Credit Card Users Churn Prediction

Description

Background & 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

As a Data scientist at Thera bank we 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

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

Objective

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

Data Dictionary:

- 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

Let's start by importing necessary libraries

In [116…

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

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

# To tune model, get different metric scores and split data
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val
from sklearn.metrics import (
    f1_score,
    make_scorer,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    plot_confusion_matrix,
)

# To impute missing values
from sklearn.impute import SimpleImputer

# To build a logistic regression model
from sklearn.linear_model import LogisticRegression

# Libraries to import decision tree classifier and different ensemble classifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
#from xgboost import XGBClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier
```

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

# To suppress the warnings
import warnings

warnings.filterwarnings("ignore")
```

Load and overview the dataset

In [45]:
```
df = pd.read_csv('BankChurners.csv')
```

In [46]:
```
# Checking the number of rows and columns in the data
df.shape
```

Out[46]: (10127, 21)

In [47]:
```
df.head(10)
```

Out[47]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital
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	
4	709106358	Existing Customer	40	M	3	Uneducated	
5	713061558	Existing Customer	44	M	2	Graduate	
6	810347208	Existing Customer	51	M	4	NaN	
7	818906208	Existing Customer	32	M	0	High School	
8	710930508	Existing Customer	37	M	3	Uneducated	
9	719661558	Existing Customer	48	M	2	Graduate	

10 rows × 21 columns

In [48]:
```
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
```

Observations:

- There are total of 21 columns and 10127 observations in the dataset
- We can see that Education_Level and Martial_Status columns have less than 10127 non-null values

Check the percentage of missing values in each column

In [49]:
```
pd.DataFrame(data={'% of Missing Values':round(df.isna().sum()/df.isna().count()
```

Out[49]:

	% of Missing Values
CLIENTNUM	0.0
Attrition_Flag	0.0
Customer_Age	0.0
Gender	0.0
Dependent_count	0.0
Education_Level	15.0
Marital_Status	7.4
Income_Category	0.0
Card_Category	0.0
Months_on_book	0.0
Total_Relationship_Count	0.0

	% of Missing Values
Months_Inactive_12_mon	0.0
Contacts_Count_12_mon	0.0
Credit_Limit	0.0
Total_Revolving_Bal	0.0
Avg_Open_To_Buy	0.0
Total_Amt_Chng_Q4_Q1	0.0
Total_Trans_Amt	0.0
Total_Trans_Ct	0.0
Total_Ct_Chng_Q4_Q1	0.0
Avg_Utilization_Ratio	0.0

Observations:

- 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.

Let's check the number of unique values in each column

In [50]:
```
df.nunique()
```

Out[50]:
```
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
```

Observations:

- We can drop the column - CLIENTNUM as it is unique for each client and will not add value to the model.

In [51]:
```
#Dropping CustomerID column
df.drop(columns='CLIENTNUM',inplace=True)
```

Summary of the data

In [52]:
```
df.describe().T
```

Out[52]:

	count	mean	std	min	25%	50%	75
Customer_Age	10127.0	46.325960	8.016814	26.0	41.000	46.000	52.0
Dependent_count	10127.0	2.346203	1.298908	0.0	1.000	2.000	3.0
Months_on_book	10127.0	35.928409	7.986416	13.0	31.000	36.000	40.0
Total_Relationship_Count	10127.0	3.812580	1.554408	1.0	3.000	4.000	5.0
Months_Inactive_12_mon	10127.0	2.341167	1.010622	0.0	2.000	2.000	3.0
Contacts_Count_12_mon	10127.0	2.455317	1.106225	0.0	2.000	2.000	3.0
Credit_Limit	10127.0	8631.953698	9088.776650	1438.3	2555.000	4549.000	11067.5
Total_Revolving_Bal	10127.0	1162.814061	814.987335	0.0	359.000	1276.000	1784.0
Avg_Open_To_Buy	10127.0	7469.139637	9090.685324	3.0	1324.500	3474.000	9859.0
Total_Amt_Chng_Q4_Q1	10127.0	0.759941	0.219207	0.0	0.631	0.736	0.8
Total_Trans_Amt	10127.0	4404.086304	3397.129254	510.0	2155.500	3899.000	4741.0
Total_Trans_Ct	10127.0	64.858695	23.472570	10.0	45.000	67.000	81.0
Total_Ct_Chng_Q4_Q1	10127.0	0.712222	0.238086	0.0	0.582	0.702	0.8
Avg_Utilization_Ratio	10127.0	0.274894	0.275691	0.0	0.023	0.176	0.5

In [53]:
```
# let's view the statistical summary of the non-numerical columns in the data
df.describe(exclude=np.number).T
```

Out[53]:

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

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

In [54]:
```
# list of all categorical variables
cat_col = df.columns

# printing the number of occurrences of each unique value in each categorical co
for column in cat_col:
```

```
    print(df[column].value_counts())
    print("-" * 50)
```

```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
--------------------------------------------------
44    500
49    495
46    490
45    486
47    479
43    473
48    472
50    452
42    426
51    398
53    387
41    379
52    376
40    361
39    333
54    307
38    303
55    279
56    262
37    260
57    223
36    221
35    184
58    157
59    157
34    146
33    127
60    127
32    106
65    101
61     93
62     93
31     91
26     78
30     70
63     65
29     56
64     43
27     32
28     29
67      4
68      2
66      2
70      1
73      1
Name: Customer_Age, dtype: int64
--------------------------------------------------
F    5358
M    4769
Name: Gender, dtype: int64
--------------------------------------------------
3    2732
2    2655
1    1838
4    1574
0     904
5     424
Name: Dependent_count, dtype: int64
```

```
--------------------------------------------------
Graduate         3128
High School      2013
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64
--------------------------------------------------
Married     4687
Single      3943
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
--------------------------------------------------
36    2463
37     358
34     353
38     347
39     341
40     333
31     318
35     317
33     305
30     300
41     297
32     289
28     275
43     273
42     271
29     241
44     230
45     227
27     206
46     197
26     186
47     171
25     165
48     162
24     160
49     141
23     116
22     105
56     103
50      96
21      83
51      80
53      78
20      74
13      70
19      63
52      62
```

```
18      58
54      53
55      42
17      39
15      34
16      29
14      16
Name: Months_on_book, dtype: int64
--------------------------------------------------
3    2305
4    1912
5    1891
6    1866
2    1243
1     910
Name: Total_Relationship_Count, dtype: int64
--------------------------------------------------
3    3846
2    3282
1    2233
4     435
5     178
6     124
0      29
Name: Months_Inactive_12_mon, dtype: int64
--------------------------------------------------
3    3380
2    3227
1    1499
4    1392
0     399
5     176
6      54
Name: Contacts_Count_12_mon, dtype: int64
--------------------------------------------------
34516.0    508
1438.3     507
15987.0     18
9959.0      18
23981.0     12
           ...
10587.0      1
15340.0      1
34427.0      1
4975.0       1
3741.0       1
Name: Credit_Limit, Length: 6205, dtype: int64
--------------------------------------------------
0       2470
2517     508
1965      12
1480      12
1664      11
         ...
709        1
637        1
1809       1
613        1
1269       1
Name: Total_Revolving_Bal, Length: 1974, dtype: int64
--------------------------------------------------
1438.3     324
34516.0     98
31999.0     26
787.0        8
```

```
701.0        7
           ...
16177.0      1
33791.0      1
5843.0       1
29.0         1
1570.0       1
Name: Avg_Open_To_Buy, Length: 6813, dtype: int64
--------------------------------------------------
0.791    36
0.712    34
0.743    34
0.718    33
0.735    33
         ..
1.585     1
1.198     1
1.430     1
2.275     1
1.787     1
Name: Total_Amt_Chng_Q4_Q1, Length: 1158, dtype: int64
--------------------------------------------------
4509     11
4253     11
4518     10
2229     10
4042      9
         ..
804       1
2869      1
10468     1
15163     1
8192      1
Name: Total_Trans_Amt, Length: 5033, dtype: int64
--------------------------------------------------
81     208
75     203
71     203
69     202
82     202
      ...
11       2
134      1
132      1
138      1
139      1
Name: Total_Trans_Ct, Length: 126, dtype: int64
--------------------------------------------------
0.667    171
1.000    166
0.500    161
0.750    156
0.600    113
        ...
0.859      1
2.083      1
0.473      1
1.075      1
1.074      1
Name: Total_Ct_Chng_Q4_Q1, Length: 830, dtype: int64
--------------------------------------------------
0.000    2470
0.073      44
0.057      33
0.048      32
```

```
0.060      30
           ...
0.335       1
0.985       1
0.949       1
0.818       1
0.972       1
Name: Avg_Utilization_Ratio, Length: 964, dtype: int64
--------------------------------------------------
```

In [55]:
```
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  object 
 1   Customer_Age              10127 non-null  int64  
 2   Gender                    10127 non-null  object 
 3   Dependent_count           10127 non-null  int64  
 4   Education_Level           8608 non-null   object 
 5   Marital_Status            9378 non-null   object 
 6   Income_Category           10127 non-null  object 
 7   Card_Category             10127 non-null  object 
 8   Months_on_book            10127 non-null  int64  
 9   Total_Relationship_Count  10127 non-null  int64  
 10  Months_Inactive_12_mon    10127 non-null  int64  
 11  Contacts_Count_12_mon     10127 non-null  int64  
 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: float64(5), int64(9), object(6)
memory usage: 1.5+ MB
```

EDA

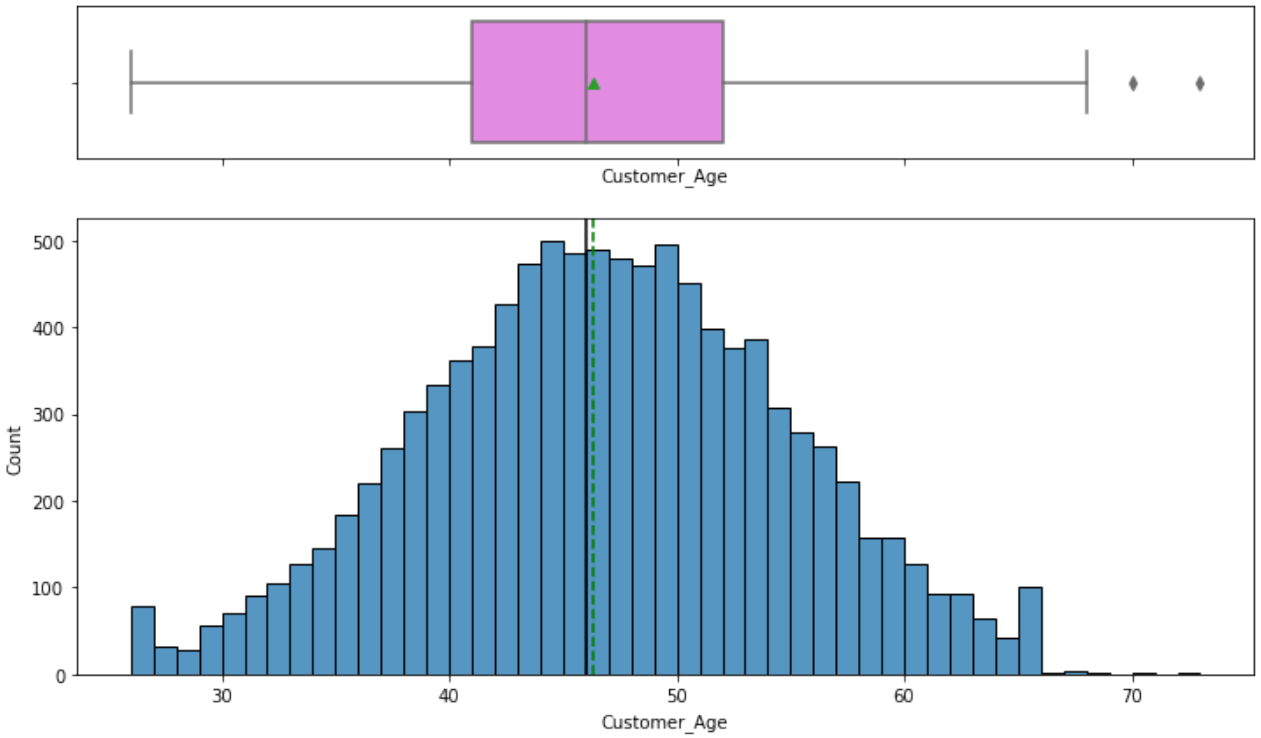
Univariate Analysis

In [56]:
```
# 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 star will indicate the mean value of the
    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
```

Observations on Customer_Age

In [57]:
```
histogram_boxplot(df, "Customer_Age")
```

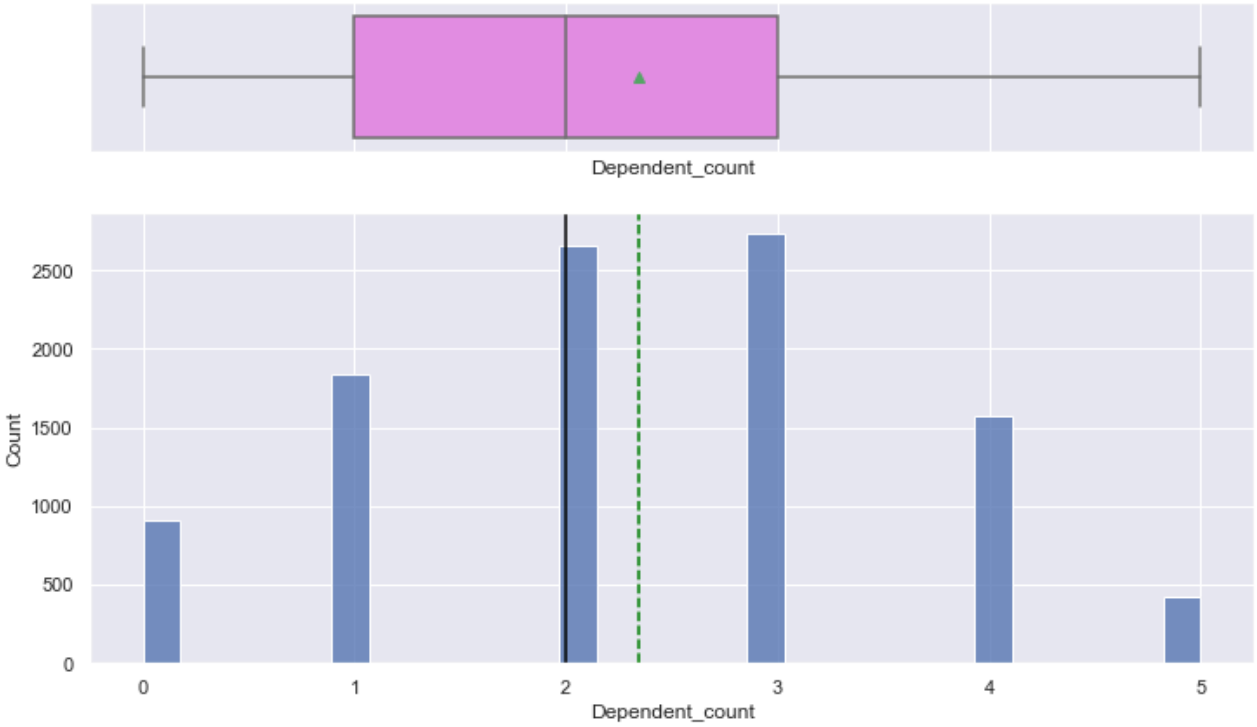


Observations:

- Customer_Age distribution looks approximately normally distributed.
- The boxplot for the Customer_Age column confirms that there are few outliers for this variable
- Customer_Age can be an important variable while targeting customers for the tourism package. We will further explore this in bivariate analysis.

Observations on Dependent_count

In [351…
```
histogram_boxplot(df, "Dependent_count")
```

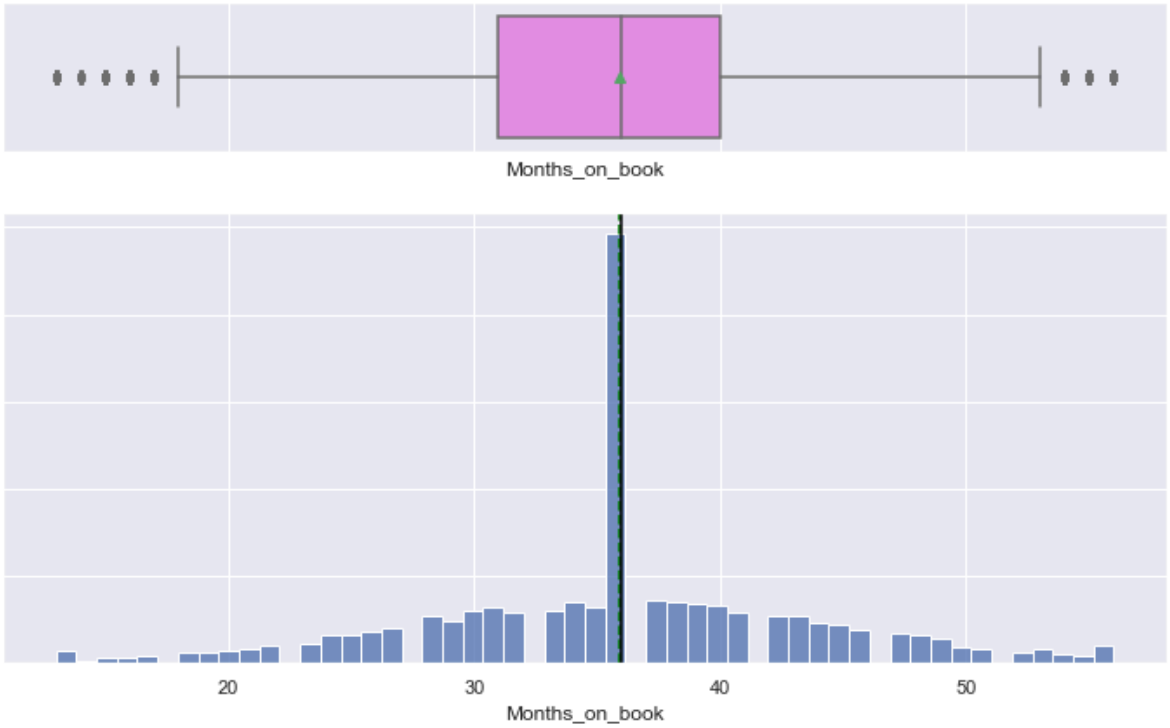


Observations:

- Dependent_count distribution looks approximately normally distributed.
- The boxplot for the Dependent_count column confirms that there are no outliers for this variable

Observations on Months_on_book

In [352…
```
histogram_boxplot(df, "Months_on_book")
```



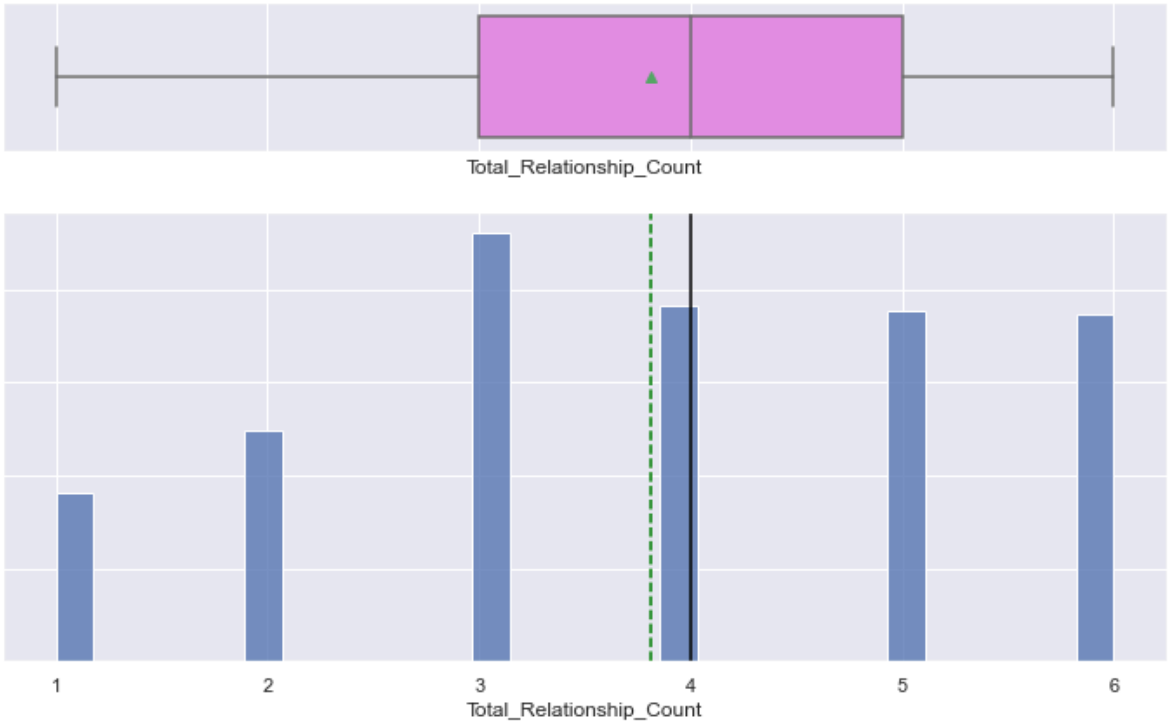
Observations:

- The boxplot for the Months_on_book column confirms that there are few outliers in both the ends for this variable

