        gbm2_val.T,
        adb2_val.T,
        adb_val.T,
        xgb_un_val.T,
        xgb_val.T,
    ],
    axis=1,
)
models_val_comp_df.columns = [
    "Gradient boosting validated with Undersampled data",
    "Gradient boosting validated with Original data",
    "AdaBoost validated with Undersampled data",
    "AdaBoost validated with Original data",
    "XGBBoost validated with Undersample data",
    "XGBBoost validated with Original data",
]
print("Validation performance comparison:")
models_val_comp_df
```

```
Validation performance comparison:
```

Out[401]:

	Gradient boosting validated with Undersampled data	Gradient boosting validated with Original data	AdaBoost validated with Undersampled data	AdaBoost validated with Original data	XGBBoost validated with Undersample data	XGBBoost validated with Original data
Accuracy	0.943	0.959	0.945	0.966	0.542	0.949
Recall	0.946	0.784	0.959	0.892	1.000	0.946
Precision	0.737	0.921	0.740	0.880	0.242	0.761
F1	0.828	0.847	0.835	0.886	0.389	0.843

Now we have our final model, so let's find out how our final model is performing on unseen test data.

In [386]:

```
# Let's check the performance on test set
## Write the code to check the performance of best model on test data


# Assuming 'randomized_cv' is the RandomizedSearchCV object that trained the XGBoost mode
l
best_model = randomized_cv.best_estimator_

# Evaluate the best model on the test set
test_performance = model_performance_classification_sklearn(best_model, X_test, y_test)

print("best_model", best_model)

