Problem Overview and solution approach:

- 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.
- 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
- Identify the key variables using EDA
- Build 6 models on the training and validation sets.
- Find the 3 best models that perform well on the recall score in Training and Validation sets.
- Hyper tune the models using random search cross-validation and run the models on Oversampled and Under sampled data.

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 [1]:
```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')


from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import BaggingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder, LabelEncoder
from sklearn.impute import SimpleImputer

# To impute missing values
from sklearn.impute import KNNImputer

# Libtune to tune model, get different metric scores
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_scor
```

Load and overview the dataset

In [2]:
```
data = pd.read_csv('BankChurners.csv')
data.head()
```

Out[2]:

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

5 rows × 21 columns

In [3]:
```
data.drop('CLIENTNUM', inplace = True, axis = 1) #dropping the first column as valuable information wont be gaine
```

In [4]:
```
data.shape #shape of the data
```

Out[4]:
```
(10127, 20)
```

In [5]:
```
data.info() #data type info of every variable
```

```
<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
```

In [6]:
```
data.isnull().sum().sort_values(ascending = False) #null values
```

Out[6]:
```
Education_Level             1519
Marital_Status               749
Avg_Utilization_Ratio          0
Total_Ct_Chng_Q4_Q1            0
Customer_Age                   0
```

```
Gender                      0
Dependent_count             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
Attrition_Flag              0
dtype: int64
```

In [7]:
```
pd.DataFrame(data={'% of Missing Values':round(data.isna().sum()/data.isna().count()*100,2)}) #null values as a p
```

Out[7]:

	% of Missing Values
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
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

In [8]:
```
data.nunique() #unique values within each variable
```

Out[8]:
```
Attrition_Flag               2
Customer_Age                45
Gender                       2
Dependent_count              6
Education_Level              6
Marital_Status               3
Income_Category              6
Card_Category                4
Months_on_book              44
Total_Relationship_Count     6
Months_Inactive_12_mon       7
Contacts_Count_12_mon        7
Credit_Limit              6205
Total_Revolving_Bal       1974
Avg_Open_To_Buy           6813
Total_Amt_Chng_Q4_Q1      1158
Total_Trans_Amt           5033
Total_Trans_Ct             126
Total_Ct_Chng_Q4_Q1        830
Avg_Utilization_Ratio      964
dtype: int64
```

In [9]:
```
#Making a list of all catrgorical variables
```

```
cat_col=['Attrition_Flag', 'Gender','Dependent_count', 'Education_Level', 'Marital_Status',
       'Income_Category', 'Card_Category', "Total_Relationship_Count", "Months_Inactive_12_mon","Contacts_Count_1

#Printing number of count of each unique value in each column
for column in cat_col:
    print(data[column].value_counts())
    print('-'*50)
```

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

In [10]:
```
#Converting the data type of each categorical variable to 'category'
for column in cat_col:
```

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

In [11]:
```
data['Income_Category'] = data['Income_Category'].replace('abc', np.nan) #converting abc value into none values a
```

In [12]:
```
print(data['Income_Category'].value_counts())
print('------')
print(f"Missing Values for Income Category:{(data['Income_Category'].isnull().sum())} values")
```

```
Less than $40K    3561
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
$120K +            727
Name: Income_Category, dtype: int64
------
Missing Values for Income Category:1112 values
```

In [13]:
```
data.isnull().sum().sort_values(ascending = False)
```

Out[13]:
```
Education_Level             1519
Income_Category             1112
Marital_Status               749
Avg_Utilization_Ratio          0
Total_Ct_Chng_Q4_Q1            0
Customer_Age                   0
Gender                         0
Dependent_count                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
Attrition_Flag                 0
dtype: int64
```

In [14]:
```
# data['Education_Level'] = data['Education_Level'].astype(str).replace('nan', 'is_missing').astype('category')
# data['Income_Category'] = data['Income_Category'].astype(str).replace('nan', 'is_missing').astype('category')
# data['Marital_Status'] = data['Marital_Status'].astype(str).replace('nan', 'is_missing').astype('category')
```

In [15]:
```
data.isnull().sum().sort_values(ascending = False)
```

Out[15]:
```
Education_Level             1519
Income_Category             1112
Marital_Status               749
Avg_Utilization_Ratio          0
Total_Ct_Chng_Q4_Q1            0
Customer_Age                   0
Gender                         0
Dependent_count                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
Attrition_Flag                 0
dtype: int64
```

