Credit Card Users Churn Prediction

Problem Statement

Business Context

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

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

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

Data Description

- CLIENTNUM: Client number. Unique identifier for the customer holding the account
- Attrition_Flag: Internal event (customer activity) variable - if the account is closed then "Attrited Customer" else "Existing Customer"
- Customer_Age: Age in Years
- Gender: Gender of the account holder
- Dependent_count: Number of dependents
- Education_Level: Educational Qualification of the account holder - Graduate, High School, Unknown, Uneducated, College(refers to college student), Post-Graduate, Doctorate
- Marital_Status: Marital Status of the account holder
- Income_Category: Annual Income Category of the account holder
- Card_Category: Type of Card
- Months_on_book: Period of relationship with the bank (in months)
- Total_Relationship_Count: Total no. of products held by the customer
- Months_Inactive_12_mon: No. of months inactive in the last 12 months
- Contacts_Count_12_mon: No. of Contacts in the last 12 months
- Credit_Limit: Credit Limit on the Credit Card
- Total_Revolving_Bal: Total Revolving Balance on the Credit Card
- Avg_Open_To_Buy: Open to Buy Credit Line (Average of last 12 months)
- Total_Amt_Chng_Q4_Q1: Change in Transaction Amount (Q4 over Q1)
- Total_Trans_Amt: Total Transaction Amount (Last 12 months)
- Total_Trans_Ct: Total Transaction Count (Last 12 months)
- Total_Ct_Chng_Q4_Q1: Change in Transaction Count (Q4 over Q1)
- Avg_Utilization_Ratio: Average Card Utilization Ratio

What Is a Revolving Balance?

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

What is the Average Open to buy?

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

What is the Average utilization Ratio?

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

Relation b/w Avg_Open_To_Buy, Credit_Limit and Avg_Utilization_Ratio:

- (Avg_Open_To_Buy / Credit_Limit) + Avg_Utilization_Ratio = 1

Please read the instructions carefully before starting the project.

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

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

Importing necessary libraries

In [104…

```
# Install Imblearn libraries. We need this for over and under sampling
!pip install Imblearn
!pip install imbalanced-learn
```

```
Requirement already satisfied: Imblearn in /usr/local/lib/python3.10/dist-packages (0.0)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packag
es (from Imblearn) (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages
(from imbalanced-learn->Imblearn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages
(from imbalanced-learn->Imblearn) (1.10.1)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-pac
kages (from imbalanced-learn->Imblearn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages
(from imbalanced-learn->Imblearn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pa
ckages (from imbalanced-learn->Imblearn) (3.2.0)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packag
es (0.10.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages
(from imbalanced-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages
(from imbalanced-learn) (1.10.1)
```

```
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.2.0)
```

In [4]:

```
!pip install Pyppeteer
```

```
Collecting Pyppeteer
  Downloading pyppeteer-1.0.2-py3-none-any.whl (83 kB)
     ---------------------------------------- 83.4/83.4 kB 1.6 MB/s eta 0:00:00
Collecting websockets<11.0,>=10.0
  Downloading websockets-10.4-cp310-cp310-win_amd64.whl (101 kB)
     -------------------------------------- 101.4/101.4 kB 1.5 MB/s eta 0:00:00
Requirement already satisfied: importlib-metadata>=1.4 in c:\app\anaconda3\lib\site-packages (from Pyppeteer) (4.11.3)
Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in c:\app\anaconda3\lib\site-packages (from Pyppeteer) (1.26.14)
Collecting pyee<9.0.0,>=8.1.0
  Downloading pyee-8.2.2-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in c:\app\anaconda3\lib\site-packages (from Pyppeteer) (1.4.4)
Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in c:\app\anaconda3\lib\site-packages (from Pyppeteer) (4.64.1)
Requirement already satisfied: certifi>=2021 in c:\app\anaconda3\lib\site-packages (from Pyppeteer) (2023.7.22)
Requirement already satisfied: zipp>=0.5 in c:\app\anaconda3\lib\site-packages (from importlib-metadata>=1.4->Pyppeteer) (3.11.0)
Requirement already satisfied: colorama in c:\app\anaconda3\lib\site-packages (from tqdm<5.0.0,>=4.42.1->Pyppeteer) (0.4.6)
Installing collected packages: pyee, websockets, Pyppeteer
Successfully installed Pyppeteer-1.0.2 pyee-8.2.2 websockets-10.4
```

In [105…

```
# to help with reading and manipulation of data
import numpy as np
import pandas as pd

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

# To split the data
from sklearn.model_selection import train_test_split


# To build the decision tree modle
from sklearn.tree import DecisionTreeClassifier


# to impute missing values
from sklearn.impute import SimpleImputer

# To build a Random forest classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier

# To do one-hot endcoding
from sklearn.preprocessing import OneHotEncoder

# To build the decision tree modle
from sklearn.tree import DecisionTreeClassifier

#To install xgboost library use - !pip install xgboost
```

```
from xgboost import XGBClassifier

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

# To tune a model
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

# To get different performance metrics
import sklearn.metrics as metrics
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    recall_score,
    accuracy_score,
    precision_score,
    f1_score
)

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

Loading the dataset

In [1]:

```
# Mount the google drive
#from google.colab import drive
#drive.mount("/content/drive")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[1], line 2
      1 # Mount the google drive
----> 2 from google.colab import drive
      3 drive.mount("/content/drive")

ModuleNotFoundError: No module named 'google'
```

In [107…

```
#df = pd.read_csv("/content/drive/MyDrive/Machine Learning/BankChurners.csv")
#df = pd.read_csv("/content/drive/MyDrive/Machine Learning/BankChurners.csv")
df = pd.read_csv("/content/drive/MyDrive/Machine Learning/BankChurners.csv")
data = df.copy()
```

Data Overview

- Observations
- Sanity checks

In [108…

```
data.head()
```

Out[108]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_
0	768805383	Existing Customer	45	M	3	High School	Married	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less

2	713982108	Existing Customer	51	M	3	Graduate	Married	8
3	769911858	Existing Customer	40	F	4	High School	NaN	Less
4	709106358	Existing Customer	40	M	3	Uneducated	Married	

5 rows × 21 columns

In [109…
```
data.info()
```

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

In [110…
```
#Let us drop the CLIENTNUM as it is not useful
data = data.drop(["CLIENTNUM"],axis=1)
```

In [111…
```
#Check if there are any duplicates
data.duplicated().sum()
```

Out[111]:
```
0
```

In [112…
```
# Let us check the number of rows and columns
df.shape
```

Out[112]:
```
(10127, 21)
```

In [113…
```
# Check for nulls in the columsn values
data.isna().sum()
```

Out[113]:
```
Attrition_Flag              0
Customer_Age                0
Gender                      0
Dependent_count             0
Education_Level          1519
```

```
Marital_Status              749
Income_Category               0
Card_Category                 0
Months_on_book                0
Total_Relationship_Count      0
Months_Inactive_12_mon        0
Contacts_Count_12_mon         0
Credit_Limit                  0
Total_Revolving_Bal           0
Avg_Open_To_Buy               0
Total_Amt_Chng_Q4_Q1          0
Total_Trans_Amt               0
Total_Trans_Ct                0
Total_Ct_Chng_Q4_Q1           0
Avg_Utilization_Ratio         0
dtype: int64
```

In [115…

```
# Looking at value counts for non-numeric features

num_to_display = 10

for colname in data.dtypes[df.dtypes == "object"].index:
  val_counts = data[colname].value_counts(dropna=False)
  print(val_counts[:num_to_display])

  if  len(val_counts) > num_to_display:
    print(f"Only displaying first {num_to_display} of {len(val_counts)} values.")
  print("-" * 50, "\n") # just for more in between
```

```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
--------------------------------------------------

F    5358
M    4769
Name: Gender, dtype: int64
--------------------------------------------------

Graduate         3128
High School      2013
NaN              1519
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64
--------------------------------------------------

Married     4687
Single      3943
NaN          749
Divorced     748
Name: Marital_Status, dtype: int64
--------------------------------------------------

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

Blue        9436
```

```
Silver        555
Gold          116
Platinum       20
Name: Card_Category, dtype: int64
--------------------------------------------------
```

In [116…

```
# Let us encode the Attrition Flag with numbers
data["Attrition_Flag"].replace("Existing Customer",0,inplace=True)
data["Attrition_Flag"].replace("Attrited Customer",1,inplace=True)
```

In [117…

```
#Convert the objects to category variables
data['Gender'] = data['Gender'].astype("category")
data['Education_Level'] = data["Education_Level"].astype("category")
data['Marital_Status'] = data['Marital_Status'].astype("category")
data['Income_Category'] = data['Income_Category'].astype("category")
data['Card_Category'] = data["Card_Category"].astype("category")
```

In [118…

```
# Let us check the standard measures fo the different columns
data.describe().T
```

Out[118]:

	count	mean	std	min	25%	50%	75%	max
Attrition_Flag	10127.0	0.160660	0.367235	0.0	0.000	0.000	0.000	1.000
Customer_Age	10127.0	46.325960	8.016814	26.0	41.000	46.000	52.000	73.000
Dependent_count	10127.0	2.346203	1.298908	0.0	1.000	2.000	3.000	5.000
Months_on_book	10127.0	35.928409	7.986416	13.0	31.000	36.000	40.000	56.000
Total_Relationship_Count	10127.0	3.812580	1.554408	1.0	3.000	4.000	5.000	6.000
Months_Inactive_12_mon	10127.0	2.341167	1.010622	0.0	2.000	2.000	3.000	6.000
Contacts_Count_12_mon	10127.0	2.455317	1.106225	0.0	2.000	2.000	3.000	6.000
Credit_Limit	10127.0	8631.953698	9088.776650	1438.3	2555.000	4549.000	11067.500	34516.000
Total_Revolving_Bal	10127.0	1162.814061	814.987335	0.0	359.000	1276.000	1784.000	2517.000
Avg_Open_To_Buy	10127.0	7469.139637	9090.685324	3.0	1324.500	3474.000	9859.000	34516.000
Total_Amt_Chng_Q4_Q1	10127.0	0.759941	0.219207	0.0	0.631	0.736	0.859	3.397
Total_Trans_Amt	10127.0	4404.086304	3397.129254	510.0	2155.500	3899.000	4741.000	18484.000
Total_Trans_Ct	10127.0	64.858695	23.472570	10.0	45.000	67.000	81.000	139.000
Total_Ct_Chng_Q4_Q1	10127.0	0.712222	0.238086	0.0	0.582	0.702	0.818	3.714
Avg_Utilization_Ratio	10127.0	0.274894	0.275691	0.0	0.023	0.176	0.503	0.999

Exploratory Data Analysis (EDA)

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

Questions:

1. How is the total transaction amount distributed?
2. What is the distribution of the level of education of customers?
3. What is the distribution of the level of income of customers?
4. How does the change in transaction amount between Q4 and Q1 (`total_ct_change_Q4_Q1`) vary by the customer's account status (`Attrition_Flag`)?
5. How does the number of months a customer was inactive in the last 12 months (`Months_Inactive_12_mon`) vary by the customer's account status (`Attrition_Flag`)?
6. What are the attributes that have a strong correlation with each other?

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

In [119…

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


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

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

In [120…