# Print the test performance metrics
test_performance
```

```
best_model XGBClassifier(base_score=None, booster=None, callbacks=None,
```

```
best_model XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=3, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.05, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=75,
              n_jobs=None, num_parallel_tree=None, random_state=1, ...)
```

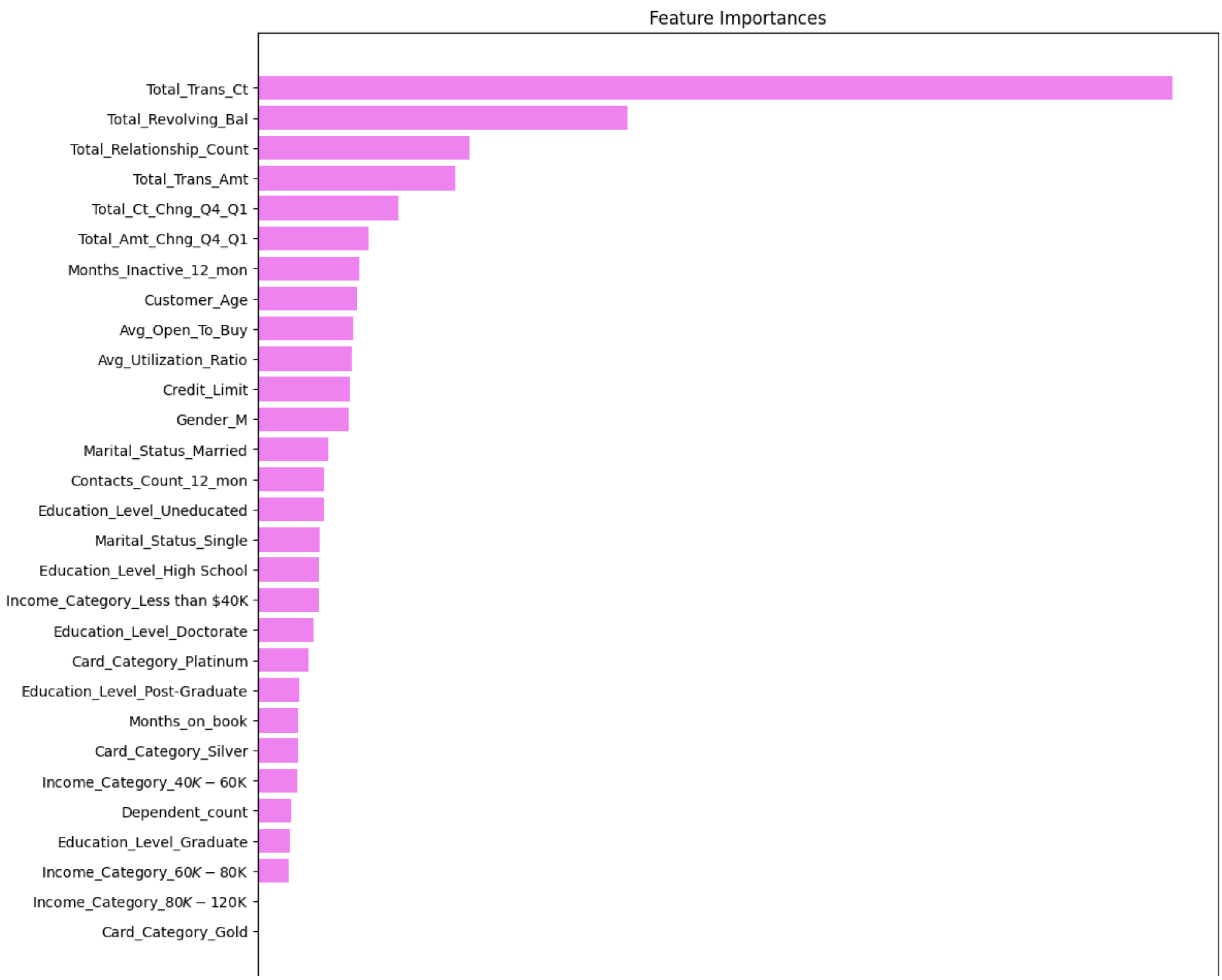
Out[386]:

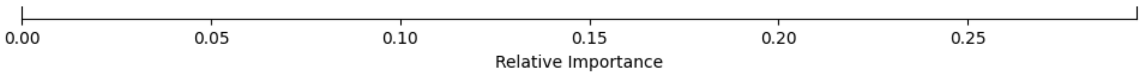
	Accuracy	Recall	Precision	F1
0	0.953	0.925	0.818	0.868

Feature Importances

In [387]:

```python
feature_names = X_train.columns
importances =  best_model.feature_importances_ ## Complete the code to check the feature importance of the best model
indices = np.argsort(importances)

plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```





Lets create a model with XGBooster & undersample data to see if we can improve the performace further

In [395]:

```
%%time

# defining model
Model = XGBClassifier(random_state=1,eval_metric='logloss')

#Parameter grid to pass in RandomSearchCV
param_grid={'n_estimators':np.arange(50,110,25),
            'scale_pos_weight':[1,2,5],
            'learning_rate':[0.01,0.1,0.05],
            'gamma':[1,3],
            'subsample':[0.7,0.9]
           }
from sklearn import metrics

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_ite
r=50, n_jobs = -1, scoring=scorer, cv=5, random_state=1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_un, y_train_un) ## Complete the code to fit the model on origin
al data

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
omized_cv.best_score_))
```

```
Best parameters are {'subsample': 0.7, 'scale_pos_weight': 5, 'n_estimators': 50, 'learni
ng_rate': 0.01, 'gamma': 3} with CV score=0.9984615384615385:
CPU times: user 1.6 s, sys: 263 ms, total: 1.86 s
Wall time: 46.9 s
```

Lets train the model with undersample data

In [ ]:

```

```

In [396]:

```
tuned_xgb_un = XGBClassifier(
    random_state=1,
    eval_metric="logloss",
    subsample=0.7,
    scale_pos_weight=5,
    n_estimators=50,
    learning_rate=0.01,
    gamma=3,
)## Complete the code with the best parameters obtained from tuning

tuned_xgb_un.fit(X_train_un, y_train_un)
```