Observations on Total_Relationship_Count

In [353…

```
histogram_boxplot(df, "Total_Relationship_Count")
```



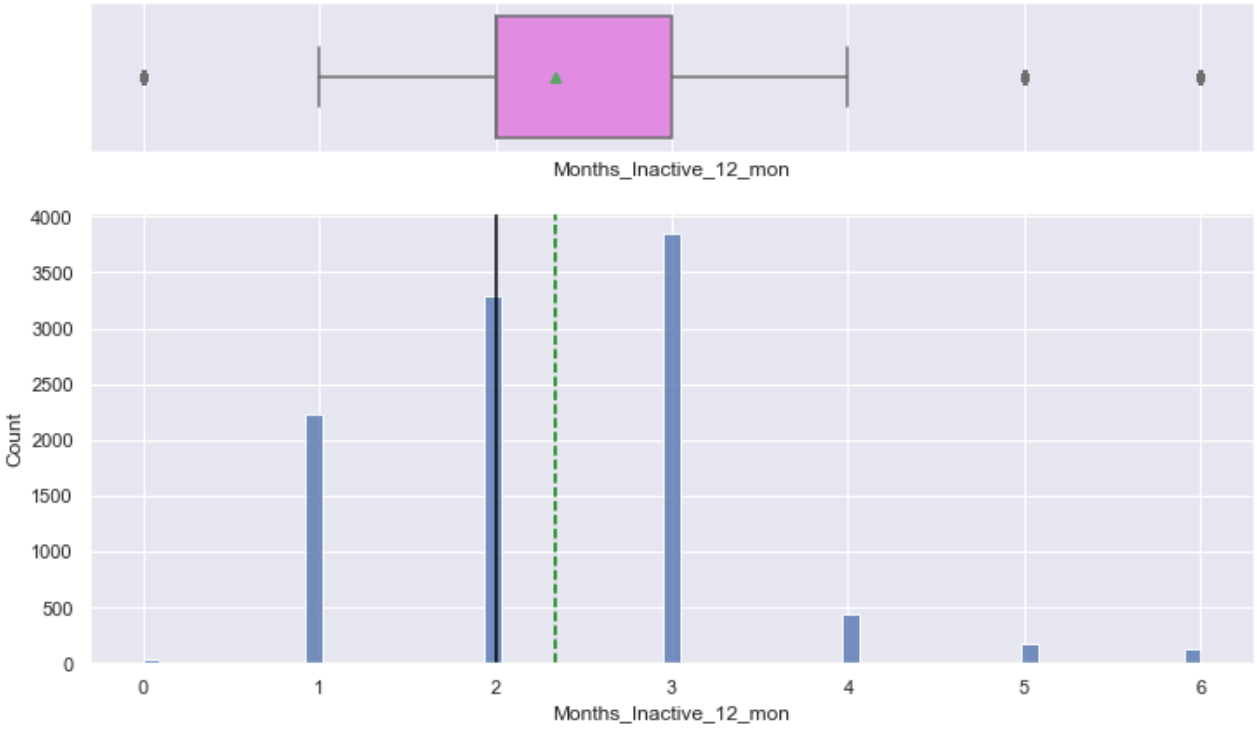
Observations:

- Total_Relationship_Count distribution looks approximately normally distributed.
- The boxplot for the Total_Relationship_Count column confirms that there are no outliers for this variable

Observations on Months_Inactive_12_mon

In [354…

```
histogram_boxplot(df, "Months_Inactive_12_mon")
```



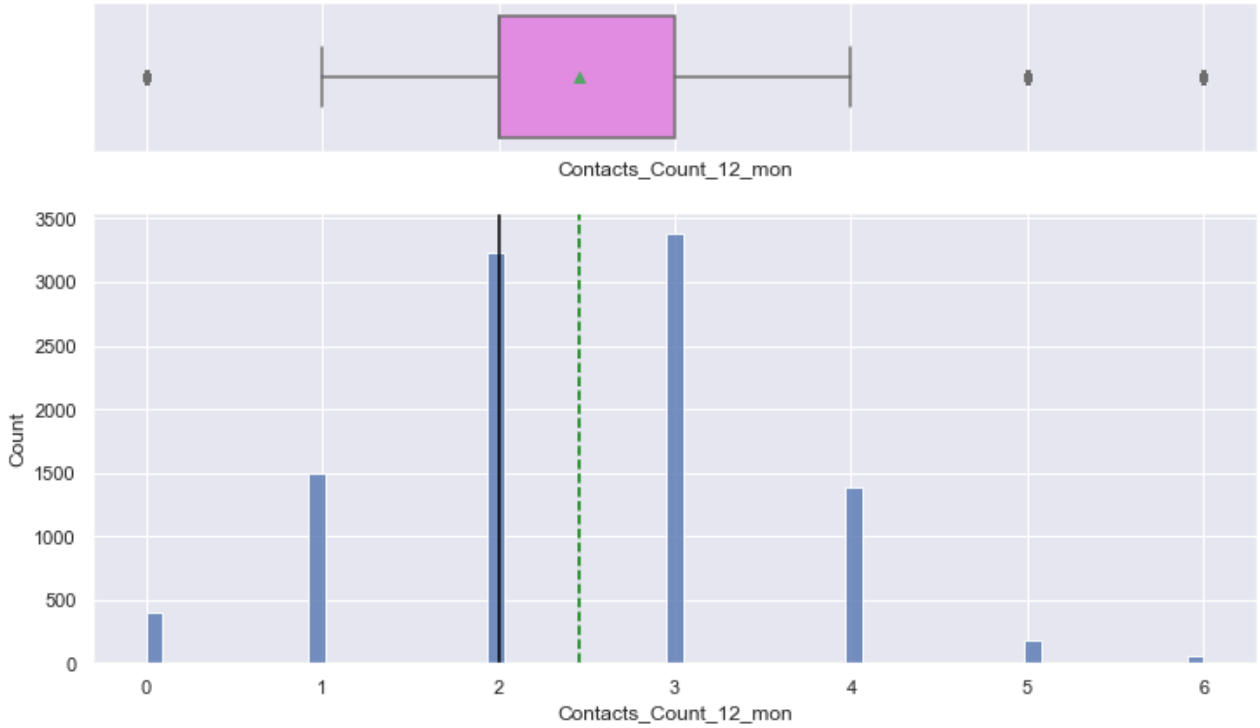
Observations:

- The boxplot for the Months_Inactive_12_mon column confirms that there are few outliers in both the ends for this variable
- Most No. of months inactive in the last 12 months is 3/<3

Observations on Contacts_Count_12_mon

In [355…

```
histogram_boxplot(df, "Contacts_Count_12_mon")
```



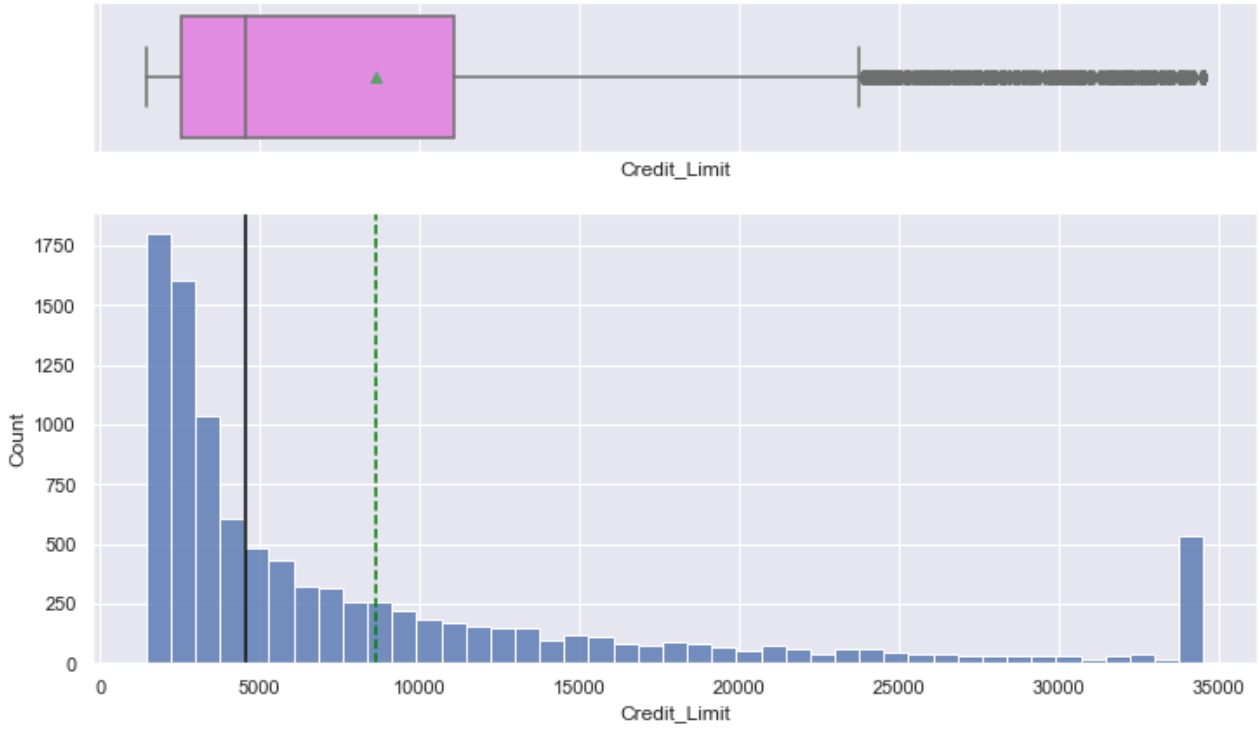
Observations:

- The boxplot for the Contacts_Count_12_mon column confirms that there are few outliers in both the ends for this variable
- Most No. of Contacts between the customer and bank in the last 12 months is 3/<3

Observations on Credit_Limit

In [356…

```
histogram_boxplot(df, "Credit_Limit")
```



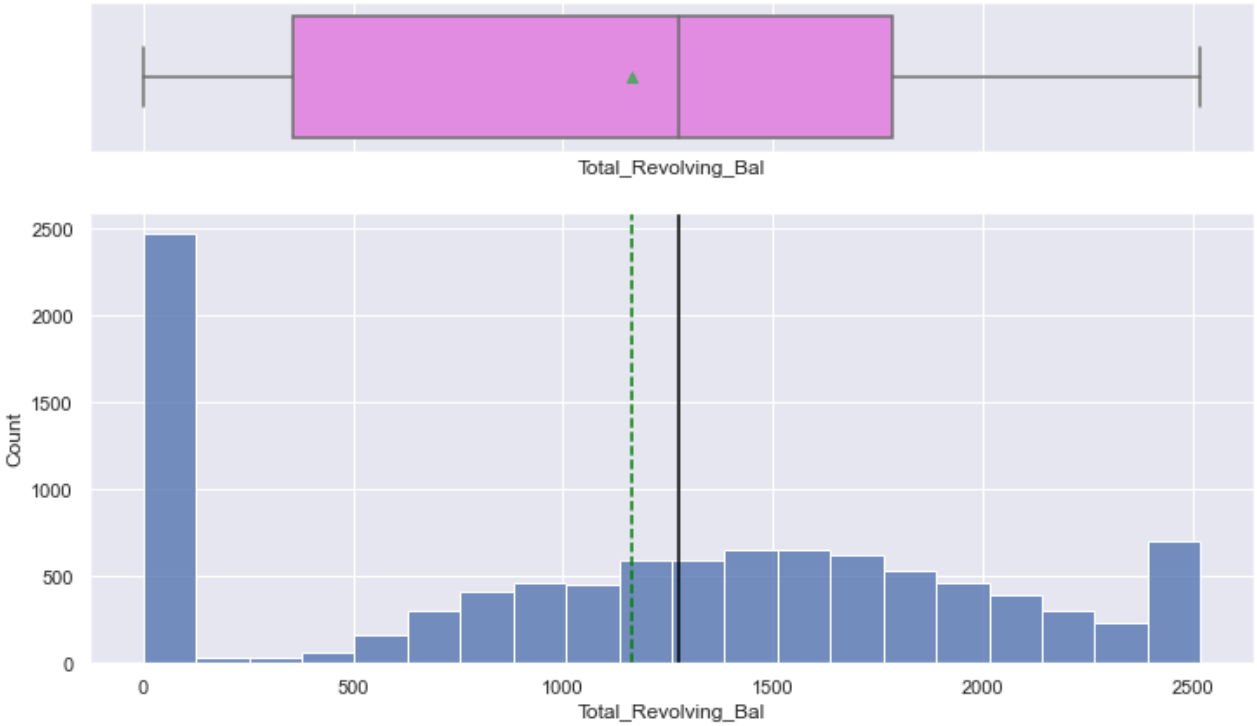
Observations:

- The boxplot for the Credit_Limit column confirms that there are some outliers for this variable
- Average Credit Limit on the Credit Card is less than 10000

Observations on Total_Revolving_Bal

In [268…

```
histogram_boxplot(df, "Total_Revolving_Bal")
```



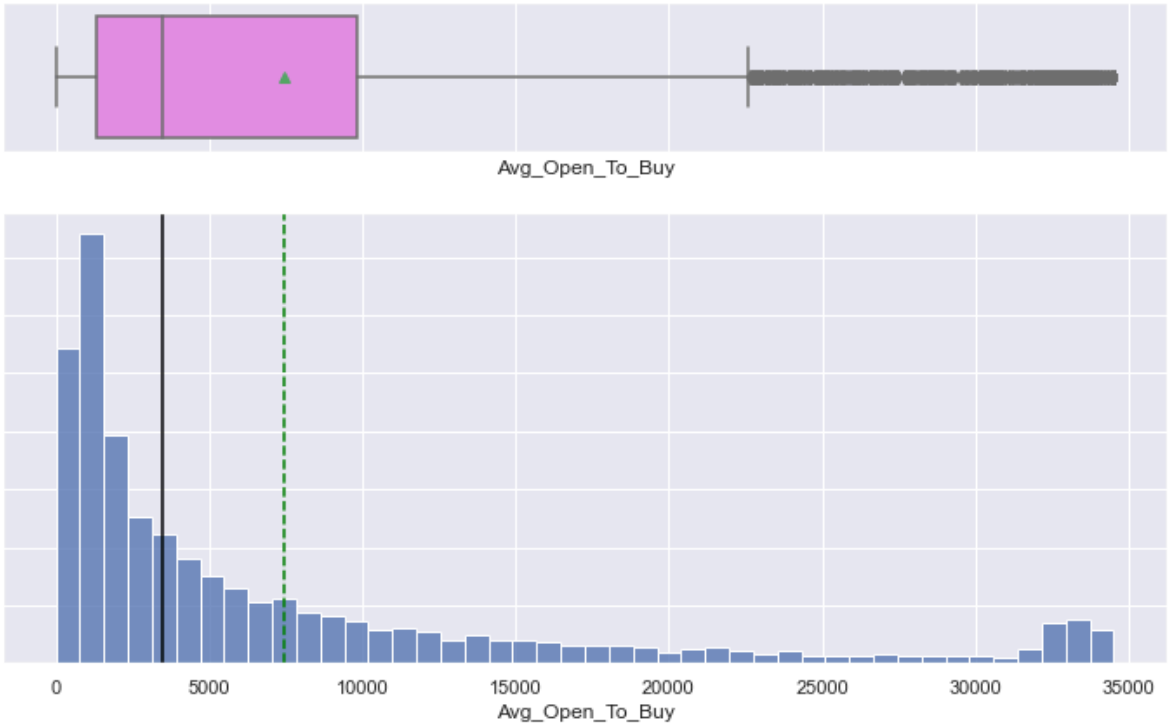
Observations:

- The boxplot for the Total_Revolving_Bal column confirms that there are some outliers for this variable
- Average Revolving Bal is less than 1500
- There are so many 0 Resolving bal in the data

Observations on Avg_Open_To_Buy

In [269…

```
histogram_boxplot(df, "Avg_Open_To_Buy")
```



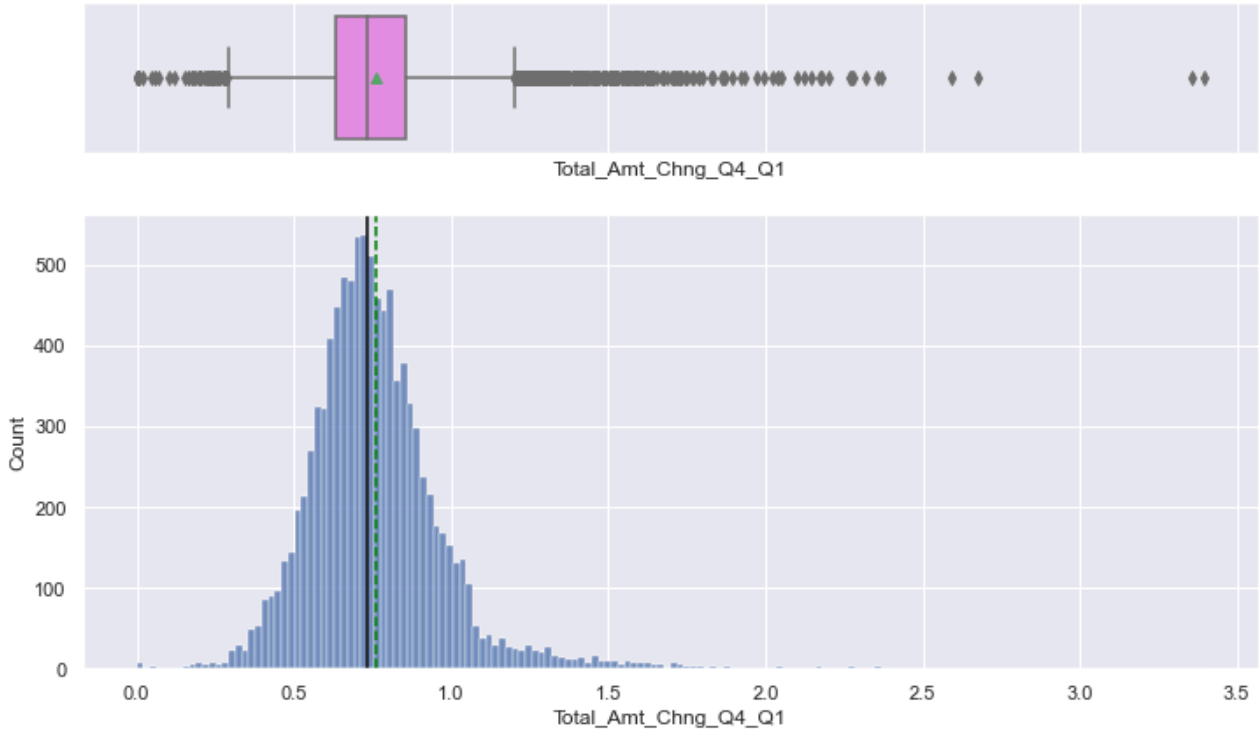
Observations:

- The boxplot for the Avg_Open_To_Buy column confirms that there are some outliers for this variable
- Average open to buy amount is less than 10000

Observations on Total_Amt_Chng_Q4_Q1

In [270…

```
histogram_boxplot(df, "Total_Amt_Chng_Q4_Q1")
```



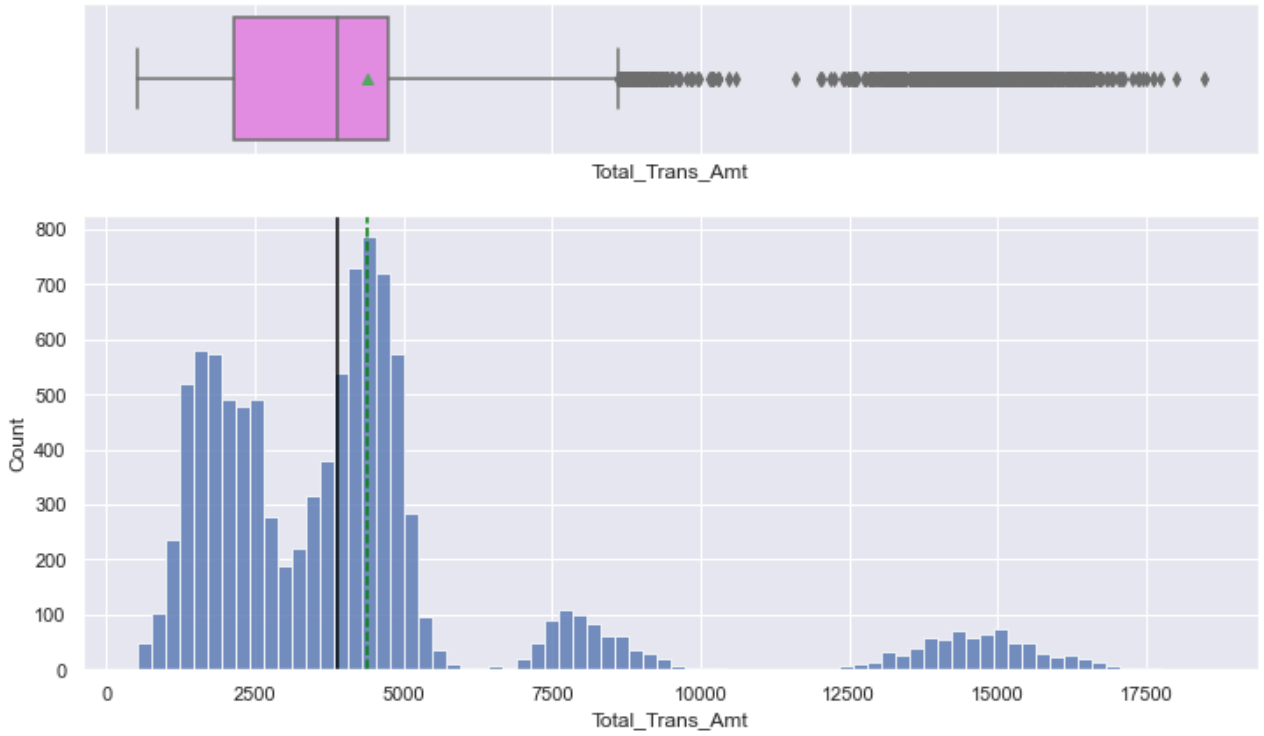
Observations:

- The boxplot for the Total_Amt_Chng_Q4_Q1 column confirms that there are some outliers for this variable

Observations on Total_Trans_Amt

In [271…

```
histogram_boxplot(df, "Total_Trans_Amt")
```



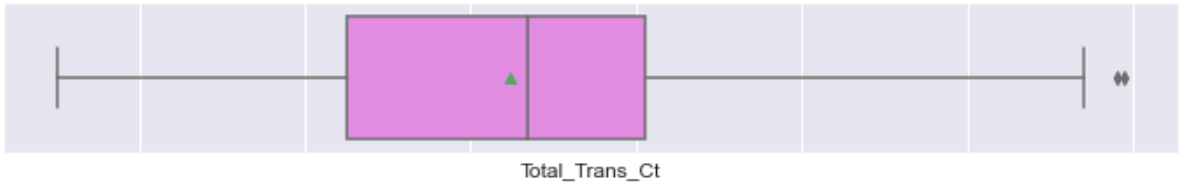
Observations:

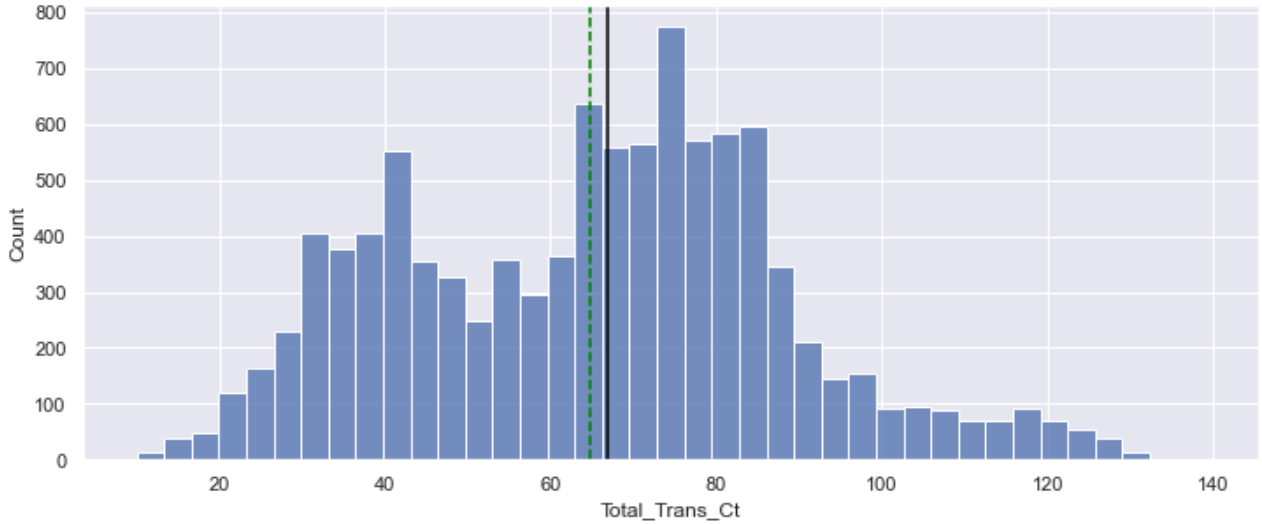
- The boxplot for the Total_Trans_Amt column confirms that there are some outliers for this variable