In [16]:
```
data.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  category
 1   Customer_Age              10127 non-null  int64
 2   Gender                    10127 non-null  category
 3   Dependent_count           10127 non-null  category
 4   Education_Level           8608 non-null   category
 5   Marital_Status            9378 non-null   category
 6   Income_Category           9015 non-null   category
 7   Card_Category             10127 non-null  category
 8   Months_on_book            10127 non-null  int64
 9   Total_Relationship_Count  10127 non-null  category
 10  Months_Inactive_12_mon    10127 non-null  category
 11  Contacts_Count_12_mon     10127 non-null  category
 12  Credit_Limit              10127 non-null  float64
 13  Total_Revolving_Bal       10127 non-null  int64
 14  Avg_Open_To_Buy           10127 non-null  float64
 15  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 16  Total_Trans_Amt           10127 non-null  int64
 17  Total_Trans_Ct            10127 non-null  int64
 18  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 19  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: category(10), float64(5), int64(5)
memory usage: 892.2 KB
```

In [17]:
```
data.describe().T
```

Out[17]:

	count	mean	std	min	25%	50%	75%	max
Customer_Age	10127.0	46.325960	8.016814	26.0	41.000	46.000	52.000	73.000
Months_on_book	10127.0	35.928409	7.986416	13.0	31.000	36.000	40.000	56.000
Credit_Limit	10127.0	8631.953698	9088.776650	1438.3	2555.000	4549.000	11067.500	34516.000
Total_Revolving_Bal	10127.0	1162.814061	814.987335	0.0	359.000	1276.000	1784.000	2517.000
Avg_Open_To_Buy	10127.0	7469.139637	9090.685324	3.0	1324.500	3474.000	9859.000	34516.000
Total_Amt_Chng_Q4_Q1	10127.0	0.759941	0.219207	0.0	0.631	0.736	0.859	3.397
Total_Trans_Amt	10127.0	4404.086304	3397.129254	510.0	2155.500	3899.000	4741.000	18484.000
Total_Trans_Ct	10127.0	64.858695	23.472570	10.0	45.000	67.000	81.000	139.000
Total_Ct_Chng_Q4_Q1	10127.0	0.712222	0.238086	0.0	0.582	0.702	0.818	3.714
Avg_Utilization_Ratio	10127.0	0.274894	0.275691	0.0	0.023	0.176	0.503	0.999

Observations:

1. The average customer age is around 46 years with the minimum being 26 years old and maximum being 73 years old.
2. On average months on book by each customer is 40 months with the maximum being 56 months.
3. There is a big range in Credit limit with lowest being 1438 and highest limit being 34516.
4. Total amount changed between q4 and q1 values are all mostly below 1 with max being 3.39.
5. Average utlization ratio is as low as 0 and as max as 0.99.