```
# function to create labeled barplots



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

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

    total = len(data[feature])  # length of the column
```

```
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

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

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

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

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

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

In [121…

```
# function to plot stacked bar chart

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

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

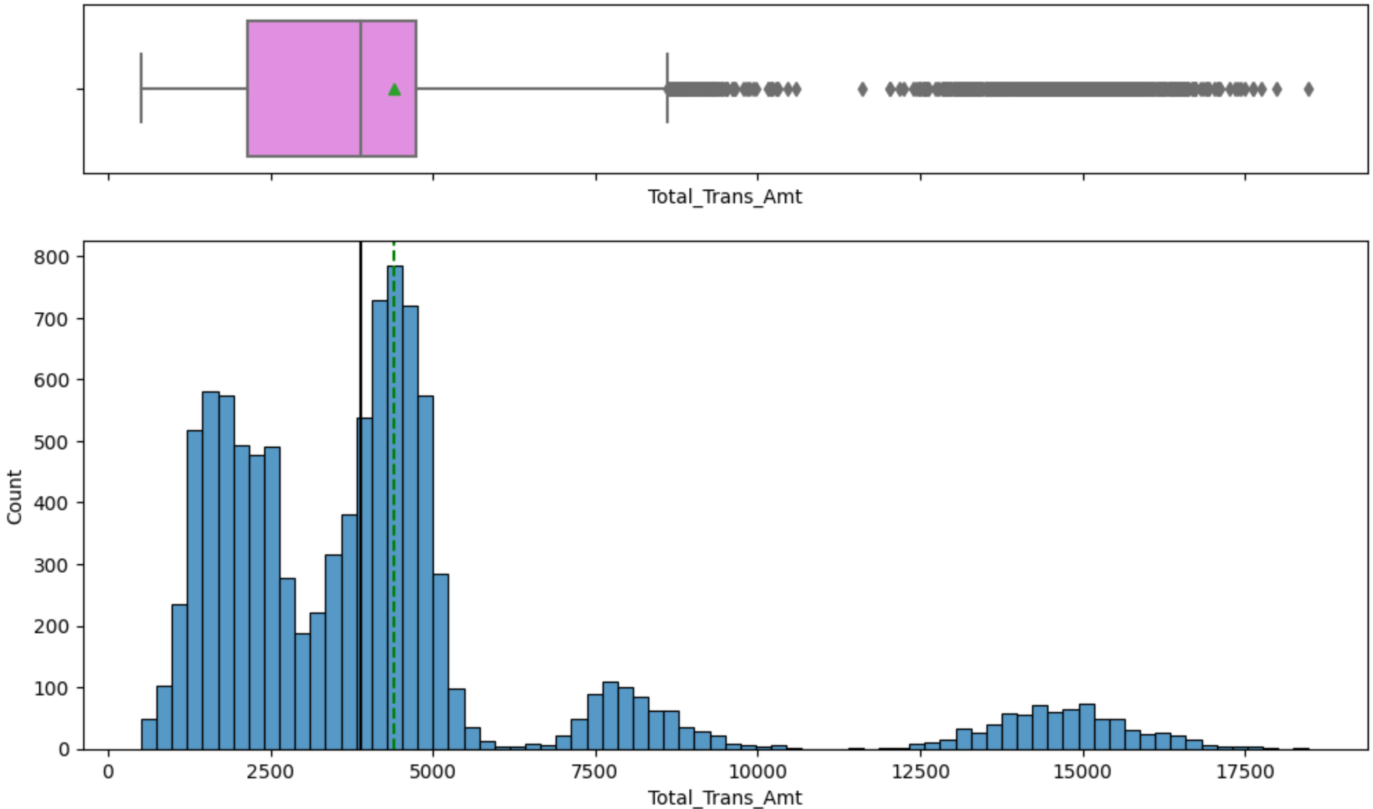
In [122…

```
### Function to plot distributions
```

```python
def distribution_plot_wrt_target(data, predictor, target):

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

    target_uniq = data[target].unique()

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

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

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

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

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

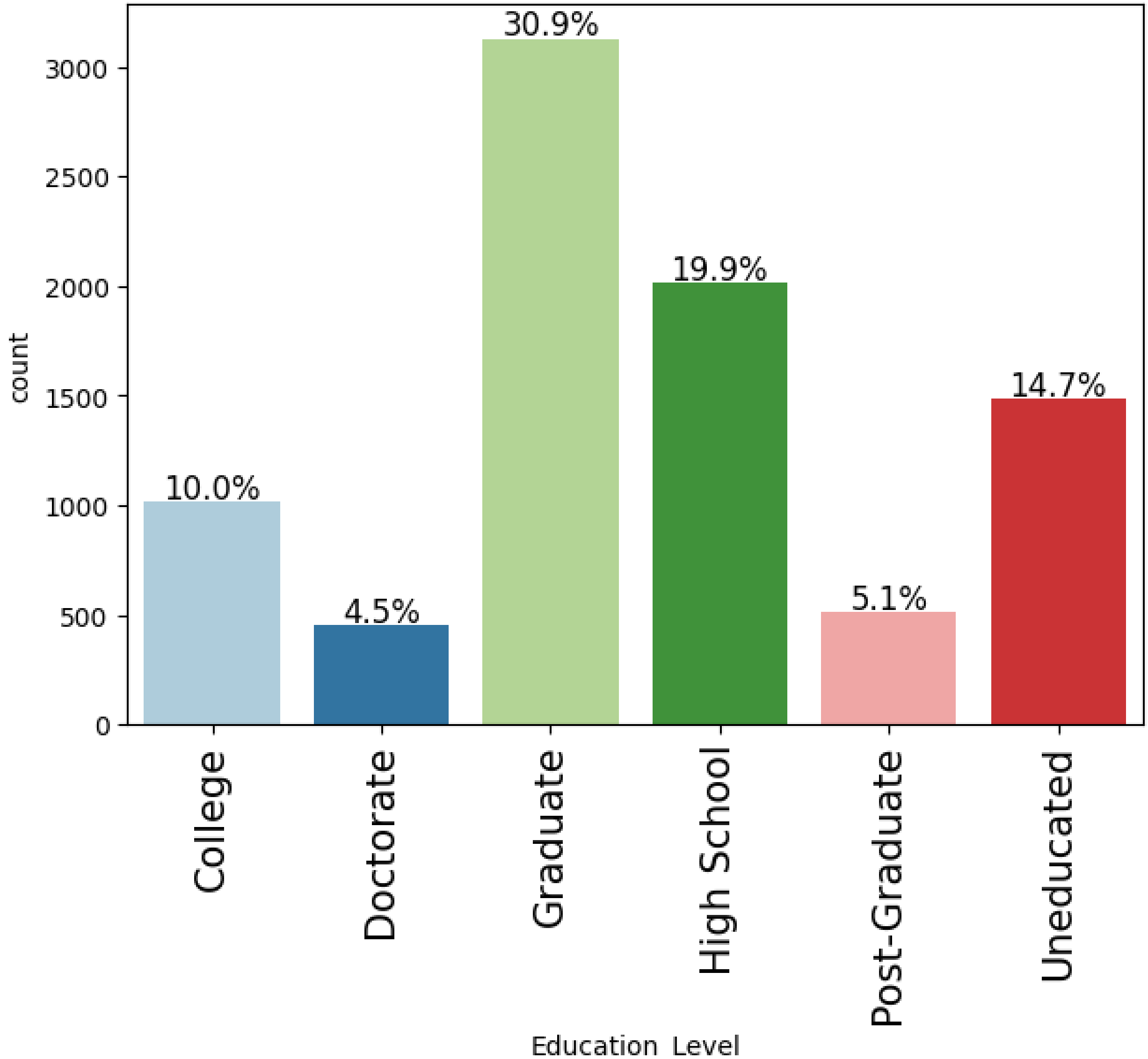
In [123…

```python
histogram_boxplot(data,"Total_Trans_Amt")
```



Majority of the total transaction amounts are within the range of $1000 and 5000$. There is considerarble transactions found btween 7500 and 10000, also 12500 and 17000

In [125…
```
labeled_barplot(data,"Education_Level",perc=True)
```

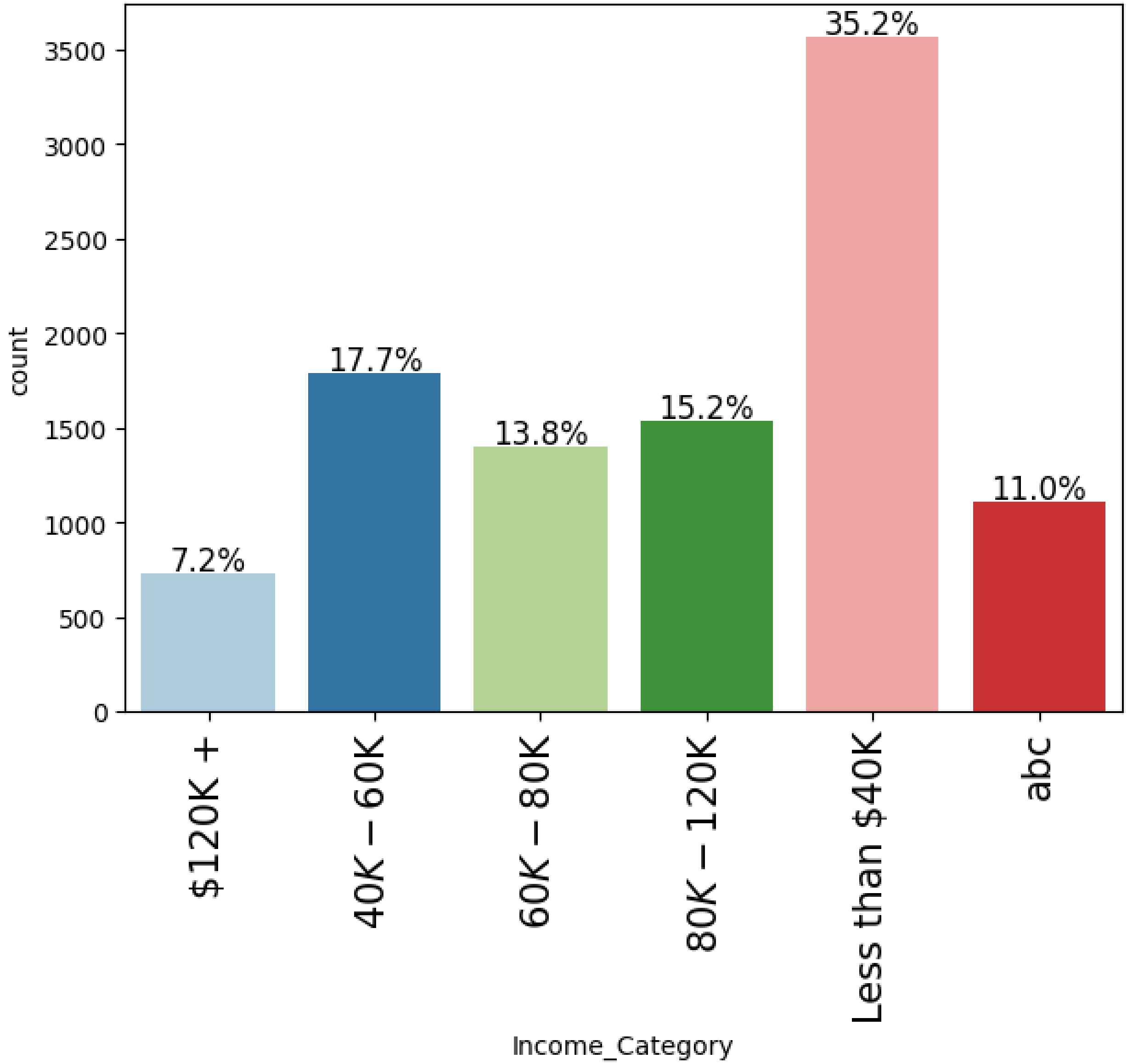


Distribution of education level is follows:

Graduate - 31% High School - 20% Uneducated - 15% College - 10% Post graduate - 5% Doctorate - 5%

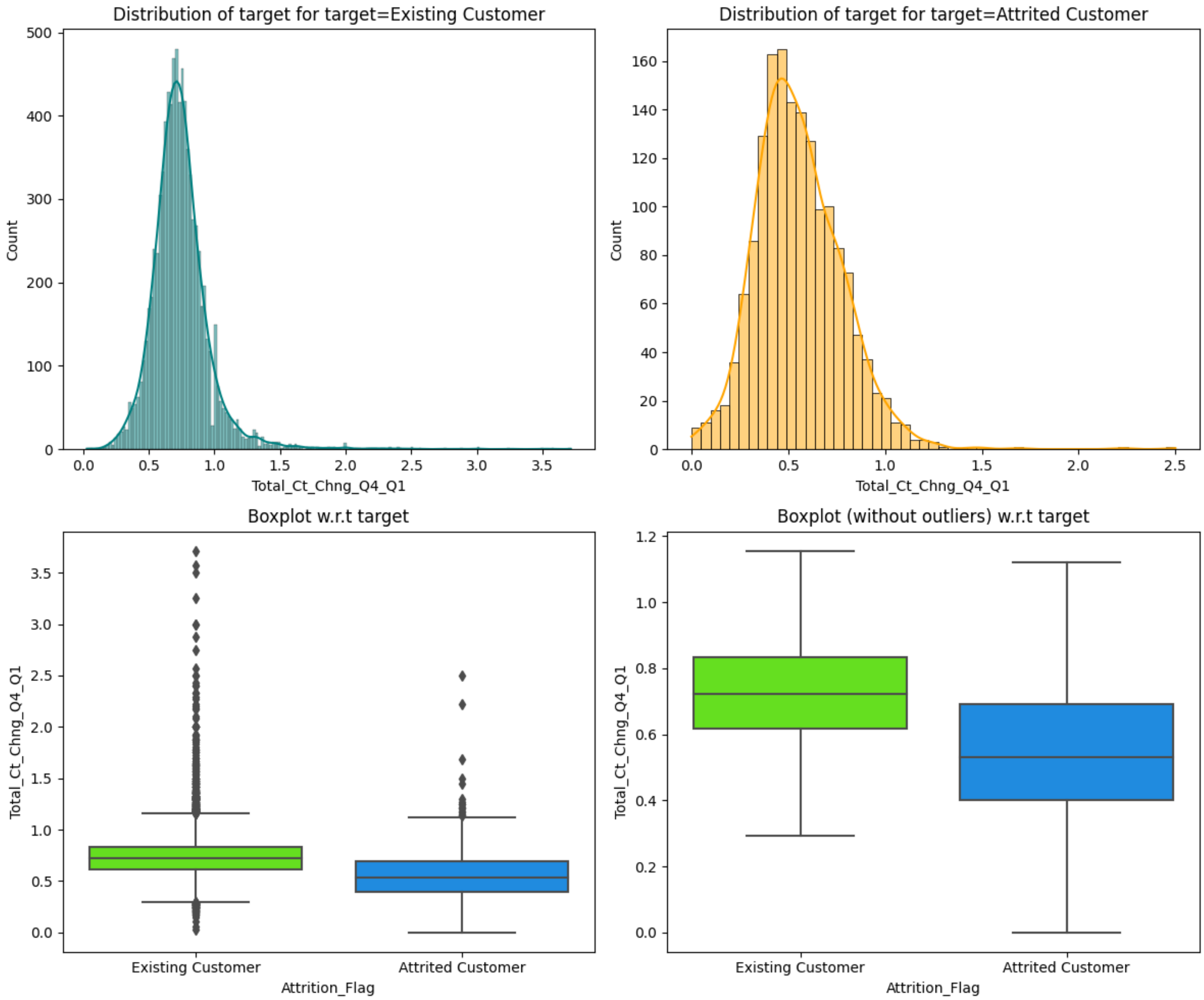
In [126…

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



In [127…

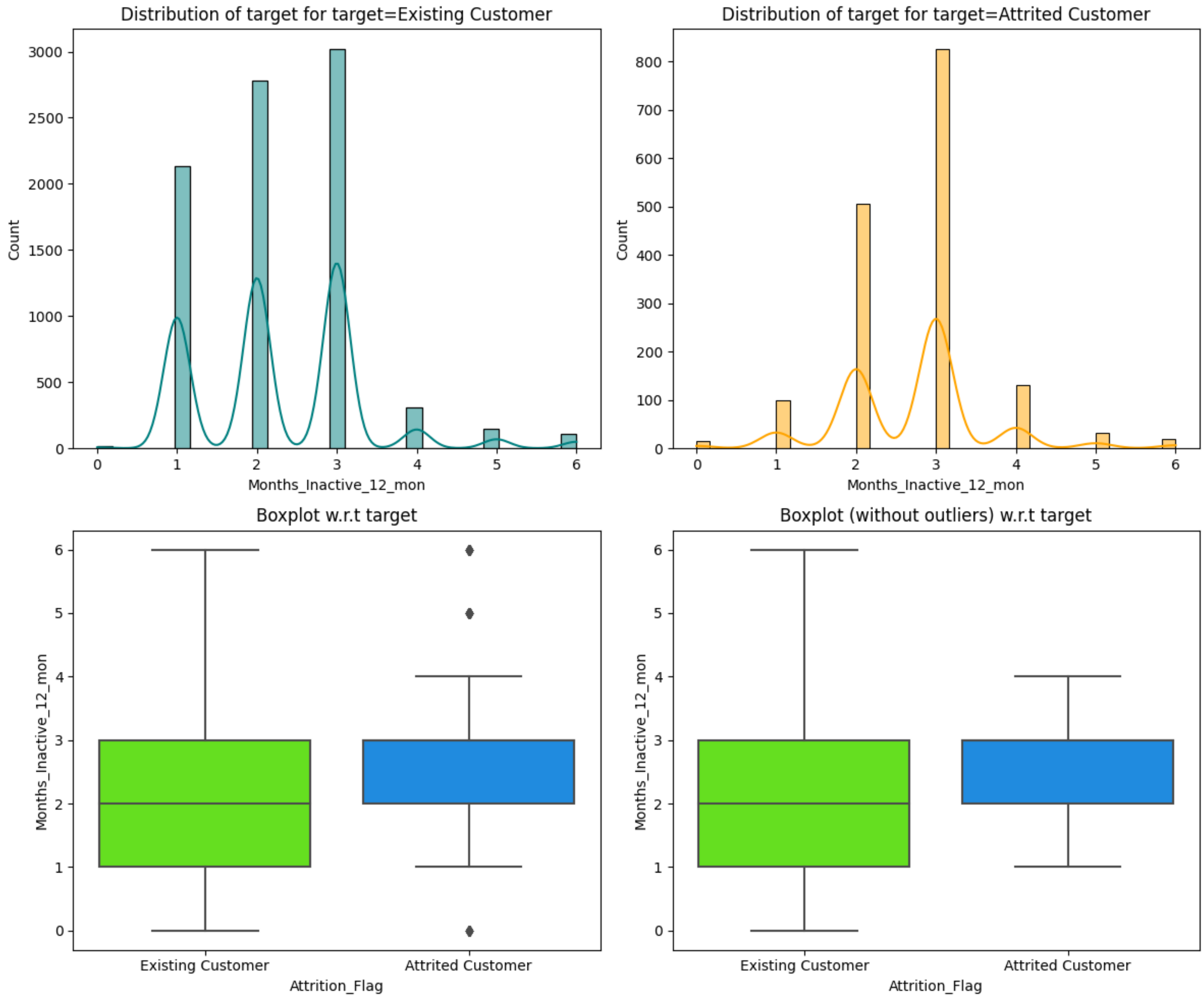
```
distribution_plot_wrt_target(df,"Total_Ct_Chng_Q4_Q1","Attrition_Flag" )
```



Total count change between Q4 and Q1 is between 0.6 and 0.8 for Existing customers and for the Attrited customer it is between 0.4 and 0.7