Observations on Total_Trans_Ct

In [272…

```
histogram_boxplot(df, "Total_Trans_Ct")
```





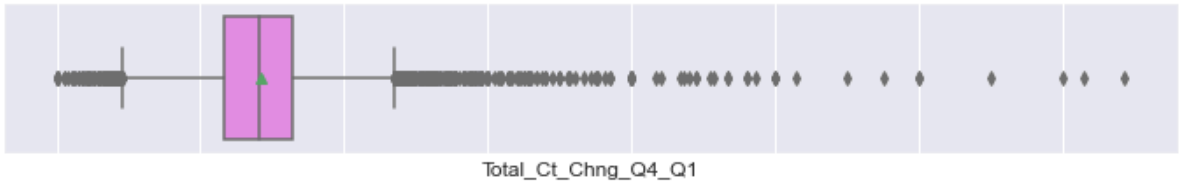
Observations:

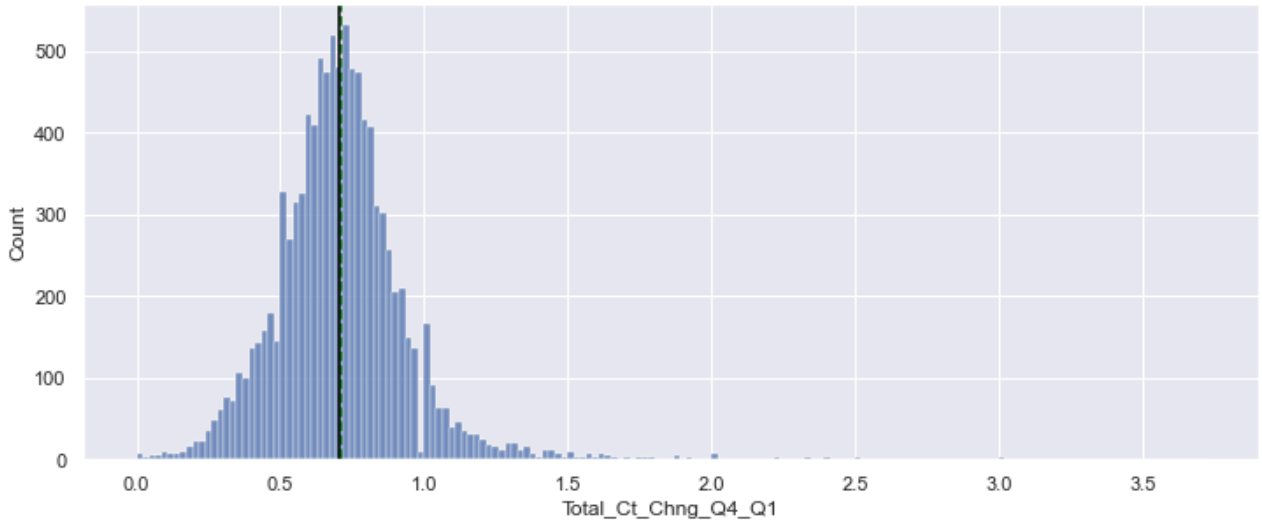
- The boxplot for the Total_Trans_Ct column confirms that there are few outliers for this variable

Observations on Total_Ct_Chng_Q4_Q1

In [273…

```
histogram_boxplot(df, "Total_Ct_Chng_Q4_Q1")
```





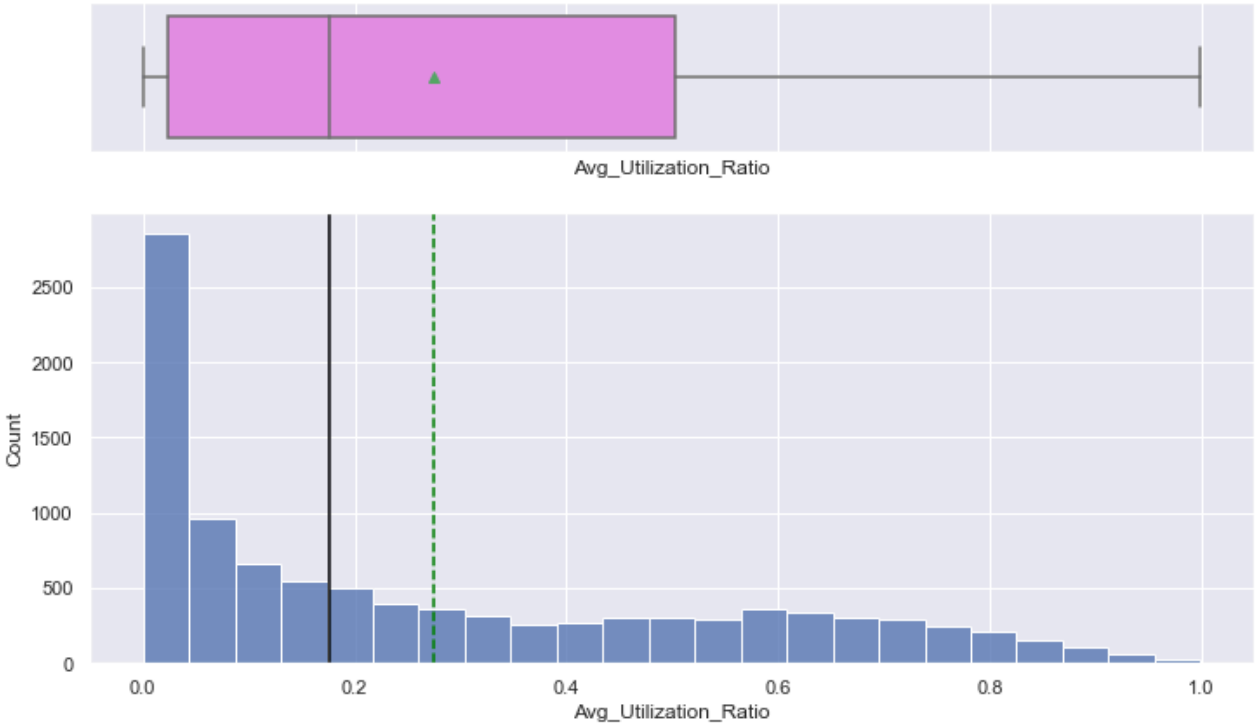
Observations:

- The boxplot for the Total_Ct_Chng_Q4_Q1 column confirms that there are some outliers in both the ends for this variable

Observations on Avg_Utilization_Ratio

In [274…

```
histogram_boxplot(df, "Avg_Utilization_Ratio")
```



Observations:

- The boxplot for the Avg_Utilization_Ratio column confirms that there are no outliers for this variable
- There are so many 0 Avg Utilization Ratio in the data

Let's define a function to create barplots for the categorical variables indicating percentage of each category for that variables.

In [20]:

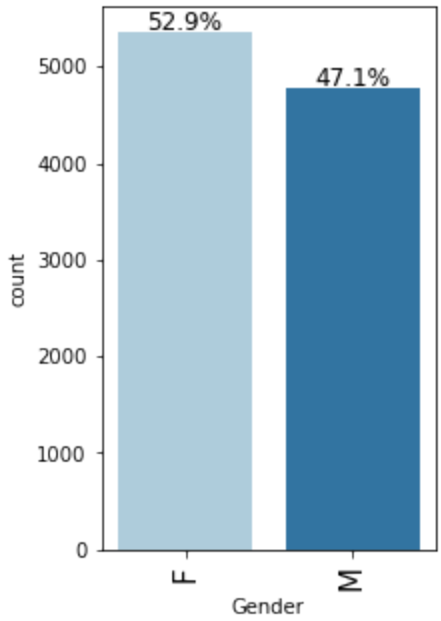
```
# function to create labeled barplots


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 le
    """

    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
```

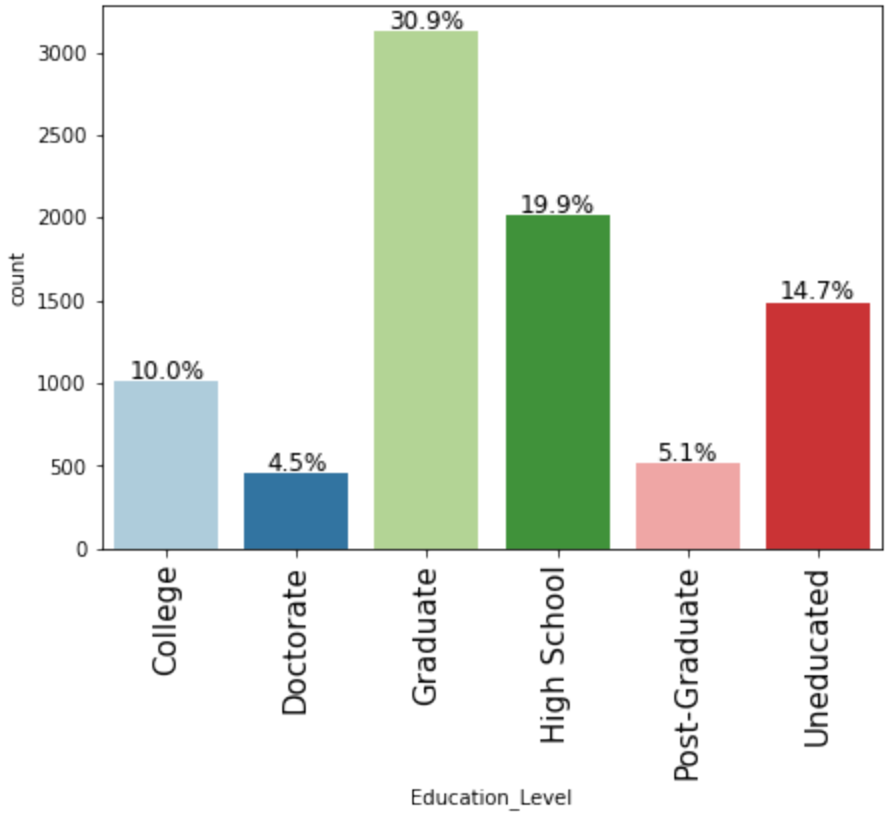
Observations on Category Columns

In [21]:

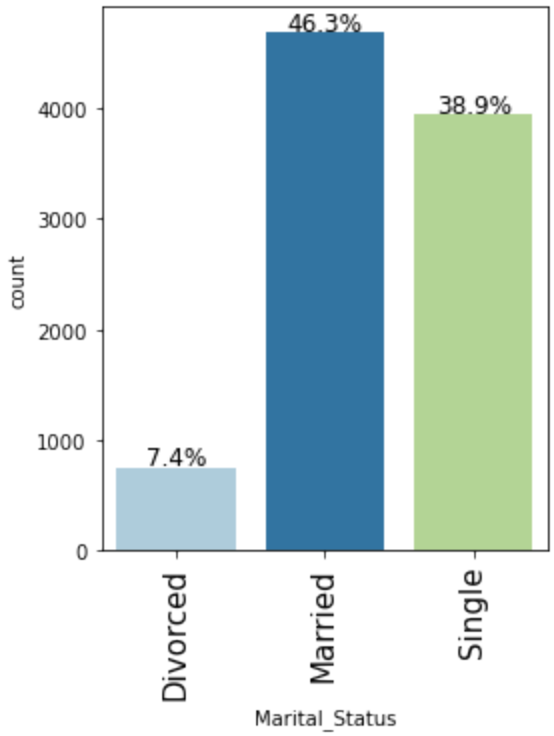
```
labeled_barplot(df, "Gender", perc=True)
```



In [22]:
```
labeled_barplot(df, "Education_Level", perc=True)
```

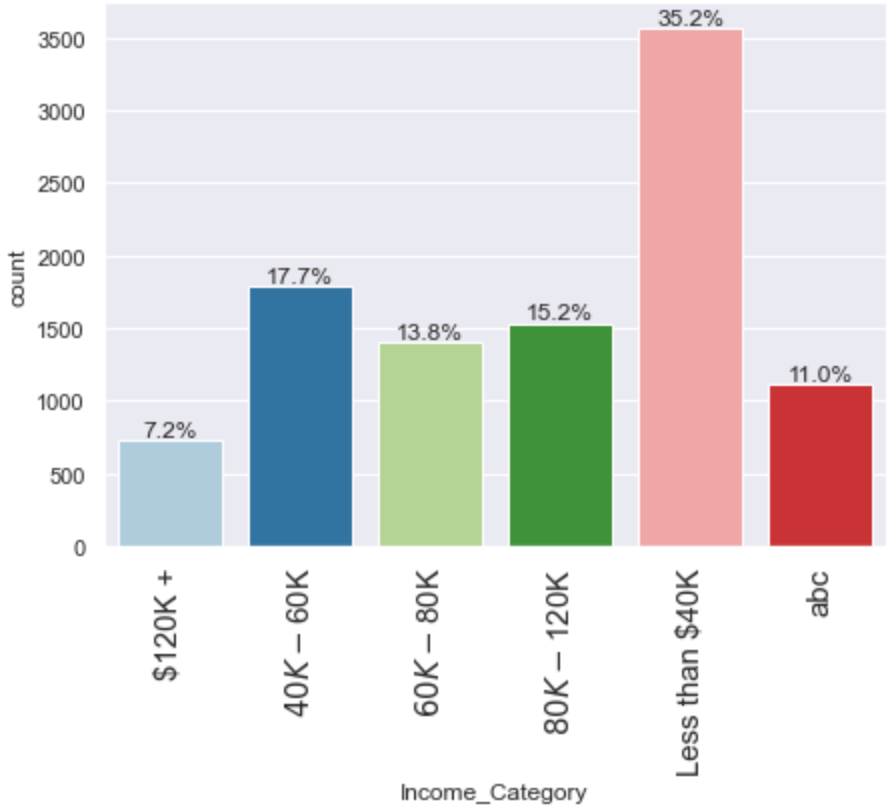


In [23]:
```
labeled_barplot(df, "Marital_Status", perc=True)
```



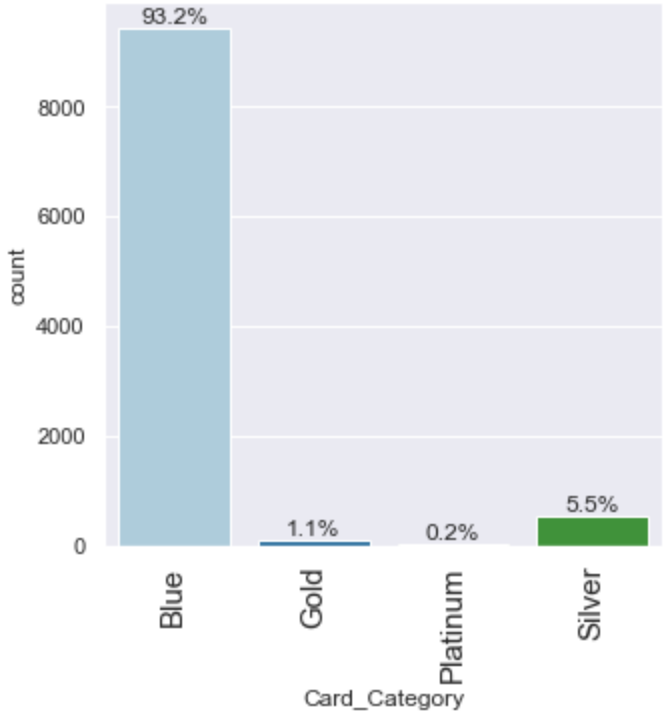
In [279…

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



In [280…

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



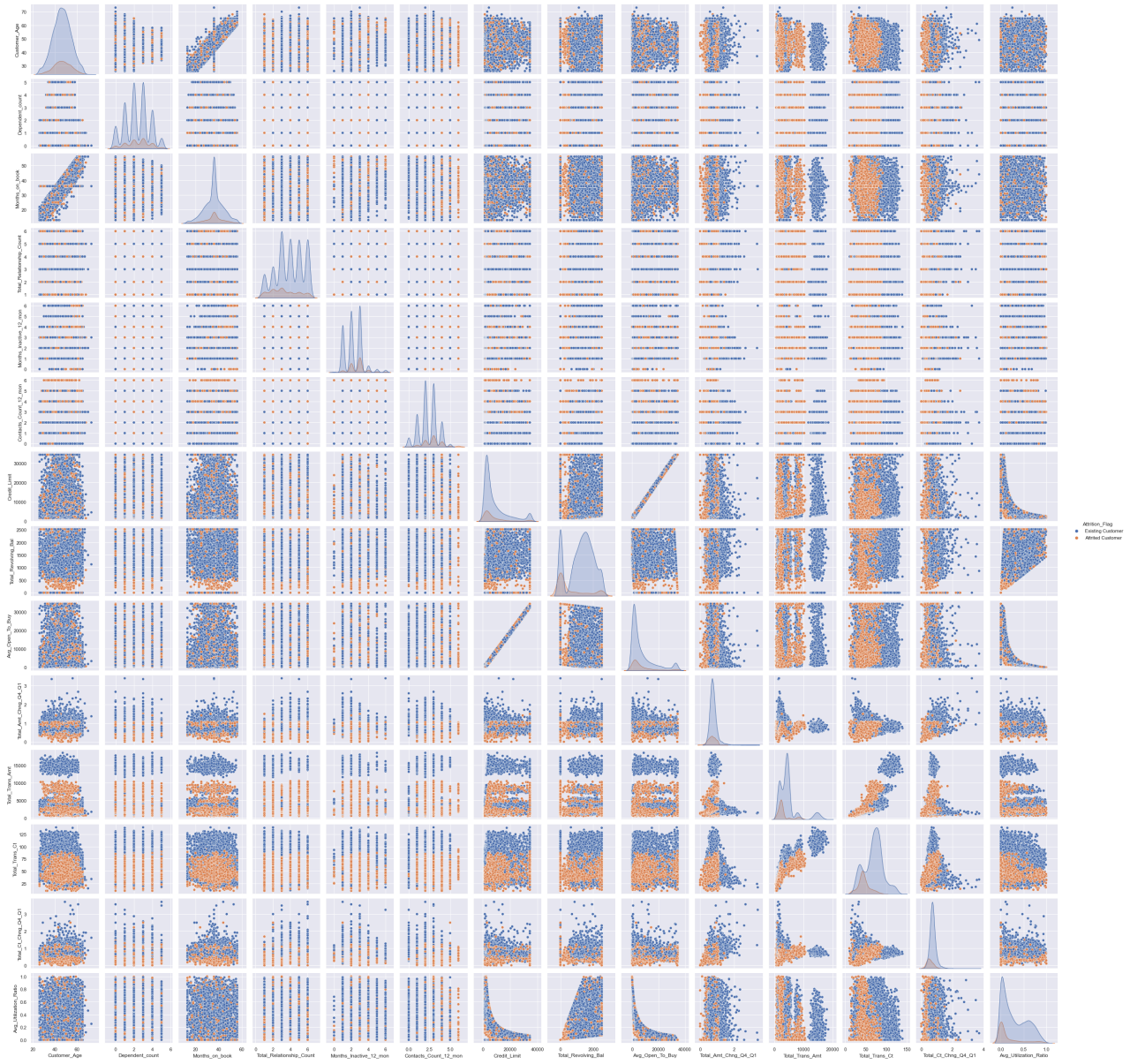
Observations:

- Female account holders are more than male account holders
- Persons who holds Graduation degress are more in the data
- Married people holds accounts than unmarried and divorced
- Less than 40K income people holds 35.2% accounts in the total accounts holders data
- Blue card holders are more than any other category. They holds 93.2% in the total accounts

Bivariate Analysis

In [281…
```
sns.pairplot(data=df,hue='Attrition_Flag')
```

Out[281…
```
<seaborn.axisgrid.PairGrid at 0x7fdc4a65bbb0>
```



Observations:

- There are overlaps i.e. no clear distinction in the distribution of variables for people who have taken the product and did not take the product.
- Let's explore this further with the help of other plots.