EDA

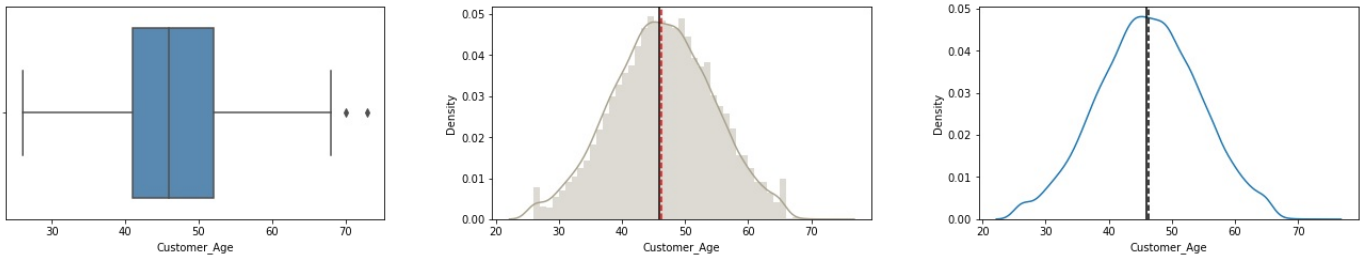
Univariate Analysis

In [18]:
```
def histogram_boxplot(feature):
    """ Boxplot and histogram combined
    feature: 1-d feature array
    """
    figure, (ax_box2, ax_hist2, ax_hist3) = plt.subplots(
        nrows = 1, ncols=3,# Number of rows of the subplot grid= 2
        figsize  = (20,5)) # creating the 2 subplots
    figure.tight_layout(pad = 7)
    sns.boxplot(x = feature,ax=ax_box2, color = '#4B8BBE', orient = 'v') # boxplot will be created
    sns.distplot(feature, kde=True, ax=ax_hist2, color = '#a9a38f') # For histogram
    sns.distplot(feature, kde= True, ax=ax_hist3, hist = False) #Making an outline of the histogram
    ax_hist2.axvline(np.mean(feature), color='r', linestyle='--') # Add mean to the histogram
    ax_hist2.axvline(np.median(feature), color='black', linestyle='-') # Add median to the histogram
    ax_hist3.axvline(np.mean(feature), color = 'black', linestyle = '--') #Adding mean to second histogram
    ax_hist3.axvline(np.median(feature), color='black', linestyle='-') #Adding median to second histogram
```

Observations on Customer Age

In [19]:
```
histogram_boxplot(data['Customer_Age'])
```

In [19]:
```
histogram_boxplot(data['Customer_Age'])
```

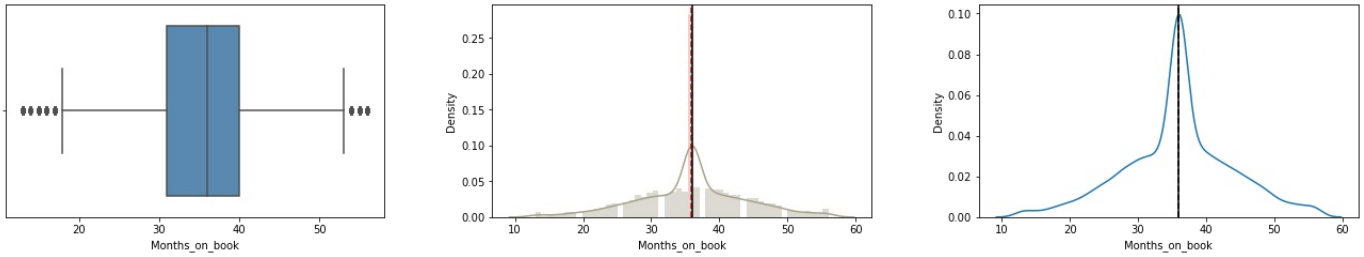


Observations:

1. 2 Outliers are visible.
2. The histogram graph seems to show that the variable is normally distributed as the shape is of a bell-curve.
3. The average age is 46 years.

Observations on Months on Book

In [20]:
```
histogram_boxplot(data['Months_on_book'])
```

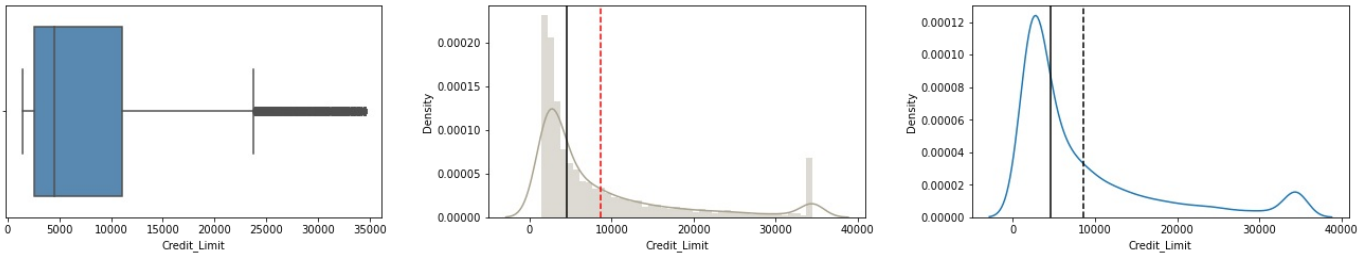


Observations:

1. There are outliers on both end of the boxplot
2. Most of the values converge to the mean around 35.9 months.
3. Most of the values are located between 25 and 45 months.

