Case Study: Thera Bank | Credit Card Users Churn Prediction

Model Tuning Techniques

Problem Statement:

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

You need to identify the best possible model that will give the required performance

Objective:

1. Explore and visualize the dataset.
2. Build a classification model to predict if the customer is going to churn or not
3. Optimize the model using appropriate techniques
4. Generate a set of insights and recommendations that will help the bank

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 a 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
- 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 between the customer and bank in the last 12 months
- Credit_Limit: Credit Limit on the Credit Card
- Total_Revolving_Bal: The balance that carries over from one month to the next is the revolving balance
- Avg_Open_To_Buy: Open to Buy refers to the amount left on the credit card to use (Average of last 12 months)
- Total_Trans_Amt: Total Transaction Amount (Last 12 months)
- Total_Trans_Ct: Total Transaction Count (Last 12 months)
- Total_Ct_Chng_Q4_Q1: Ratio of the total transaction count in 4th quarter and the total transaction count in 1st quarter
- Total_Amt_Chng_Q4_Q1: Ratio of the total transaction amount in 4th quarter and the total transaction amount in 1st quarter
- Avg_Utilization_Ratio: Represents how much of the available credit the customer spent

---

Importing necessary libraries

In [1]:
```
# To help with reading and manipulating data
import pandas as pd
import numpy as np

# To help with data visualization
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

# To be used for missing value imputation
from sklearn.impute import SimpleImputer

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

# To get different metric scores, and split data
from sklearn import metrics
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    plot_confusion_matrix,
```

```
)

# To impute missing values
from sklearn.impute import KNNImputer

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

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

# To be used for tuning the model
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

# To be used for creating pipelines and personalizing them
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

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

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

# To supress warnings
import warnings

warnings.filterwarnings("ignore")

# This will help in making the Python code more structured automatically (goo
%load_ext nb_black
```

Load and overview the dataset

In [2]:
```
# Load the data into pandas dataframe
Bank = pd.read_csv("BankChurners.csv")
```

In [3]:
```
# Coping the data to another variable to avoid any changes to the original da
df = Bank.copy()
```

In [4]:
```
df.head()
```

Out[4]:

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

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Ma
4	709106358	Existing Customer	40	M	3	Uneducated	

In [5]:
```
# Understanding the shape of the data
print(
    f"There are \033[1;4m{df.shape[0]}\033[m rows and \033[1;4m{df.shape[1]}\
)
```

There are **10127** rows and **21** columns.

In [6]:
```
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
```

- Considering that Attrition_Flag is our target columns, we gonna convert the string to number 0 (Existing Customer) and 1 (Attrited Customer) and Dtype as int64.
- We can see that 6 columns are Dtype object, we will convert it to categorical and optimize the memory usage.
- We can see that 2 columns have less than 10127 non-null values i.e. columns have missing values.

In [7]:
```
# Changing target columns to 0 and 1
df.Attrition_Flag = df.Attrition_Flag.apply(
    lambda x: 0 if x == "Existing Customer" else "1"
).astype("int64")
```

In [8]:
```
# let's check for duplicate values in the data
df.duplicated().sum()
```

Out[8]:
```
0
```

In [9]:
```
# Checking the percentage of missing values in each column

pd.DataFrame(
    data={
        "% of Missing Values": round(
            df.isna().sum() / df.isna().count() * 100, 2
        ).sort_values(ascending=False)
    }
)
```

Out[9]:

	% of Missing Values
Education_Level	15.000
Marital_Status	7.400
Avg_Utilization_Ratio	0.000
Months_on_book	0.000
Attrition_Flag	0.000
Customer_Age	0.000
Gender	0.000
Dependent_count	0.000
Income_Category	0.000
Card_Category	0.000
Total_Relationship_Count	0.000
Total_Ct_Chng_Q4_Q1	0.000
Months_Inactive_12_mon	0.000
Contacts_Count_12_mon	0.000
Credit_Limit	0.000
Total_Revolving_Bal	0.000
Avg_Open_To_Buy	0.000
Total_Amt_Chng_Q4_Q1	0.000
Total_Trans_Amt	0.000
Total_Trans_Ct	0.000
CLIENTNUM	0.000

- The `Education_Level` column has 15% missing values out of the total observations.
- `Marital_Status` column has 7.4% missing values out of the total observations.
- We will impute these values after we split the data into train, validation and test sets.

In [10]:
```
# Checking the number of unique values in each columns
df.nunique()
```

Out[10]:
```
CLIENTNUM                 10127
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
```

- We can drop the column: `CLIENTNUM` as it is unique for each customer and will not add value to the model.
- Out of the 20 columns, half we'll treat as categorical (Income_Category, Marital_Status ...) and the other half numerical (Credit_Limit, Total_Trans_Amt...)

In [11]:
```
# Dropping CLIENTNUM column
df.drop(columns="CLIENTNUM", inplace=True)
```

Summary of the dataset

In [12]:
```
df.describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	7
Attrition_Flag	10127.000	0.161	0.367	0.000	0.000	0.000	0.
Customer_Age	10127.000	46.326	8.017	26.000	41.000	46.000	52.
Dependent_count	10127.000	2.346	1.299	0.000	1.000	2.000	3.
Months_on_book	10127.000	35.928	7.986	13.000	31.000	36.000	40.
Total_Relationship_Count	10127.000	3.813	1.554	1.000	3.000	4.000	5.
Months_Inactive_12_mon	10127.000	2.341	1.011	0.000	2.000	2.000	3.
Contacts_Count_12_mon	10127.000	2.455	1.106	0.000	2.000	2.000	3.
Credit_Limit	10127.000	8631.954	9088.777	1438.300	2555.000	4549.000	11067.
Total_Revolving_Bal	10127.000	1162.814	814.987	0.000	359.000	1276.000	1784.
Avg_Open_To_Buy	10127.000	7469.140	9090.685	3.000	1324.500	3474.000	9859.
Total_Amt_Chng_Q4_Q1	10127.000	0.760	0.219	0.000	0.631	0.736	0.
Total_Trans_Amt	10127.000	4404.086	3397.129	510.000	2155.500	3899.000	4741.
Total_Trans_Ct	10127.000	64.859	23.473	10.000	45.000	67.000	81.
Total_Ct_Chng_Q4_Q1	10127.000	0.712	0.238	0.000	0.582	0.702	0

	count	mean	std	min	25%	50%	7
Avg_Utilization_Ratio	10127.000	0.275	0.276	0.000	0.023	0.176	0.

- We can see that the target variable - Attrition_Flag is imbalanced as most of the values are 0.
- Customer_Age has a very close mean and median, approx 46.32 and 46 respectively.
- Months_on_Book seems to have some outliers at both ends. We need to explore this further.
- Credit_Limit and Avg_Open_To_Buy seems to have outliers at the higher end. We gonna explore this on UniVariable analysis.
- Total_Revolving_Bal goes from 0 to 2517 with median greather than mean.

In [13]:
```
df.describe(include=object).T
```

Out[13]:

	count	unique	top	freq
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

- Most customers have Blue card, are Married and income less than $40K.

Let's check the count of each unique category in each of the categorical variables.

In [14]:
```
# Making a list of all categorical variables

cat_col = df.select_dtypes(include="object").columns.tolist()

cat_num = [
    "Dependent_count",
    "Total_Relationship_Count",
    "Months_Inactive_12_mon",
    "Contacts_Count_12_mon",
]
for col in cat_num:
    cat_col.append(col)
```

In [15]:
```
cat_col
```

Out[15]:
```
['Gender',
 'Education_Level',
 'Marital_Status',
 'Income_Category',
 'Card_Category',
 'Dependent_count',
 'Total_Relationship_Count',
 'Months_Inactive_12_mon',
 'Contacts_Count_12_mon']
```

In [16]:
```
# Printing number of count of each unique values in each categorical column

for i in cat_col:
    print(f"Unique values in \033[1m{i}\033[m are:")
    print(df[i].value_counts())
    print("\n", "*" * 50, "\n")
```

```
Unique values in Gender are:
F    5358
M    4769
Name: Gender, dtype: int64

 **************************************************

Unique values in Education_Level are:
Graduate         3128
High School      2013
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64

 **************************************************

Unique values in Marital_Status are:
Married     4687
Single      3943
Divorced     748
Name: Marital_Status, dtype: int64

 **************************************************

Unique values in Income_Category are:
Less than $40K    3561
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
abc               1112
$120K +            727
Name: Income_Category, dtype: int64

 **************************************************

Unique values in Card_Category are:
Blue        9436
Silver       555
Gold         116
Platinum      20
Name: Card_Category, dtype: int64

 **************************************************

Unique values in Dependent_count are:
3    2732
2    2655
1    1838
4    1574
0     904
5     424
Name: Dependent_count, dtype: int64

 **************************************************

Unique values in Total_Relationship_Count are:
3    2305
4    1912
5    1891
```

```
6    1866
2    1243
1     910
Name: Total_Relationship_Count, dtype: int64

 **************************************************

Unique values in Months_Inactive_12_mon are:
3    3846
2    3282
1    2233
4     435
5     178
6     124
0      29
Name: Months_Inactive_12_mon, dtype: int64

 **************************************************

Unique values in Contacts_Count_12_mon are:
3    3380
2    3227
1    1499
4    1392
0     399
5     176
6      54
Name: Contacts_Count_12_mon, dtype: int64

 **************************************************
```

- The Post-Graduate and Doctorate category in the Education_Level column represents less than 0.10% of entries out of total observations.
- We can see that Income_Category has abc as unique values. This must be a data input error, we should replace it with NaN and treat it as missing value.
- Months_Inactive_12_mon equal to 0 has a count equal to 29 only.
- The majority of the customers hava a Blue card.

In [17]:
```
# Replacing 'abc' with Nan
df.Income_Category = df.Income_Category.replace("abc", np.nan)
```

In [18]:
```
# Checking the replaciment on Income_Category as NaN
df.isnull().sum()
```

Out[18]:
```
Attrition_Flag                0
Customer_Age                  0
Gender                        0
Dependent_count               0
Education_Level            1519
Marital_Status              749
Income_Category            1112
Card_Category                 0
Months_on_book                0
Total_Relationship_Count      0
Months_Inactive_12_mon        0
Contacts_Count_12_mon         0
Credit_Limit                  0
Total_Revolving_Bal           0
Avg_Open_To_Buy               0
Total_Amt_Chng_Q4_Q1          0
Total_Trans_Amt               0
Total_Trans_Ct                0
```

```
Total_Ct_Chng_Q4_Q1           0
Avg_Utilization_Ratio         0
dtype: int64
```

In [19]:
```
# Converting the data type of each categorical variable to 'category'
for column in cat_col:
    df[column] = df[column].astype("category")
```

In [20]:
```
# Checking the DType
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Attrition_Flag            10127 non-null  int64
 1   Customer_Age              10127 non-null  int64
 2   Gender                    10127 non-null  category
 3   Dependent_count           10127 non-null  category
 4   Education_Level           8608 non-null   category
 5   Marital_Status            9378 non-null   category
 6   Income_Category           9015 non-null   category
 7   Card_Category             10127 non-null  category
 8   Months_on_book            10127 non-null  int64
 9   Total_Relationship_Count  10127 non-null  category
 10  Months_Inactive_12_mon    10127 non-null  category
 11  Contacts_Count_12_mon     10127 non-null  category
 12  Credit_Limit              10127 non-null  float64
 13  Total_Revolving_Bal       10127 non-null  int64
 14  Avg_Open_To_Buy           10127 non-null  float64
 15  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 16  Total_Trans_Amt           10127 non-null  int64
 17  Total_Trans_Ct            10127 non-null  int64
 18  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 19  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: category(9), float64(5), int64(6)
memory usage: 961.3 KB
```

EDA

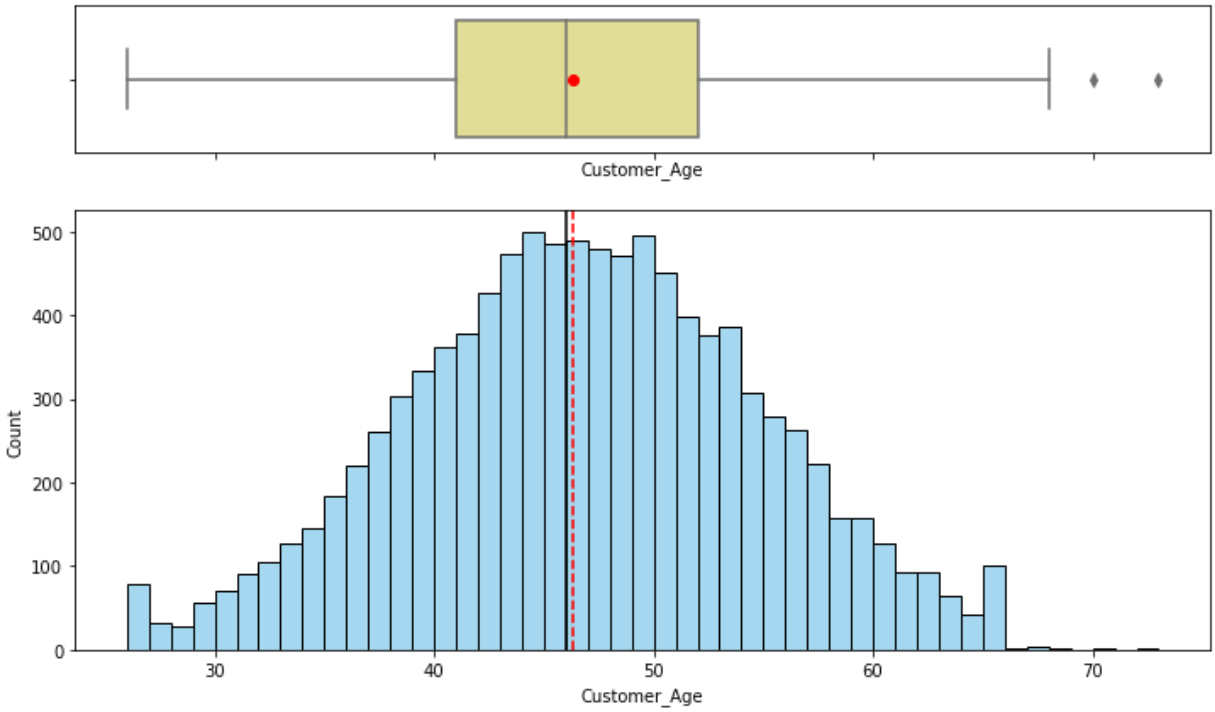
Univariate Analysis

In [21]:
```
# 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 show the 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,
        color="Khaki",
        showmeans=True,
        meanprops={
            "marker": "o",
            "markerfacecolor": "red",
            "markeredgecolor": "red",
            "markersize": "6",
        },
    )  # boxplot will be created and a star will indicate the mean value of t
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, color="SkyBlue
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, color="SkyBlue"
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="red", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

Observarions on Customer_Age

In [22]:
```
histogram_boxplot(df, "Customer_Age")
```



- Age distribution looks approximately normally distributed.
- Mean and Midian is close to each other.
- The boxplot for the Customer Age column shows that there are few outliers for this variable.
- Age can be an important variable to describle customer behaviers with Credit Card. We will further explore this in bivariate analysis.

In [23]:
```
df[df["Customer_Age"] > 69]
```

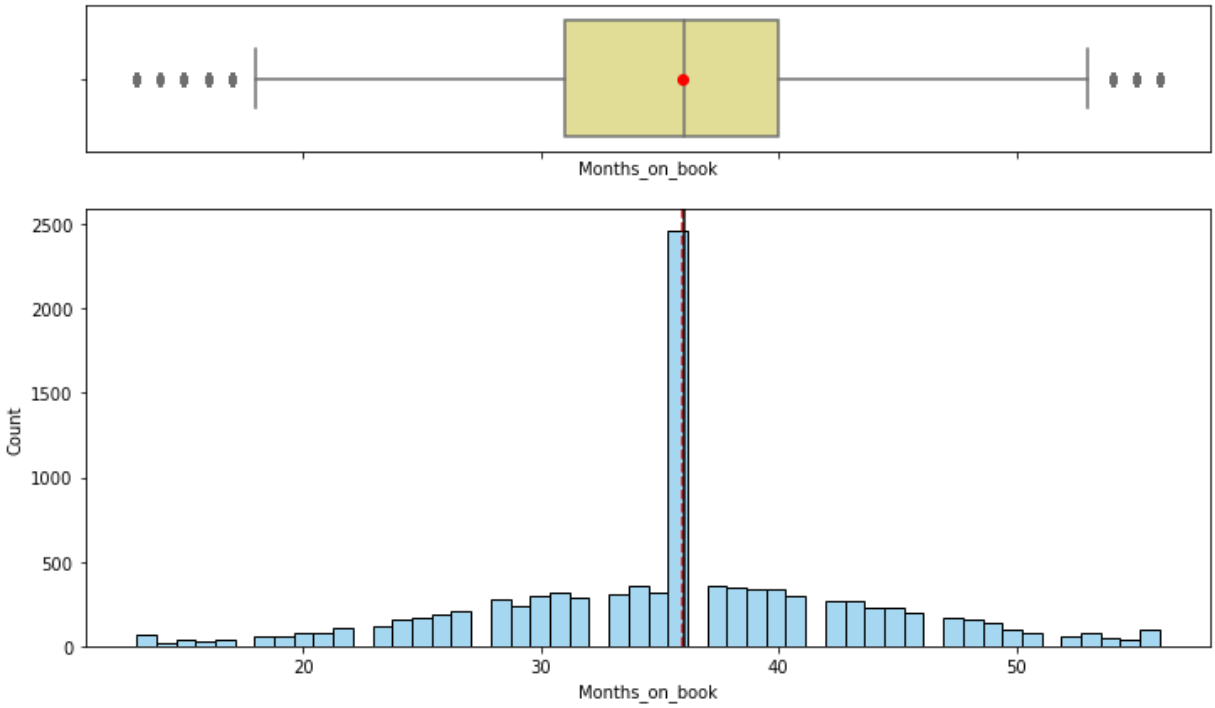
Out[23]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status
251	0	73	M	0	High School	Married
254	0	70	M	0	High School	Married

- We can see that there are just two observations greater than 69 years, it is likely similar values will continue to appear (further data), so we not gonna consider it as outliers.

Observarions on Months_on_book

In [24]:
```
histogram_boxplot(df, "Months_on_book")
```



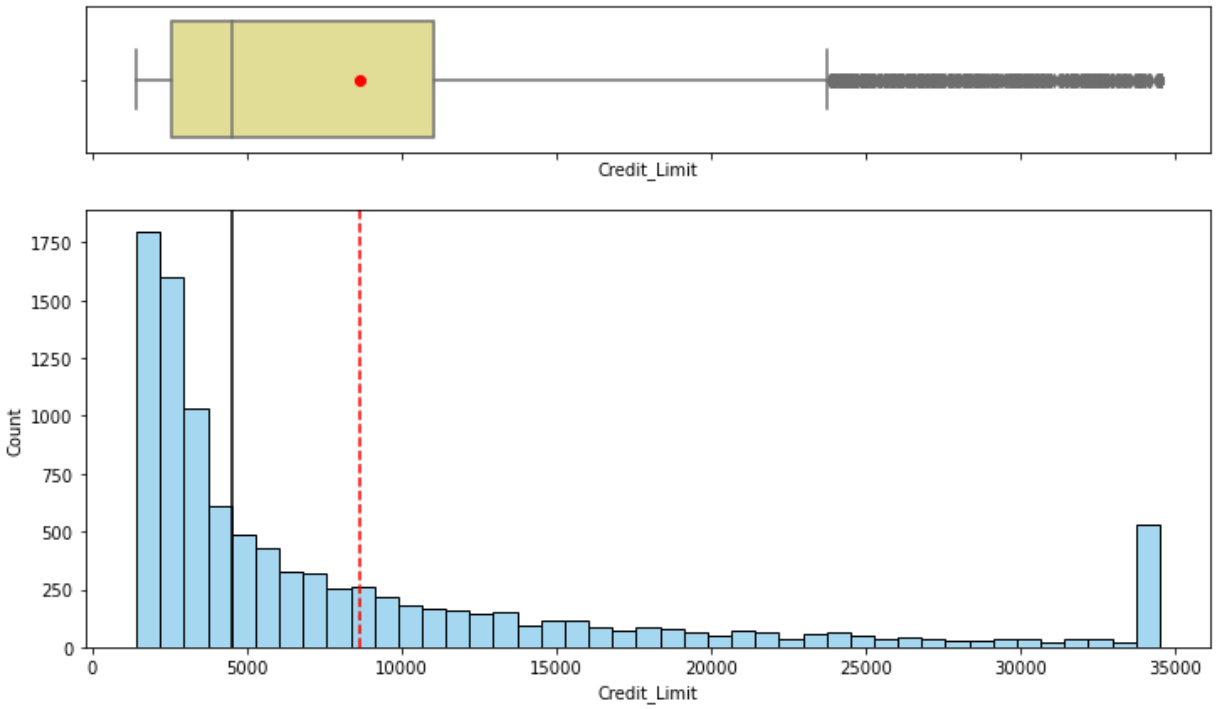
In [25]:
```
df[(df["Months_on_book"] < 18) | (df["Months_on_book"] > 53)]["Months_on_book
```

Out[25]: 386

- Months_on_book distribution looks approximately normally distributed with Mean and Midian very close to each other aprox 36 months.
- We can see outliers in both ends, considering that is the Period of relationship with the bank, it is expected to se a huge range on this variable, some new customers and customers that have been with the bank for long period. We gonna do more analysis with Multivariables.

Observarions on Credit_Limit

In [26]:
```
histogram_boxplot(df, "Credit_Limit")
```



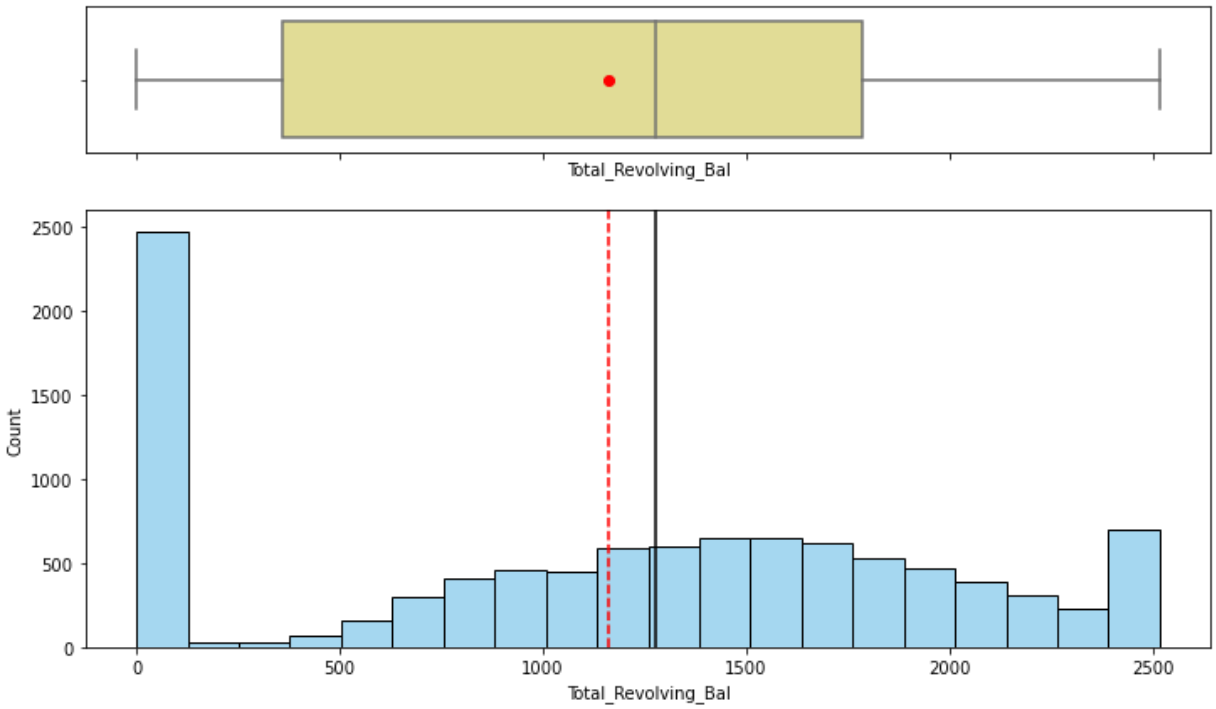
In [27]:
```
df[df["Credit_Limit"] > 25000]["Credit_Limit"].count()
```

Out[27]: 892

- The distribution for the Credit_Limit is right-skewed.
- The Credit_Limit for most of the customers is less than 4500.
- There are some observations that can be considered as outliers as they are very far from the upper whisker in the boxplot. We checked and there are 892 extreme values, we gonna need further analysis to check it, but it seems not to be a wrong input.

Observarions on Total_Revolving_Bal

In [28]:
```
histogram_boxplot(df, "Total_Revolving_Bal")
```



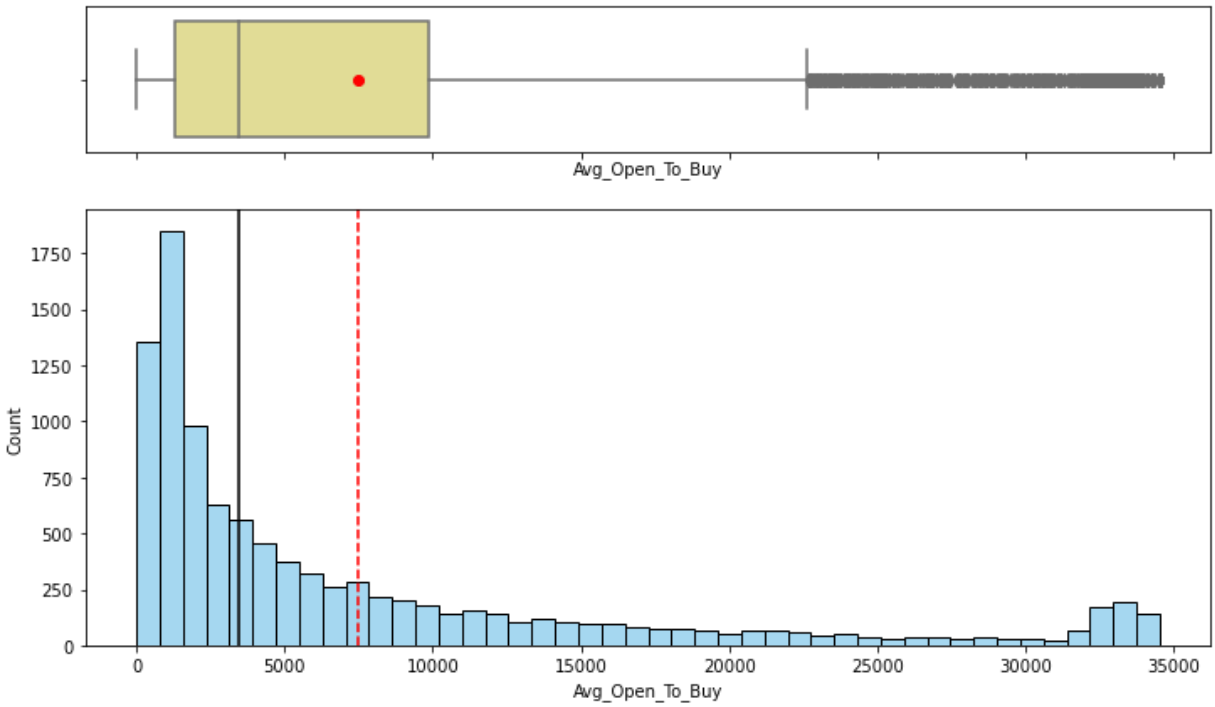
In [29]:
```
# Checking % of customers with Total_Revolving_Bal equal to Zero
df[df["Total_Revolving_Bal"] == 0]["Total_Revolving_Bal"].count() / len(
    df["Total_Revolving_Bal"]
) * 100
```

Out[29]: 24.390243902439025

- Total_Revolving_Bal distribution looks approximately normally distributed with Mean and Midian close to each other aprox 1162 and 1276 respectively.
- 25% of customers have Total_Revolving_Bal less than 359.
- The range goes from 0 to 2517, we gonna do further analysis comparing it with other variables.

Observarions on Avg_Open_To_Buy

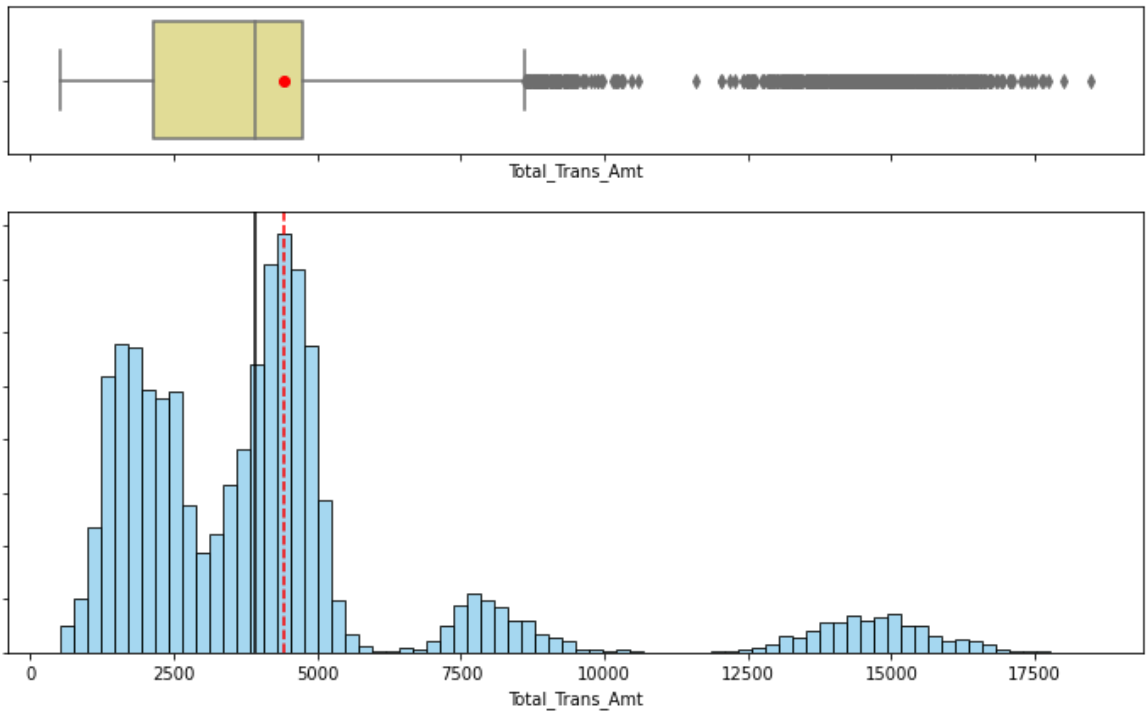
In [30]:
```
histogram_boxplot(df, "Avg_Open_To_Buy")
```



- The distribution for the Avg_Open_To_Buy is right-skewed, like expected.
- Mean is 7469 while median is around 3474.
- There are some observations that can be considered as outliers like we can see in the boxplot.
- Considering that few customers have a high Credit_Limit is expected for them to have a Avg_Open_To_Buy high as well. We gonna do further analysis comparing Credit_Limit, Total_Revolving_Bal and Avg_Open_To_Buy

Observarions on Total_Trans_Amt

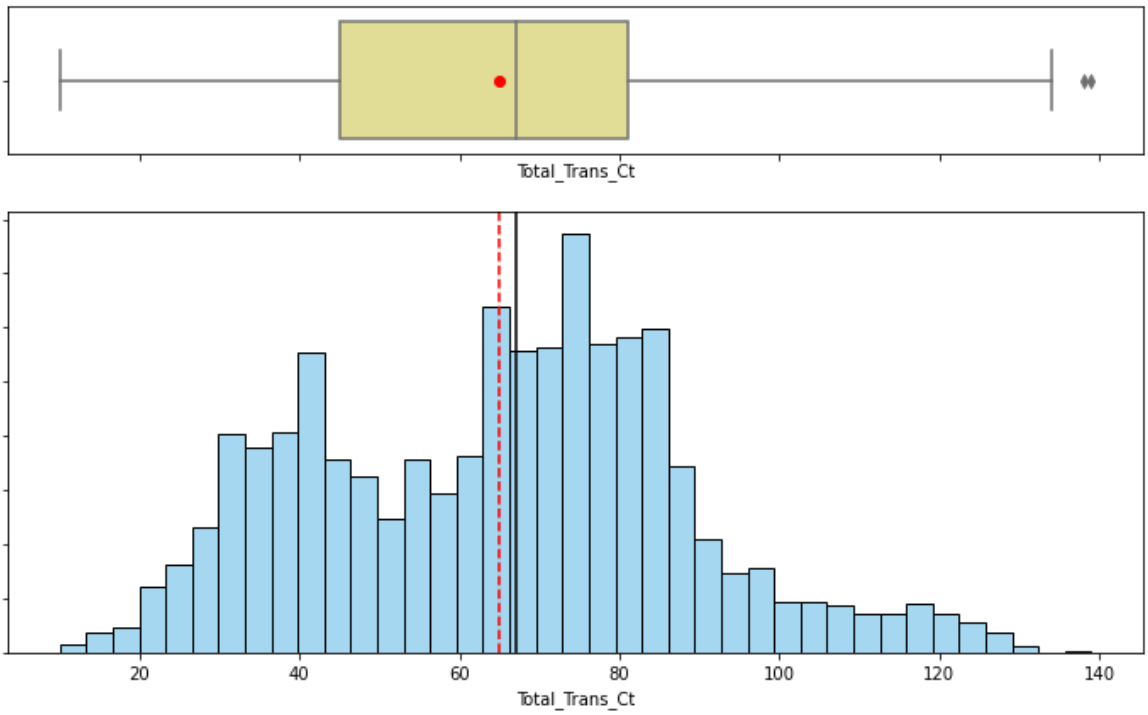
In [31]:
```
histogram_boxplot(df, "Total_Trans_Amt")
```



- Total_Trans_Amt has a huge range going from 510 to 18484.
- The most frequence Total_Trans_Amt in 12months is around mean 4404.
- 75% of customers spend less than 5000 in 12months.
- We can see some outliers on upper wiskers, but as saw before, few customers have high Credit Limit so is expected that they also will have high Total_Trans_Amt

Observarions on Total_Trans_Ct

In [32]:
```
histogram_boxplot(df, "Total_Trans_Ct")
```



In [33]:
```
df[df["Total_Trans_Ct"] > 135]
```

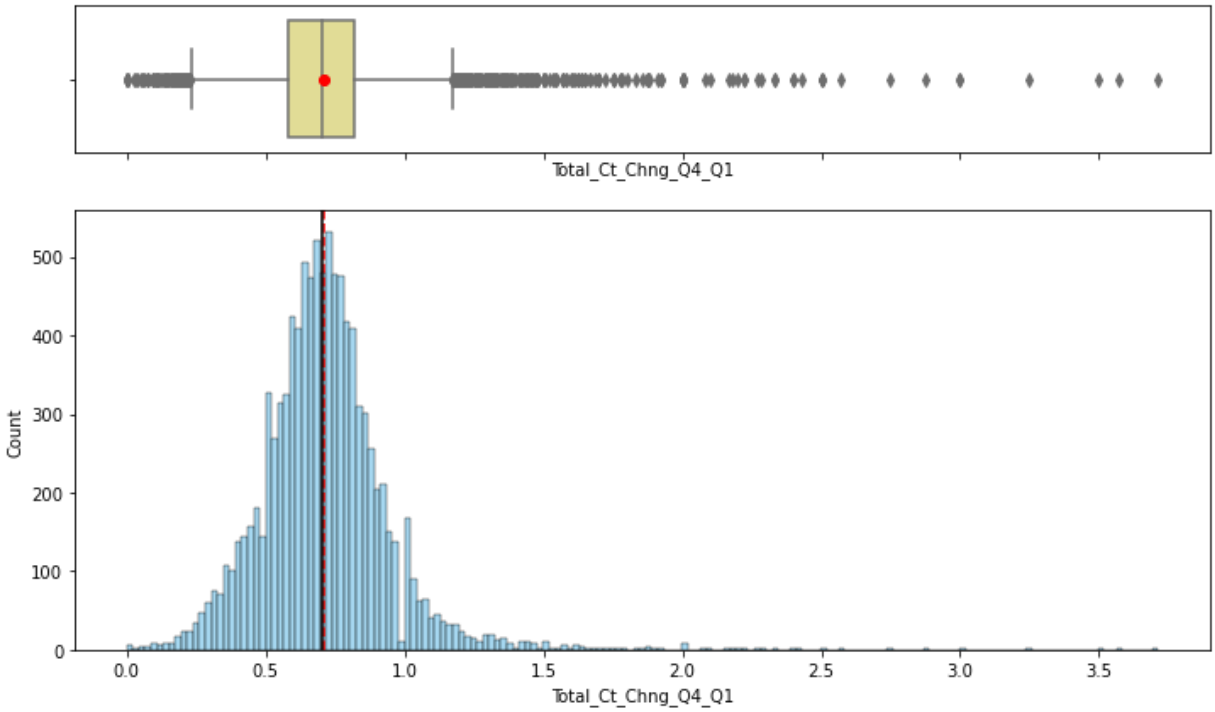
Out[33]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Statu
9324	0	41	M	3	NaN	Marrie
9586	0	56	F	1	High School	Marrie

- Total_Trans_Ct is the total count of transition on the last 12 months.
- As we can analyse, it is almost a normal distribution with mean and median very close (64.85 and 67 respectively).
- There are 2 outliers on upper wisker, but as we can analyse it, they are customers with high Credit_Limit and also high Total_Trans_Amt, so it is expected that they will have a high number of Tota_Trans_Ct

Observarions on Total_Ct_Chng_Q4_Q1

In [34]:
```
histogram_boxplot(df, "Total_Ct_Chng_Q4_Q1")
```



In [35]:
```
df[df["Total_Ct_Chng_Q4_Q1"] > 1]["Total_Ct_Chng_Q4_Q1"].count() / len(
    df["Total_Ct_Chng_Q4_Q1"]
)
```

Out[35]:
```
0.06596227905598893
```

In [36]:
```
df[df["Total_Ct_Chng_Q4_Q1"] < 0.5]["Total_Ct_Chng_Q4_Q1"].count() / len(
    df["Total_Ct_Chng_Q4_Q1"]
)
```

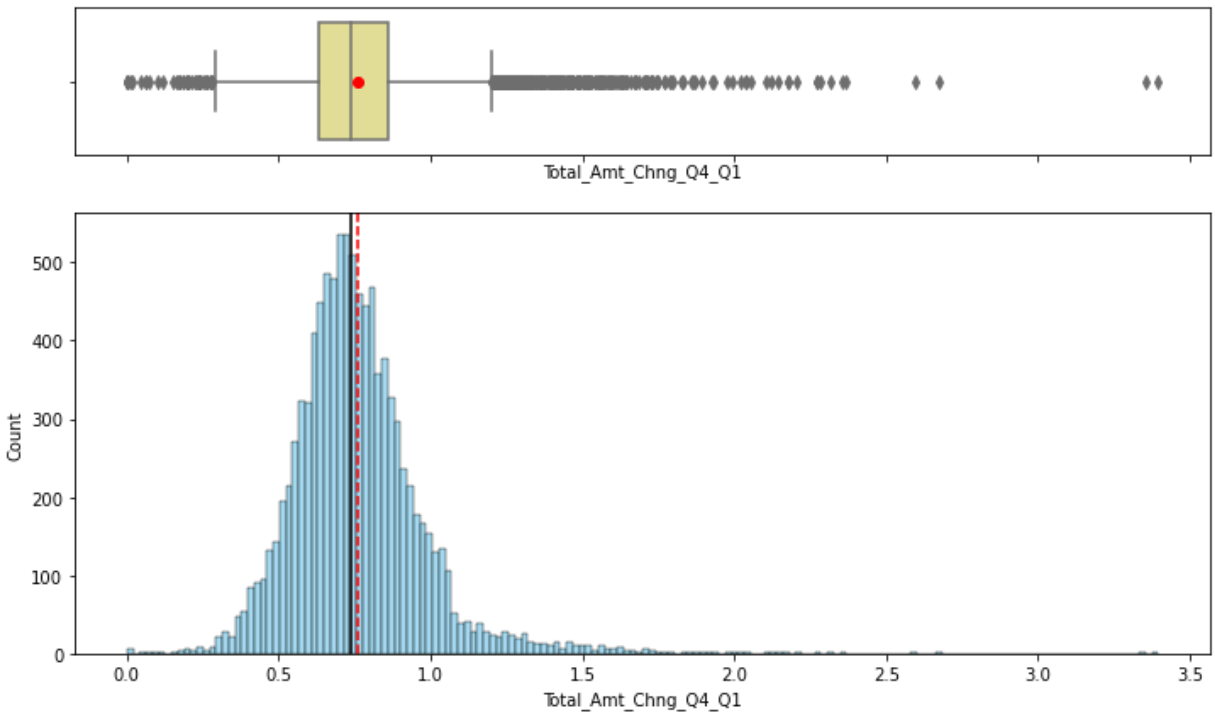
Out[36]:
```
0.13528191962081565
```

- It is a normal distribution with thin tail on both sides.

- Mean and Median is very close (0.71 and 0.70) meaning that most of customers are doing 30% less transitions on Q4 comparing to Q1.
- Only 6.6% of customers are doing more transitions compare to Q1.
- 13.5% of customers are doing less than half transitions of what they usually do.
- This is a important variable, as we can see customer behavier and try to improve it.

Observarions on Total_Amt_Chng_Q4_Q1

In [37]:
```
histogram_boxplot(df, "Total_Amt_Chng_Q4_Q1")
```



In [38]:
```
df[df["Total_Amt_Chng_Q4_Q1"] > 1]["Total_Amt_Chng_Q4_Q1"].count() / len(
    df["Total_Amt_Chng_Q4_Q1"]
)
```

Out[38]:
```
0.09805470524340872
```

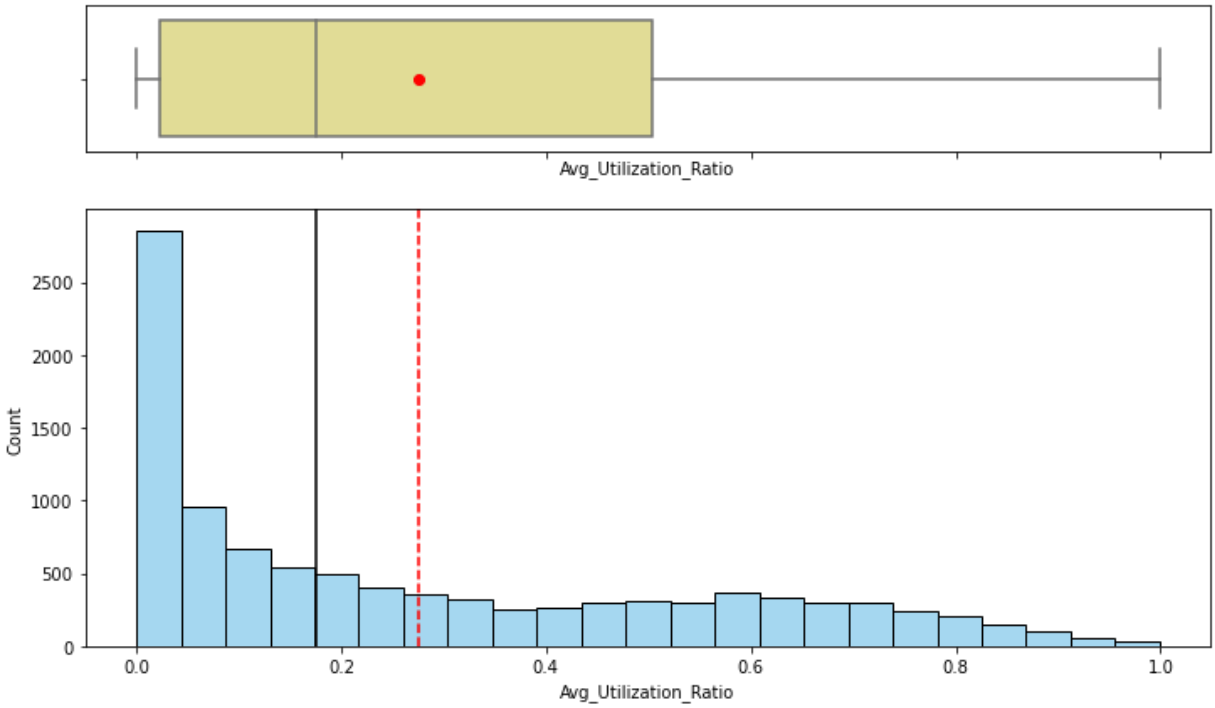
In [39]:
```
df[df["Total_Amt_Chng_Q4_Q1"] < 0.5]["Total_Amt_Chng_Q4_Q1"].count() / len(
    df["Total_Amt_Chng_Q4_Q1"]
)
```

Out[39]:
```
0.07198578058655081
```

- Total_Amt_Chng_Q4_Q1 is almost same as Total_Ct_Chng_Q4_Q1:
- It is a normal distribution with thin tail on both sides.
- Mean and Median is very close (0.76 and 0.74) meaning that most of customers are spending 25% less on Q4 comparing to Q1.
- Only 9.8 % of customers are spending more compare to Q1.
- 7.2% of customers are spending less than half of what they spent on Q1.
- This is a important variable, as we can see customer behavier and try to improve it.

Observarions on Avg_Utilization_Ratio

In [40]:
```
histogram_boxplot(df, "Avg_Utilization_Ratio")
```



In [41]:
```
df[df["Avg_Utilization_Ratio"] > 0.8]["Avg_Utilization_Ratio"].count() / len(
    df["Avg_Utilization_Ratio"]
)
```

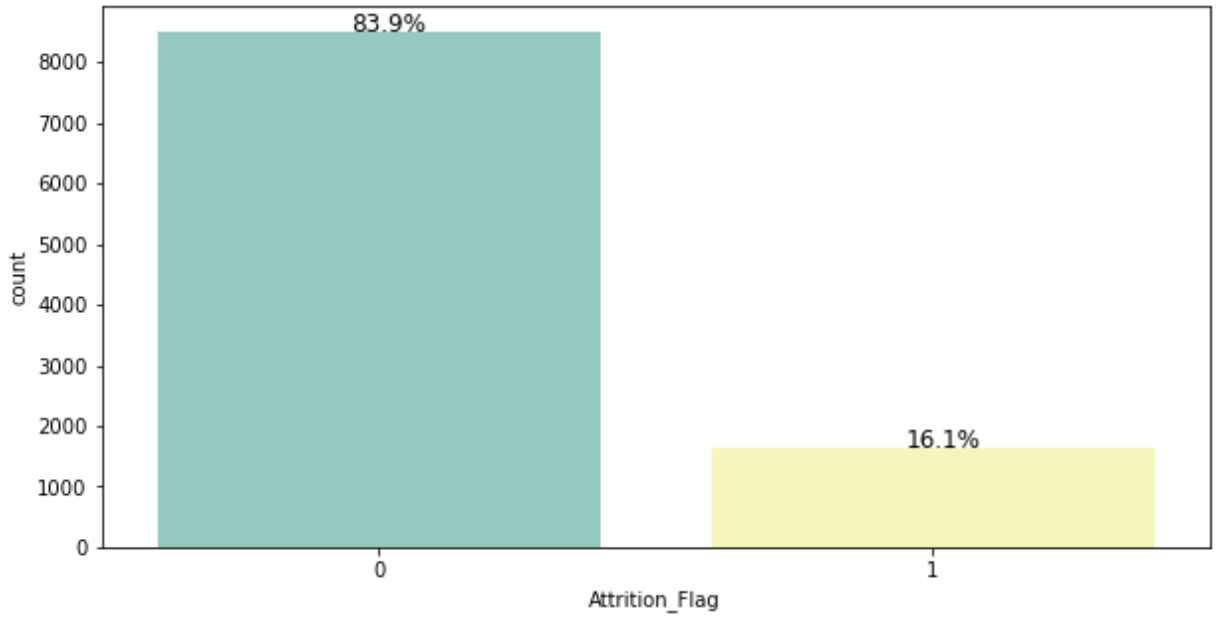
Out[41]:
```
0.046114347783153944
```

- The distribution for the Avg_Utilization_Ratio is right-skewed.
- 25% of customers are spending less than 3% of their available credit on the other hand 4.6% of customers are spending more than 80% of their available credit.
- Mean is 27% of available credit and median 17.6%.
- This is also a good variable to analyse the customer behavier.

In [42]:
```
# Function to create barplots that indicate percentage for each category.


def perc_on_bar(dataframe):
    """
    plot
    feature: categorical feature
    the function won't work if a column is passed in hue parameter
    """

    total = len(dataframe)  # length of the column
    plt.figure(figsize=(10, 5))
    ax = sns.countplot(dataframe, palette="Set3")
    for p in ax.patches:
        percentage = "{:.1f}%".format(
            100 * p.get_height() / total
        )  # percentage of each class of the category
        x = p.get_x() + p.get_width() / 2 - 0.05  # width of the plot
        y = p.get_y() + p.get_height()  # hieght of the plot

        ax.annotate(percentage, (x, y), size=12)  # annotate the percantage
    plt.show()  # show the plot
```

Observarions on Attrition_Flag

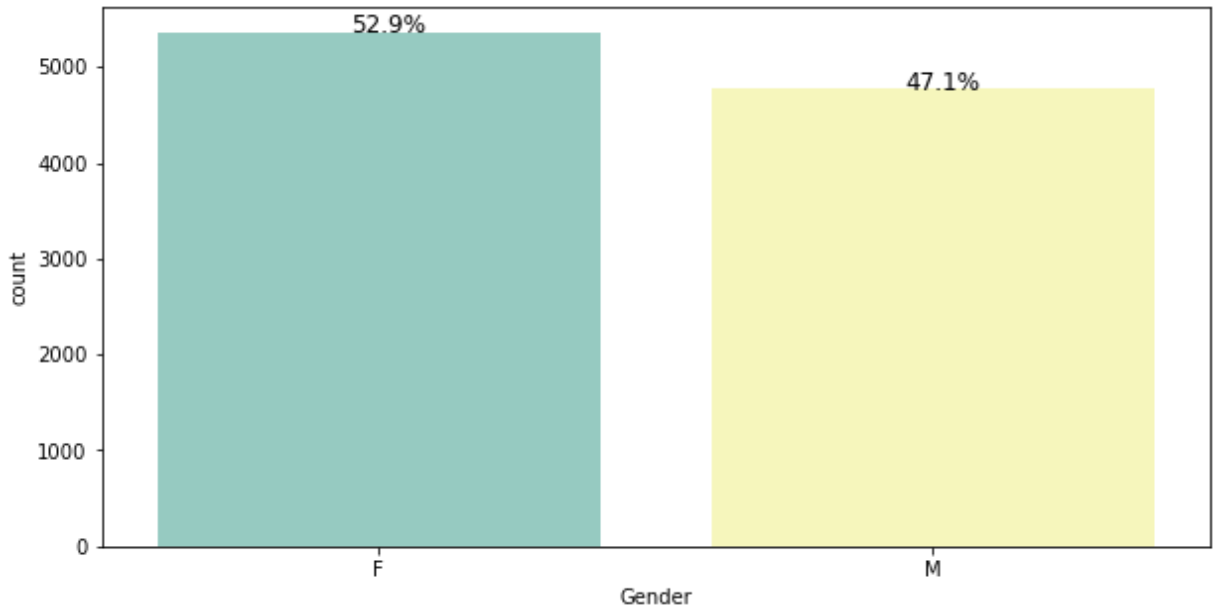
In [43]:
```
perc_on_bar(df.Attrition_Flag)
```



- Attrition_Flag is our target variable.
- We can see that our data is imbalanced.
- 83.9% of our customers are Existing Customers on the other hand 16.1% are Attrited Customer.

Observarions on Gender

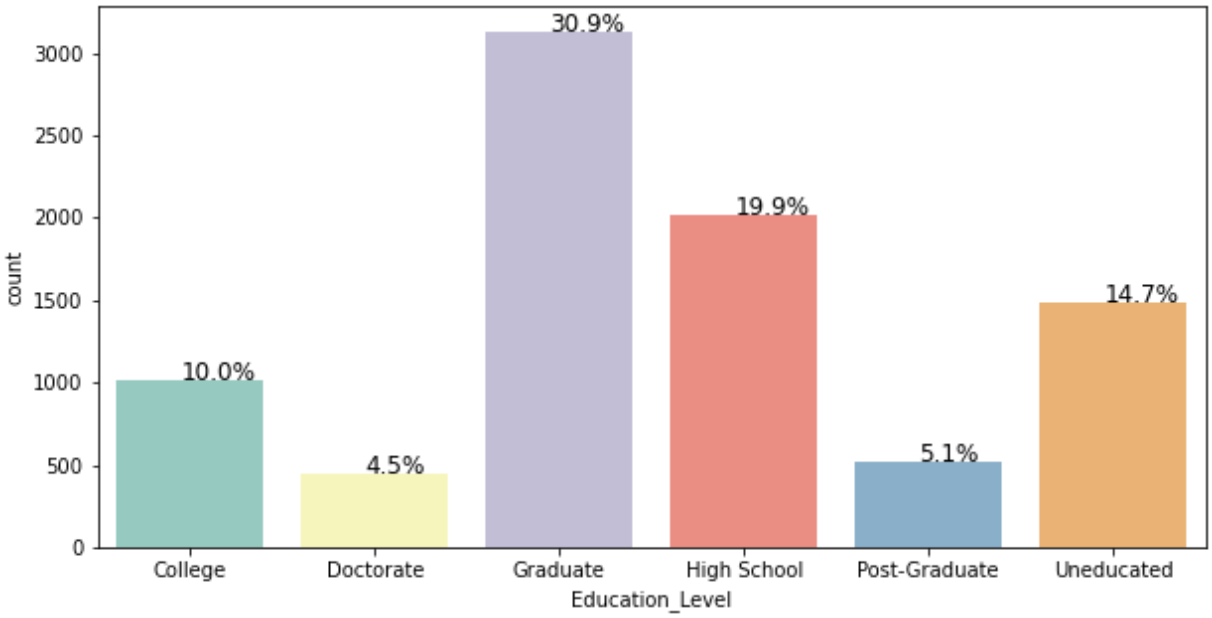
In [44]:
```
perc_on_bar(df.Gender)
```



- Female customers are more than the number of Male customers
- There are approx 53% Female customers as compared to 47% Male customers
- This might be because Females usually buy more than Male.

Observarions on Education_Level

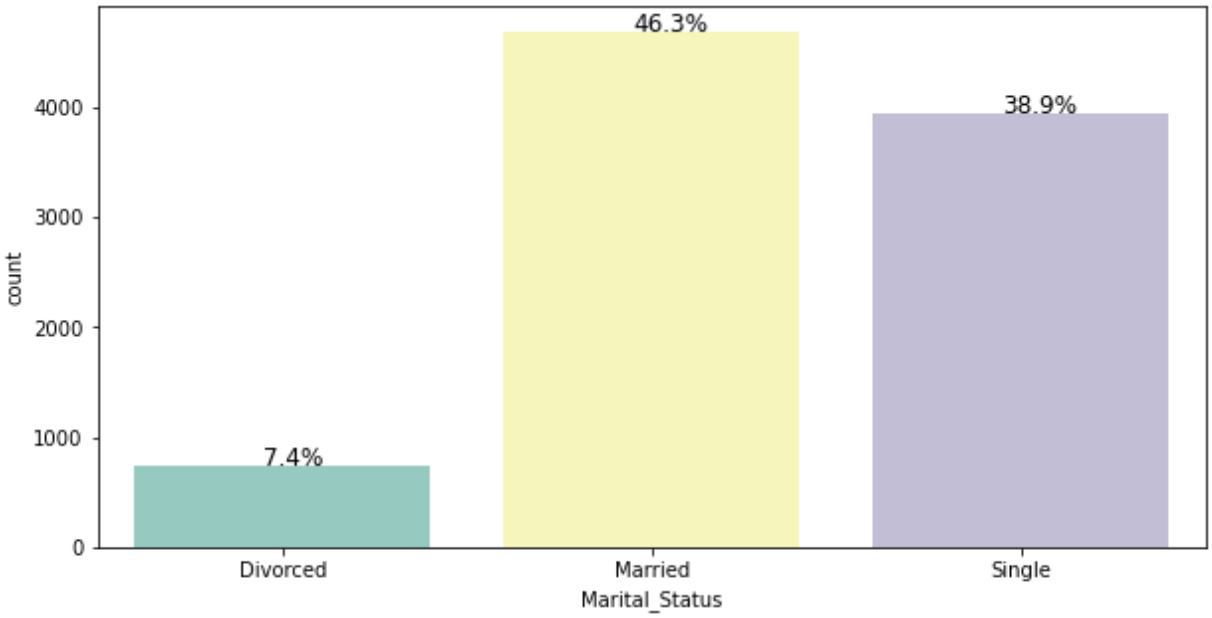
In [45]:
```
perc_on_bar(df.Education_Level)
```



- The majority of customers 30.9% are Graduate follow by 19.9% with High School level.
- 14.7% are Uneducated.
- 10% of customers are college student.

Observarions on Marital_Status

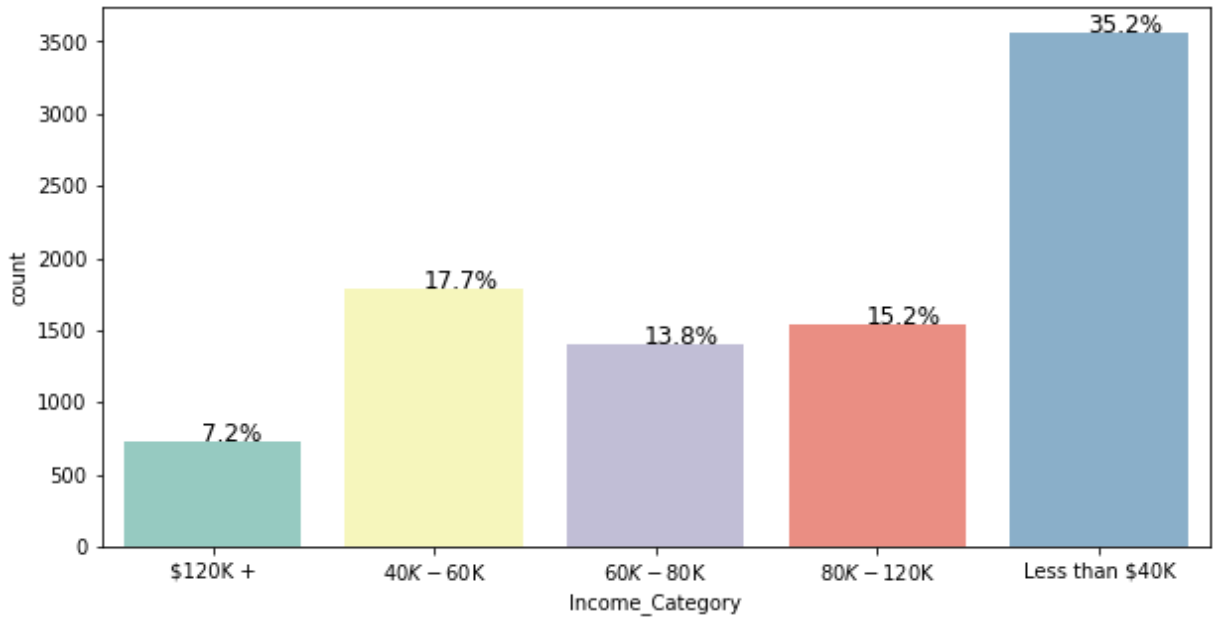
In [46]:
```
perc_on_bar(df.Marital_Status)
```



- 46% of the customer base of the company is from the married people.
- 38.9% is from Single peole.

Observarions on Income_Category

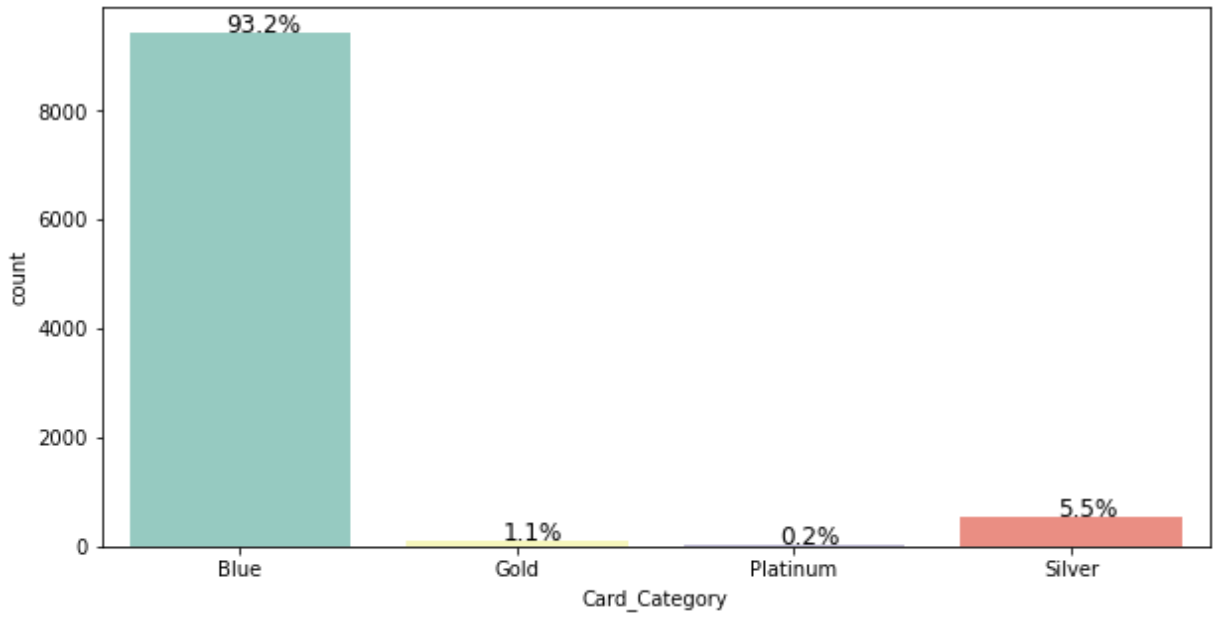
In [47]:
```
perc_on_bar(df.Income_Category)
```



- Most of the customer base have Income less than $40K (35.2%)
- 46.7% have Income between 40k and 120k

Observarions on Card_Category

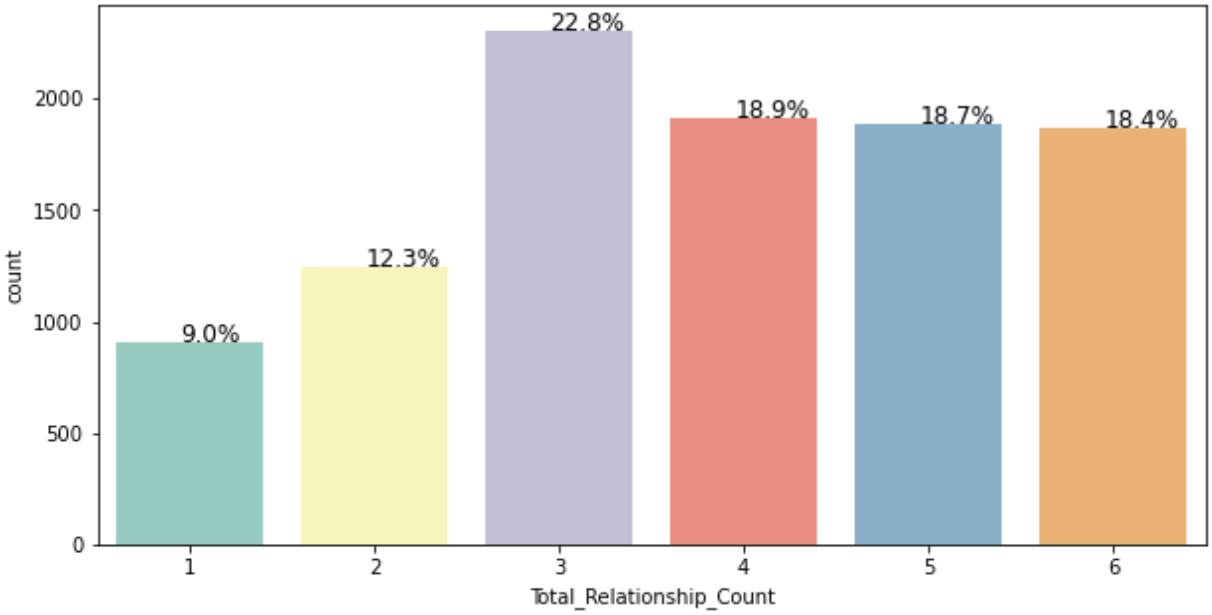
In [48]:
```
perc_on_bar(df.Card_Category)
```



- 93.2% of customers have Blue card.
- Less than 7% have Silver, Gold and Platinum
- We can explore this further and observe the 7% customers profit.

Observarions on Total_Relationship_Count

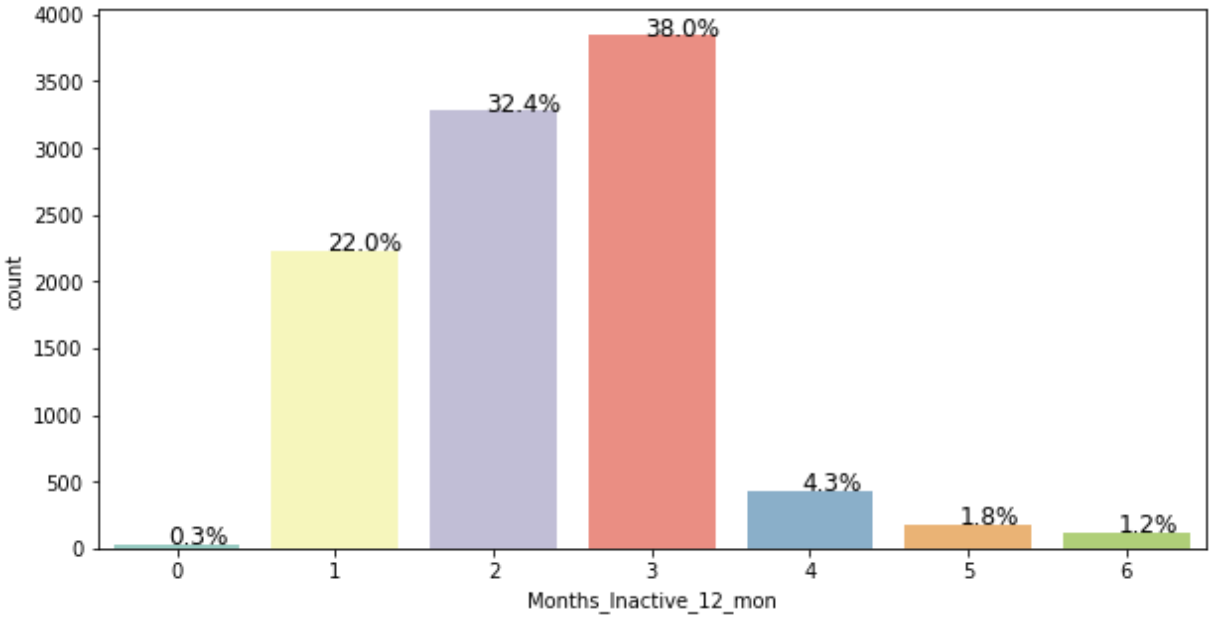
In [49]:
```
perc_on_bar(df.Total_Relationship_Count)
```



- 56% of customers held 4 or more products.
- 22.8% held 3.
- Only 9% held only 1 (Only Credit Card).
- Maybe for futher analysis, have a Knowledge of those product can also help understand customer behavier and prfit.

Observarions on Months_Inactive_12_mon

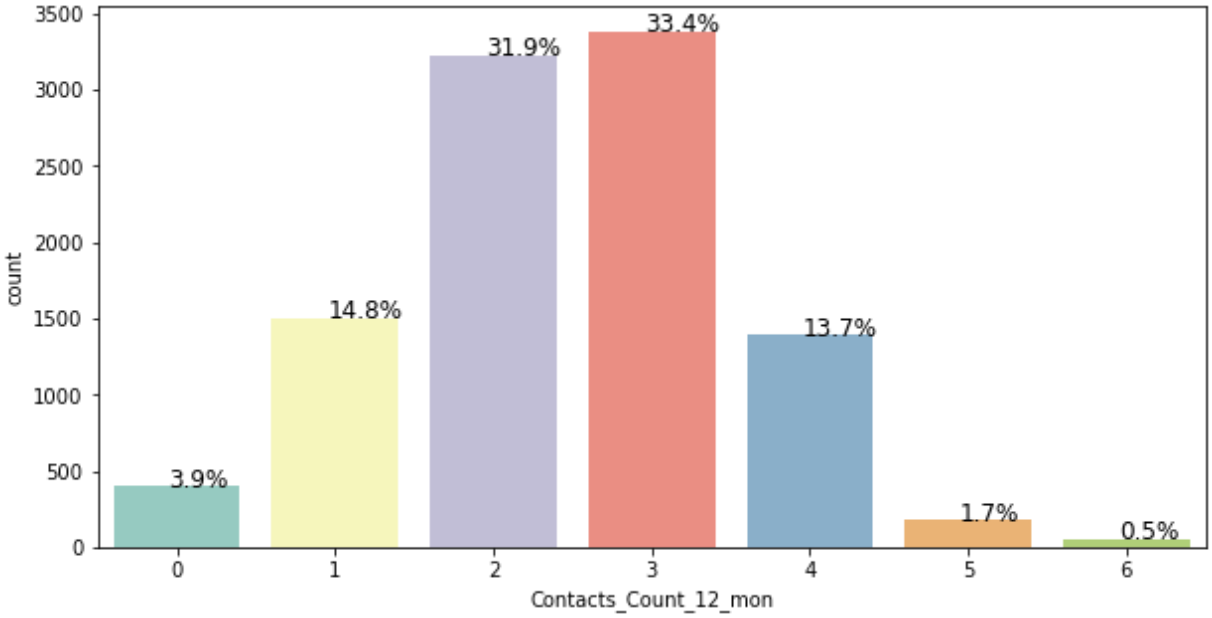
In [50]:
```
perc_on_bar(df.Months_Inactive_12_mon)
```



- Only 0.3% of customers uses their credit card every month.
- Customers used it at leat once on the past 12 months.
- 92.4% of our customer base used stay inactive for 1 to 3 months.
- We need a strategy to make customers that stay inactive for more than 3 months to use it, lets check for more analysis.

Observarions on Contacts_Count_12_mon

In [51]:
```
perc_on_bar(df.Contacts_Count_12_mon)
```



- 65% of customers had between 2 and 3 contact with the bank in the past 12 months.
- 19% of customers had 1 or 0 contact with the bank.
- 16% had more than 4 contacts. Know what the contact is about can also help to understand customer behavier and also understand if there is a relation with Attrited Customer

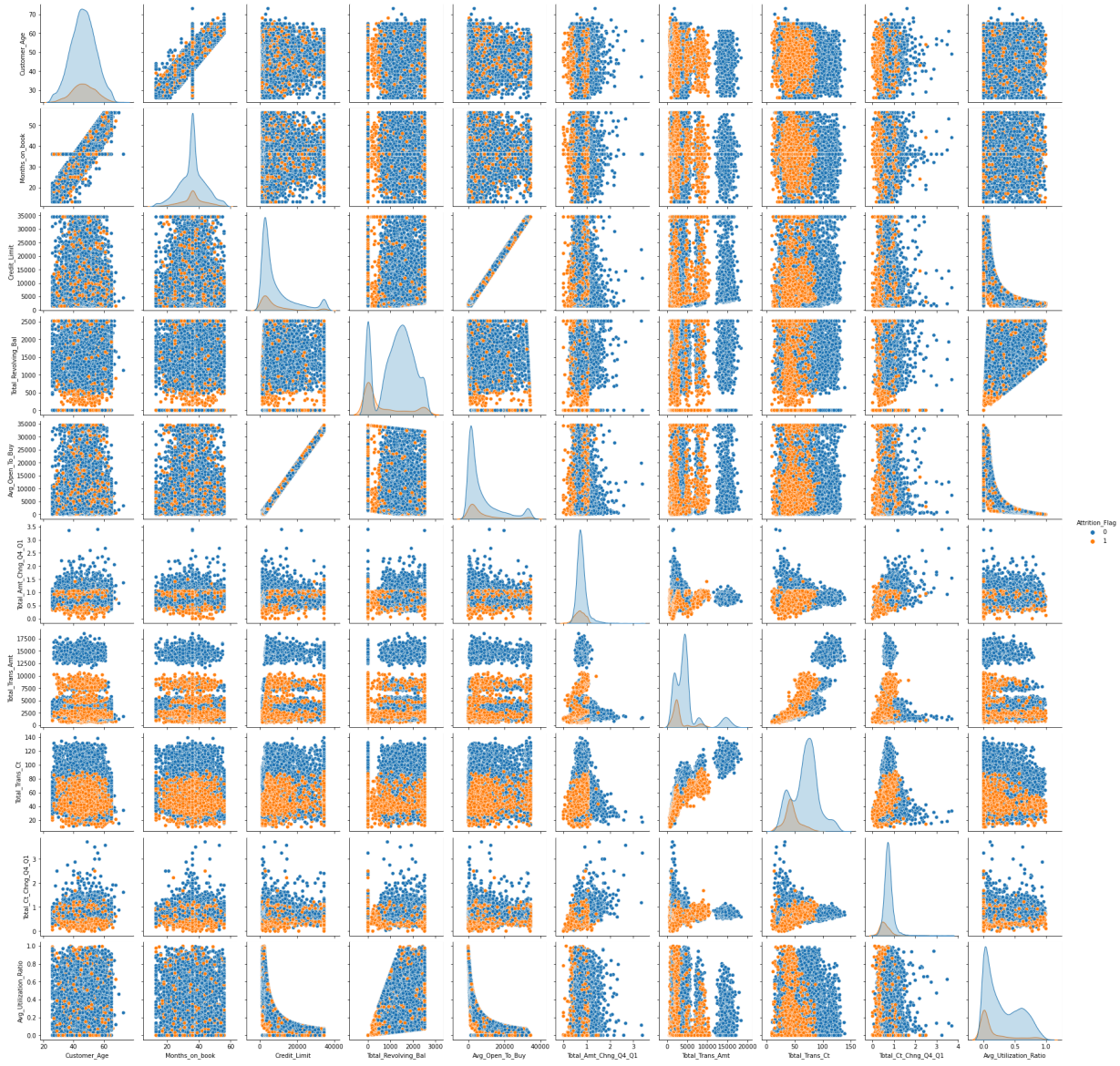
Bivariate Analysis

In [52]:
```
df.corr()["Attrition_Flag"].sort_values(ascending=False)
```

Out[52]:
```
Attrition_Flag           1.000
Customer_Age             0.018
Months_on_book           0.014
Avg_Open_To_Buy         -0.000
Credit_Limit            -0.024
Total_Amt_Chng_Q4_Q1    -0.131
Total_Trans_Amt         -0.169
Avg_Utilization_Ratio   -0.178
Total_Revolving_Bal     -0.263
Total_Ct_Chng_Q4_Q1     -0.290
Total_Trans_Ct          -0.371
Name: Attrition_Flag, dtype: float64
```

In [53]:
```
sns.pairplot(data=df, hue="Attrition_Flag")
```

Out[53]:
```
<seaborn.axisgrid.PairGrid at 0x1337993cc70>
```



As expected:

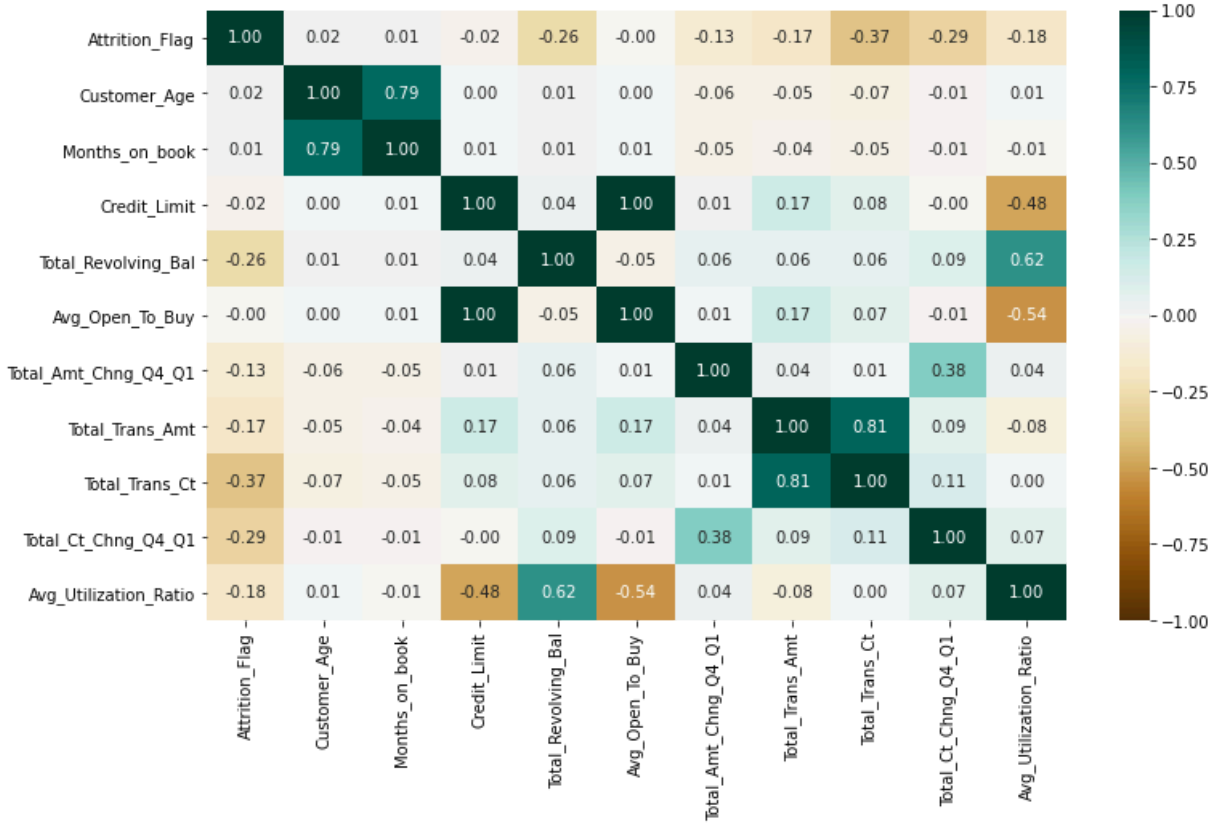
- Month_on_book shows a positive correlation with Age.
- Avr_open_to_buy shows a positive and strong correlation of 1 with Credit_Limited (we gonna drop Avr_open_to_buy, the 1 correlation signifies that both variables move in the same direction).
- Avg_Utilization_Ratio have a negative correlation with Credit_Limited and Avr_open_to_buy

Attrited Customer (account is closed) shows some partner as:

- Total_Revolving_Bal has a negative correlation with customers who closed account, meaning that small the amount on Total_Revolvint_Bal more likehood is the customer to close the account.
- Customers that spend less than 11000 in a year (Total_Trans_Amt) and spent less than 1 on Q4 compare to Q1, usualy close account.
- Customers that spend less on Q4 compared to Q1 is more likehood to close account.

In [54]:
```
plt.figure(figsize=(12, 7))
sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="BrBG")
plt.show
```

Out[54]: <function matplotlib.pyplot.show(close=None, block=None)>

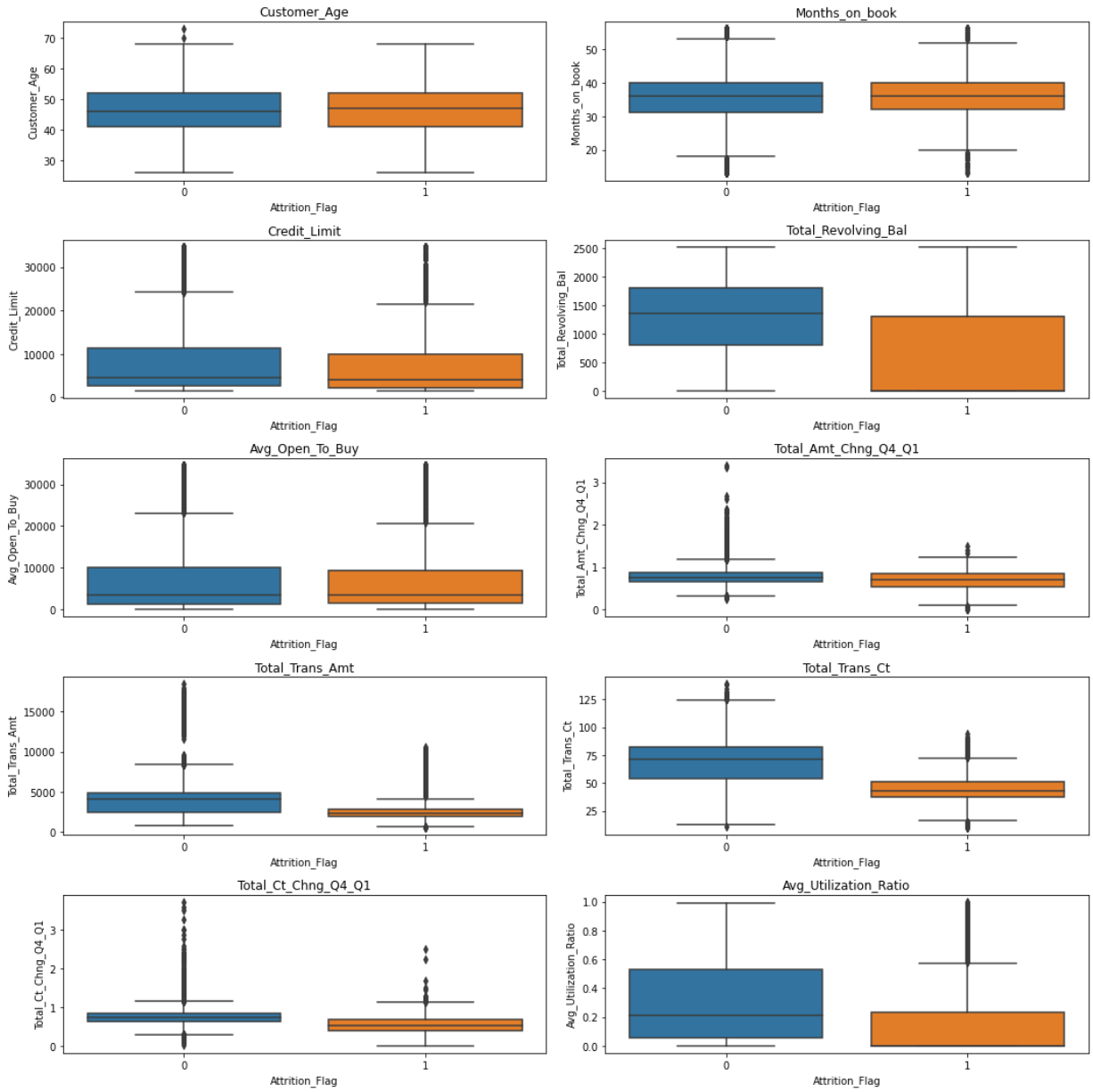


- As expected, customer age and months on book have a high positive correlation. We will do more analysis to see how does it influence on other variables.
- We can drop one of the columns in Avg Open to Buy or Credit Limit as they are `perfectly correlated` and will not add value to our analysis.
- Total Trans Ct is positively correlated with Total Trans Amt which can be expected as customers with higher number of transition might spend more than customers with lower number of transition.
- Customers that carries over from one month to the next (Total Revolving Bal) have spent a high value of the available credit (Avg Utilization Ratio), this explain the positive correlation between them.
- As also expected, Avg Utilization Ratio have a negative correlation with Credit Limit and Avg Open to Buy. (More customers buy, less is the average open to buy.

In [55]:
```
# Creating a lis of numerical variables
num_col = []
for col in df.columns:
    if col not in cat_col:
        num_col.append(col)

num_col.remove("Attrition_Flag")
```

In [56]:
```
# Boxplot for numerical variables
plt.figure(figsize=(15, 15))
for i, variable in enumerate(num_col):
    plt.subplot(len(num_col) / 2, 2, i + 1)
    sns.boxplot(df["Attrition_Flag"], df[variable])
    plt.tight_layout()
```

```
    plt.title(variable)
plt.show()
```



- We can see that Customers with ***Low*** Total_Revolving_Bal, Total_Trans_Amt, Total_Trans_Ct, Total_Ct_Chng_Q4_Q1, Total_Amt_Chng_Q4_Q1 and Avg_Utilization_Ratio are more likehood to churn.
- Clearly we can see a pattern on the customer behavier that shows when they gonna churn, we need to work on this variables to keep our customer.
- Other variables doesn't show much influence on customer churn or not.

Let's define one more function to plot stacked bar charts

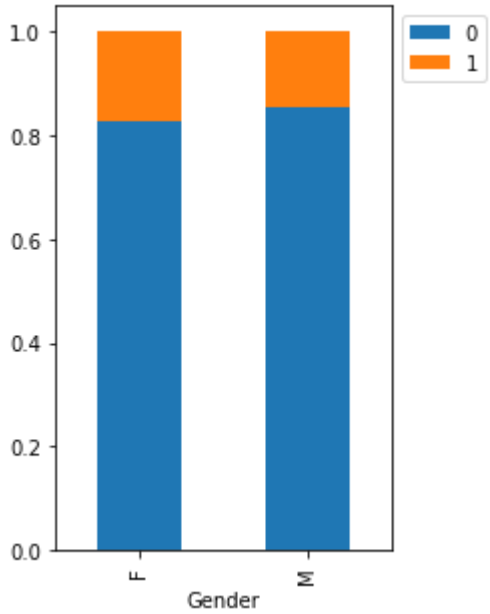
In [57]:
```
# function to plot stacked bar chart


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

    data: dataframe
    predictor: independent variable
    target: target variable
```

```
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_valu
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 1, 5))
    plt.legend(
        loc="lower left",
        frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

Attrition_Flag vs Gender

In [58]:
```
stacked_barplot(df, "Gender", "Attrition_Flag")
```

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



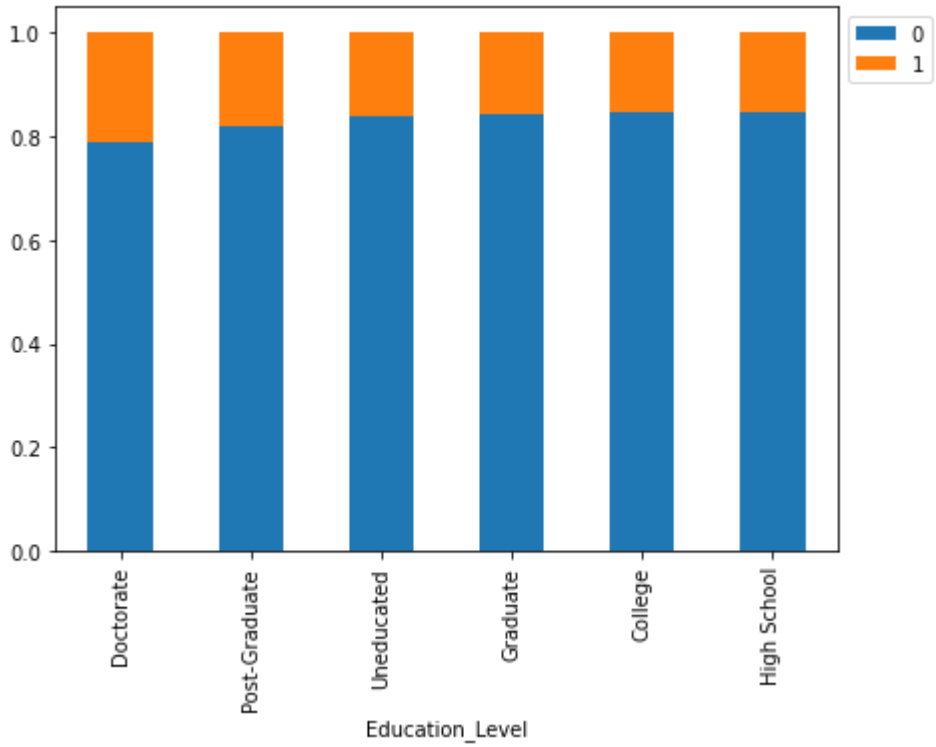
- Customer Gender does not seems to make that much influence on customer behavior of closing the account.
- Only 2% of Females customer close account more than Males.

Attrition_Flag vs Education_Level

In [59]:
```
stacked_barplot(df, "Education_Level", "Attrition_Flag")
```

```
Attrition_Flag      0     1   All
Education_Level
```

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

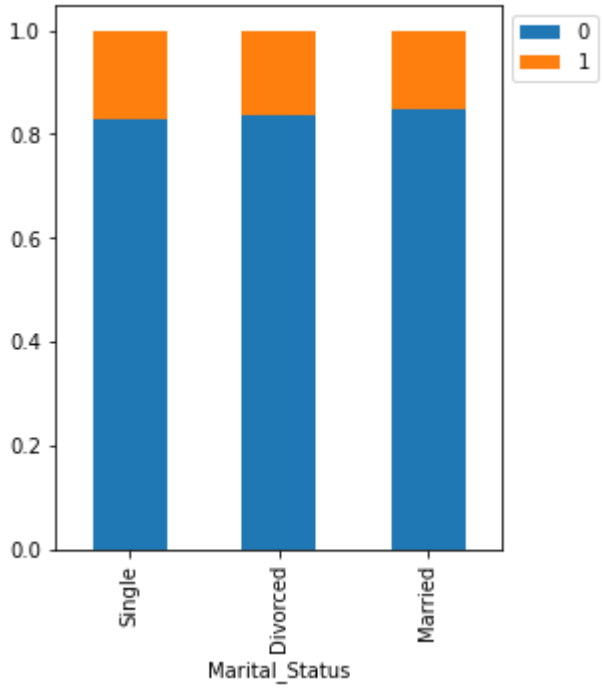


- Considering Education_Level the mean of customers closing account is 15%
- More than 20% of Doctorate customers closed account, we need further analysis to understand this customers behavior.

Attrition_Flag vs Marital_Status

In [60]:
```
stacked_barplot(df, "Marital_Status", "Attrition_Flag")
```

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

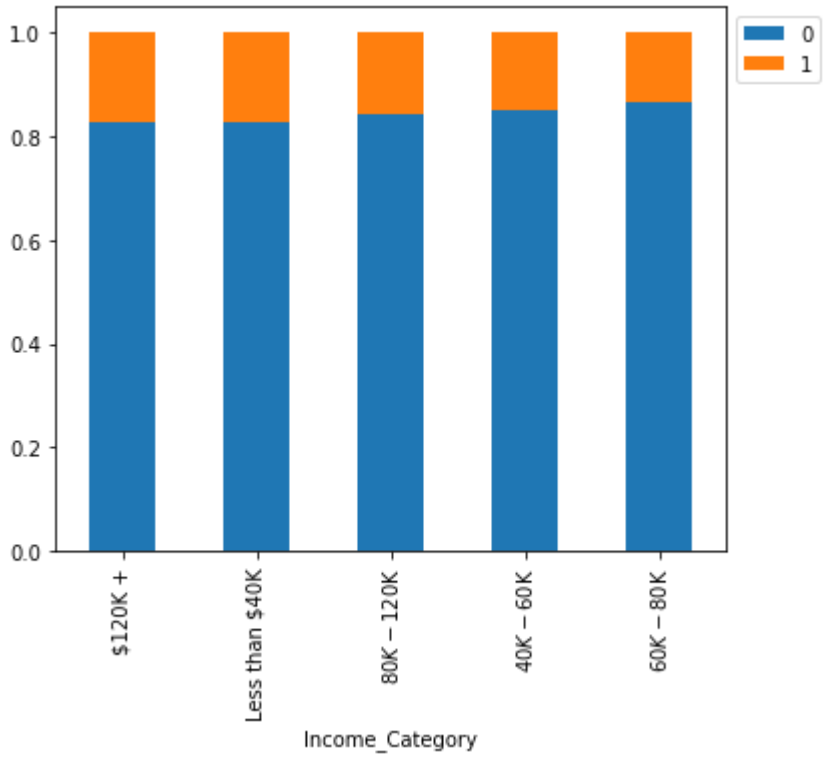


- The cancelation rate for customers with Marital Status as Single is higher as compared to the other customers, but the difference is not considerable.

Attrition_Flag vs Income_Category

In [61]:
```
stacked_barplot(df, "Income_Category", "Attrition_Flag")
```

```
Attrition_Flag      0     1   All
Income_Category
All              7575  1440  9015
Less than $40K   2949   612  3561
$40K - $60K      1519   271  1790
$80K - $120K     1293   242  1535
$60K - $80K      1213   189  1402
$120K +           601   126   727
------------------------------------------------------------------------------
------------------------------------------
```

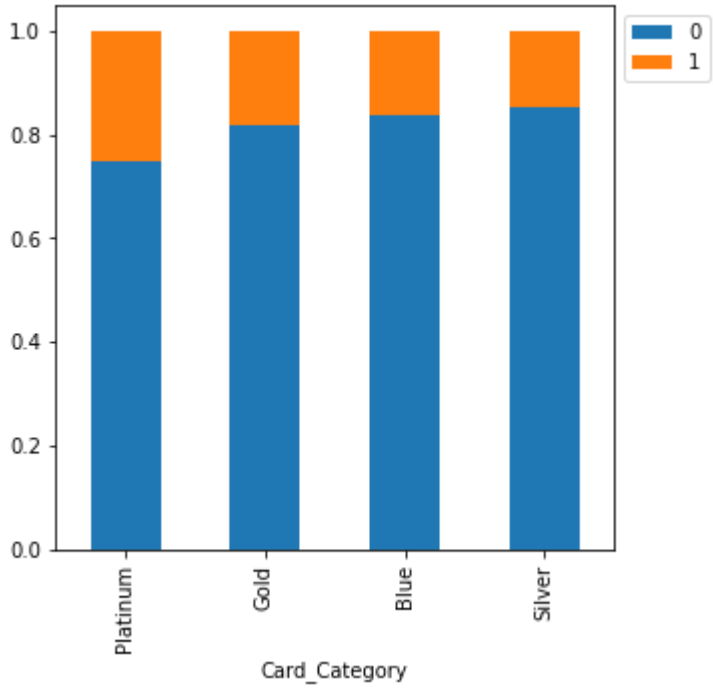


- Customers with Income greater than $120k are more likehood to cancel the account.
- On the other hand, customers with Income between 60k to 80k are less likehod to cancel the account.
- We need to keep in mind that the difference is not considerable.

Attrition_Flag vs Card_Category

In [62]:
```
stacked_barplot(df, "Card_Category", "Attrition_Flag")
```

```
Attrition_Flag     0     1    All
Card_Category
All             8500  1627  10127
Blue            7917  1519   9436
Silver           473    82    555
Gold              95    21    116
Platinum          15     5     20
------------------------------------------------------------------------------
------------------------------------------
```

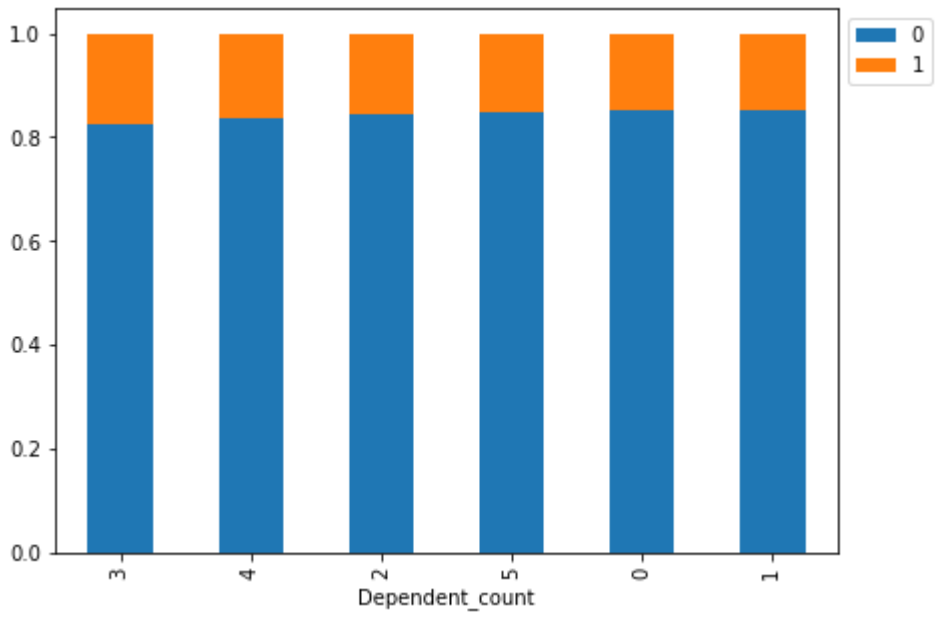


- 25% of Platinum customers churn., followed by 18% of Gold customers.
- We have seen that customer wtih greater income, high level of education, Platinum and Gold Card, are more likehood to cancel the credit card.
- The company can target this customers more and modify the Credit Card beneficis

Attrition_Flag vs Dependent_count

In [63]:
```
stacked_barplot(df, "Dependent_count", "Attrition_Flag")
```

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

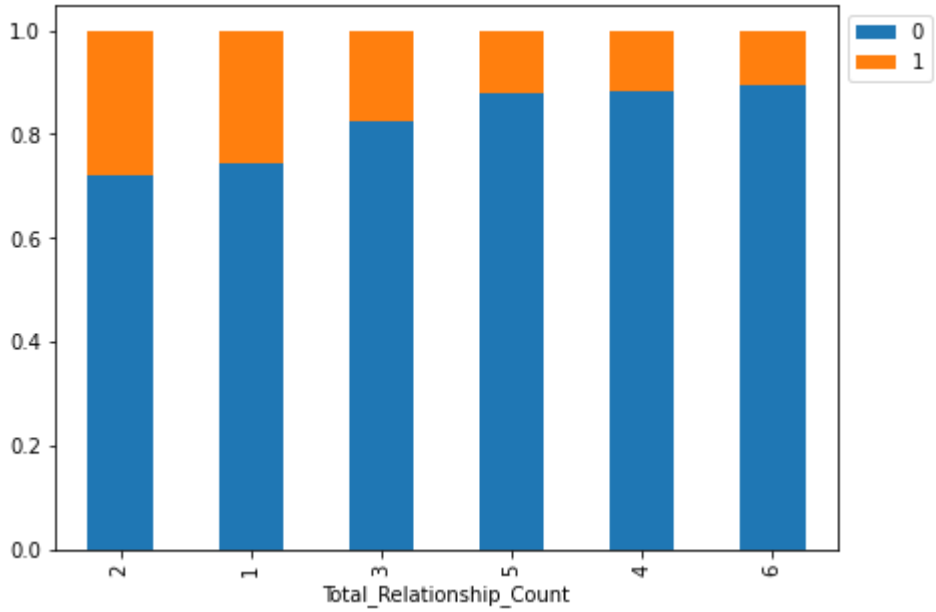


- All categories have almost the same rate of closing account (around 16).
- It does not seems that Dependent Count makes difference for us in understanding customer behavior.

Attrition_Flag vs Total_Relationship_Count

In [64]:
```
stacked_barplot(df, "Total_Relationship_Count", "Attrition_Flag")
```

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

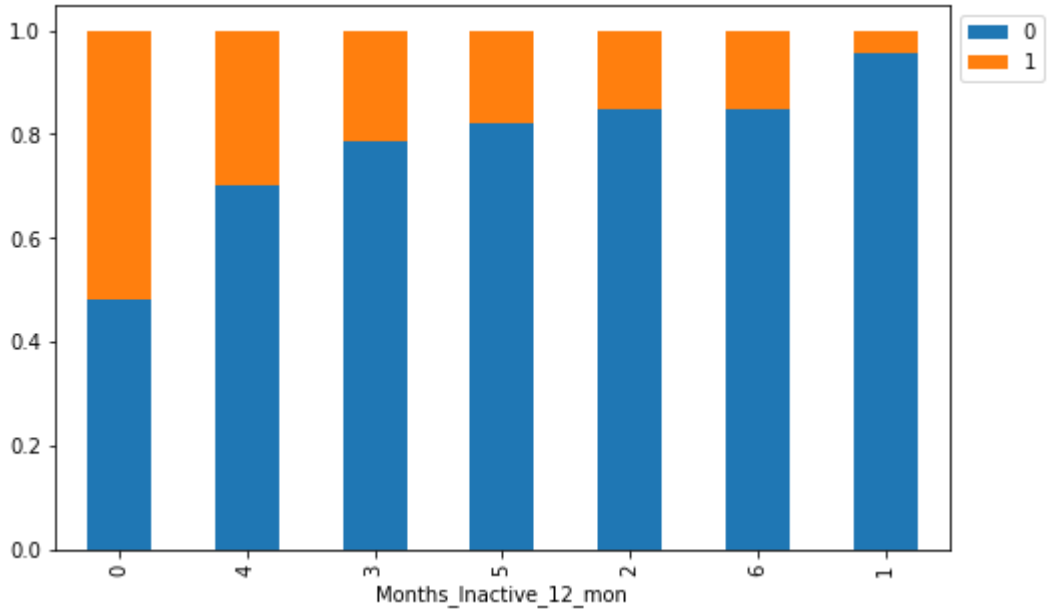


- The cancelation rate for customers with only 1 or 2 products held is higher as compared to the customers that held more.
- The company should work in make customers hold more products keeping them loyal.

Attrition_Flag vs Months_Inactive_12_mon

In [65]:
```
stacked_barplot(df, "Months_Inactive_12_mon", "Attrition_Flag")
```

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

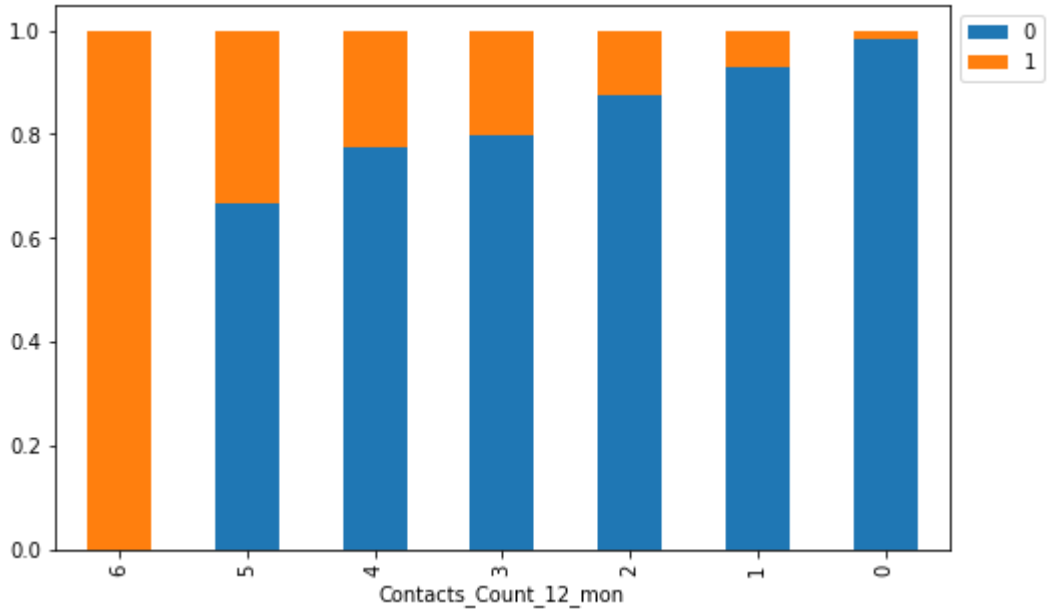


- The cancelation rate of customers is higher if Months_Inactive_12_mon is equal to 0. That is a surprese and needs further analysis to understand it.
- If customer is Inactive for more than 1 month, greater is the chance of cancelation, company need a strategy to keep customers using the credit card.

Attrition_Flag vs Contacts_Count_12_mon

In [66]:
```
stacked_barplot(df, "Contacts_Count_12_mon", "Attrition_Flag")
```

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



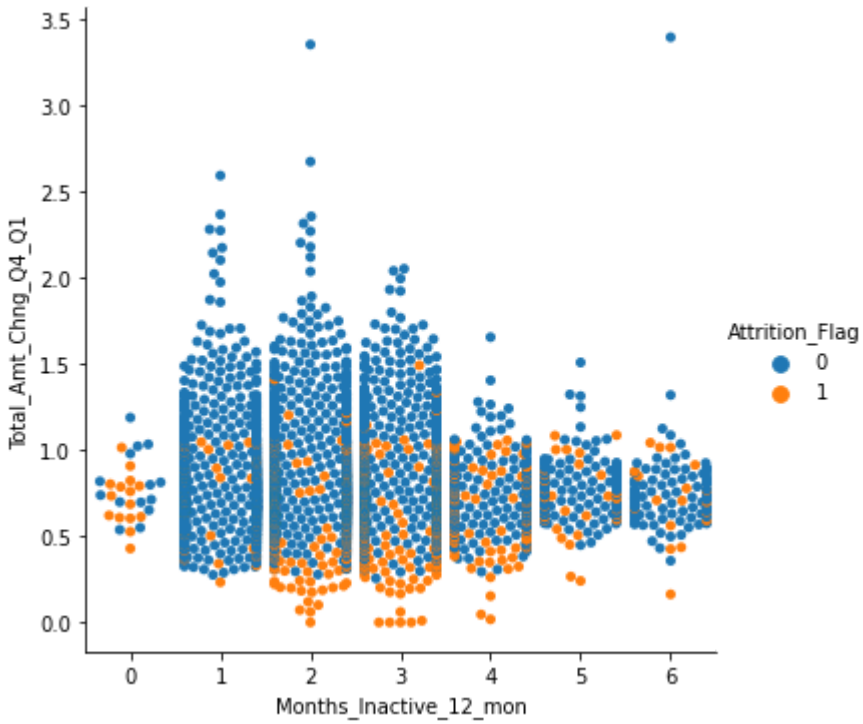
- All the customers that contact the bank 6 times on the last 12 months cancel their account.
- We can see that more the customer contact, more is the likehood to cancel.
- For further analysis, we should consider mapping the reason for the contact, reason for cancelation and make a action plan treat it.

Multivariate Analysis

Total_Amt_Chng_Q4_Q1 VS Months_Inactive_12_mon per Attrition_Flag

In [67]:
```
sns.catplot(
    x="Months_Inactive_12_mon",
    y="Total_Amt_Chng_Q4_Q1",
    hue="Attrition_Flag",
    kind="swarm",
    data=df,
)
```

Out[67]:
```
<seaborn.axisgrid.FacetGrid at 0x13304bfe610>
```

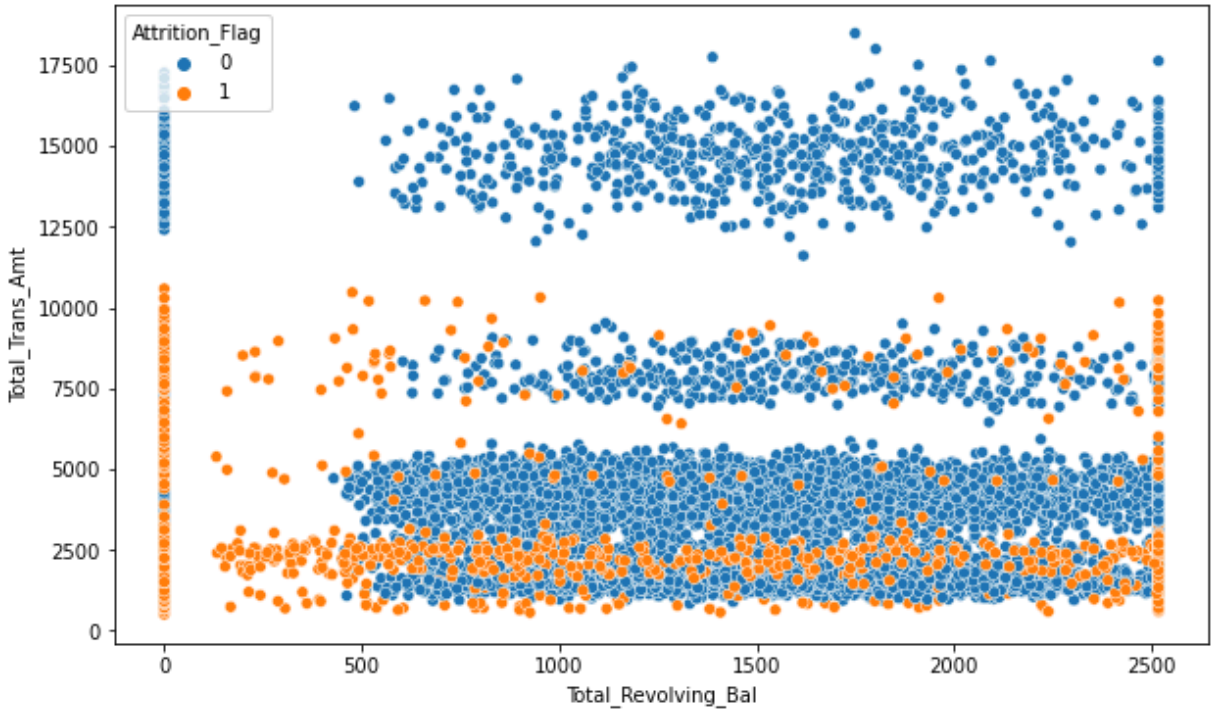


- Customers that spent less than 0.3 on Q4 compared to Q1 are more like hood to churn
- Increase the usage of Credit Card is a way to keep customers.

Total Trans Amt VS Total Revolving Bal per Attrition_Flag

In [68]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(
    df["Total_Revolving_Bal"], df["Total_Trans_Amt"], hue=df["Attrition_Flag"
)
```

Out[68]:
```
<AxesSubplot:xlabel='Total_Revolving_Bal', ylabel='Total_Trans_Amt'>
```

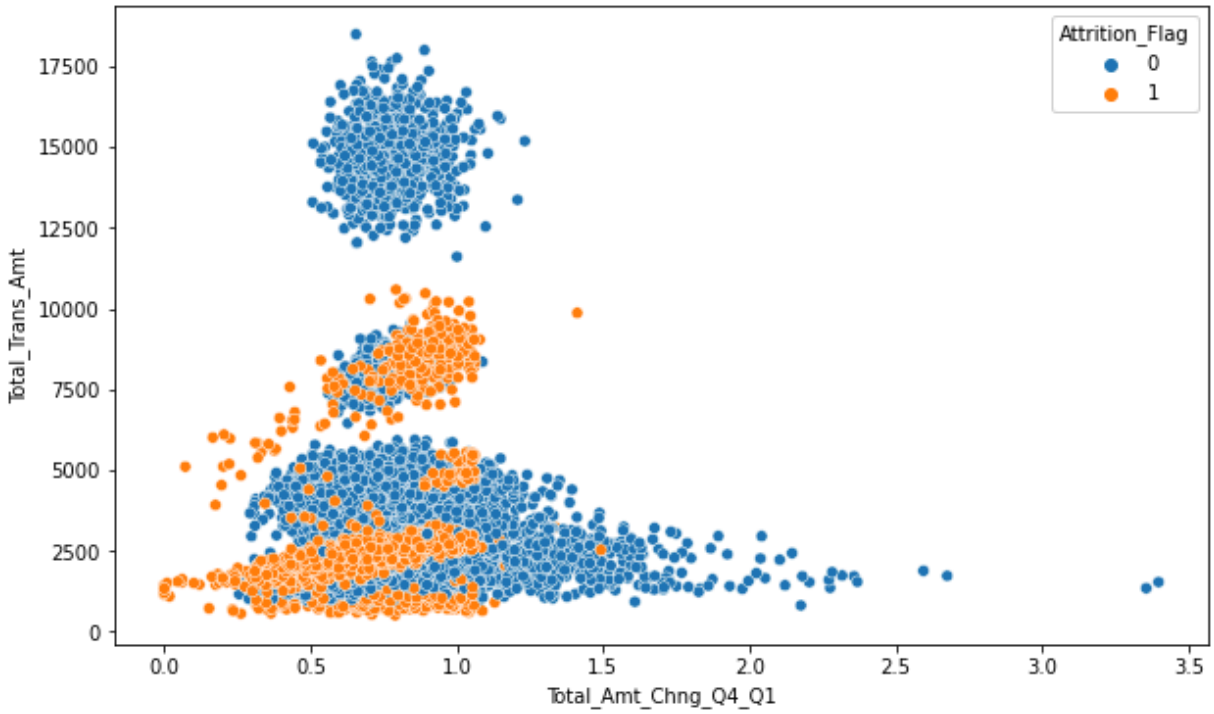


- Customers with Total Revolving Balance 0 and that spent less than 11000 on the past 12 month, churn.
- Customers with Total Revolving Balance less than 500 and Total Trans Amt less than 11000, usually cancel they account.

Total Trans Amt VS Total_Amt_Chng_Q4_Q1 per Attrition_Flag

In [69]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(
    df["Total_Amt_Chng_Q4_Q1"], df["Total_Trans_Amt"], hue=df["Attrition_Flag
)
```

Out[69]:
```
<AxesSubplot:xlabel='Total_Amt_Chng_Q4_Q1', ylabel='Total_Trans_Amt'>
```

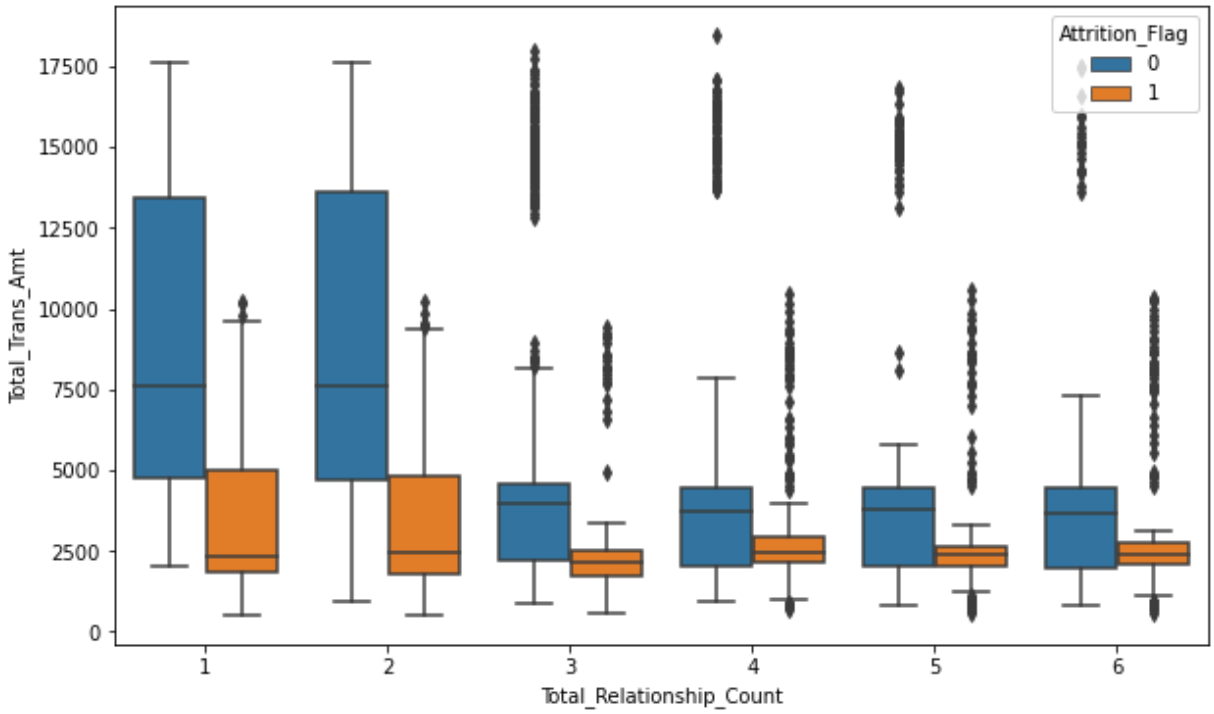


- We can see some pattern that needs further analyses. Customers with Q4/Q1 less than 1 and Total Trans less 3000 or Customers with Total Trans between 6000 and 11000 with Q4/Q1 less than 1 are more like hood to churn.

Total Trans Amt VS Total_Relationship_Count per Attrition_Flag

In [70]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(
    df["Total_Relationship_Count"], df["Total_Trans_Amt"], hue=df["Attrition_
)
```

Out[70]: <AxesSubplot:xlabel='Total_Relationship_Count', ylabel='Total_Trans_Amt'>

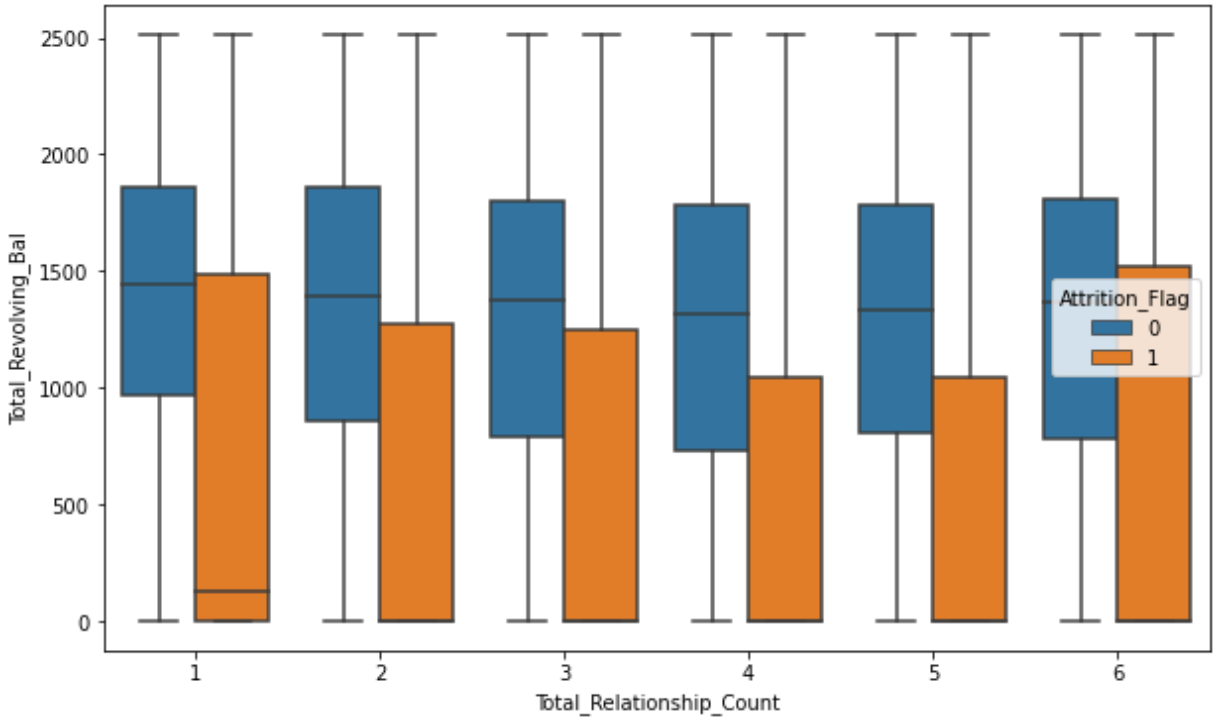


- Customers with 1 or 2 Total Relationship and less than 5000 Total Trans Amt are more likehood to cancel the account.
- Customers with high Total Relationship but Low than 3000 Total Trans Amt are also more likehood to cancel the account.

Total Revolving Bal VS Total_Relationship_Count per Attrition_Flag

In [71]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(
    df["Total_Relationship_Count"], df["Total_Revolving_Bal"], hue=df["Attrit
)
```

Out[71]: <AxesSubplot:xlabel='Total_Relationship_Count', ylabel='Total_Revolving_Bal'>

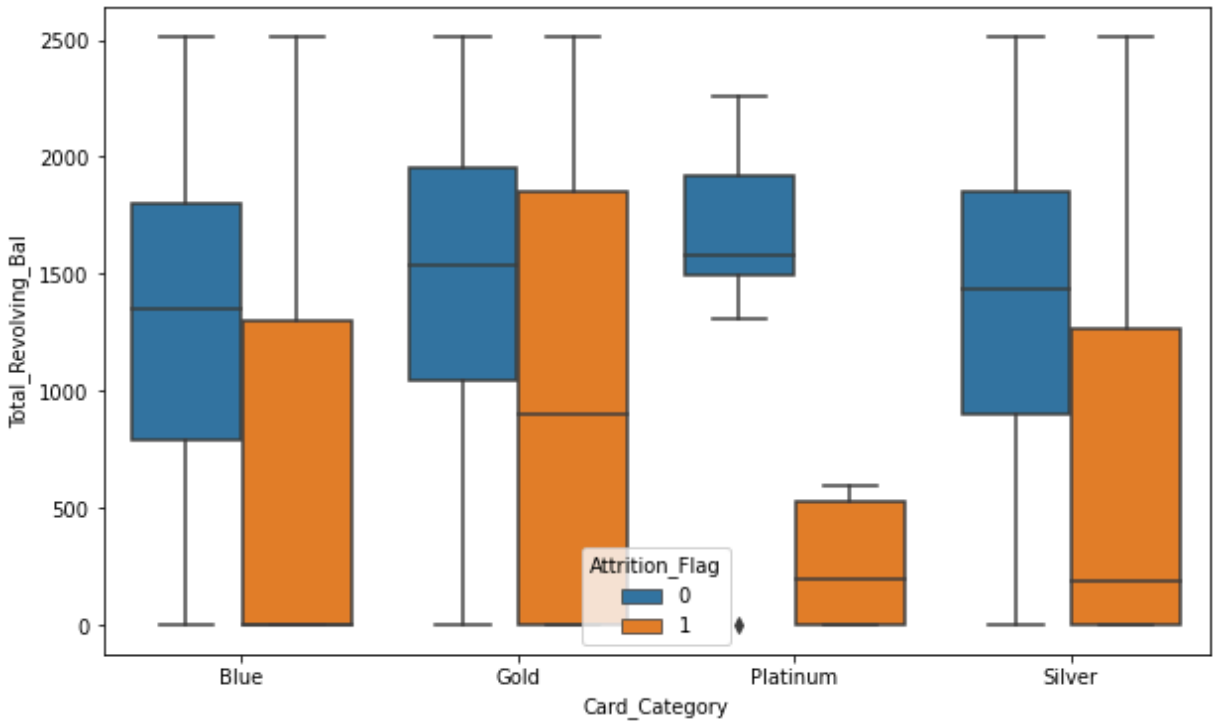


- Customers with ZERO Total_Revolving_Bal has high probabilite of canceling the credit card, it does not matter the total Relationship

Total Revolving Bal VS Card Category per Attrition_Flag

In [72]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(df["Card_Category"], df["Total_Revolving_Bal"], hue=df["Attrition
```

Out[72]:
```
<AxesSubplot:xlabel='Card_Category', ylabel='Total_Revolving_Bal'>
```



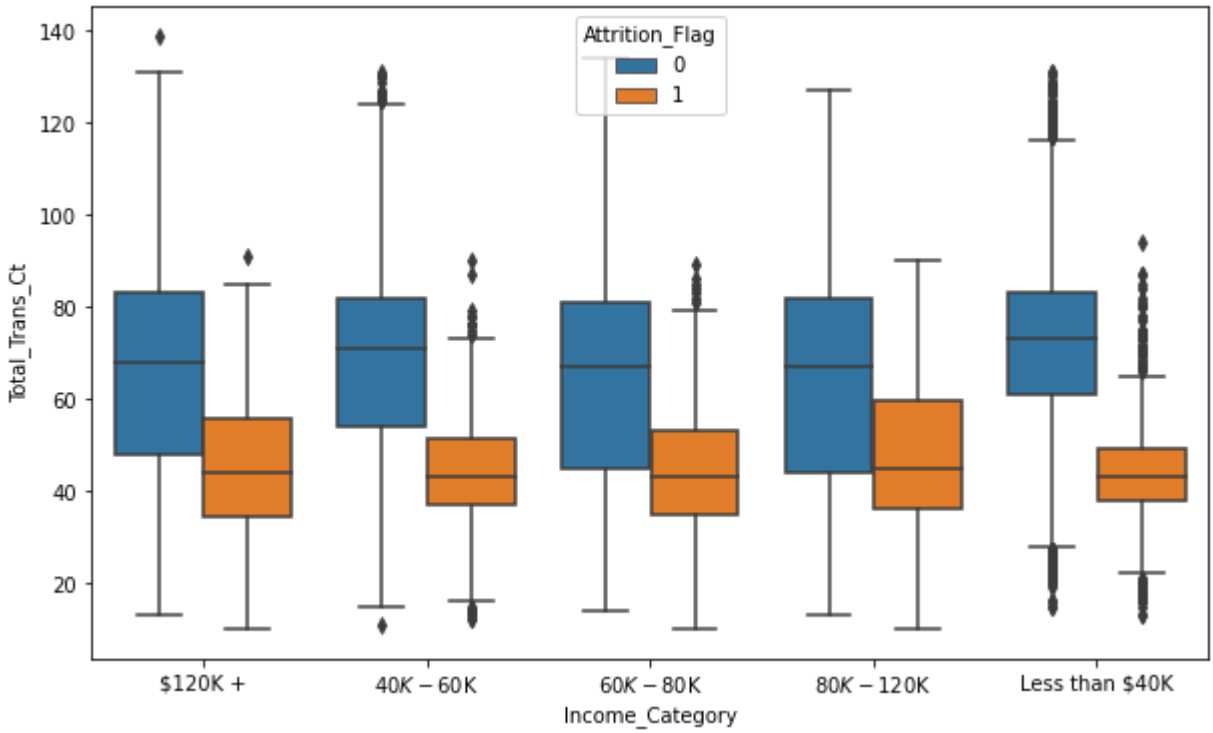
- Customers with Gold Credid Card, even when they carries over a high amount they can churn.

- Platinum Customers that have low Total Revolving Bal are more likehood to churn.

Total Trans CT VS Income Category per Attrition_Flag

In [73]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(df["Income_Category"], df["Total_Trans_Ct"], hue=df["Attrition_Fl
```

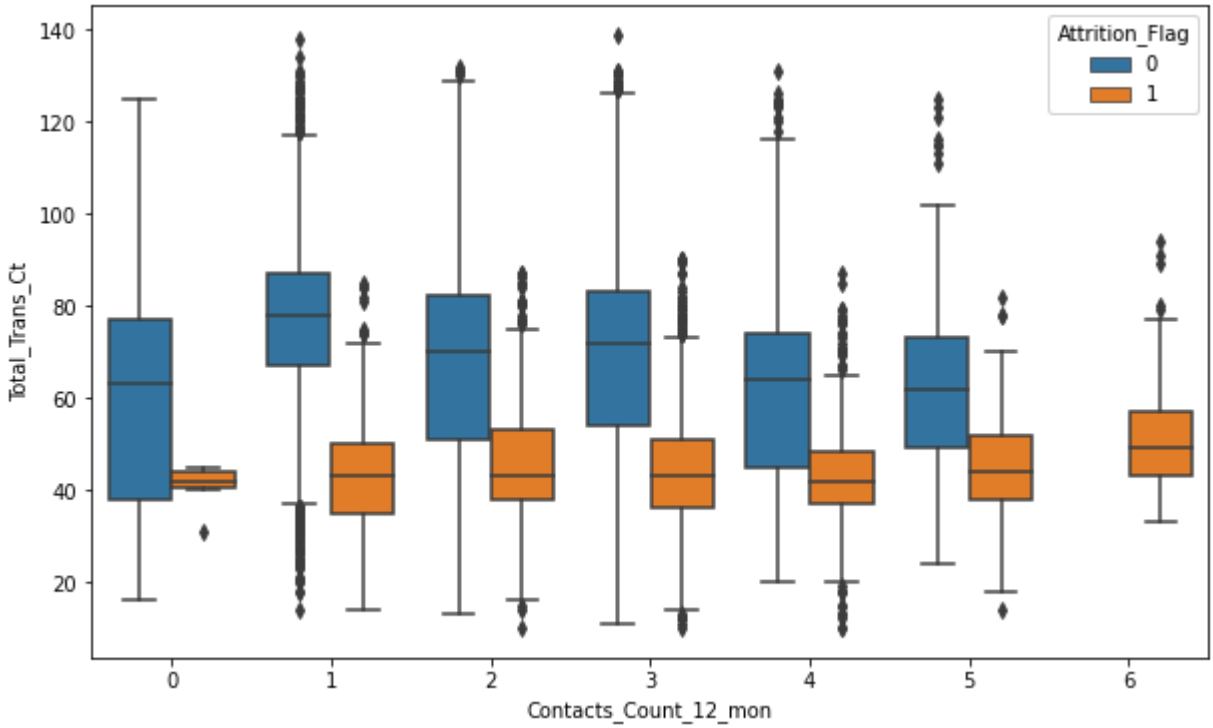
Out[73]:
```
<AxesSubplot:xlabel='Income_Category', ylabel='Total_Trans_Ct'>
```



- Customers with Total Trans CT less than 40 are more likelihood to churn independently of the Income category.
- Income Category does not show much pattern with Total Trans Ct to describle churn customer pattern.

In [74]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(df["Contacts_Count_12_mon"], df["Total_Trans_Ct"], hue=df["Attrit
```

Out[74]:
```
<AxesSubplot:xlabel='Contacts_Count_12_mon', ylabel='Total_Trans_Ct'>
```



- Customers with less than 50 transition are more likehood to churn.
- We can work with our call center to identify this customers and work on understanding the frustration so we can work on it and try to revert the situation.

Data Preparation for Modeling

In [75]:
```
data1 = df.copy()
```

In [76]:
```
# Dropping Avg Open to Buy  considering their high correlation with other var
data1.drop(
    columns=[
        "Avg_Open_To_Buy",
    ],
    inplace=True,
)
```

In [77]:
```
# Separating target variable and other variables
X = data1.drop(columns="Attrition_Flag")
Y = data1["Attrition_Flag"]
```

In [78]:
```
# Splitting data into training, validation and test set:
# first we split data into 2 parts, say temporary and test

X_temp, X_test, y_temp, y_test = train_test_split(
    X, Y, test_size=0.2, random_state=1, stratify=Y
)

# then we split the temporary set into train and validation

X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, random_state=1, stratify=y_temp
```

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

```
(6075, 18) (2026, 18) (2026, 18)
```

Missing-Value Treatment

- We will use most frequent to impute missing values in Education_Level, Marital_Status and Income_Category column.

In [79]:
```
data1.isnull().sum()
```

Out[79]:
```
Attrition_Flag                0
Customer_Age                  0
Gender                        0
Dependent_count               0
Education_Level            1519
Marital_Status              749
Income_Category            1112
Card_Category                 0
Months_on_book                0
Total_Relationship_Count      0
Months_Inactive_12_mon        0
Contacts_Count_12_mon         0
Credit_Limit                  0
Total_Revolving_Bal           0
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 [80]:
```
# Let's impute the missing values
imp_mode = SimpleImputer(missing_values=np.nan, strategy="most_frequent")

# fit the imputer on train data and transform the train data
X_train["Education_Level"] = imp_mode.fit_transform(X_train[["Education_Level

# transform the validation and test data using the imputer fit on train data
X_val["Education_Level"] = imp_mode.transform(X_val[["Education_Level"]])

# transform the validation and test data using the imputer fit on train data
X_test["Education_Level"] = imp_mode.transform(X_test[["Education_Level"]])
```

In [81]:
```
# Let's impute the missing values
imp_mode = SimpleImputer(missing_values=np.nan, strategy="most_frequent")

# fit the imputer on train data and transform the train data
X_train["Marital_Status"] = imp_mode.fit_transform(X_train[["Marital_Status"]

# transform the validation and test data using the imputer fit on train data
X_val["Marital_Status"] = imp_mode.transform(X_val[["Marital_Status"]])

# transform the validation and test data using the imputer fit on train data
X_test["Marital_Status"] = imp_mode.transform(X_test[["Marital_Status"]])
```

In [82]:
```
# Let's impute the missing values
imp_mode = SimpleImputer(missing_values=np.nan, strategy="most_frequent")

# fit the imputer on train data and transform the train data
X_train["Income_Category"] = imp_mode.fit_transform(X_train[["Income_Category

# transform the validation and test data using the imputer fit on train data
X_val["Income_Category"] = imp_mode.transform(X_val[["Income_Category"]])

# transform the validation and test data using the imputer fit on train data
X_test["Income_Category"] = imp_mode.transform(X_test[["Income_Category"]])
```

Creating dummy variables for categorical variables

In [83]:
```
# Creating dummy variables for categorical variables
X_train = pd.get_dummies(data=X_train, drop_first=True)
X_val = pd.get_dummies(data=X_val, drop_first=True)
X_test = pd.get_dummies(data=X_test, drop_first=True)
```

In [84]:
```
X_train.shape
```

Out[84]:
```
(6075, 46)
```

- After encoding there are 46 columns.

Building the model

Model evaluation criterion:

Model can make wrong predictions as:

1. Predicting that the customer is going to churn and the customer doesn't churn - Loss of resources
2. Predicting that the customer is not going to churn and the customer churn - Loss of opportunity

Which case is more important?

- Predicting that the customer is not going to churn and the customer churn i.e. losing on a potential source of income for the company because that customer will not be targeted by the marketing team when it should be targeted.

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

- Company wants Recall to be maximized (>95), greater the Recall lesser the chances of false negatives but also a Precission greater than 70.

1. Model Building: Default parameters

Let's evaluate the model performance by using KFold and cross_val_score

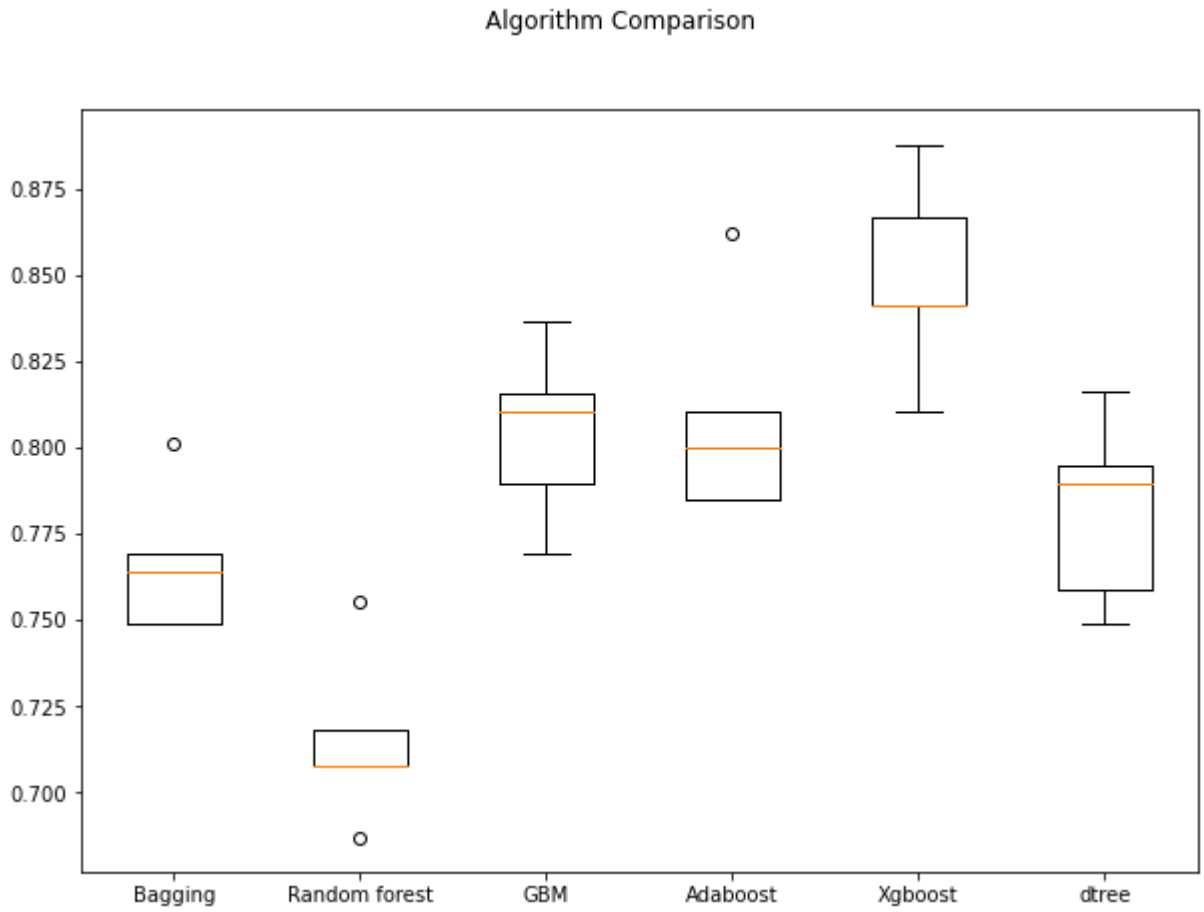
- `K-Folds cross-validation` provides dataset indices to split data into train/validation sets. Split dataset into k consecutive stratified folds (without shuffling by default). Each fold is then used once as validation while the k - 1 remaining folds form the training set.

In [85]:
```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1)))
models.append(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("Xgboost", XGBClassifier(random_state=1, eval_metric="logloss"
models.append(("dtree", DecisionTreeClassifier(random_state=1)))

results = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models


# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation Performance:" "\n")

for name, model in models:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Setting number of splits equal to 5
    cv_result = cross_val_score(
        estimator=model, X=X_train, y=y_train, scoring=scoring, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {:.3f}".format(name, cv_result.mean() * 100))

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)) * 100
    print("{}: {:.3f}".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)) * 100
    print("{}: {:.3f}".format(name, scores_val))
```

```
Cross-Validation Performance:

Bagging: 76.636
Random forest: 71.512
GBM: 80.427
Adaboost: 80.835
Xgboost: 84.935
dtree: 78.173

Training Performance:

Bagging: 98.463
Random forest: 100.000
GBM: 87.090
Adaboost: 83.607
```

```
Xgboost: 100.000
dtree: 100.000

Validation Performance:

Bagging: 80.675
Random forest: 73.620
GBM: 81.595
Adaboost: 81.288
Xgboost: 89.571
dtree: 78.221
```

In [86]:
```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results)
ax.set_xticklabels(names)

plt.show()
```



- We can see that XGBoost is giving the highest cross-validated recall followed by GBM
- The boxplot shows that the performance of XGBoost is consistent with 1 outlier and with 1 one outlier for GBM.
- The recall scores still not good so we will do Oversampled and Undersampled data to see if the performance improves.

2. Model building - Oversampled data

In [87]:
```
### Oversampling train data using SMOTE

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

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


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


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

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

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

After OverSampling, the shape of train_X: (10198, 46)
After OverSampling, the shape of train_y: (10198,)
```

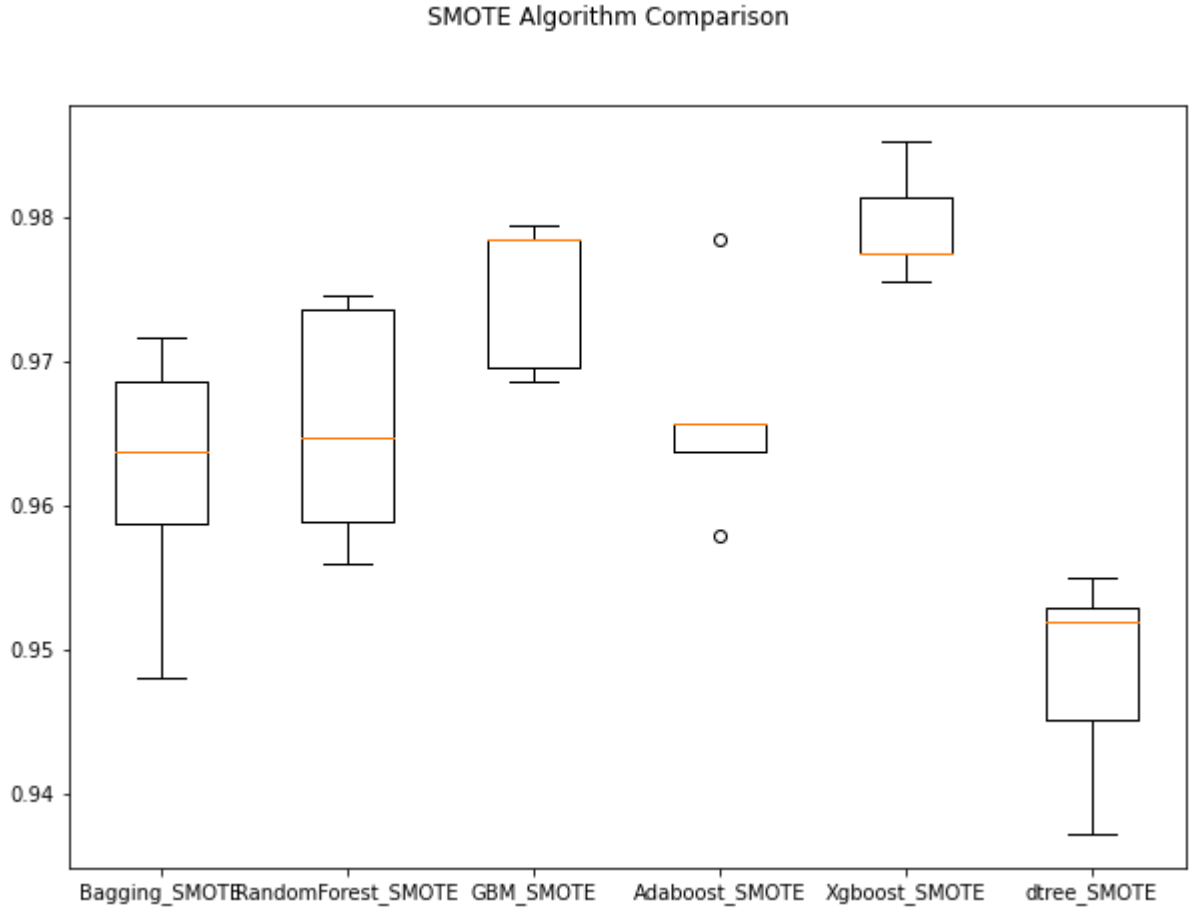
In [88]:
```
models_over = []  # Empty list to store all the models

# Appending models into the list
models_over.append(("Bagging_SMOTE", BaggingClassifier(random_state=1)))
models_over.append(("RandomForest_SMOTE", RandomForestClassifier(random_state
models_over.append(("GBM_SMOTE", GradientBoostingClassifier(random_state=1)))
models_over.append(("Adaboost_SMOTE", AdaBoostClassifier(random_state=1)))
models_over.append(
    ("Xgboost_SMOTE", XGBClassifier(random_state=1, eval_metric="logloss"))
)
models_over.append(("dtree_SMOTE", DecisionTreeClassifier(random_state=1)))

results_over = []  # Empty list to store all model's CV scores
names_over = []  # Empty list to store name of the models


# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation Performance:" "\n")

for name, model in models_over:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Setting number of splits equal to 5
    cv_result_over = cross_val_score(
        estimator=model, X=X_train_over, y=y_train_over, scoring=scoring, cv=
    )
    results_over.append(cv_result_over)
    names_over.append(name)
    print("{}: {:.3f}".format(name, cv_result_over.mean() * 100))

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

for name, model in models_over:
    model.fit(X_train_over, y_train_over)
    scores_over = recall_score(y_train_over, model.predict(X_train_over)) * 1
```

```
    print("{}: {:.3f}".format(name, scores_over))

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

for name, model in models_over:
    model.fit(X_train_over, y_train_over)
    scores_over_val = recall_score(y_val, model.predict(X_val)) * 100
    print("{}: {:.3f}".format(name, scores_over_val))
```

```
Cross-Validation Performance:

Bagging_SMOTE: 96.215
RandomForest_SMOTE: 96.549
GBM_SMOTE: 97.490
Adaboost_SMOTE: 96.627
Xgboost_SMOTE: 97.941
dtree_SMOTE: 94.842

Training Performance:

Bagging_SMOTE: 99.804
RandomForest_SMOTE: 100.000
GBM_SMOTE: 98.137
Adaboost_SMOTE: 97.058
Xgboost_SMOTE: 100.000
dtree_SMOTE: 100.000

Validation Performance:

Bagging_SMOTE: 84.969
RandomForest_SMOTE: 77.914
GBM_SMOTE: 88.650
Adaboost_SMOTE: 88.344
Xgboost_SMOTE: 89.877
dtree_SMOTE: 81.902
```

In [89]:

```
# Plotting boxplots for CV scores of all Oversampled models  defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("SMOTE Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results_over)
ax.set_xticklabels(names_over)

plt.show()
```



- Performance on the training and cross validation set improved.
- Model is capturing 100 on training and more than 90 in cross validation but ***on validation set is not performing well***.
- We can see that the model fits exactly against training data and cannot perform accurately against unseen data (validation set), our models are overfitting.
- XGBoost still giving the highest cross-validated recall (96.8) followed by GBM (95.6). Both consistent with no outliers.
- Lets try Undersampling the train to handle the imbalance between classes and check the model performance.

3. Model building - Undersampled data

In [90]:
```
### Undersampling train data

rus = RandomUnderSampler(random_state=1)
X_train_under, y_train_under = rus.fit_resample(X_train, y_train)

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

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

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

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

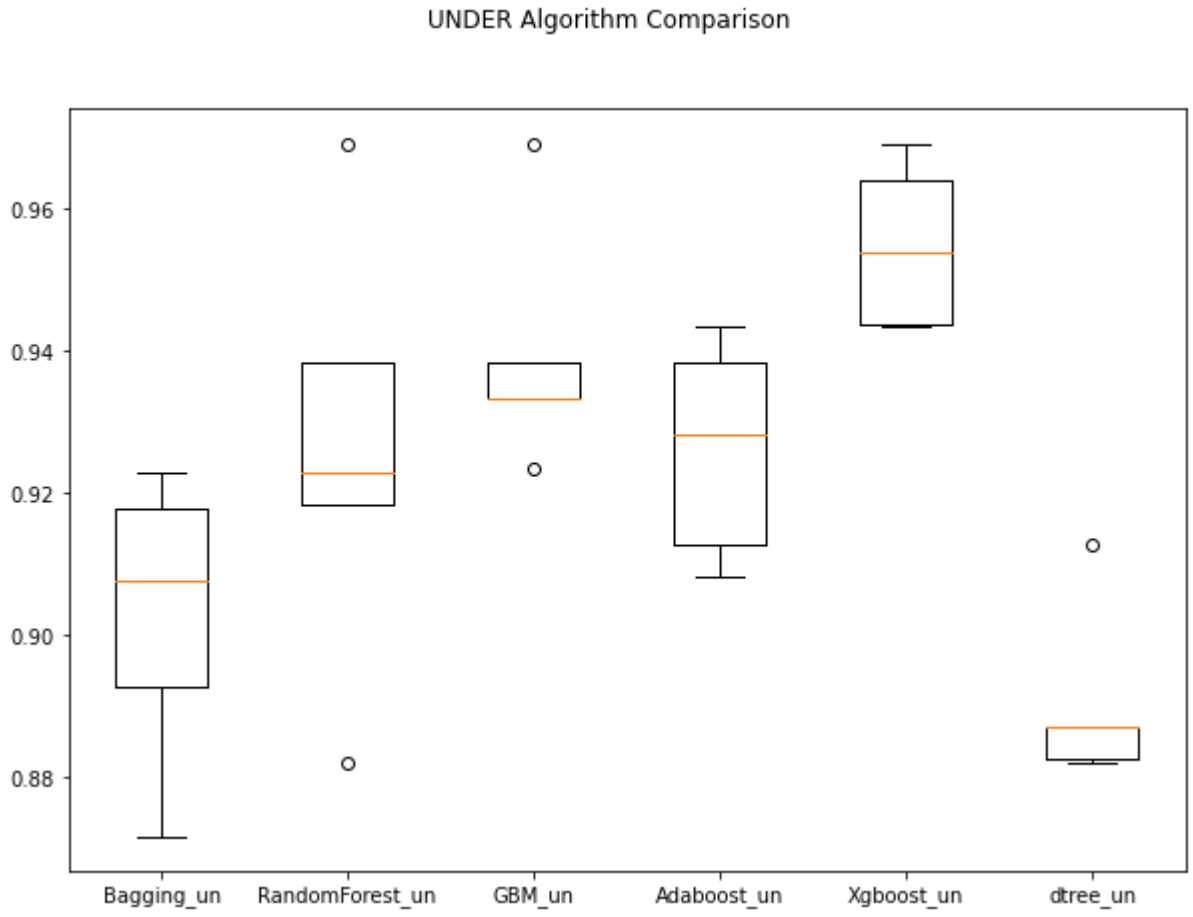
```
After Under Sampling, counts of label 'Yes': 976
After Under Sampling, counts of label 'No': 976

After Under Sampling, the shape of train_X: (1952, 46)
After Under Sampling, the shape of train_y: (1952,)
```

In [91]:
```
models_under = []  # Empty list to store all the models

# Appending models into the list
models_under.append(("Bagging_un", BaggingClassifier(random_state=1)))
models_under.append(("RandomForest_un", RandomForestClassifier(random_state=1
models_under.append(("GBM_un", GradientBoostingClassifier(random_state=1)))
models_under.append(("Adaboost_un", AdaBoostClassifier(random_state=1)))
models_under.append(
    ("Xgboost_un", XGBClassifier(random_state=1, eval_metric="logloss"))
)
models_under.append(("dtree_un", DecisionTreeClassifier(random_state=1)))

results_under = []  # Empty list to store all model's CV scores
names_under = []  # Empty list to store name of the models


# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation Performance:" "\n")

for name, model in models_under:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Setting number of splits equal to 5
    cv_result_under = cross_val_score(
        estimator=model, X=X_train_under, y=y_train_under, scoring=scoring, c
    )
    results_under.append(cv_result_under)
    names_under.append(name)
    print("{}: {:.3f}".format(name, cv_result_under.mean() * 100))

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

for name, model in models_under:
    model.fit(X_train_under, y_train_under)
    scores_under = recall_score(y_train_under, model.predict(X_train_under))
    print("{}: {:.3f}".format(name, scores_under))


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

for name, model in models_under:
    model.fit(X_train_under, y_train_under)
    scores_under_val = recall_score(y_val, model.predict(X_val)) * 100
    print("{}: {:.3f}".format(name, scores_under_val))
```

```
Cross-Validation Performance:

Bagging_un: 90.267
RandomForest_un: 92.624
GBM_un: 93.957
Adaboost_un: 92.625
Xgboost_un: 95.493
dtree_un: 89.038

Training Performance:
```

```
Bagging_un: 98.975
RandomForest_un: 100.000
GBM_un: 97.951
Adaboost_un: 94.877
Xgboost_un: 100.000
dtree_un: 100.000

Validation Performance:

Bagging_un: 91.718
RandomForest_un: 92.945
GBM_un: 94.172
Adaboost_un: 94.172
Xgboost_un: 96.626
dtree_un: 88.344
```

In [92]:
```
# Plotting boxplots for CV scores of all Oversampled models  defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("UNDER Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results_under)
ax.set_xticklabels(names_under)

plt.show()
```



- Performance on validation set improved.
- Model is capturing almost 100 on training and between 86 to 92 in cross validation and on validation set it is performing well.
- We can see that the model fits exactly against training data and cannot perform accurately against unseen data (validation set), our models are overfitting.

- XGBoost still giving the highest cross-validated recall (92.9) followed by GBM (92.4) and Bagging (90.1). All consistent and only XGB is showing 2 outliers.

> The best Cross Validation performance happened using ***oversampling***, on the other hand ***undersampling*** is showing more consistenc between Cross Validation and Validation Set. Considering that on ***oversampling*** all the validation scores were not on the range of cross validation and with low score, we gonna choose use the data set ***undersampling*** where validation score lay on the range of cross validation, hence is the best aproach for us.

4. Choose to tune 3 models

Our top 3 better expected performance on unseen data considering Cross Validation and Performance on Validation set happens with UNDERSAMPLING data set on the following models:

1. ***XGBoost***: Is performing better than all models, the boxplot shows a consistet performance with mean 95.5 and range between 94 to 96.5 and no outliers.
2. ***GBM***: Boxplot shows a consistent performance with 2 outliers, mean 94 and max around 96.5.
3. ***Adaboost***: This model is showing consistenc and no outlier, with range between 0.91 to 0.945 and mean 93

- As we saw before, the performance on validation set was overfitting, we gonna proced with Hyperparamenter tuning using random search to try to find the best set of hyperparameters and this tecnic also takes less time than GridSearch.

5. Hyperparameter tuning using random search

We will tune XGB, GBM and Adaboost models using Randomized Search CV.

In [93]:
```
# defining a function to compute different metrics to check performance of a
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model perfo

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

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

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

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1": f1,
```

```
        },
        index=[0],
    )

    return df_perf
```

In [94]:
```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

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

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

Adaboost

Adaboost Default

In [95]:
```
Adaboost_default = AdaBoostClassifier(random_state=1)
Adaboost_default.fit(X_train_under, y_train_under)
```

Out[95]:
```
AdaBoostClassifier(random_state=1)
```

In [96]:
```
# Calculating different metrics on train set
Adaboost_df_train = model_performance_classification_sklearn(
    Adaboost_default, X_train_under, y_train_under
)

# Calculating different metrics on validation set
Adaboost_df_val = model_performance_classification_sklearn(
    Adaboost_default, X_val, y_val
)


print("Training performance:")
print(Adaboost_df_train)
print("Validation performance:")
print(Adaboost_df_val)
```

```
Training performance:
   Accuracy  Recall  Precision    F1
0     0.943   0.949      0.938 0.943
Validation performance:
   Accuracy  Recall  Precision    F1
0     0.921   0.942      0.684 0.792
```

Adaboost Hyperparameter Tuning - Random Search

In [97]:
```
%%time

# Choose the type of classifier.
Adaboost_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    "n_estimators": np.arange(10, 110, 10),
    "learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}
# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
Adaboost_tuned_random = RandomizedSearchCV(estimator=Adaboost_tuned, param_di

#Fitting parameters in RandomizedSearchCV
Adaboost_tuned_random.fit(X_train_under,y_train_under)

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

```
Best parameters are {'n_estimators': 90, 'learning_rate': 0.1, 'base_estimato
r': DecisionTreeClassifier(max_depth=3, random_state=1)} with CV score=0.9498
11616954474:
Wall time: 12.2 s
```

In [98]:
```
# building model with best parameters
Adaboost_best = AdaBoostClassifier(
    random_state=1,
    n_estimators=90,
    learning_rate=0.1,
    base_estimator=DecisionTreeClassifier(max_depth=3, random_state=1),
)

# Fit the model on training data
Adaboost_best.fit(X_train_under, y_train_under)
```

Out[98]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3,
                                                         random_state=1),
                   learning_rate=0.1, n_estimators=90, random_state=1)
```

In [99]:
```
# Calculating different metrics on train set
Adaboost_train = model_performance_classification_sklearn(
    Adaboost_best, X_train_under, y_train_under
)
print("Training performance:")
Adaboost_train
```

```
Training performance:
```

Out[99]:

	Accuracy	Recall	Precision	F1
0	0.986	0.995	0.977	0.986

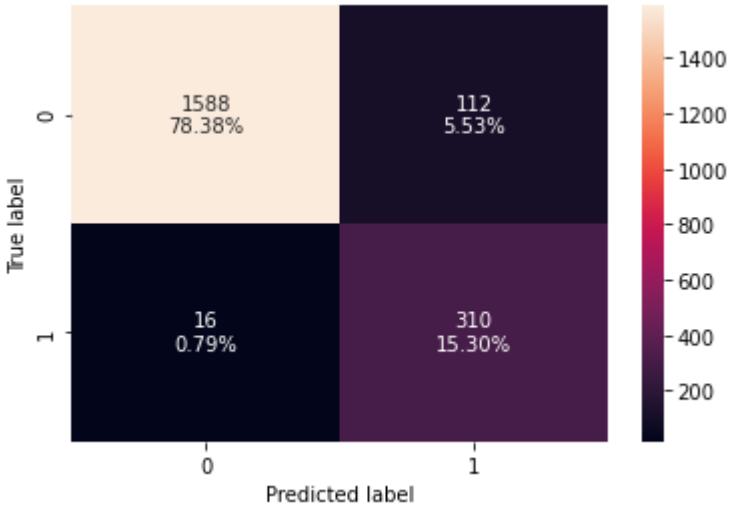
In [100…
```
# Calculating different metrics on validation set
Adaboost_val = model_performance_classification_sklearn(Adaboost_best, X_val,
print("Validation performance:")
Adaboost_val
```

```
Validation performance:
```

Out[100…

	Accuracy	Recall	Precision	F1
0	0.937	0.951	0.735	0.829

In [101…
```
# creating confusion matrix
confusion_matrix_sklearn(Adaboost_best, X_val, y_val)
```



- AdaBoost performance improved on Recall and Precission compared to the Default scores. | | Model | Recall | Precission| |---:|:-------------|:------|:------| | 1 | Default | 94.2 | 68.4 | | 2 | Hyperparameter | 95.1 | 73.5 |

Gradient Boosting

GB Default parameters

In [102…
```
GB_default = GradientBoostingClassifier(random_state=1)
GB_default.fit(X_train_under, y_train_under)
```

Out[102…
```
GradientBoostingClassifier(random_state=1)
```

In [103…
```
# Calculating different metrics on train set
GB_df_train = model_performance_classification_sklearn(
    GB_default, X_train_under, y_train_under
)

# Calculating different metrics on validation set
GB_df_val = model_performance_classification_sklearn(GB_default, X_val, y_val


print("Training performance:")
print(GB_df_train)
print("Validation performance:")
print(GB_df_val)
```

```
Training performance:
   Accuracy  Recall  Precision    F1
0     0.972   0.980      0.966 0.973
Validation performance:
   Accuracy  Recall  Precision    F1
0     0.933   0.942      0.724 0.819
```

Gradient Boosting Default Hyperparameter Tuning - Random Search

In [104…

```
%%time
# Choose the type of classifier.
GB_tuned = GradientBoostingClassifier(
    init=AdaBoostClassifier(random_state=1), random_state=1
)

# Grid of parameters to choose from
## add from article
parameters ={"n_estimators": np.arange(50,200,25),
             "subsample":[0.5,0.7,0.8,0.9,1],
             "max_features":[0.7,0.8,0.9,1],
             "learning_rate" : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30]

}

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

#Calling RandomizedSearchCV
GB_tunned_random = RandomizedSearchCV(estimator=GB_tuned, param_distributions

#Fitting parameters in RandomizedSearchCV
GB_tunned_random.fit(X_train_under,y_train_under)

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

```
Best parameters are {'subsample': 1, 'n_estimators': 175, 'max_features': 0.
8, 'learning_rate': 0.2} with CV score=0.9549188906331763:
Wall time: 17.2 s
```

In [105…

```
# building model with best parameters
GB_best = GradientBoostingClassifier(
    random_state=1,
    n_estimators=175,
    subsample=1,
    max_features=0.8,
    learning_rate=0.2,
)

# Fit the model on training data
GB_best.fit(X_train_under, y_train_under)
```

Out[105…

```
GradientBoostingClassifier(learning_rate=0.2, max_features=0.8,
                           n_estimators=175, random_state=1, subsample=1)
```

In [106…

```
# Calculating different metrics on train set
GB_train = model_performance_classification_sklearn(
    GB_best, X_train_under, y_train_under
)
print("Training performance:")
GB_train
```

```
Training performance:
```

Out[106…

	Accuracy	Recall	Precision	F1
0	0.999	1.000	0.998	0.999

In [107…

```
# Calculating different metrics on validation set
GB_val = model_performance_classification_sklearn(GB_best, X_val, y_val)
print("Validation performance:")
GB_val
```

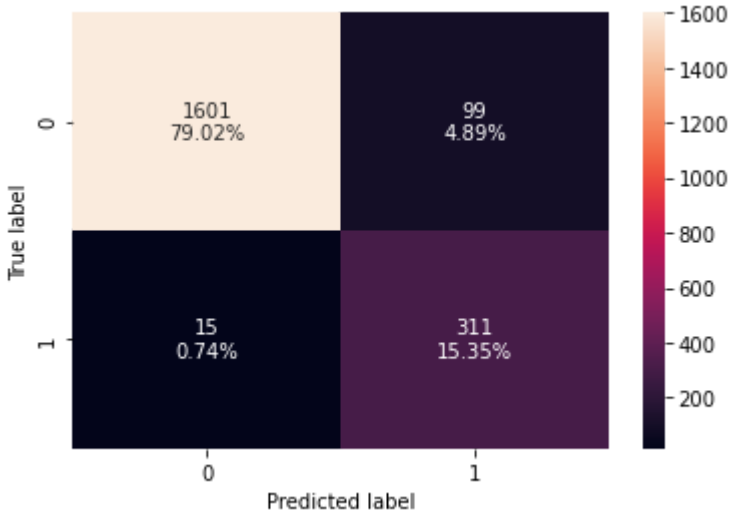
```
Validation performance:
```

Out[107…

	Accuracy	Recall	Precision	F1
0	0.944	0.954	0.759	0.845

In [108…

```
# creating confusion matrix
confusion_matrix_sklearn(GB_best, X_val, y_val)
```



- Model performance has improved using undersampling - GBoosting is now able to differentiate well between positive and negative classes.
- Gradient Boosting performance improved on Recall and Precission compared to the Default undersampling scores. | | Model | Recall | Precission| |---:|:-------------|:------|:------| | 1 | Default | 94.2 | 72.4 | | 2 | Hyperparameter | 95.4 | 75.9 |

XGBoost

XGB Default

In [109…

```
XGB_default = XGBClassifier(random_state=1, eval_metric="logloss")
XGB_default.fit(X_train_under, y_train_under)
```

Out[109…

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=12,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [110…
```
# Calculating different metrics on train set
XGB_df_train = model_performance_classification_sklearn(
    XGB_default, X_train_under, y_train_under
)

# Calculating different metrics on validation set
XGB_df_val = model_performance_classification_sklearn(XGB_default, X_val, y_v


print("Training performance:")
print(XGB_df_train)
print("Validation performance:")
print(XGB_df_val)
```

```
Training performance:
   Accuracy  Recall  Precision    F1
0     1.000   1.000      1.000 1.000
Validation performance:
   Accuracy  Recall  Precision    F1
0     0.941   0.966      0.745 0.841
```

XGB Hyperparameter Tuning - Random Search

In [111…
```
%%time

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

# Parameter grid to pass in RandomizedSearchCV
parameters ={"n_estimators": np.arange(50,200,25),
             "subsample":[0.5,0.7,0.8,0.9,1],
             "learning_rate" : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
}


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

#Calling RandomizedSearchCV
xgb_tuned_random = RandomizedSearchCV(estimator=XGB_tunned, param_distributio

#Fitting parameters in RandomizedSearchCV
xgb_tuned_random.fit(X_train_under,y_train_under)

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

```
Best parameters are {'subsample': 0.8, 'n_estimators': 175, 'learning_rate':
0.2} with CV score=0.9559445316588174:
Wall time: 21 s
```

In [112…
```
# building model with best parameters
XGB_best = XGBClassifier(
    random_state=1,
    eval_metric="logloss",
    learning_rate=0.2,
    subsample=0.8,
    n_estimators=175,
)
# Fit the model on training data
XGB_best.fit(X_train_over, y_train_over)
```

Out[112…
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
```

```
                gamma=0, gpu_id=-1, importance_type='gain',
                interaction_constraints='', learning_rate=0.2, max_delta_step=
0,
                max_depth=6, min_child_weight=1, missing=nan,
                monotone_constraints='()', n_estimators=175, n_jobs=12,
                num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
                scale_pos_weight=1, subsample=0.8, tree_method='exact',
                validate_parameters=1, verbosity=None)
```

In [113…
```
# Calculating different metrics on train set
XGB_train = model_performance_classification_sklearn(
    XGB_best, X_train_under, y_train_under
)
print("Training performance:")
XGB_train
```

```
Training performance:
```

Out[113…

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

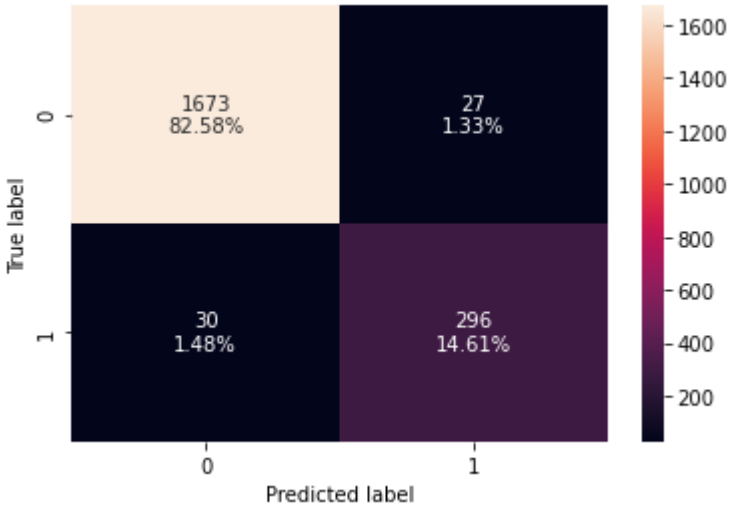
In [114…
```
# Calculating different metrics on validation set
XGB_val = model_performance_classification_sklearn(XGB_best, X_val, y_val)
print("Validation performance:")
XGB_val
```

```
Validation performance:
```

Out[114…

	Accuracy	Recall	Precision	F1
0	0.972	0.908	0.916	0.912

In [115…
```
# creating confusion matrix
confusion_matrix_sklearn(XGB_best, X_val, y_val)
```



- XGBoost performance decrised on Recall and increased on Precission compared to the Default scores. | | Model | Recall | Precission| |---:|:-------------|:------|:------| | 1 | Default | 96.6 | 74.5 | | 2 | Hyperparameter | 90.8 | 91.6 |

6. Model Performances

In [116…
```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        Adaboost_train.T,
        Adaboost_val.T,
        GB_train.T,
        GB_val.T,
        XGB_train.T,
        XGB_val.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Adaboost Train",
    "Adaboost Validation",
    "GB Train",
    "GB Validation",
    "XGB Train",
    "XGB Validation",
]
print("Training X Validation performance on Random Search:")
models_train_comp_df
```

```
Training X Validation performance on Random Search:
```

Out[116…

	Adaboost Train	Adaboost Validation	GB Train	GB Validation	XGB Train	XGB Validation
Accuracy	0.986	0.937	0.999	0.944	1.000	0.972
Recall	0.995	0.951	1.000	0.954	1.000	0.908
Precision	0.977	0.735	0.998	0.759	1.000	0.916
F1	0.986	0.829	0.999	0.845	1.000	0.912

- The Gradient Boosting model tuned using Random Search is giving the best validation recall of 0.954 and it has a precision value greater than 70 on under sampling data.
- XGB is givind a good Recall and Precision around 90, but our metric of interest is Recall.
- Let's check the model's performance on test set and then see the feature importance from the tuned xgboost model

Performance on the test set

In [117…
```
# Calculating different metrics on validation set
GB_val = model_performance_classification_sklearn(GB_best, X_test, y_test)
print("Test dataset performance:")
GB_val
```

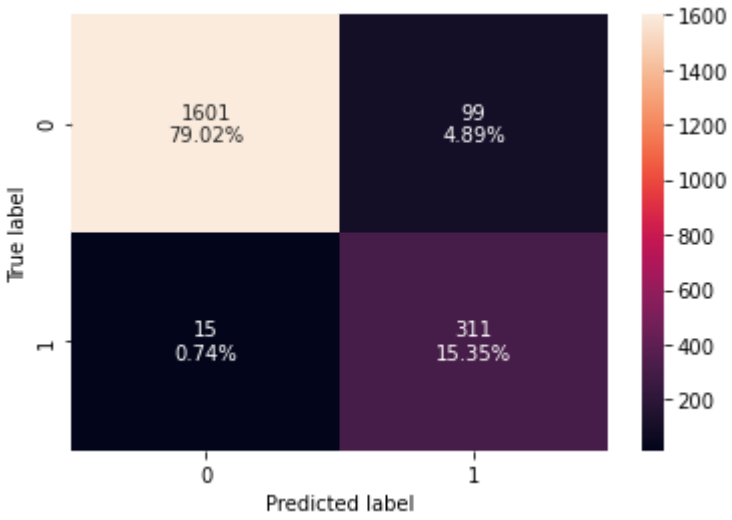
```
Test dataset performance:
```

Out[117…

	Accuracy	Recall	Precision	F1
0	0.941	0.960	0.746	0.840

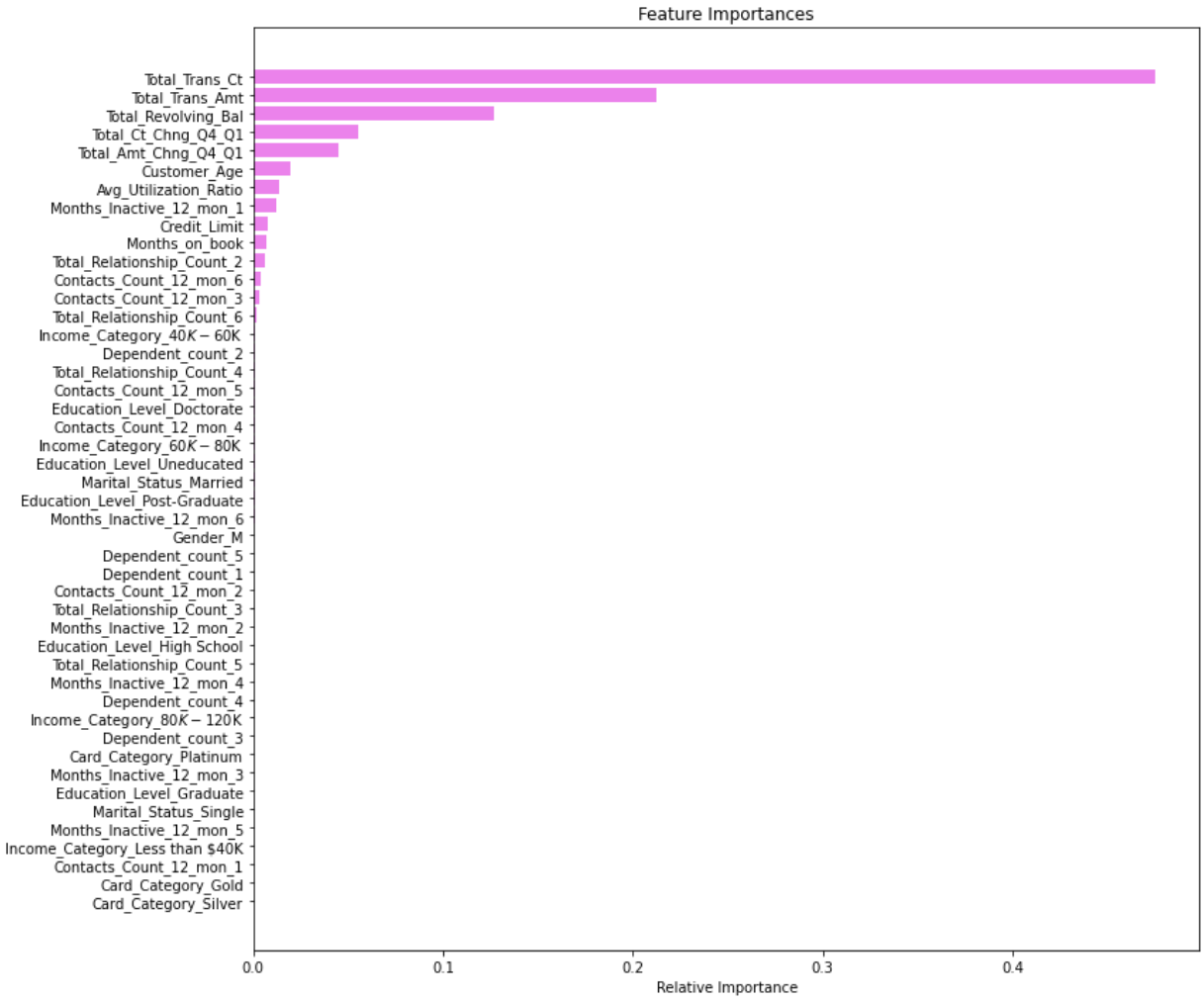
- Test data is performing well, with a really good Recall score of 0.96 and a good Precission 0.746.

In [118…
```
# creating confusion matrix
confusion_matrix_sklearn(GB_best, X_val, y_val)
```



- Let's see the feature importance from the Gradient Boosting model tuned with Random Search.

In [119…
```
feature_names = X_train.columns
importances = GB_best.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="ce
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Total Trans Ct is the most important feature, followed by Total Trans Amt, Total Revolving Bal, Total CT Chng Q4 Q1 and Total Amt Chng Q4 Q1, as per the tuned undersampling data set on Gradient Boosting model.

Pipelines for productionizing the model

- Now, we have a final model. let's use pipelines to put the model into production

- We will create 2 different pipelines, one for numerical columns and one for categorical columns
- For numerical columns, we will do missing value imputation as pre-processing
- For categorical columns, we will do one hot encoding and missing value imputation as pre-processing

- We are doing missing value imputation for the whole data, so that if there is any missing value in the data in future that can be taken care of.

In [120…
```
# List of Categorical Variables
cat_col
```

Out[120…
```
['Gender',
 'Education_Level',
 'Marital_Status',
```

```
'Income_Category',
'Card_Category',
'Dependent_count',
'Total_Relationship_Count',
'Months_Inactive_12_mon',
'Contacts_Count_12_mon']
```

In [121…
```
# Creating List of Numerical Variables
num_col = []
for col in data1.columns:
    if col not in cat_col:
        num_col.append(col)

num_col.remove("Attrition_Flag")
num_col
```

Out[121…
```
['Customer_Age',
 'Months_on_book',
 'Credit_Limit',
 'Total_Revolving_Bal',
 'Total_Amt_Chng_Q4_Q1',
 'Total_Trans_Amt',
 'Total_Trans_Ct',
 'Total_Ct_Chng_Q4_Q1',
 'Avg_Utilization_Ratio']
```

In [122…
```
# ist of numerical variables
numerical_features = num_col

# creating a transformer for numerical variables, which will apply simple imp
numeric_transformer = Pipeline(steps=[("imputer", SimpleImputer(strategy="med

# List of categorical variables
categorical_features = cat_col

# creating a transformer for categorical variables, which will first apply si
# then do one hot encoding for categorical variables
categorical_transformer = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("onehot", OneHotEncoder(handle_unknown="ignore")),
    ]
)

# handle_unknown = "ignore", allows model to handle any unknown category in t

# combining categorical transformer and numerical transformer using a column

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numerical_features),
        ("cat", categorical_transformer, categorical_features),
    ],
    remainder="passthrough",
)
# remainder = "passthrough" has been used, it will allow variables that are p
# but not in "numerical_columns" and "categorical_columns" to pass through th
```

In [123…
```
# Separating target variable and other variables
X = data1.drop("Attrition_Flag", axis=1)
Y = data1["Attrition_Flag"]
```

In [124…
```
# Now we already know the best model we need to process with, so we don't nee
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.30, random_state=1, stratify=Y
)
print(X_train.shape, X_test.shape)
```

```
(7088, 18) (3039, 18)
```

In [125…
```
### Undersampling train data
rus = RandomUnderSampler(random_state=1)
X_train_under, y_train_under = rus.fit_resample(X_train, y_train)
```

In [126…
```
# Creating new pipeline with best parameters
model = Pipeline(
    steps=[
        ("pre", preprocessor),
        (
            "GBoost",
            GradientBoostingClassifier(
                random_state=1,
                n_estimators=175,
                subsample=1,
                max_features=0.8,
                learning_rate=0.2,
            ),
        ),
    ]
)
# Fit the model on under sampling training data
model.fit(X_train_under, y_train_under)
```

Out[126…
```
Pipeline(steps=[('pre',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('num',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImpu
ter(strategy='median'))]),
                                                  ['Customer_Age',
                                                   'Months_on_book',
                                                   'Credit_Limit',
                                                   'Total_Revolving_Bal',
                                                   'Total_Amt_Chng_Q4_Q1',
                                                   'Total_Trans_Amt',
                                                   'Total_Trans_Ct',
                                                   'Total_Ct_Chng_Q4_Q1',
                                                   'Avg_Utilization_Ratio']),
                                                 ('cat',
                                                  Pipeline(steps=[('...
                                                                   SimpleImpu
ter(strategy='most_frequent')),
                                                                   ('onehot',
                                                                   OneHotEnco
der(handle_unknown='ignore'))]),
                                                  ['Gender', 'Education_Leve
l',
                                                   'Marital_Status',
                                                   'Income_Category',
                                                   'Card_Category',
                                                   'Dependent_count',
                                                   'Total_Relationship_Coun
t',
                                                   'Months_Inactive_12_mon',
                                                   'Contacts_Count_12_mo
```

```
n'])])),
                ('GBoost',
                 GradientBoostingClassifier(learning_rate=0.2, max_features=
0.8,
                                            n_estimators=175, random_state=1,
                                            subsample=1))])
```

In [127…

```
# Calculating different metrics on Test set
GB_val_pipe = model_performance_classification_sklearn(model, X_test, y_test)
print("Pipeline - Test dataset performance:")
GB_val_pipe
```

```
Pipeline - Test dataset performance:
```

Out[127…

	Accuracy	Recall	Precision	F1
0	0.950	0.961	0.779	0.861

- Our Pipeline model is performing really well on our data, with 0.961 Recall and 0.779 Precision.
- We did Simple Imputer for missing values considering median for numericals and most frequent for categorical.
- We used undersampling data set and applyed the Hyperparameter tuning with Random Search on Gradient Boosting model.

Note:

- I used KNNImputer for missing values on my project, because of Pipeline I change it to Simple Imputer.
- I didn't know how to do map and inversing map inside de pipeline.
- The scores didn't change much.
- For undersamplig, I applyed after split and the new X_train_under and y_train_under I fit on model inside pipeline. Once againg, I didn't Know how to do it inside pipeline.
- For both doubts I oppend a support request and until friday night I didn't get a return.
- I saw that create a function inside pipleine is not that easy and we didn't learn it yet.

Insights & Recommendations

- The best model is Gradient Boosting on undersampling set, that score on Recall is greater than 95 and Precision greater than 70. This means that the model is good at identifying churn customers.
- The model performance still really good on test set, Recall 96 and Precision 75 and appling Pipeline it still giving a great performance 96 and 78 Recall and Precision respectively, overall we can see that our model is showing consistence.
- Total Trans Ct, Total Trans Amt, Total Revolving Bal, Total CT Chng Q4 Q1 and Total Amt Chng Q4 Q1, are the important variables in determining if the customer will churn or not, less they use the credit card, greater is the cance of canceling it.
- We saw in our analysis that customers with high usage of the Credit Card are more likely to not churn.

- How can we increase the usage of the Credit Card? Promotions, Points, CashBack, Free Fee.
- Understand what customers (Soliciting Customers feedback via surveys) wants and what others Credit Card are ofering can help us develop the best marketing campaing.
- Understand how satisfied are the customers, using our call center to tabulate the data capturing customers concerns, monitor complains and develop dashboards to detect addressable patterns can help us to keep our customers and increase our customer base.