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 [7]:

```
# 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==0.13.2 matplotlib==3.7.1 numpy==1.2
```

In [8]:

```
# 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
# !pip install --upgrade -q threadpoolctl
```

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

In [9]:

```
# 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_

# 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 [10]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [11]:
```
churn = pd.read_csv("/content/drive/MyDrive/BankChurners.csv")
```

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 [12]:
```
# Checking the number of rows and columns in the training data
churn.shape ##  Complete the code to view dimensions of the train data
```

Out[12]: (10127, 21)

In [13]:
```
churn.head()
```

Out[13]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_L
0	768805383	Existing Customer	45	M	3	High Sc
1	818770008	Existing Customer	49	F	5	Grad
2	713982108	Existing Customer	51	M	3	Grad
3	769911858	Existing Customer	40	F	4	High Sc
4	709106358	Existing Customer	40	M	3	Uneduc

In [14]:
```
data = churn.copy()
```

In [15]:
```
# let's view the first 5 rows of the data
data.head() ##  Complete the code to view top 5 rows of the data
```

Out[15]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_L
0	768805383	Existing Customer	45	M	3	High Sc
1	818770008	Existing Customer	49	F	5	Grad
2	713982108	Existing Customer	51	M	3	Grad
3	769911858	Existing Customer	40	F	4	High Sc
4	709106358	Existing Customer	40	M	3	Uneduc

In [16]:

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

Out[16]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Educat
10122	772366833	Existing Customer	50	M	2	
10123	710638233	Attrited Customer	41	M	2	
10124	716506083	Attrited Customer	44	F	1	Hi
10125	717406983	Attrited Customer	30	M	2	
10126	714337233	Attrited Customer	43	F	2	

Checking the data types of the columns for the dataset

In [17]:

```
# 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 [18]:

```
# let's check for duplicate values in the data
data.duplicated().sum() ##  Complete the code to check duplicate entries in t
```

Out[18]: 0

Checking for missing values

In [19]:

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

Out[19]:

```
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 [20]:

```
# let's view the statistical summary of the numerical columns in the data
data.describe() ##  Complete the code to print the statitical summary of the
```

Out[20]:

	CLIENTNUM	Customer_Age	Dependent_count	Months_on_book	Total_Relati

count	10127.000	10127.000	10127.000	10127.000
mean	739177606.334	46.326	2.346	35.928
std	36903783.450	8.017	1.299	7.986
min	708082083.000	26.000	0.000	13.000
25%	713036770.500	41.000	1.000	31.000
50%	717926358.000	46.000	2.000	36.000
75%	773143533.000	52.000	3.000	40.000
max	828343083.000	73.000	5.000	56.000

In [21]:

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

Out[21]:

	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 [22]:

```
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 :
Attrition_Flag
Existing Customer    8500
Attrited Customer    1627
Name: count, dtype: int64
**************************************************
Unique values in Gender are :
Gender
F    5358
M    4769
Name: count, dtype: int64
**************************************************
Unique values in Education_Level are :
Education_Level
Graduate         3128
High School      2013
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: count, dtype: int64
**************************************************
Unique values in Marital_Status are :
Marital_Status
```

```
Married      4687
Single       3943
Divorced      748
Name: count, dtype: int64
**************************************************
Unique values in Income_Category are :
Income_Category
Less than $40K    3561
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
abc               1112
$120K +            727
Name: count, dtype: int64
**************************************************
Unique values in Card_Category are :
Card_Category
Blue        9436
Silver       555
Gold         116
Platinum      20
Name: count, dtype: int64
**************************************************
```

In [23]:

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

In [24]:

```
## Encoding Existing and Attrited customers to 0 and 1 respectively, for anal
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 [25]:

```
# 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
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winte
    ) 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 [26]:

```
# function to create labeled barplots


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
    """

    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 [27]:

```
# 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_valu
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_
        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 [28]:

```
### 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_uni
    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_uni
    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

    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(
        data=data,
```

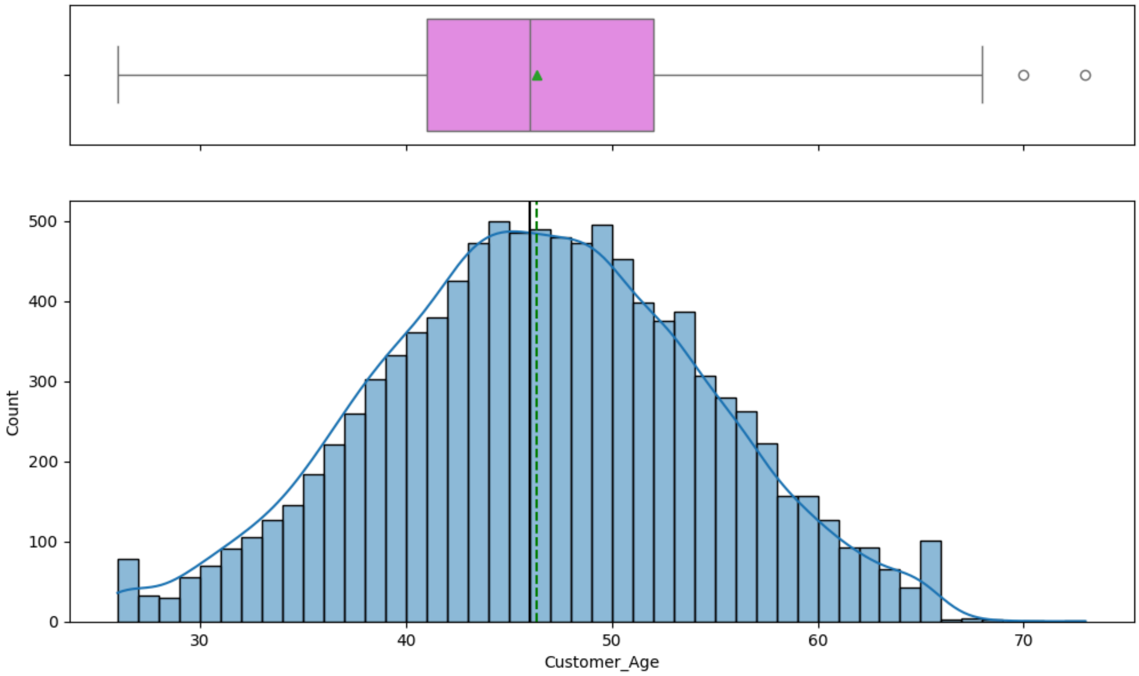
```
        x=target,
        y=predictor,
        ax=axs[1, 1],
        showfliers=False,
        palette="gist_rainbow",
    )

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

Univariate analysis

Customer_Age

In [29]:

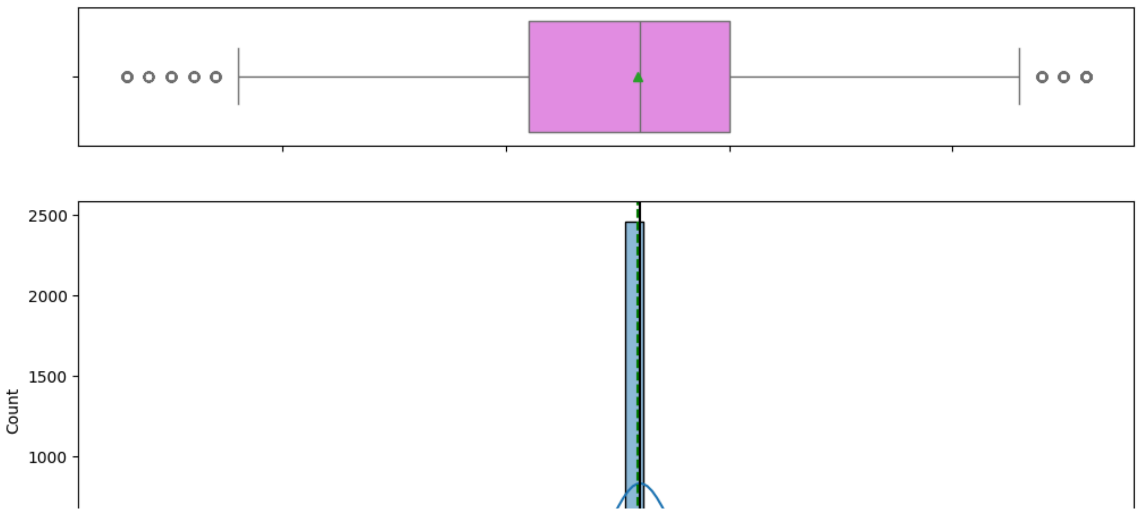
```
histogram_boxplot(data, "Customer_Age", kde=True)
```

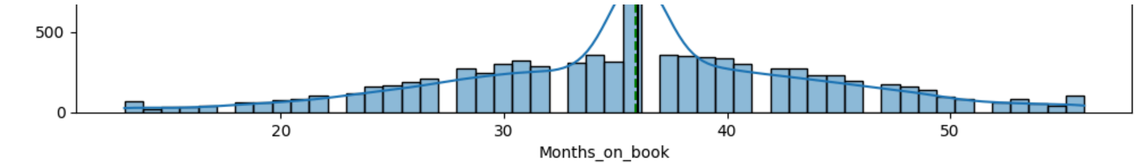


Months_on_book

In [30]:

```
histogram_boxplot(data,'Months_on_book', kde=True)  ## Complete the code to c
```

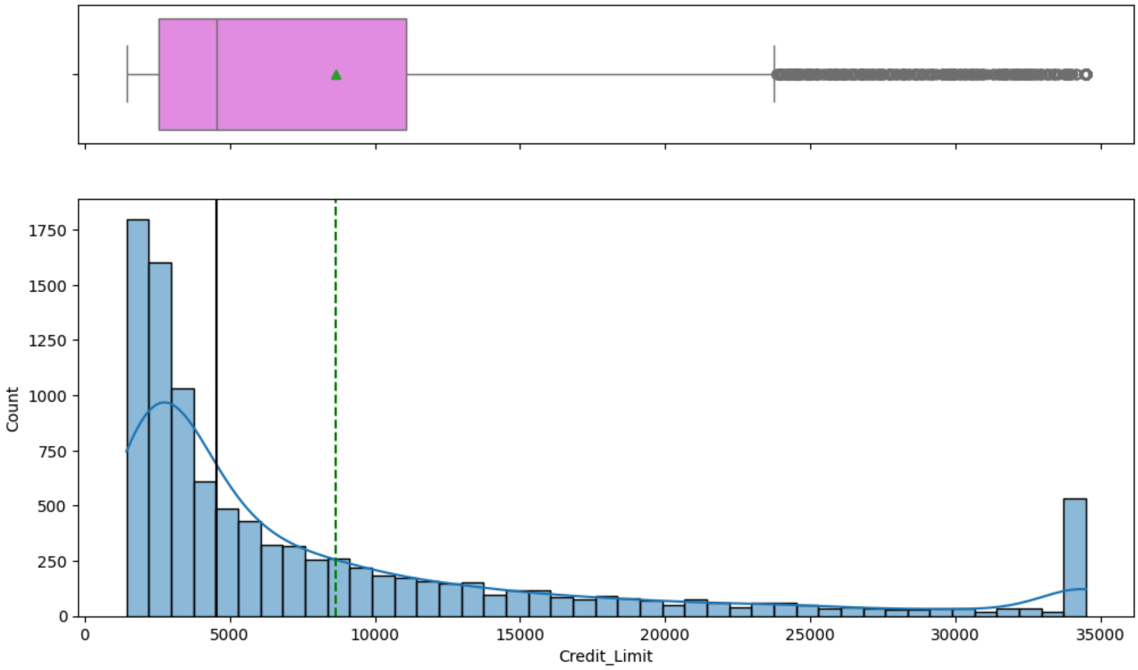




Credit_Limit

In [31]:

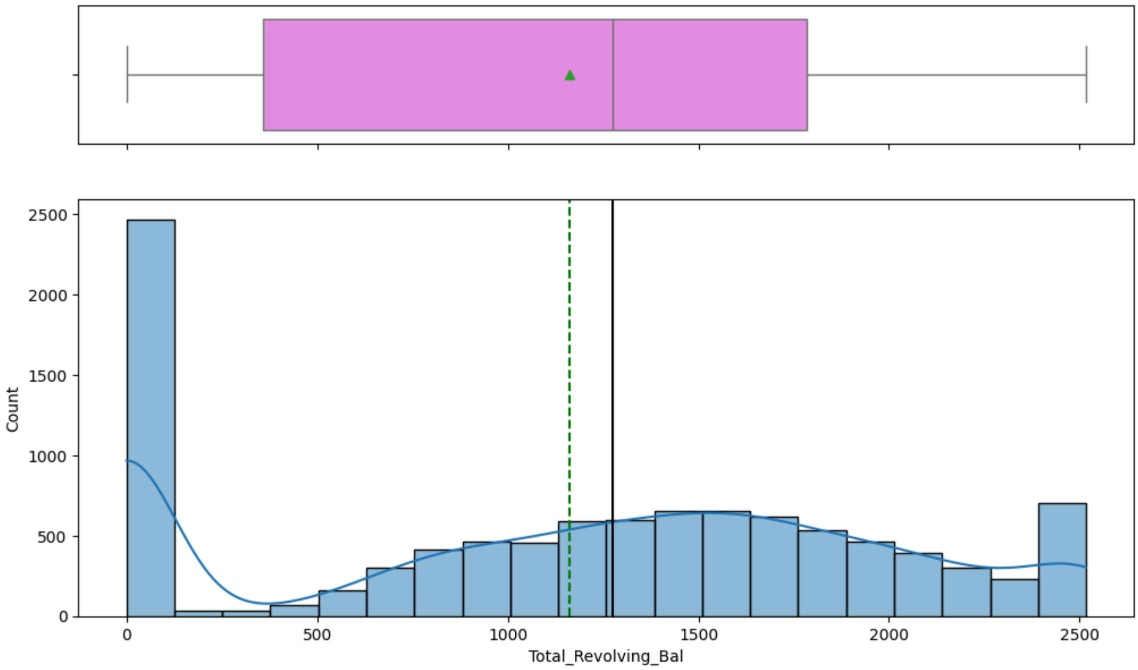
```
histogram_boxplot(data,'Credit_Limit', kde=True)  ## Complete the code to cre
```



Total_Revolving_Bal

In [32]:

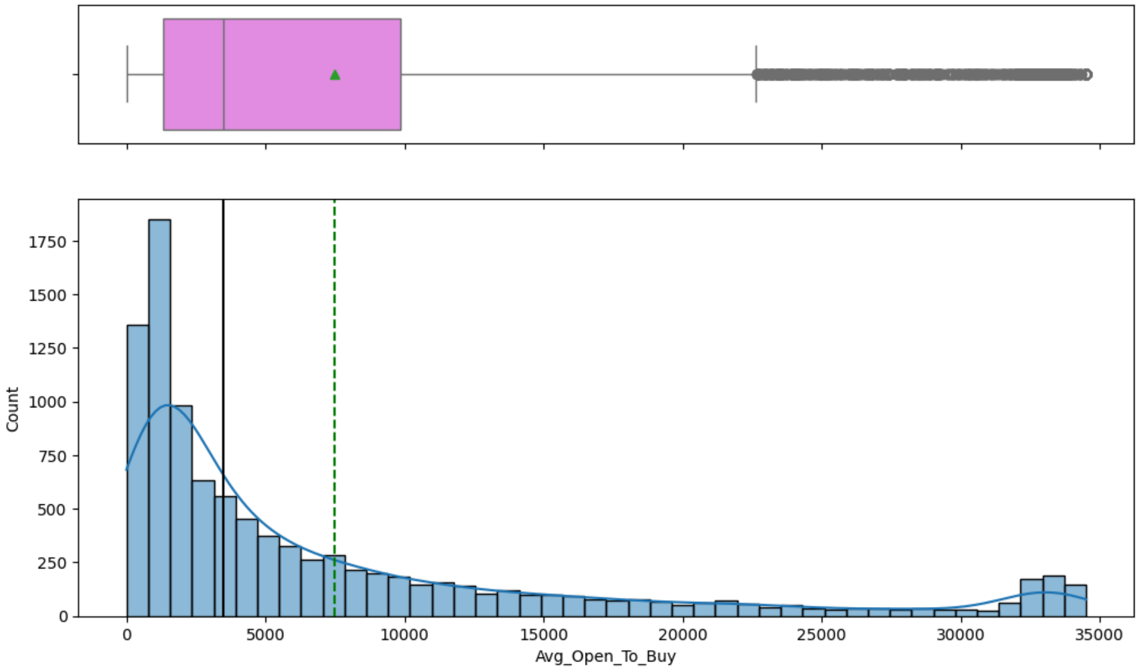
```
histogram_boxplot(data, 'Total_Revolving_Bal', kde=True)  ## Complete the cod
```



Avg_Open_To_Buy

In [33]:

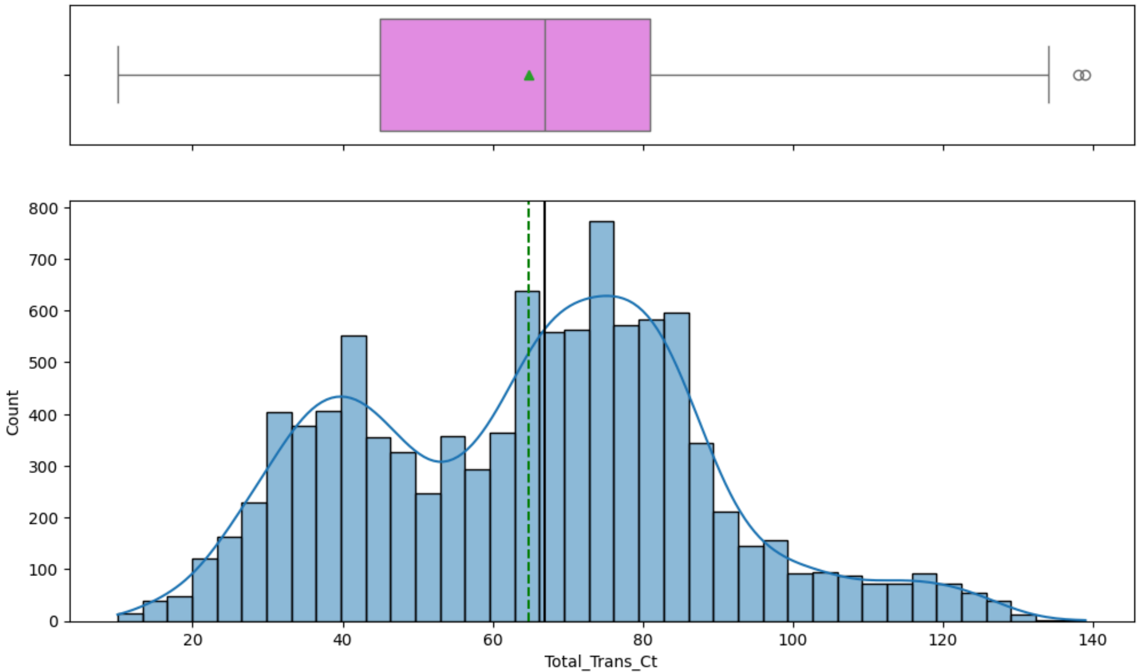
```
histogram_boxplot(data,'Avg_Open_To_Buy', kde=True)  ## Complete the code to
```



Total_Trans_Ct

In [34]:

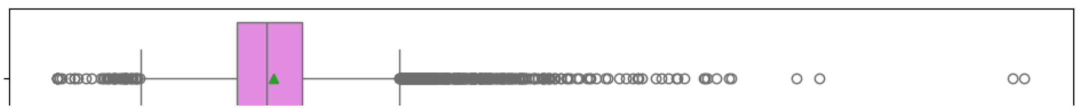
```
histogram_boxplot(data,'Total_Trans_Ct', kde=True)  ## Complete the code to c
```

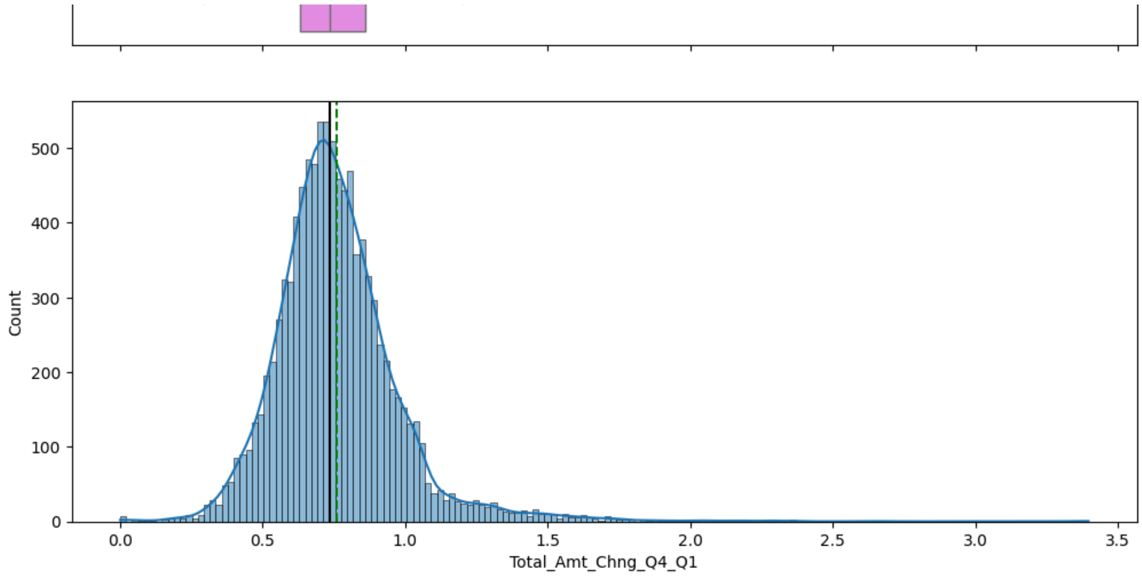


Total_Amt_Chng_Q4_Q1

In [35]:

```
histogram_boxplot(data,'Total_Amt_Chng_Q4_Q1', kde=True)  ## Complete the cod
```



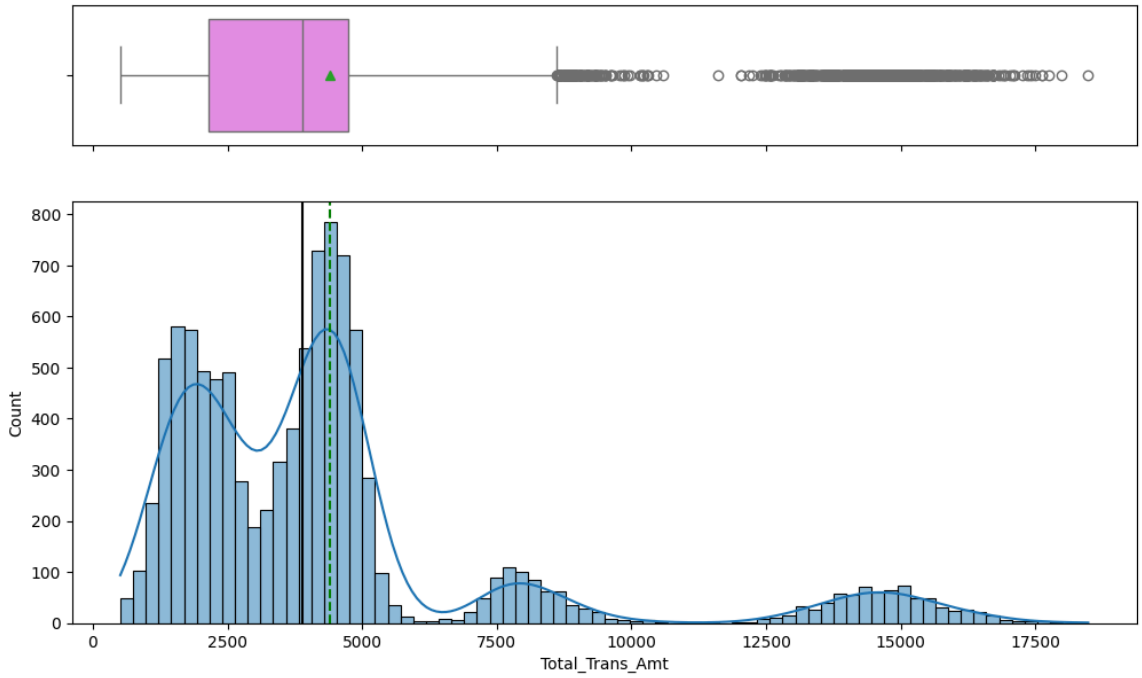


Let's see total transaction amount distributed

Total_Trans_Amt

In [36]:

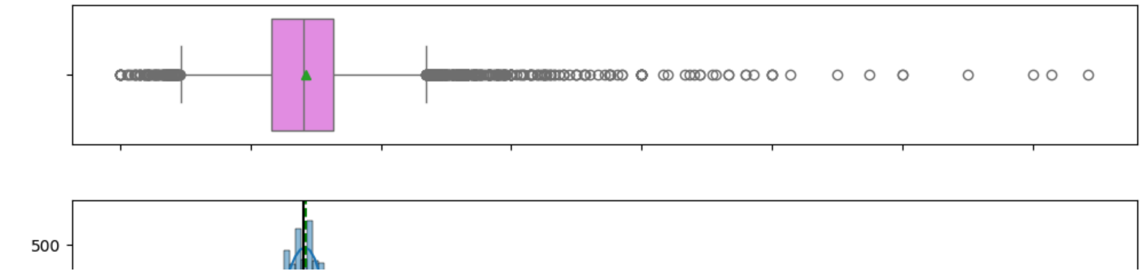
```
histogram_boxplot(data, 'Total_Trans_Amt', kde=True)  ## Complete the code to
```

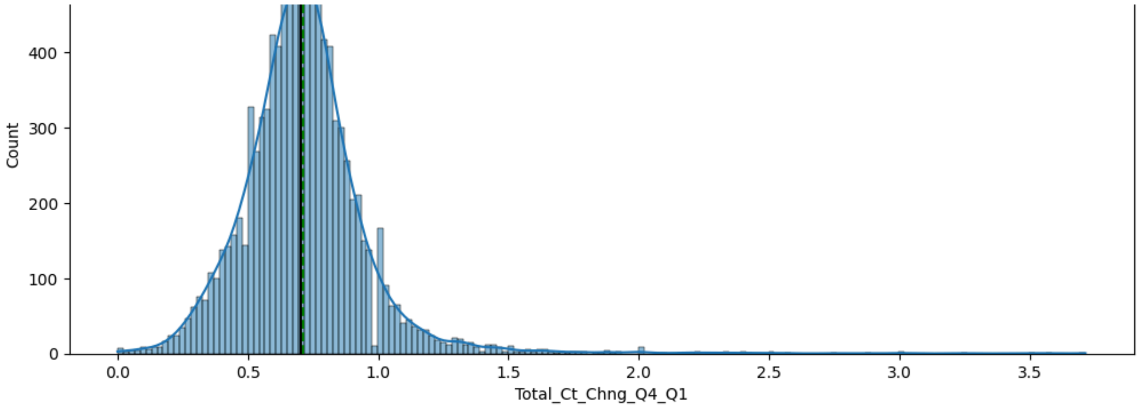


Total_Ct_Chng_Q4_Q1

In [37]:

```
histogram_boxplot(data, 'Total_Ct_Chng_Q4_Q1', kde=True)  ## Complete the cod
```

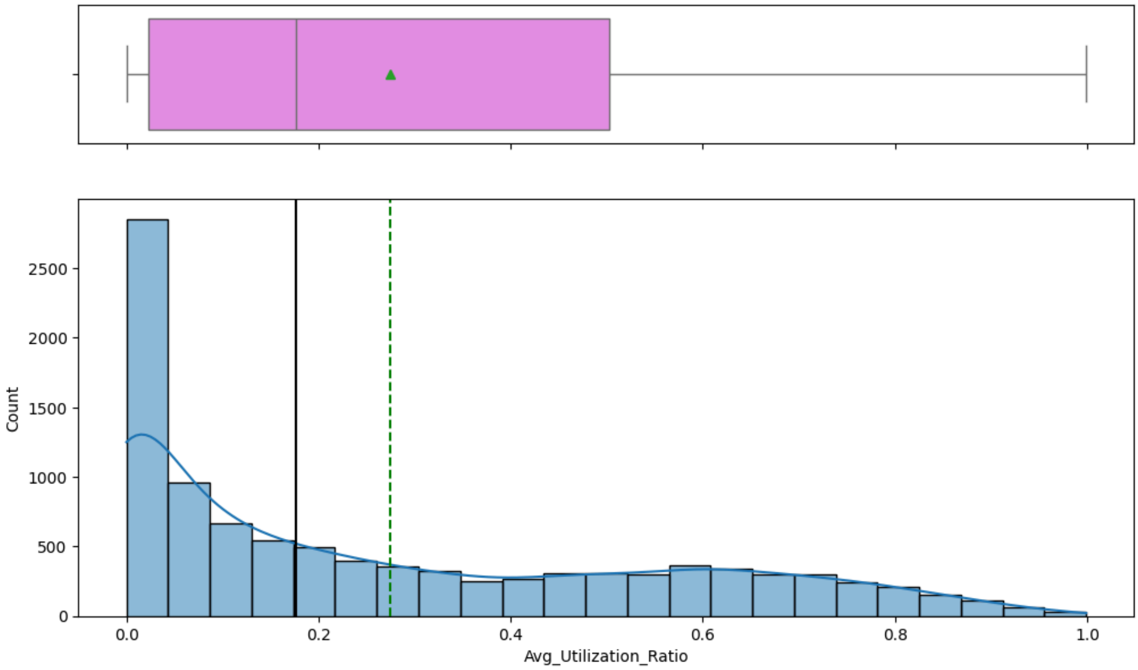




Avg_Utilization_Ratio

In [38]:

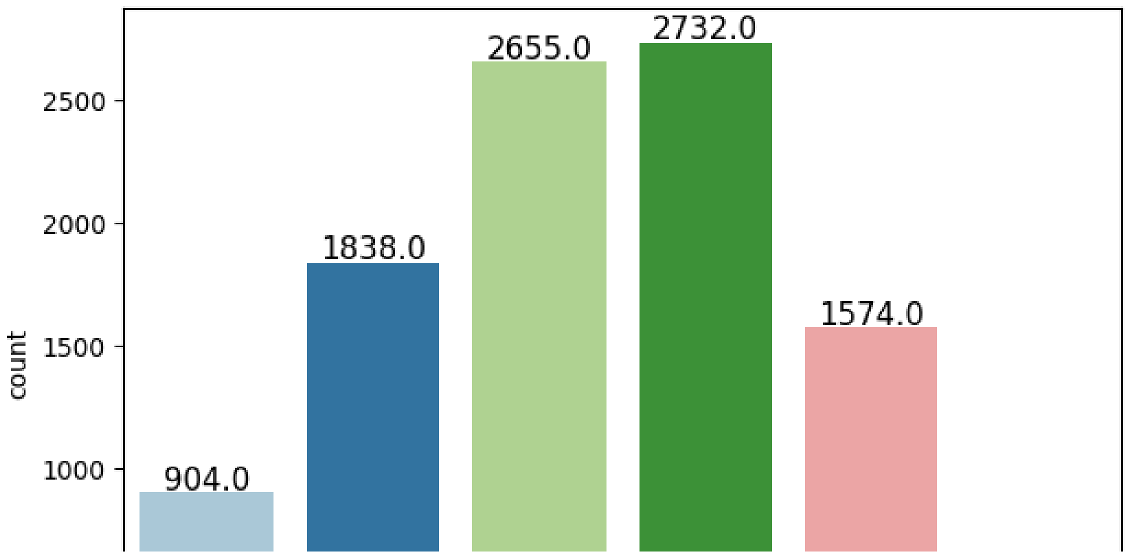
```
histogram_boxplot(data, 'Avg_Utilization_Ratio', kde=True)  ## Complete the c
```

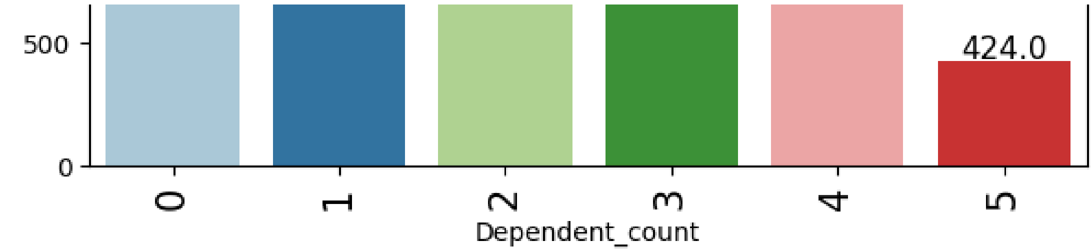


Dependent_count

In [39]:

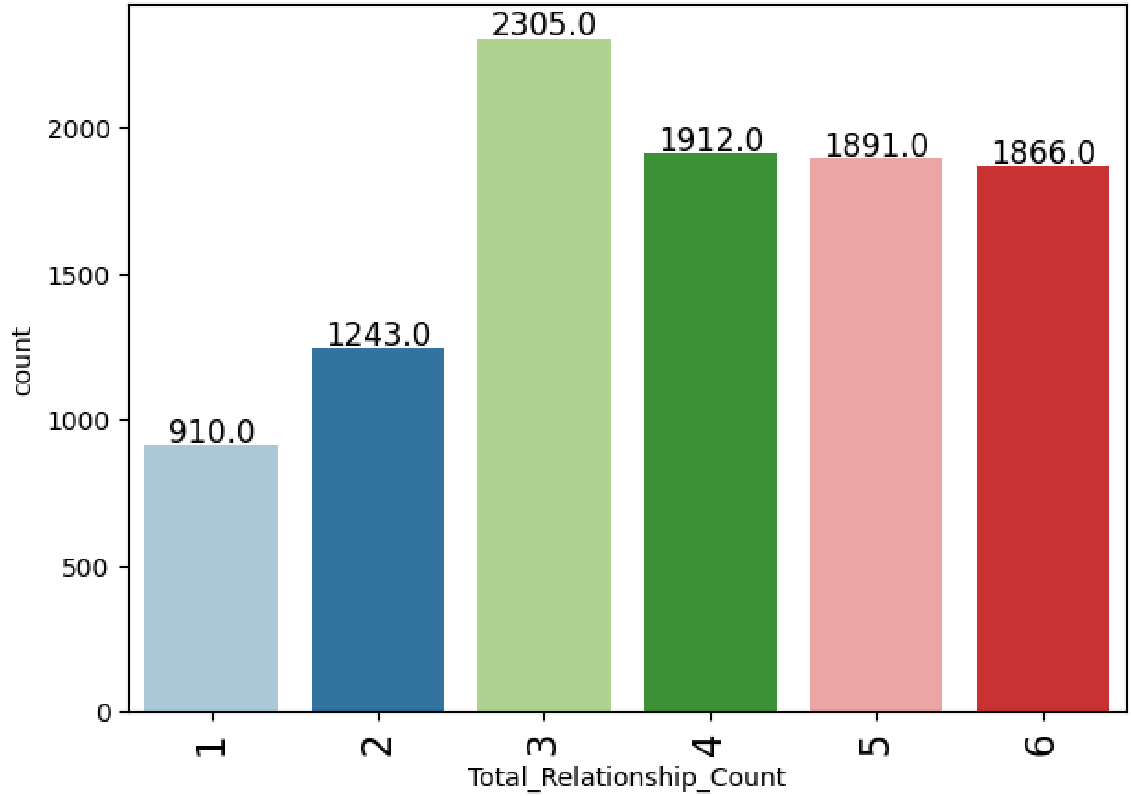
```
labeled_barplot(data, "Dependent_count")
```





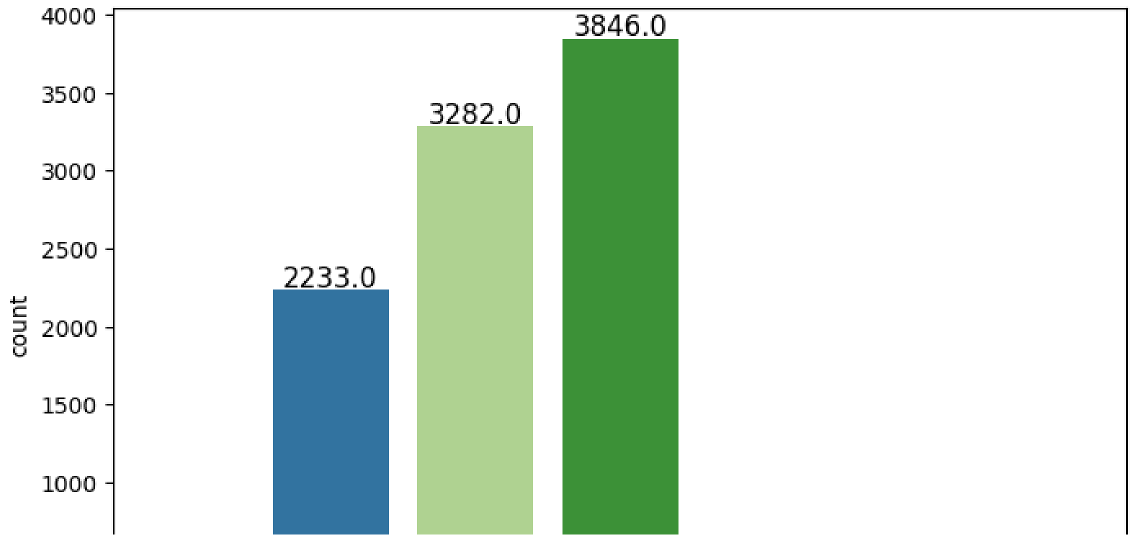
Total_Relationship_Count

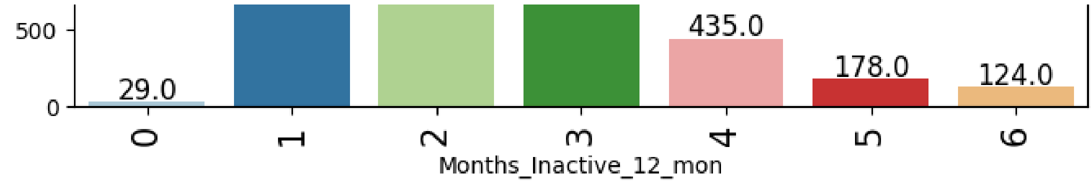
In [40]:
```
labeled_barplot(data,'Total_Relationship_Count') ## Complete the code to crea
```



Months_Inactive_12_mon

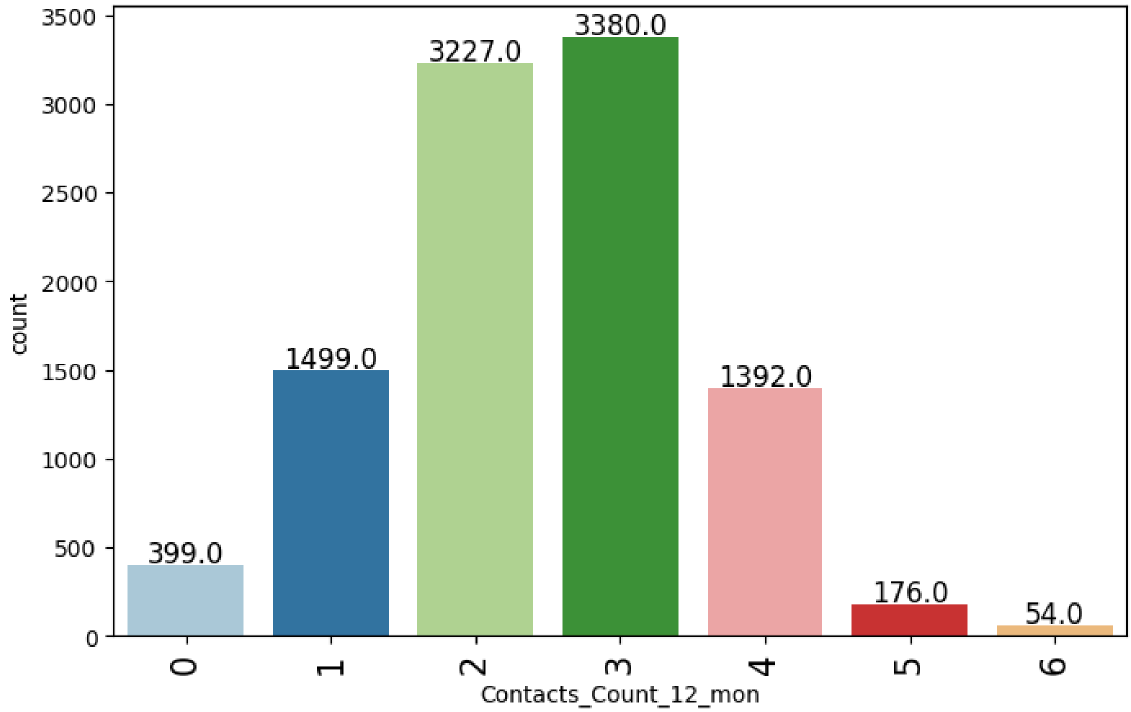
In [41]:
```
labeled_barplot(data, 'Months_Inactive_12_mon') ## Complete the code to creat
```





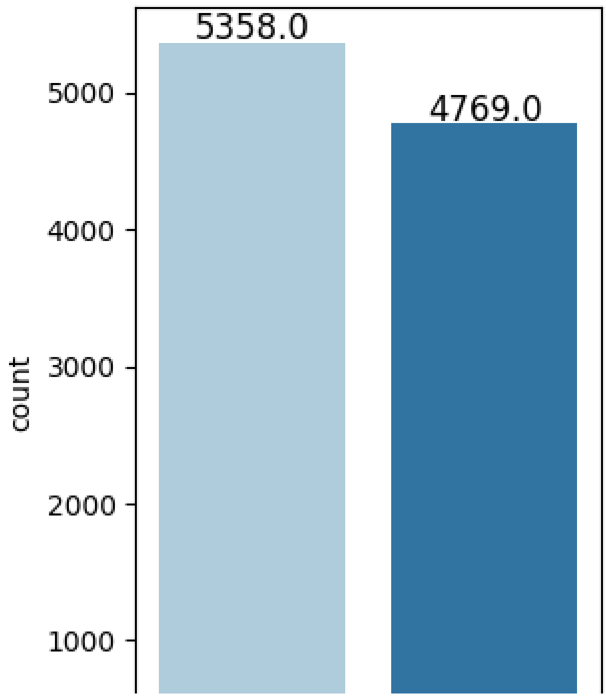
Contacts_Count_12_mon

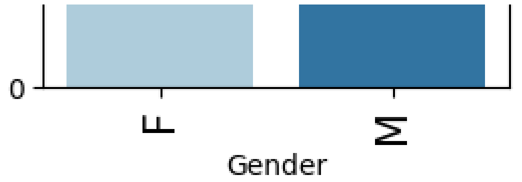
In [42]:
```
labeled_barplot(data, 'Contacts_Count_12_mon') ## Complete the code to create
```



Gender

In [43]:
```
labeled_barplot(data, 'Gender') ## Complete the code to create labeled_barplo
```



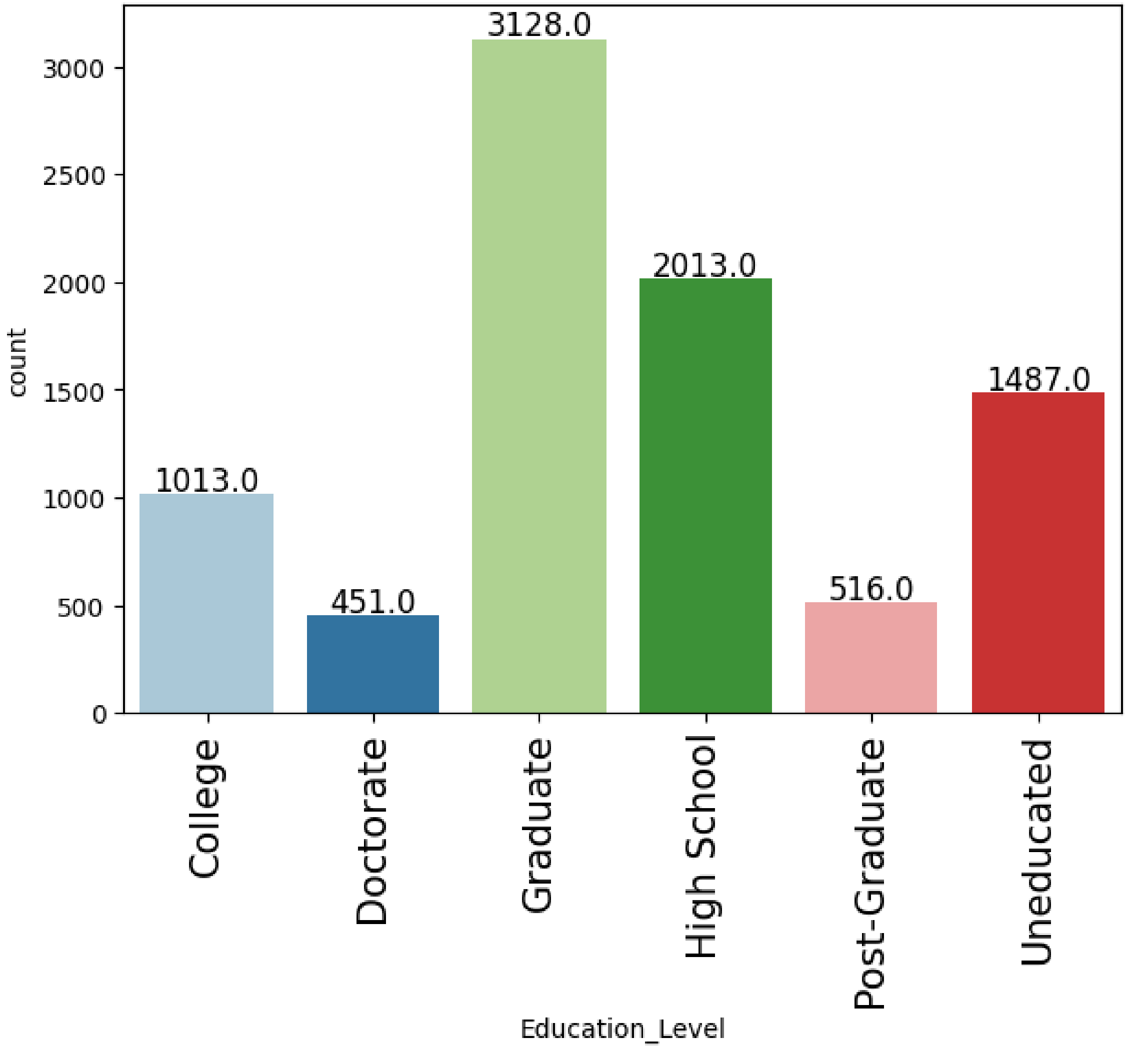


Let's see the distribution of the level of education of customers

```
Education_Level
```

In [44]:

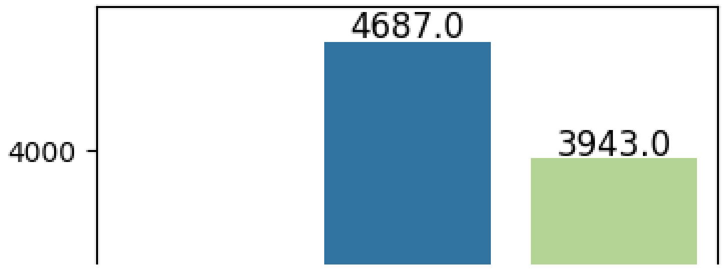
```
labeled_barplot(data, 'Education_Level') ## Complete the code to create label
```

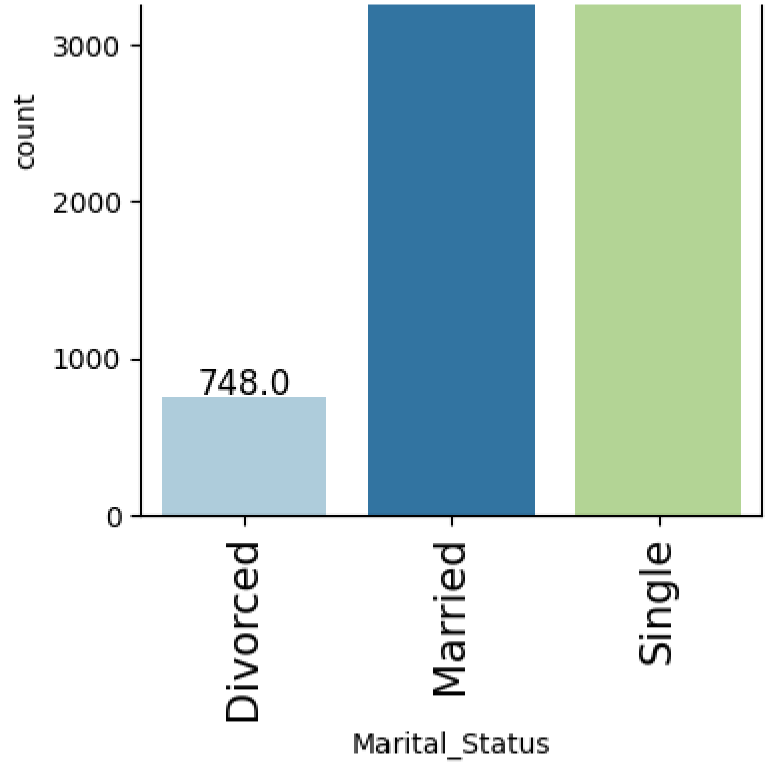



```
Marital_Status
```

In [45]:

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



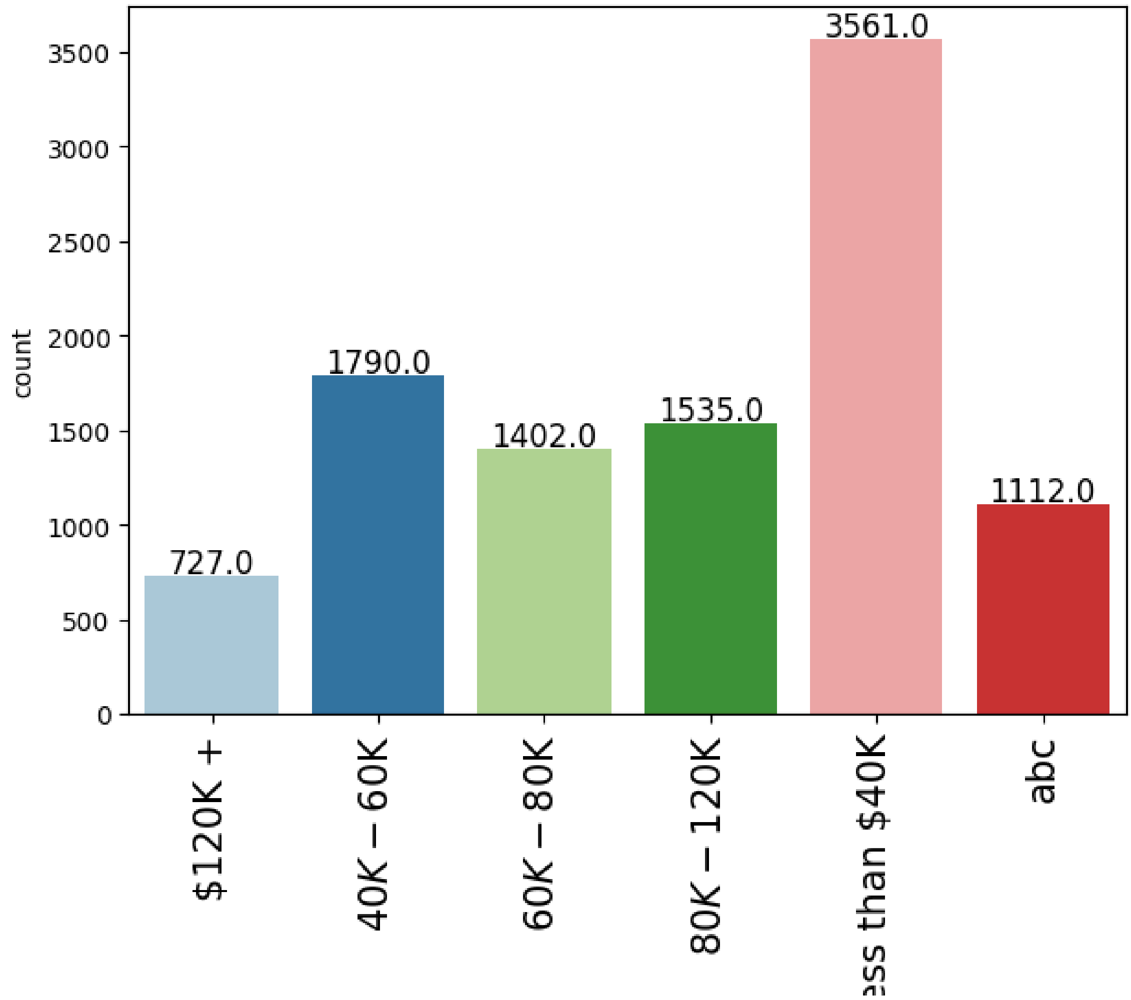


Let's see the distribution of the level of income of customers

`Income_Category`

In [46]:

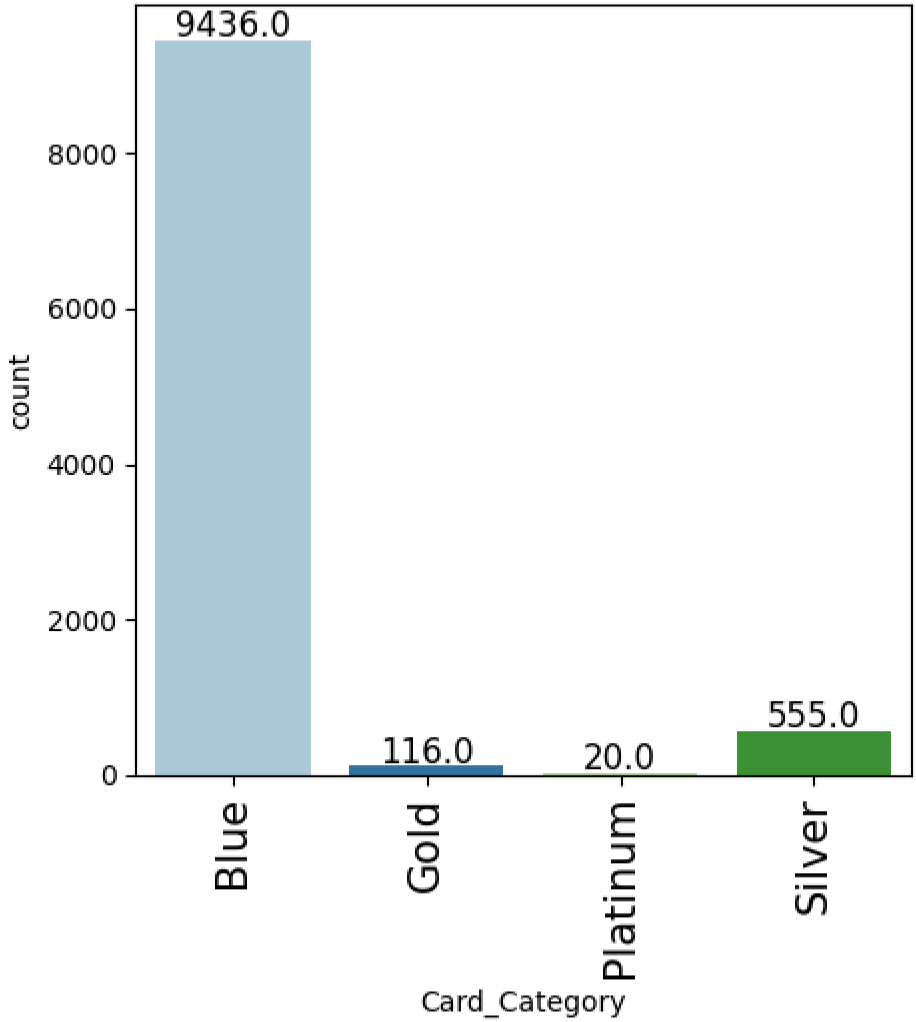
```
labeled_barplot(data, 'Income_Category') ## Complete the code to create label
```



Income_Category

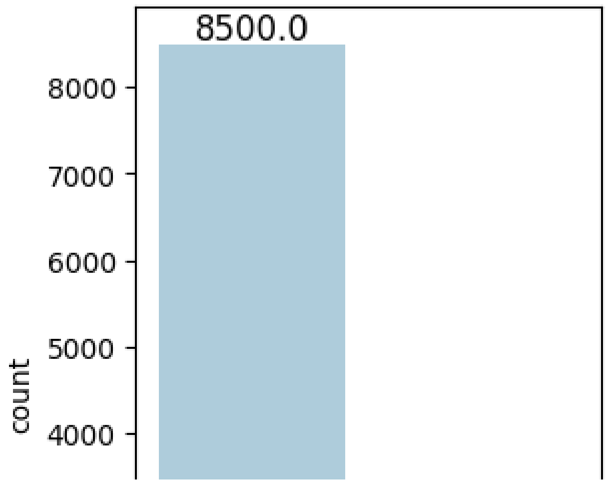
Card_Category

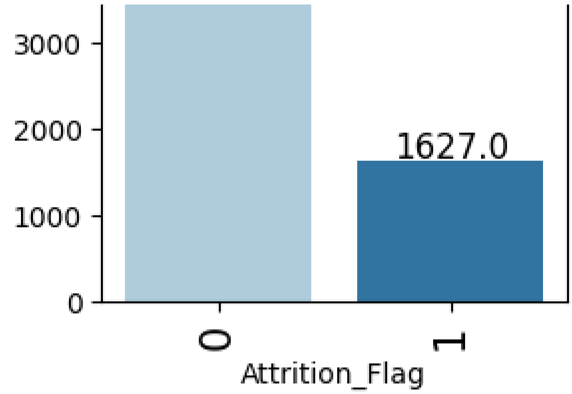
In [47]:
```
labeled_barplot(data, 'Card_Category') ## Complete the code to create labeled_
```



Attrition_Flag

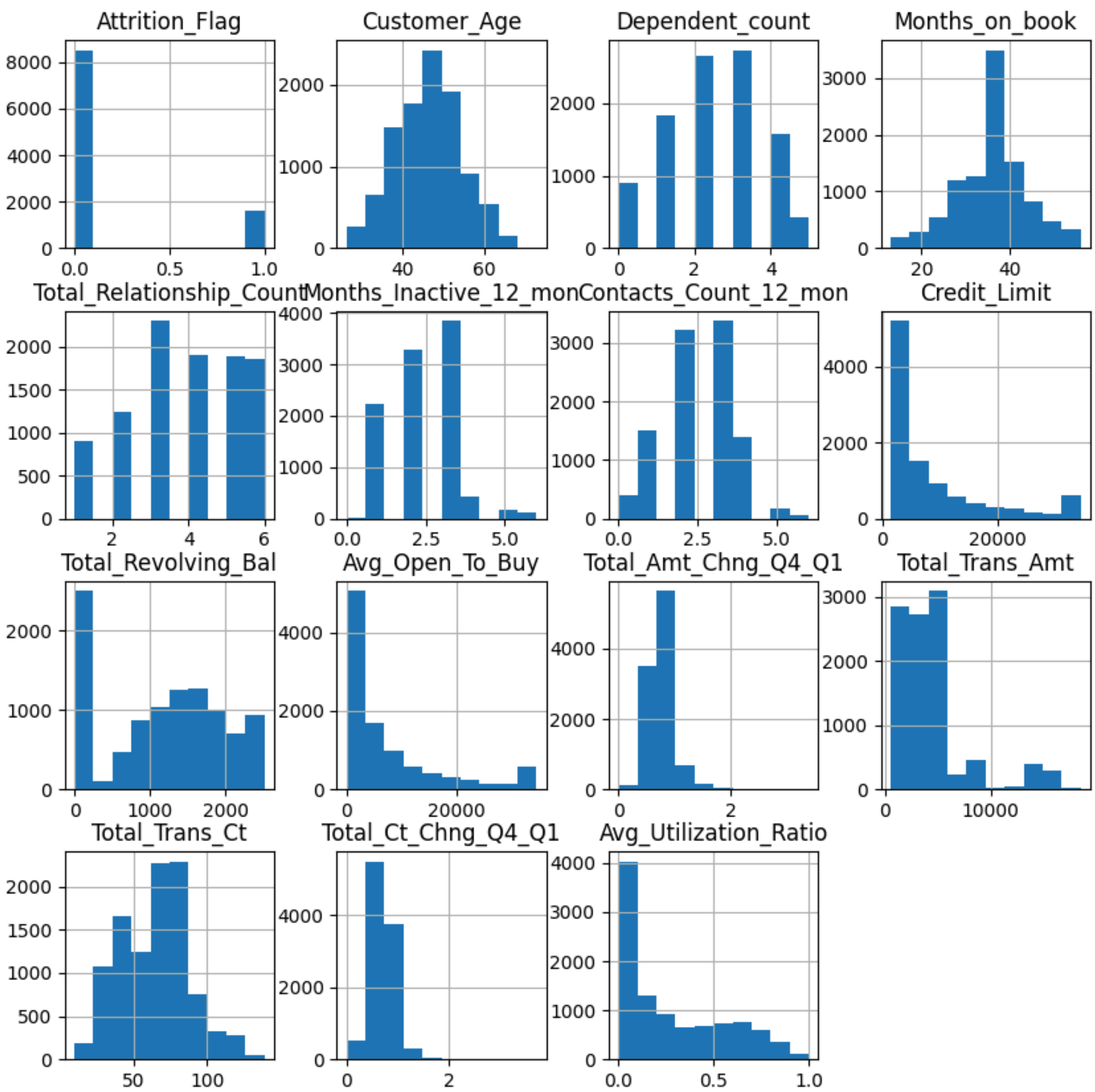
In [48]:
```
labeled_barplot(data, 'Attrition_Flag') ## Complete the code to create labele
```





In [49]:

```
# creating histograms
data.hist(figsize=(10, 10))
plt.show()
```



Bivariate Distributions

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

In [50]:

```
data['Gender'] = data['Gender'].astype('category')
data['Education_Level'] = data['Education_Level'].astype('category')
data['Marital_Status'] = data['Marital_Status'].astype('category')
data['Income_Category'] = data['Income_Category'].astype('category')
```

```
data['Card_Category'] = data['Card_Category'].astype('category')
```

Correlation Check

In [51]:
```
data7 = churn.copy()
```

In [52]:
```
data7.drop(["CLIENTNUM"], axis=1, inplace=True)
```

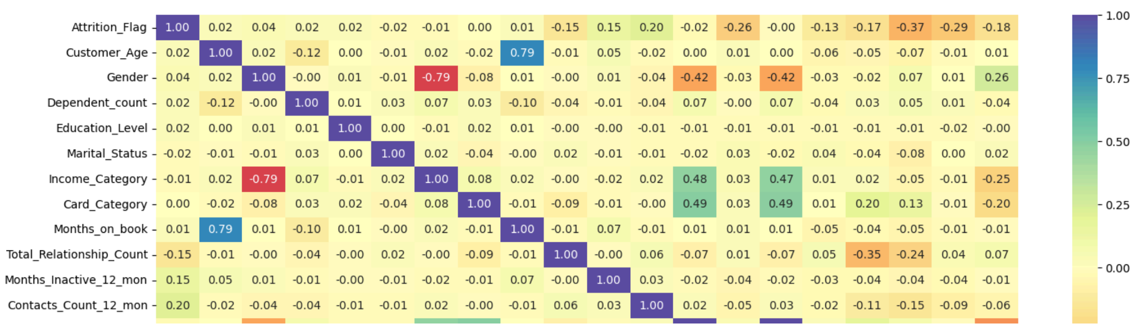
In [53]:
```
Attrition_Flag = {'Existing Customer' : 0, 'Attrited Customer' : 1}
data7['Attrition_Flag'] = data7['Attrition_Flag'].map(Attrition_Flag)

Gender = {'M' : 0, 'F' : 1}
data7['Gender'] = data7['Gender'].map(Gender)

Education_Level = {'Uneducated' : 0, 'High School' : 1, 'College' : 2, 'Gradu
data7['Education_Level'] = data7['Education_Level'].map(Education_Level)

Marital_Status = {'Single' : 0, 'Married' : 1, 'Divorced' : 2}
data7['Marital_Status'] = data7['Marital_Status'].map(Marital_Status)

Income_Category = {'abc' : 0, 'Less than $40K' : 1, '$40K - $60K' : 2, '$60K
data7['Income_Category'] = data7['Income_Category'].map(Income_Category)

Card_Category = {'Blue' : 0, 'Silver' : 1, 'Gold' : 2, 'Platinum' : 3}
data7['Card_Category'] = data7['Card_Category'].map(Card_Category)
```

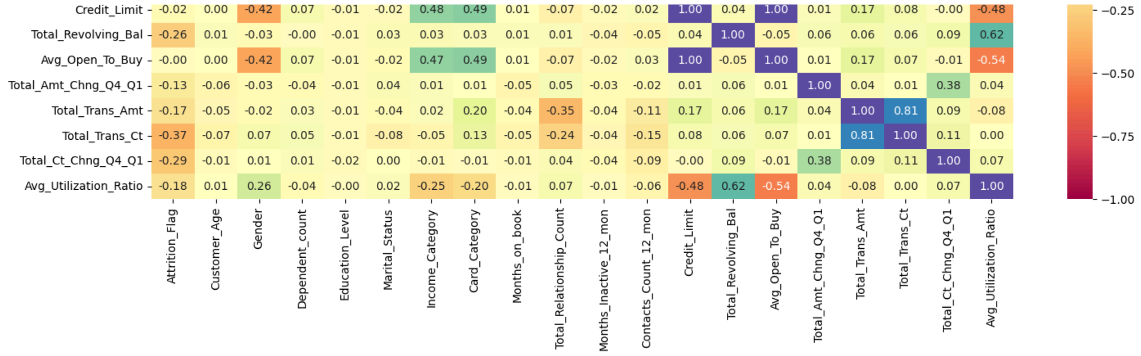
In [54]:
```
data7.head()
```

Out[54]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_
0	0	45	0	3	1.000	
1	0	49	1	5	3.000	
2	0	51	0	3	3.000	
3	0	40	1	4	1.000	
4	0	40	0	3	0.000	

In [55]:
```
plt.figure(figsize=(17, 8))
sns.heatmap(data7.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spect
plt.show()
```

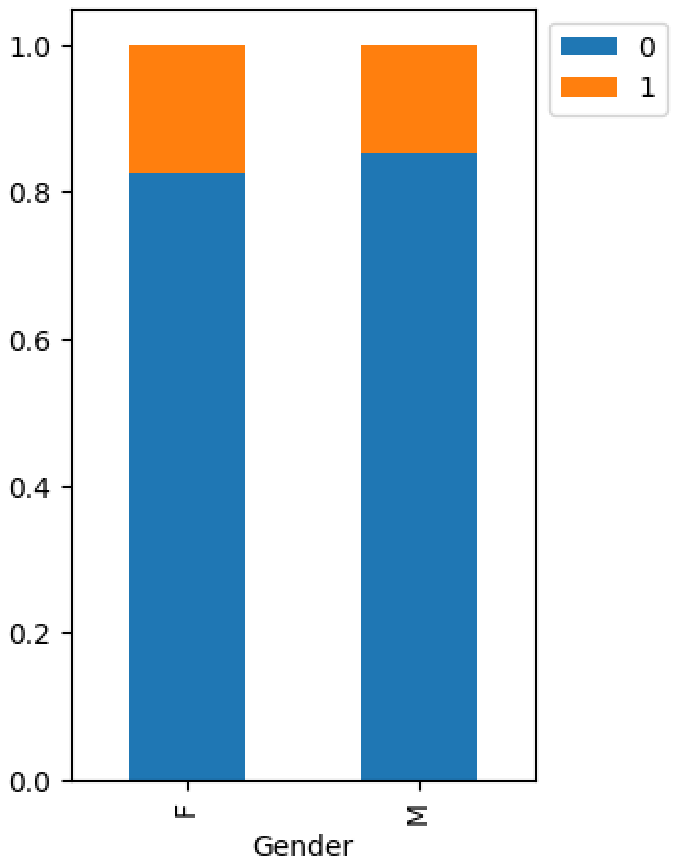




Attrition_Flag vs Gender

In [56]:
```python
stacked_barplot(data, "Gender", "Attrition_Flag")
```

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

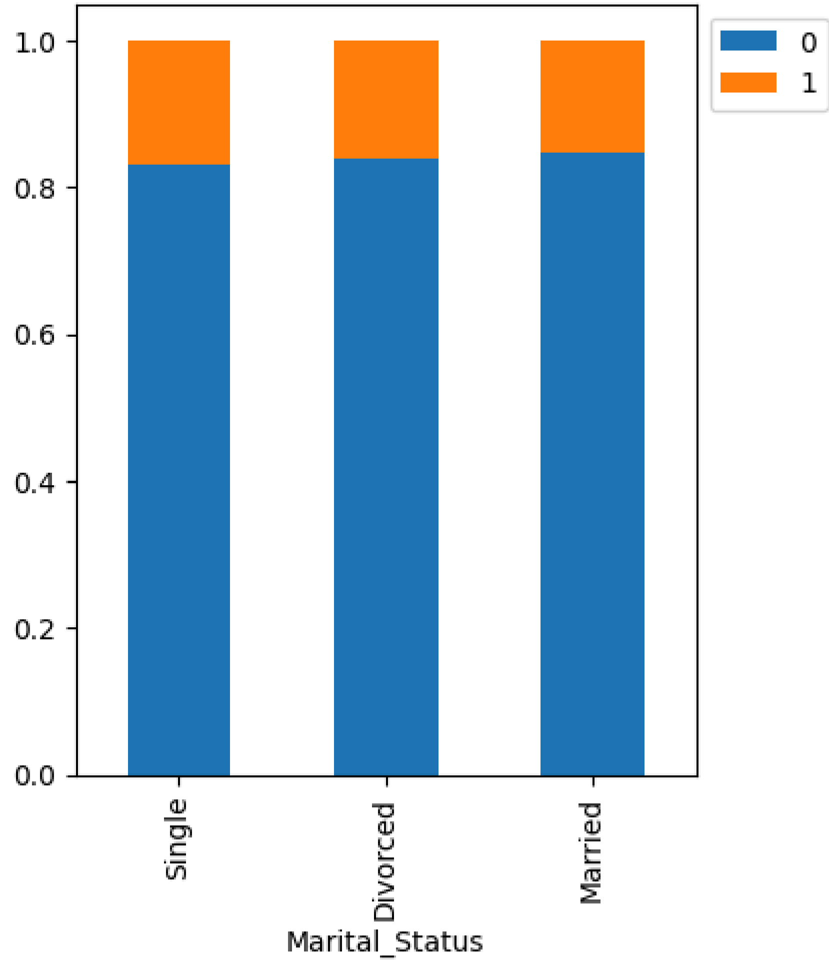


Attrition_Flag vs Marital_Status

In [57]:
```python
stacked_barplot(data,"Marital_Status", "Attrition_Flag") ## Complete the code
```

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

```
Divorced          627   121   748
--------------------------------------------------------------------------------
-----------------------------------------
```

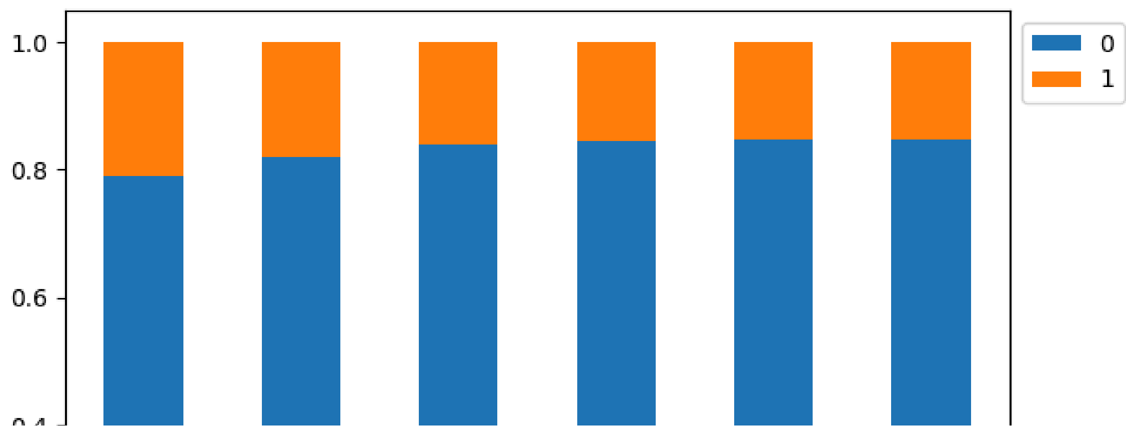


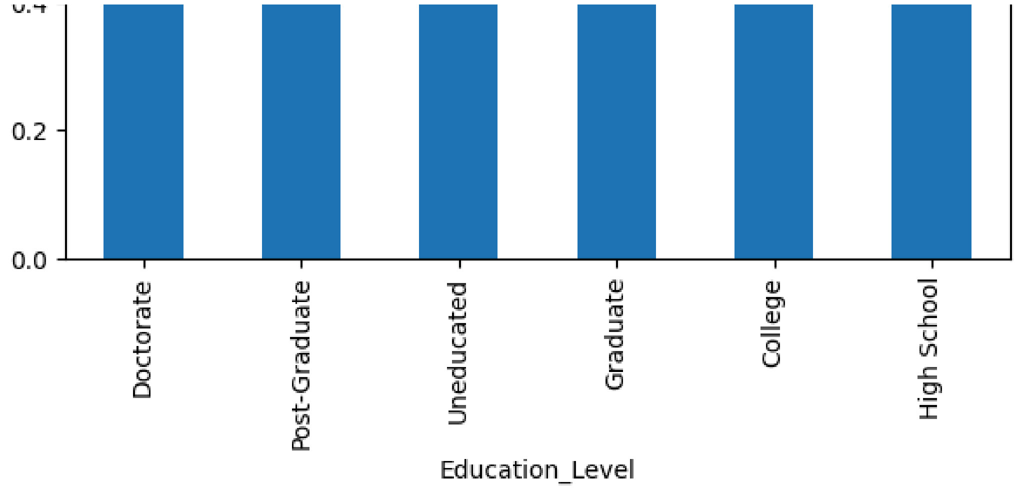
Attrition_Flag vs Education_Level

In [58]:

```
stacked_barplot(data,"Education_Level", "Attrition_Flag") ## Complete the cod
```

```
Attrition_Flag      0     1   All
Education_Level
All              7237  1371  8608
Graduate         2641   487  3128
High School      1707   306  2013
Uneducated       1250   237  1487
College           859   154  1013
Doctorate         356    95   451
Post-Graduate     424    92   516
--------------------------------------------------------------------------------
-----------------------------------------
```



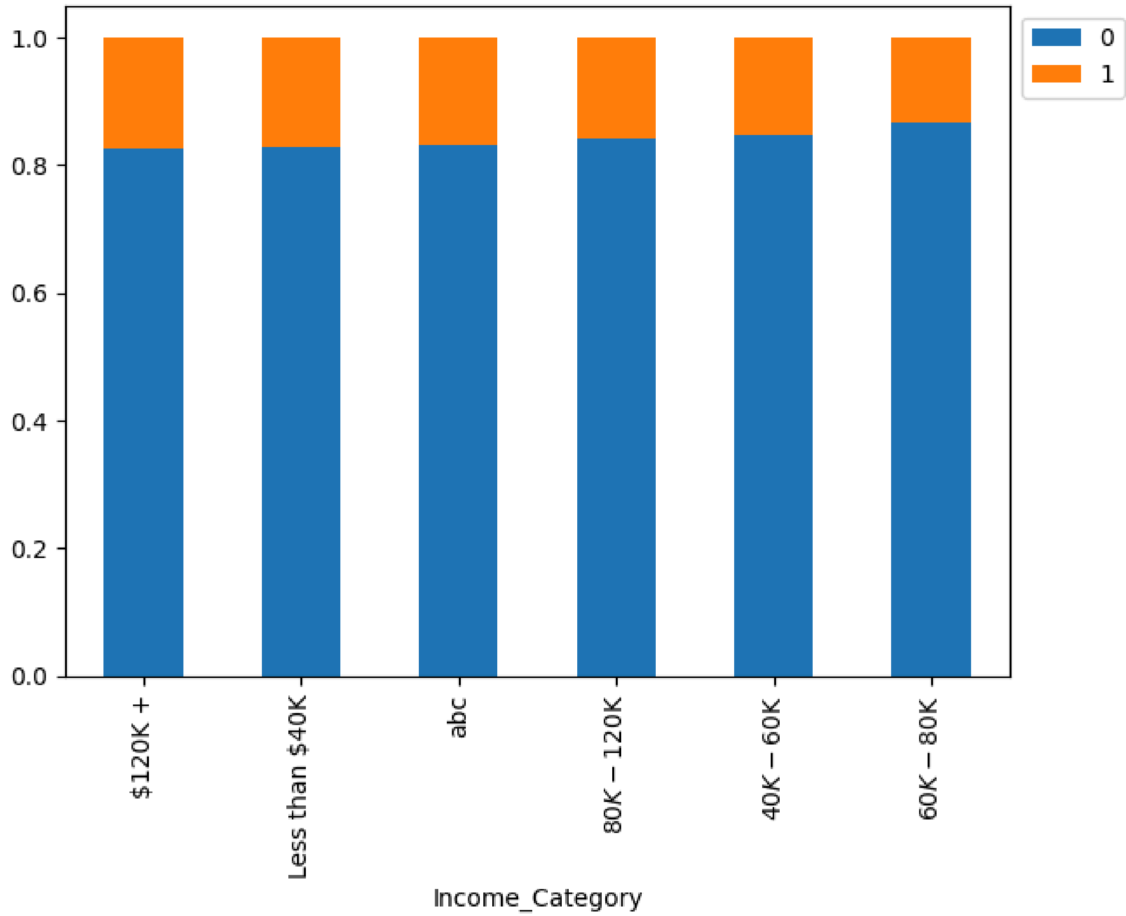


Attrition_Flag vs Income_Category

In [59]:

```
stacked_barplot(data,"Income_Category", "Attrition_Flag") ## Complete the cod
```

```
Attrition_Flag      0     1    All
Income_Category
All              8500  1627  10127
Less than $40K   2949   612   3561
$40K - $60K      1519   271   1790
$80K - $120K     1293   242   1535
$60K - $80K      1213   189   1402
abc               925   187   1112
$120K +           601   126    727
------------------------------------------------------------------------------------------------------------------------
```

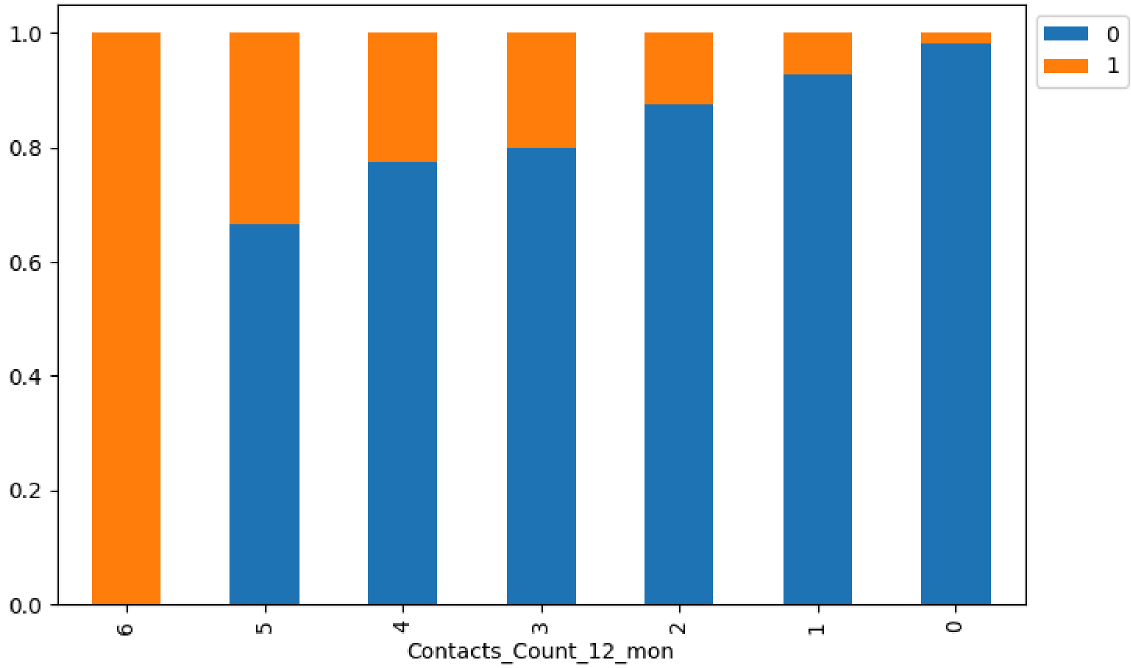


Attrition_Flag vs Contacts_Count_12_mon

In [60]:

```
stacked_barplot(data,"Contacts_Count_12_mon", "Attrition_Flag") ## Complete t
```

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



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)

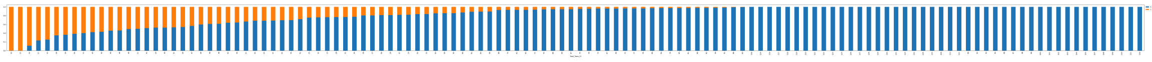
Attrition_Flag vs Months_Inactive_12_mon

In [61]:

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

```
Attrition_Flag     0     1    All
Total_Trans_Ct
All             8500  1627  10127
43                62    85    147
42                57    75    132
40                67    69    136
44                58    69    127
...              ...   ...    ...
109               22     0     22
110               25     0     25
111               22     0     22
112               24     0     24
105               32     0     32

[127 rows x 3 columns]
------------------------------------------------------------------------------
```

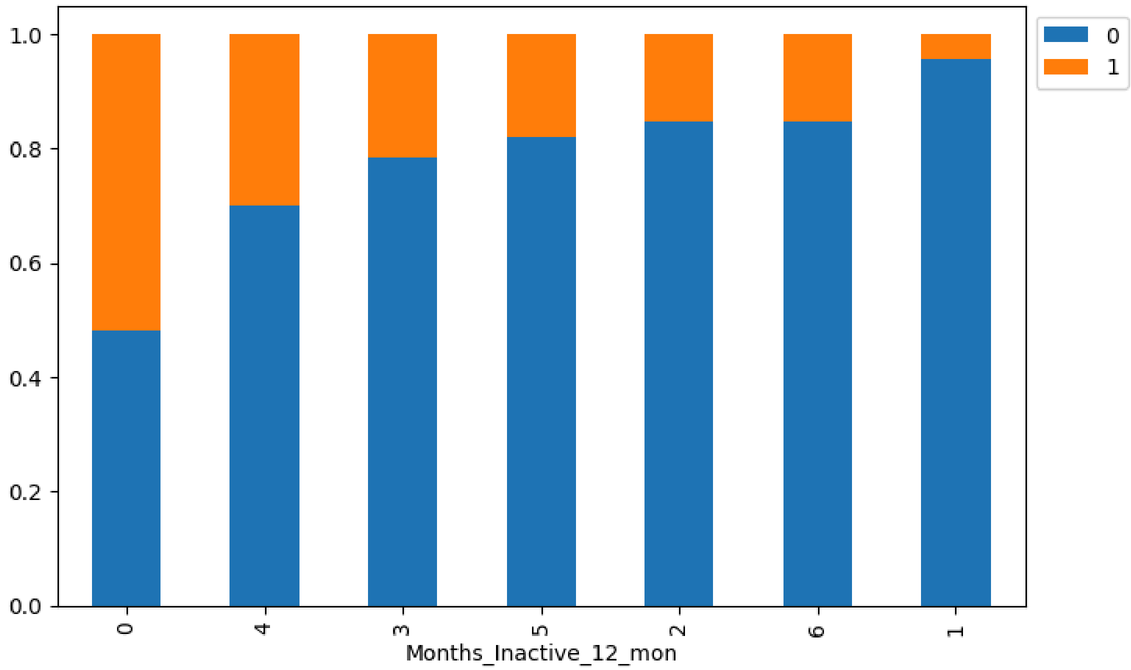
```
----------------------------------------
```



In [55]:

```python
stacked_barplot(data,"Months_Inactive_12_mon", "Attrition_Flag") ## Complete
```

```
Attrition_Flag             0     1    All
Months_Inactive_12_mon
All                     8500  1627  10127
3                       3020   826   3846
2                       2777   505   3282
4                        305   130    435
1                       2133   100   2233
5                        146    32    178
6                        105    19    124
0                         14    15     29
------------------------------------------------------------------------------------------------------------------------
----------------------------------------
```

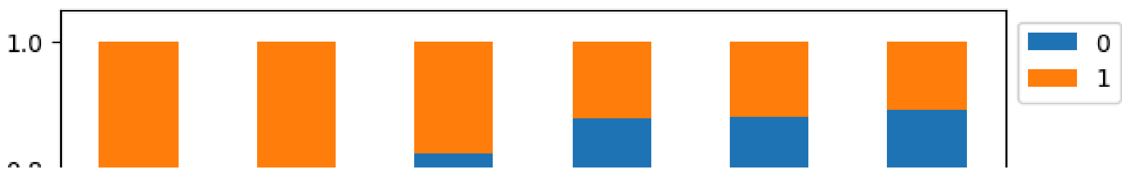


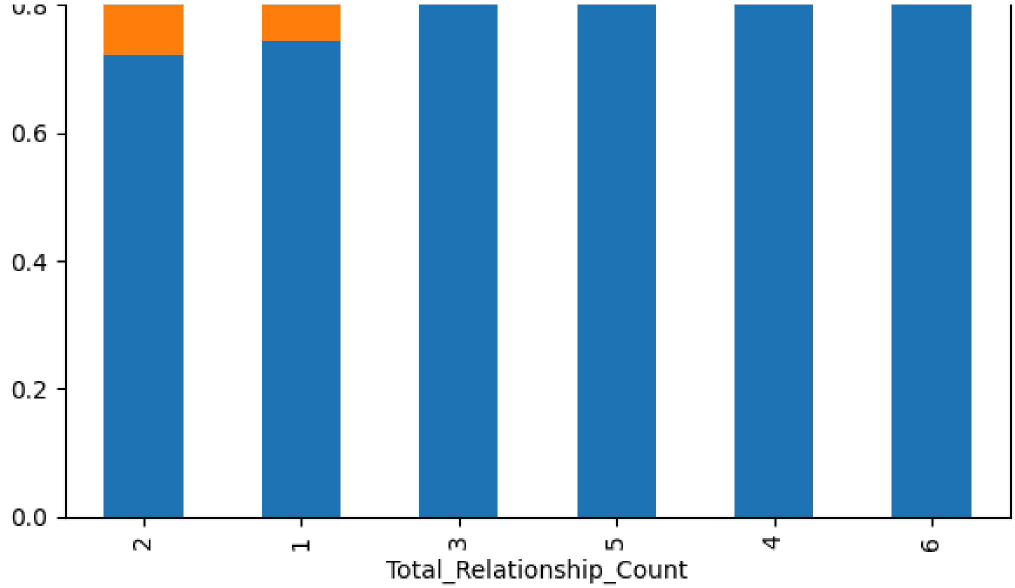
Attrition_Flag vs Total_Relationship_Count

In [56]:

```python
stacked_barplot(data,"Total_Relationship_Count", "Attrition_Flag") ## Complet
```

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



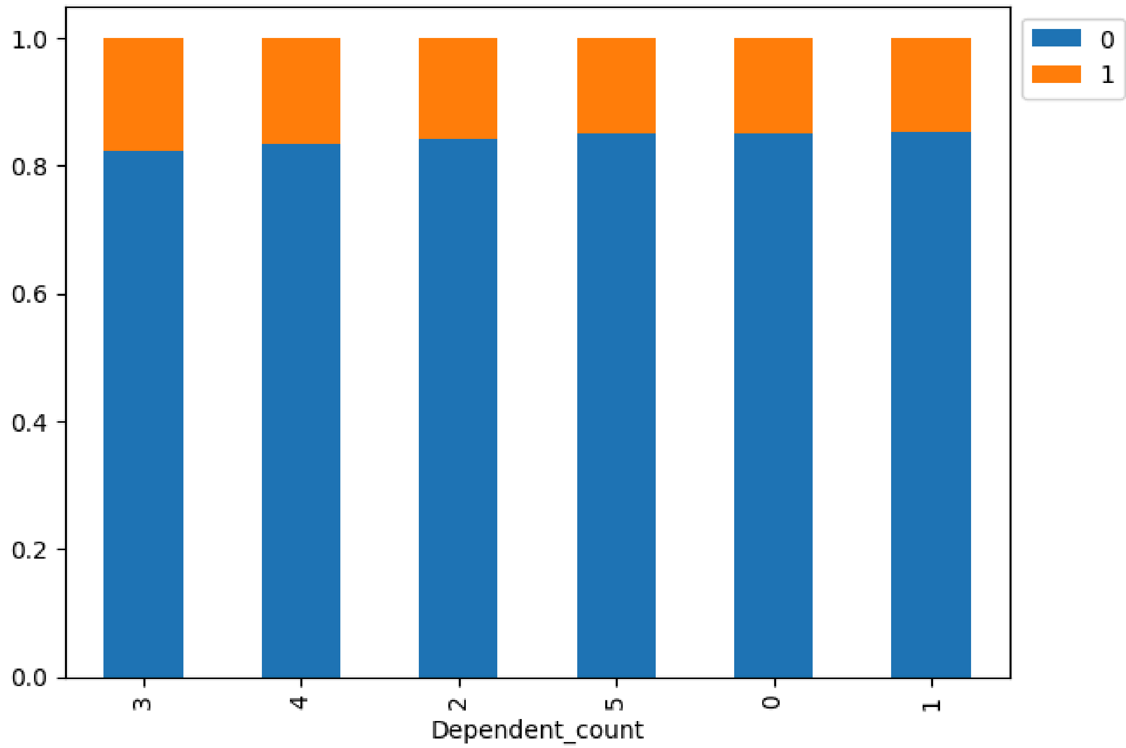


Attrition_Flag vs Dependent_count

In [57]:

```
stacked_barplot(data,"Dependent_count", "Attrition_Flag") ## Complete the cod
```

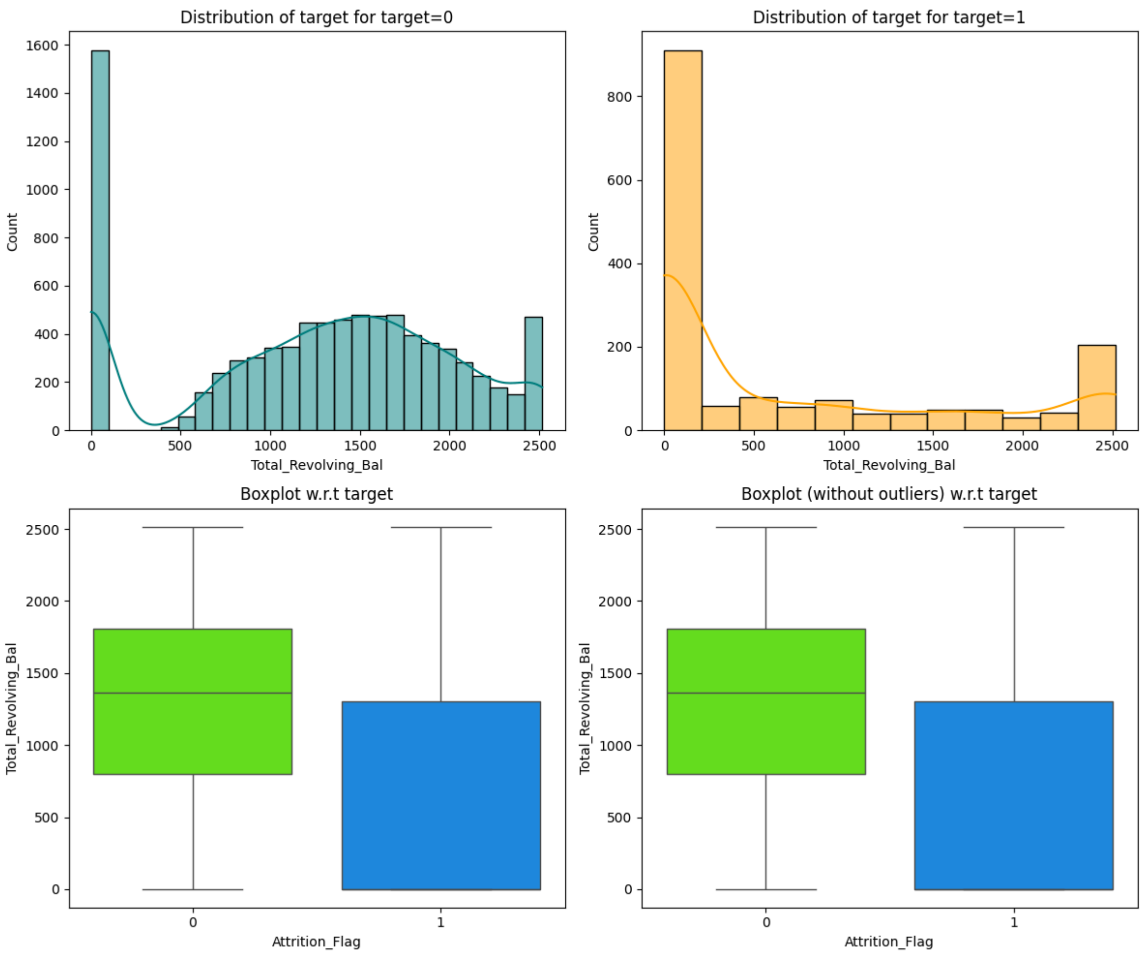
```
Attrition_Flag      0     1    All
Dependent_count
All              8500  1627  10127
3                2250   482   2732
2                2238   417   2655
1                1569   269   1838
4                1314   260   1574
0                 769   135    904
5                 360    64    424
----------------------------------------------------------------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



Total_Revolving_Bal vs Attrition_Flag

In [58]:

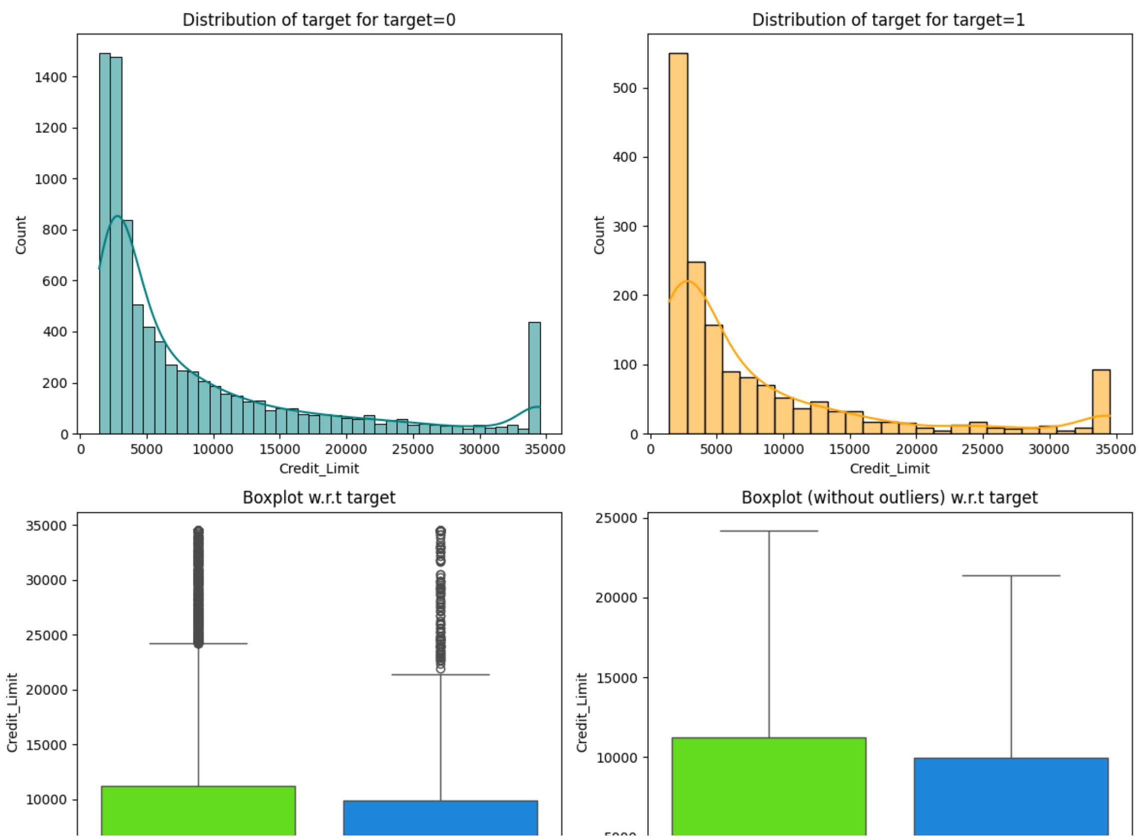
```
distribution_plot_wrt_target(data, "Total_Revolving_Bal", "Attrition_Flag")
```



Attrition_Flag vs Credit_Limit

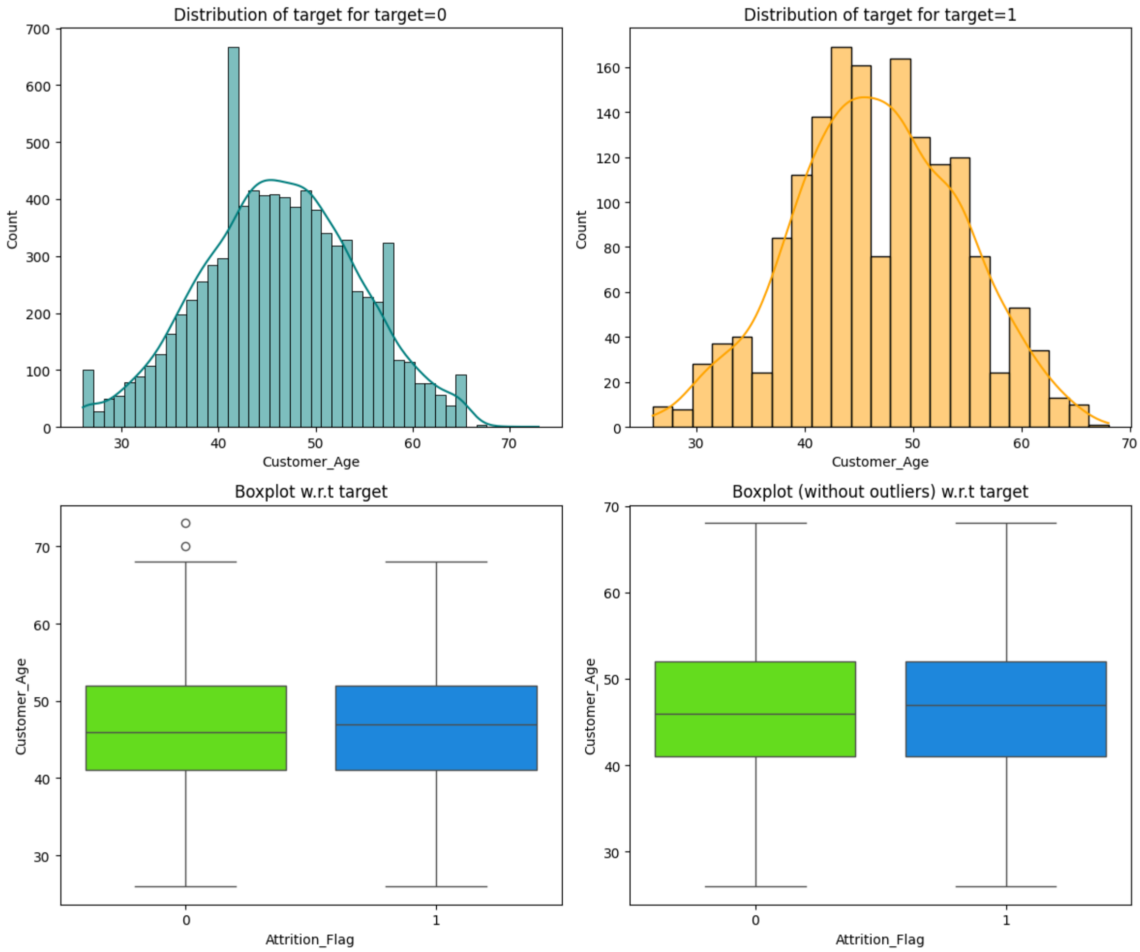
In [59]:

```
distribution_plot_wrt_target(data, "Credit_Limit", "Attrition_Flag") ## Compl
```



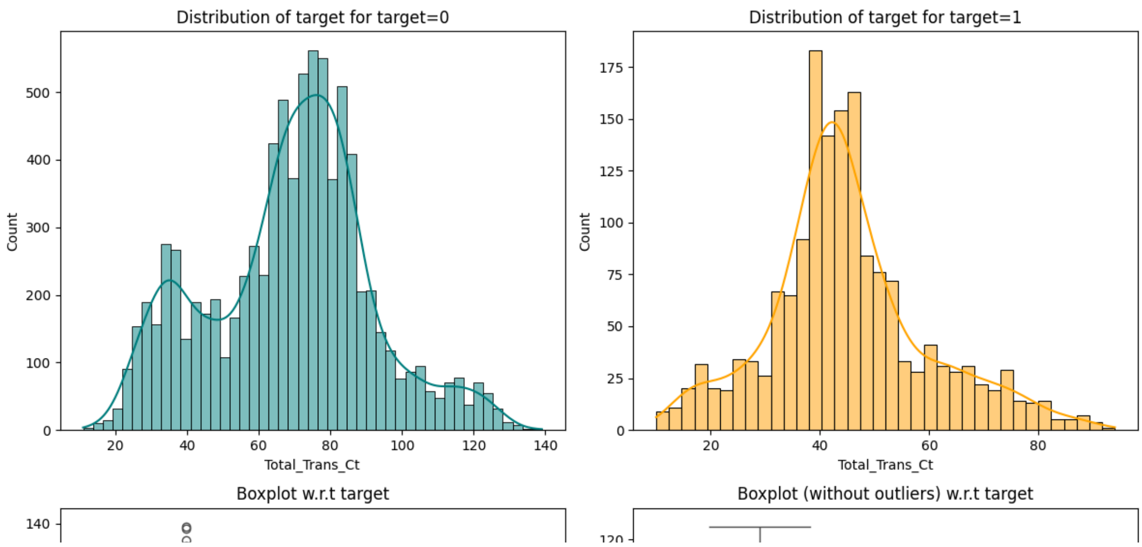
Attrition_Flag vs Customer_Age

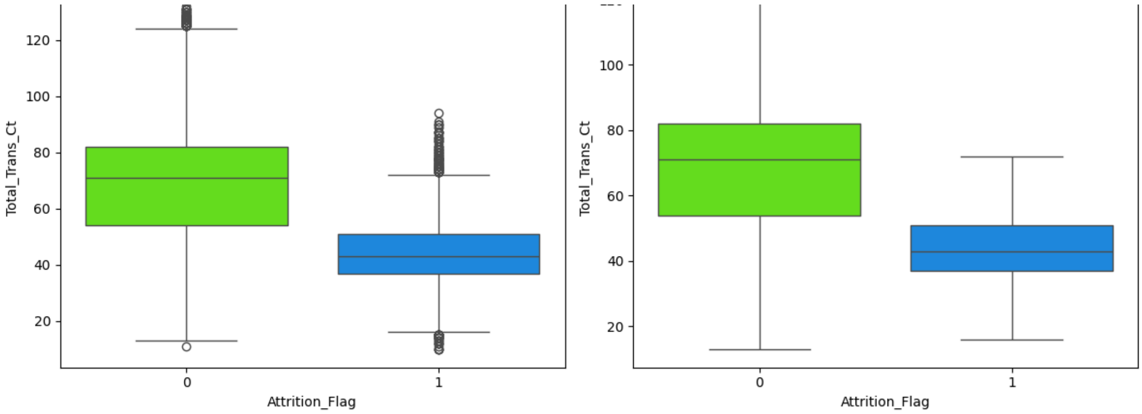
In [60]:
```
distribution_plot_wrt_target(data, "Customer_Age", "Attrition_Flag") ## Compl
```



Total_Trans_Ct vs Attrition_Flag

In [61]:
```
distribution_plot_wrt_target(data, "Total_Trans_Ct", "Attrition_Flag") ## Com
```

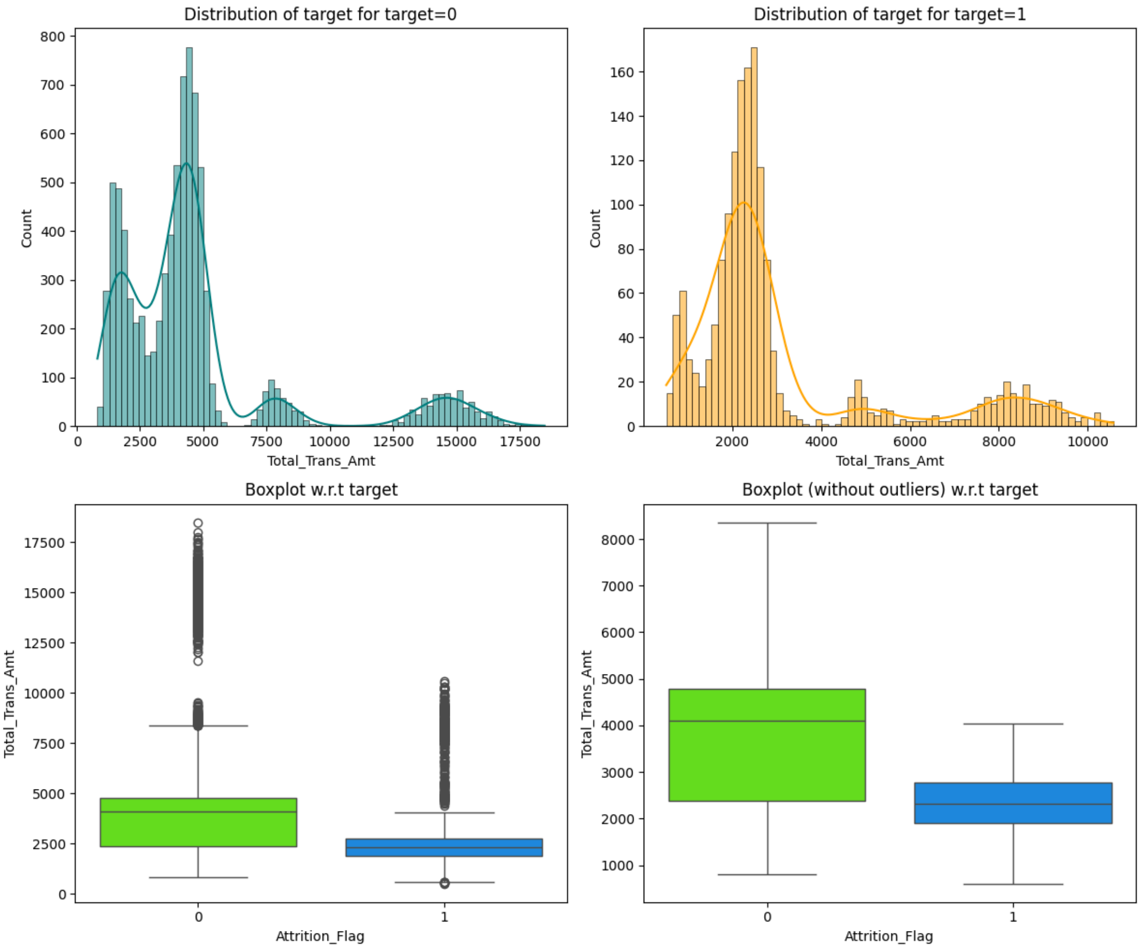




Total_Trans_Amt vs Attrition_Flag

In [62]:

```
distribution_plot_wrt_target(data, "Total_Trans_Amt", "Attrition_Flag") ## Co
```

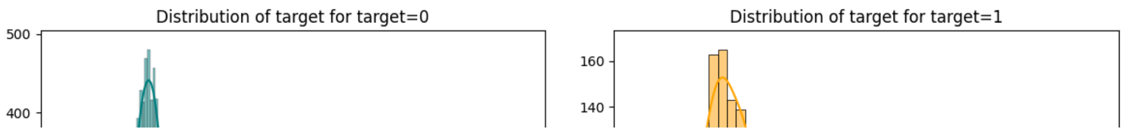


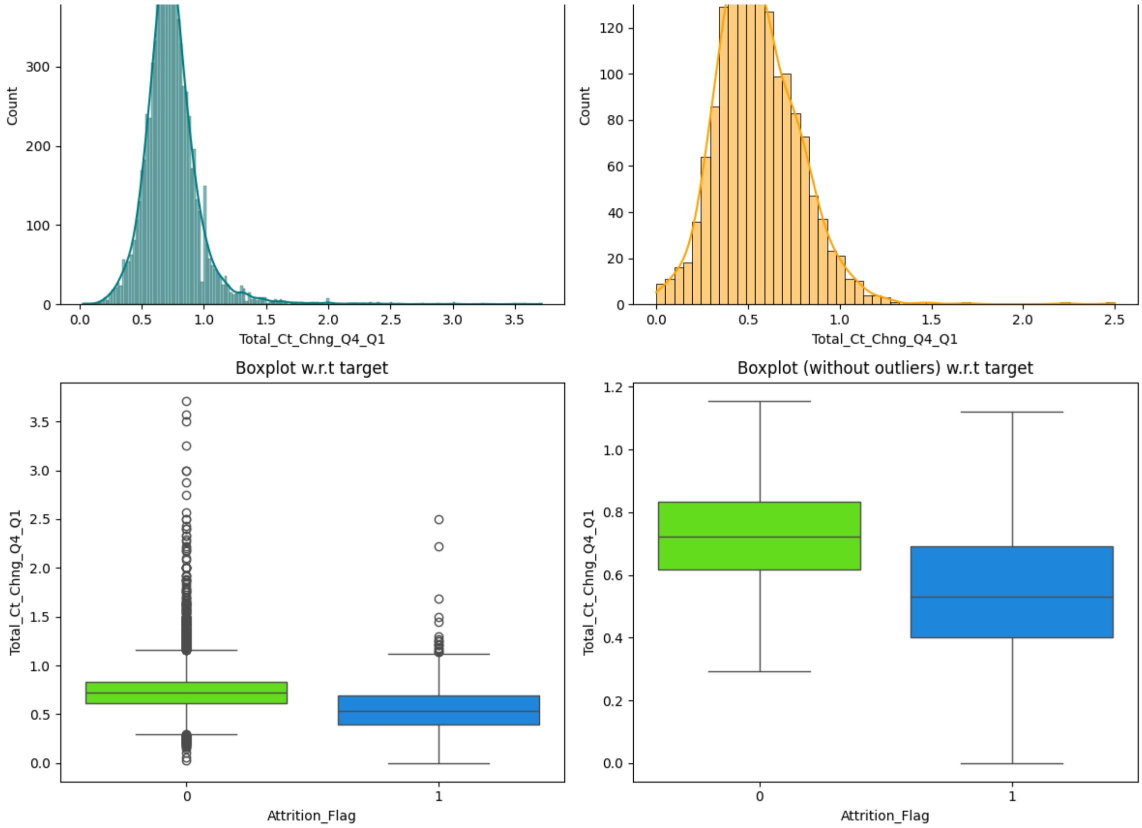
Let's see the change in transaction amount between Q4 and Q1 (total_ct_change_Q4_Q1) vary by the customer's account status (Attrition_Flag)

Total_Ct_Chng_Q4_Q1 vs Attrition_Flag

In [63]:

```
distribution_plot_wrt_target(data, "Total_Ct_Chng_Q4_Q1", "Attrition_Flag") #
```

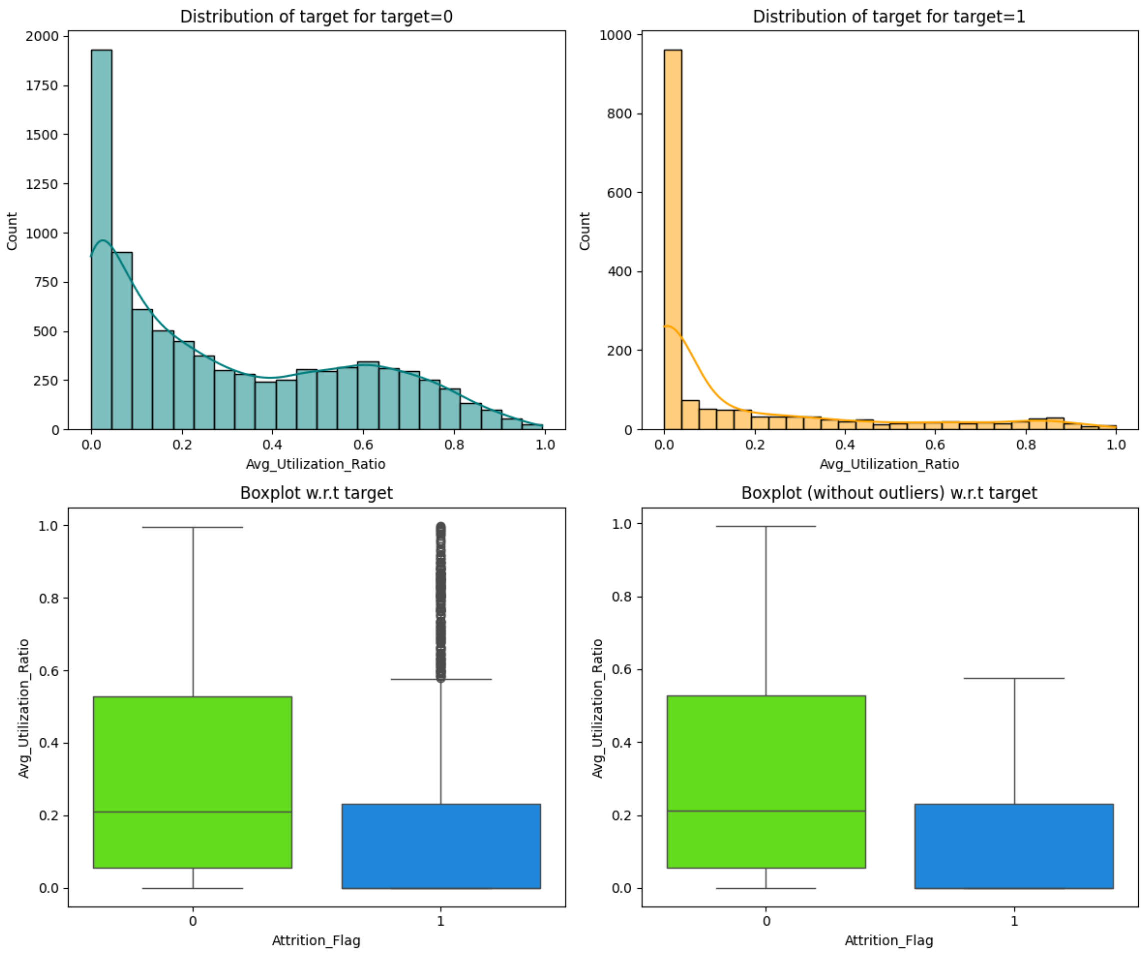




Avg_Utilization_Ratio vs Attrition_Flag

In [64]:

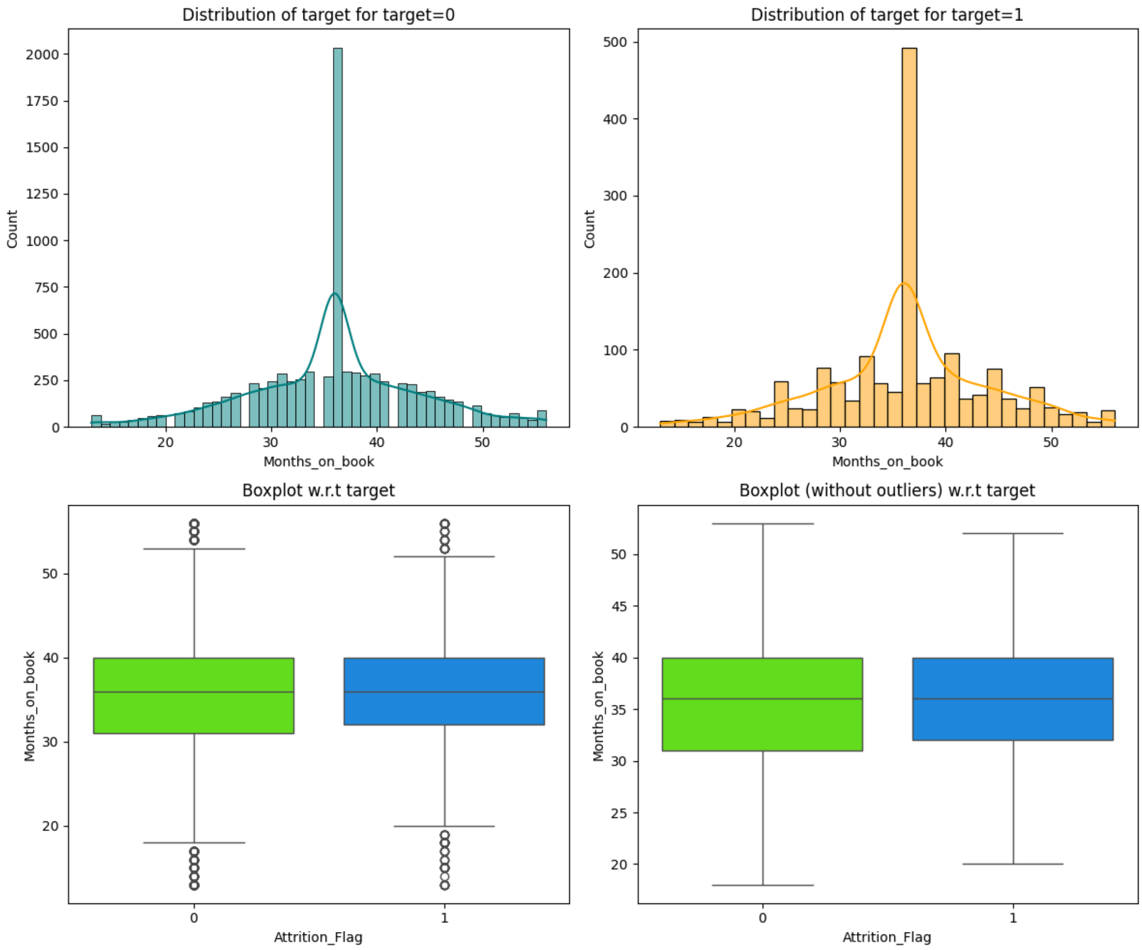
```
distribution_plot_wrt_target(data, "Avg_Utilization_Ratio", "Attrition_Flag")
```



Attrition_Flag vs Months_on_book

In [65]:

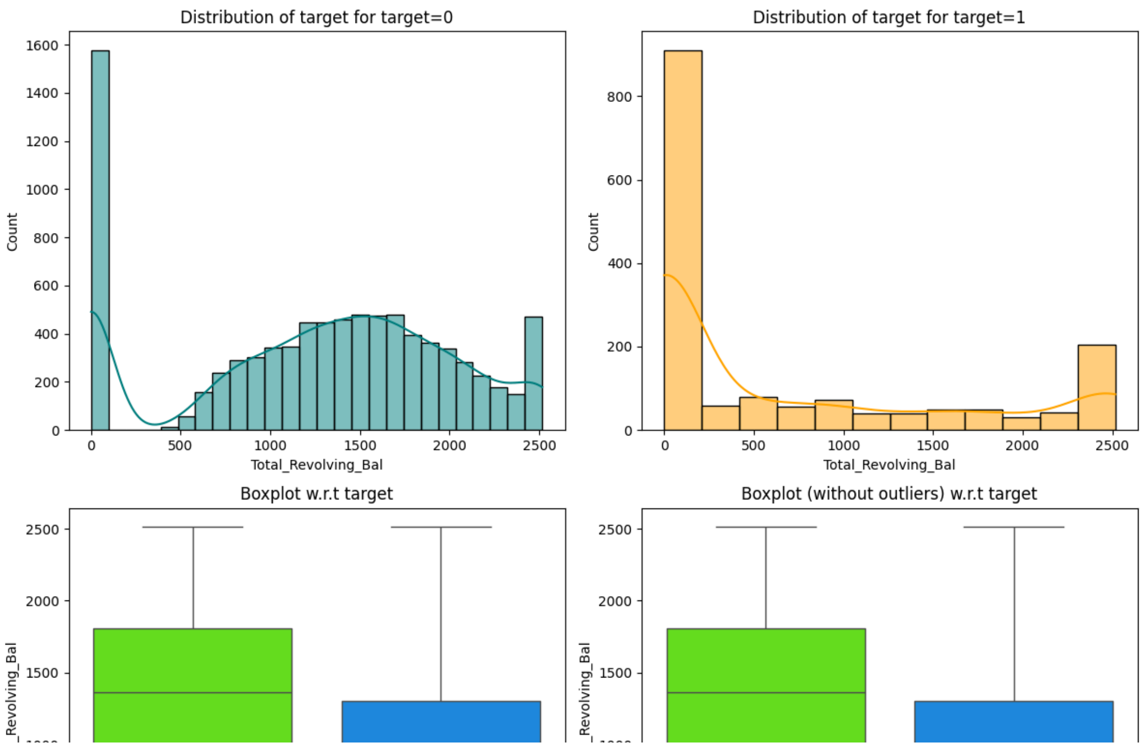
```
distribution_plot_wrt_target(data, "Months_on_book", "Attrition_Flag") ## Com
```

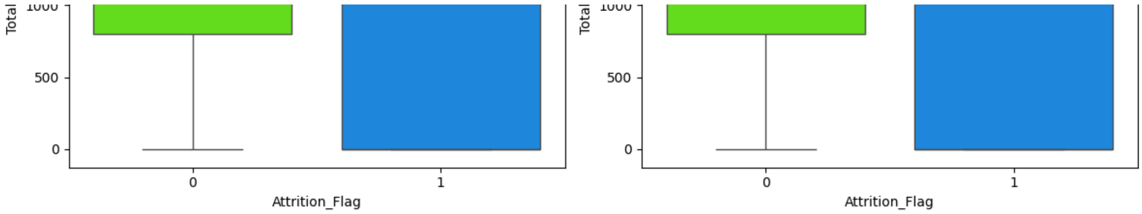


Attrition_Flag vs Total_Revolving_Bal

In [66]:

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

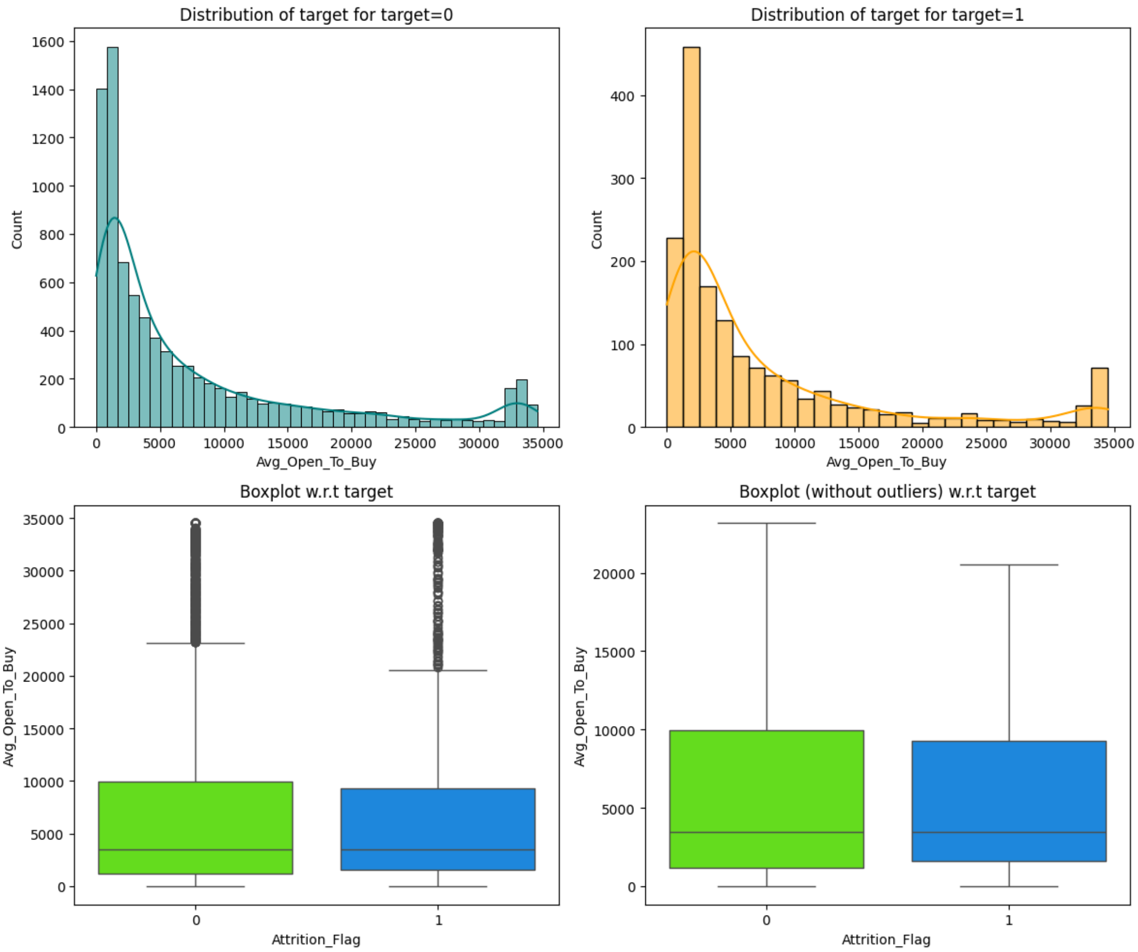




Attrition_Flag vs Avg_Open_To_Buy

In [67]:

```
distribution_plot_wrt_target(data, "Avg_Open_To_Buy", "Attrition_Flag") ## Co
```



Data Preprocessing

In [68]:

```
data8 = data7.copy()
```

In [69]:

```
Q1 = data8.quantile(0.25)  # To find the 25th percentile
Q3 = data8.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 bou
lower = (Q1 - 1.5 * IQR)
upper = (Q3 + 1.5 * IQR)
```

In [70]:

```
# checking the % outliers
```

```
((data8.select_dtypes(include=["float64", "int64"]) < lower) | (data8.select_
```

Out[70]:

```
Attrition_Flag             16.066
Customer_Age                0.020
Gender                      0.000
Dependent_count             0.000
Education_Level             0.000
Marital_Status              0.000
Income_Category             0.000
Card_Category               6.823
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
Total_Ct_Chng_Q4_Q1         3.891
Avg_Utilization_Ratio       0.000
dtype: float64
```

Train-Test Split

In [71]:

```
data1 = data.copy()
```

In [72]:

```
data1["Income_Category"].replace("NaN", np.nan, inplace=True) ### complete t
```

In [73]:

```
data1["Income_Category"].value_counts()
```

Out[73]:

```
Income_Category
Less than $40K    3561
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
abc               1112
$120K +            727
Name: count, dtype: int64
```

In [74]:

```
# Dividing train data into X and y

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

In [75]:

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

X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.2, random

X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0

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

```
(6075, 19) (2026, 19) (2026, 19)
```

Missing value imputation

In [76]:
```python
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 6075 entries, 800 to 4035
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   Customer_Age              6075 non-null   int64   
 1   Gender                    6075 non-null   category
 2   Dependent_count           6075 non-null   int64   
 3   Education_Level           5147 non-null   category
 4   Marital_Status            5618 non-null   category
 5   Income_Category           6075 non-null   category
 6   Card_Category             6075 non-null   category
 7   Months_on_book            6075 non-null   int64   
 8   Total_Relationship_Count  6075 non-null   int64   
 9   Months_Inactive_12_mon    6075 non-null   int64   
 10  Contacts_Count_12_mon     6075 non-null   int64   
 11  Credit_Limit              6075 non-null   float64 
 12  Total_Revolving_Bal       6075 non-null   int64   
 13  Avg_Open_To_Buy           6075 non-null   float64 
 14  Total_Amt_Chng_Q4_Q1      6075 non-null   float64 
 15  Total_Trans_Amt           6075 non-null   int64   
 16  Total_Trans_Ct            6075 non-null   int64   
 17  Total_Ct_Chng_Q4_Q1       6075 non-null   float64 
 18  Avg_Utilization_Ratio     6075 non-null   float64 
dtypes: category(5), float64(5), int64(9)
memory usage: 742.5 KB
```

In [77]:
```python
reqd_col_for_impute = ["Education_Level", "Marital_Status"]
```

In [78]:
```python
# creating an instace of the imputer to be used
imputer = SimpleImputer(strategy="most_frequent")
```

In [79]:
```python
# Fit and transform the train data
X_train[reqd_col_for_impute] = imputer.fit_transform(X_train[reqd_col_for_imp

# Transform the validation data
X_val[reqd_col_for_impute]  =  imputer.transform(X_val[reqd_col_for_impute])

# Transform the test data
X_test[reqd_col_for_impute] = imputer.transform(X_test[reqd_col_for_impute])
```

In [80]:
```python
# 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 [81]:

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

```
Gender
F    3193
M    2882
Name: count, dtype: int64
******************************
Education_Level
Graduate         2782
High School      1228
Uneducated        881
College           618
Post-Graduate     312
Doctorate         254
Name: count, dtype: int64
******************************
Marital_Status
Married     3276
Single      2369
Divorced     430
Name: count, dtype: int64
******************************
Income_Category
Less than $40K    2129
$40K - $60K       1059
$80K - $120K       953
$60K - $80K        831
abc                654
$120K +            449
Name: count, dtype: int64
******************************
Card_Category
Blue        5655
Silver       339
Gold          69
Platinum      12
Name: count, dtype: int64
******************************
```

In [82]:

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

```
Gender
F    1095
M     931
Name: count, dtype: int64
******************************
Education_Level
Graduate         917
High School      404
Uneducated       306
College          199
Post-Graduate    101
Doctorate         99
Name: count, dtype: int64
******************************
Marital_Status
Married     1100
Single       770
Divorced     156
Name: count, dtype: int64
******************************
Income_Category
Less than $40K     736
```

```
Less than $40K      736
$40K - $60K         361
$80K - $120K        293
$60K - $80K         279
abc                 221
$120K +             136
Name: count, dtype: int64
******************************
Card_Category
Blue        1905
Silver        97
Gold          21
Platinum       3
Name: count, dtype: int64
******************************
```

In [83]:

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

```
Gender
F    3193
M    2882
Name: count, dtype: int64
******************************
Education_Level
Graduate         2782
High School      1228
Uneducated        881
College           618
Post-Graduate     312
Doctorate         254
Name: count, dtype: int64
******************************
Marital_Status
Married     3276
Single      2369
Divorced     430
Name: count, dtype: int64
******************************
Income_Category
Less than $40K     2129
$40K - $60K        1059
$80K - $120K        953
$60K - $80K         831
abc                 654
$120K +             449
Name: count, dtype: int64
******************************
Card_Category
Blue        5655
Silver       339
Gold          69
Platinum      12
Name: count, dtype: int64
******************************
```

Encoding categorical variables

In [84]:

```
X_train = pd.get_dummies(X_train, drop_first=True)
X_val = pd.get_dummies(X_val, drop_first=True)  ## Complete the code to imput
X_test = pd.get_dummies(X_test, drop_first=True)  ## Complete the code to imp
```

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

(6075, 30) (2026, 30) (2026, 30)

In [85]:

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

Out[85]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	M
800	40	2	21	6	
498	44	1	34	6	
4356	48	4	36	5	
407	41	2	36	6	
8728	46	4	36	2	

- After encoding there are 29 columns.

Creating Bins

In [86]:

```
data1["Credit_Limit"] = pd.qcut(
    data1["Credit_Limit"],
    q=[0, 0.25, 0.5, 1],
    labels=["Low_Credit_Limit", "Medium_Credit_Limit", "High_Credit_Limit"],
)
```

In [87]:

```
data1["Total_Revolving_Bal"] = pd.qcut(
    data1["Total_Revolving_Bal"],
    q=[0, 0.25, 0.5, 1],
    labels=["Low_Revolving_Bal", "Medium_Revolving_Bal", "High_Revolving_Bal"
)
```

In [88]:

```
data1['Avg_Open_To_Buy'] = pd.qcut(
    data1['Avg_Open_To_Buy'],
    q=[0, 0.25, 0.5, 1],
    labels=["Low_Avg_Open_To_Buy", "Medium_Avg_Open_To_Buy", "High_Avg_Open_T
)
```

In [90]:

```
data1['Total_Trans_Amt'] = pd.qcut(
    data1['Total_Trans_Amt'],
    q=[0, 0.25, 0.5, 1],
    labels=["Low_Total_Trans_Amt", "Medium_Total_Trans_Amt", "High_Total_Tran
)
```

In [89]:

```
data1['Total_Amt_Chng_Q4_Q1'] = pd.qcut(
    data1['Total_Amt_Chng_Q4_Q1'],
```

```
    data1['Total_Amt_Chng_Q4_Q1'],
    q=[0, 0.25, 0.5, 1],
    labels=["Low_Total_Amt_Chng_Q4_Q1", "Medium_Total_Amt_Chng_Q4_Q1", "High_
)
```

In [91]:

```
data1['Total_Ct_Chng_Q4_Q1'] = pd.qcut(
    data1['Total_Ct_Chng_Q4_Q1'],
    q=[0, 0.25, 0.5, 1],
    labels=["Low_Total_Ct_Chng_Q4_Q1", 'Medium_Total_Ct_Chng_Q4_Q1', 'High_To
)
```

In [92]:

```
data1['Avg_Utilization_Ratio'] = pd.qcut(
    data1['Avg_Utilization_Ratio'],
    q=[0, 0.25, 0.5, 1],
    labels=["Low_Avg_Utilization_Ratio", "Medium_Avg_Utilization_Ratio", "Hig
)
```

In [93]:

```
data1['Total_Trans_Ct'] = pd.qcut(
    data1['Total_Trans_Ct'],
    q=[0, 0.25, 0.5, 1],
    labels=["Low_Total_Trans_Ct", "Medium_Total_Trans_Ct", "High_Total_Trans_
)
```

In [94]:

```
data1["Credit_Limit"].value_counts()
```

Out[94]:

```
Credit_Limit
High_Credit_Limit      5061
Low_Credit_Limit       2535
Medium_Credit_Limit    2531
Name: count, dtype: int64
```

In [95]:

```
data1['Total_Revolving_Bal'].value_counts()
```

Out[95]:

```
Total_Revolving_Bal
High_Revolving_Bal      5063
Low_Revolving_Bal       2532
Medium_Revolving_Bal    2532
Name: count, dtype: int64
```

In [96]:

```
data1['Avg_Open_To_Buy'].value_counts()
```

Out[96]:

```
Avg_Open_To_Buy
High_Avg_Open_To_Buy      5063
Low_Avg_Open_To_Buy       2532
Medium_Avg_Open_To_Buy    2532
Name: count, dtype: int64
```

In [97]:

```
data1['Total_Trans_Amt'].value_counts()
```

Out[97]:

```
Total_Trans_Amt
High_Total_Trans_Amt      5063
Low_Total_Trans_Amt       2532
Medium_Total_Trans_Amt    2532
```

```
Medium_Total_Trans_Amt    2532
Name: count, dtype: int64
```

In [98]:
```python
data1['Total_Amt_Chng_Q4_Q1'].value_counts()
```

Out[98]:
```
Total_Amt_Chng_Q4_Q1
High_Total_Amt_Chng_Q4_Q1      5063
Low_Total_Amt_Chng_Q4_Q1       2557
Medium_Total_Amt_Chng_Q4_Q1    2507
Name: count, dtype: int64
```

In [99]:
```python
data1['Total_Ct_Chng_Q4_Q1'].value_counts()
```

Out[99]:
```
Total_Ct_Chng_Q4_Q1
High_Total_Ct_Chng_Q4_Q1      5049
Medium_Total_Ct_Chng_Q4_Q1    2541
Low_Total_Ct_Chng_Q4_Q1       2537
Name: count, dtype: int64
```

In [100…
```python
data1['Total_Trans_Ct'].value_counts()
```

Out[100…
```
Total_Trans_Ct
High_Total_Trans_Ct      5004
Low_Total_Trans_Ct       2611
Medium_Total_Trans_Ct    2512
Name: count, dtype: int64
```

In [102…
```python
data1['Avg_Utilization_Ratio'].value_counts()
```

Out[102…
```
Avg_Utilization_Ratio
High_Avg_Utilization_Ratio      5057
Low_Avg_Utilization_Ratio       2541
Medium_Avg_Utilization_Ratio    2529
Name: count, dtype: int64
```

Model Building

Model evaluation criterion

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 [86]:

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

    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 [87]:

```
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()
            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 [88]:

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

```python
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(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("dtree", DecisionTreeClassifier(random_state=1, class_weight='
models.append(("XGBoost", XGBClassifier(random_state=1, eval_metric='logloss'
 ## 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: 0.985655737704918
Random forest: 1.0
GBM: 0.875
Adaboost: 0.826844262295082
dtree: 1.0
XGBoost: 1.0

Validation Performance:

Bagging: 0.8067484662576687
Random forest: 0.8128834355828221
GBM: 0.8588957055214724
Adaboost: 0.852760736196319
dtree: 0.7822085889570553
XGBoost: 0.9079754601226994
```

Model Building - Oversampled Data

In [89]:

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

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
print("After Oversampling, counts of label 'No': {} \n".format(sum(y_train_ov


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

```
Before Oversampling, counts of label 'Yes': 976
```

```
Before Oversampling, counts of label 'No': 5099

After Oversampling, counts of label 'Yes': 5099
After Oversampling, counts of label 'No': 5099

After Oversampling, the shape of train_X: (10198, 30)
After Oversampling, the shape of train_y: (10198,)
```

In [90]:

```
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(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("dtree", DecisionTreeClassifier(random_state=1, class_weight='
models.append(("XGBoost", XGBClassifier(random_state=1, eval_metric='logloss'
 ## 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_train_over, model.predict(X_train_over))  ## Comp
    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: 0.9974504804863699
Random forest: 1.0
GBM: 0.9792116101196313
Adaboost: 0.9645028436948421
dtree: 1.0
XGBoost: 1.0

Validation Performance:

Bagging: 0.8773006134969326
Random forest: 0.8588957055214724
GBM: 0.9079754601226994
Adaboost: 0.8926380368098159
dtree: 0.843558282208589
XGBoost: 0.901840490797546
```

Model Building - Undersampled Data

In [91]:

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

In [92]:

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

```
print("After Under Sampling, counts of label 'Yes': {}".format(sum(y_train_un
print("After Under Sampling, counts of label 'No': {} \n".format(sum(y_train_

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

```
Before Under Sampling, counts of label 'Yes': 976
Before Under Sampling, counts of label 'No': 5099

After Under Sampling, counts of label 'Yes': 976
After Under Sampling, counts of label 'No': 976

After Under Sampling, the shape of train_X: (1952, 30)
After Under Sampling, the shape of train_y: (1952,)
```

In [93]:

```
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(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("dtree", DecisionTreeClassifier(random_state=1, class_weight='
models.append(("XGBoost", XGBClassifier(random_state=1, eval_metric='logloss'
## 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_train_un, model.predict(X_train_un))  ## Complete
    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: 0.9907786885245902
Random forest: 1.0
GBM: 0.9795081967213115
Adaboost: 0.9528688524590164
dtree: 1.0
XGBoost: 1.0

Validation Performance:

Bagging: 0.9171779141104295
Random forest: 0.9263803680981595
GBM: 0.9570552147239264
Adaboost: 0.9601226993865031
dtree: 0.8957055214723927
XGBoost: 0.9601226993865031
```

In [94]:

```
print("\nTraining and Validation Performance Difference:\n")

for name, model in models:
```

```
    model.fit(X_train_un, y_train_un)
    scores_train = recall_score(y_train_un, model.predict(X_train_un))
    scores_val = recall_score(y_val, model.predict(X_val))
    difference3 = scores_train - scores_val
    print("{}: Training Score: {:.4f}, Validation Score: {:.4f}, Difference:
```

```
Training and Validation Performance Difference:

Bagging: Training Score: 0.9908, Validation Score: 0.9172, Difference: 0.0736
Random forest: Training Score: 1.0000, Validation Score: 0.9264, Difference: 0.
0736
GBM: Training Score: 0.9795, Validation Score: 0.9571, Difference: 0.0225
Adaboost: Training Score: 0.9529, Validation Score: 0.9601, Difference: -0.0073
dtree: Training Score: 1.0000, Validation Score: 0.8957, Difference: 0.1043
XGBoost: Training Score: 1.0000, Validation Score: 0.9601, Difference: 0.0399
```

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 [95]:

```
%%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

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

```
print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_p
```

```
Best parameters are {'n_estimators': 100, 'learning_rate': 0.1, 'base_estimato
r': DecisionTreeClassifier(max_depth=3, random_state=1)} with CV score=0.835034
0136054422:
CPU times: user 4.18 s, sys: 270 ms, total: 4.45 s
Wall time: 1min 48s
```

In [96]:

```
# Creating new pipeline with best parameters
tuned_adb = AdaBoostClassifier( random_state= 1,
    n_estimators= 100, learning_rate= 0.1, base_estimator= DecisionTreeClassi
) ## Complete the code with the best parameters obtained from tuning

tuned_adb.fit(X_train, y_train) ## Complete the code to fit the model on orig
```

Out[96]:

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3,
                                                         random_state=
1),
                   learning_rate=0.1, n_estimators=100, random_state=
1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [97]:

```
adb_train = model_performance_classification_sklearn(tuned_adb, X_train, y_tr
adb_train
```

Out[97]:

	Accuracy	Recall	Precision	F1
0	0.982	0.927	0.961	0.944

In [98]:

```
# Checking model's performance on validation set
adb_val =  model_performance_classification_sklearn(tuned_adb, X_val,y_val) #
adb_val
```

Out[98]:

	Accuracy	Recall	Precision	F1
0	0.968	0.862	0.934	0.896

Tuning Ada Boost using undersampled data

In [99]:

```
# Creating new pipeline with best parameters
tuned_ada2 = AdaBoostClassifier( random_state=1,
    n_estimators= 100, learning_rate= 0.1, base_estimator= DecisionTreeClassi
) ## 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
```

Out[99]:

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3,
```

```
Out[99]:                                                    random_state=
1),
                    learning_rate=0.1, n_estimators=100, random_state=
1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [100…

```
ada2_train = model_performance_classification_sklearn(tuned_ada2, X_train_un,
ada2_train
```

Out[100…

	Accuracy	Recall	Precision	F1
0	0.991	0.997	0.985	0.991

In [101…

```
# Checking model's performance on validation set
ada2_val =  model_performance_classification_sklearn(tuned_ada2, X_val, y_val
ada2_val
```

Out[101…

	Accuracy	Recall	Precision	F1
0	0.938	0.969	0.731	0.834

Tuning Gradient Boosting using undersampled data

In [102…

```
%%time

#Creating pipeline
Model = GradientBoostingClassifier(random_state=1)

#Parameter grid to pass in RandomSearchCV
param_grid = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random
    "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

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_un, y_train_un) ## Complete the code to fit the mod


print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_p
```

```
Best parameters are {'subsample': 0.9, 'n_estimators': 100, 'max_features': 0.
```

```
5, learning_rate : 0.1, init : AdaBoostClassifier(random_state=1)} with CV sc
ore=0.9518576661433805:
CPU times: user 2.24 s, sys: 173 ms, total: 2.41 s
Wall time: 1min 10s
```

In [103…

```
# 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[103…

```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.5, random_state=1, subsample
=0.9)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [104…

```
gbm1_train = model_performance_classification_sklearn(tuned_gbm1, X_train_un,
gbm1_train
```

Out[104…

	Accuracy	Recall	Precision	F1
0	0.978	0.985	0.973	0.979

In [105…

```
gbm1_val = model_performance_classification_sklearn(tuned_gbm1, X_val, y_val)
gbm1_val
```

Out[105…

	Accuracy	Recall	Precision	F1
0	0.940	0.957	0.743	0.836

Tuning Gradient Boosting using original data

In [106…

```
%%time

#defining model
Model = GradientBoostingClassifier(random_state=1)

#Parameter grid to pass in RandomSearchCV
param_grid = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random_
    "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_

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


print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_p
```

```
Best parameters are {'subsample': 0.9, 'n_estimators': 100, 'max_features': 0.
5, 'learning_rate': 0.1, 'init': AdaBoostClassifier(random_state=1)} with CV sc
ore=0.8022187336473051:
CPU times: user 4.57 s, sys: 436 ms, total: 5.01 s
Wall time: 2min 46s
```

In [107…

```
# 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[107…

```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.5, random_state=1, subsample
=0.9)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [108…

```
gbm2_train = model_performance_classification_sklearn(tuned_gbm2, X_train, y_
gbm2_train
```

Out[108…

	Accuracy	Recall	Precision	F1
0	0.974	0.874	0.958	0.914

In [109…

```
gbm2_val = model_performance_classification_sklearn(tuned_gbm2, X_val, y_val)
gbm2_val
```

Out[109…

	Accuracy	Recall	Precision	F1
0	0.968	0.853	0.946	0.897

Tuning Gradient Boosting using over sampled data

In [110…

```
%%time

#defining model
Model = GradientBoostingClassifier(random_state=1)

#Parameter grid to pass in RandomSearchCV
param_grid = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random
    "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

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_over, y_train_over) ## Complete the code to fit the


print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_p
```

```
Best parameters are {'subsample': 0.9, 'n_estimators': 100, 'max_features': 0.
5, 'learning_rate': 0.1, 'init': AdaBoostClassifier(random_state=1)} with CV sc
ore=0.9523506321076025:
CPU times: user 7.41 s, sys: 613 ms, total: 8.02 s
Wall time: 4min 31s
```

In [111…

```
# Creating new pipeline with best parameters
tuned_gbm3 = 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_gbm3.fit(X_train_over, y_train_over)
```

Out[111…

```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.5, random_state=1, subsample
=0.9)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [112…

```
gbm3_train = model_performance_classification_sklearn(tuned_gbm3, X_train_ove
gbm3_train
```

Out[112…

Accuracy	Recall	Precision	F1

0	0.975	0.980	0.971	0.975

In [113…

```
gbm3_val =  model_performance_classification_sklearn(tuned_gbm3, X_val, y_val
gbm3_val
```

Out[113…

	Accuracy	Recall	Precision	F1
0	0.957	0.911	0.834	0.871

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 [114…

```
%%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_

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

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_p
```

```
Best parameters are {'subsample': 0.9, 'scale_pos_weight': 5, 'n_estimators': 1
00, 'learning_rate': 0.05, 'gamma': 3} with CV score=0.9221297749869178:
CPU times: user 3.48 s, sys: 205 ms, total: 3.68 s
Wall time: 1min 2s
```

In [115…

```
tuned_xgb = XGBClassifier(
    random_state=1,
    eval_metric="logloss",
    subsample= 0.9,
    scale_pos_weight= 5,
    n_estimators= 100,
    learning_rate= 0.05,
    gamma= 3,
)## Complete the code with the best parameters obtained from tuning

tuned_xgb.fit(X_train, y_train)
```

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

Out[115...
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_round
s=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=Non
e,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=N
one,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estima
tors=100,
              n_jobs=None, num_parallel_tree=None, random_state=1,
...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [116...
```
xgb_train = model_performance_classification_sklearn(tuned_xgb, X_train, y_tr
xgb_train
```

Out[116...

	Accuracy	Recall	Precision	F1
0	0.981	1.000	0.892	0.943

In [117...
```
xgb_val = model_performance_classification_sklearn(tuned_xgb, X_val, y_val) #
xgb_val
```

Out[117...

	Accuracy	Recall	Precision	F1
0	0.958	0.942	0.821	0.877

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 [118...
```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        gbm1_train.T,
        gbm2_train.T,
        ada2_train.T,
        gbm3_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",
    "Gradient boosting trained with Oversampled data",
    "XGBoost trained with Original data"
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[118…

	Gradient boosting trained with Undersampled data	Gradient boosting trained with Original data	AdaBoost trained with Undersampled data	Gradient boosting trained with Oversampled data	XGBoost trained with Original data
Accuracy	0.978	0.974	0.991	0.975	0.981
Recall	0.985	0.874	0.997	0.980	1.000
Precision	0.973	0.958	0.985	0.971	0.892
F1	0.979	0.914	0.991	0.975	0.943

In [119…

```
# validation performance comparison

models_val_comp_df = pd.concat(
    [
        gbm1_val.T,
        gbm2_val.T,
        ada2_val.T,
        gbm3_val.T,
        xgb_val.T,

    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Gradient boosting validation with Undersampled data",
    "Gradient boosting validation with Original data",
    "AdaBoost validation with Undersampled data",
    "Gradient boosting validation with Oversampled data",
    "XGBoost validation with Original data"
]
print("Validation performance comparison:")
models_train_comp_df
## Write the code to compare the performance on validation set
```

```
Validation performance comparison:
```

Out[119…

	Gradient boosting validation with Undersampled data	Gradient boosting validation with Original data	AdaBoost validation with Undersampled data	Gradient boosting validation with Oversampled data	XGBoost validation with Original data
Accuracy	0.978	0.974	0.991	0.975	0.981

Recall	0.985	0.874	0.997	0.980	1.000
Precision	0.973	0.958	0.985	0.971	0.892
F1	0.979	0.914	0.991	0.975	0.943

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

In [122…

```
# Let's check the performance on test set
ada2_test = model_performance_classification_sklearn(tuned_ada2, X_test, y_te
ada2_test  ## Write the code to check the performance of best model on test d
```

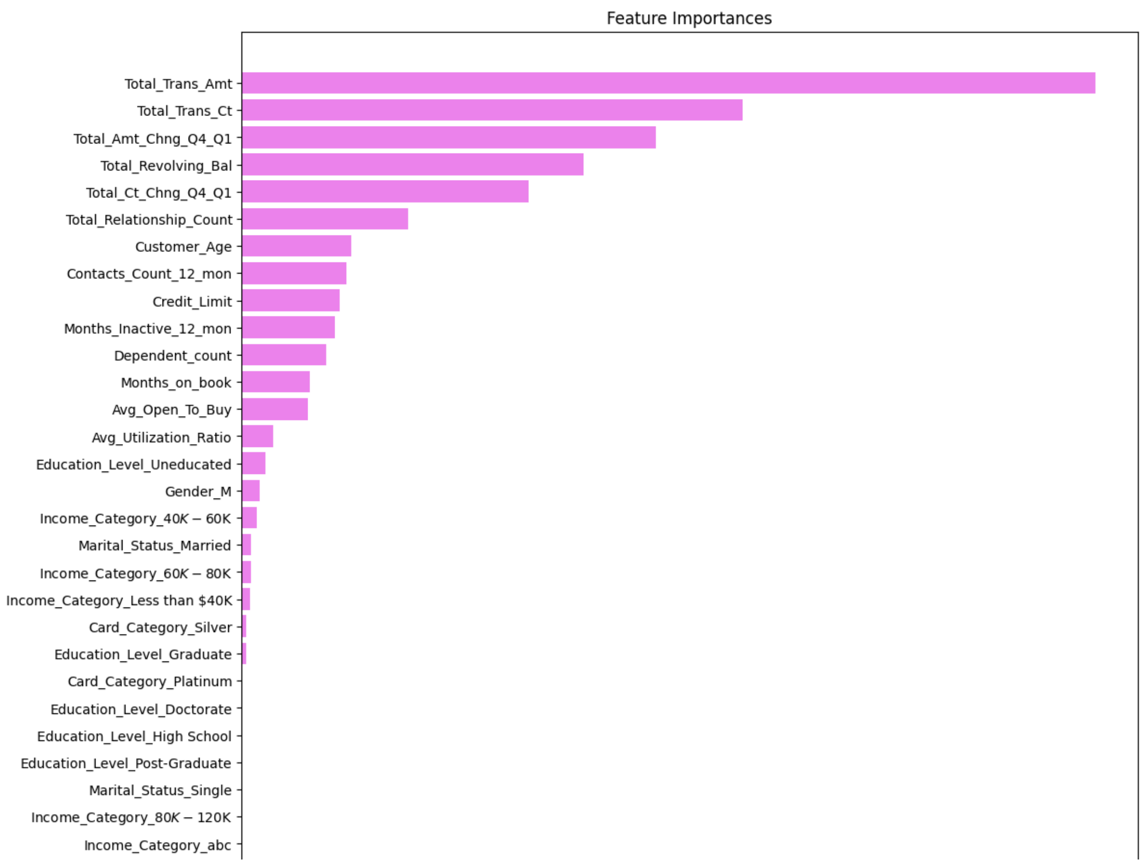
Out[122…

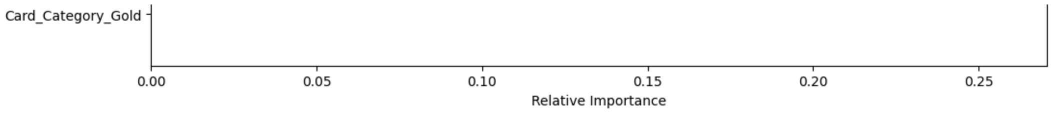
	Accuracy	Recall	Precision	F1
0	0.938	0.966	0.732	0.833

Feature Importances

In [121…

```
feature_names = X_train.columns
importances =  tuned_ada2.feature_importances_ ## Complete the code to check
indices = np.argsort(importances)

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





Business Insights and Conclusions