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

In [24]:

```
# 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()
```

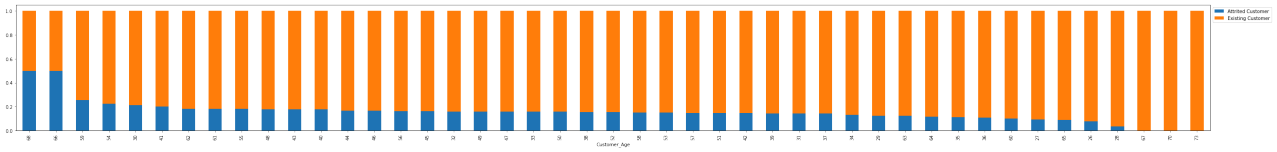
```
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_val
        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 Customer_Age

In [25]:
```
stacked_barplot(df, "Customer_Age", "Attrition_Flag" )
```

```
Attrition_Flag  Attrited Customer  Existing Customer    All
Customer_Age
All                          1627               8500  10127
43                             85                388    473
48                             85                387    472
44                             84                416    500
46                             82                408    490
45                             79                407    486
49                             79                416    495
47                             76                403    479
41                             76                303    379
50                             71                381    452
54                             69                238    307
40                             64                297    361
42                             62                364    426
53                             59                328    387
52                             58                318    376
51                             58                340    398
55                             51                228    279
39                             48                285    333
38                             47                256    303
56                             43                219    262
59                             40                117    157
37                             37                223    260
57                             33                190    223
58                             24                133    157
36                             24                197    221
35                             21                163    184
33                             20                107    127
34                             19                127    146
32                             17                 89    106
61                             17                 76     93
62                             17                 76     93
30                             15                 55     70
31                             13                 78     91
60                             13                114    127
65                              9                 92    101
63                              8                 57     65
29                              7                 49     56
26                              6                 72     78
64                              5                 38     43
```

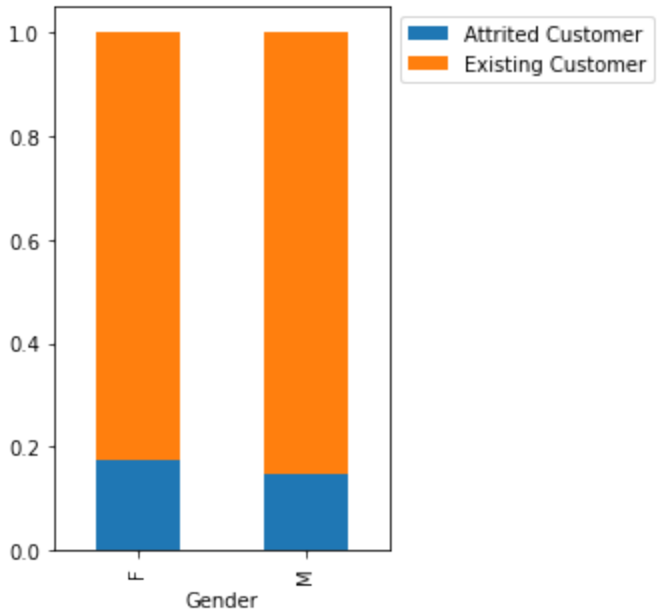
```
27                                3                    29      32
28                                1                    28      29
66                                1                     1       2
68                                1                     1       2
67                                0                     4       4
70                                0                     1       1
73                                0                     1       1
--------------------------------------------------------------------------------
----------------------------------------
```



Attrition_Flag vs Gender

In [26]:
```
stacked_barplot(df, "Gender", "Attrition_Flag" )
```

```
Attrition_Flag  Attrited Customer  Existing Customer    All
Gender
All                          1627               8500  10127
F                             930               4428   5358
M                             697               4072   4769
--------------------------------------------------------------------------------
----------------------------------------
```

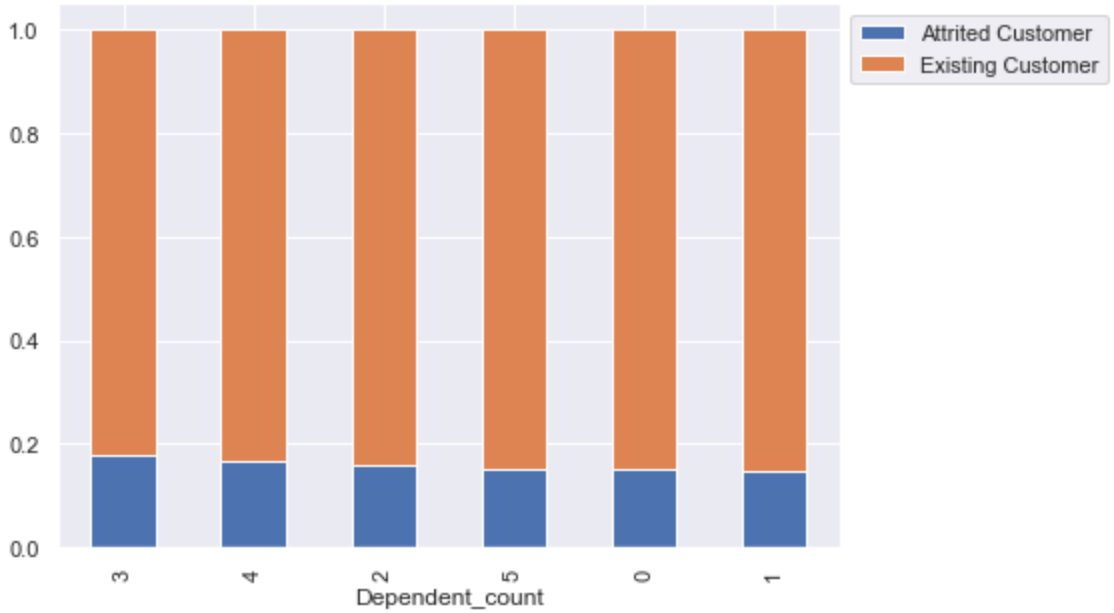


Attrition_Flag vs Dependent_count

In [364…
```
stacked_barplot(df, "Dependent_count", "Attrition_Flag" )
```

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

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

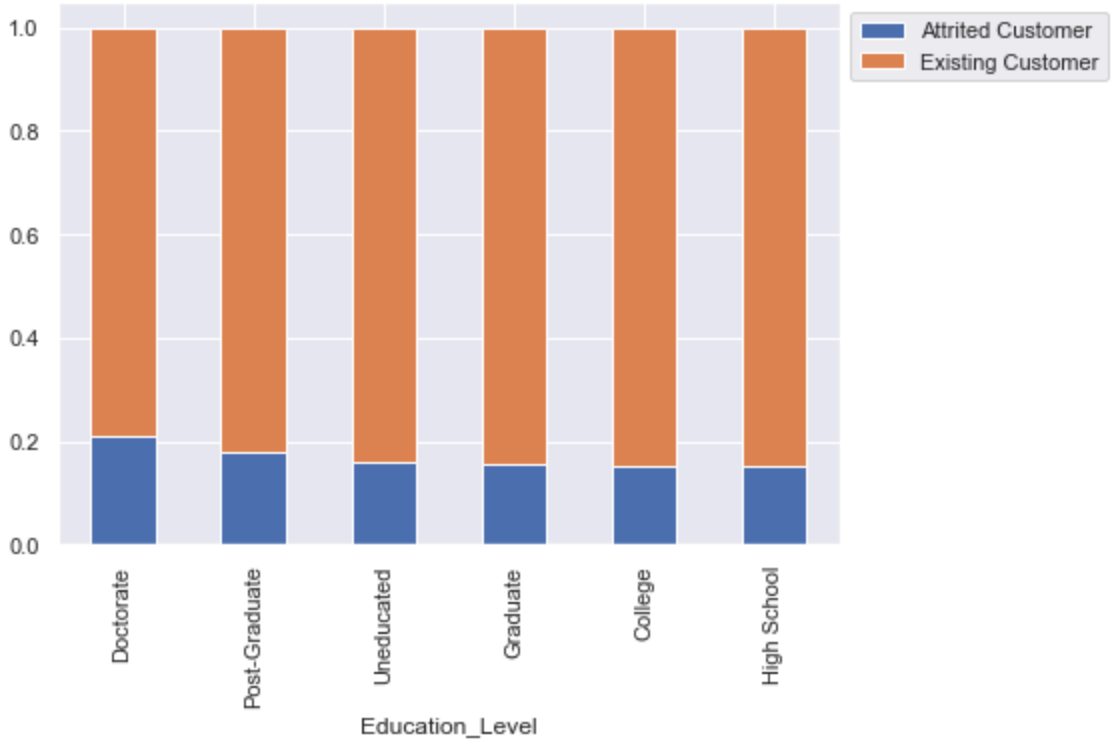


Attrition_Flag vs Education_Level

In [286…

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

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

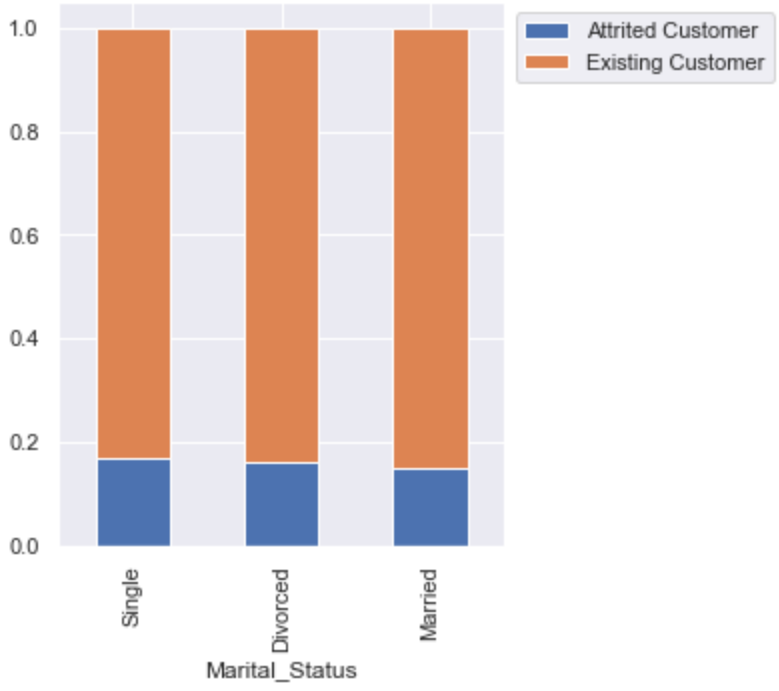


Attrition_Flag vs Marital_Status

In [287…

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

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

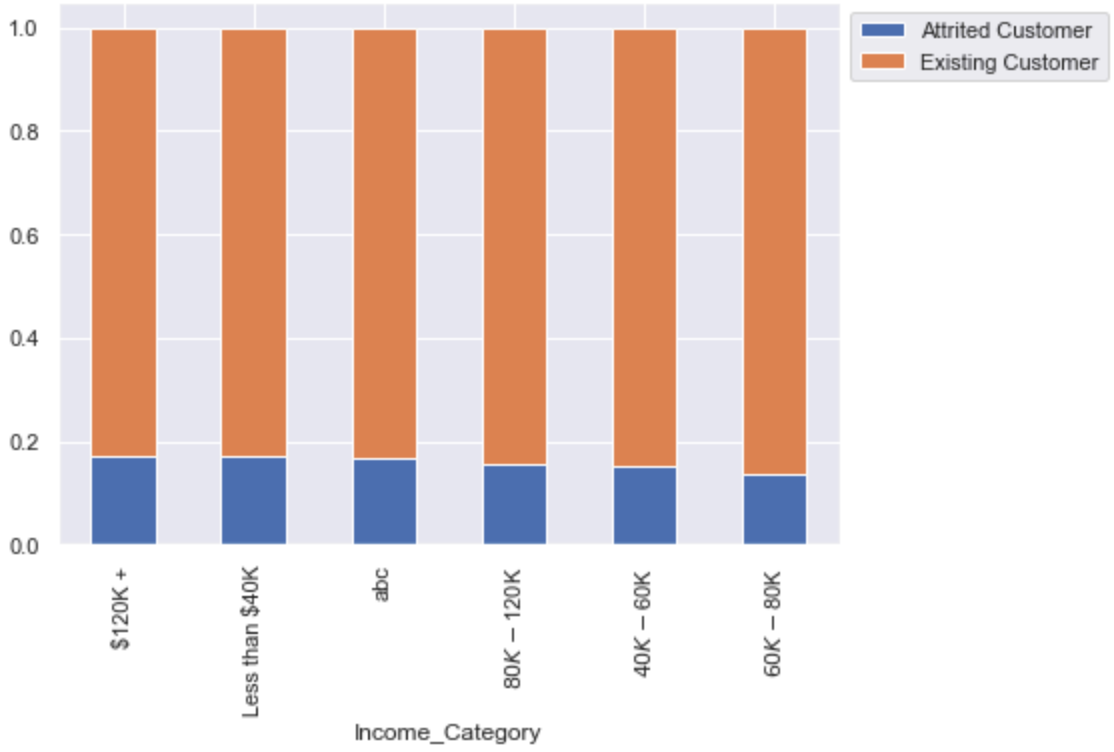


Attrition_Flag vs Income_Category

In [288…

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

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

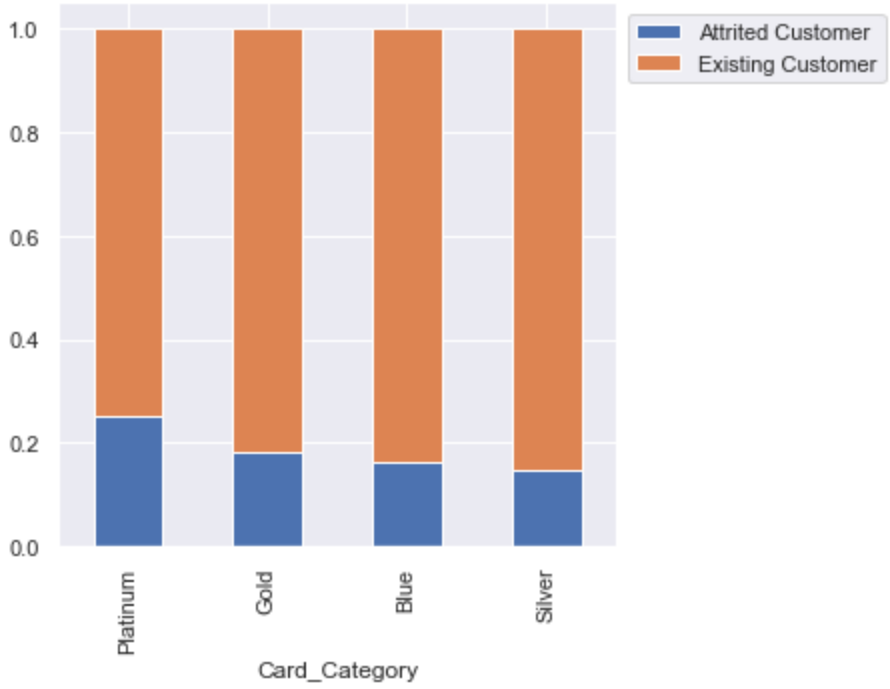


Attrition_Flag vs Card_Category

In [289…

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

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



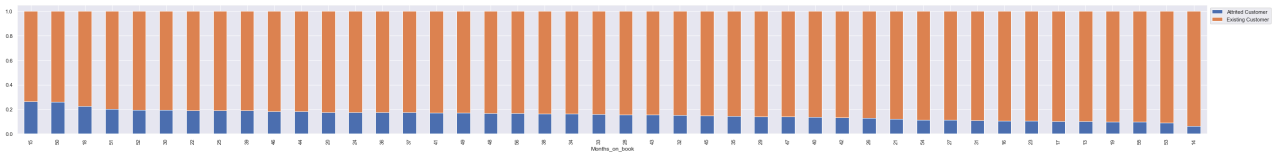
Attrition_Flag vs Months_on_book

In [290…

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

Attrition_Flag	Attrited Customer	Existing Customer	All
Months_on_book			
All	1627	8500	10127
36	430	2033	2463
39	64	277	341
37	62	296	358
30	58	242	300
38	57	290	347
34	57	296	353
41	51	246	297
33	48	257	305
40	45	288	333
35	45	272	317
32	44	245	289
28	43	232	275
44	42	188	230
43	42	231	273
46	36	161	197
42	36	235	271
29	34	207	241
31	34	284	318
45	33	194	227
25	31	134	165
24	28	132	160
48	27	135	162
50	25	71	96
49	24	117	141
26	24	162	186
47	24	147	171
27	23	183	206
22	20	85	105
56	17	86	103
51	16	64	80
18	13	45	58
20	13	61	74

```
52                          12                  50      62
23                          12                 104     116
21                          10                  73      83
15                           9                  25      34
53                           7                  71      78
13                           7                  63      70
19                           6                  57      63
54                           6                  47      53
17                           4                  35      39
55                           4                  38      42
16                           3                  26      29
14                           1                  15      16
------------------------------------------------------------------------------
--------------------------------------
```

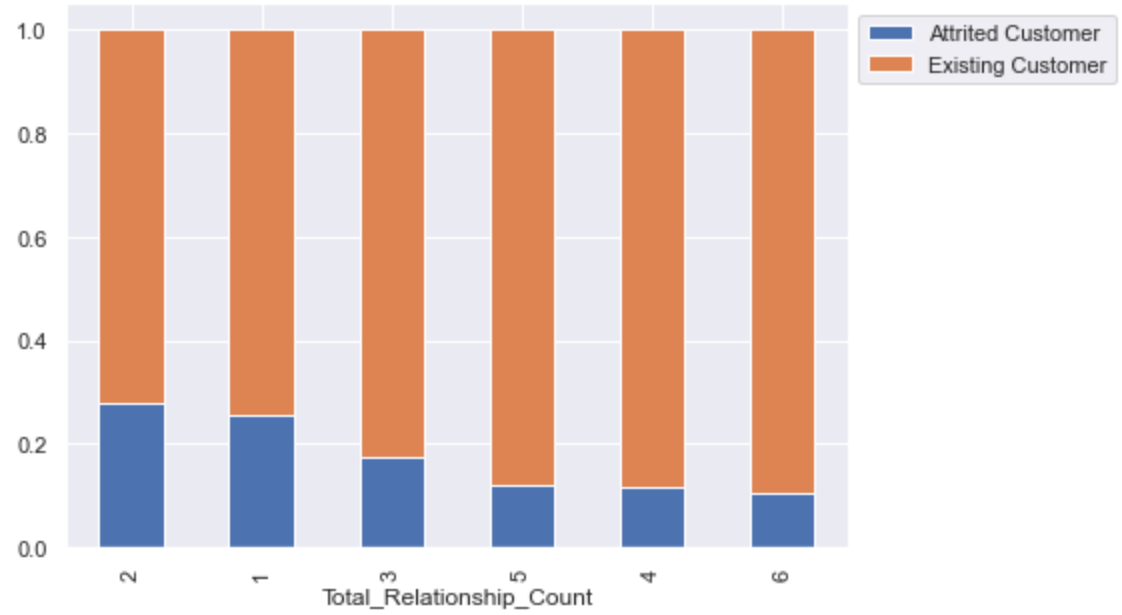


Attrition_Flag vs Total_Relationship_Count

In [291…

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

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

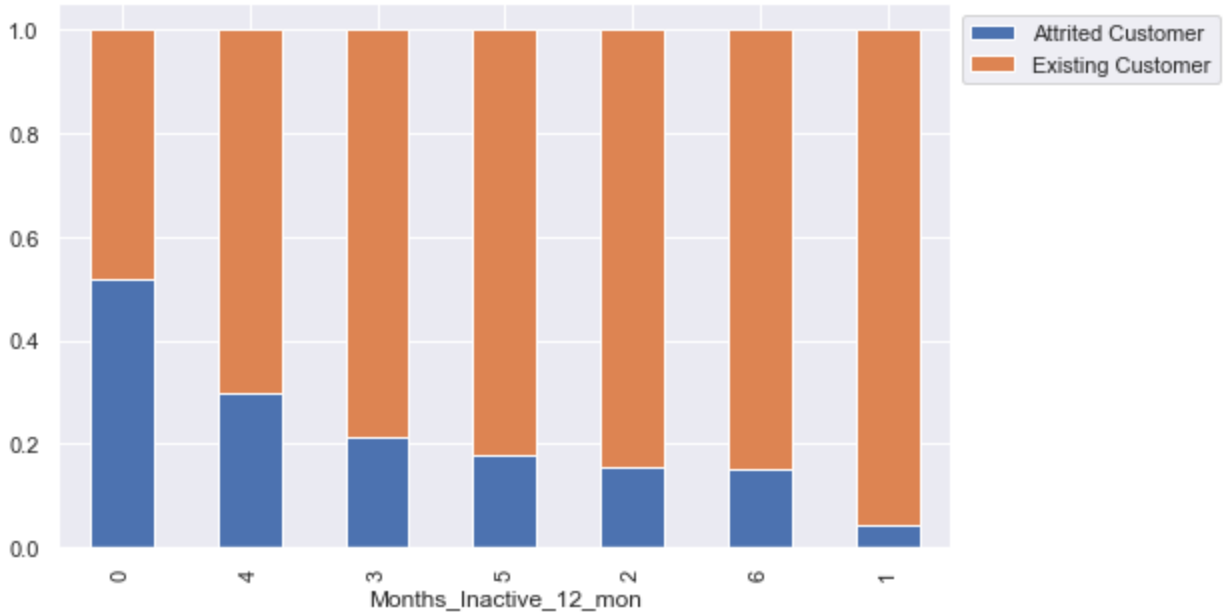


Attrition_Flag vs Months_Inactive_12_mon

In [292…

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

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

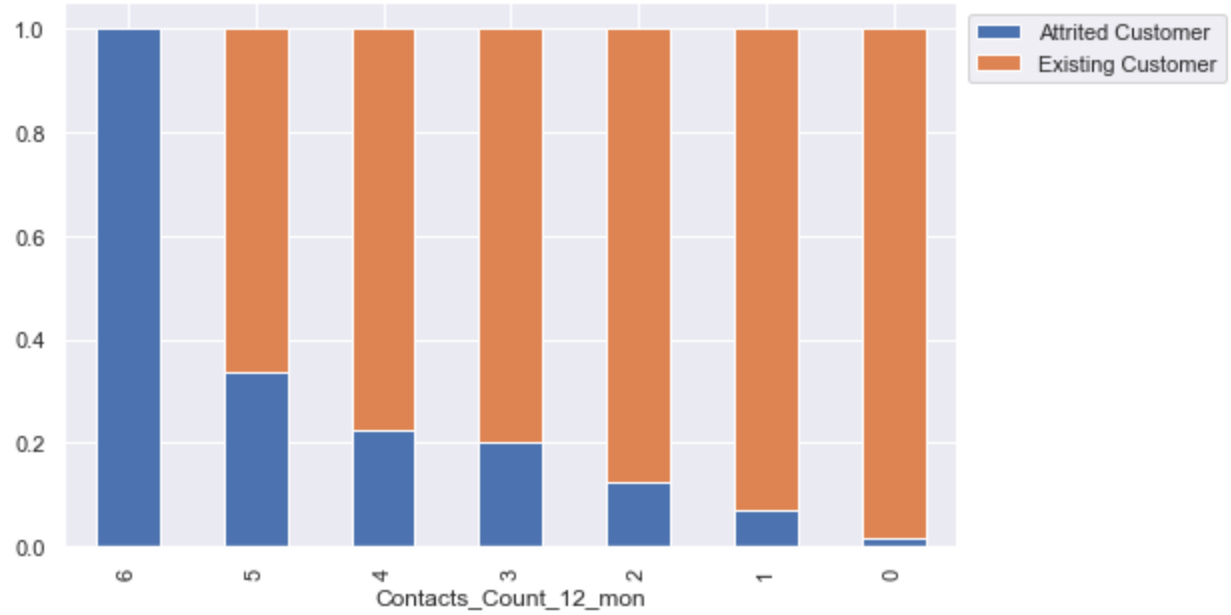


Attrition_Flag vs Contacts_Count_12_mon

In [293…

```
stacked_barplot(df, "Contacts_Count_12_mon", "Attrition_Flag" )
```

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

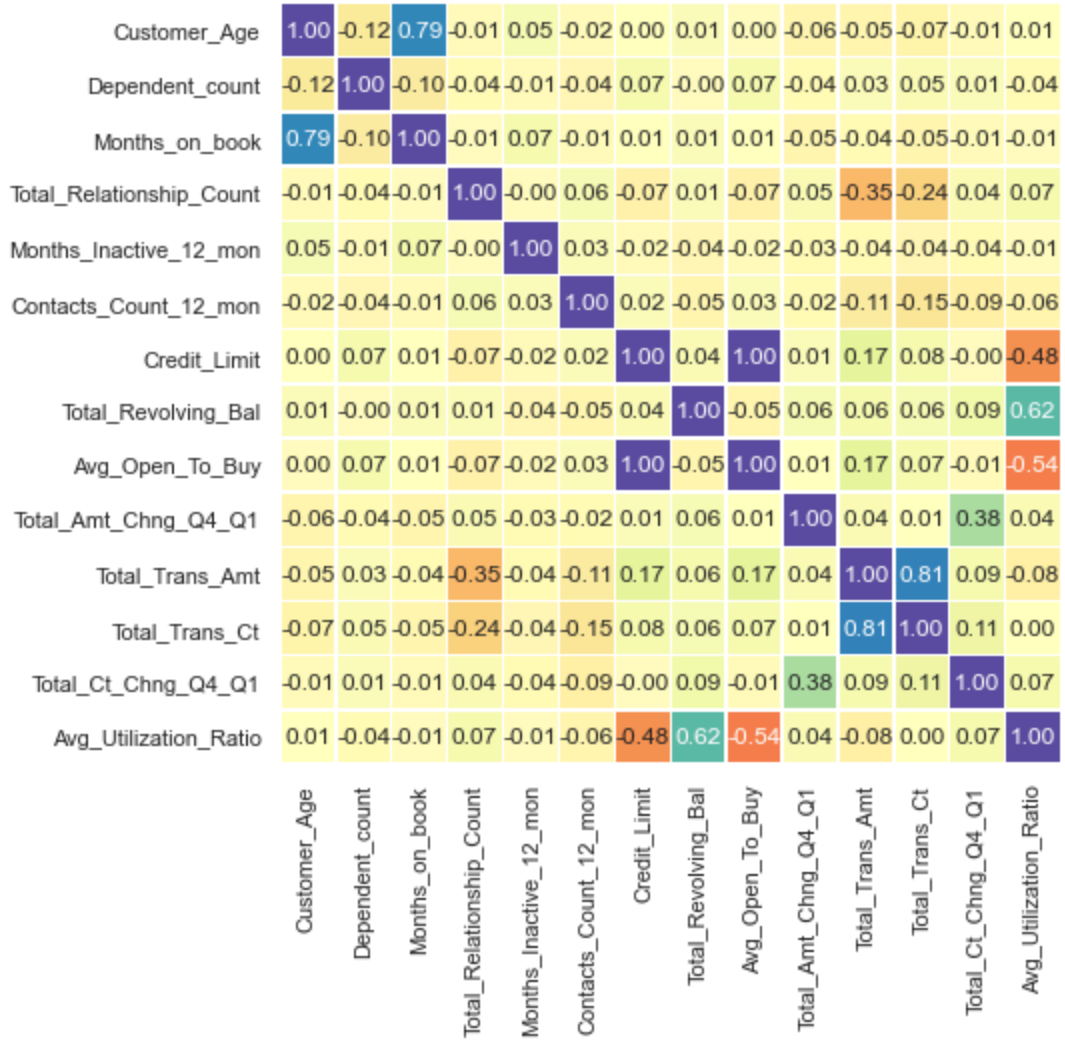


Observations:

- At the age of 68 and 66 Attrited customers are more
- No of months increased in the contacts count in last 12 months increased attrited customers rate also increasing
- Platinum customers are comparetively higher than others in attrited customers
- other columns not making much impacts on the Attrited customers so we need to check further deep to understand this

Correlation Heatmap

In [27]:
```
sns.set(rc={'figure.figsize':(7,7)})
sns.heatmap(df.corr(),
            annot=True,
            linewidths=.5,
            center=0,
            cbar=False,
            cmap="Spectral",
            fmt='0.2f')
plt.show()
```



Observations:

- Months_on_book and Customer_age has high correlation
- Total_Trans_Ct and Total_Trans_Amt has very good correlation has well
- Total_Revolving_Bal and Avg_Utilization_Ratio has third highest correlation
- No other variables have a high correlation among them.

In [365…

```
df.isna().sum()
```

Out[365…

```
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
```

Feature Engineering

In [58]:
```
#Replacing abc values with None value
df.loc[df.Income_Category=='abc','Income_Category']= np.NaN
```

In [59]:
```
df['Income_Category'] = df['Income_Category'].replace(
    {
        'Unknown': 0, 'Less than $40K': 1,
        '$40K - $60K': 2, '$60K - $80K': 3,
        '$80K - $120K': 4, '$120K +': 5
    }
)
df['Attrition_Flag'] = df['Attrition_Flag'].replace(
    {
        'Attrited Customer': 0, 'Existing Customer': 1
    }
)
df['Gender'] = df['Gender'].replace(
    {
        'F': 0, 'M': 1
    }
)
df['Education_Level'] = df['Education_Level'].replace(
    {
        'Unknown': 0, 'Uneducated': 1, 'High School': 2,
        'College': 3, 'Graduate': 4, 'Post-Graduate': 5, 'Doctorate': 6
    }
)
df['Marital_Status'] = df['Marital_Status'].replace(
    {
        'Unknown': 0, 'Single': 1, 'Married': 2, 'Divorced': 3
    }
)
df['Card_Category'] = df['Card_Category'].replace(
    {
        'Blue': 0, 'Silver': 1, 'Gold': 2, 'Platinum': 3
    }
)
```

In [60]:
```
#Separating target variable and other variables
X=df.drop(columns='Attrition_Flag')
Y=df['Attrition_Flag']
```

Split the dataset into train and test sets

In [197…
```
#Splitting the data into train and test sets
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, 19) (2026, 19) (2026, 19)
```

As we saw earlier, our data has missing values. We will impute missing values using median for continuous variables and mode for categorical variables. We will use `SimpleImputer` to do this.

The `SimpleImputer` provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median, or most frequent) of each column in which the missing values are located.

In [198…

```
si2=SimpleImputer(strategy='most_frequent')

mode_imputed_col=['Education_Level','Marital_Status', 'Income_Category']

#Fit and transform the train data
X_train[mode_imputed_col]=si2.fit_transform(X_train[mode_imputed_col])

#Transform the validation data i.e. replace missing values with the mode calcula
X_val[mode_imputed_col]=si2.transform(X_val[mode_imputed_col])

#Transform the test data i.e. replace missing values with the mode calculated us
X_test[mode_imputed_col]=si2.transform(X_test[mode_imputed_col])
```

In [200…

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

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

```
------------------------------
Customer_Age                0
Gender                      0
Dependent_count             0
Education_Level             0
Marital_Status              0
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
------------------------------
Customer_Age                0
Gender                      0
Dependent_count             0
Education_Level             0
Marital_Status              0
Income_Category             0
Card_Category               0
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
```

In [201…

```
#List of columns to create a dummy variables
col_dummy=['Gender', 'Marital_Status']
```

In [202…

```
#Encoding categorical varaibles
X_train=pd.get_dummies(X_train, columns=col_dummy, drop_first=True)
X_val=pd.get_dummies(X_val, columns=col_dummy, drop_first=True)
X_test=pd.get_dummies(X_test, columns=col_dummy, drop_first=True)
```

Let's create dummy variables for string type variables and convert other column types back to float.

In [69]:

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

    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 [68]:

```
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().su
            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")
```

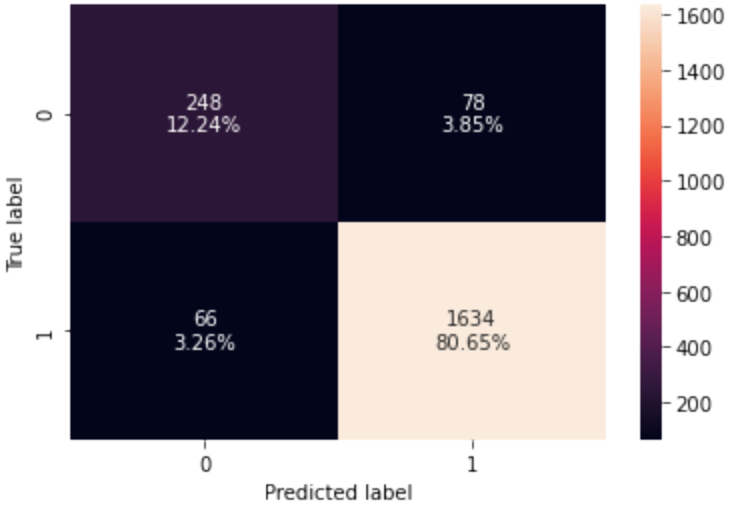
Decision Tree Classifier

In [71]:

```
#Fitting the model
d_tree = DecisionTreeClassifier(random_state=1)
d_tree.fit(X_train,y_train)

#Calculating different metrics
d_tree_model_train_perf=model_performance_classification_sklearn(d_tree, X_train
print("Training performance:\n", d_tree_model_train_perf)
d_tree_model_val_perf=model_performance_classification_sklearn(d_tree, X_val,y_v
```

```
print("Validation performance:\n", d_tree_model_val_perf)
#Creating confusion matrix
confusion_matrix_sklearn(d_tree,X_val,y_val)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.928924  0.961176   0.954439  0.957796
```



- The model is slightly overfitting the training data as training accuracy/recall/precision is higher than the validation data accuracy/recall/precision

Cost Complexity Pruning

Let's try pruning the tree and see if the performance improves.

In [72]:
```
path = d_tree.cost_complexity_pruning_path(X_train, y_train)

ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

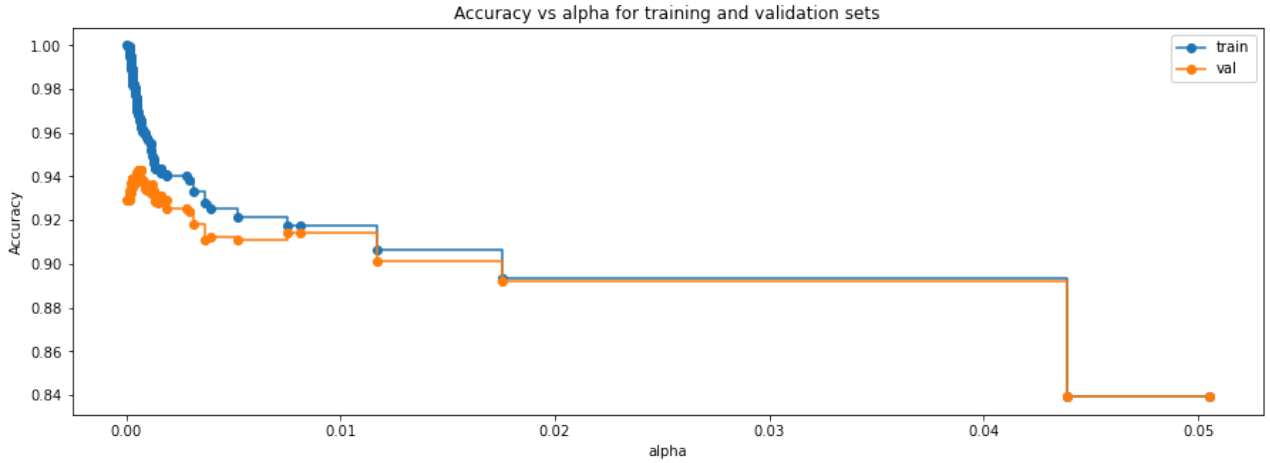
In [73]:
```
clfs_list = []

for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs_list.append(clf)

print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(clfs_l
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.05048682913535829
```

In [74]:
```
#Fitting model for each value of alpha and saving the train acc in a list
acc_train=[]
for clf in clfs_list:
    pred_train=clf.predict(X_train)
    values_train=accuracy_score(y_train,pred_train)
    acc_train.append(values_train)
```

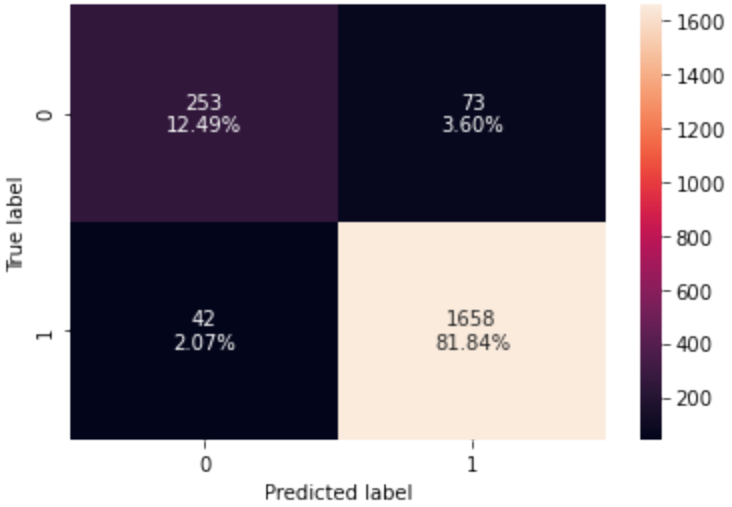
In [76]:
```
#Fitting model for each value of alpha and saving the test acc in a list
acc_val=[]
for clf in clfs_list:
    pred_val=clf.predict(X_val)
    values_val=accuracy_score(y_val,pred_val)
    acc_val.append(values_val)
```

In [77]:
```
#Plotting the graph for Acc VS alpha
fig, ax = plt.subplots(figsize=(15,5))
ax.set_xlabel("alpha")
ax.set_ylabel("Accuracy")
ax.set_title("Accuracy vs alpha for training and validation sets")
ax.plot(ccp_alphas, acc_train, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, acc_val, marker='o', label="val",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



In [78]:
```
#Creating the model where we get highest test accuracy
index_best_pruned_model = np.argmax(acc_val)

pruned_dtree_model = clfs_list[index_best_pruned_model]

#Calculating different metrics
pruned_dtree_model_train_perf=model_performance_classification_sklearn(pruned_dt
print("Training performance:\n", pruned_dtree_model_train_perf)
pruned_dtree_model_val_perf=model_performance_classification_sklearn(pruned_dtre
print("Validation performance:\n", pruned_dtree_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(pruned_dtree_model,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.968066  0.984703   0.977419  0.981047
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.943238  0.975294   0.957828  0.966482
```



- Overfitting issue seems fixed now but overrall performance has been reduced on both validation and training sets
- Let's try hyperparameter tuning, with class weights to compensate for the imbalanced data, and see if the model performance improves.

Hyperparameter Tuning

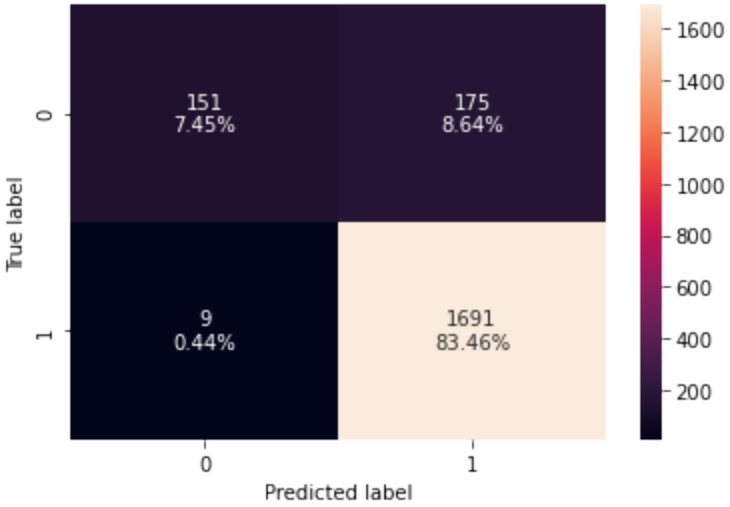
In [79]:
```
#Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight={0:0.18,1:0.72},random_sta

# Grid of parameters to choose from
parameters = {'max_depth': np.arange(2,30),
              'min_samples_leaf': [1, 2, 5, 7, 10],
              'max_leaf_nodes' : [2, 3, 5, 10,15],
              'min_impurity_decrease': [0.0001,0.001,0.01,0.1]
             }

# Type of scoring used to compare parameter combinations
scorer = make_scorer(accuracy_score)

# Run the grid search
grid_obj = GridSearchCV(dtree_estimator, parameters, scoring=scorer,n_jobs=-1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
dtree_estimator = grid_obj.best_estimator_

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

Out[79]:
```
DecisionTreeClassifier(class_weight={0: 0.18, 1: 0.72}, max_depth=6,
                       max_leaf_nodes=15, min_impurity_decrease=0.0001,
                       random_state=1)
```

In [80]:
```
#Calculating different metrics
dtree_estimator_model_train_perf=model_performance_classification_sklearn(dtree_
print("Training performance:\n", dtree_estimator_model_train_perf)
dtree_estimator_model_val_perf=model_performance_classification_sklearn(dtree_es
print("Validation performance:\n", dtree_estimator_model_val_perf)
```

```
#Creating confusion matrix
confusion_matrix_sklearn(dtree_estimator,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.923951  0.995489   0.920399  0.956473
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.909181  0.994706   0.906217  0.948402
```



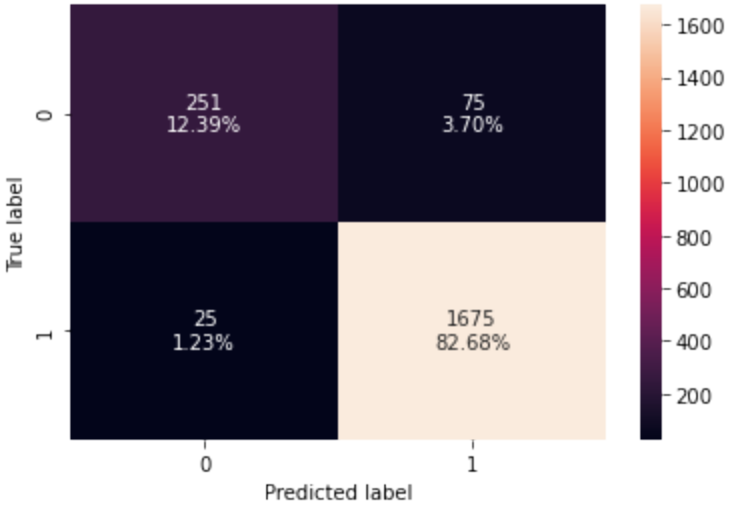
- Overfitting issue seems fixed now but overrall performance has been reduced on both validation and training sets
- Let's try Random Forest Classifier and see if the model performance improves.

Random Forest Classifier

In [81]:

```
#Fitting the model
rf_estimator = RandomForestClassifier(random_state=1)
rf_estimator.fit(X_train,y_train)

#Calculating different metrics
rf_estimator_model_train_perf=model_performance_classification_sklearn(rf_estima
print("Training performance:\n",rf_estimator_model_train_perf)
rf_estimator_model_val_perf=model_performance_classification_sklearn(rf_estimato
print("Validation performance:\n",rf_estimator_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(rf_estimator,X_val,y_val)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.950642  0.985294   0.957143  0.971014
```



- The model is slightly overfitting the training data as training accuracy/recall/precision is higher than the validation accuracy/recall/precision
- We'll try to reduce overfitting and improve recall by hyperparameter tuning.

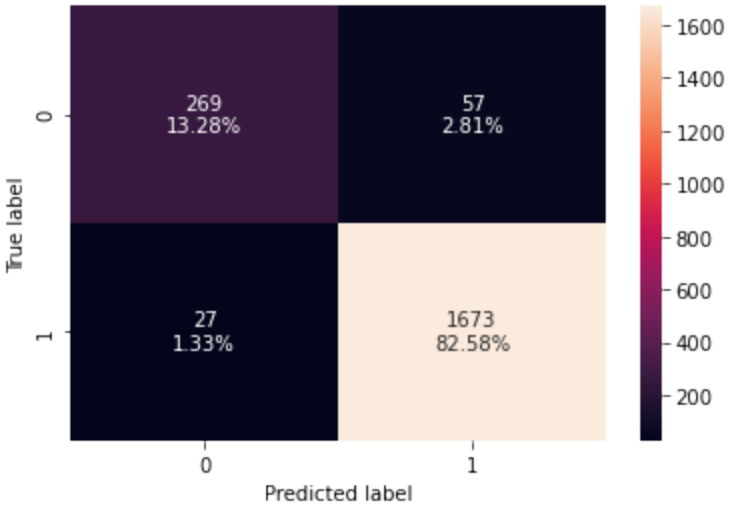
Hyperparameter Tuning

In [82]:
```
# Choose the type of classifier.
rf_tuned = RandomForestClassifier(class_weight={0:0.18,1:0.82},random_state=1,oc

parameters = {
                'max_depth': list(np.arange(5,30,5)) + [None],
                'max_features': ['sqrt','log2',None],
                'min_samples_leaf': np.arange(1,15,5),
                'min_samples_split': np.arange(2, 20, 5),
                'n_estimators': np.arange(10,110,10)}


# Run the grid search
grid_obj = GridSearchCV(rf_tuned, parameters, scoring='accuracy',cv=5,n_jobs=-1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
rf_tuned = grid_obj.best_estimator_

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

Out[82]:
```
RandomForestClassifier(class_weight={0: 0.18, 1: 0.82}, max_depth=15,
                       max_features=None, oob_score=True, random_state=1)
```

In [83]:
```
#Calculating different metrics
rf_tuned_model_train_perf=model_performance_classification_sklearn(rf_tuned, X_t
print("Training performance:\n",rf_tuned_model_train_perf)
rf_tuned_model_val_perf=model_performance_classification_sklearn(rf_tuned, X_val
print("Validation performance:\n",rf_tuned_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(rf_tuned,X_val,y_val)
```

```
Training performance:
    Accuracy  Recall  Precision        F1
0  0.998683     1.0   0.998434  0.999216
Validation performance:
    Accuracy    Recall  Precision       F1
0  0.958539  0.984118   0.967052  0.97551
```



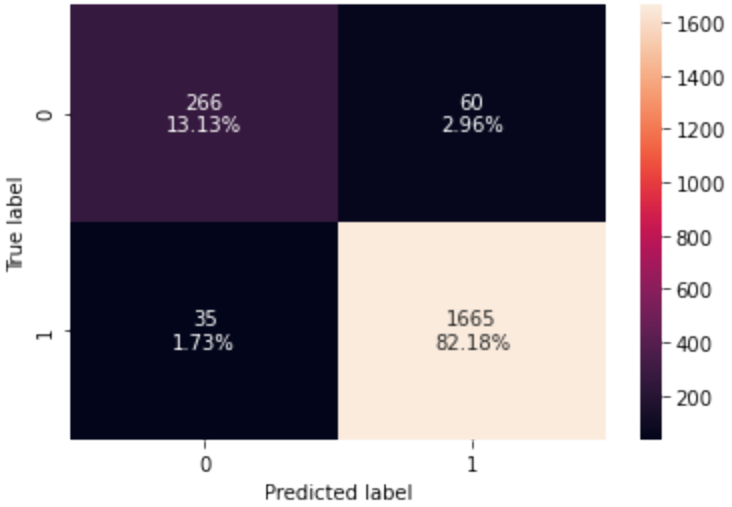
- The model is slightly overfitting the training data as training accuracy/recall/precision is higher than the validation accuracy/recall/precision

Bagging Classifier

In [84]:

```
#Fitting the model
bagging_classifier = BaggingClassifier(random_state=1)
bagging_classifier.fit(X_train,y_train)

#Calculating different metrics
bagging_classifier_model_train_perf=model_performance_classification_sklearn(bag
print("Training performance:\n",bagging_classifier_model_train_perf)
bagging_classifier_model_val_perf=model_performance_classification_sklearn(baggi
print("Validation performance:\n",bagging_classifier_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(bagging_classifier,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.997202  0.997647   0.999018  0.998332
Validation performance:
    Accuracy    Recall  Precision        F1
0   0.95311  0.979412   0.965217  0.972263
```



- The model is slightly overfitting the training data as training accuracy/recall/precision is higher than the validation accuracy/recall/precision
- We'll try to reduce overfitting and improve accuracy by hyperparameter tuning.

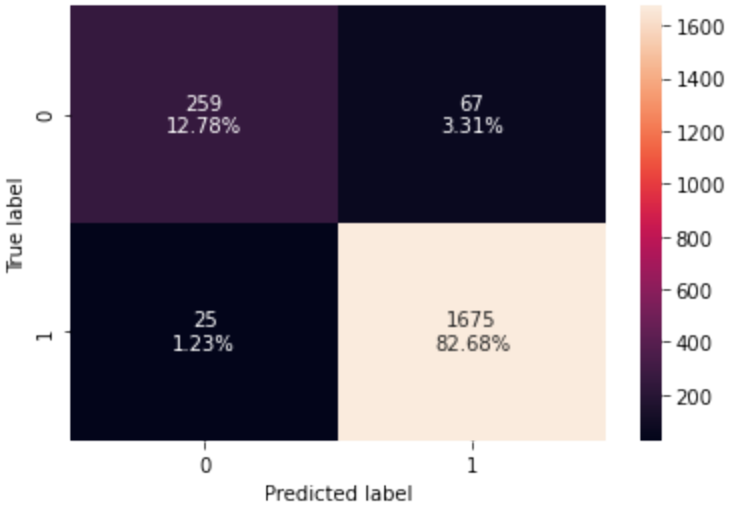
Hyperparameter Tuning

In [85]:
```
# Choose the type of classifier.
bagging_estimator_tuned = BaggingClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {'max_samples': [0.7,0.8,0.9,1],
              'max_features': [0.7,0.8,0.9,1],
              'n_estimators' : [10,20,30,40,50],
             }

# Type of scoring used to compare parameter combinations
acc_scorer = make_scorer(accuracy_score)

# Run the grid search
grid_obj = GridSearchCV(bagging_estimator_tuned, parameters, scoring=acc_scorer,
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
bagging_estimator_tuned = grid_obj.best_estimator_

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

Out[85]:
```
BaggingClassifier(max_features=0.8, max_samples=0.8, n_estimators=50,
                  random_state=1)
```

In [86]:
```
#Calculating different metrics
bagging_estimator_tuned_model_train_perf=model_performance_classification_sklear
print("Training performance:\n",bagging_estimator_tuned_model_train_perf)
bagging_estimator_tuned_model_val_perf=model_performance_classification_sklearn(
print("Validation performance:\n",bagging_estimator_tuned_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(bagging_estimator_tuned,X_val,y_val)
```

```
Training performance:
    Accuracy  Recall  Precision        F1
0  0.999671     1.0   0.999608  0.999804
Validation performance:
    Accuracy    Recall  Precision        F1
0   0.95459  0.985294   0.961538  0.973271
```



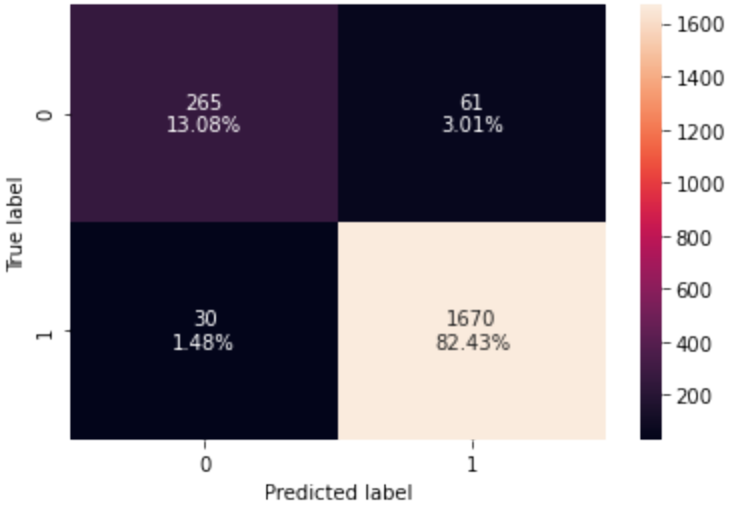
- Overfitting issue seems fixed now but overrall performance has been reduced on both validation and training sets
- Let's try Random Forest Classifier and see if the model performance improves.

AdaBoost Classifier

In [87]:

```
#Fitting the model
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train,y_train)

#Calculating different metrics
ab_classifier_model_train_perf=model_performance_classification_sklearn(ab_class
print("Training performance:\n",ab_classifier_model_train_perf)
ab_classifier_model_val_perf=model_performance_classification_sklearn(ab_classif
print("Validation performance:\n",ab_classifier_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(ab_classifier,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.961317  0.982546   0.971683  0.977084
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.955084  0.982353    0.96476  0.973477
```



- Performance seems improved but will also try to implement hyperparameter tuning and see the result

Hyperparameter Tuning

In [184…

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

# Grid of parameters to choose from
parameters = {
    #Let's try different max_depth for base_estimator
    "base_estimator":[DecisionTreeClassifier(max_depth=1),DecisionTreeClassifier
                      DecisionTreeClassifier(max_depth=3)],
    "n_estimators": np.arange(10,110,10),
    "learning_rate":np.arange(0.1,2,0.1)
}

# Type of scoring used to compare parameter  combinations
acc_scorer = make_scorer(accuracy_score)

# Run the grid search
grid_obj = GridSearchCV(abc_tuned, parameters, scoring=acc_scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
abc_tuned = grid_obj.best_estimator_

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

Out[184…

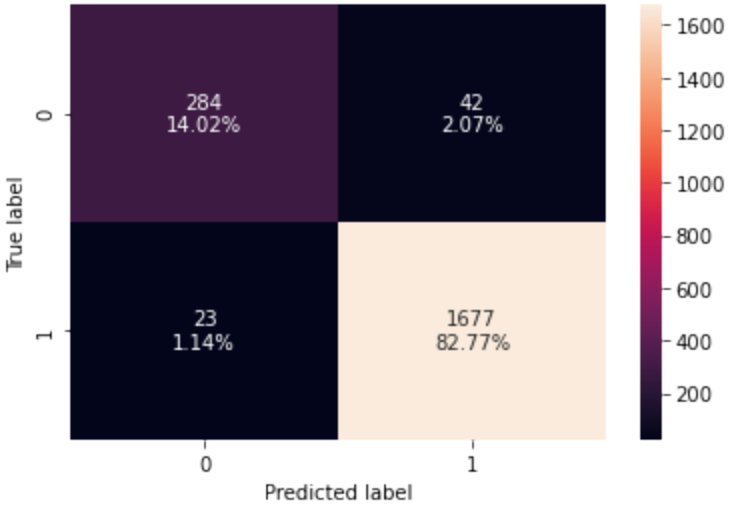
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2),
                   learning_rate=0.5, n_estimators=90, random_state=1)
```

In [186…

```
#Calculating different metrics
abc_tuned_model_train_perf=model_performance_classification_sklearn(abc_tuned, X
print("Training performance:\n",abc_tuned_model_train_perf)
abc_tuned_model_val_perf=model_performance_classification_sklearn(abc_tuned, X_v
print("Validation performance:\n",abc_tuned_model_val_perf)
```

```
#Creating confusion matrix
confusion_matrix_sklearn(abc_tuned,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.989136  0.994705   0.992369  0.993536
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.967917  0.986471   0.975567  0.980989
```

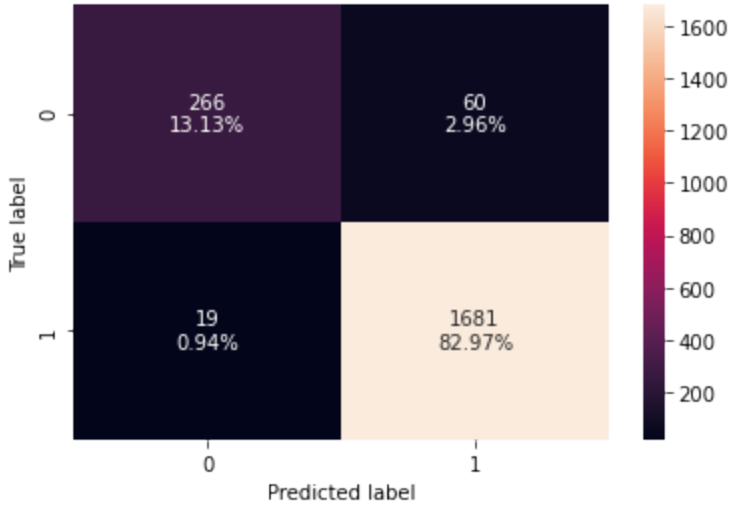


Gradient Boosting Classifier

In [90]:

```
#Fitting the model
gb_classifier = GradientBoostingClassifier(random_state=1)
gb_classifier.fit(X_train,y_train)

#Calculating different metrics
gb_classifier_model_train_perf=model_performance_classification_sklearn(gb_class
print("Training performance:\n",gb_classifier_model_train_perf)
gb_classifier_model_val_perf=model_performance_classification_sklearn(gb_classif
print("Validation performance:\n",gb_classifier_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(gb_classifier,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.978107  0.992155   0.981949  0.987026
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.961007  0.988824   0.965537  0.977042
```



- Performance less than AdaBoost

Stacking Classifier

- Stacking classifier stacks the output of individual estimators and use a classifier to compute the final prediction
- Stacking allows using the strength of each estimator by using their output as input of a final estimator

In [91]:
```
estimators = [('Random Forest',rf_tuned), ('Bagging Classifier',bagging_estimato

final_estimator = abc_tuned

stacking_classifier= StackingClassifier(estimators=estimators,final_estimator=fi

stacking_classifier.fit(X_train,y_train)
```

Out[91]:
```
StackingClassifier(estimators=[('Random Forest',
                                RandomForestClassifier(class_weight={0: 0.18,
                                                                     1: 0.82},
                                                       max_depth=15,
                                                       max_features=None,
                                                       oob_score=True,
                                                       random_state=1)),
                               ('Bagging Classifier',
                                BaggingClassifier(max_features=0.8,
                                                  max_samples=0.8,
                                                  n_estimators=50,
                                                  random_state=1)),
                               ('Decision Tree',
                                DecisionTreeClassifier(class_weight={0: 0.18,
                                                                     1: 0.72},
                                                       max_depth=6,
                                                       max_leaf_nodes=15,
                                                       min_impurity_decrease=0.0
001,
                                                       random_state=1))],
                   final_estimator=AdaBoostClassifier(base_estimator=DecisionTre
eClassifier(max_depth=2),
                                                      learning_rate=0.5,
```

```
                                                    n_estimators=90,
                                                    random_state=1))
```

Calculating different metrics

stacking_classifier_model_train_perf=model_performance_classification_sklearn(stacking_classifie
X_train,y_train) print("Training performance:\n",stacking_classifier_model_train_perf)
stacking_classifier_model_val_perf=model_performance_classification_sklearn(stacking_classifier,
X_val,y_val) print("Validation performance:\n",stacking_classifier_model_val_perf)

Creating confusion matrix

confusion_matrix_sklearn(stacking_classifier,X_val,y_val)

- Performance seems less than AdaBoost but will also try to implement hyperparameter tuning and see the result

Comparing all models

In [154…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [d_tree_model_train_perf.T,pruned_dtree_model_train_perf.T, dtree_estimator_
    rf_tuned_model_train_perf.T,bagging_classifier_model_train_perf.T,bagging_es
     abc_tuned_model_train_perf.T,gb_classifier_model_train_perf.T, stacking_cla
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree",
    "Prunned Decision Tree",
    "Decision Tree Estimator",
    "Random Forest Estimator",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Estimator Tuned",
    "Adaboost Classifier",
    "Adaboost Classifier Tuned",
     "Gradient Boost Classifier","Stacking Classifier"]
print("Training performance comparison:")
models_train_comp_df
```

Out[154…

```
Training performance comparison:
```

	Decision Tree	Prunned Decision Tree	Decision Tree Estimator	Random Forest Estimator	Random Forest Tuned	Bagging Classifier	Bagging Estimator Tuned	Adaboost Classifier	Ad Cl
Accuracy	1.0	1.0	0.907335	1.0	0.998683	0.996926	0.999671	0.945697	0.
Recall	1.0	1.0	0.956266	1.0	1.000000	0.993852	1.000000	0.940574	0.
Precision	1.0	1.0	0.871025	1.0	0.998434	1.000000	0.999608	0.950311	0.
F1	1.0	1.0	0.911657	1.0	0.999216	0.996917	0.999804	0.945417	0.

In [156…

```
# Validation performance comparison

models_val_comp_df = pd.concat(
    [d_tree_model_val_perf.T,pruned_dtree_model_val_perf.T, dtree_estimator_mode
    rf_tuned_model_val_perf.T,bagging_classifier_model_val_perf.T,bagging_estima
     abc_tuned_model_val_perf.T,gb_classifier_model_val_perf.T, stacking_classif
    axis=1,
)
models_val_comp_df.columns = [
    "Decision Tree",
    "Prunned Decision Tree",
    "Decision Tree Estimator",
    "Random Forest Estimator",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Estimator Tuned",
    "Adaboost Classifier",
    "Adaboost Classifier Tuned",
     "Gradient Boost Classifier","Stacking Classifier"]
print("Validation performance comparison:")
models_val_comp_df
```

```
Validation performance comparison:
```

Out[156…

	Decision Tree	Prunned Decision Tree	Decision Tree Estimator	Random Forest Estimator	Random Forest Tuned	Bagging Classifier	Bagging Estimator Tuned	Adaboost Classifier	A C
Accuracy	0.912142	0.912142	0.909181	0.941264	0.958539	0.919052	0.954590	0.921027	(
Recall	0.918824	0.918824	0.994706	0.945882	0.984118	0.920000	0.985294	0.920000	(
Precision	0.975031	0.975031	0.906217	0.983486	0.967052	0.982412	0.961538	0.984887	(
F1	0.946093	0.946093	0.948402	0.964318	0.975510	0.950182	0.973271	0.951338	0

Oversampling train data using SMOTE

In [93]:

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

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


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


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.shap
```

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

After Oversampling, counts of label 'Yes': 5099
```

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

After Oversampling, the shape of train_X: (10198, 20)
After Oversampling, the shape of train_y: (10198,)
```

Logistic Regression on oversampled data

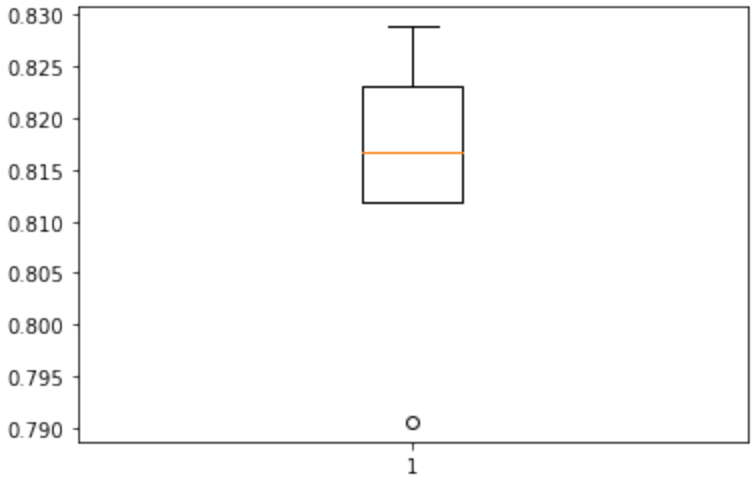
In [94]:
```
log_reg_over = LogisticRegression(random_state=1)

# Training the basic logistic regression model with training set
log_reg_over.fit(X_train_over, y_train_over)
```

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

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 [95]:
```
scoring = "accuracy"
kfold = StratifiedKFold(
    n_splits=5, shuffle=True, random_state=1
)  # Setting number of splits equal to 5
cv_result_over = cross_val_score(
    estimator=log_reg_over, X=X_train_over, y=y_train_over, scoring=scoring, cv=
)
# Plotting boxplots for CV scores of model defined above
plt.boxplot(cv_result_over)
plt.show()
```



- Performance of model on training set varies between 0.78 to 0.79, which is an improvement from the previous model
- Let's check the performance on the validation set.

In [96]:
```
# Calculating different metrics on train set
log_reg_over_train_perf = model_performance_classification_sklearn(
    log_reg_over, X_train_over, y_train_over
```

```
)
print("Training performance:")
log_reg_over_train_perf
```

Training performance:

Out[96]:

	Accuracy	Recall	Precision	F1
0	0.826338	0.823495	0.828205	0.825843

In [97]:

```
# Calculating different metrics on validation set
log_reg_over_val_perf = model_performance_classification_sklearn(
    log_reg_over, X_val, y_val
)
print("validation performance:")
log_reg_over_val_perf
```

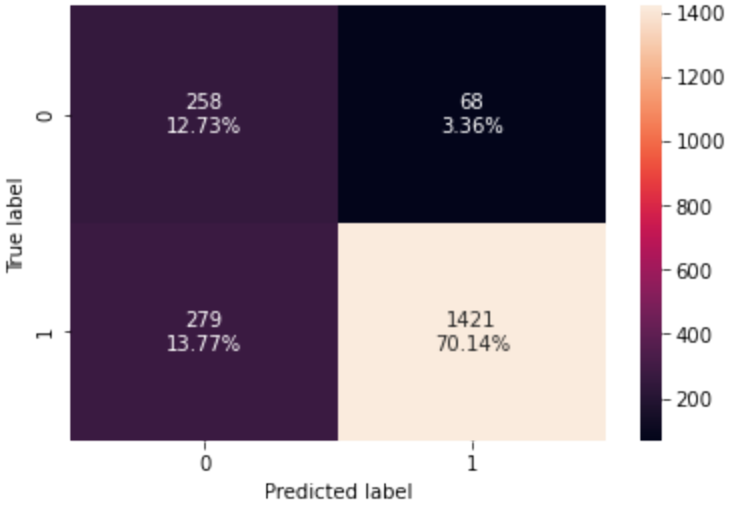
validation performance:

Out[97]:

	Accuracy	Recall	Precision	F1
0	0.828727	0.835882	0.954332	0.891188

In [99]:

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

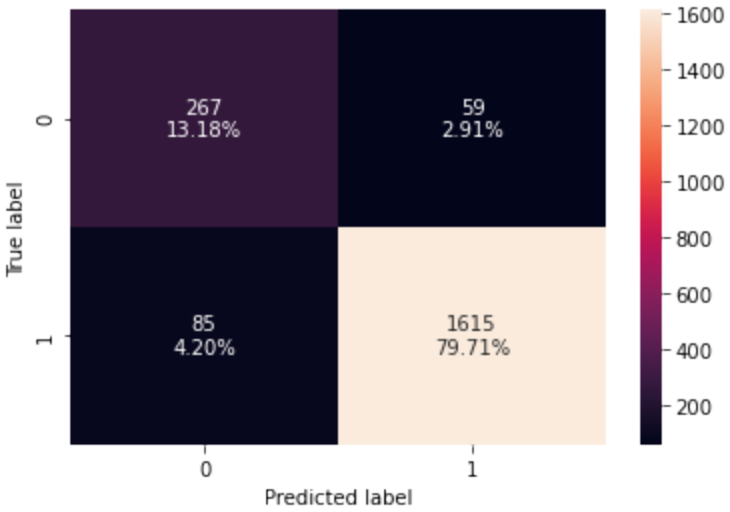


Decision Tree Classifier on oversampled data

In [103…

```
#Fitting the model
d_tree = DecisionTreeClassifier(random_state=1)
d_tree.fit(X_train_over,y_train_over)

#Calculating different metrics
d_tree_model_train_perf=model_performance_classification_sklearn(d_tree, X_train
print("Training performance:\n", d_tree_model_train_perf)
d_tree_model_val_perf=model_performance_classification_sklearn(d_tree, X_val,y_v
print("Validation performance:\n", d_tree_model_val_perf)
#Creating confusion matrix
confusion_matrix_sklearn(d_tree,X_val,y_val)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Validation performance:
    Accuracy  Recall  Precision        F1
0  0.928924    0.95   0.964755  0.957321
```



- Model performance increased nicely than logistic regression
- But we can see overfitting in the data set

Hyperparameter Tuning

In [105…

```
#Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight={0:0.18,1:0.72},random_sta

# Grid of parameters to choose from
parameters = {'max_depth': np.arange(2,30),
              'min_samples_leaf': [1, 2, 5, 7, 10],
              'max_leaf_nodes' : [2, 3, 5, 10,15],
              'min_impurity_decrease': [0.0001,0.001,0.01,0.1]
             }

# Type of scoring used to compare parameter combinations
scorer = make_scorer(accuracy_score)

# Run the grid search
grid_obj = GridSearchCV(dtree_estimator, parameters, scoring=scorer,n_jobs=-1)
grid_obj = grid_obj.fit(X_train_over, y_train_over)

# Set the clf to the best combination of parameters
dtree_estimator = grid_obj.best_estimator_

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

Out[105…

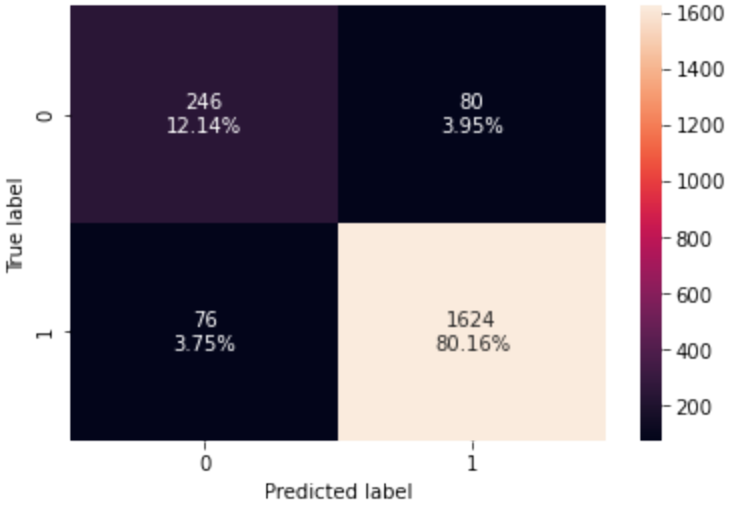
```
DecisionTreeClassifier(class_weight={0: 0.18, 1: 0.72}, max_depth=6,
                       max_leaf_nodes=10, min_impurity_decrease=0.0001,
                       random_state=1)
```

In [106…

```
#Calculating different metrics
dtree_estimator_model_train_perf=model_performance_classification_sklearn(dtree_
```

```
print("Training performance:\n", dtree_estimator_model_train_perf)
dtree_estimator_model_val_perf=model_performance_classification_sklearn(dtree_es
print("Validation performance:\n", dtree_estimator_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(dtree_estimator,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.907335  0.956266   0.871025  0.911657
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.923001  0.955294   0.953052  0.954172
```



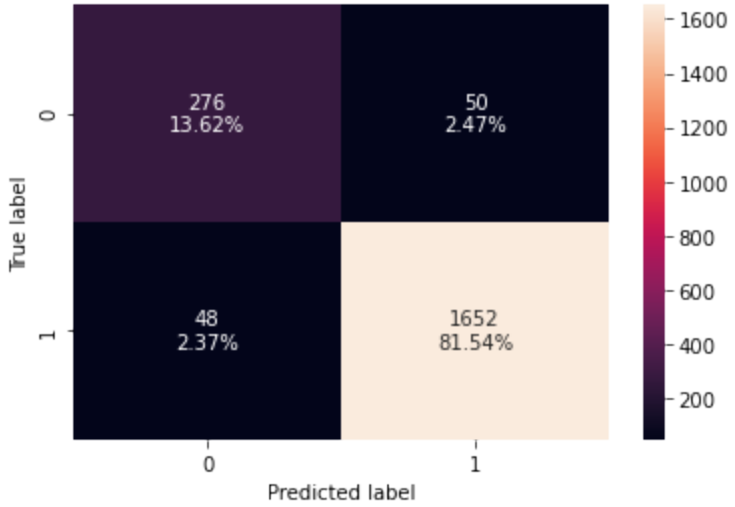
- Overfittng has been reduced but model performance reduced comparatively

Random Forest Classifier on oversampled data

In [107…

```
#Fitting the model
rf_estimator = RandomForestClassifier(random_state=1)
rf_estimator.fit(X_train_over,y_train_over)

#Calculating different metrics
rf_estimator_model_train_perf=model_performance_classification_sklearn(rf_estima
print("Training performance:\n",rf_estimator_model_train_perf)
rf_estimator_model_val_perf=model_performance_classification_sklearn(rf_estimato
print("Validation performance:\n",rf_estimator_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(rf_estimator,X_val,y_val)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.951629  0.971765   0.970623  0.971193
```



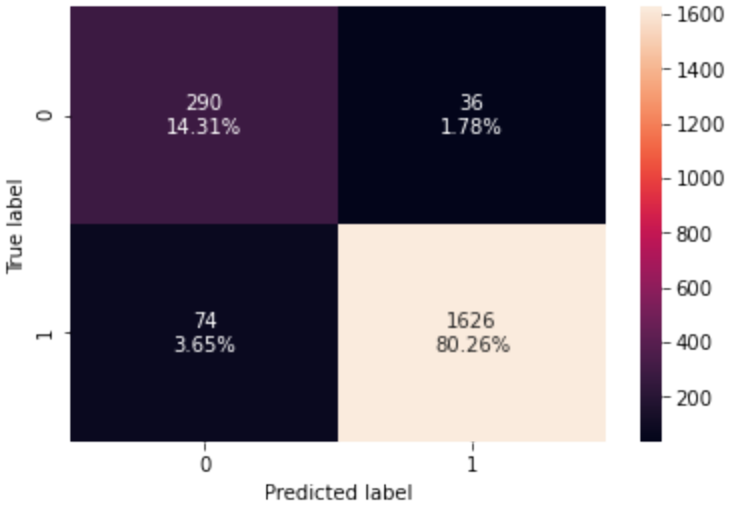
- All the metrics are perfoming well than Decision tree
- We can see model slightly overfiting the training set
- We'll try to reduce overfitting and improve accuracy/recall by hyperparameter tuning.

Bagging Classifier on oversampled data

In [111…

```
#Fitting the model
bagging_classifier = BaggingClassifier(random_state=1)
bagging_classifier.fit(X_train_over,y_train_over)

#Calculating different metrics
bagging_classifier_model_train_perf=model_performance_classification_sklearn(bag
print("Training performance:\n",bagging_classifier_model_train_perf)
bagging_classifier_model_val_perf=model_performance_classification_sklearn(baggi
print("Validation performance:\n",bagging_classifier_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(bagging_classifier,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.997058  0.995097   0.999016  0.997052
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.945706  0.956471   0.978339  0.967281
```



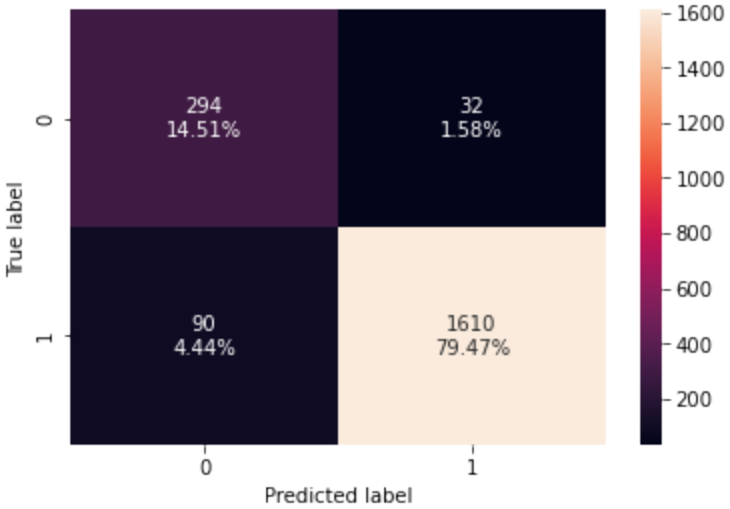
- We can see model slightly overfiting the training set

AdaBoost Classifier on oversampled data

In [112…

```
#Fitting the model
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train_over,y_train_over)

#Calculating different metrics
ab_classifier_model_train_perf=model_performance_classification_sklearn(ab_class
print("Training performance:\n",ab_classifier_model_train_perf)
ab_classifier_model_val_perf=model_performance_classification_sklearn(ab_classif
print("Validation performance:\n",ab_classifier_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(ab_classifier,X_val,y_val)
```

```
Training performance:
    Accuracy   Recall  Precision        F1
0  0.960777  0.95254   0.968495  0.960451
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.939783  0.947059   0.980512  0.963495
```



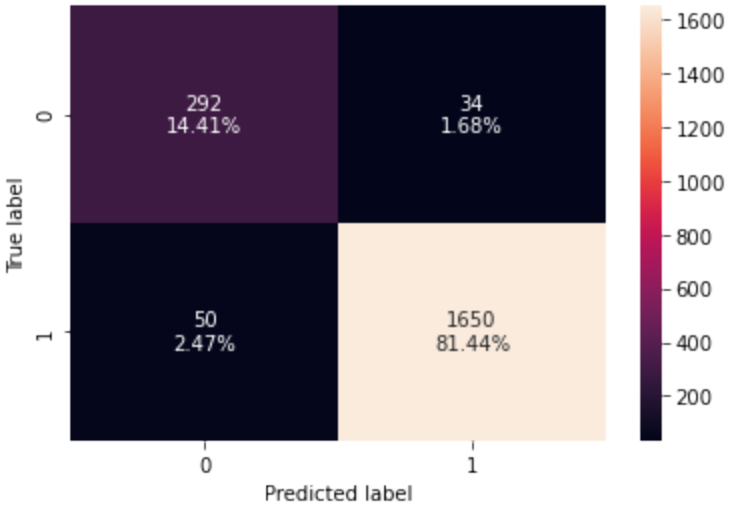
- Model performance seems good but still there are some slight overfitting issue on all the metrics

Gradient Boosting Classifier on oversampled data

In [113…

```
#Fitting the model
gb_classifier = GradientBoostingClassifier(random_state=1)
gb_classifier.fit(X_train_over,y_train_over)

#Calculating different metrics
gb_classifier_model_train_perf=model_performance_classification_sklearn(gb_class
print("Training performance:\n",gb_classifier_model_train_perf)
gb_classifier_model_val_perf=model_performance_classification_sklearn(gb_classif
print("Validation performance:\n",gb_classifier_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(gb_classifier,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.976172  0.972544   0.979652  0.976085
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.958539  0.970588    0.97981  0.975177
```



- Performance on the training set improved but the model is not able to replicate the same for the validation set.
- Model is slightly overfitting.
- Lets try:

  a) Regularization to see if overfitting can be reduced

  b) Undersampling the train to handle the imbalance between classes and check the model performance.

Regularization

In [118…
```
# Choose the type of classifier.
lr_estimator = LogisticRegression(random_state=1, solver="saga")

# Grid of parameters to choose from
parameters = {"C": np.arange(0.1, 1.1, 0.1)}

# Run the grid search
grid_obj = GridSearchCV(lr_estimator, parameters, scoring="accuracy")
grid_obj = grid_obj.fit(X_train_over, y_train_over)

# Set the clf to the best combination of parameters
lr_estimator = grid_obj.best_estimator_

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

Out[118… `LogisticRegression(C=0.1, random_state=1, solver='saga')`

In [119…
```
# Calculating different metrics on train set
log_reg_reg_train_perf = model_performance_classification_sklearn(
    lr_estimator, X_train_over, y_train_over
)
print("Training performance:")
log_reg_reg_train_perf
```

Training performance:

Out[119…

	Accuracy	Recall	Precision	F1
0	0.704648	0.849774	0.658611	0.742079

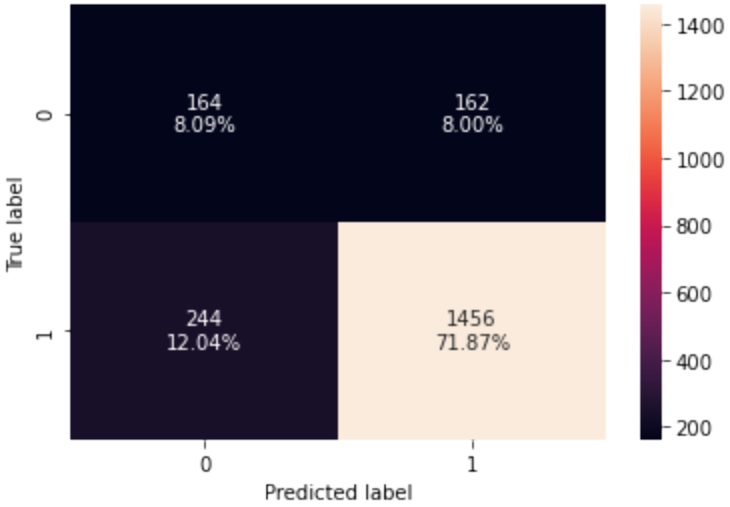
In [120…
```
# Calculating different metrics on validation set
log_reg_reg_val_perf = model_performance_classification_sklearn(
    lr_estimator, X_val, y_val
)
print("Validation performance:")
log_reg_reg_val_perf
```

Validation performance:

Out[120…

	Accuracy	Recall	Precision	F1
0	0.799605	0.856471	0.899876	0.877637

In [121…
```
# creating confusion matrix
confusion_matrix_sklearn(lr_estimator, X_val, y_val)
```



- After regularization, overfitting has reduced but the model is perfomance has been reduced.
- Let's try undersampling now.

Comparing all models on oversampled data

In [158…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [log_reg_over_train_perf.T,d_tree_model_train_perf.T, dtree_estimator_model_
    bagging_classifier_model_train_perf.T,ab_classifier_model_train_perf.T,gb_cl
    axis=1,
)
models_train_comp_df.columns = [
    "Logistic Regression",
    "Decision Tree",
    "Decision Tree Estimator",
    "Random Forest Estimator",
    "Bagging Classifier",
    "Adaboost Classifier",
    "Gradient Boost Classifier",
    "Stacking Classifier"]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[158…

	Logistic Regression	Decision Tree	Decision Tree Estimator	Random Forest Estimator	Bagging Classifier	Adaboost Classifier	Gradient Boost Classifier	Stacking Classifier
Accuracy	0.826338	1.0	0.907335	1.0	0.996926	0.945697	0.973873	0.704648
Recall	0.823495	1.0	0.956266	1.0	0.993852	0.940574	0.967213	0.849774
Precision	0.828205	1.0	0.871025	1.0	1.000000	0.950311	0.980270	0.658611
F1	0.825843	1.0	0.911657	1.0	0.996917	0.945417	0.973698	0.742079

In [159…

```
# Validation performance comparison
```

```
models_val_comp_df = pd.concat(
    [log_reg_over_val_perf.T,d_tree_model_val_perf.T, dtree_estimator_model_val_
    bagging_classifier_model_val_perf.T,ab_classifier_model_val_perf.T,gb_classi
    axis=1,
)
models_val_comp_df.columns = [
    "Logistic Regression",
    "Decision Tree",
    "Decision Tree Estimator",
    "Random Forest Estimator",
    "Bagging Classifier",
    "Adaboost Classifier",
    "Gradient Boost Classifier",
    "Stacking Classifier"]
print("Validation performance comparison:")
models_val_comp_df
```

```
Validation performance comparison:
```

Out[159…

	Logistic Regression	Decision Tree	Decision Tree Estimator	Random Forest Estimator	Bagging Classifier	Adaboost Classifier	Gradient Boost Classifier	Stacking Classifier
Accuracy	0.828727	0.912142	0.909181	0.941264	0.919052	0.921027	0.946199	0.799605
Recall	0.835882	0.918824	0.994706	0.945882	0.920000	0.920000	0.950588	0.856471
Precision	0.954332	0.975031	0.906217	0.983486	0.982412	0.984887	0.984765	0.899876
F1	0.891188	0.946093	0.948402	0.964318	0.950182	0.951338	0.967375	0.877637

Undersampling train data using Random Under Sampler

In [123…

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

In [124…

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

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

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

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

After Undersampling, counts of label 'Yes': 976
After Undersampling, counts of label 'No': 976

After Undersampling, the shape of train_X: (1952, 20)
After Undersampling, the shape of train_y: (1952,)
```

Logistic Regression on undersampled data

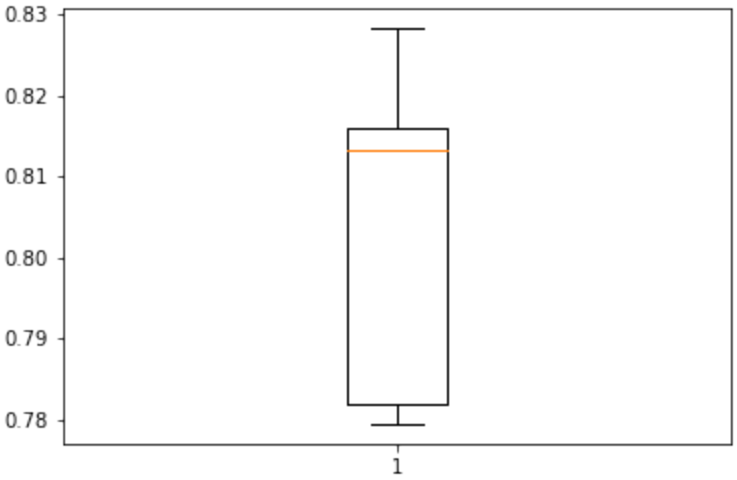
In [125…
```
log_reg_under = LogisticRegression(random_state=1)
log_reg_under.fit(X_train_un, y_train_un)
```

Out[125… LogisticRegression(random_state=1)

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 [126…
```
scoring = "accuracy"
kfold = StratifiedKFold(
    n_splits=5, shuffle=True, random_state=1
)  # Setting number of splits equal to 5
cv_result_under = cross_val_score(
    estimator=log_reg_under, X=X_train_un, y=y_train_un, scoring=scoring, cv=kfo
)
# Plotting boxplots for CV scores of model defined above
plt.boxplot(cv_result_under)
plt.show()
```



In [127…
```
# Calculating different metrics on train set
log_reg_under_train_perf = model_performance_classification_sklearn(
    log_reg_under, X_train_un, y_train_un
)
print("Training performance:")
log_reg_under_train_perf
```

Training performance:

Out[127…

	Accuracy	Recall	Precision	F1
0	0.815061	0.806352	0.820647	0.813437

In [128…
```
# Calculating different metrics on validation set
log_reg_under_val_perf = model_performance_classification_sklearn(
    log_reg_under, X_val, y_val
)
```

```
print("Validation performance:")
log_reg_under_val_perf
```

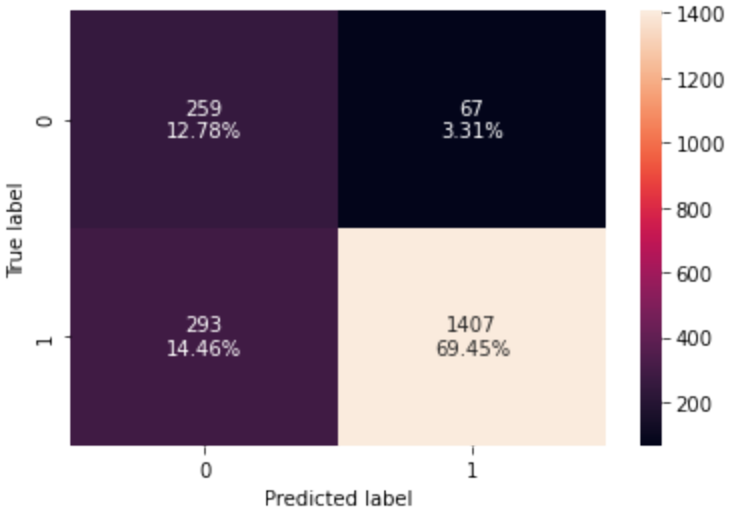
```
Validation performance:
```

Out[128…

	Accuracy	Recall	Precision	F1
0	0.82231	0.827647	0.954545	0.886578

In [129…

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



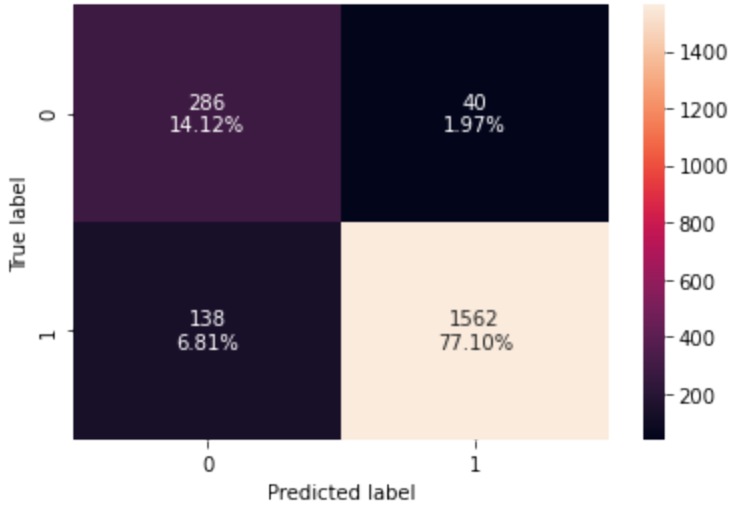
- Model is performing better in the validation data
- But overall model performance is reduced

Decision Tree Classifier on undersampled data

In [137…

```
#Fitting the model
d_tree = DecisionTreeClassifier(random_state=1)
d_tree.fit(X_train_un,y_train_un)

#Calculating different metrics
d_tree_model_train_perf=model_performance_classification_sklearn(d_tree, X_train
print("Training performance:\n", d_tree_model_train_perf)
d_tree_model_val_perf=model_performance_classification_sklearn(d_tree, X_val,y_v
print("Validation performance:\n", d_tree_model_val_perf)
#Creating confusion matrix
confusion_matrix_sklearn(d_tree,X_val,y_val)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.912142  0.918824   0.975031  0.946093
```



- Model is performing better than logistic regression
- Slight overfitting we can able to see it in the validation data set

Cost Complexity Pruning on undersampled data

Let's try pruning the tree and see if the performance improves.

In [138…
```
path = d_tree.cost_complexity_pruning_path(X_train_un, y_train_un)

ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

In [139…
```
clfs_list = []

for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=1, ccp_alpha=ccp_alpha)
    clf.fit(X_train_un, y_train_un)
    clfs_list.append(clf)

print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(clfs_l
```

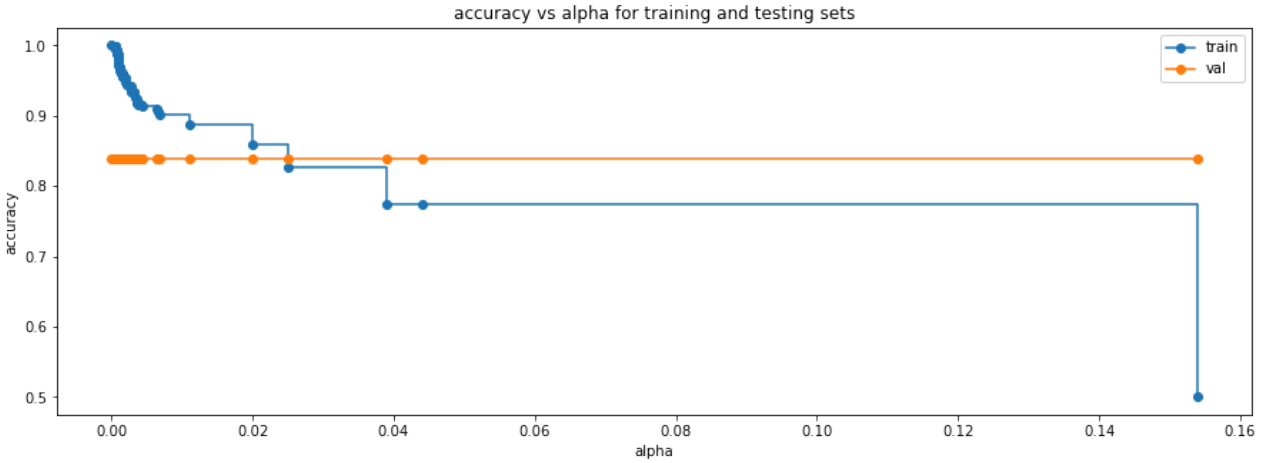
```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.15396751899298589
```

In [140…
```
#Fitting model for each value of alpha and saving the train acc in a list
acc_train_un=[]
for clf in clfs_list:
    pred_train_un=clf.predict(X_train_un)
    values_train_un=accuracy_score(y_train_un,pred_train_un)
    acc_train_un.append(values_train_un)
```

In [142…
```
#Fitting model for each value of alpha and saving the val acc in a list
acc_val_un=[]
for clf in clfs_list:
    pred_val_over=clf.predict(X_val)
    values_val=accuracy_score(y_val,pred_val)
    acc_val_un.append(values_val)
```

In [143…

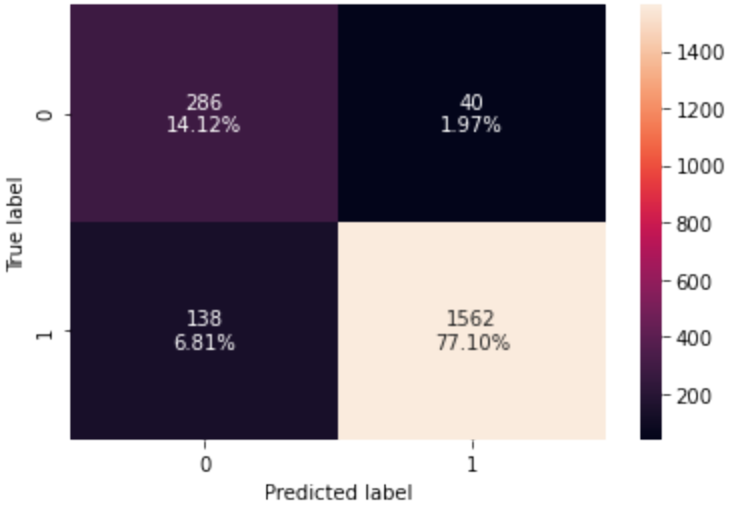
```
#Plotting the graph for Recall VS alpha
fig, ax = plt.subplots(figsize=(15,5))
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, acc_train_un, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, acc_val_un, marker='o', label="val",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



In [144…

```
#Creating the model where we get highest test recall
index_best_pruned_model = np.argmax(acc_val_un)

pruned_dtree_model = clfs_list[index_best_pruned_model]

#Calculating different metrics
pruned_dtree_model_train_perf=model_performance_classification_sklearn(pruned_dt
print("Training performance:\n", pruned_dtree_model_train_perf)
pruned_dtree_model_val_perf=model_performance_classification_sklearn(pruned_dtre
print("Validation performance:\n", pruned_dtree_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(pruned_dtree_model,X_val,y_val)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.912142  0.918824   0.975031  0.946093
```



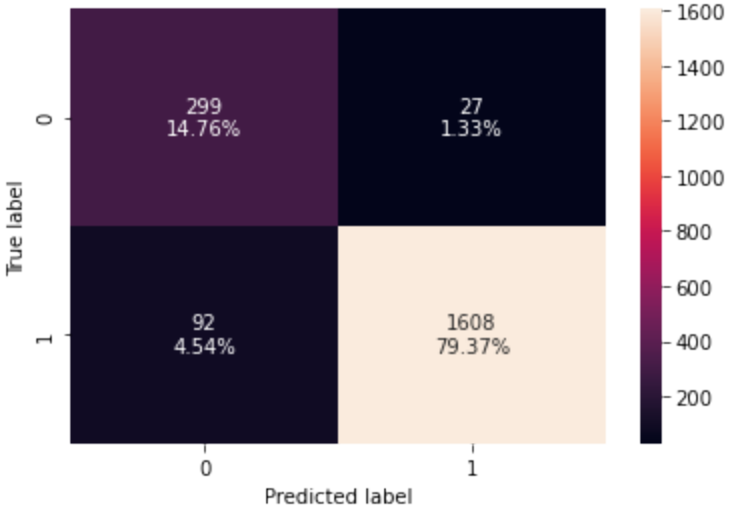
- There is no much difference in the data between Decision tree and after pruning

Random Forest Classifier on undersampled data

In [145…

```
#Fitting the model
rf_estimator = RandomForestClassifier(random_state=1)
rf_estimator.fit(X_train_un,y_train_un)

#Calculating different metrics
rf_estimator_model_train_perf=model_performance_classification_sklearn(rf_estima
print("Training performance:\n",rf_estimator_model_train_perf)
rf_estimator_model_val_perf=model_performance_classification_sklearn(rf_estimato
print("Validation performance:\n",rf_estimator_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(rf_estimator,X_val,y_val)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.941264  0.945882   0.983486  0.964318
```



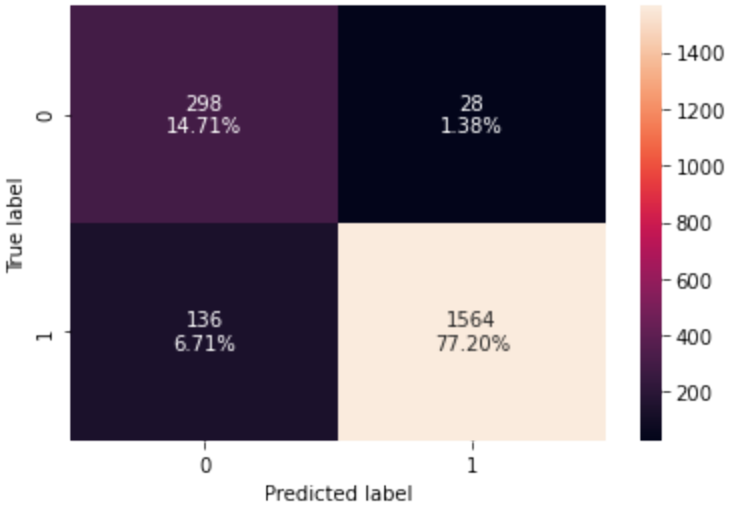
- Model is performing better than logistic regression and Decision tree
- Slight overfitting we can able to see it in the validation data set

Bagging Classifier on undersampled data

In [146…

```
#Fitting the model
bagging_classifier = BaggingClassifier(random_state=1)
bagging_classifier.fit(X_train_un,y_train_un)

#Calculating different metrics
bagging_classifier_model_train_perf=model_performance_classification_sklearn(bag
print("Training performance:\n",bagging_classifier_model_train_perf)
bagging_classifier_model_val_perf=model_performance_classification_sklearn(baggi
print("Validation performance:\n",bagging_classifier_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(bagging_classifier,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.996926  0.993852        1.0  0.996917
Validation performance:
    Accuracy  Recall  Precision        F1
0  0.919052    0.92   0.982412  0.950182
```



- Model is not performing better than Random forrest classifier
- Slight overfitting we can able to see it in the validation data set

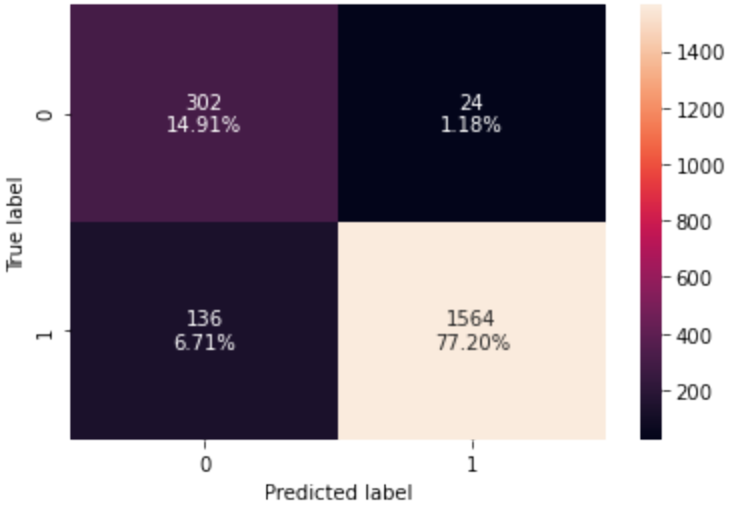
AdaBoost Classifier on undersampled data

In [148…

```
#Fitting the model
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train_un,y_train_un)

#Calculating different metrics
ab_classifier_model_train_perf=model_performance_classification_sklearn(ab_class
print("Training performance:\n",ab_classifier_model_train_perf)
ab_classifier_model_val_perf=model_performance_classification_sklearn(ab_classif
print("Validation performance:\n",ab_classifier_model_val_perf)
```

```
#Creating confusion matrix
confusion_matrix_sklearn(ab_classifier,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.945697  0.940574   0.950311  0.945417
Validation performance:
    Accuracy  Recall  Precision        F1
0  0.921027    0.92   0.984887  0.951338
```



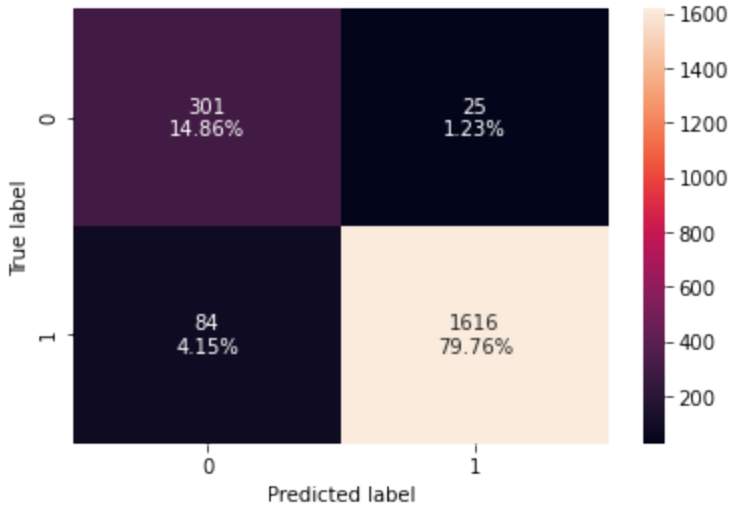
- Overfitting has been reduced but overrall performance of the model also reduced

Gradient Boosting Classifier on undersampled data

In [150…

```
#Fitting the model
gb_classifier = GradientBoostingClassifier(random_state=1)
gb_classifier.fit(X_train_un,y_train_un)

#Calculating different metrics
gb_classifier_model_train_perf=model_performance_classification_sklearn(gb_class
print("Training performance:\n",gb_classifier_model_train_perf)
gb_classifier_model_val_perf=model_performance_classification_sklearn(gb_classif
print("Validation performance:\n",gb_classifier_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(gb_classifier,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.973873  0.967213    0.98027  0.973698
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.963804  0.989416   0.968163  0.978674
```



- Overfitting has been reduced but overrall performance already increased slightly than AdaBoost

Hyperparameter Tuning

In [152…

```
# Choose the type of classifier.
gbc_tuned = GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),r

# Grid of parameters to choose from
parameters = {
    "n_estimators": [100,150,200,250],
    "subsample":[0.8,0.9,1],
    "max_features":[0.7,0.8,0.9,1]
}

# Type of scoring used to compare parameter combinations
acc_scorer = make_scorer(accuracy_score)

# Run the grid search
grid_obj = GridSearchCV(gbc_tuned, parameters, scoring=acc_scorer,cv=5)
grid_obj = grid_obj.fit(X_train_un, y_train_un)

# Set the clf to the best combination of parameters
gbc_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
gbc_tuned.fit(X_train_un, y_train_un)
```

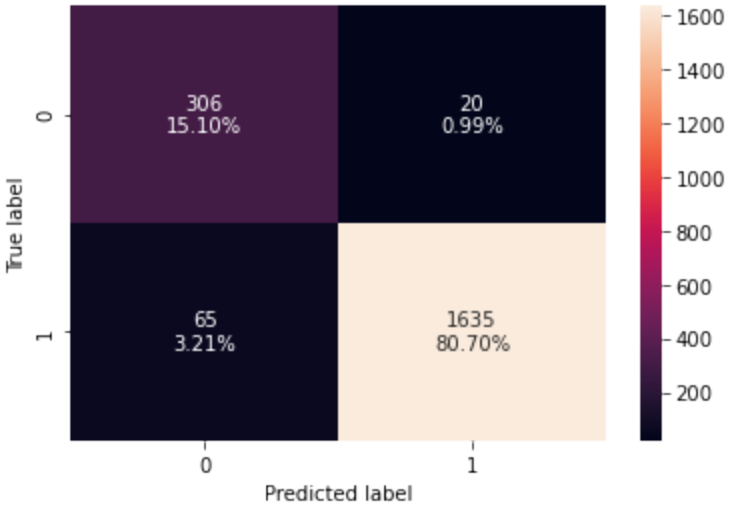
Out[152…

```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.9, n_estimators=200, random_state=1,
                           subsample=0.8)
```

In [160…

```
#Calculating different metrics
gbc_tuned_model_train_perf=model_performance_classification_sklearn(gbc_tuned, X
print("Training performance:\n",gbc_tuned_model_train_perf)
gbc_tuned_model_val_perf=model_performance_classification_sklearn(gbc_tuned, X_v
print("Validation performance:\n",gbc_tuned_model_val_perf)

#Creating confusion matrix
confusion_matrix_sklearn(gbc_tuned,X_val,y_val)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.996414  0.995902   0.996923  0.996412
Validation performance:
    Accuracy    Recall  Precision        F1
0  0.958045  0.961765   0.987915  0.974665
```



Comparing all models on undersampled data

In [170…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [log_reg_under_train_perf.T,d_tree_model_train_perf.T, pruned_dtree_model_tr
    bagging_classifier_model_train_perf.T,ab_classifier_model_train_perf.T,gb_cl
    axis=1,
)
models_train_comp_df.columns = [
    "Logistic Regression",
    "Decision Tree",
    "Pruned Decision Tree",
    "Random Forest Estimator",
    "Bagging Classifier",
    "Adaboost Classifier",
    "Gradient Boost Classifier","Gradient Boost Classifier Tuned"]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[170…

	Logistic Regression	Decision Tree	Pruned Decision Tree	Random Forest Estimator	Bagging Classifier	Adaboost Classifier	Gradient Boost Classifier	Gradient Boost Classifier Tuned
Accuracy	0.815061	1.0	1.0	1.0	0.996926	0.945697	0.973873	0.996414
Recall	0.806352	1.0	1.0	1.0	0.993852	0.940574	0.967213	0.995902
Precision	0.820647	1.0	1.0	1.0	1.000000	0.950311	0.980270	0.996923
F1	0.813437	1.0	1.0	1.0	0.996917	0.945417	0.973698	0.996412

In [171…

```
# Validation performance comparison

models_val_comp_df = pd.concat(
    [log_reg_under_val_perf.T,d_tree_model_val_perf.T, pruned_dtree_model_val_pe
    bagging_classifier_model_val_perf.T,ab_classifier_model_val_perf.T,gb_classi
    axis=1,
)
models_val_comp_df.columns = [
    "Logistic Regression",
    "Decision Tree",
    "Pruned Decision Tree",
    "Random Forest Estimator",
    "Bagging Classifier",
    "Adaboost Classifier",
    "Gradient Boost Classifier","Gradient Boost Classifier Tuned"]
print("Validation performance comparison:")
models_val_comp_df
```

```
Validation performance comparison:
```

Out[171…

	Logistic Regression	Decision Tree	Pruned Decision Tree	Random Forest Estimator	Bagging Classifier	Adaboost Classifier	Gradient Boost Classifier	Gradient Boost Classifier Tuned
Accuracy	0.822310	0.912142	0.912142	0.941264	0.919052	0.921027	0.946199	0.958045
Recall	0.827647	0.918824	0.918824	0.945882	0.920000	0.920000	0.950588	0.961765
Precision	0.954545	0.975031	0.975031	0.983486	0.982412	0.984887	0.984765	0.987915
F1	0.886578	0.946093	0.946093	0.964318	0.950182	0.951338	0.967375	0.974665

Comparison on best tuned models

The below 3 models performed well comparitvely on all the metrics so tuned using hyperparameter tuning

- Adaboost Classifier Tuned
- Bagging Estimator Tuned
- Gradient Boost Classifier Tuned on undersampled data

In [165…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [bagging_estimator_tuned_model_train_perf.T,abc_tuned_model_train_perf.T,gbc
    axis=1,
)
models_train_comp_df.columns = [
    "Bagging Estimator Tuned",
    "Adaboost Classifier Tuned",
    "Gradient Boost Classifier Tuned"]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[165…

	Bagging Estimator Tuned	Adaboost Classifier Tuned	Gradient Boost Classifier Tuned
Accuracy	0.999671	0.989136	0.996414
Recall	1.000000	0.994705	0.995902
Precision	0.999608	0.992369	0.996923
F1	0.999804	0.993536	0.996412

In [166…

```
# validation performance comparison

models_val_comp_df = pd.concat(
    [bagging_estimator_tuned_model_val_perf.T,abc_tuned_model_val_perf.T,gbc_tur
    axis=1,
)
models_val_comp_df.columns = [
    "Bagging Estimator Tuned",
    "Adaboost Classifier Tuned",
    "Gradient Boost Classifier Tuned"]
print("Training performance comparison:")
models_val_comp_df
```

```
Training performance comparison:
```

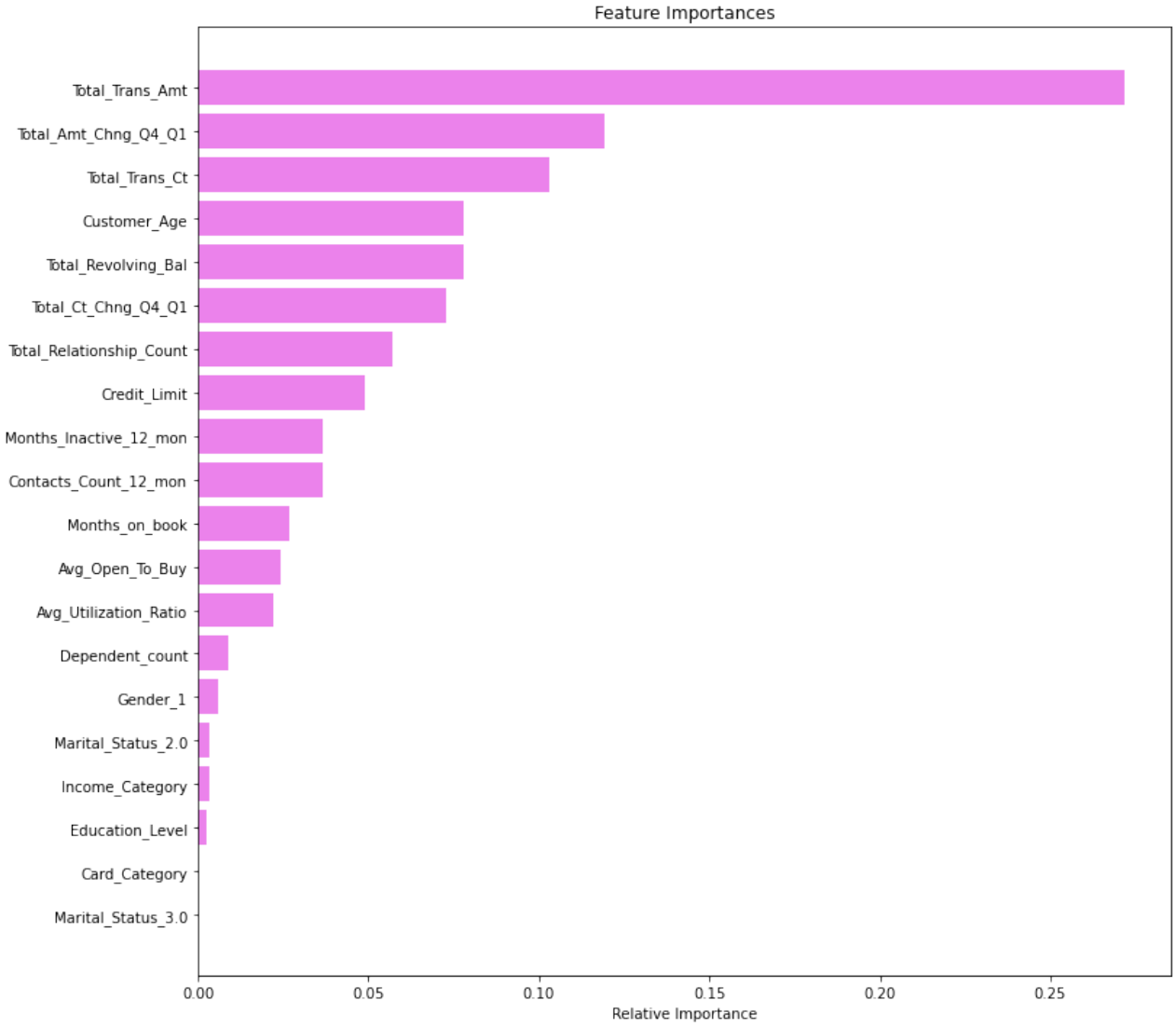
Out[166…

	Bagging Estimator Tuned	Adaboost Classifier Tuned	Gradient Boost Classifier Tuned
Accuracy	0.954590	0.967917	0.958045
Recall	0.985294	0.986471	0.961765
Precision	0.961538	0.975567	0.987915
F1	0.973271	0.980989	0.974665

- Choosing Adaboost tuned model as the best performed model compared with other models

In [172…

```
feature_names = X_train.columns
importances = abc_tuned.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='cente
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- Total_Trans_Amt is the most important feature, followed by Total_Amt_Chng_Q4_Q1 and Total_Trans_Ct as per AdaBoost Tuned model

Productionize the model: make_pipeline for AdaBoost tuned model on Test data

- make_pipeline is an extended version of the pipeline. Here, we don't need to assign names separately to each element of the pipeline.

In [188…
```
#Fitting the model
# defining pipe using make_pipeline
pipe = make_pipeline((AdaBoostClassifier(random_state=1)))
```

In [189…
```
pipe.steps
```

Out[189…
```
[('adaboostclassifier', AdaBoostClassifier(random_state=1))]
```

In [190…
```
pipe.fit(X_train,y_train)
```

Out[190… `Pipeline(steps=[('adaboostclassifier', AdaBoostClassifier(random_state=1))])`

In [193…
```
# pipeline object's accuracy on the train set
pipe.score(X_train, y_train)
```

Out[193… `0.9613168724279836`

In [203…
```
# pipeline object's accuracy on the test set
pipe.score(X_test, y_test)
```

Out[203… `0.9570582428430404`

Business Recommendations

- There are 16.07% of customers who have churned.
- The proportion of gender count is almost equally distributed (52.9% male and 47.1%) compare to proportion of existing and Attrition customer count (83.9% and 16.1%) which is highly imbalanced.
- Customers who have churned are highly educated - A high proportion of education level of attrited customer is Graduate level (30.9%), followed by High School level (19.3%)
- A high proportion of marital status of customers who have churned is Married (46.3%), followed by Single (38.9%) compared to Divorced (7.4%) - marital status of the Attrition customers are highly clustered in Married status and Single. So we can provide offers to attract Married customers.
- As you can see from the proportion of income category of attrited customer, it is highly concentrated around less than 40K income (35.2%), followed by 40K-60K income (17.7%) compare to attrited customers with higher annual income of 80K-120K(15.2%) and over 120K+ (7.2%). I assume that customers with higher income doesn't likely leave their credit card services than less income customers. So we can provide some additional point based offers to attract lower income salary customers.