In [128…

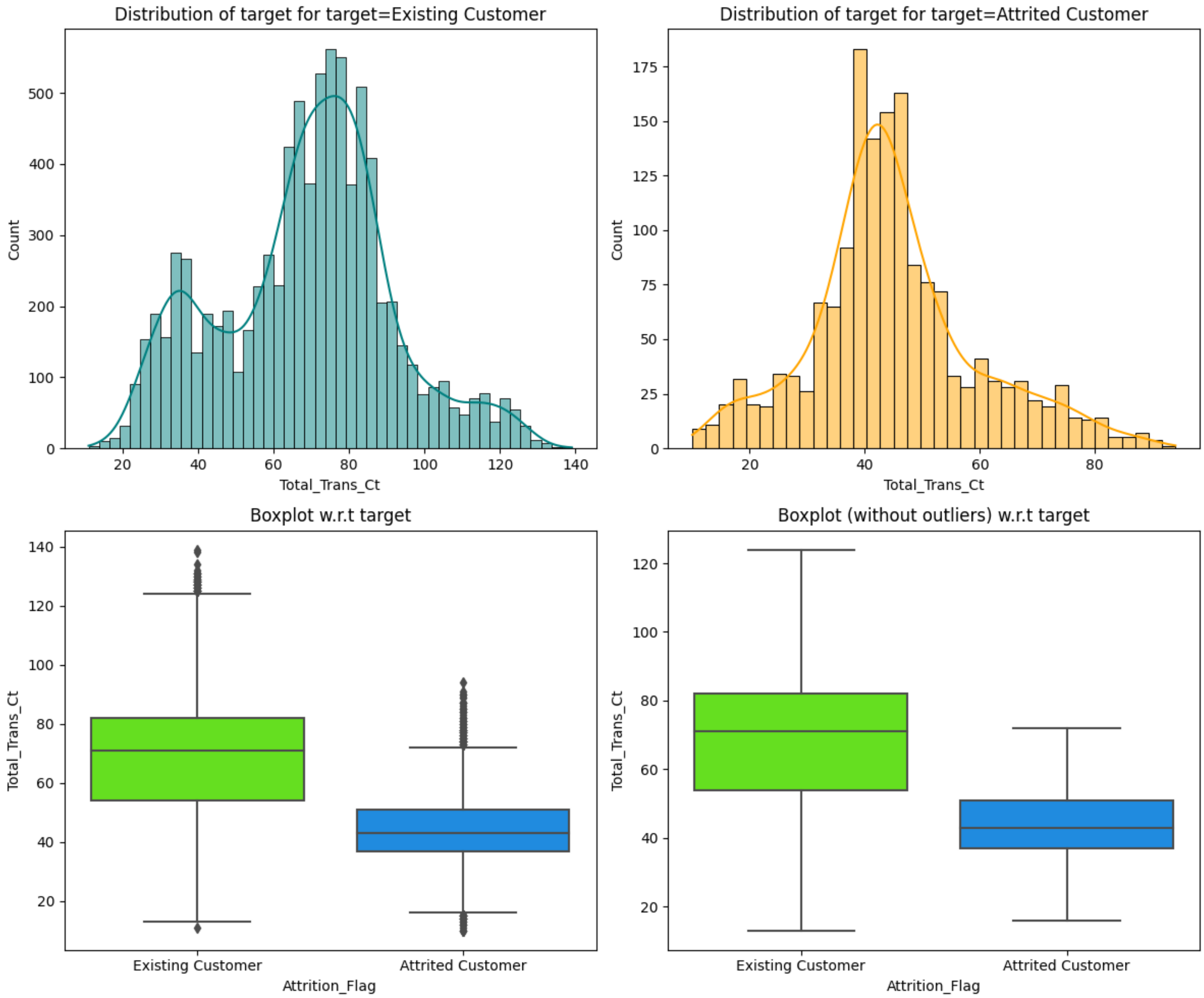
```
distribution_plot_wrt_target(df,"Months_Inactive_12_mon","Attrition_Flag" )
```



Existing customer is inactive in the range of 1 to three months. Attrited customer is inactive in the range of 2-3 months. This information is not that helpful

In [208…

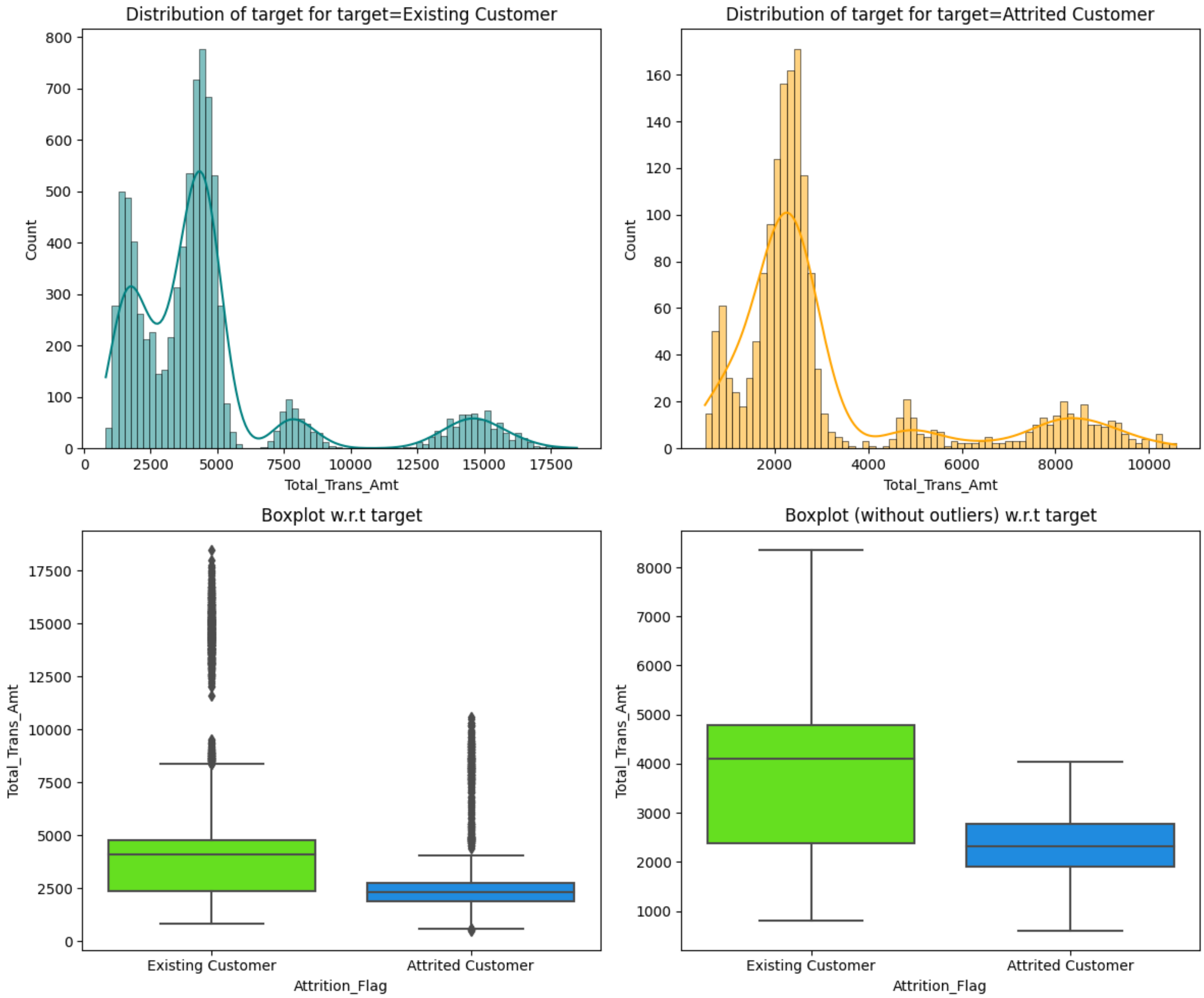
```
distribution_plot_wrt_target(df,"Total_Trans_Ct","Attrition_Flag" )
```



Less number of Transaction count can be seen with Attrited customer.

In [209…

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



Total transcations is comparatively less for Attrited Customer

In [129…

```
df.head()
```

Out[129]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_
0	768805383	Existing Customer	45	M	3	High School	Married	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less
2	713982108	Existing Customer	51	M	3	Graduate	Married	8
3	769911858	Existing Customer	40	F	4	High School	NaN	Less
4	709106358	Existing Customer	40	M	3	Uneducated	Married	

5 rows × 21 columns

In [130…

```
data.corr(method='kendall')
```

Out[130]:

	Attrition_Flag	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Cou

Attrition_Flag	1.000000	0.014541	0.018786	0.012909	-0.1326
Customer_Age	0.014541	1.000000	-0.109732	0.613382	-0.0105
Dependent_count	0.018786	-0.109732	1.000000	-0.088070	-0.0283
Months_on_book	0.012909	0.613382	-0.088070	1.000000	-0.0104
Total_Relationship_Count	-0.132604	-0.010539	-0.028358	-0.010424	1.0000
Months_Inactive_12_mon	0.159147	0.034375	-0.007615	0.045357	-0.0054
Contacts_Count_12_mon	0.172018	-0.010912	-0.033652	-0.006405	0.0491
Credit_Limit	-0.041671	0.001835	0.037302	0.004765	-0.0427
Total_Revolving_Bal	-0.201344	0.009431	-0.002673	0.004449	0.0088
Avg_Open_To_Buy	0.022468	-0.001169	0.039961	0.005326	-0.0513
Total_Amt_Chng_Q4_Q1	-0.083322	-0.048314	-0.019210	-0.037716	0.0182
Total_Trans_Amt	-0.182744	-0.023698	0.042314	-0.018914	-0.1991
Total_Trans_Ct	-0.309061	-0.035966	0.038785	-0.026471	-0.1634
Total_Ct_Chng_Q4_Q1	-0.255235	-0.027597	0.006866	-0.023385	0.0173
Avg_Utilization_Ratio	-0.201005	0.007341	-0.026187	-0.002473	0.0482

1. Total_Trans_Ct and Total_Ct_Chng_Q4_Q1 has good correlation with attrition_flag
2. Customer_Age and Months_on_bool is strongly related
3. Credit limit and Ave_utiliztion_Ratio has string corelation
4. Total_revolving_bal and Avg_Utlization_Ratio has good corelation
5. Total_Trans_Amt and Total_trans_ct has strong corelation 6.

Data Pre-processing

In [131…
```
data["Income_Category"].replace("abc","Less than $40K",inplace=True)
```

Missing value imputation

In [207…
```
#Replacing the "abc" with "Unknown" in Income_Category
data["Income_Category"].replace("abc","Unknown",inplace=True)
# Replacing Null with "Unknown" for Marital_status and Education_level
imputer = SimpleImputer(missing_values=np.NaN, strategy='constant',fill_value='Unknown')
data.Marital_Status = imputer.fit_transform(data['Marital_Status'].values.reshape(-1,1))
data.Education_Level = imputer.fit_transform(data['Education_Level'].values.reshape(-1,1
```

In [133…
```
# Looking at value counts for non-numeric features

num_to_display = 10

for colname in data.dtypes[df.dtypes == "object"].index:
  val_counts = data[colname].value_counts(dropna=False)
  print(val_counts[:num_to_display])

  if  len(val_counts) > num_to_display:
    print(f"Only displaying first {num_to_display} of {len(val_counts)} values.")
  print("-" * 50, "\n") # just for more in between
```

```
0    8500
1    1627
Name: Attrition_Flag, dtype: int64
--------------------------------------------------

F    5358
M    4769
Name: Gender, dtype: int64
--------------------------------------------------

Graduate         3128
High School      2013
Unknown          1519
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64
--------------------------------------------------

Married     4687
Single      3943
Unknown      749
Divorced     748
Name: Marital_Status, dtype: int64
--------------------------------------------------

Less than $40K    4673
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
$120K +            727
Name: Income_Category, dtype: int64
--------------------------------------------------

Blue        9436
Silver       555
Gold         116
Platinum      20
Name: Card_Category, dtype: int64
--------------------------------------------------
```

In [133…

```

```

Model Building

In [134…

```
# Let us check the ratio of Attrition customers to existsing customers
data['Attrition_Flag'].value_counts(1)
```

Out[134]:

```
0    0.83934
1    0.16066
Name: Attrition_Flag, dtype: float64
```

In [135…

```
# separating the independent and dependent variabes
X = data.drop(["Attrition_Flag"],axis =1)
y=data["Attrition_Flag"]

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

In [136…

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