Out[396]:

```
▾                          XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None
```

```
,
             enable_categorical=False, eval_metric='logloss',
             feature_types=None, gamma=3, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.01, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
```

Model performace improved on training and validation dataset (100% recall) Which s important paramter to identify all false negative cases

In [398]:

```
xgb_un_train = model_performance_classification_sklearn(tuned_xgb_un, X_train_un, y_trai
n_un) ## Complete the code to check the performance on undersampled train set
xgb_un_train
```

Out[398]:

	Accuracy	Recall	Precision	F1
0	0.774	1.000	0.689	0.816

In [399]:

```
xgb_un_val = model_performance_classification_sklearn(tuned_xgb_un, X_val, y_val) ## Com
plete the code to check the performance on undersampled train set
xgb_un_val
```

Out[399]:

	Accuracy	Recall	Precision	F1
0	0.542	1.000	0.242	0.389

ADA, GB and XGB model performace on TEST SET with original and undersample data.

In [392]:

```
test_performance_adb_original_data = model_performance_classification_sklearn(tuned_adb,
X_test, y_test)
print("test_performance_adb_original_data")
print(test_performance_adb_original_data)
print("*" * 50)
test_performance_adb_undersample_data = model_performance_classification_sklearn(tuned_ad
a2, X_test, y_test)
print("test_performance_adb_undersample_data")
print(test_performance_adb_undersample_data)
print("*" * 50)
test_performance_GB_original_data = model_performance_classification_sklearn(tuned_gbm2,
X_test, y_test)
print("test_performance_GB_original_data")
print(test_performance_GB_original_data)
print("*" * 50)
test_performance_GB_undersample_data = model_performance_classification_sklearn(tuned_gbm
1, X_test, y_test)
print("test_performance_GB_undersample_data")
print(test_performance_GB_undersample_data)
print("*" * 50)
test_performance_XGB_original_data = model_performance_classification_sklearn(tuned_xgb,
X_test, y_test)
print("test_performance_XGB_original_data")
```

```python
print(test_performance_XGB_original_data)
test_performance_XGB_undersample_data = model_performance_classification_sklearn(tuned_xg
b_un, X_test, y_test)
print("test_performance_XGB_undersample_data")
print(test_performance_XGB_undersample_data)
```

```
test_performance_adb_original_data
   Accuracy  Recall  Precision    F1
0     0.967   0.846      0.951 0.895
**************************************************
test_performance_adb_undersample_data
   Accuracy  Recall  Precision    F1
0     0.935   0.957      0.736 0.832
**************************************************
test_performance_GB_original_data
   Accuracy  Recall  Precision    F1
0     0.960   0.810      0.940 0.870
**************************************************
test_performance_GB_undersample_data
   Accuracy  Recall  Precision    F1
0     0.948   0.957      0.781 0.860
**************************************************
test_performance_XGB_original_data
   Accuracy  Recall  Precision    F1
0     0.953   0.917      0.820 0.866
test_performance_XGB_undersample_data
   Accuracy  Recall  Precision    F1
0     0.594   0.996      0.290 0.450
```

Business Insights and Conclusions

XGB with original dataset gives recall 92% We can improve the model perfomace further with undersample technique which improved the performace to 100% recall rate.

Total Transaction Count, Total Revolving Balance, Total Relationship Count, Total Transaction Amount, and Total Count Change from Q4 to Q1 are identified as the top five reasons for customer attrition at Thera Bank, here are tailored recommendations to address each factor and reduce the likelihood of customer churn: Targeted Promotions and Rewards, Seasonal or Limited-Time Offers, Balance Transfer Offers, Customer Engagement, Loyalty Programs, Exclusive Services for Multi-Product Customers, Increase Engagement During the Holiday Season, Customer Segmentation are some of the recommendations, we can think of to increase Thera Bank's credit card customers.

In [ ]: