Credit Card Users Churn Prediction

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 [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
# 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
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.6/9.6 MB 51.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.2/18.2 MB 24.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.1/12.1 MB 43.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 226.0/226.0 kB 13.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 297.1/297.1 MB 4.3 MB/s eta 0:00:00
  WARNING: The scripts f2py, f2py3 and f2py3.10 are installed in '/root/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
cudf-cu12 24.4.1 requires pandas<2.2.2dev0,>=2.0, but you have pandas 1.5.3 which is incompatible.
google-colab 1.0.0 requires pandas==2.1.4, but you have pandas 1.5.3 which is incompatible.
pandas-stubs 2.1.4.231227 requires numpy>=1.26.0; python_version < "3.13", but you have numpy 1.25.2 which is incompatible.
xarray 2024.6.0 requires pandas>=2.0, but you have pandas 1.5.3 which is incompatible.
```

In [3]:
```
# 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
# !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 [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
import scipy.stats as stats
from sklearn import metrics
from sklearn import tree
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
import warnings
warnings.filterwarnings('ignore')


from sklearn.ensemble import BaggingRegressor,RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegress
from xgboost import XGBRegressor
from sklearn import metrics
from sklearn.model_selection import GridSearchCV, train_test_split
```

Loading the dataset

In [5]:
```
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/BankChurners.csv")
```

Data Overview

- Observations
- Sanity checks

Viewing the first 5 rows of the dataset

In [6]:
```
df.head()
```

Out[6]:

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

5 rows × 21 columns

Viewing the last 5 rows of the dataset

In [7]:
```
df.tail()
```

Out[7]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Catego
10122	772366833	Existing Customer	50	M	2	Graduate	Single	$40K - 60K$	Bl
10123	710638233	Attrited Customer	41	M	2	NaN	Divorced	$40K - 60K$	Bl
10124	716506083	Attrited Customer	44	F	1	High School	Married	Less than $40K	Bl
10125	717406983	Attrited Customer	30	M	2	Graduate	NaN	$40K - 60K$	Bl
10126	714337233	Attrited Customer	43	F	2	Graduate	Married	Less than $40K	Silv

5 rows × 21 columns

Understanding the shape of the dataset

In [8]:
```
df.shape
```

Out[8]:
```
(10127, 21)
```

There are 10,127 rows and 21 columns.

Checking the data types in the dataset.

In [9]:
```
df.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
```

- As seen above, the columns 'Education_Level' & 'Martial_Status' contain missing data

- As seen above, the columns 'Education_Level' & 'Marital_Status' contain missing data.
- There are 3 types of data types in the data, 'int64', 'object', and 'float64'.
- We can convert the object data types to a category, this reduces the storage of the data frame.

Converting the 'object' dtype to 'column'

In [10]:
```
# First creating a cols var to diplay the object dtype columns
cols = df.select_dtypes(['object'])
cols.columns
```

Out[10]:
```
Index(['Attrition_Flag', 'Gender', 'Education_Level', 'Marital_Status',
       'Income_Category', 'Card_Category'],
      dtype='object')
```

In [11]:
```
# Converting the dtypes
for i in cols.columns:
    df[i] = df[i].astype('category')
```

In [12]:
```
# Checking if the dtypes where converted correctly
df.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  category
 2   Customer_Age              10127 non-null  int64
 3   Gender                    10127 non-null  category
 4   Dependent_count           10127 non-null  int64
 5   Education_Level           8608 non-null   category
 6   Marital_Status            9378 non-null   category
 7   Income_Category           10127 non-null  category
 8   Card_Category             10127 non-null  category
 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: category(6), float64(5), int64(10)
memory usage: 1.2 MB
```

- As seen above, the object dtypes were converted to categorical data types.
- Also, the memory usage has also decreased from the convertage.

In [13]:
```
df.isna().sum()
```

Out[13]:

	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

- There is missing values in the 'Education_Level' & 'Martial Status' columns which may need to be treated.

Summary of the dataset

In [14]:
```
df.describe()
```

Out[14]:

	CLIENTNUM	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_
count	1.012700e+04	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000	1012
mean	7.391776e+08	46.325960	2.346203	35.928409	3.812580	2.341167	
std	3.690378e+07	8.016814	1.298908	7.986416	1.554408	1.010622	
min	7.080821e+08	26.000000	0.000000	13.000000	1.000000	0.000000	
25%	7.130368e+08	41.000000	1.000000	31.000000	3.000000	2.000000	
50%	7.179264e+08	46.000000	2.000000	36.000000	4.000000	2.000000	
75%	7.731435e+08	52.000000	3.000000	40.000000	5.000000	3.000000	
max	8.283431e+08	73.000000	5.000000	56.000000	6.000000	6.000000	

- The `Attrition_Flag` column, which is the target variable, shows that there are more existing customers than attrited customers. Furthermore, there is 84% of customers are existing than attrited, which is a significant comparison between the two.
- The `Gender` column shows that there are slightly more females than males, 53% to be exact.
- For the `Education_Level`, the most frequent are graduates compared to the rest, 36% out of 8608 customers. However, there are 1,519 missing values in the column.
- `Martial_Status` also contains missing values in the column, however 50% majority of customers are married, compared to the 'single' and 'NaN' values.
- `Income_Category` displays that 35% of customers have an annual income of 40K or less, this can be considered by the type of background the customers whose data were taken from or the situations they are currently in.
- `Card_Category` shows that almost all the customers have the 'Blue' card type compared to the other 3 types of cards, however, it is unknown what the colors represent for the cards.

Viewing the summary of the categorical columns

In [15]:
```
df.describe(include=['category']).T
```

Out[15]:

	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

- The `Attrition_Flag` column, which is the target variable, shows that there are more existing customers than attrited customers. Furthermore, there is 84% of customers are existing than attrited, which is a significant comparison between the two.
- The `Gender` column shows that there are slightly more females than males, 53% to be exact.
- For the `Education_Level`, the most frequent are graduates compared to the rest, 36% out of 8608 customers. However, there are 1,519 missing values in the column.
- `Martial_Status` also contains missing values in the column, however 50% majority of customers are married, compared to the 'single' and 'NaN' values.
- `Income_Category` displays that 35% of customers have an annual income of 40K or less, this can be considered by the type of background the customers whose data were taken from or the situations they are currently in.
- `Card_Category` shows that almost all the customers have the 'Blue' card type compared to the other 3 types of cards, however, it is unknown what the colors represent for the cards.

Dropping the 'CLIENTNUM' column, which is not needed

In [16]:
```
df.drop(['CLIENTNUM'],axis=1,inplace=True)
```

Viewing the unique values of all the categories

In [17]:
```
df.nunique()
```

Out[17]:

	0
Attrition_Flag	2
Customer_Age	45
Gender	2
Dependent_count	6
Education_Level	6
Marital_Status	3
Income_Category	6
Card_Category	4
Months_on_book	44
Total_Relationship_Count	6
Months_Inactive_12_mon	7
Contacts_Count_12_mon	7
Credit_Limit	6205
Total_Revolving_Bal	1974
Avg_Open_To_Buy	6813
Total_Amt_Chng_Q4_Q1	1158
Total_Trans_Amt	5033
Total_Trans_Ct	126
Total_Ct_Chng_Q4_Q1	830
Avg_Utilization_Ratio	964

dtype: int64

In [18]:
```
cols_cat= df.select_dtypes(['category'])
```

In [19]:
```
for i in cols_cat.columns:
    print('Unique values in',i, 'are :')
    print(cols_cat[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 the `Income_Category` column, there appears to be an error, where one category is labled as 'abc', this will need to be changed.

Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

Questions:

1. How is the total transaction amount distributed?
2. What is the distribution of the level of education of customers?
3. What is the distribution of the level of income of customers?
4. How does the change in transaction amount between Q4 and Q1 (`total_ct_change_Q4_Q1`) vary by the customer's account status (`Attrition_Flag`)?
5. How does 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`)?
6. What are the attributes that have a strong correlation with each other?

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

In [20]:
```
# 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 column
    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 [21]:
```
# 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 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
```

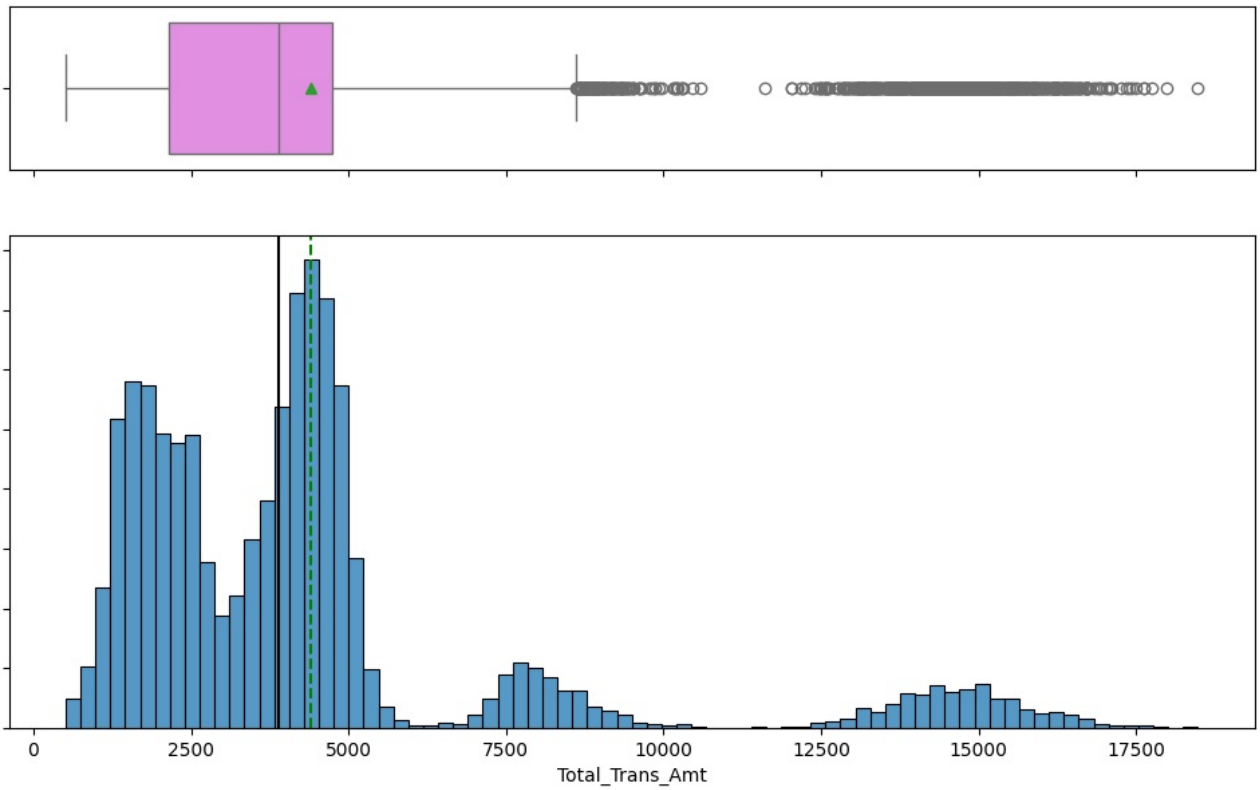
In [22]:
```
# function to plot stacked bar chart

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 [23]:
```
### 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()
```

Question 1

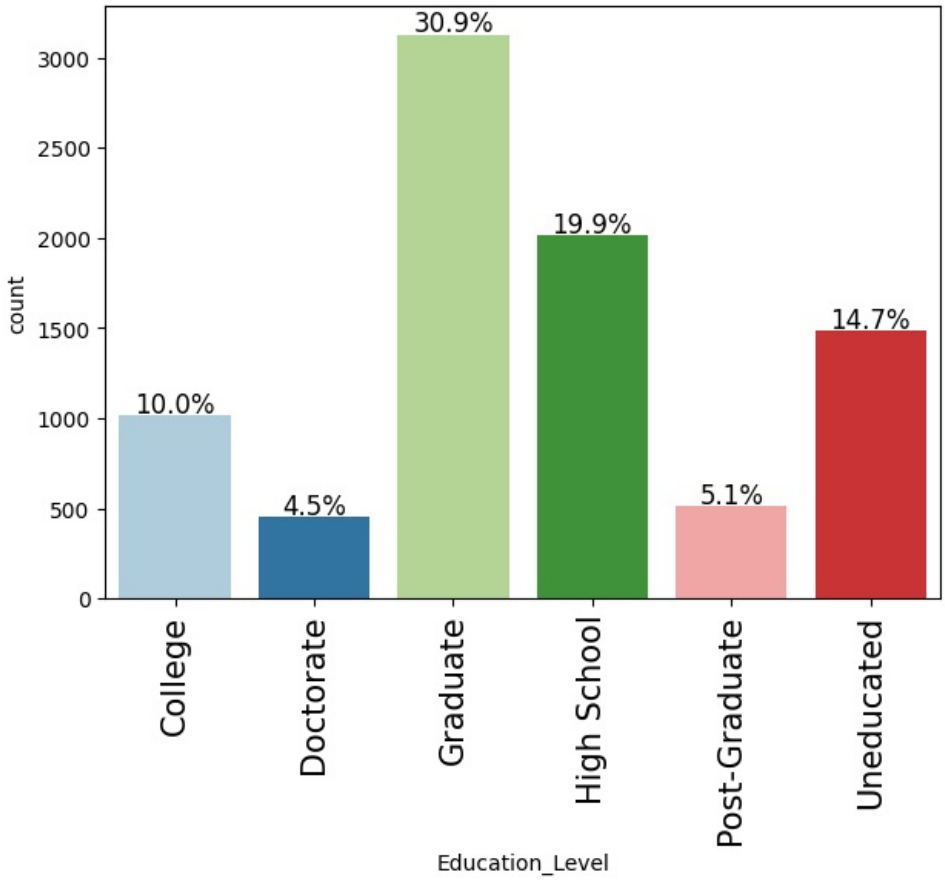
In [24]:
```
# How is the total transaction amount distributed?

histogram_boxplot(data=df, feature="Total_Trans_Amt")
```



- The total transaction amount is distributed from the majority of the amount being skewed to the right. This means that the transactions do not often cross the 5,000 dollar threshold, they often are below that total amount every year.

Question 2

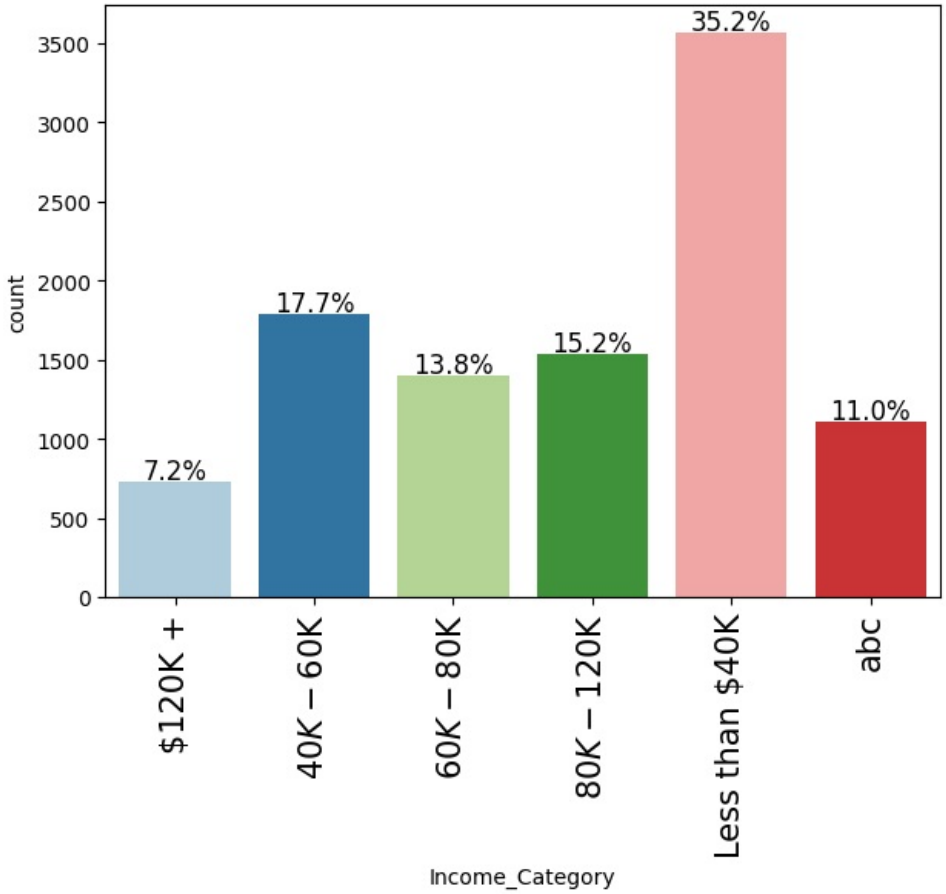
In [25]:
```
# What is the distribution of the level of education of customers?

labeled_barplot(data=df, feature="Education_Level", perc=True)
```



- The distribution is quite normal, as there appears to be no skewness for this column. However, there are more 'Graduate' & 'High School' education levels compared to the rest of the customers, which possibly aligns with the age group or certain similar backgrounds the customers have with one another.

Question 3

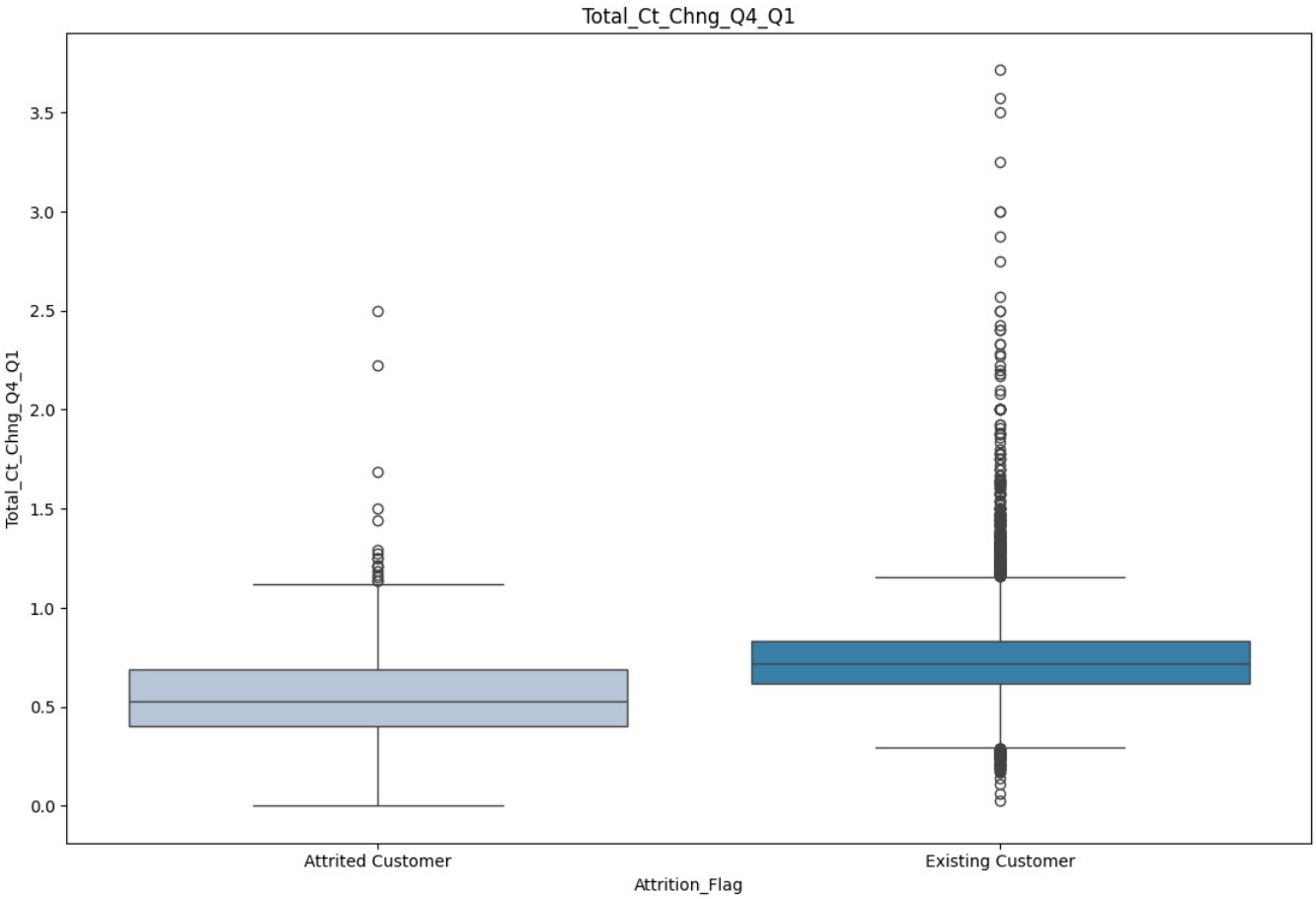
In [26]:
```
# What is the distribution of the level of income of customers?

labeled_barplot(data=df, feature="Income_Category", perc=True)
```



- The distribution for the level of income of customers is normally distributed for all the categories except customers that have an annual income of less than 40,000 dollars. Similar to the education level, the vast majority of the customers that have an income of 40K or less correlate to the type of area or job they have, or the type of wages in that particular location.

Question 4

In [27]:
```
# How does the change in transaction amount between Q4 and Q1 (total_ct_change_Q4_Q1) vary by the customer's ac

cols = df[['Total_Ct_Chng_Q4_Q1']].columns.tolist()
plt.figure(figsize=(22,22))

for i, variable in enumerate(cols):
                    plt.subplot(3,2,i+1)
                    sns.boxplot(data=df,x="Attrition_Flag",y=variable,palette="PuBu")
                    plt.tight_layout()
                    plt.title(variable)
plt.show()
```

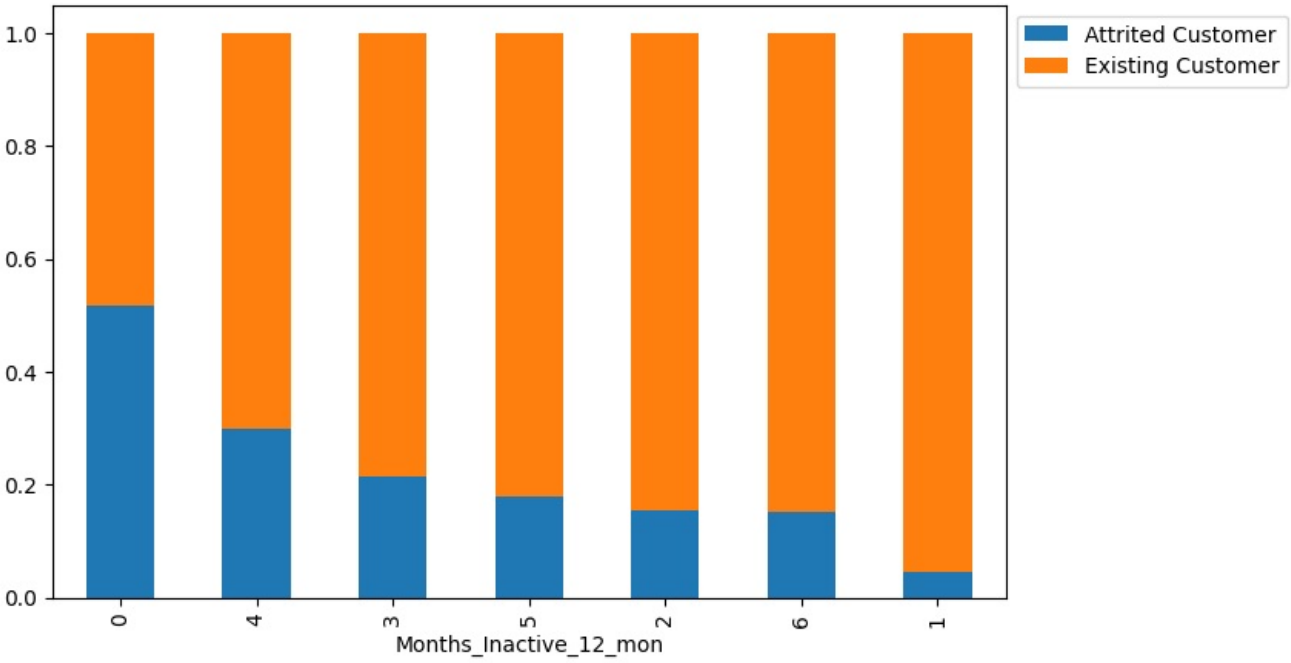


- The change in transaction amount between quarters 4 and 1 varies for the type of customer status only slightly. For the existing customer status, there are more outliers present as well as a slightly higher amount of ratio compared to those who are attrited. Depending on when the customer has stopped doing business with the company, those who are still existing are always going to have a higher total change in transaction count.

Question 5

In [28]:
```
# How does 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)?

stacked_barplot(df,"Months_Inactive_12_mon","Attrition_Flag")
```

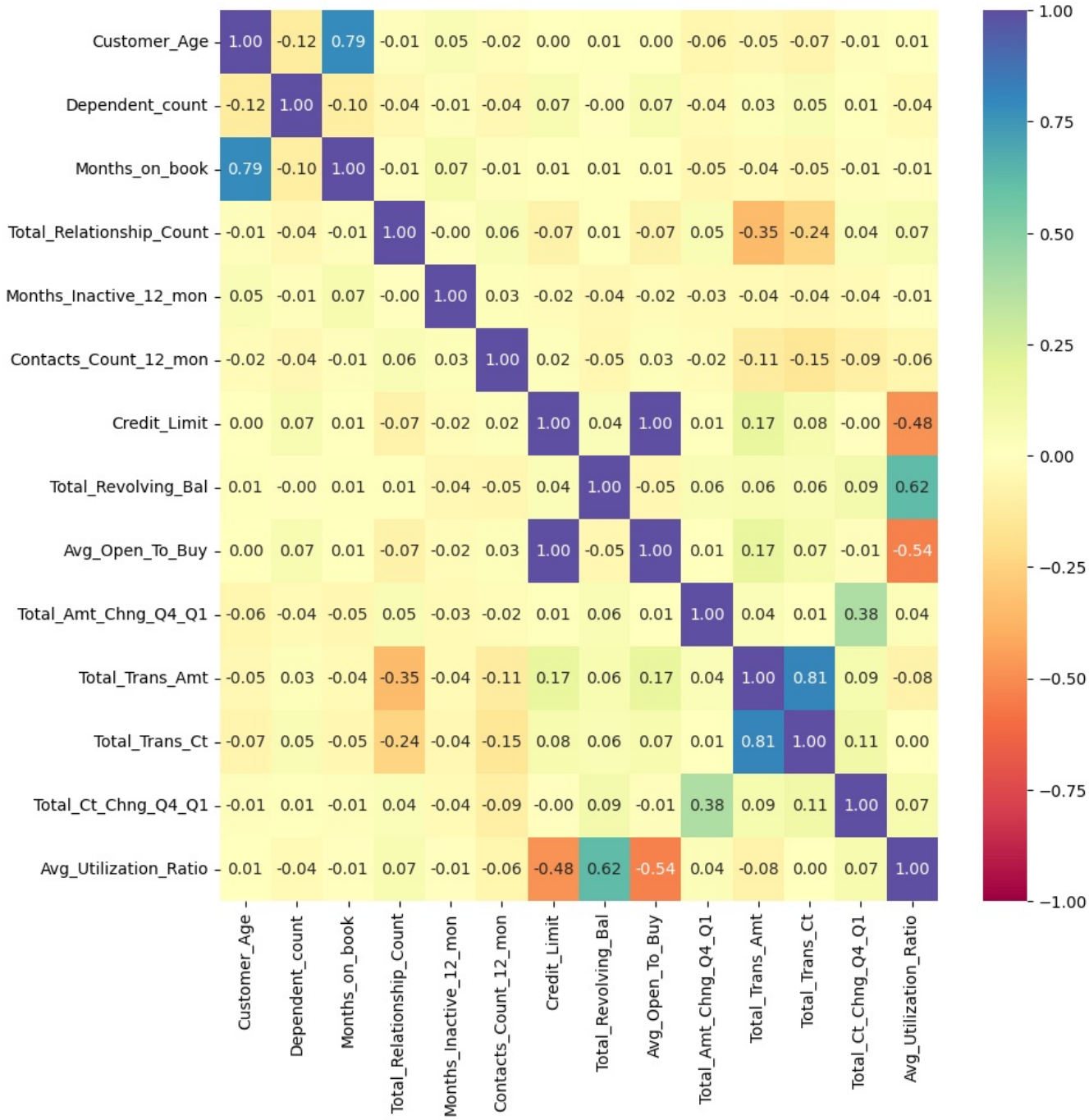
```
Attrition_Flag          Attrited Customer  Existing Customer    All
Months_Inactive_12_mon
All                                  1627               8500  10127
3                                     826               3020   3846
2                                     505               2777   3282
4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



- As seen in the above-stacked barplot, since there is a vast majority of existing customers compared to those who are attrited, there are always going to be more inactive customers for the ones who still exist. However, in the 0 months portion, more customers were inactive that were attrited than existing, but that was by one.
- Also, there is no correlation on the length of months inactive, as depicted above, the majority of customers who are attracted and existing have been inactive for 3 months.

Question 6

In [29]:
```
# What are the attributes that have a strong correlation with each other?


# Creating a only number data type heat map, as the categorical columns are not accepted.
numerical_df = df.select_dtypes(include=['number'])

plt.figure(figsize=(10,10))
sns.heatmap(numerical_df.corr(), annot=True, vmin=-1, vmax=1, fmt='.2f', cmap="Spectral")
plt.show()
```



The attributes that have a strong correlation with each other are:

- `Avg_Open_To_Buy` & `Credit_Limit` are perfectly correlated with 1, this will need to be removed during the data splitting phase.
- `Months_on_book` & `Customer_Age` have a strong correlation, this is possible because the older you are the more your relationship with the bank will be.
- `Total_Trans_Amt` & `Total_Trans_Ct` seem to have the highest correlation between all the other numerical data types. This can be because as the total count of transactions increases, the amount also increases with it. This could indicate that the customers chosen in this dataset like to shop frequently and spend more on a yearly average.

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

In [30]:
```
# 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 column
    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 [31]:
```
# 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 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
```

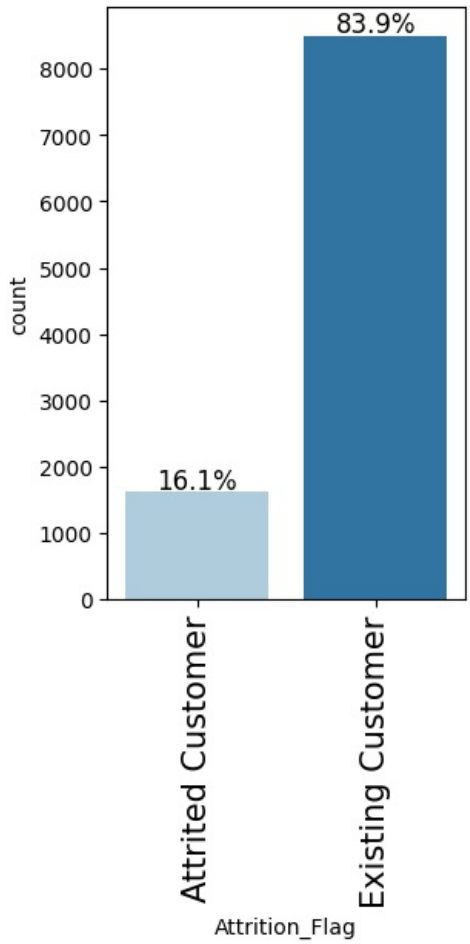
In [32]:
```
# function to plot stacked bar chart

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 [33]:
```
### 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()
```

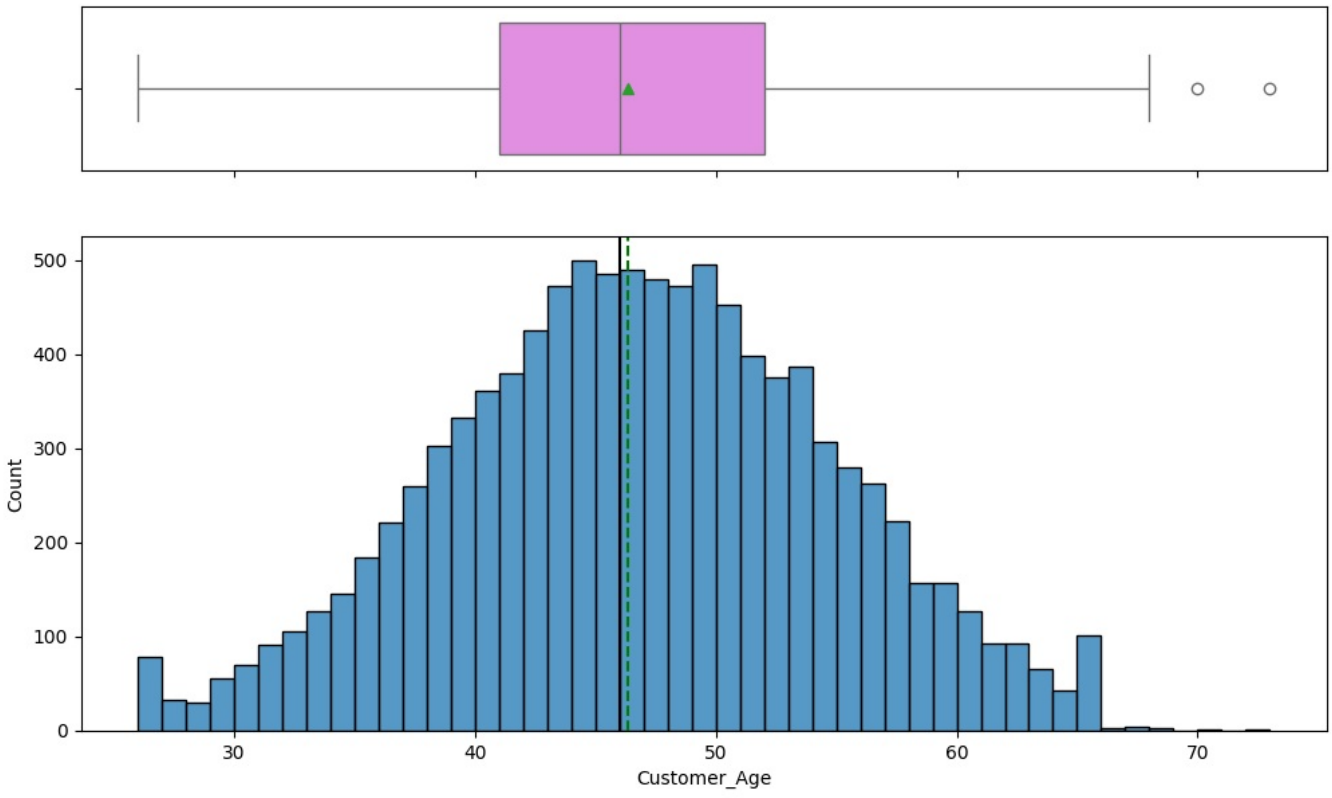
Univariate Analysis

In [34]:
```
# Observation on Attrition

labeled_barplot(df, "Attrition_Flag", perc=True)
```



- There are significantly more existing customers than attrited, a 68% difference, this is a good statistic to have for a company that wants to decrease the attrited customer percentage even more.

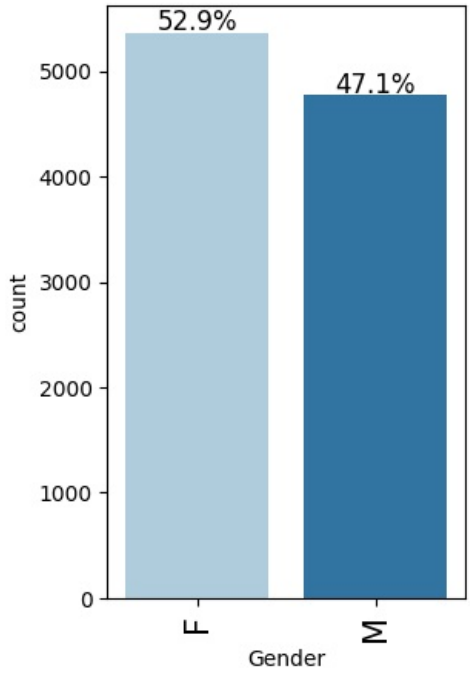
In [35]:
```
# Observation on Customer_Age

histogram_boxplot(df,'Customer_Age')
```



- The customers age throughout the dataset is almost perfectly normally distributed, with only a few possible outliers being pointed out.

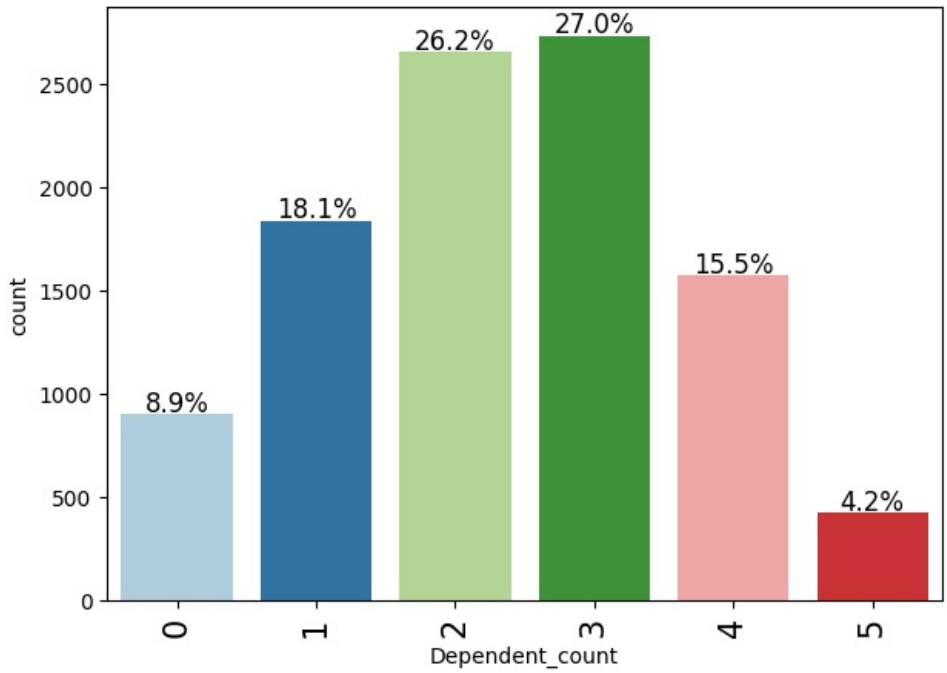
In [36]:
```
# Observation on Gender
```

```
labeled_barplot(df, "Gender", perc=True)
```



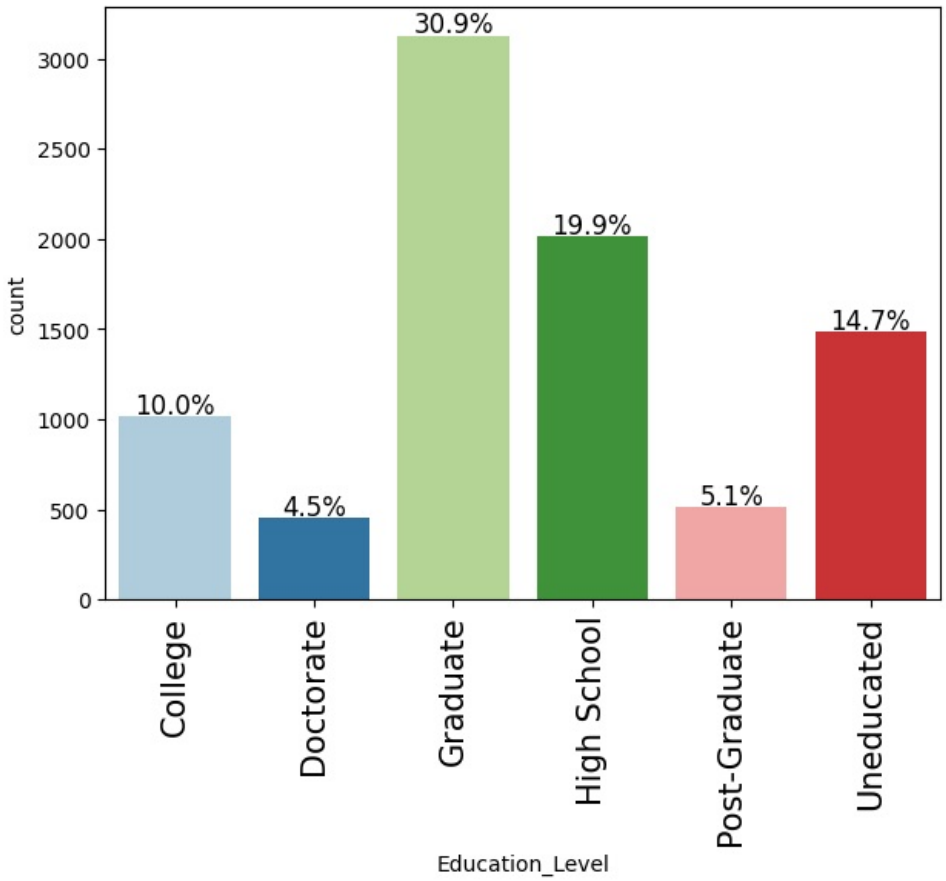
- In this dataset, there seems to be some sort of balance between female & males, although there is slighlty a bit more females.

In [37]:
```
# Observation on Dependent_count

labeled_barplot(df, "Dependent_count", perc=True)
```



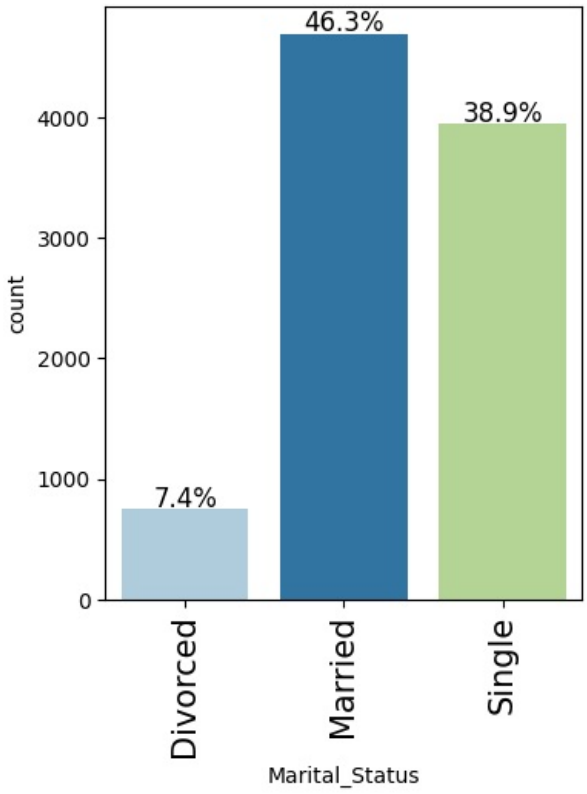
- The majority number of dependents being 2 to 3 means that most of the customers have family or children that financially depend on them.

In [38]:
```
# Observation on Education_Level

labeled_barplot(df, "Education_Level", perc=True)
```



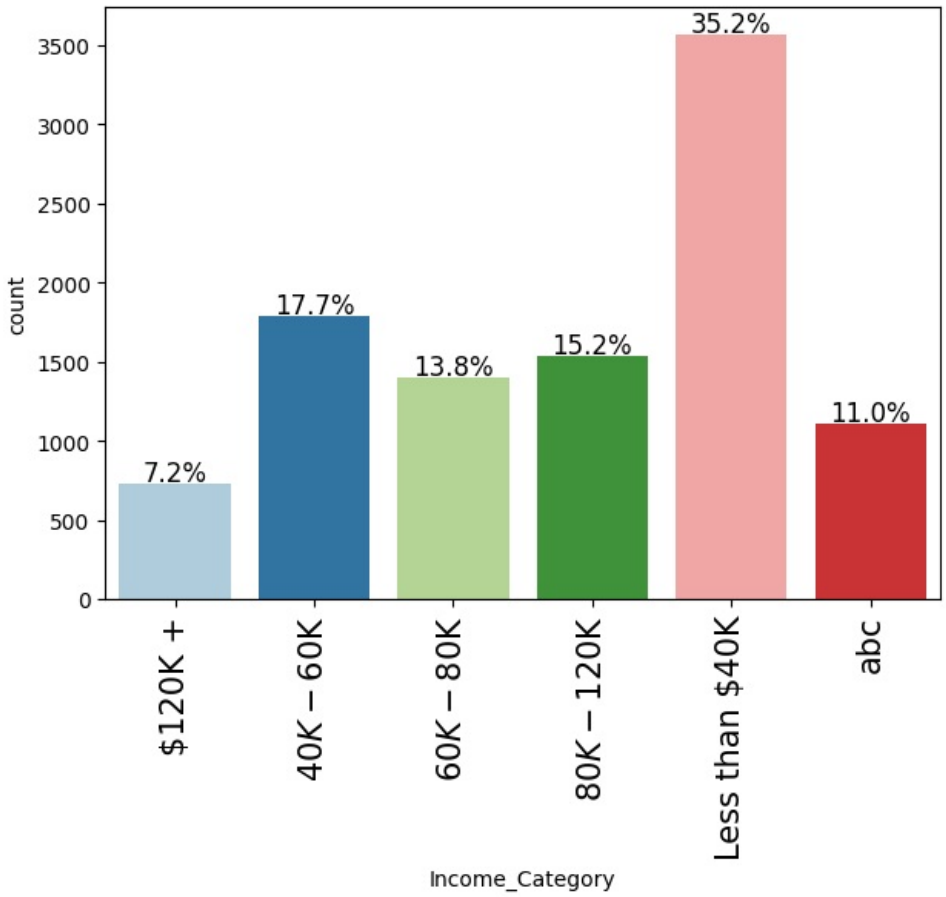
- The majority of customers are either graduates, high school grads, or uneducated. This can portray what and where the customers where picked out from, or their financial background as well.

In [39]:
```
# Observation on Marital_Status

labeled_barplot(df, "Marital_Status", perc=True)
```



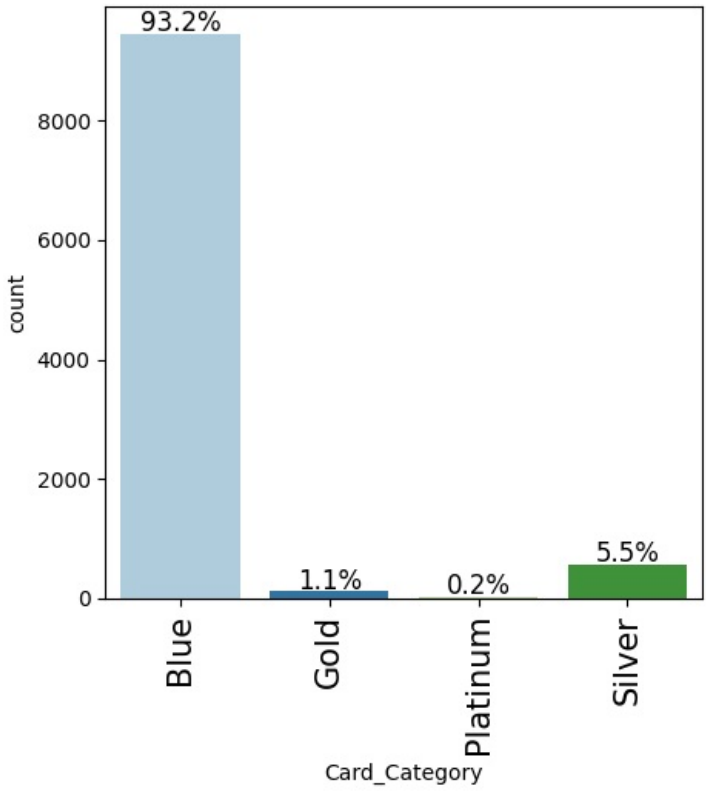
- There seems to be a close percentage of customers who are married & single, and a small chunk who are divorced.

In [40]:
```
# Observation on Income_Category

labeled_barplot(df, "Income_Category", perc=True)
```



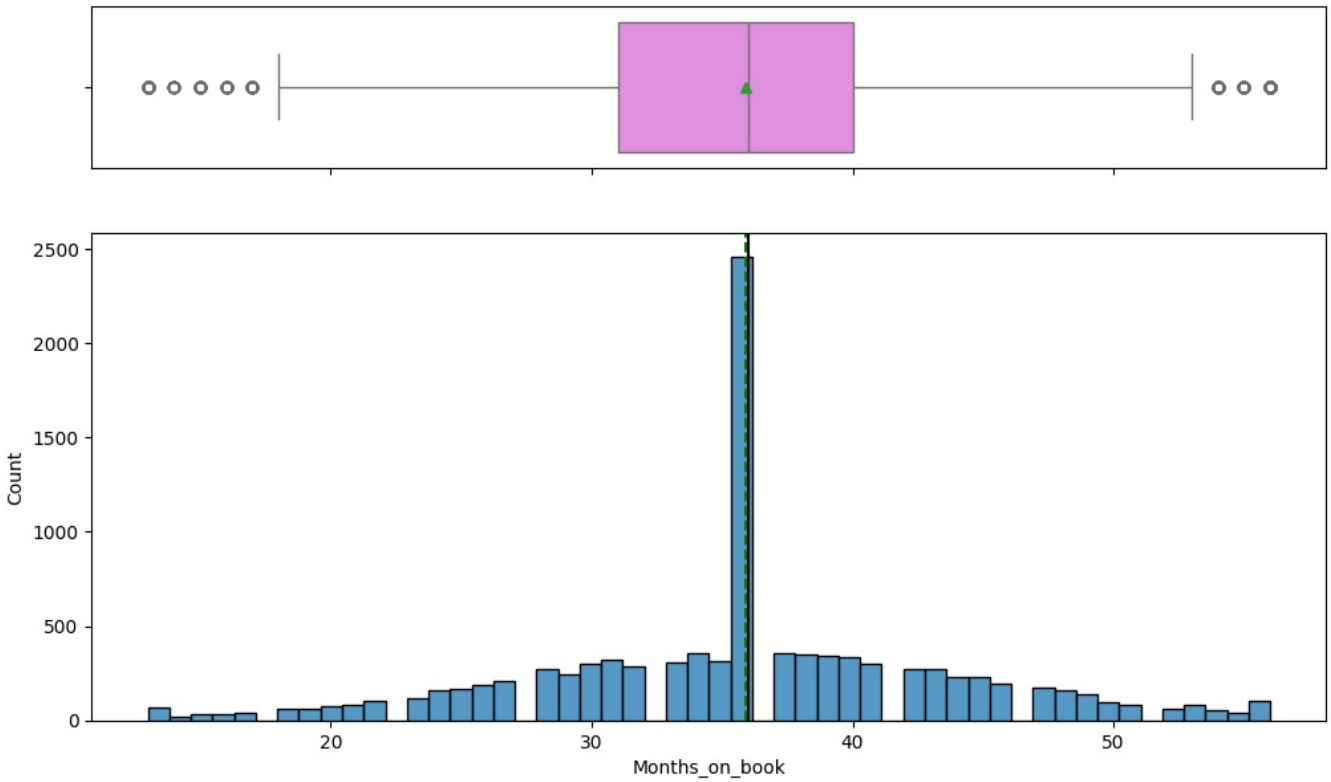
- There is a significant amount of customers who have an annual income of less than 40,000 dollars, this can be correlated to their martial status or even the amount of dependents they have.

In [41]:
```
# Observation on Card_Category

labeled_barplot(df, "Card_Category", perc=True)
```



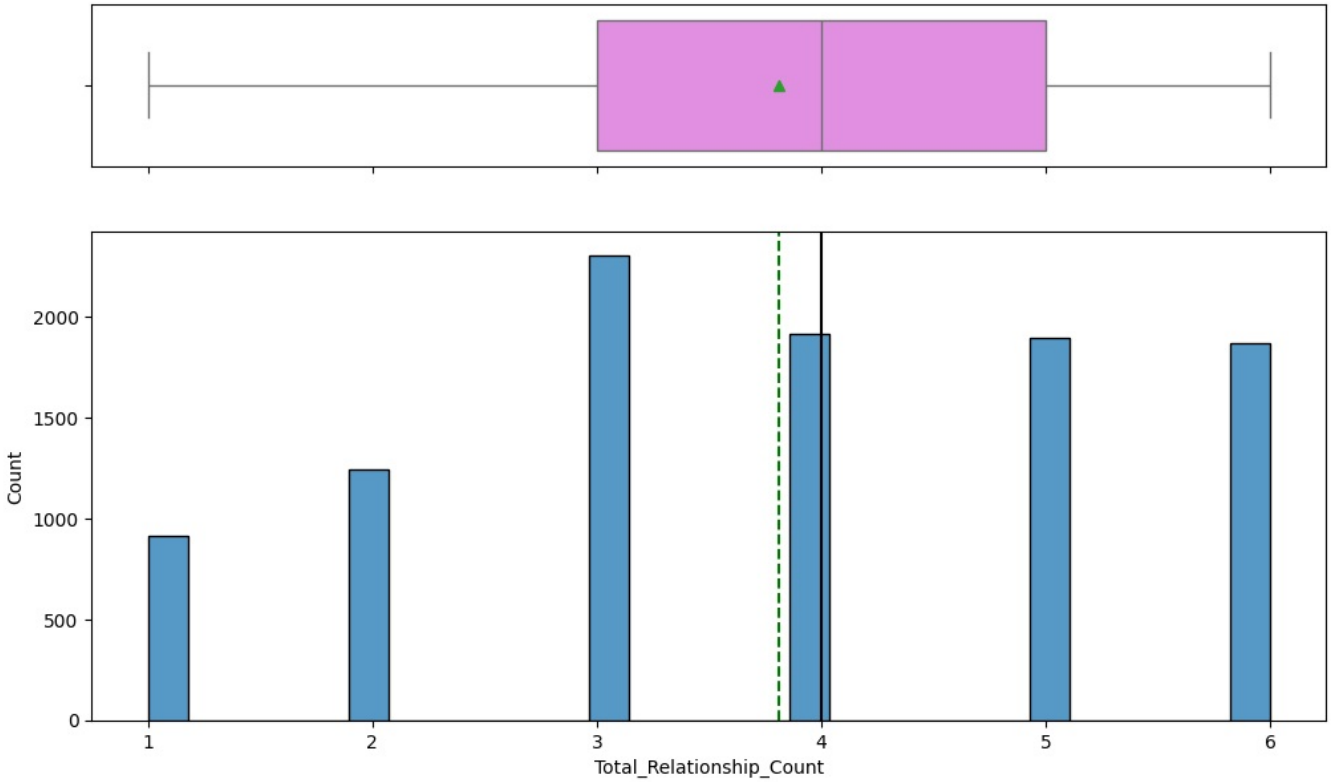
- Almost all customers have a 'Blue' type of card compared to the others.

In [42]:
```
# Observation on Months_on_book

histogram_boxplot(df,'Months_on_book')
```



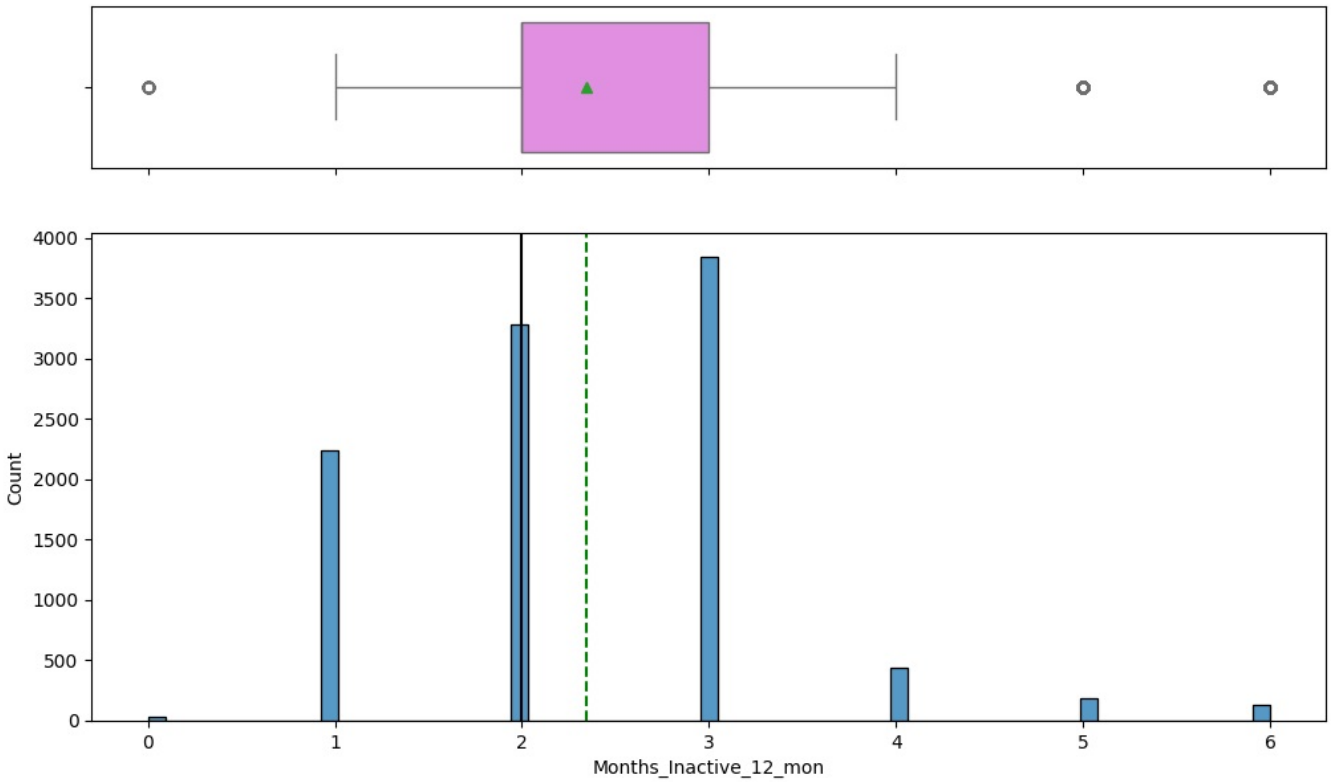
- The distribtion is normal, however there is a significant amount of customers who have about 35-36 months (3 years) of relationship with the bank.

In [43]:
```
# Observation on Total_Relationship_Count

histogram_boxplot(df,'Total_Relationship_Count')
```



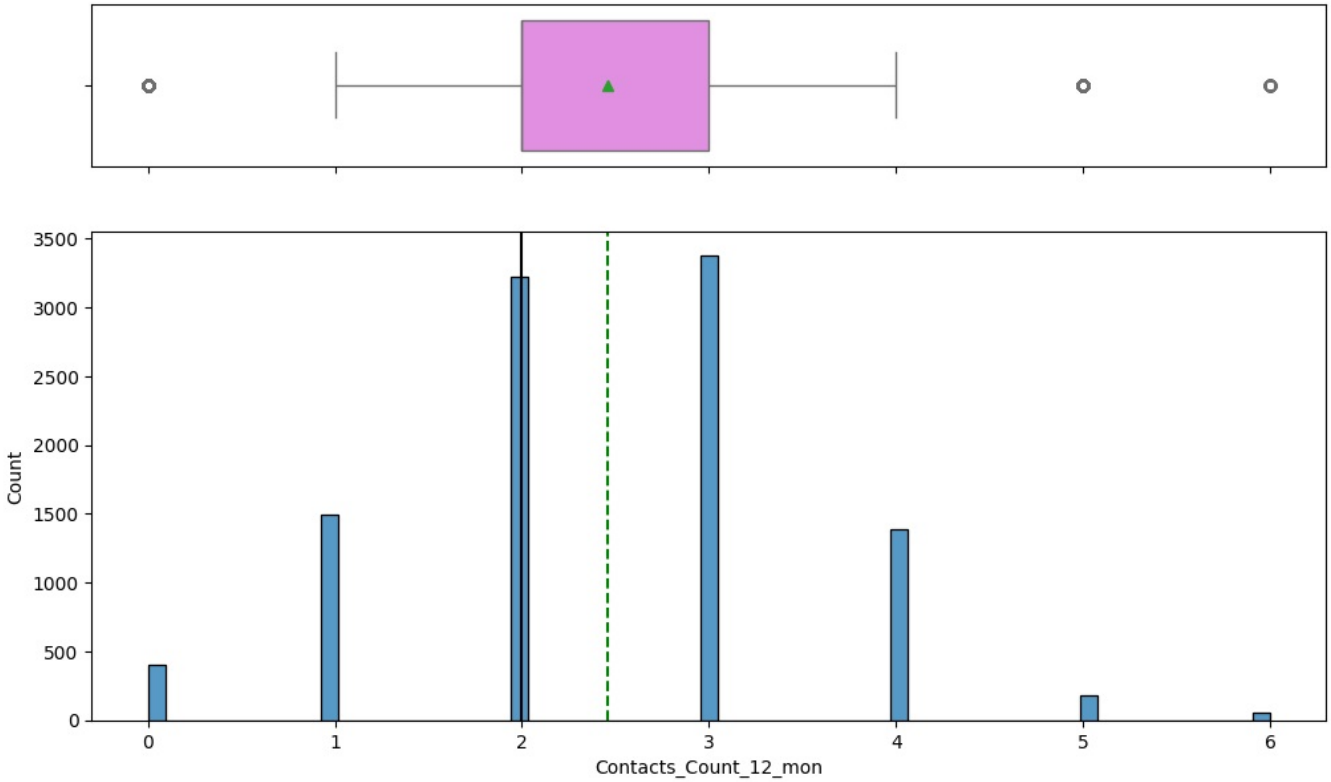
- The total number of products held by the customer seems to be normally distributed with the majority being 3 products.

In [44]:
```
# Observation on Months_Inactive_12_mon

histogram_boxplot(df,'Months_Inactive_12_mon')
```



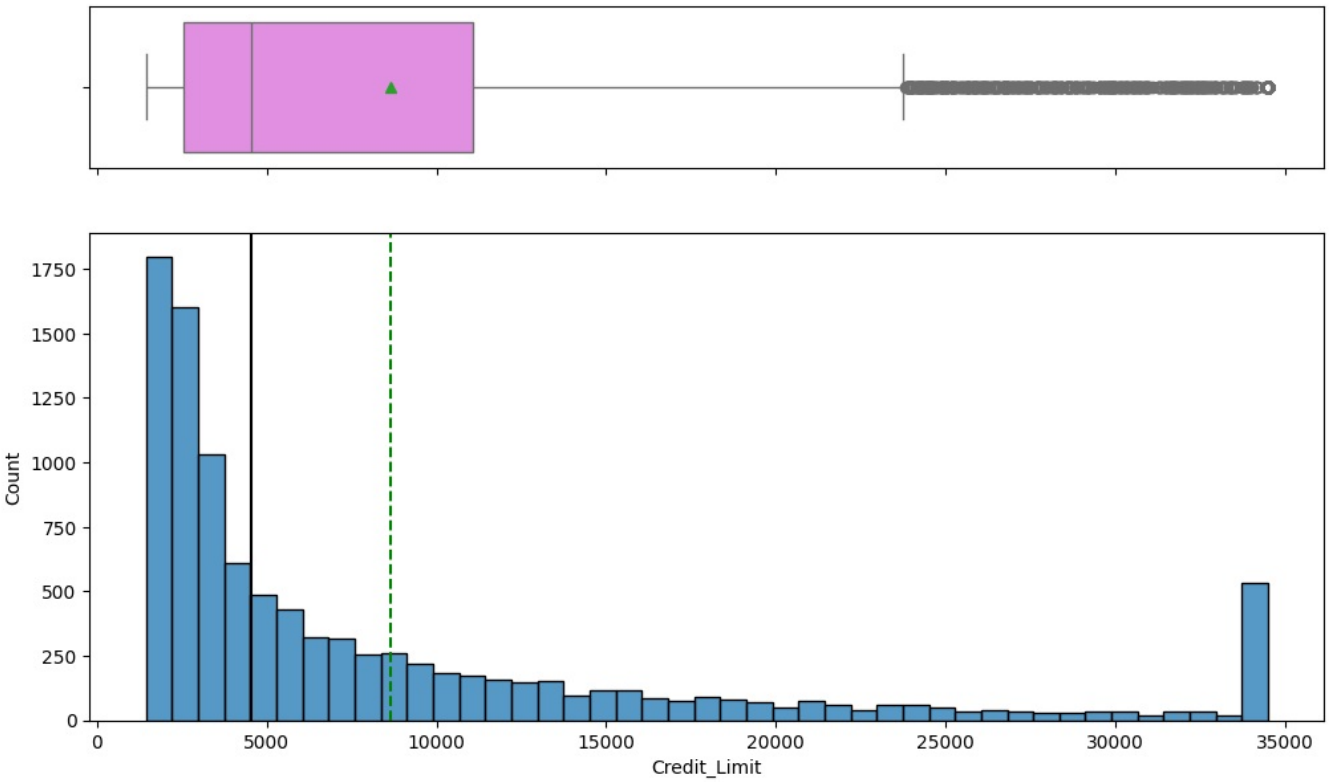
- The number of months inactive is skewed to the right, with a few outliers past the min an max whiskers. The majority of months is 3 for all customers.

In [45]:
```
# Observation on Contacts_Count_12_mon

histogram_boxplot(df,'Contacts_Count_12_mon')
```



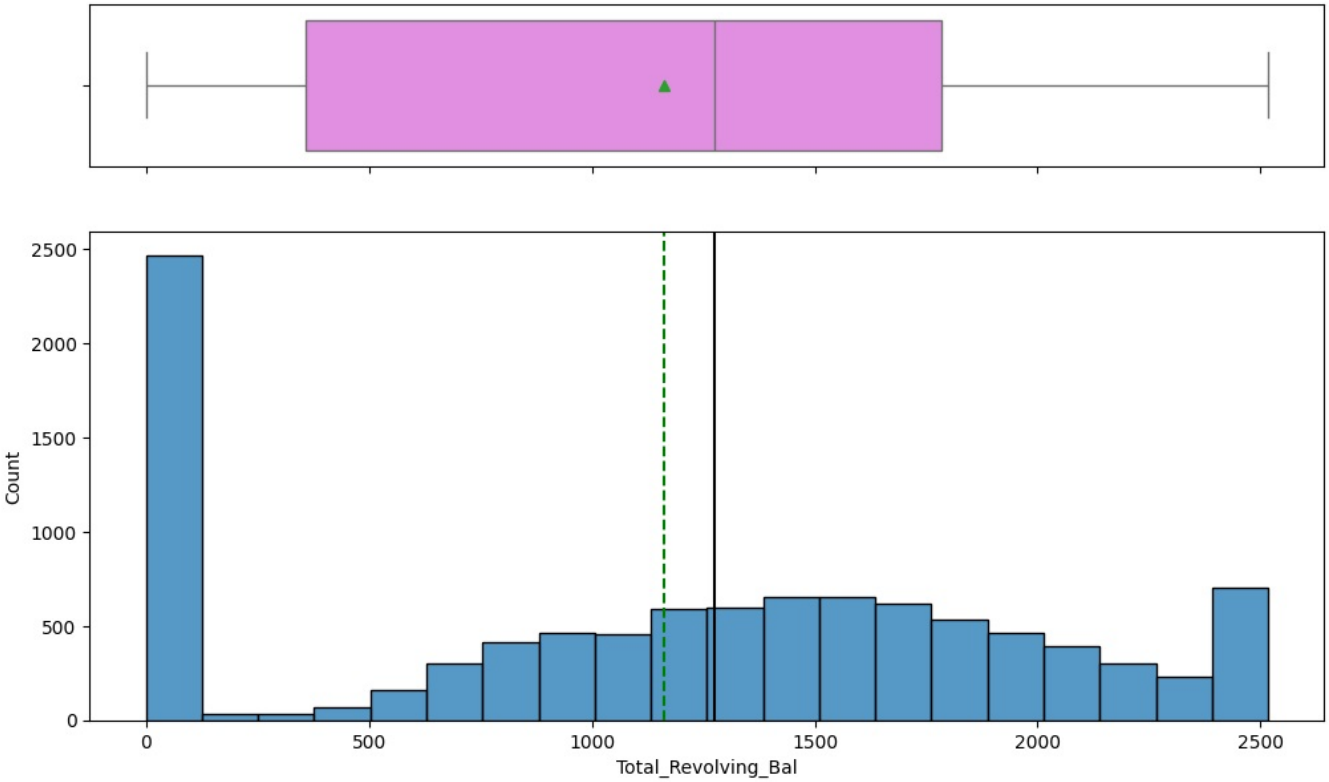
- The data seems to be normall distributed, with no skewness. Most customers do not really contact the bank more than 2-3 times a year.

In [46]:
```
# Observation on Credit_Limit

histogram_boxplot(df,'Credit_Limit')
```



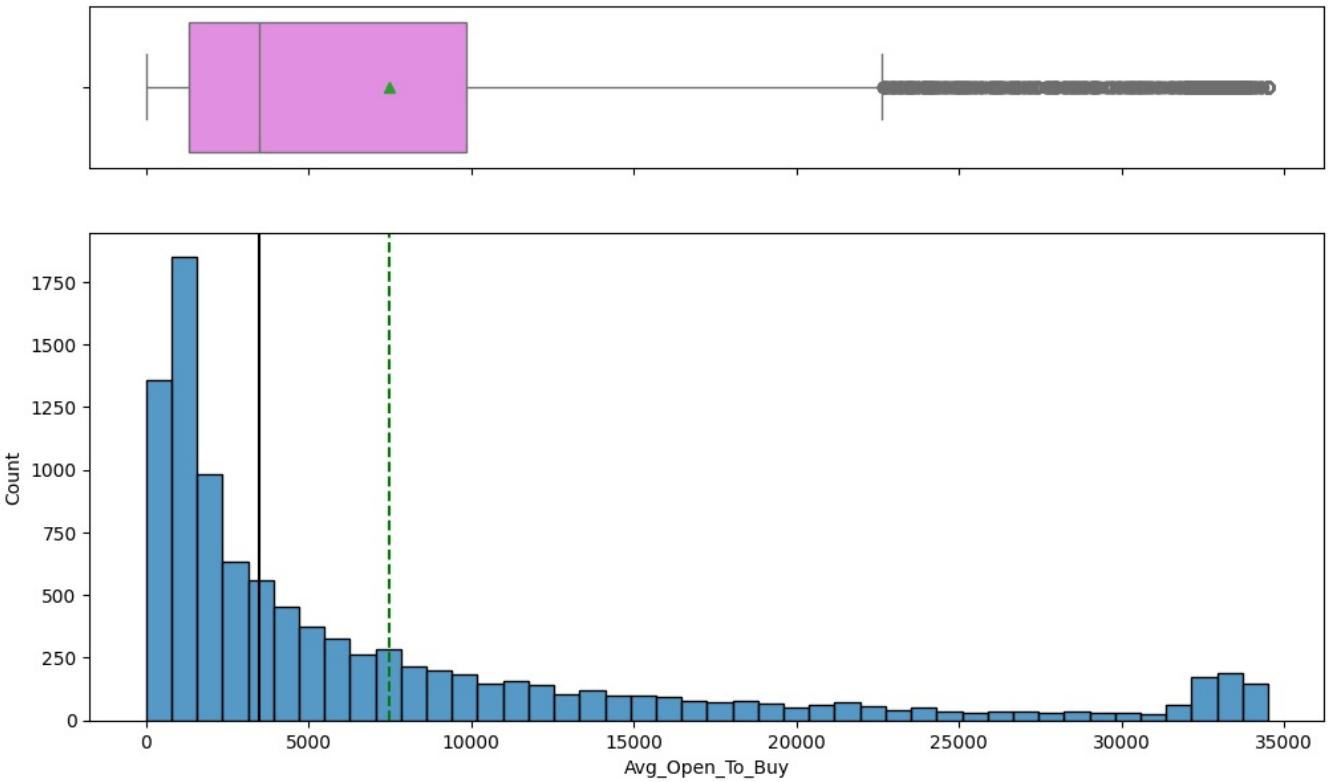
- The data is heavily skewed to the right, with the majority of the customers having a credit limit of less than 5,000 dollars. There are also a lot of outliers that are high valued.

In [47]:
```
# Observation on Total_Revolving_Bal

histogram_boxplot(df,'Total_Revolving_Bal')
```



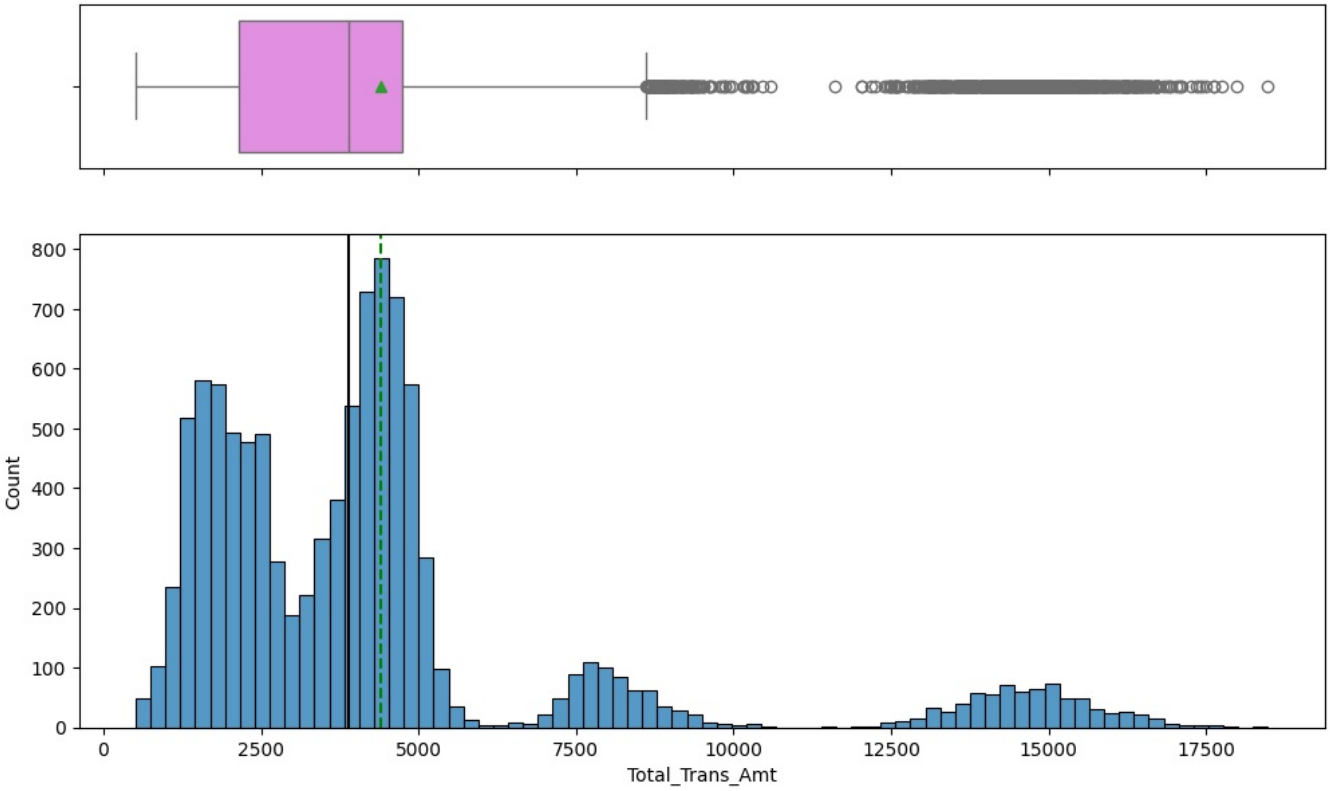
- The data is distributed normally, with a significant amount of customers having no money being carreid over to the next month.

In [48]:
```
# Observation on Avg_Open_To_Buy

histogram_boxplot(df,'Avg_Open_To_Buy')
```



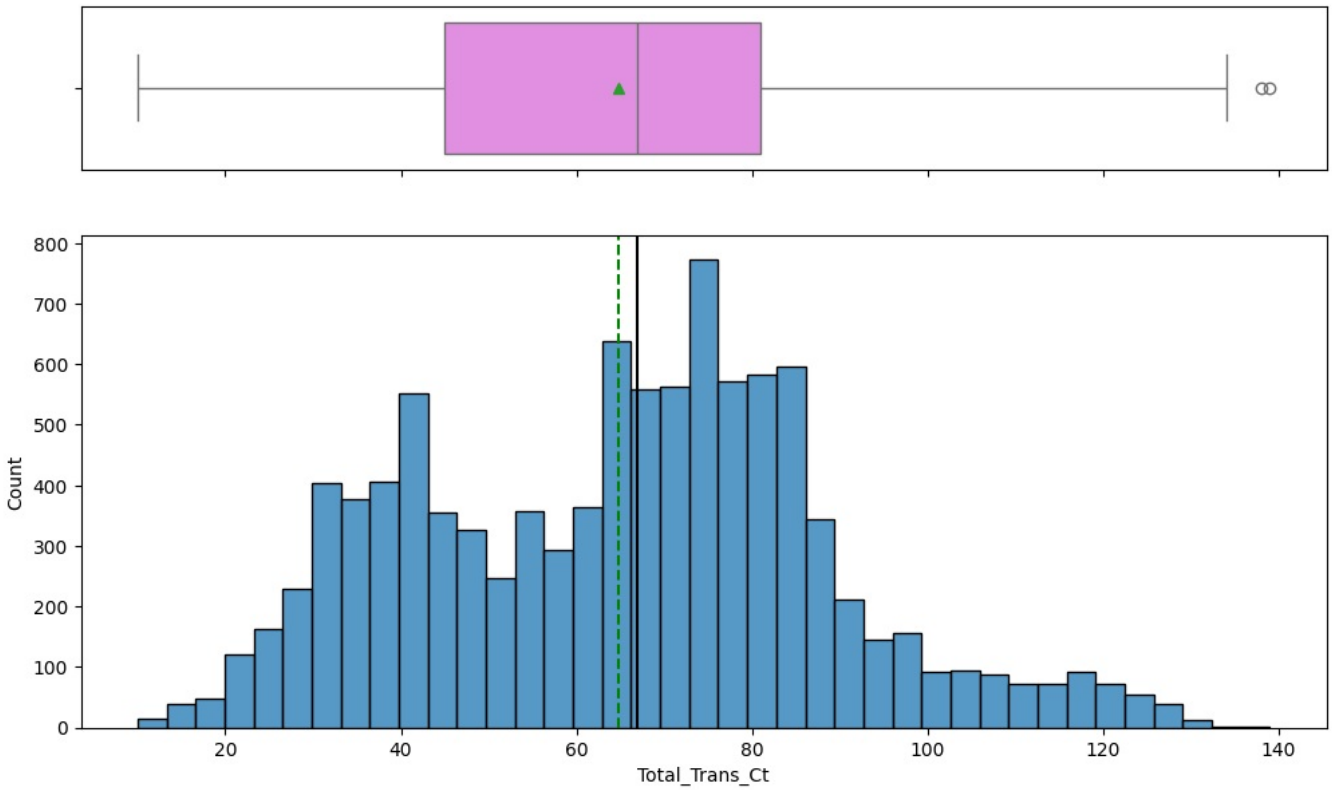
- The data is skewed to the right, this indicates that the customers do not have a good habit of saving their money, where the majority of them do not pass the 3,000 dollar threshold.

In [49]:
```
# Observation on Total_Trans_Amt

histogram_boxplot(df,'Total_Trans_Amt')
```



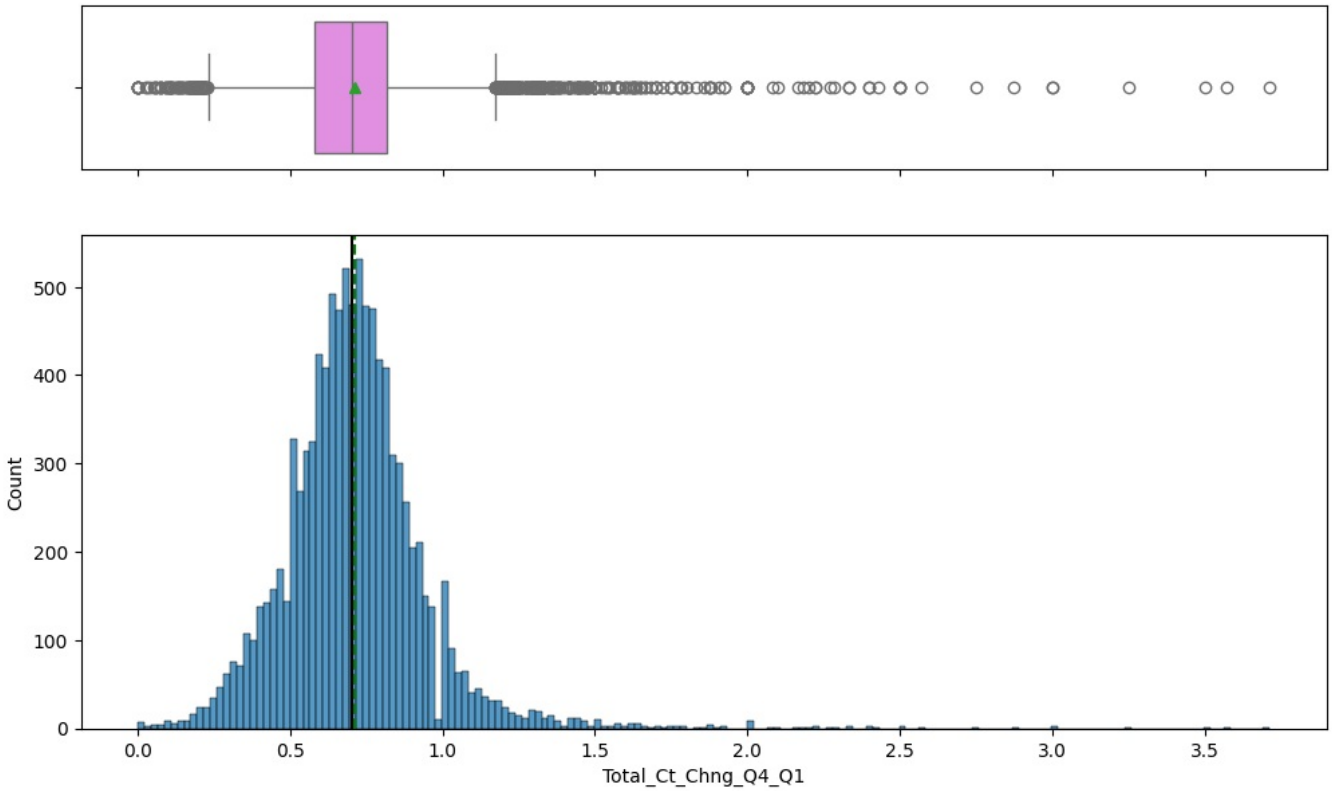
- The total transaction amount is somewhat skewed to the right, most customers spend no more than 5,000 dollars a year which is pretty low. However, for most customers average of around 40,0000 dollars, it isn't that surprising.

In [50]:
```
# Observation on Total_Trans_Ct

histogram_boxplot(df,'Total_Trans_Ct')
```



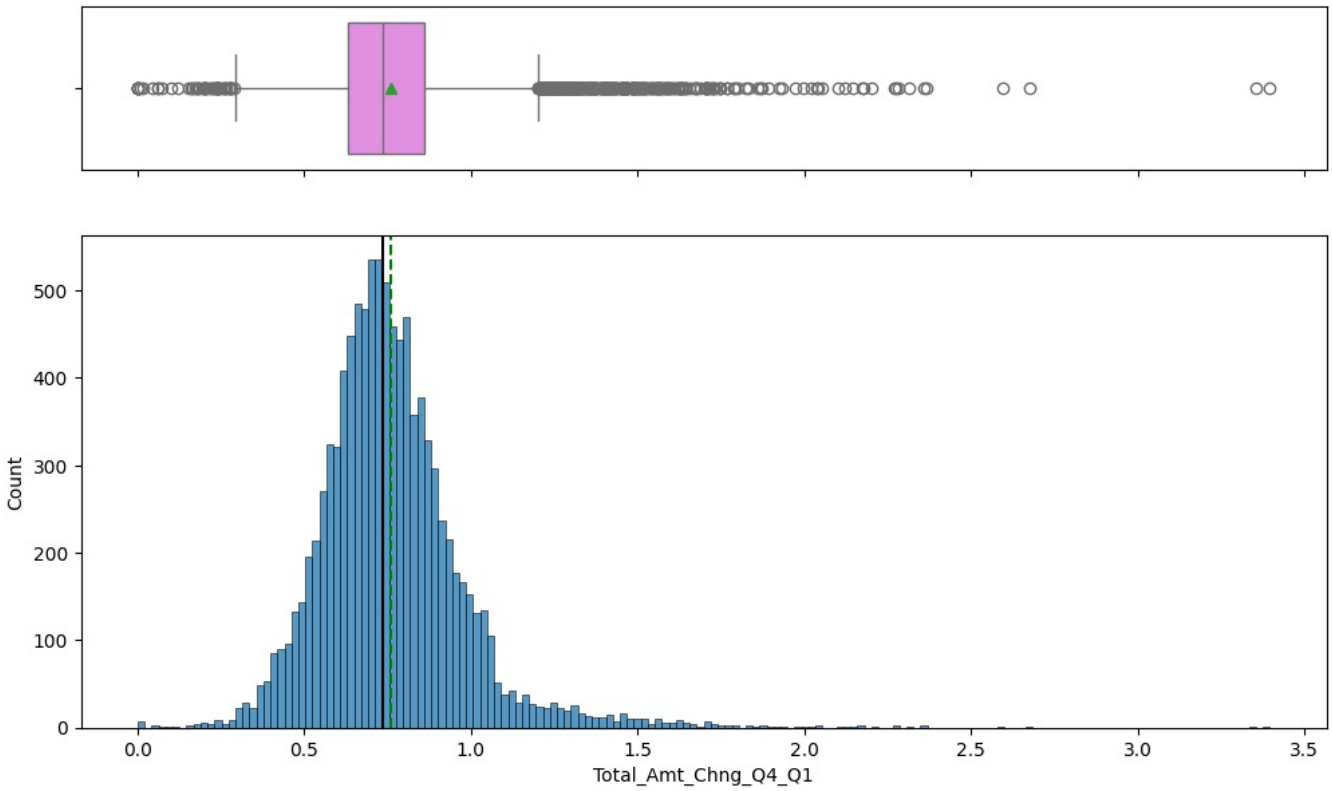
- The data is normally distributed, with the average customer having roughly 65 transaction over the past year.

In [51]:
```
# Observation on Total_Ct_Chng_Q4_Q1

histogram_boxplot(df,'Total_Ct_Chng_Q4_Q1')
```



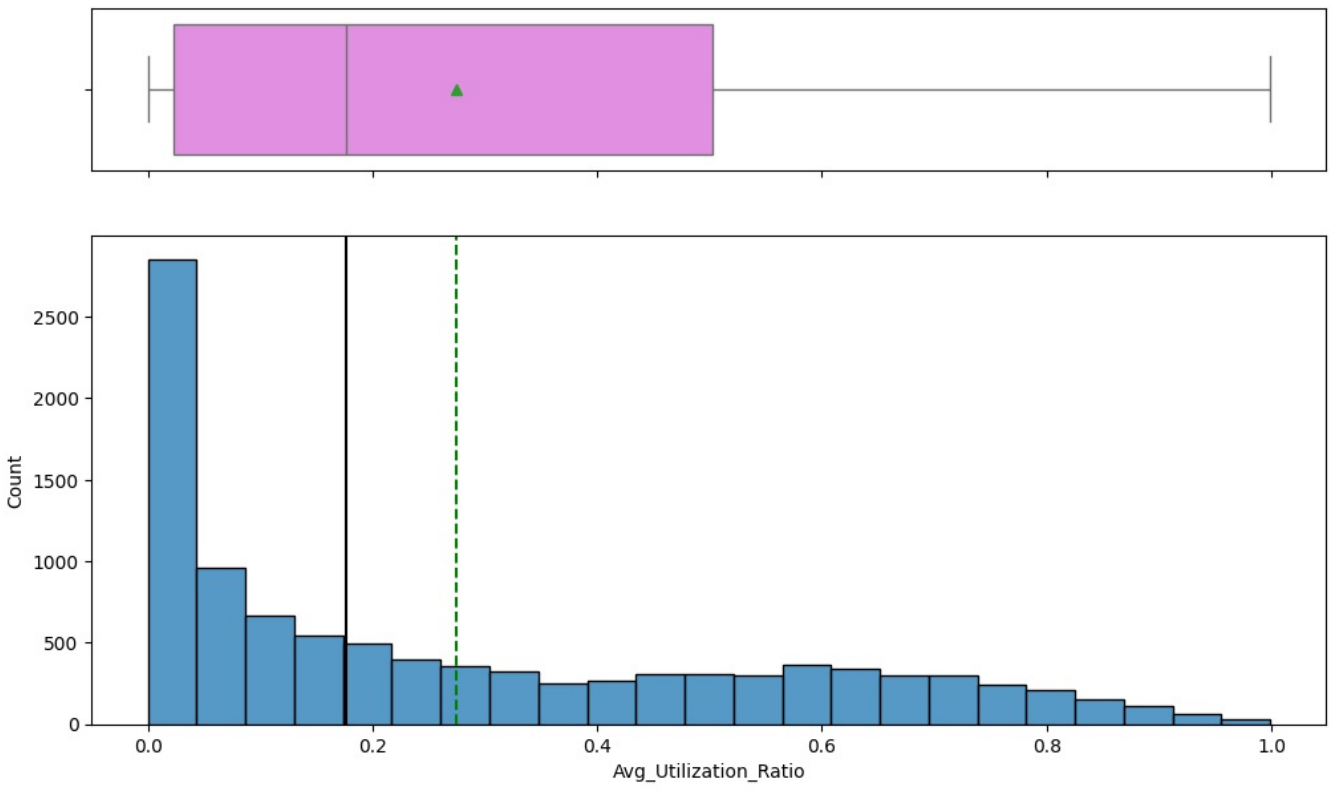
- The data is heavily skewed to the right, the average ratio of the total transaction count between quarter 4 and 1 is rougly 0.7.

In [52]:
```
# Observation on Total_Amt_Chng_Q4_Q1

histogram_boxplot(df,'Total_Amt_Chng_Q4_Q1')
```



- The data is also skewed to the right, the average ratio amount of transactions made in the 4th and 1st quarter is roughly 7.5.

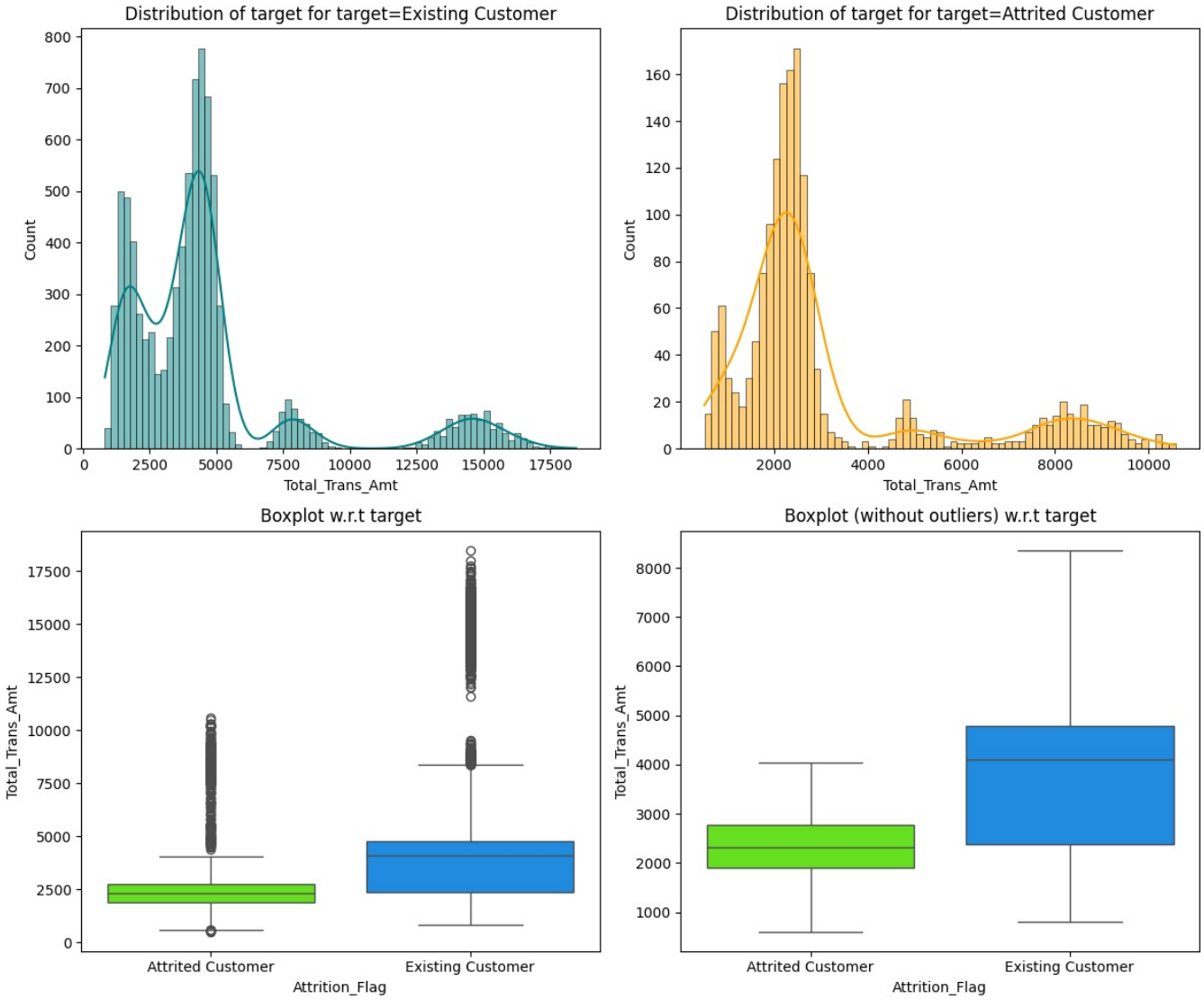
In [53]:
```
# Observation on Avg_Utilization_Ratio

histogram_boxplot(df,'Avg_Utilization_Ratio')
```



- The average utilization ratio is somewhat skewed to the right, signifying that most customers do not spend much of their available credit. Also, the majority of customers do not spend their credit at all having 0 count.

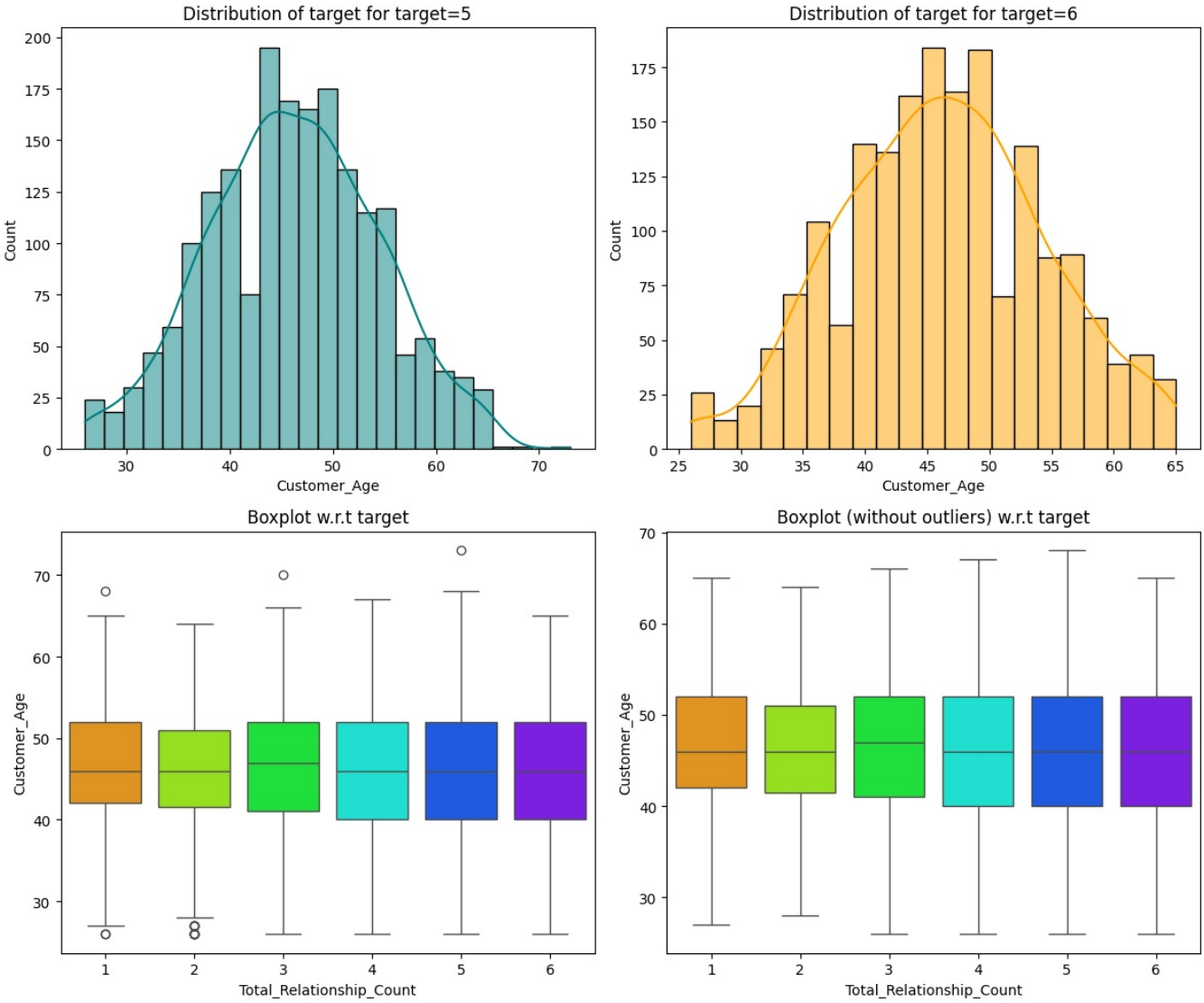
Bivariate Analysis

In [54]:
```
# Relationship Between Total_Trans_Amt & Attrition_Flag

distribution_plot_wrt_target(df, "Total_Trans_Amt", "Attrition_Flag")
```



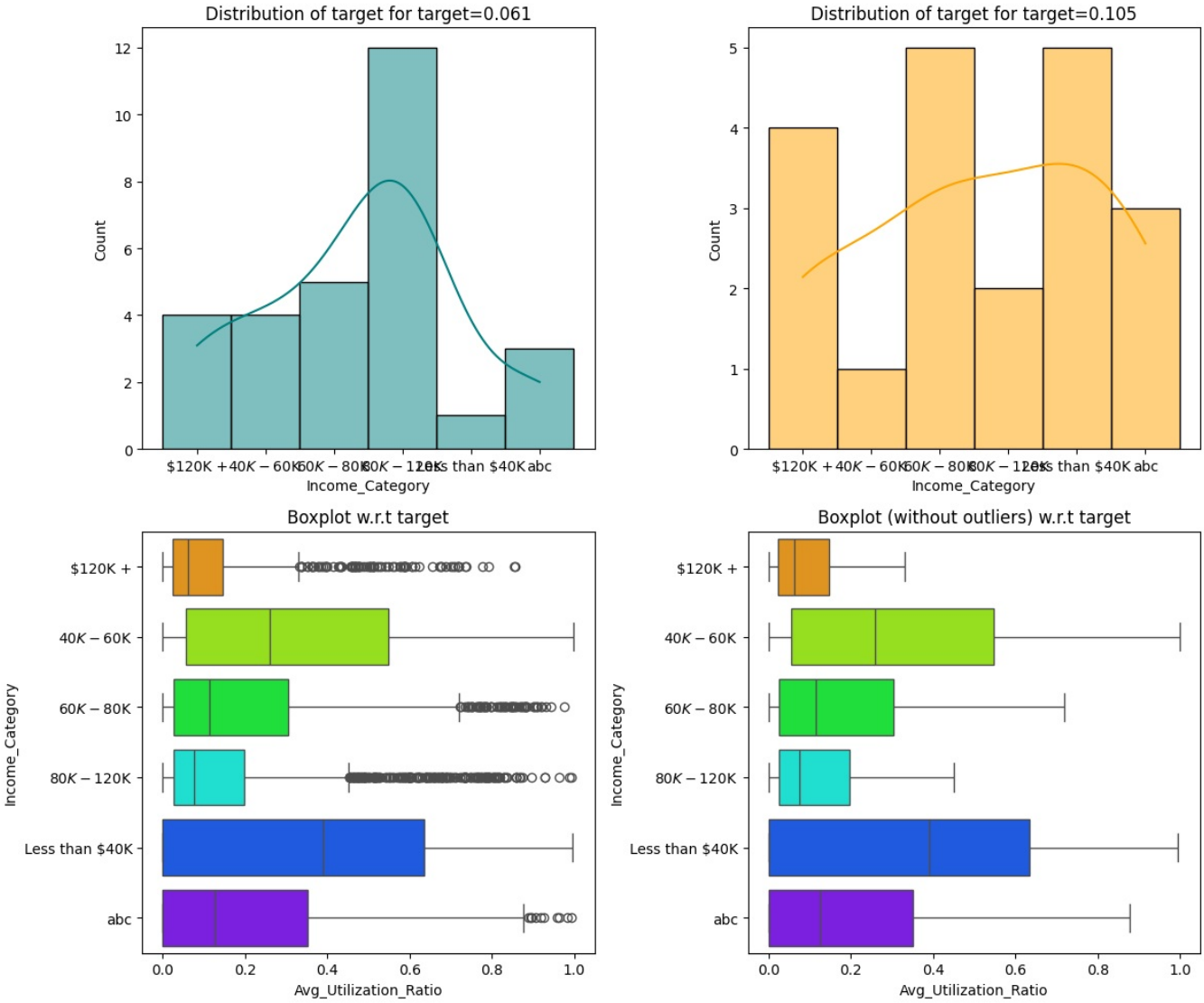
- From the above distribution plot, it seems like the existing customers have a higher total transaction amount than those who are attrited. This can provide insights into the relationship between the customers spending behavior and their retention.
- The data also is heavily skewed to the right.

In [55]:
```
# Relationship between Customer_Avg & Total_Relationship_Count

distribution_plot_wrt_target(df, "Customer_Age", "Total_Relationship_Count")
```



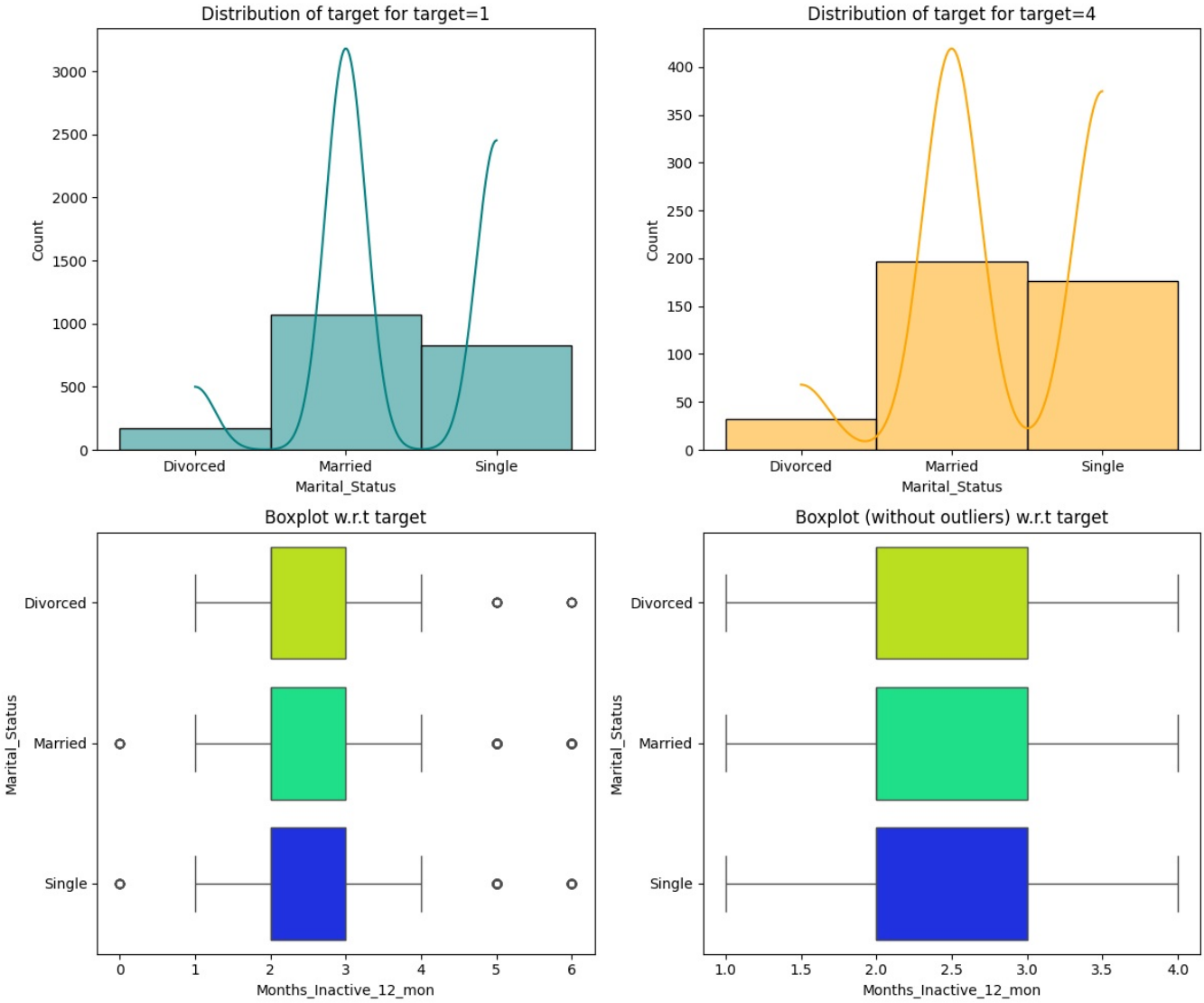
- The data is normally distributed, therefore there does not seem to be any correlation on whether younger or older customers have more or fewer products they hold with the bank.

In [56]:
```
# Relationship between Income_Category & Avg_Utilizaiton_Ratio

distribution_plot_wrt_target(df, "Income_Category", "Avg_Utilization_Ratio")
```



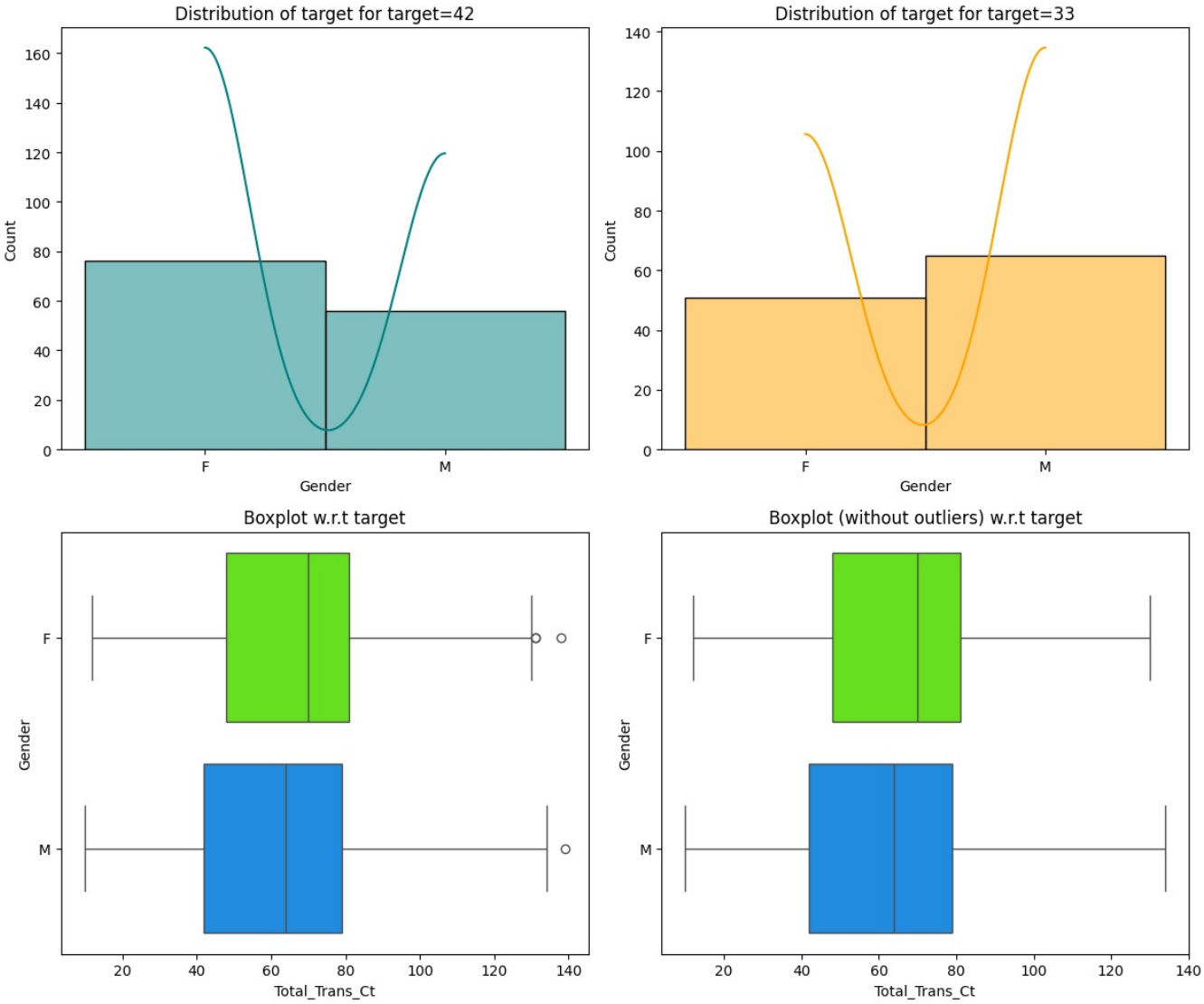
- Ignoring the 'abc' data which will be removed, the data does not seem to have any skewness. However, the majority of the customers have an income of less than 40,000 dollars, and some have a little more than that, ranging from 60,000 dollars.

In [57]:
```
# Relationship between Martial_Status & Months_Inactive_12_mon

distribution_plot_wrt_target(df, "Marital_Status", "Months_Inactive_12_mon")
```



- There does not seem to be any impact on customer engagement or inactivity as the data is normally distributed.

In [58]:
```
# Relationship between Gender vs. Total_Trans_Ct

distribution_plot_wrt_target(df, "Gender", "Total_Trans_Ct")
```

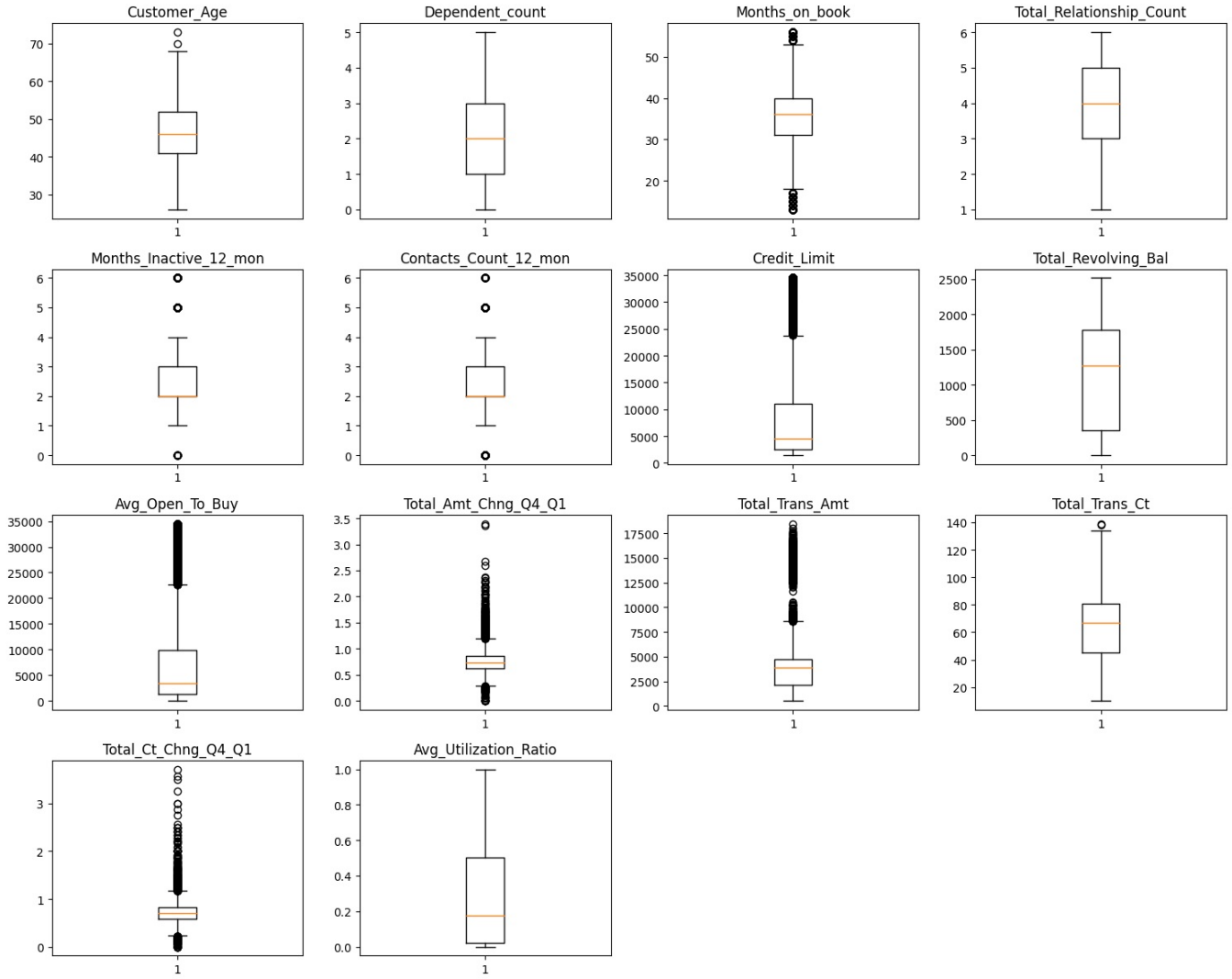


- There does not seem to be any significant differences in how often a male or female customer makes their transactions as the statistics are normally distributed.

Data Pre-processing

Outlier Detection

In [59]:
```
# outlier detection using boxplots
numeric_columns = df.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(15, 12))

for i, variable in enumerate(numeric_columns):
    plt.subplot(4, 4, i + 1)
    plt.boxplot(df[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```



- There seem to be outliers in multiple columns
  - The `Months_on_book` columns outliers do not need to be treated
  - The `Credit_Limit` column outliers seem to be normal as there can be some customers who exceed their credit limit from the average.
  - The `Avg_Open_To_Buy` columns outliers do not need to be treated.
  - The rest of the columns that contain outliers do not need to be treated as there can be instances of customers having or not having the same range of income or balance in their cards.

Missing value imputation

Education_Level

In [60]:
```
# Count the percentage of missing values (if any) in each column

pd.DataFrame({'Count':df.isnull().sum()[df.isnull().sum()>0],'Percentage':(df.isnull().sum()[df.isnull().sum()>
```

Out[60]:

	Count	Percentage
Education_Level	1519	14.999506
Marital_Status	749	7.396070

- As seen above, there are two columns with missing data, the 'Education_Level' column has 15% missing data, whilst the Martial_Status has 7%.

In [61]:
```
# Checking the missing values further
df.Education_Level.value_counts(normalize=True, dropna=False)
```

Out[61]:

Education_Level	proportion
Graduate	0.308877
High School	0.198776
NaN	0.149995
Uneducated	0.146835
College	0.100030
Post-Graduate	0.050953
Doctorate	0.044534

dtype: float64

The categories contain a mixed set of proportions, the mode is the 'Graduate'.

In [62]:
```
df.loc[df['Education_Level'].isnull()==True]
```

Out[62]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_o
6	Existing Customer	51	M	4	NaN	Married	$120K +	Gold	
11	Existing Customer	65	M	1	NaN	Married	$40K - 60K	Blue	
15	Existing Customer	44	M	4	NaN	NaN	$80K - 120K	Blue	
17	Existing Customer	41	M	3	NaN	Married	$80K - 120K	Blue	
23	Existing Customer	47	F	4	NaN	Single	Less than $40K	Blue	
...	...	...	...	...	...	...	...	...	
10090	Existing Customer	36	F	3	NaN	Married	$40K - 60K	Blue	
10094	Existing Customer	59	M	1	NaN	Single	$60K - 80K	Blue	
10095	Existing Customer	46	M	3	NaN	Married	$80K - 120K	Blue	
10118	Attrited Customer	50	M	1	NaN	NaN	$80K - 120K	Blue	
10123	Attrited Customer	41	M	2	NaN	Divorced	$40K - 60K	Blue	

1519 rows × 20 columns

- It seems like there is no correlation or other columns that correlate on why the values are missing from the column, this could be due to the vast amount of missing data.

Therefore we can replace all the 'NaN' values with 'Unknown' in the Education_Level column

In [63]:
```
# Check if 'Unknown' is already a category
if 'Unknown' not in df['Education_Level'].cat.categories:
    df['Education_Level'] = df['Education_Level'].cat.add_categories('Unknown')

# Now you can fill NaN values with 'Unknown'
df['Education_Level'].fillna('Unknown', inplace=True)
```

In [64]:
```
df['Education_Level'].unique()
```

Out[64]:
```
['High School', 'Graduate', 'Uneducated', 'Unknown', 'College', 'Post-Graduate', 'Doctorate']
Categories (7, object): ['College', 'Doctorate', 'Graduate', 'High School', 'Post-Graduate',
                         'Uneducated', 'Unknown']
```

There is now an Unkown category to replace all the NaN values

In [65]:
```
df.isnull().sum()
```

Out[65]:

	0
Attrition_Flag	0
Customer_Age	0
Gender	0
Dependent_count	0
Education_Level	0
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

The Education_Level category missing values are now treated.

Marital_Status

In [66]:
```
df.isnull().sum()
```

Out[66]:

	0
Attrition_Flag	0
Customer_Age	0
Gender	0
Dependent_count	0
Education_Level	0
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

In [67]:
```
# Checking the missing values further
df.Marital_Status.value_counts(normalize=True, dropna=False)
```

Out[67]:

Marital_Status	proportion
Married	0.462822
Single	0.389355
NaN	0.073961
Divorced	0.073862

dtype: float64

- There does not seem to be much of a proportion of the amount of missing data compared to the other categories in the column, although it still needs to be treated.

In [68]:
```
# See if there is any correlation for the missing values

df.loc[df['Marital_Status'].isnull()==True]
```

Out[68]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_o
3	Existing Customer	40	F	4	High School	NaN	Less than $40K	Blue	
7	Existing Customer	32	M	0	High School	NaN	$60K - 80K	Silver	
10	Existing Customer	42	M	5	Uneducated	NaN	$120K +	Blue	
13	Existing Customer	35	M	3	Graduate	NaN	$60K - 80K	Blue	
15	Existing Customer	44	M	4	Unknown	NaN	$80K - 120K	Blue	
...	...	...	...	...	...	...	...	...	
10070	Existing Customer	47	M	3	High School	NaN	$80K - 120K	Silver	
10100	Existing Customer	39	M	2	Graduate	NaN	$60K - 80K	Silver	
10101	Existing Customer	42	M	2	Graduate	NaN	$40K - 60K	Blue	
10118	Attrited Customer	50	M	1	Unknown	NaN	$80K - 120K	Blue	
10125	Attrited Customer	30	M	2	Graduate	NaN	$40K - 60K	Blue	

749 rows × 20 columns

In [69]:
```
# Replacing NaN with Unknown for the Marital_Status

# Check if 'Unknown' is already a category, and only add it if it's not.
if 'Unknown' not in df['Marital_Status'].cat.categories:
    df['Marital_Status'] = df['Marital_Status'].cat.add_categories('Unknown')

# Now fill the missing values
df['Marital_Status'].fillna('Unknown', inplace=True)
```

In [70]:
```
df['Marital_Status'].unique()
```

Out[70]:
```
['Married', 'Single', 'Unknown', 'Divorced']
Categories (4, object): ['Divorced', 'Married', 'Single', 'Unknown']
```

- Unknown category within the column has been added to transfer the NaN values

In [71]:
```
df.isnull().sum()
```

Out[71]:

	0
Attrition_Flag	0
Customer_Age	0
Gender	0
Dependent_count	0
Education_Level	0
Marital_Status	0
Income_Category	0
Card_Category	0
Months_on_book	0
Total_Relationship_Count	0
Months_Inactive_12_mon	0
Contacts_Count_12_mon	0
Credit_Limit	0
Total_Revolving_Bal	0
Avg_Open_To_Buy	0
Total_Amt_Chng_Q4_Q1	0
Total_Trans_Amt	0
Total_Trans_Ct	0
Total_Ct_Chng_Q4_Q1	0
Avg_Utilization_Ratio	0

dtype: int64

There is now no missing values to be treated within the dataset.

In [72]:
```
df.head()
```

Out[72]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_bo
0	Existing Customer	45	M	3	High School	Married	$60K - 80K	Blue	
1	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	
2	Existing Customer	51	M	3	Graduate	Married	$80K - 120K	Blue	
3	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	
4	Existing Customer	40	M	3	Uneducated	Married	$60K - 80K	Blue	

Model Building

Data Preparation for model building

In [73]:
```
df.head()
```

Out[73]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_bo
0	Existing Customer	45	M	3	High School	Married	$60K - 80K	Blue	
1	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	
2	Existing Customer	51	M	3	Graduate	Married	$80K - 120K	Blue	
3	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	
4	Existing Customer	40	M	3	Uneducated	Married	$60K - 80K	Blue	

In [74]:
```
# Creating dummy variables throughout the data set:

df = pd.get_dummies(df, drop_first=True)
```

In [75]:
```
# Make the dummy variables 1s and 0s instead of true and false

df = df.replace({False: 0, True: 1})
```

In [76]:
```
df.head()
```

Out[76]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_Li
0	45	3	39	5	1	3	1269
1	49	5	44	6	1	2	825
2	51	3	36	4	1	0	341
3	40	4	34	3	4	1	331
4	40	3	21	5	1	0	471

5 rows × 33 columns

In [77]:
```
# Removing 'Income_Category_abc' column

df = df.drop(columns=['Income_Category_abc'])
```

- Now all the necessary variables are one-hot encoded, the data is now ready to be split.

In [78]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 32 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Customer_Age                      10127 non-null  int64  
 1   Dependent_count                   10127 non-null  int64  
 2   Months_on_book                    10127 non-null  int64  
 3   Total_Relationship_Count          10127 non-null  int64  
 4   Months_Inactive_12_mon            10127 non-null  int64  
 5   Contacts_Count_12_mon             10127 non-null  int64  
 6   Credit_Limit                      10127 non-null  float64
 7   Total_Revolving_Bal               10127 non-null  int64  
 8   Avg_Open_To_Buy                   10127 non-null  float64
 9   Total_Amt_Chng_Q4_Q1              10127 non-null  float64
 10  Total_Trans_Amt                   10127 non-null  int64  
 11  Total_Trans_Ct                    10127 non-null  int64  
 12  Total_Ct_Chng_Q4_Q1               10127 non-null  float64
 13  Avg_Utilization_Ratio             10127 non-null  float64
 14  Attrition_Flag_Existing Customer  10127 non-null  int64  
 15  Gender_M                          10127 non-null  int64  
 16  Education_Level_Doctorate         10127 non-null  int64  
 17  Education_Level_Graduate          10127 non-null  int64  
 18  Education_Level_High School       10127 non-null  int64  
 19  Education_Level_Post-Graduate     10127 non-null  int64  
 20  Education_Level_Uneducated        10127 non-null  int64  
 21  Education_Level_Unknown           10127 non-null  int64  
 22  Marital_Status_Married            10127 non-null  int64  
 23  Marital_Status_Single             10127 non-null  int64  
 24  Marital_Status_Unknown            10127 non-null  int64  
 25  Income_Category_$40K - $60K       10127 non-null  int64  
 26  Income_Category_$60K - $80K       10127 non-null  int64  
 27  Income_Category_$80K - $120K      10127 non-null  int64  
 28  Income_Category_Less than $40K    10127 non-null  int64  
 29  Card_Category_Gold                10127 non-null  int64  
 30  Card_Category_Platinum            10127 non-null  int64  
 31  Card_Category_Silver              10127 non-null  int64  
dtypes: float64(5), int64(27)
memory usage: 2.5 MB
```

In [79]:
```
from sklearn.model_selection import train_test_split

# Assuming `df` is your DataFrame and 'Attrition_Flag' is the target column
X = df.drop(columns=['Attrition_Flag_Existing Customer', 'Credit_Limit'])  # Features
y = df['Attrition_Flag_Existing Customer']  # Target

# Step 1: Split the data into training + validation and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Step 2: Split the training + validation set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.25, random_state=42, st
```

In [80]:
```
y.value_counts(1)
```

Out[80]:

Attrition_Flag_Existing Customer	proportion
1	0.83934
0	0.16066

dtype: float64

In [81]:
```
y_test.value_counts(1)
```

Out[81]:

Attrition_Flag_Existing Customer	proportion
1	0.839585
0	0.160415

dtype: float64

Model evaluation criterion

The nature of predictions made by the classification model will translate as follows:

- True positives (TP) are failures correctly predicted by the model.
- False negatives (FN) are real failures in a generator where there is no detection by model.
- False positives (FP) are failure detections in a generator where there is no failure.

Which metric to optimize?

- We need to choose the metric which will ensure that the maximum number of generator failures are predicted correctly by the model.
- We would want Recall to be maximized as greater the Recall, the higher the chances of minimizing false negatives.
- We want to minimize false negatives because if a model predicts that a machine will have no failure when there will be a failure, it will increase the maintenance cost.

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 [236…
```
# defining a function to compute different metrics to check performance of a classification model built using s
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 [237…
```
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 with original data

Sample code for model building with original data

In [84]:
```
# Import necessary libraries
!pip install xgboost
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier, AdaBoostClassifier, GradientBoostingCla
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

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

# Appending models into the list
models.append(("Decision Tree", DecisionTreeClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1)))
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("AdaBoost", AdaBoostClassifier(random_state=1)))
models.append(("XGBoost", XGBClassifier(random_state=1)))

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))
```

```
Requirement already satisfied: xgboost in /root/.local/lib/python3.10/site-packages (2.0.3)
Requirement already satisfied: numpy in /root/.local/lib/python3.10/site-packages (from xgboost) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)

Training Performance:

Decision Tree: 1.0
Random forest: 1.0
Bagging: 0.9966660129437145
AdaBoost: 0.9825455971759168
XGBoost: 1.0

Validation Performance:

Decision Tree: 0.96
Random forest: 0.9858823529411764
Bagging: 0.9688235294117648
AdaBoost: 0.9764705882352941
XGBoost: 0.9852941176470589
```

- As seen from the Training and Validation performances above, scores all turned out to be high, with the lowest score being the decision tree in the validation section.

Decision Tree Model

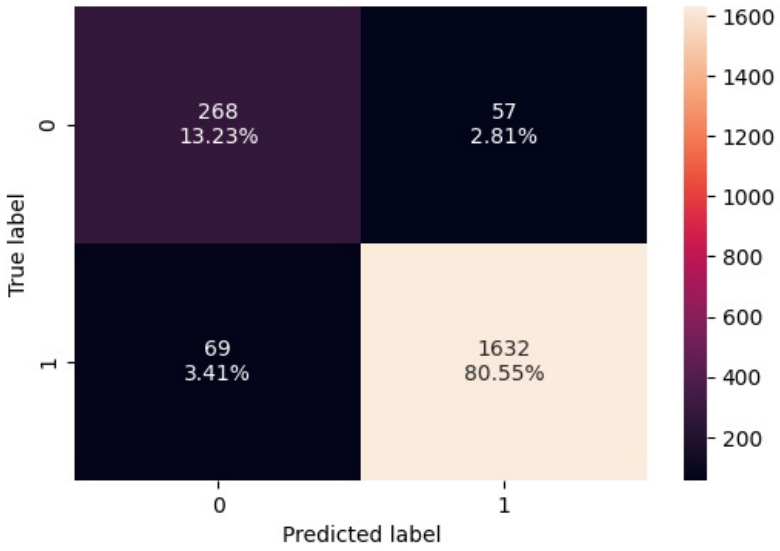
In [85]:
```
dtree = DecisionTreeClassifier(criterion='gini',class_weight={0:0.17,1:0.83},random_state=1)
```

In [86]:
```
dtree.fit(X_train, y_train)
```

Out[86]:
```
DecisionTreeClassifier(class_weight={0: 0.17, 1: 0.83}, random_state=1)
```

In [87]:
```
confusion_matrix_sklearn(dtree, X_test, y_test)
```

In [87]:
```
confusion_matrix_sklearn(dtree, X_test, y_test)
```



Confusion Matrix -

- The customer renounced their credit card and the model predicted correctly that the customer would attrite : True Positive (observed=1,predicted=1)
  - The model predicted 80% correct which is a pretty good statistic.
- The customer didn't renounce their credit card and the model predicted the customer would attrite : False Positive (observed=0,predicted=1)
- TThe customer didn't renounce their credit card and the model predicted the customer will not attrite: True Negative (observed=0,predicted=0)
- The customer didn't renounce their credit card and the model predicted that the customer won't: False Negative (observed=1,predicted=0)

In [88]:
```
dtree_model_train_perf=model_performance_classification_sklearn(dtree, X_train, y_train)
print("Training performance \n",dtree_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [89]:
```
dtree_model_test_perf=model_performance_classification_sklearn(dtree, X_test, y_test)
print("Testing performance \n",dtree_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.937808  0.959436   0.966252  0.962832
```

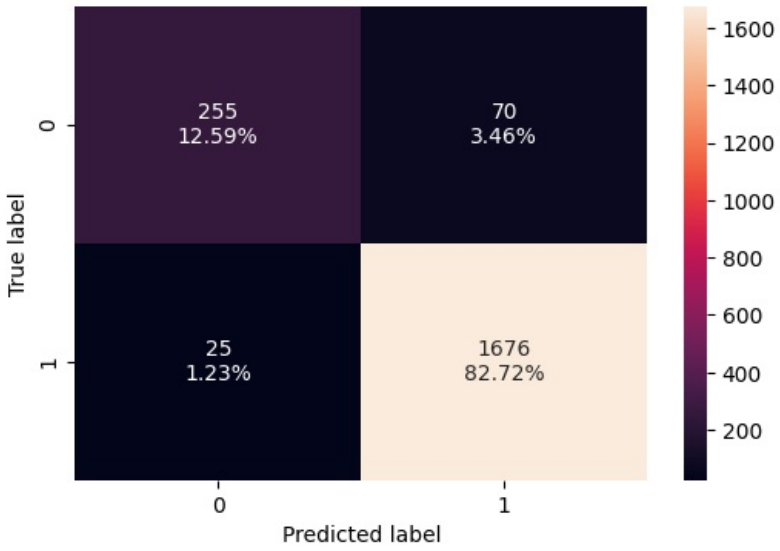
- From the recall scores on both the training and testing performances, the decision tree model is performing well, with a slight drop in recall on the test data which can be renounced to overfitting.

Random Forest

In [90]:
```
rf = RandomForestClassifier(random_state=1)
rf.fit(X_train,y_train)
```

Out[90]:
```
RandomForestClassifier(random_state=1)
```

In [91]:
```
confusion_matrix_sklearn(rf,X_test,y_test)
```



In [92]:
```
rf_model_train_perf=model_performance_classification_sklearn(rf,X_train,y_train)
print("Training performance \n",rf_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [93]:
```
rf_model_test_perf=model_performance_classification_sklearn(rf,X_test,y_test)
print("Testing performance \n",rf_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision       F1
0   0.95311  0.985303   0.959908  0.97244
```

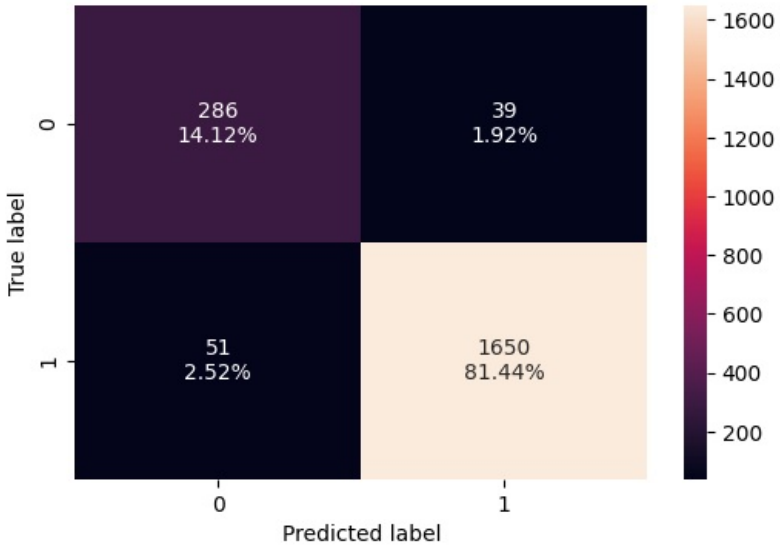
- The random forest performs well in the training scores, although a slight drop in comparison to the testing performances.

Bagging Classifier

In [94]:
```
bagging = BaggingClassifier(random_state=1)
bagging.fit(X_train,y_train)
```

Out[94]:
```
BaggingClassifier(random_state=1)
```

In [95]:
```
confusion_matrix_sklearn(bagging, X_test, y_test)
```



In [96]:
```
bagging_model_train_perf=model_performance_classification_sklearn(bagging, X_train, y_train)
print("Training performance \n",bagging_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision       F1
0  0.996049  0.996666   0.998624  0.997644
```

In [97]:
```
bagging_model_test_perf=model_performance_classification_sklearn(bagging, X_test, y_test)
print("Testing performance \n",bagging_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.955577  0.970018   0.976909  0.973451
```

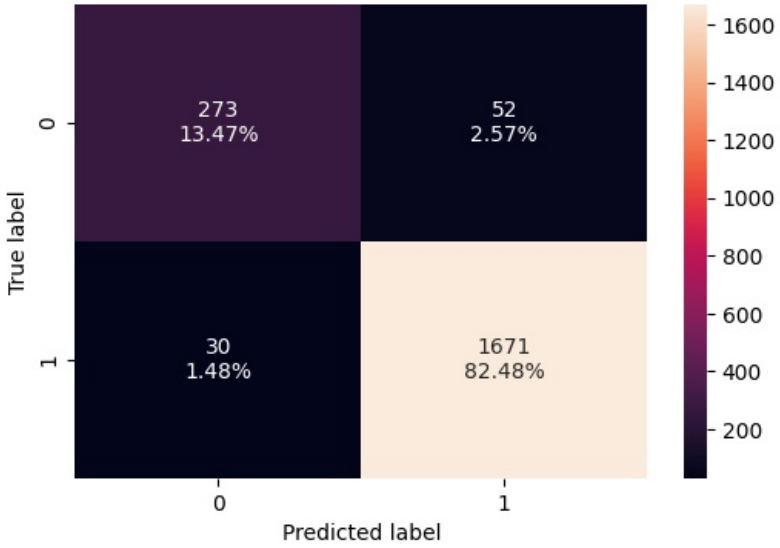
- Compared to the random forests training scores, the bagging classifiers' training scores are very slightly down, but the testing performances improved.

AdaBoost Classifier

In [98]:
```
abc = AdaBoostClassifier(random_state=1)
abc.fit(X_train,y_train)
```

Out[98]:
```
AdaBoostClassifier(random_state=1)
```

In [99]:
```
confusion_matrix_sklearn(abc, X_test, y_test)
```



In [100…
```
abc_model_train_perf=model_performance_classification_sklearn(abc, X_train, y_train)
print("Training performance \n",abc_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.963457  0.982546    0.97414  0.978325
```

In [101…
```
abc_model_test_perf=model_performance_classification_sklearn(abc, X_test, y_test)
print("Testing performance \n",abc_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.959526  0.982363    0.96982  0.976051
```

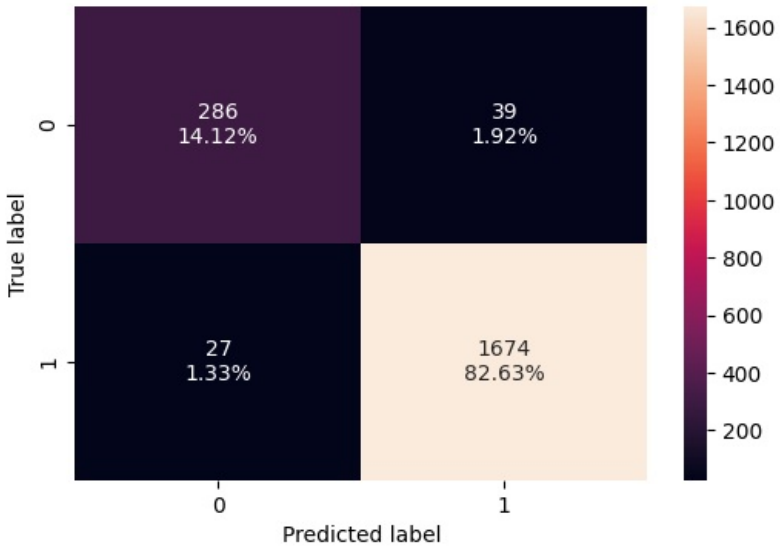
- Both the training & testing performances are extremely identical to each other in terms of numbers, therefore there is no over/underfitting in the AdaBosst Classifier model.

XGBoost Classifier

In [102…
```
xgb = XGBClassifier(random_state=1,eval_metric='logloss')
xgb.fit(X_train, y_train)
```

Out[102]:
```
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=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, 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=Non
e,
```

In [103…
```
confusion_matrix_sklearn(xgb, X_test, y_test)
```



In [104…
```
xgb_model_train_perf=model_performance_classification_sklearn(xgb, X_train, y_train)
print("Training performance \n",xgb_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [105…
```
xgb_model_test_perf=model_performance_classification_sklearn(xgb, X_test, y_test)
print("Testing performance \n",xgb_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.967423  0.984127   0.977233  0.980668
```

- Perfect traning performances with a slight decrease in the traning.

Model Building with Oversampled data

In [214…
```
!pip install imblearn
from imblearn.over_sampling import SMOTE # Import the SMOTE class

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

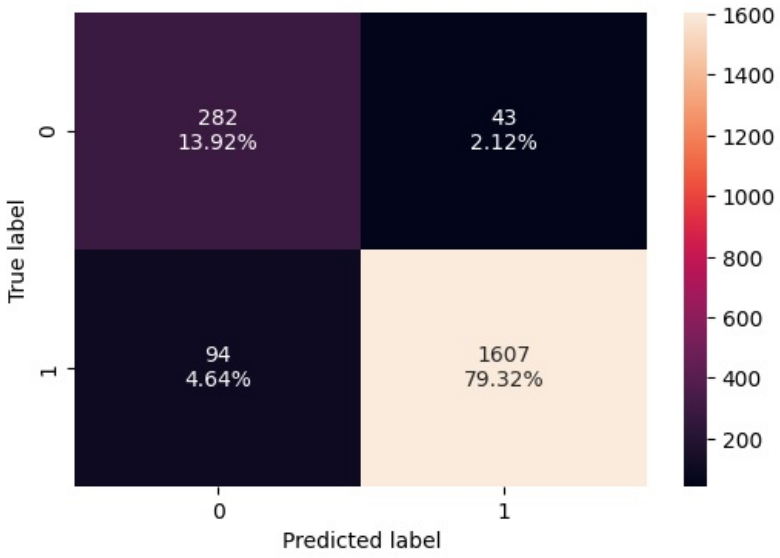
```
Requirement already satisfied: imblearn in /usr/local/lib/python3.10/dist-packages (0.0)
Requirement already satisfied: imbalanced-learn in /root/.local/lib/python3.10/site-packages (from imblearn) (0
.10.1)
Requirement already satisfied: numpy>=1.17.3 in /root/.local/lib/python3.10/site-packages (from imbalanced-lear
n->imblearn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->
imblearn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in /root/.local/lib/python3.10/site-packages (from imbalance
d-learn->imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn-
>imblearn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced
-learn->imblearn) (3.5.0)
```

Decision Tree Model

In [215…
```
dtree_over = DecisionTreeClassifier(criterion='gini',class_weight={0:0.17,1:0.83},random_state=1)
```

In [216…
```
dtree_over = dtree_over.fit(X_train_over, y_train_over)
```

In [217…
```
confusion_matrix_sklearn(dtree_over, X_test, y_test)
```



- There seems to be a slight decrease in the number of True Positives compared to the original tree models.

In [218…
```
dtree_over_model_train_perf=model_performance_classification_sklearn(dtree_over, X_train_over, y_train_over)
print("Training performance \n",dtree_over_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [219…
```
dtree_over_model_test_perf=model_performance_classification_sklearn(dtree_over, X_test, y_test)
print("Testing performance \n",dtree_over_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.932379  0.944738   0.973939  0.959117
```

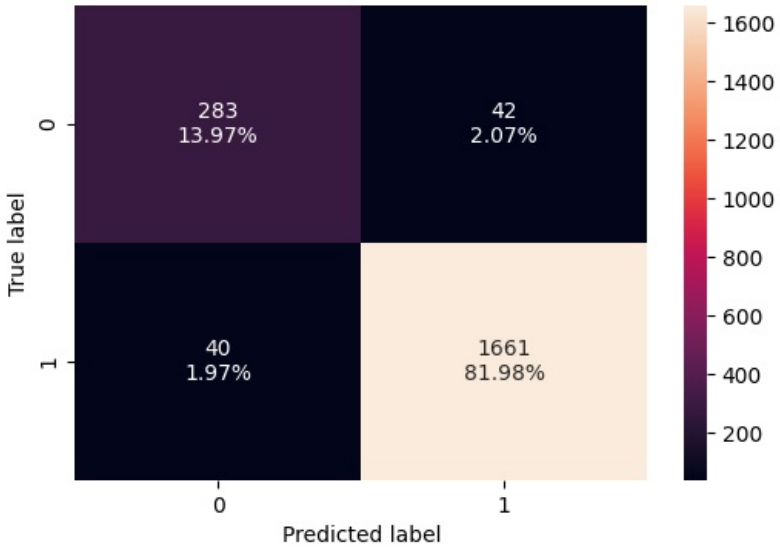
- Perfect traning performance whilst a good amount of drop from the recall score in the traningshows that the model is overfitting by a minority.

Random Forest

In [220…
```
rf_over = RandomForestClassifier(random_state=1)
rf_over.fit(X_train_over,y_train_over)
```

Out[220]:
```
RandomForestClassifier(random_state=1)
```

In [221…
```
confusion_matrix_sklearn(rf_over, X_test, y_test)
```



- There is about 14% of True Negatives being predicted which is a good sign that the model is predicting the correct negatives and positives.

In [222…
```
rf_over_model_train_perf=model_performance_classification_sklearn(rf_over, X_train_over, y_train_over)
```

```
print("Training performance \n",rf_over_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [223…
```
rf_over_model_test_perf=model_performance_classification_sklearn(rf_over, X_test, y_test)
print("Testing performance \n",rf_over_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.959526  0.976484   0.975338  0.975911
```

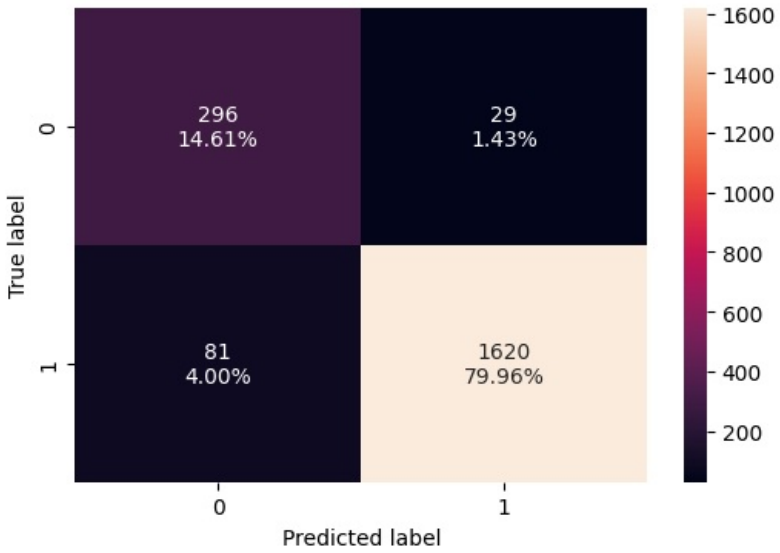
- Similar to the Decision Tree Oversampled data model, the traning performances are perfect with the testing being a bit under the par.

Bagging Classifier

In [224…
```
bagging_over = BaggingClassifier(random_state=1)
bagging_over.fit(X_train_over,y_train_over)
```

Out[224]:
```
BaggingClassifier(random_state=1)
```

In [225…
```
confusion_matrix_sklearn(bagging_over, X_test, y_test)
```



- There is an increase in the True Negatives with a slight decrease in the True Positives compared to the previous oversampled data models.

In [226…
```
bagging_over_model_train_perf=model_performance_classification_sklearn(bagging_over, X_train_over, y_train_over
print("Training performance \n",bagging_over_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.997058  0.995097   0.999016  0.997052
```

In [227…
```
bagging_over_model_test_perf=model_performance_classification_sklearn(bagging_over, X_test, y_test)
print("Testing performance \n",bagging_over_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.945706  0.952381   0.982414  0.967164
```

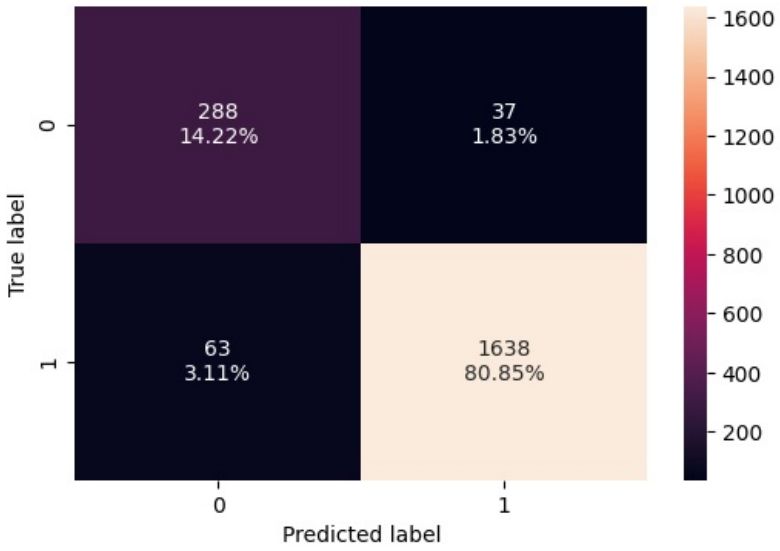
- In this instance, the baggin classifer model with oversampled data is just a fraction off from being completely out performed. With that being said, this model makes it less overfitting from both performances being slightly more close to each other.

AdaBoost Classifier

In [228…
```
adaboost_over = AdaBoostClassifier(random_state=1)
adaboost_over.fit(X_train_over,y_train_over)
```

Out[228]:
```
AdaBoostClassifier(random_state=1)
```

In [229…
```
confusion_matrix_sklearn(adaboost_over, X_test, y_test)
```



- The AdaBoost Classifier model with oversampled data has shown the best True Positive & Negative metrics from the previous oversampled data models seen.

In [230…
```
adaboost_over_model_train_perf=model_performance_classification_sklearn(adaboost_over, X_train_over, y_train_ov
print("Training performance \n",adaboost_over_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.966072  0.962542   0.969386  0.965952
```

In [231…
```
adaboost_over_model_test_perf=model_performance_classification_sklearn(adaboost_over, X_test, y_test)
print("Testing performance \n",adaboost_over_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.950642  0.962963    0.97791  0.970379
```

- From the traning and testing performances almost being identical with each other, the model is shown to be well fit with each other and generalized.

XGBoost Classifier

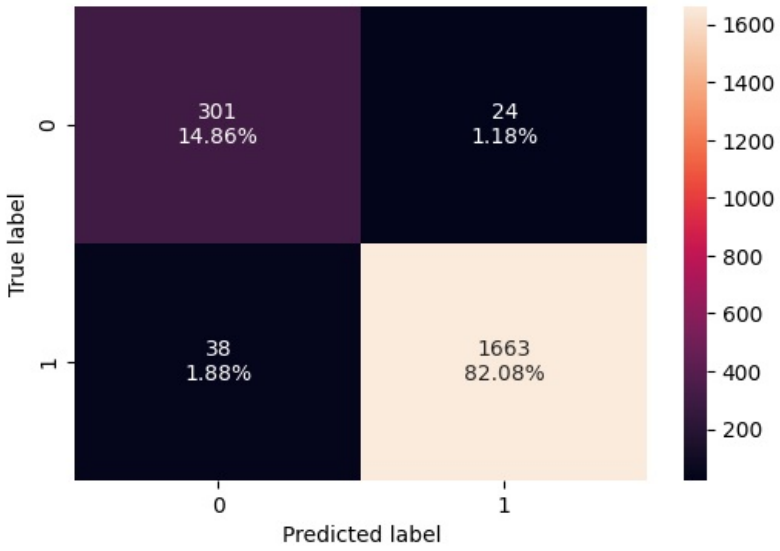
In [232…
```
xgb_over = XGBClassifier(random_state=1,eval_metric='logloss')
xgb_over.fit(X_train_over, y_train_over)
```

Out[232]:

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=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, 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=Non
e,
```

In [233…
```
confusion_matrix_sklearn(xgb_over, X_test, y_test)
```



- From all the previous oversampled data models, the xgboost classifier has shown the best true predicted results.

In [234…
```
xgb_over_model_train_perf=model_performance_classification_sklearn(xgb_over, X_train_over, y_train_over)
print("Training performance \n",xgb_over_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [235…
```
xgb_over_model_test_perf=model_performance_classification_sklearn(xgb_over, X_test, y_test)
print("Testing performance \n",xgb_over_model_test_perf)
```

```
Testing performance
    Accuracy   Recall  Precision      F1
0  0.969398  0.97766   0.985774  0.9817
```

- Compared to the other models, this model is slightly overfitting but still contains a good generalization.

Model Building with Undersampled data

In [192…
```
!pip install imblearn
from imblearn.under_sampling import RandomUnderSampler

# Random undersampler for under sampling the data
rus = RandomUnderSampler(random_state=1, sampling_strategy=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

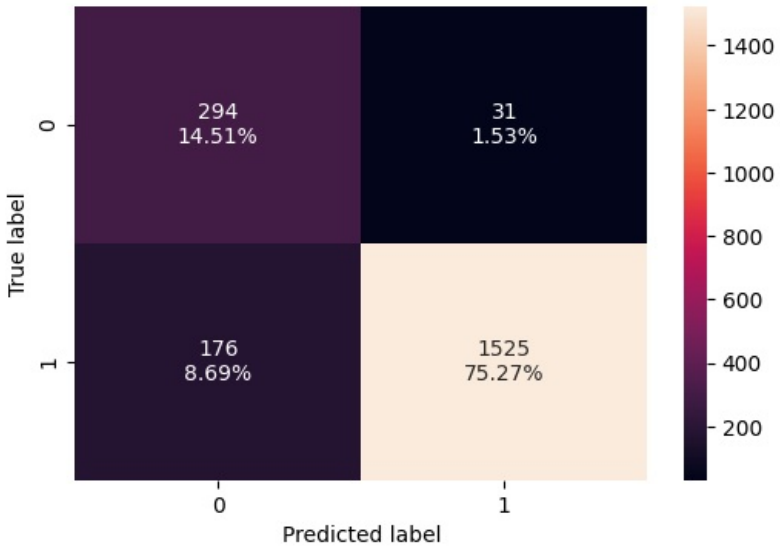
```
Requirement already satisfied: imblearn in /usr/local/lib/python3.10/dist-packages (0.0)
Requirement already satisfied: imbalanced-learn in /root/.local/lib/python3.10/site-packages (from imblearn) (0
.10.1)
Requirement already satisfied: numpy>=1.17.3 in /root/.local/lib/python3.10/site-packages (from imbalanced-lear
n->imblearn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn->
imblearn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in /root/.local/lib/python3.10/site-packages (from imbalance
d-learn->imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn-
>imblearn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced
-learn->imblearn) (3.5.0)
```

Decision Tree Model

In [193…
```
dtree_un = DecisionTreeClassifier(criterion='gini',class_weight={0:0.17,1:0.83},random_state=1)
```

In [194…
```
dtree_un = dtree_un.fit(X_train_un, y_train_un)
```

In [195…
```
confusion_matrix_sklearn(dtree_un, X_test, y_test)
```



- There is a slight decrease in the number of True Positives in this model compared to the previous ones seen.

In [196…
```
dtree_un_model_train_perf=model_performance_classification_sklearn(dtree_un, X_train_un, y_train_un)
print("Training performance \n",dtree_un_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [197…
```
dtree_un_model_test_perf=model_performance_classification_sklearn(dtree_un, X_test, y_test)
print("Testing performance \n",dtree_un_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.897828  0.896531   0.980077  0.936445
```

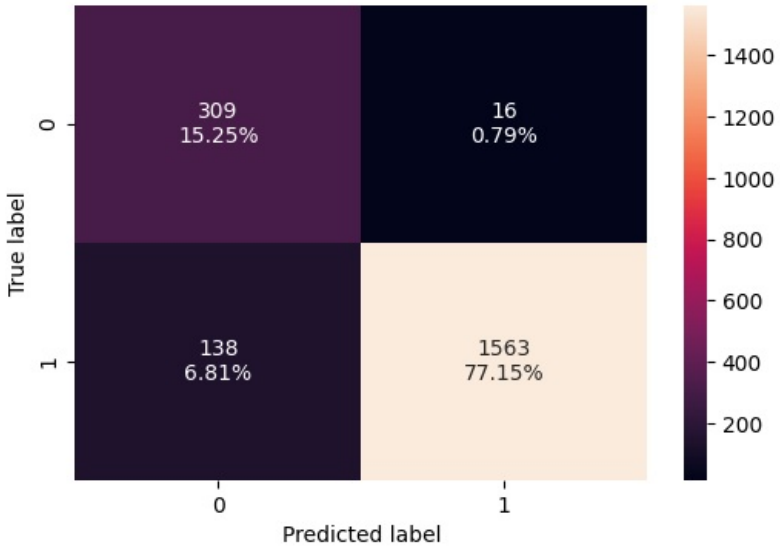
- The training performances seem to be perfect in terms of the scores, however, the accuracy and recall scores for the testing performance are the lowest that has been seen throughout all the models.

Random Forest

In [198…
```
random_forest_un = RandomForestClassifier(random_state=1)
random_forest_un.fit(X_train_un,y_train_un)
```

Out[198]:
```
RandomForestClassifier(random_state=1)
```

In [199…
```
confusion_matrix_sklearn(random_forest_un, X_test, y_test)
```



- The random forest model does slightly better than the decision tree model, there are still signs of overfitting within the model but it is somewhat generalized.

In [200…
```
random_forest_un_model_train_perf=model_performance_classification_sklearn(random_forest_un, X_train_un, y_trai
```

```
print("Training performance \n",random_forest_un_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [201…
```
random_forest_un_model_test_perf=model_performance_classification_sklearn(random_forest_un, X_test, y_test)
print("Testing performance \n",random_forest_un_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.923988  0.918871   0.989867  0.953049
```

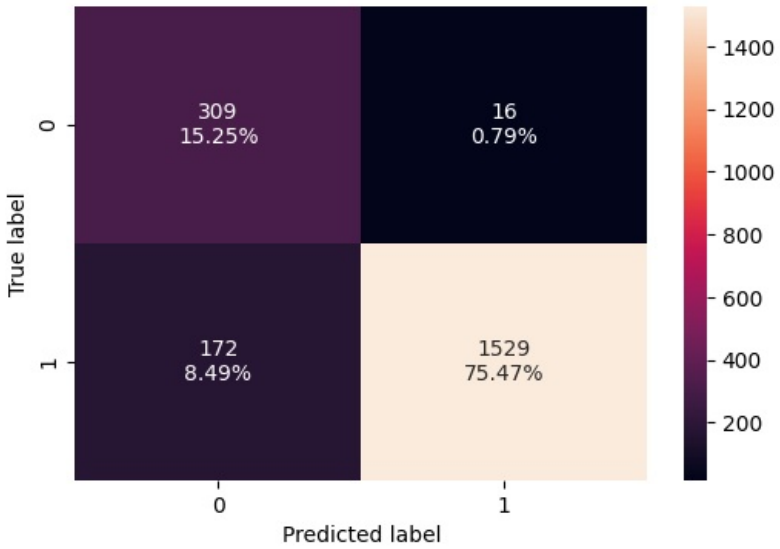
- The random forest model does slightly better than the decision tree model, there are still signs of overfitting within the model but it is somewhat generalized.

Bagging Classifier

In [202…
```
bagging_un = BaggingClassifier(random_state=1)
bagging_un.fit(X_train_un,y_train_un)
```

Out[202]:
```
BaggingClassifier(random_state=1)
```

In [203…
```
confusion_matrix_sklearn(bagging_un, X_test, y_test)
```



- The true positives has decreased slightly but the true negatives percentage has stayed the same from the previous model.

In [204…
```
bagging_un_model_train_perf=model_performance_classification_sklearn(bagging_un, X_train_un, y_train_un)
print("Training performance \n",bagging_un_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.991291  0.985656   0.996891  0.991242
```

In [205…
```
bagging_un_model_test_perf=model_performance_classification_sklearn(bagging_un, X_test, y_test)
print("Testing performance \n",bagging_un_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.907206  0.898883   0.989644  0.942083
```

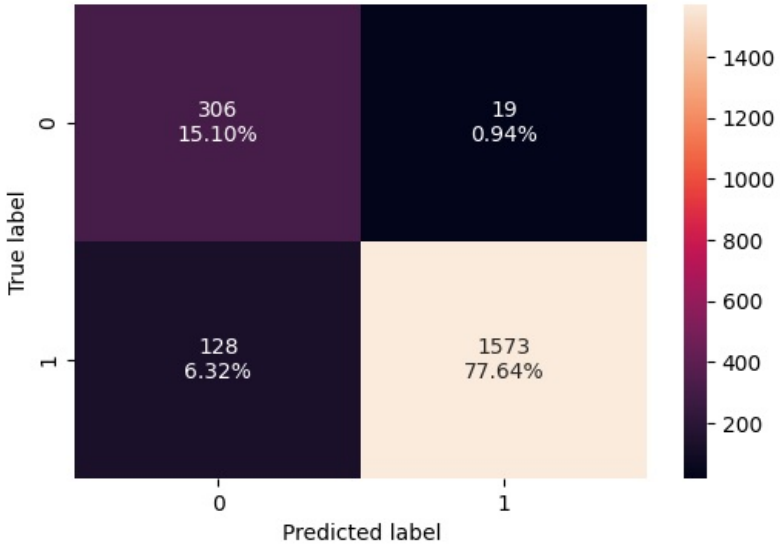
- From the performances in this mode, the scores still indicate that there is still overfitting but not so much compared to the previous random forest model.

AdaBoost Classifier

In [206…
```
adaboost_un = AdaBoostClassifier(random_state=1)
adaboost_un.fit(X_train_un,y_train_un)
```

Out[206]:
```
AdaBoostClassifier(random_state=1)
```

In [207…
```
confusion_matrix_sklearn(adaboost_un, X_test, y_test)
```



There is not much that has changed from the previous models with this adaboost confusion matrix.

In [208…
```
adaboost_un_model_train_perf=model_performance_classification_sklearn(adaboost_un, X_train_un, y_train_un)
print("Training performance \n",adaboost_un_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.951332  0.939549   0.962225  0.950752
```

In [209…
```
adaboost_un_model_test_perf=model_performance_classification_sklearn(adaboost_un, X_test, y_test)
print("Testing performance \n",adaboost_un_model_test_perf)
```

```
Testing performance
    Accuracy   Recall  Precision       F1
0  0.927443  0.92475   0.988065  0.95536
```

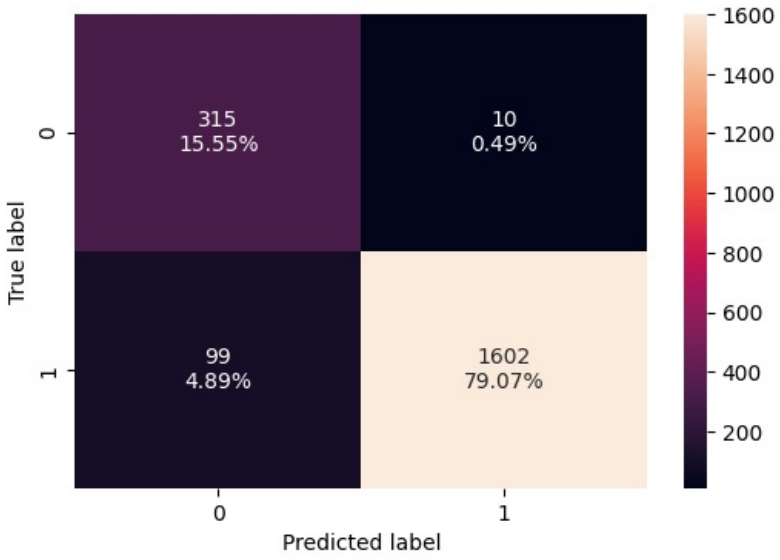
- This model shows a better generalization compared to the rest of the undersampled data models with the training and testing scores having a much lesser gap between them.

XGBoost Classifier

In [210…
```
xgb_un = XGBClassifier(random_state=1,eval_metric='logloss')
xgb_un.fit(X_train_un, y_train_un)
```

Out[210]:
```
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=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, 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=Non
e,
```

In [211…
```
confusion_matrix_sklearn(xgb_un, X_test, y_test)
```



- Just like the previous models, the XGBoost Classifier model has always performed the best with a good increase in true positives in the undersampled models.

In [212…
```
xgb_un_model_train_perf=model_performance_classification_sklearn(xgb_un, X_train_un, y_train_un)
print("Training performance \n",xgb_un_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
```

In [213…
```
xgb_un_model_test_perf=model_performance_classification_sklearn(xgb_un, X_test, y_test)
print("Testing performance \n",xgb_un_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.946199  0.941799   0.993797  0.967099
```

- The is an almost exact balance between the precision scores for the testing and training performances, although compared to the adaboost model it is slightly more overfitted.

Hyperparameter Tuning

Sample Parameter Grids

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

- For Gradient Boosting:

```
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],
}
```

- For Adaboost:

```
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),
    ],
}
```

- For Bagging Classifier:

```
param_grid = {
    'max_samples': [0.8,0.9,1],
    'max_features': [0.7,0.8,0.9],
    'n_estimators' : [30,50,70],
}
```

- For Random Forest:

```
param_grid = {
    "n_estimators": [50,110,25],
    "min_samples_leaf": np.arange(1, 4),
    "max_features": [np.arange(0.3, 0.6, 0.1),'sqrt'],
    "max_samples": np.arange(0.4, 0.7, 0.1)
}
```

- For Decision Trees:

```
param_grid = {
    'max_depth': np.arange(2,6),
    'min_samples_leaf': [1, 4, 7],
    'max_leaf_nodes' : [10, 15],
    'min_impurity_decrease': [0.0001,0.001]
}
```

- For XGBoost (optional):

```
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]
}
```

Example tuning method for Decision tree with original data

In [191…
```
# defining model
Model = DecisionTreeClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {'max_depth': np.arange(2,6),
              'min_samples_leaf': [1, 4, 7],
              'max_leaf_nodes' : [10,15],
              'min_impurity_decrease': [0.0001,0.001] }

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_iter=10, n_jobs = -1, sco

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train,y_train)

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

```
Best parameters are {'min_samples_leaf': 7, 'min_impurity_decrease': 0.0001, 'max_leaf_nodes': 15, 'max_depth':
5} with CV score=0.9267489711934156:
```

Example tuning method for Decision tree with oversampled data

In [190…
```
# defining model
Model = DecisionTreeClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {'max_depth': np.arange(2,6),
              'min_samples_leaf': [1, 4, 7],
              'max_leaf_nodes' : [10,15],
              'min_impurity_decrease': [0.0001,0.001] }

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_iter=10, n_jobs = -1, sco

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_over,y_train_over)

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

```
Best parameters are {'min_samples_leaf': 7, 'min_impurity_decrease': 0.0001, 'max_leaf_nodes': 15, 'max_depth':
5} with CV score=0.9107692640567754:
```

Example tuning method for Decision tree with undersampled data

In [183…
```
# defining model
Model = DecisionTreeClassifier(random_state=1)
```

```
# Parameter grid to pass in RandomSearchCV
param_grid = {'max_depth': np.arange(2,6),
              'min_samples_leaf': [1, 4, 7],
              'max_leaf_nodes' : [10,15],
              'min_impurity_decrease': [0.0001,0.001] }

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_grid, n_iter=10, n_jobs = -1, sco

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_un,y_train_un)

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

```
Best parameters are {'min_samples_leaf': 7, 'min_impurity_decrease': 0.0001, 'max_leaf_nodes': 15, 'max_depth':
5} with CV score=0.896010230179028:
```

Model Chosen for Original Data

- **Decision Tree Classifier** - Performed the lowest testing performances compared to the other models with original data, will see if hyperparameter tuning can increase the testing performances and make the model less overfitting and more generalized.

Model Chosen for Oversampled Data

- **XGBoost Classifier** - This model was chosen to see if the testing performance metrics would improve and prevent the model further from overfitting.

Model Chosen for Undersampled Data

- **Bagging Classifier** - Contains a really low recall score, will test to see if hypertuning can improve the score and make the model more generalized.

Tuning the Decision Tree Classifier model with the Original Data

In [184…

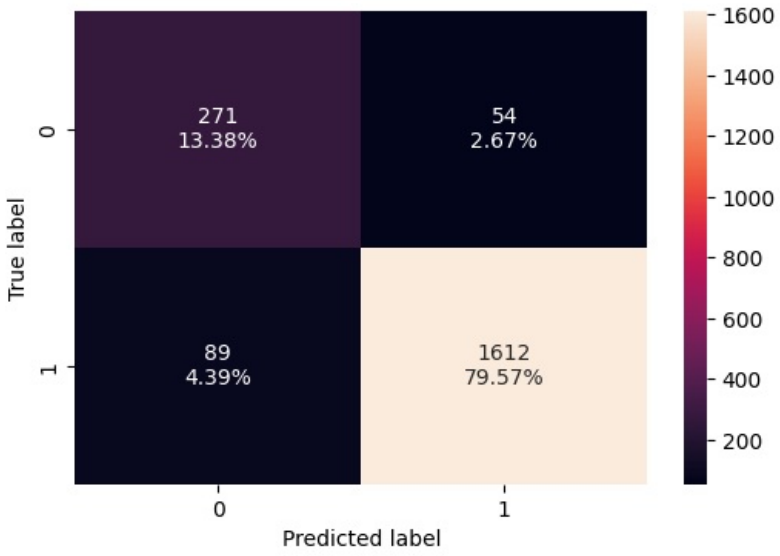
```
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import make_scorer, accuracy_score # Import accuracy_score or any other metric

# defining model
dtree_estimator = DecisionTreeClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    'max_depth': np.arange(2,6),
    'min_samples_leaf': [1, 4, 7],
    'max_leaf_nodes' : [10, 15],
    'min_impurity_decrease': [0.0001,0.001]
}

# Define a scoring metric
scorer = make_scorer(accuracy_score) # Use accuracy_score or any other relevant metric

#Calling RandomizedSearchCV
dtree_estimator = RandomizedSearchCV(estimator=dtree_estimator, param_distributions=param_grid, n_iter=10, n_jo

#Fitting parameters in RandomizedSearchCV
dtree_estimator.fit(X_train,y_train)

print("Best parameters are {} with CV score={}:" .format(dtree_estimator.best_params_,dtree_estimator.best_scor
```

```
Best parameters are {'min_samples_leaf': 7, 'min_impurity_decrease': 0.0001, 'max_leaf_nodes': 15, 'max_depth':
5} with CV score=0.9267489711934156:
```

- The model has picked the pest parameters from the paramamter grid.

In [185…

```
confusion_matrix_sklearn(dtree_estimator, X_test,y_test)
```



- The model performs slightly better in the True Positive predictions as well as lesser predictions for True Negatives, the rest of the two sections make the model worse from hypertuning.

In [186…
```
dtree_estimator_model_train_perf=model_performance_classification_sklearn(dtree_estimator, X_train,y_train)
print("Training performance \n",dtree_estimator_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.937449  0.958227    0.96695  0.962569
```

In [187…
```
dtree_estimator_model_test_perf=model_performance_classification_sklearn(dtree_estimator, X_test,y_test)
print("Testing performance \n",dtree_estimator_model_test_perf)
```

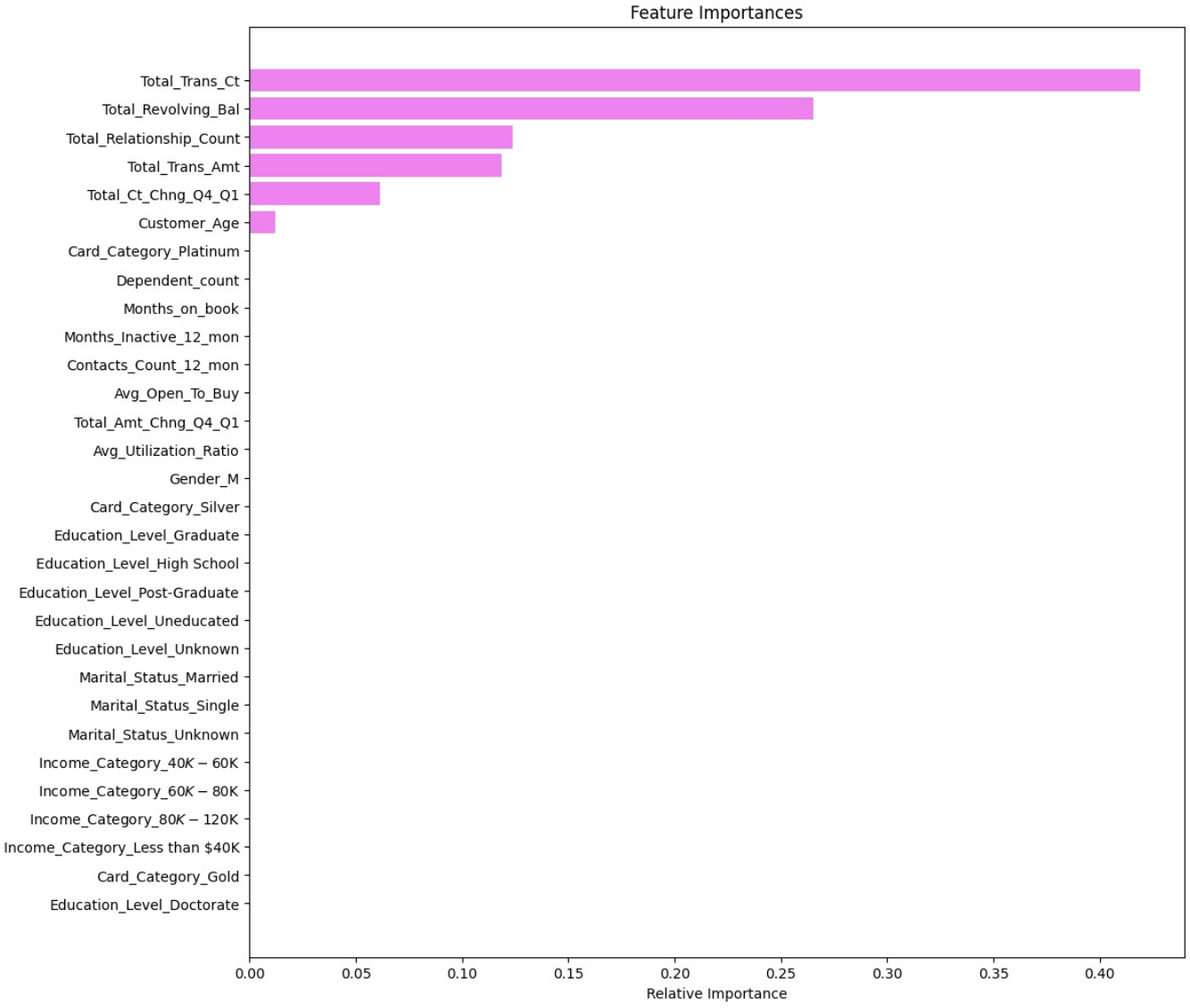
```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.929418  0.947678   0.967587  0.957529
```

- The model performs much better compared to the original not hypertuned one. Now there is much less overfitting and the model is essentially generalized up to par. Thus showing that there is model performance improvement from hyperparameter tuning.

In [188…
```
# Viewing the important features in this model

# Access the best estimator from the RandomizedSearchCV object
best_estimator = dtree_estimator.best_estimator_

# Now you can access feature_importances_
print(pd.DataFrame(best_estimator.feature_importances_,
                   columns = ["Imp"],
                   index = X_train.columns).sort_values(by = 'Imp', ascending = False))
```

```
                                Imp
Total_Trans_Ct                0.419011
Total_Revolving_Bal           0.265197
Total_Relationship_Count      0.123562
Total_Trans_Amt               0.118705
Total_Ct_Chng_Q4_Q1           0.061501
Customer_Age                  0.012024
Marital_Status_Married        0.000000
Marital_Status_Single         0.000000
Marital_Status_Unknown        0.000000
Income_Category_$40K - $60K   0.000000
Income_Category_$60K - $80K   0.000000
Education_Level_Uneducated    0.000000
Income_Category_$80K - $120K  0.000000
Income_Category_Less than $40K 0.000000
Card_Category_Gold            0.000000
Card_Category_Platinum        0.000000
Education_Level_Unknown       0.000000
Education_Level_Graduate      0.000000
Education_Level_Post-Graduate 0.000000
Education_Level_High School   0.000000
Dependent_count               0.000000
Education_Level_Doctorate     0.000000
Gender_M                      0.000000
Avg_Utilization_Ratio         0.000000
Total_Amt_Chng_Q4_Q1          0.000000
Avg_Open_To_Buy               0.000000
Contacts_Count_12_mon         0.000000
Months_Inactive_12_mon        0.000000
Months_on_book                0.000000
Card_Category_Silver          0.000000
```

In [189…
```
# Visualizing the Important Features

feature_names = X_train.columns
# Access feature importances from the best estimator
importances = dtree_estimator.best_estimator_.feature_importances_
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()
```



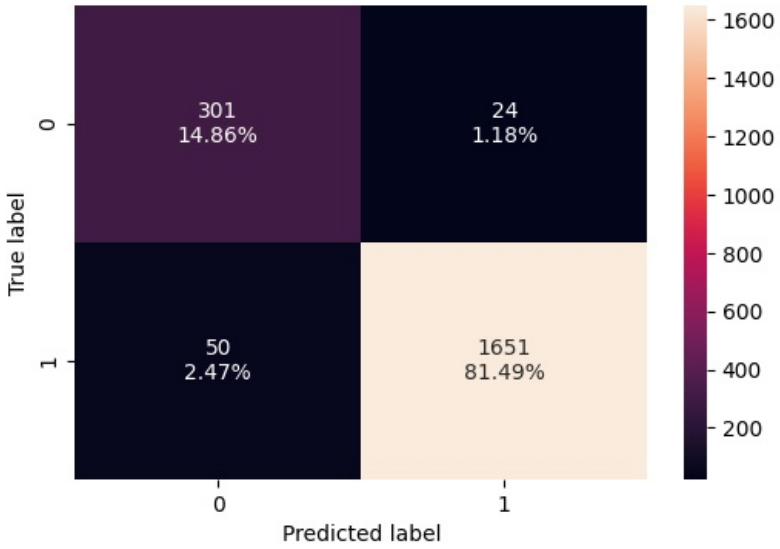
- As seen from the above bar graph, the most important feature for the tuned model is the Total_Trans_Ct by a good margin compared to the others. Following that, the second most relatively important feature is Total_Revolving_Bal and the third is Total_Relationship_Count.

Tuning XGBoost Classifier model with the Oversampled Data

In [177…
```
# defining model
xgb_estimator = 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]
}

#Calling RandomizedSearchCV
xgb_estimator = RandomizedSearchCV(estimator=xgb_estimator, param_distributions=param_grid, n_iter=10, n_jobs =

#Fitting parameters in RandomizedSearchCV
xgb_estimator.fit(X_train_over,y_train_over)

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

```
Best parameters are {'subsample': 0.7, 'scale_pos_weight': 1, 'n_estimators': 100, 'learning_rate': 0.05, 'gamm
a': 1} with CV score=0.9672514881381685:
```

In [178…
```
confusion_matrix_sklearn(xgb_estimator, X_test,y_test)
```



- Compared to the not hyper-tuned model, this model performs better on the True Positives portion, however for the rest of the sections it doesn't exceed at all, rather a few percentages down.

In [179…
```
xgb_estimator_model_train_perf=model_performance_classification_sklearn(xgb_estimator, X_train,y_train)
print("Training performance \n",xgb_estimator_model_train_perf)
```

```
Training performance
    Accuracy    Recall  Precision        F1
0  0.985021  0.987056   0.995057  0.991041
```

In [180…
```
xgb_estimator_model_test_perf=model_performance_classification_sklearn(xgb_estimator, X_test,y_test)
print("Testing performance \n",xgb_estimator_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.963475  0.970606   0.985672  0.978081
```

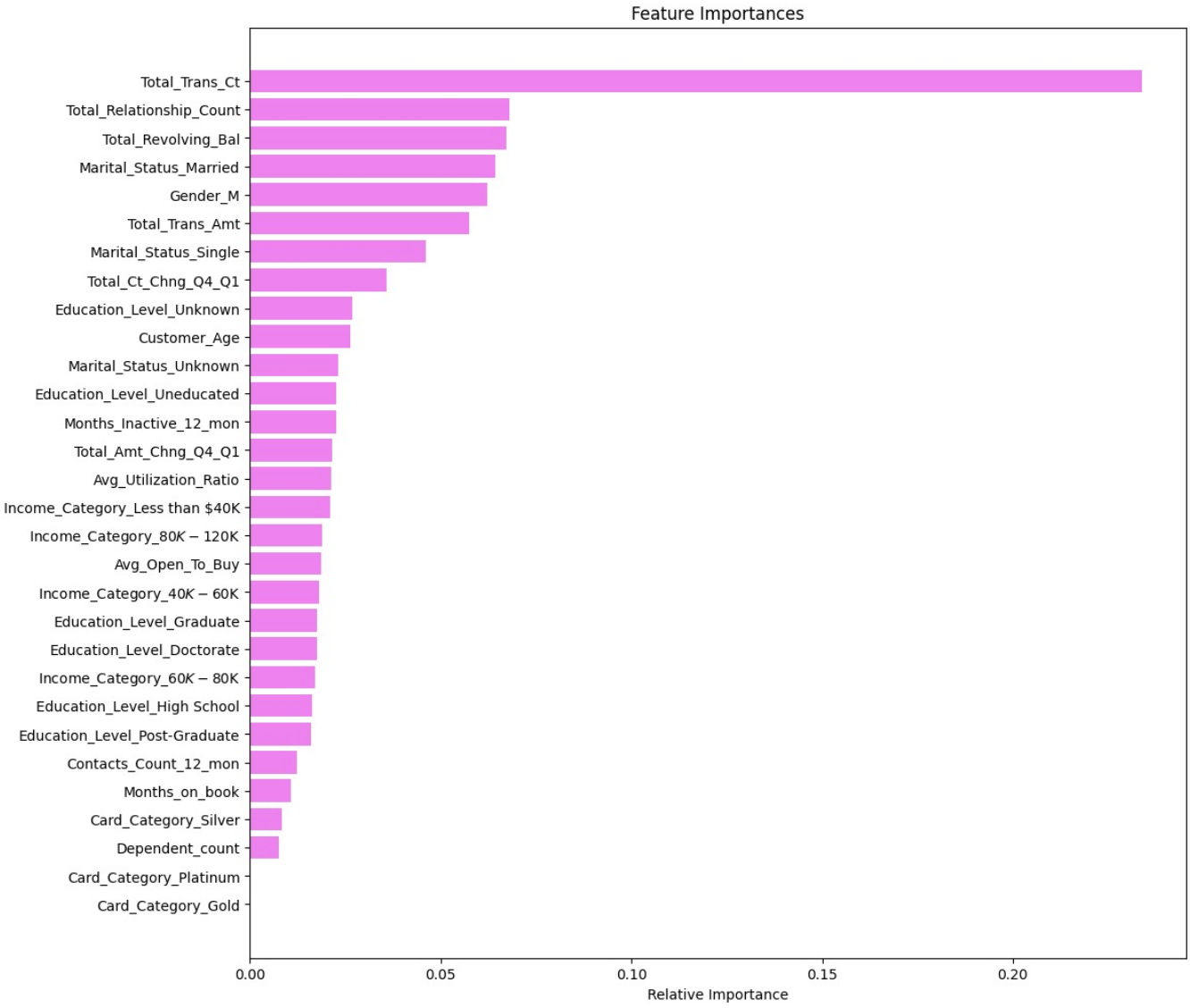
- The training and testing performances did in fact slightly better than the original model due to the results having a lesser gap between each other to reduce overfitting. Furthermore, the recall scores are almost identical with each other in this hyperparameter tuning model which is a good to have since recall is one metric of interest.

In [181…
```
# Viewing the important features in this model

# Access the best estimator from the RandomizedSearchCV object
best_estimator = xgb_estimator.best_estimator_

# Now you can access feature_importances_
print(pd.DataFrame(best_estimator.feature_importances_,
                   columns = ["Imp"],
                   index = X_train.columns).sort_values(by = 'Imp', ascending = False))
```

```
                                 Imp
Total_Trans_Ct                0.233829
Total_Relationship_Count      0.068101
Total_Revolving_Bal           0.067323
Marital_Status_Married        0.064423
Gender_M                      0.062228
Total_Trans_Amt               0.057586
Marital_Status_Single         0.046159
Total_Ct_Chng_Q4_Q1           0.035686
Education_Level_Unknown       0.026789
Customer_Age                  0.026259
Marital_Status_Unknown        0.023082
Education_Level_Uneducated    0.022723
Months_Inactive_12_mon        0.022500
Total_Amt_Chng_Q4_Q1          0.021623
Avg_Utilization_Ratio         0.021177
Income_Category_Less than $40K  0.020967
Income_Category_$80K - $120K  0.019024
Avg_Open_To_Buy               0.018691
Income_Category_$40K - $60K   0.018208
Education_Level_Graduate      0.017726
Education_Level_Doctorate     0.017715
Income_Category_$60K - $80K   0.017199
Education_Level_High School   0.016249
Education_Level_Post-Graduate  0.015904
Contacts_Count_12_mon         0.012323
Months_on_book                0.010606
Card_Category_Silver          0.008266
Dependent_count               0.007634
Card_Category_Gold            0.000000
Card_Category_Platinum        0.000000
```

In [182…

```python
# Visualizing the Important Features

feature_names = X_train.columns
# Access feature importances from the best estimator
importances = xgb_estimator.best_estimator_.feature_importances_
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()
```



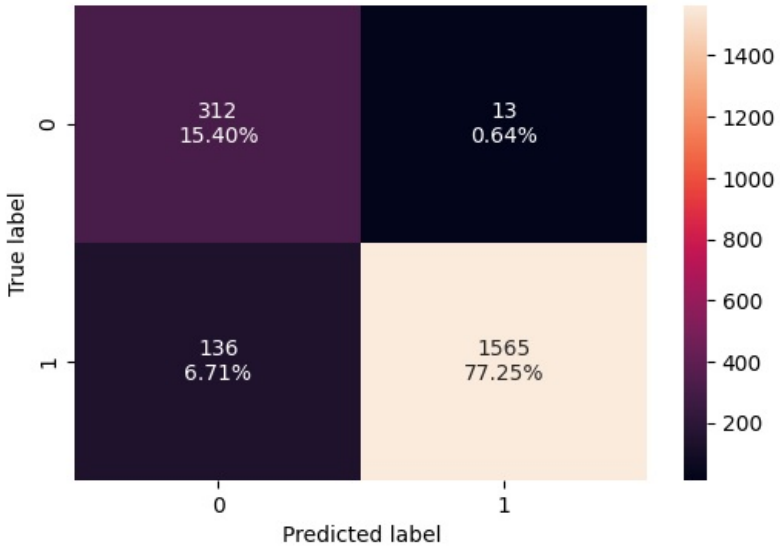
- Compared to the previous hyper-tuned model (AdaBoost with original data), there is a change in features amount/order for the best 3 important features. With Total_Trans_Ct has the highest relative importance compared to the rest of the features, the second highest importance being Total_Relationship_Count and third is the Total_Revolving_Bal.

Tuning Bagging Classifier with Undersampled Data

In [162…
```
# defining model
bagging_estimator = BaggingClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    'max_samples': [0.8,0.9,1],
    'max_features': [0.7,0.8,0.9],
    'n_estimators' : [30,50,70],
}

#Calling RandomizedSearchCV
bagging_estimator = RandomizedSearchCV(estimator=bagging_estimator, param_distributions=param_grid, n_iter=10,

#Fitting parameters in RandomizedSearchCV
bagging_estimator.fit(X_train_un,y_train_un)

print("Best parameters are {} with CV score={}:" .format(bagging_estimator.best_params_,bagging_estimator.best_
```

```
Best parameters are {'n_estimators': 70, 'max_samples': 0.9, 'max_features': 0.8} with CV score=0.9385192471637
485:
```

In [163…
```
confusion_matrix_sklearn(bagging_estimator, X_test,y_test)
```



- In comparison to the original not hypertuned model, it performs better in having more True Positive predictions as well as having less amount of True Negatives.

In [164…
```
bagging_estimator_model_train_perf=model_performance_classification_sklearn(bagging_estimator, X_train_un,y_tra
print("Training performance \n",bagging_estimator_model_train_perf)
```

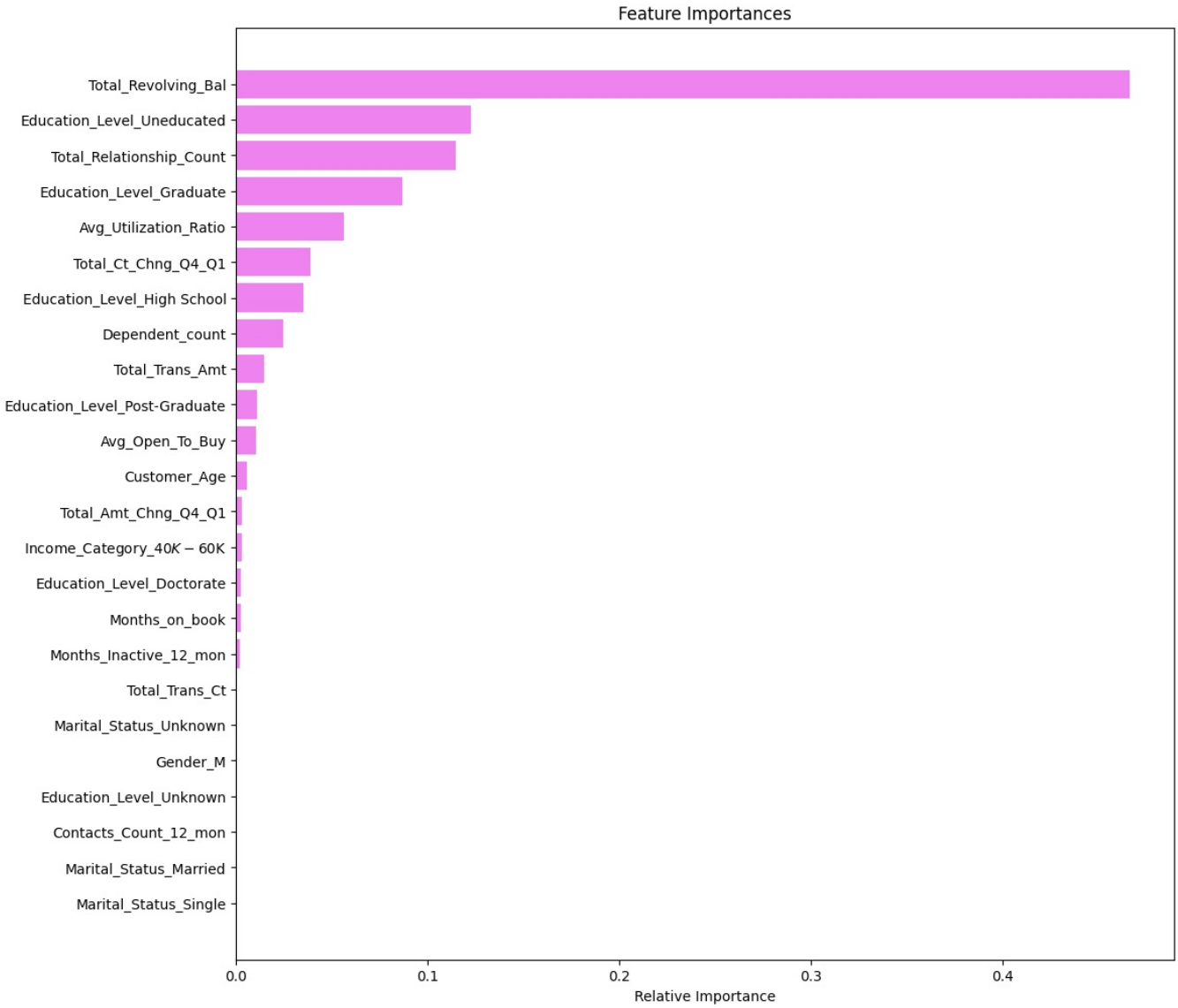
```
Training performance
    Accuracy    Recall  Precision        F1
0  0.998975  0.997951        1.0  0.998974
```

In [165…
```
bagging_estimator_model_test_perf=model_performance_classification_sklearn(bagging_estimator, X_test,y_test)
print("Testing performance \n",bagging_estimator_model_test_perf)
```

```
Testing performance
    Accuracy    Recall  Precision        F1
0  0.926456  0.920047   0.991762  0.954559
```

- The hypertuned model does perform better overall in both the training and testing performance metrics compared to the original model. This showcases that the hyperparameter tuning did result in a positive effect.

In [166…
```
# Visualizing the Important Features

feature_names = X_train.columns

# Assuming your base estimator within BaggingClassifier is a tree-based model
# Access feature importances from the base estimator of the best estimator
importances = bagging_estimator.best_estimator_.estimators_[0].feature_importances_
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()
```



- Compared to the previous hyper-tuned model (XGBoost Classifier with Oversampled Data), there is a complete change in terms of the top 3 most relatively important features. The first is Total_Revoling_Bal with a drastic value compared to the rest of the features, the second is the Education_Level_Uneducated, and the third is Total_Relationship_Count.

Model Comparison and Final Model Selection

Training set final performance

In [167…

```
# Training performance comparison

models_train_comp_df = pd.concat(
    [dtree_model_train_perf.T, dtree_estimator_model_train_perf.T, xgb_model_train_perf.T, xgb_estimator_model_
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree Classifier (Original Data)",
    "Tuned Decision Tree Classifier (Original Data)",
    "XGBoost Classifier (Oversampled Data)",
    "Tuned XGBoost Classifier (Oversampled Data)",
    "Bagging Classifier (Undersampled Data)",
    "Tuned Bagging Classifier (Undersampled Data)"]
print("Training Performance Comparison:")
models_train_comp_df
```

```
Training Performance Comparison:
```

Out[167]:

	Decision Tree Classifier (Original Data)	Tuned Decision Tree Classifier (Original Data)	XGBoost Classifier (Oversampled Data)	Tuned XGBoost Classifier (Oversampled Data)	Bagging Classifier (Undersampled Data)	Tuned Bagging Classifier (Undersampled Data)
Accuracy	1.0	0.937449	1.0	0.985021	0.996049	0.998975
Recall	1.0	0.958227	1.0	0.987056	0.996666	0.997951
Precision	1.0	0.966950	1.0	0.995057	0.998624	1.000000
F1	1.0	0.962569	1.0	0.991041	0.997644	0.998974

Test set final performance

In [168…

```
# Testing performance comparison

models_test_comp_df = pd.concat(
    [dtree_model_test_perf.T, dtree_estimator_model_test_perf.T, xgb_model_test_perf.T, xgb_estimator_model_tes
    axis=1,
)
models_test_comp_df.columns = [
    "Decision Tree Classifier (Original Data)",
    "Tuned Decision Tree Classifier (Original Data)",
    "XGBoost Classifier (Oversampled Data)",
    "Tuned XGBoost Classifier (Oversampled Data)",
    "Bagging Classifier (Undersampled Data)",
    "Tuned Bagging Classifier (Undersampled Data)"]
print("Testing Performance Comparison:")
models_test_comp_df
```

```
Testing Performance Comparison:
```

Out[168]:

	Decision Tree Classifier (Original Data)	Tuned Decision Tree Classifier (Original Data)	XGBoost Classifier (Oversampled Data)	Tuned XGBoost Classifier (Oversampled Data)	Bagging Classifier (Undersampled Data)	Tuned Bagging Classifier (Undersampled Data)
Accuracy	0.937808	0.929418	0.967423	0.963475	0.955577	0.926456
Recall	0.959436	0.947678	0.984127	0.970606	0.970018	0.920047
Precision	0.966252	0.967587	0.977233	0.985672	0.976909	0.991762
F1	0.962832	0.957529	0.980668	0.978081	0.973451	0.954559

- The XGBoost Classifier testing performance with the tuned and not tuned recall scores did not seem to contain much improvement, although they have stayed in a generalized position.
- The Bagging Classifier testing performances turned out to be the least improved model out of the three in terms of recall scores which is the metric of interest.
- Both the Bagging & XGBoost Classifier training performances stayed the same after the model was hypertuned with a randomized search.
- The Decision Tree Classifier training & testing performance resulted in the best overall metrics compared to the other tuned models, and the recall score comes up to par with being as generalized as the XGBoost tuned model.

Feature Importance of Decision Tree

In [169…

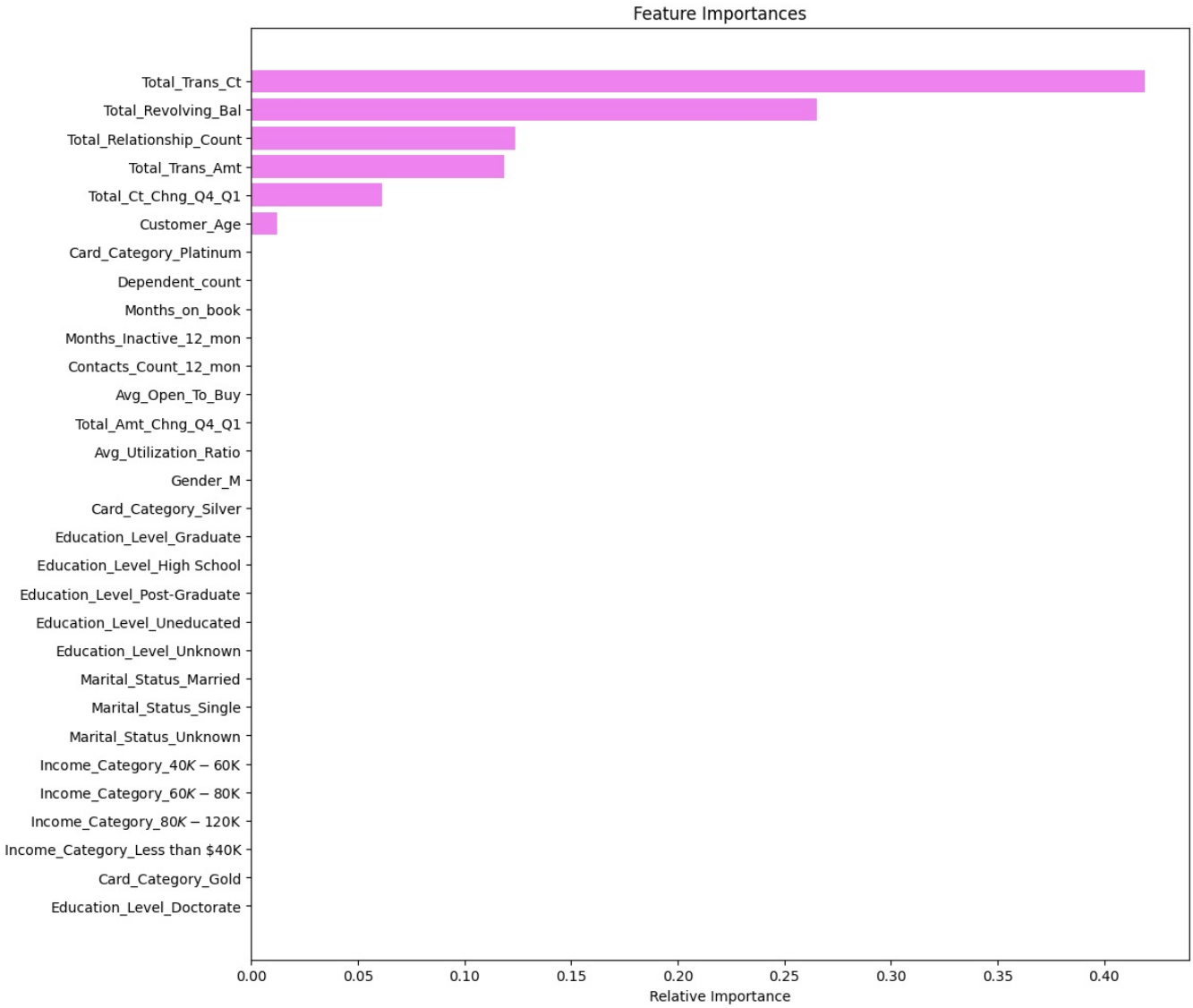
```
# Access the best estimator from the RandomizedSearchCV object
best_dtree = dtree_estimator.best_estimator_

# Now you can access feature_importances_
print(pd.DataFrame(best_dtree.feature_importances_,
                   columns=["Imp"],
                   index=X_train.columns).sort_values(by='Imp', ascending=False))
```

	Imp
Total_Trans_Ct	0.419011
Total_Revolving_Bal	0.265197
Total_Relationship_Count	0.123562
Total_Trans_Amt	0.118705
Total_Ct_Chng_Q4_Q1	0.061501
Customer_Age	0.012024
Marital_Status_Married	0.000000
Marital_Status_Single	0.000000
Marital_Status_Unknown	0.000000
Income_Category_$40K - $60K	0.000000
Income_Category_$60K - $80K	0.000000
Education_Level_Uneducated	0.000000
Income_Category_$80K - $120K	0.000000
Income_Category_Less than $40K	0.000000
Card_Category_Gold	0.000000
Card_Category_Platinum	0.000000
Education_Level_Unknown	0.000000
Education_Level_Graduate	0.000000
Education_Level_Post-Graduate	0.000000
Education_Level_High School	0.000000
Dependent_count	0.000000
Education_Level_Doctorate	0.000000
Gender_M	0.000000
Avg_Utilization_Ratio	0.000000
Total_Amt_Chng_Q4_Q1	0.000000
Avg_Open_To_Buy	0.000000
Contacts_Count_12_mon	0.000000
Months_Inactive_12_mon	0.000000
Months_on_book	0.000000
Card_Category_Silver	0.000000

The Transaction Count feature is the most important variable in being able to predict if a customer will renounce their credit cards or not. Furthermore, the secondary important feature following that is the Total Revolving Balance.

In [170…

```
# Access the best estimator from the RandomizedSearchCV object
best_dtree = dtree_estimator.best_estimator_

# Now you can extract the feature importances
importances = best_dtree.feature_importances_
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()
```



Business Insights and Conclusions

Business Insights

- Factors that drive attrition are the total transaction account, total revolving balance, and total relationship count.
- Total Transaction Count: Customers who are not active or with a low amount of transactions over the past year tend to Attrite more due to inactivity. Therefore, Thera Bank should make sure that all the customers are pleased with their service and not having a disappointing experience.
- Total Revolving Balance: Those customers who have a large amount of revolving balance might indicate that the customer is having difficulty in managing their debt which could possibly lead to being attrited. The Thera Bank needs to provide the customers who are in this position financial advising sessions if not already, as well as providing insights/advice to the customers in need.
- Total Relationship Count: Customers who have a minor amount of products in a relationship with Thera Bank will most likely end up churning over those who have a good amount of products. The bank can use this information to focus more on retention strategies, for instance offering them personalized deals in order for them to be engaged and encouraged with the bank.
- Total Transaction Amount: Similar to the total transaction account, those customers with a low transaction over the past year may either indicate that the customer doesn't have the income to be spending, or the customer doesn't find the usage bank's credit card and prefers to use another. On the contrary, those customers with high transaction amounts could be loyal and wealthy customers with high spending habits. In order for the bank to help those with low transaction amounts on a yearly basis, Thera Bank should offer personalized rewards or cashback on certain categories that the customers frequently spend upon. This can be groceries, dining, or traveling.
- Ratio of Total Transaction Account (4th Qtr to 1st Qtr): Those customers with a small ratio of transactions may lead to attrition, some of them could be in early indicators as well, thus prompting the bank to intervene with retention strategies.
- Customer Age: Those customers who are in certain age groups such as young adults, middle-aged, and older can have a varying likelihood of being attracted or not. The bank needs to offer certain age groups their related rewards. For example, for young adults, the bank could offer student credit cards with lower credit limits and benefits students such as cashback textbooks or dining. For older adults, offer retirement planning products like credit cards with benefits related to health, and leisure, and provide retirement savings accounts.

Recommendations

- Enhance customer engagement and retention, features such as total transaction amount & count can depict high-value customers and tailor loyalty programs and personalized offers to encourage them to continue the relationship with Thera Bank.
- Implement targeted financial products and services to specific demographics, using features such as the age of the customers, their income, and their educational background. The bank could design and offer financial products that can cater to those specific groups of customers.
- Focus on churn prevention and risk management, features like the number of months inactive in a year and the ratio of transaction count from the 1st quarter to the 4th can be used to identify customers who may be at risk of attracting. Thera Bank should develop strategies such as personalized outreach or exclusive offers to those customers.