```
X_train, X_temp, y_train, y_temp = train_test_split(X,y,test_size=0.5,random_state=0,str

# then we split the temporary set into train and validation
X_val, X_test,y_val, y_test = train_test_split(X_temp,y_temp,test_size=0.4,random_state=

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

```
(5063, 31) (3038, 31) (2026, 31)
```

In [137…

```
# Checking class balance for whole data, train set,

print("Target value ration in y")
print(y.value_counts(1))
print("*" * 80)
print("Target value ration in y_train")
print(y_train.value_counts(1))
print("*" * 80)
print("Target value ration in y_val")
print(y_val.value_counts(1))
print("*" * 80)
print("Target vlaue ration in y_test")
print(y_test.value_counts(1))
print("*" * 80)
```

```
Target value ration in y
0    0.83934
1    0.16066
Name: Attrition_Flag, dtype: float64
********************************************************************************
Target value ration in y_train
0    0.839423
1    0.160577
Name: Attrition_Flag, dtype: float64
********************************************************************************
Target value ration in y_val
0    0.839368
1    0.160632
Name: Attrition_Flag, dtype: float64
********************************************************************************
Target vlaue ration in y_test
0    0.839092
1    0.160908
Name: Attrition_Flag, dtype: float64
********************************************************************************
```

In [138…

```
# Let us check the future importances
dtree = DecisionTreeClassifier(random_state=1, max_depth=4)
dtree.fit(X_train,y_train)
```

Out[138]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, random_state=1)
```

In [139…

```
print(pd.DataFrame(dtree.feature_importances_,columns =["imp"], index= X_train.columns))
```

```
                                imp
Customer_Age               0.000000
Dependent_count            0.000000
Months_on_book             0.000000
Total_Relationship_Count   0.125006
Months_Inactive_12_mon     0.014911
Contacts_Count_12_mon      0.000000
Credit_Limit               0.000000
Total_Revolving_Bal        0.247207
Avg_Open_To_Buy            0.000000
```

```
Total_Amt_Chng_Q4_Q1          0.002398
Total_Trans_Amt               0.084493
Total_Trans_Ct                0.453334
Total_Ct_Chng_Q4_Q1           0.072652
Avg_Utilization_Ratio         0.000000
Gender_M                      0.000000
Education_Level_Doctorate     0.000000
Education_Level_Graduate      0.000000
Education_Level_High School   0.000000
Education_Level_Post-Graduate 0.000000
Education_Level_Uneducated    0.000000
Education_Level_Unknown       0.000000
Marital_Status_Married        0.000000
Marital_Status_Single         0.000000
Marital_Status_Unknown        0.000000
Income_Category_$40K - $60K   0.000000
Income_Category_$60K - $80K   0.000000
Income_Category_$80K - $120K  0.000000
Income_Category_Less than $40K 0.000000
Card_Category_Gold            0.000000
Card_Category_Platinum        0.000000
Card_Category_Silver          0.000000
```

In [140…

```
# Predicting the target for train and validation set
pred_train = dtree.predict(X_train)
pred_val = dtree.predict(X_val)
```

In [141…

```
# Checking recall score on oversampled train and validation set
print(recall_score(y_train, pred_train))
print(recall_score(y_val,pred_val))
```

```
0.6691266912669127
0.6639344262295082
```

In [142…

```
# Checking accuracy score on oversampled train and validation set
print(accuracy_score(y_train, pred_train))
print(accuracy_score(y_val,pred_val))
```

```
0.9275133320165909
0.9229756418696511
```

Model evaluation criterion

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

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

Which metric to optimize?

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

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

evaluating models.

In [143…

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

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

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

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

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

        },
        index=[0],
    )

    return df_perf
```

Model Building with original data

Sample code for model building with original data

In [144…

```
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)))
'_______' ## Complete the code to append remaining 3 models in the list models

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

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

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

```
Training Performance:

Bagging: 0.971709717097171
Random forest: 1.0
```

```
Validation Performance:

Bagging: 0.7786885245901639
Random forest: 0.7295081967213115
```

Model Building with Oversampled data

In [145…
```
# Synthetic Minority Over Sampling Technique
sm = SMOTE(sampling_strategy=1, k_neighbors=5, random_state=1)
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)
```

In [146…
```
print("Before OverSampling, count of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, count of label '0': {} \n".format(sum(y_train == 0)))

print("After OverSampling, count of label '1': {}".format(sum(y_train_over == 1)))
print("After OverSampling, count of label '0': {} \n".format(sum(y_train_over == 0)))

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

```
Before OverSampling, count of label '1': 813
Before OverSampling, count of label '0': 4250

After OverSampling, count of label '1': 4250
After OverSampling, count of label '0': 4250

After OverSampling, the shape of train_X: (8500, 31)
After OverSampling, the shape of train_y: (8500,)
```

In [147…
```
dtree1 = DecisionTreeClassifier(random_state=1, max_depth=4)

# training the decision tree model with oversampled training set
dtree1.fit(X_train_over, y_train_over)
```

Out[147]:
```
DecisionTreeClassifier

DecisionTreeClassifier(max_depth=4, random_state=1)
```

In [148…
```
# Predicting the target for train and validation set
pred_train = dtree1.predict(X_train_over)
pred_val = dtree1.predict(X_val)
```

In [149…
```
# Checking recall score on oversampled train and validation set
print(recall_score(y_train_over, pred_train))
print(recall_score(y_val, pred_val))
```

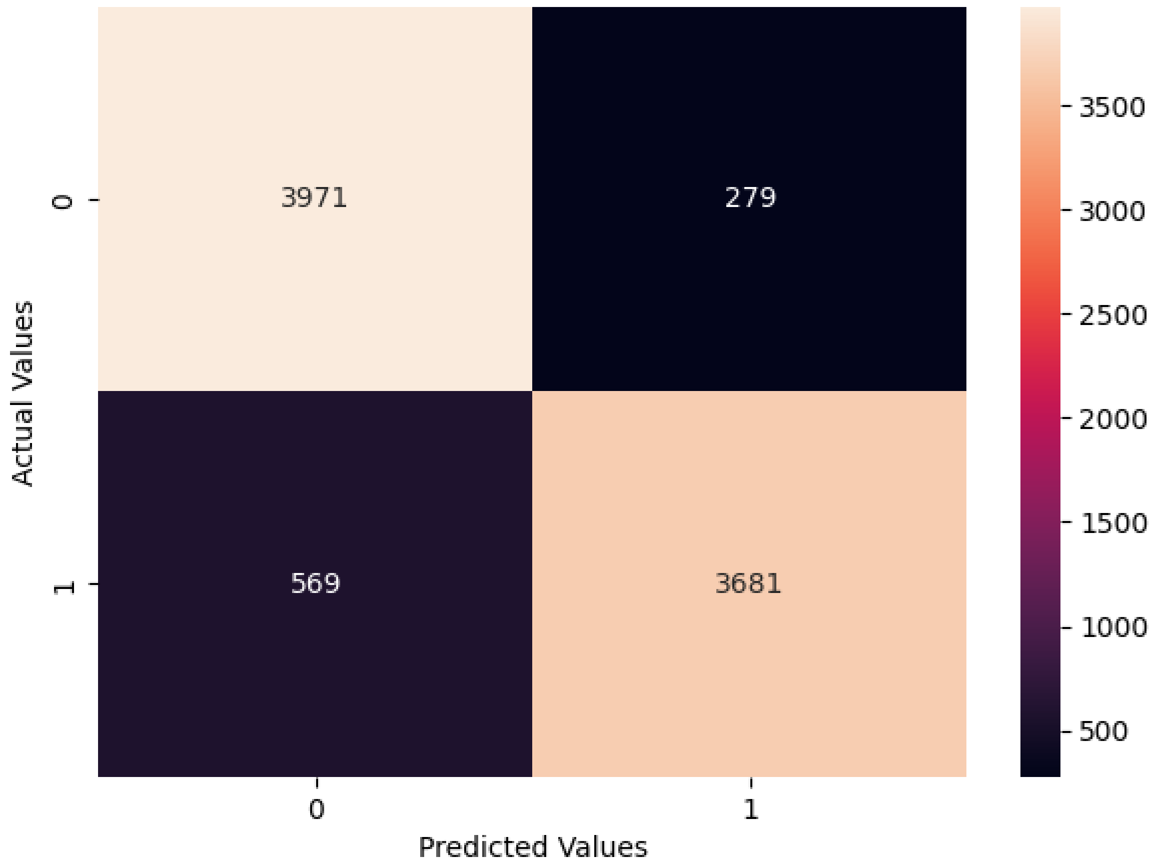
```
0.8661176470588235
0.7889344262295082
```

In [150…
```
# Checking accuracy score on oversampled train and validation set
print(accuracy_score(y_train_over, pred_train))
print(accuracy_score(y_val, pred_val))
```

```
0.900235294117647
0.9124423963133641
```

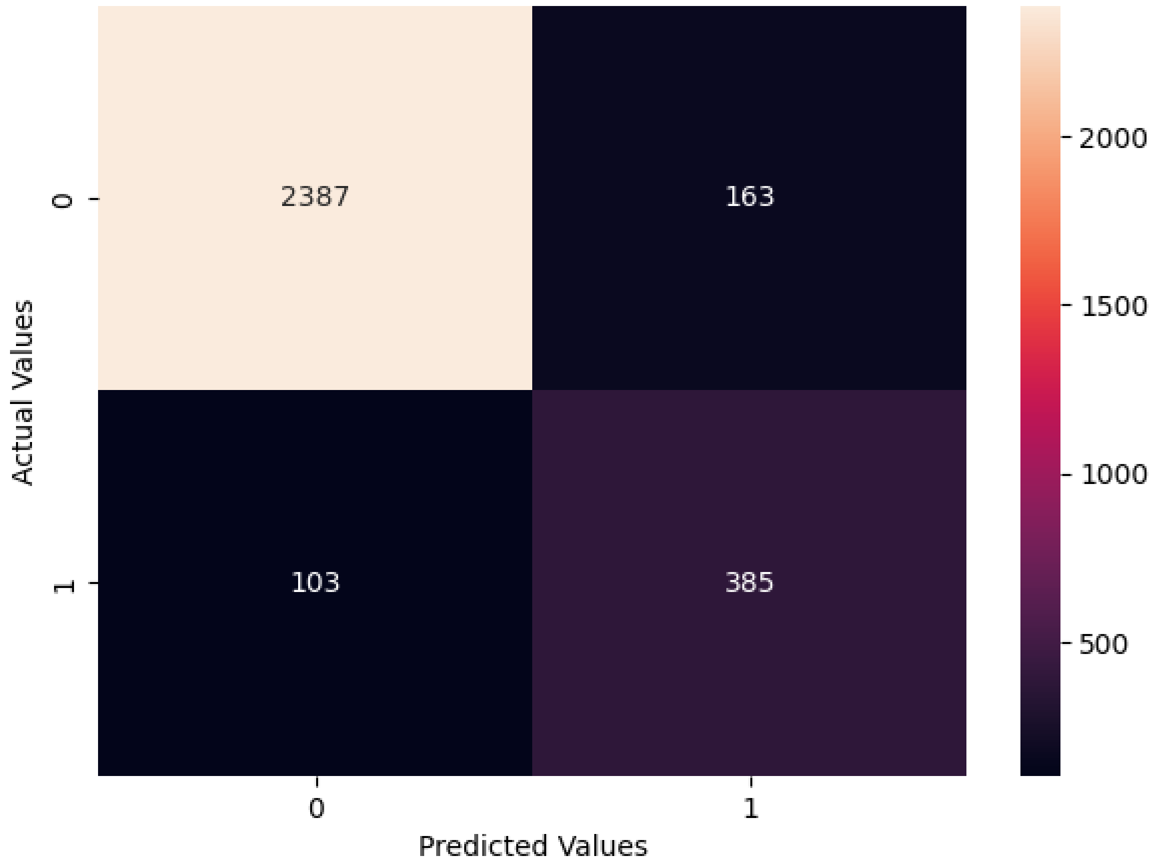
In [151…
```
# Confusion matrix for oversampled train data
cm = confusion_matrix(y_train_over,pred_train)
plt.figure(figsize=(7,5))
sns.heatmap(cm,annot=True, fmt="g")
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
```

Out[151]: Text(58.222222222222214, 0.5, 'Actual Values')



In [152…

```
# Confusion matrix for validation data
cm = confusion_matrix(y_val,pred_val)
plt.figure(figsize=(7,5))
sns.heatmap(cm, annot=True,fmt="g")
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
plt.show()
```



Model Building with Undersampled data

In [153…
```
# Random undersampler for under sampling the data
rus = RandomUnderSampler(random_state=1, sampling_strategy=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

In [154…
```
print("Before Under Sampling, count of label '1' : {}".format(sum(y_train == 1)))
print("Before Under Sampling, count of label '0' : {}".format(sum(y_train == 0)))

print("After Under Sampling, count of label '1' : {}".format(sum(y_train_un == 1)))
print("After Under Sampling, count of label '0' : {}".format(sum(y_train_un == 0)))

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

```
Before Under Sampling, count of label '1' : 813
Before Under Sampling, count of label '0' : 4250
After Under Sampling, count of label '1' : 813
After Under Sampling, count of label '0' : 813
After Under Sampling, the shape of train_X: (1626, 31)
After Under Sampling, the shape of train_y: (1626,)
```

In [155…
```
dtree2 = DecisionTreeClassifier(random_state=1, max_depth=4)

#training the decision tree model with undersampled training set
dtree2.fit(X_train_un, y_train_un)
```

Out[155]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, random_state=1)
```

In [156…
```
# Predicting the target for train and validation set
pred_train = dtree2.predict(X_train_un)
pred_val = dtree2.predict(X_val)
```

In [157…

```python
# Checking recall score on oversampled train and validation set
print(recall_score(y_train_un,pred_train))
print(recall_score(y_val, pred_val))
```

```
0.8425584255842559
0.805327868852459
```

In [158…

```python
#Checking accuracy score on undersamled train and validation set
print(accuracy_score(y_train_un, pred_train))
print(accuracy_score(y_val,pred_val))
```

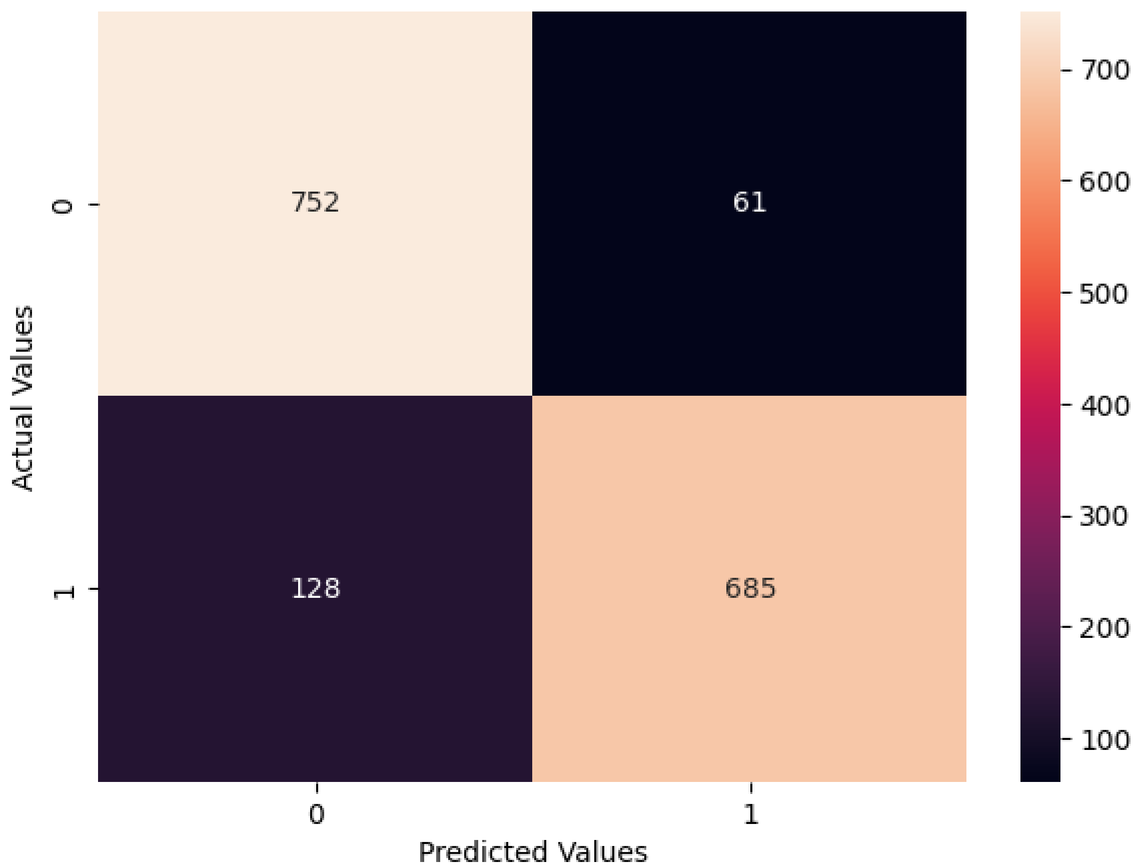
```
0.8837638376383764
0.9012508229098091
```

In [159…

```python
# Confusion matrix for undersample train data
cm = confusion_matrix(y_train_un, pred_train)
plt.figure(figsize=(7,5))
sns.heatmap(cm, annot=True, fmt="g")
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
```

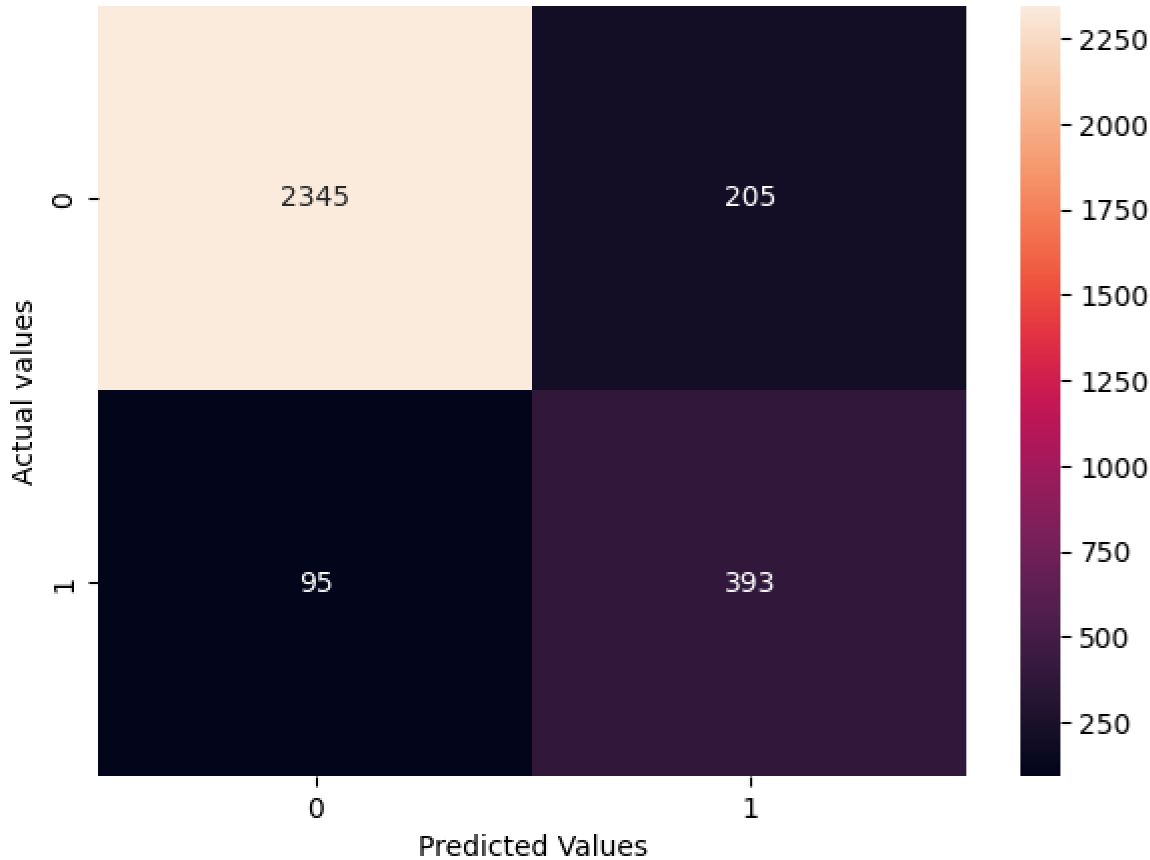
Out[159]:

```
Text(58.222222222222214, 0.5, 'Actual Values')
```



In [160…

```python
# Confusion matrix for validation data
cm = confusion_matrix(y_val,pred_val)
plt.figure(figsize=(7,5))
sns.heatmap(cm,annot=True,fmt="g")
plt.xlabel("Predicted Values")
plt.ylabel("Actual values")
plt.show()
```



HyperparameterTuning

Sample Parameter Grids

Hyperparameter tuning can take a long time to run, so to avoid that time complexity - you can use the following grids, wherever required.

- For Gradient Boosting:

```
param_grid = {
    "init":
[AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random_state=1)],
    "n_estimators": np.arange(75,150,25),
    "learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "subsample":[0.5,0.7,1],
    "max_features":[0.5,0.7,1],
}
```

- For Adaboost:

```
param_grid = {
     "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),
    ]
}
```

- For Bagging Classifier:

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

- For Random Forest:

```
param_grid = {
    "n_estimators": [200,250,300],
    "min_samples_leaf": np.arange(1, 4),
    "max_features": [np.arange(0.3, 0.6, 0.1),'sqrt'],
    "max_samples": np.arange(0.4, 0.7, 0.1)
}
```

- For Decision Trees:

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

- For XGBoost:

```
param_grid={
   'n_estimators':np.arange(50,300,50),
   'scale_pos_weight':[0,1,2,5,10],
   'learning_rate':[0.01,0.1,0.2,0.05],
   'gamma':[0,1,3,5],
   'subsample':[0.7,0.8,0.9,1]
}
```

In [161…

```
##  Function to calculate different metric scores of the model - Accuracy, Recall and Pr
def get_metrics_score(model,flag=True):
    '''
    model : classifier to predict values of X

    '''
    # defining an empty list to store train and test results
    score_list=[]

    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    train_acc = model.score(X_train,y_train)
    test_acc = model.score(X_test,y_test)

    train_recall = metrics.recall_score(y_train,pred_train)
    test_recall = metrics.recall_score(y_test,pred_test)

    train_precision = metrics.precision_score(y_train,pred_train)
    test_precision = metrics.precision_score(y_test,pred_test)

    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,test_
```

```
    # If the flag is set to True then only the following print statements will be dispay
    if flag == True:
        print("Accuracy on training set : ",model.score(X_train,y_train))
        print("Accuracy on test set : ",model.score(X_test,y_test))
        print("Recall on training set : ",metrics.recall_score(y_train,pred_train))
        print("Recall on test set : ",metrics.recall_score(y_test,pred_test))
        print("Precision on training set : ",metrics.precision_score(y_train,pred_train)
        print("Precision on test set : ",metrics.precision_score(y_test,pred_test))

    return score_list # returning the list with train and test scores
```

In [161…

Sample tuning method for Decision tree with original data

In [162…

```
#defining Grandient Boosting model

Grad_Model = GradientBoostingClassifier(random_state=1)

param_grid = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random_state=1)],
    "n_estimators": np.arange(75,150,25),
    "learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "subsample":[0.5,0.7,1],
    "max_features":[0.5,0.7,1],
}

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

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Grad_Model, param_distributions=param_grid,

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

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

```
Best parameters are {'subsample': 0.7, 'n_estimators': 100, 'max_features': 0.5, 'learni
ng_rate': 0.2, 'init': AdaBoostClassifier(random_state=1)} with CV score=0.8659774293721
124:
```

In [163…

```
# Set the clf to the best combination of parameters
Grad_Model = randomized_cv.best_estimator_

# Fit the best algorithm to the data.
Grad_Model.fit(X_train, y_train)
```

Out[163]:

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

In [164…

```
# Predicting the traget for train and validation set
pred_train = Grad_Model.predict(X_train_over)
pred_val = Grad_Model.predict(X_val)
```

In [165…

```
print(recall_score(y_train_over,pred_train))
print(recall_score(y_val, pred_val))
```

```
0.9021176470588236
```

```
0.860655737704918
```

In [166…
```
#Using above defined function to get accuracy, recall and precision on train and test se
Grad_Model=get_metrics_score(Grad_Model)
```

```
Accuracy on training set :  0.9861742050167884
Accuracy on test set :  0.9669299111549852
Recall on training set :  0.9348093480934809
Recall on test set :  0.8374233128834356
Precision on training set :  0.9781209781209781
Precision on test set :  0.9512195121951219
```

In [167…
```
# defining  Adaboost Model
abc_model = AdaBoostClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
     "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
randomized_cv = RandomizedSearchCV(estimator=abc_model, param_distributions=param_grid,

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

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
# Set the clf to the best combination of parameters
abc_model = randomized_cv.best_estimator_
```

```
Best parameters are {'n_estimators': 90, 'learning_rate': 1, 'base_estimator': DecisionT
reeClassifier(max_depth=2, random_state=1)} with CV score=0.8647125653260621:
```

In [168…
```
# Predicting the traget for train and validation set
pred_train = abc_model.predict(X_train_over)
pred_val = abc_model.predict(X_val)
```

In [169…
```
print(recall_score(y_train_over,pred_train))
print(recall_score(y_val, pred_val))
```

```
0.9409411764705883
0.875
```

In [172…
```
# Fit the best algorithm to the data.
abc_model.fit(X_train, y_train)

#Using above defined function to get accuracy, recall and precision on train and test se
abc_model=get_metrics_score(abc_model)
```

```
Accuracy on training set :  0.9990124432154849
Accuracy on test set :  0.9713721618953604
Recall on training set :  0.995079950799508
Recall on test set :  0.8895705521472392
Precision on training set :  0.9987654320987654
Precision on test set :  0.9294871794871795
```

In [198…
```
# defining XGB booster model
```

```
xgb_tuned = XGBClassifier(random_state=1,eval_metric='logloss')

param_grid={
   'n_estimators':np.arange(50,300,50),
   'scale_pos_weight':[0,1,2,5,10],
   'learning_rate':[0.01,0.1,0.2,0.05],
   'gamma':[0,1,3,5],
   'subsample':[0.7,0.8,0.9,1]
}


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

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=xgb_tuned, param_distributions=param_grid,

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

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
# Set the clf to the best combination of parameters
xgb_tuned = randomized_cv.best_estimator_
```

```
Best parameters are {'subsample': 0.7, 'scale_pos_weight': 10, 'n_estimators': 250, 'lea
rning_rate': 0.2, 'gamma': 3} with CV score=0.9188593501476937:
```

In [200…

```
# Predicting the traget for train and validation set
pred_train = xgb_tuned.predict(X_train_over)
pred_val = xgb_tuned.predict(X_val)
```

In [175…

```
print(recall_score(y_train_over,pred_train))
print(recall_score(y_val, pred_val))
```

```
0.9832941176470589
0.9385245901639344
```

In [176…

```
# Fit the best algorithm to the data.
xgb_tuned.fit(X_train, y_train)

#Using above defined function to get accuracy, recall and precision on train and test se
xgb_tuned=get_metrics_score(xgb_tuned)
```

```
Accuracy on training set :  0.9976298637171638
Accuracy on test set :  0.9659427443237907
Recall on training set :  1.0
Recall on test set :  0.9202453987730062
Precision on training set :  0.9854545454545455
Precision on test set :  0.8746355685131195
```

In [177…

```
# Defining bagging model
bag_model = BaggingClassifier(random_state=1)

param_grid = {
    'max_samples': [0.8,0.9,1],
    'max_features': [0.7,0.8,0.9],
    'n_estimators' : [30,50,70],
}
# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

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

#Calling RandomizedSearchCV
```

```
randomized_cv = RandomizedSearchCV(estimator=bag_model, param_distributions=param_grid,

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

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
# Set the clf to the best combination of parameters
bag_model = randomized_cv.best_estimator_
```

```
Best parameters are {'n_estimators': 70, 'max_samples': 0.8, 'max_features': 0.9} with C
V score=0.8228735893357569:
```

In [178…

```
# Defining Random Forest model
ran_model = RandomForestClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "n_estimators": [200,250,300],
    "min_samples_leaf": np.arange(1, 4),
    "max_features": [np.arange(0.3, 0.6, 0.1),'sqrt'],
    "max_samples": np.arange(0.4, 0.7, 0.1)
}

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

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=ran_model, param_distributions=param_grid,

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

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
# Set the clf to the best combination of parameters
ran_model = randomized_cv.best_estimator_
```

```
Best parameters are {'n_estimators': 300, 'min_samples_leaf': 1, 'max_samples': 0.6, 'ma
x_features': 'sqrt'} with CV score=0.7343179580398396:
```

In [179…

```
# Fit the best algorithm to the data.
ran_model.fit(X_train, y_train)

#Using above defined function to get accuracy, recall and precision on train and test se
ran_model=get_metrics_score(ran_model)
```

```
Accuracy on training set :  0.9984199091447759
Accuracy on test set :  0.9496544916090819
Recall on training set :  0.990159901599016
Recall on test set :  0.7300613496932515
Precision on training set :  1.0
Precision on test set :  0.9444444444444444
```

In [180…

```
# Fit the best algorithm to the data.
bag_model.fit(X_train, y_train)

#Using above defined function to get accuracy, recall and precision on train and test se
bag_model=get_metrics_score(bag_model)
```

```
Accuracy on training set :  0.9992099545723879
Accuracy on test set :  0.9629812438302073
Recall on training set :  0.995079950799508
Recall on test set :  0.8282208588957055
Precision on training set :  1.0
Precision on test set :  0.9342560553633218
```

In [181…

```
# defining model
Dec_Model = DecisionTreeClassifier(random_state=1)
```

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

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

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Dec_Model, param_distributions=param_grid,

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

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

```
Best parameters are {'min_samples_leaf': 7, 'min_impurity_decrease': 0.0001, 'max_leaf_n
odes': 15, 'max_depth': 5} with CV score=0.7847080209043399:
```

In [182…

```
# Fit the best algorithm to the data.
Dec_Model.fit(X_train, y_train)

#Using above defined function to get accuracy, recall and precision on train and test se
Dec_Model=get_metrics_score(Dec_Model)
```

```
Accuracy on training set :  1.0
Accuracy on test set :  0.9402764067127345
Recall on training set :  1.0
Recall on test set :  0.8159509202453987
Precision on training set :  1.0
Precision on test set :  0.8134556574923547
```

In [183…

```
# Checing the recall score on train and validation set
print("Recall on train and validation set")
#print(recall_score(y_train, Model.predict(X_train)))
#print(recall_score(y_val,Model.predict(X_val)))
print("")

print("Precison on train and validation set")
#print(precision_score(y_train, Model.predict(X_train)))
#print(precision_score(y_val, Model.predict(X_val)))
print("")

print("Accuracy on train and validation set")
#print(accuracy_score(y_train,Model.predict(X_train)))
#print(accuracy_score(y_val,Model.predict(X_val)))
```

```
Recall on train and validation set

Precison on train and validation set

Accuracy on train and validation set
```

Sample tuning method for Decision tree with oversampled data

In [194…

```
# defining model
Model_Oversample_data = DecisionTreeClassifier(random_state=1)

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

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


#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model_Oversample_data, param_distributions=

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

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

# Fit the best algorithm to the data.
Model_Oversample_data.fit(X_train_over, y_train_over)
```

```
Best parameters are {'min_samples_leaf': 7, 'min_impurity_decrease': 0.0001, 'max_leaf_n
odes': 15, 'max_depth': 5} with CV score=0.9209411764705882:
```

Out[194]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=1)
```

In [185…

```python
#Using above defined function to get accuracy, recall and precision on train and test se
Model_Oversample_data=get_metrics_score(Model_Oversample_data)
```

```
Accuracy on training set :  1.0
Accuracy on test set :  0.9234945705824285
Recall on training set :  1.0
Recall on test set :  0.803680981595092
Precision on training set :  1.0
Precision on test set :  0.7422096317280453
```

Sample tuning method for Decision tree with undersampled data

In [193…

```python
# defining model
Model_Undersample_data = DecisionTreeClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {'max_depth': np.arange(2,20),
              'min_samples_leaf': [1, 2, 5, 7],
              'max_leaf_nodes' : [5, 10,15],
              'min_impurity_decrease': [0.0001,0.001] }

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

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model_Undersample_data, param_distributions

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

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,rand
# Fit the best algorithm to the data.

Model_Undersample_data.fit(X_train_un,y_train_un)
```

```
Best parameters are {'min_samples_leaf': 1, 'min_impurity_decrease': 0.001, 'max_leaf_no
des': 5, 'max_depth': 14} with CV score=0.920109066121336:
```

Out[193]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=1)
```

In [187…

```python
Model_Undersample_data.fit(X_train_un,y_train_un)
```

Out[187]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=1)
```

In [188…

```
# Predicting the traget for train and validation set
pred_train = Model_Undersample_data.predict(X_train_over)
pred_val = Model_Undersample_data.predict(X_val)
```

In [189…

```
print(recall_score(y_train_over,pred_train))
print(recall_score(y_val, pred_val))
```

```
0.9658823529411765
0.8811475409836066
```

In [190…

```
#Using above defined function to get accuracy, recall and precision on train and test se
Model_Undersample_data=get_metrics_score(Model_Undersample_data)
```

```
Accuracy on training set :  0.9356112976496148
Accuracy on test set :  0.9067127344521224
Recall on training set :  1.0
Recall on test set :  0.8650306748466258
Precision on training set :  0.713784021071115
Precision on test set :  0.6604215456674473
```

In [191…

```
X_train_over.shape
```

Out[191]:

```
(8500, 31)
```

Model Comparison and Final Model Selection

In [203…

```
# defining list of models
models = [Model_Undersample_data, Model_Oversample_data,xgb_tuned]

# defining empty lists to add train and test results
acc_train = []
acc_test = []
recall_train = []
recall_test = []
precision_train = []
precision_test = []

# looping through all the models to get the accuracy, precall and precision scores
for modelc in models:
    j = get_metrics_score(modelc,False)
    acc_train.append(np.round(j[0],2))
    acc_test.append(np.round(j[1],2))
    recall_train.append(np.round(j[2],2))
    recall_test.append(np.round(j[3],2))
    precision_train.append(np.round(j[4],2))
    precision_test.append(np.round(j[5],2))
```

In [205…

```
comparison_frame = pd.DataFrame({'Model':['Model_Undersample','Model_Oversample','XGB Bo
                                          'Train_Accuracy': acc_train,'Test_Accuracy': a
                                          'Train_Recall':recall_train,'Test_Recall':reca
                                          'Train_Precision':precision_train,'Test_Precis
comparison_frame
```

Out[205]:

	Model	Train_Accuracy	Test_Accuracy	Train_Recall	Test_Recall	Train_Precision	Test_Precision
0	Model_Undersample	0.94	0.91	1.0	0.87	0.71	0.66
1	Model_Oversample	1.00	0.92	1.0	0.80	1.00	0.74

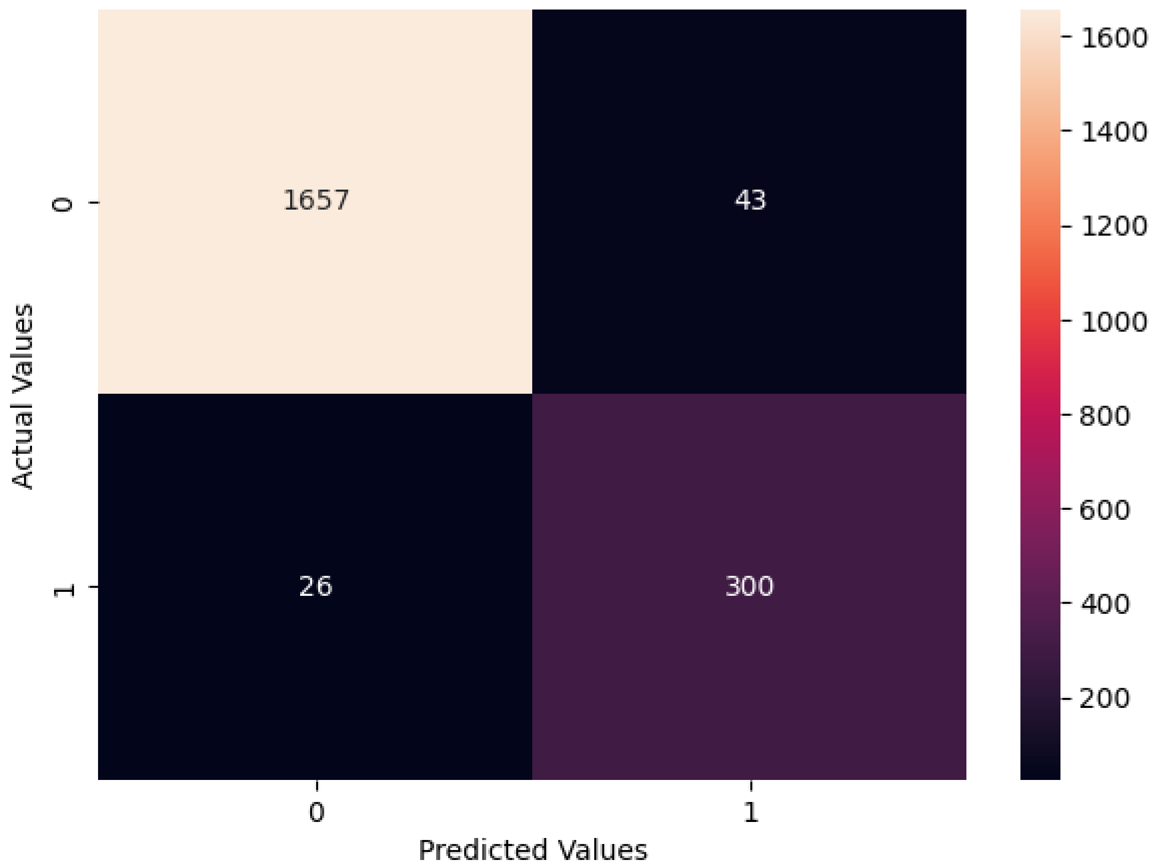
2	XGB Booster	1.00	0.97	1.0	0.92	0.99	0.87

From the various Model comparision, xgb_tuned model has the best values. We will test that with test data

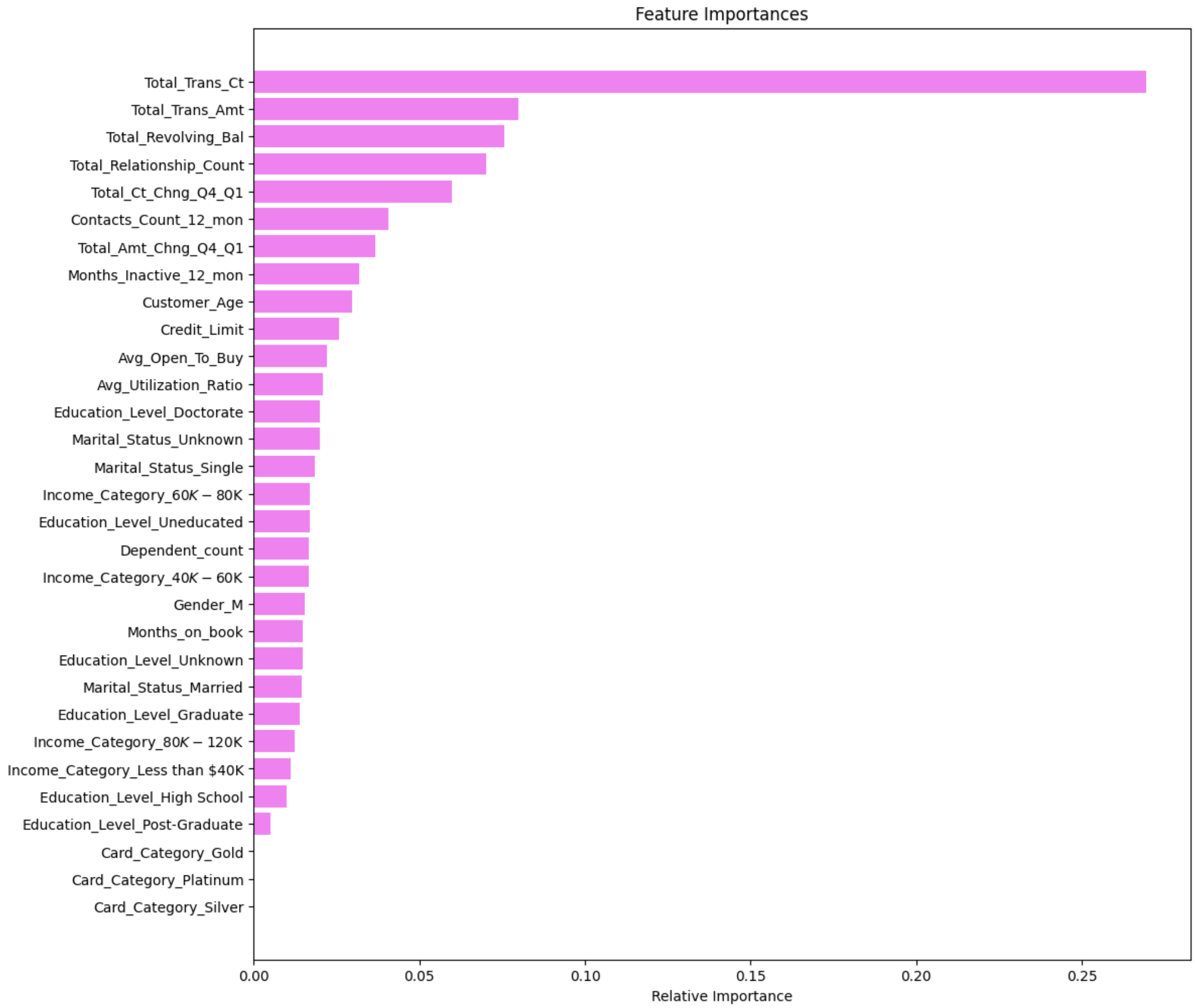
Test set final performance

In [201…
```
# xgb_tuned and Model_Undersample_data has the best performance. Let us run the tests on


print(recall_score(y_test, xgb_tuned.predict(X_test)))
cm = confusion_matrix(y_test, xgb_tuned.predict(X_test))
plt.figure(figsize=(7,5))
sns.heatmap(cm, annot=True, fmt='g')
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
plt.show()
```

```
0.9202453987730062
```



In [202…
```
importances = xgb_tuned.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

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




```
In [ ]:
```

Business Insights and Conclusions

1. Encourge the customers to increase the total transcations. The more transcations customer is likely to stick with the bank
2. Encourage the customer to maintain good revolving balance. As customer who maintain good revolving balance stick with bank.
3. Total Relationship count and Totat_ct_chng_Q4_Q1 has importance in keeping the customer.