Observations on Credit Limit

In [21]:
```
histogram_boxplot(data['Credit_Limit'])
```

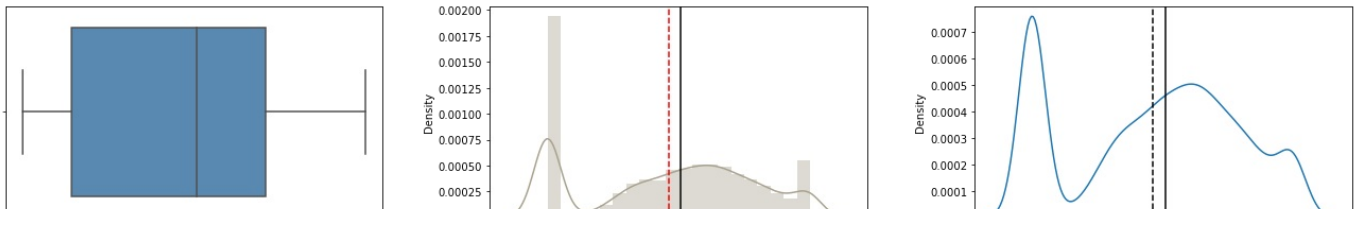


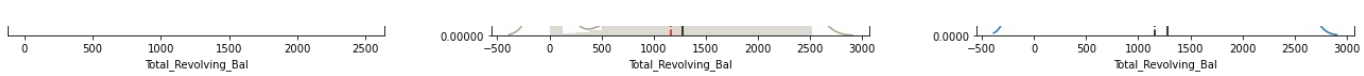
Observations:

1. All the visible outliers are above the maximum, $34,516
2. The graph is slighly right skewed.
3. There is a big range in the data

Observations on Total Revolving Blanace

In [22]:
```
histogram_boxplot(data['Total_Revolving_Bal'])
```



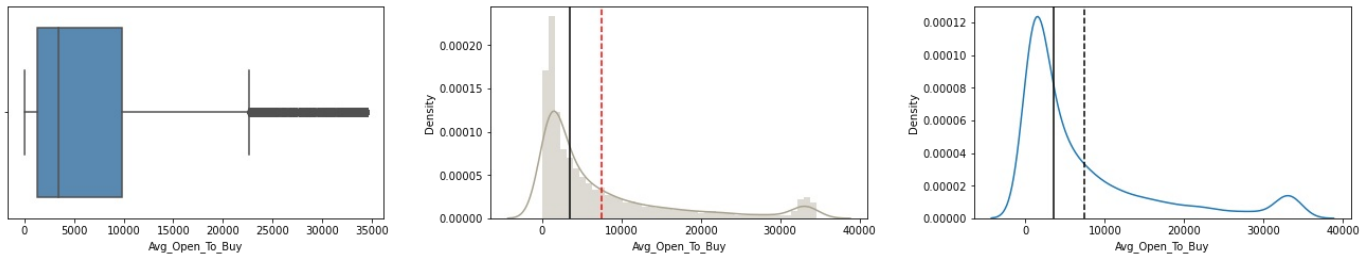


Observations:

1. No outliers seem to be visible.
2. Data mostly clustered around 0 to 100 and from 1000 to 2500.

Observations on Average Open to Buy

In [23]:
```
histogram_boxplot(data['Avg_Open_To_Buy'])
```

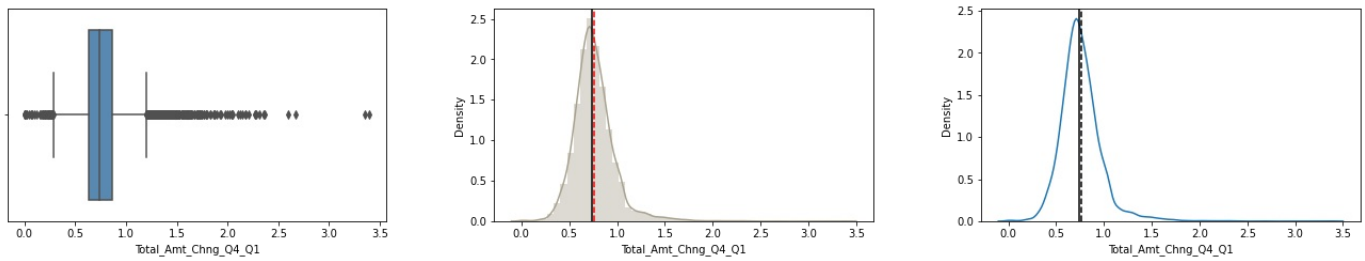


Observations:

1. Outliers visible after the maximum value, $34,516
2. The graph is rightly skewed.

Observations on Total Amount Changed Q4 to Q1

In [24]:
```
histogram_boxplot(data['Total_Amt_Chng_Q4_Q1'])
```

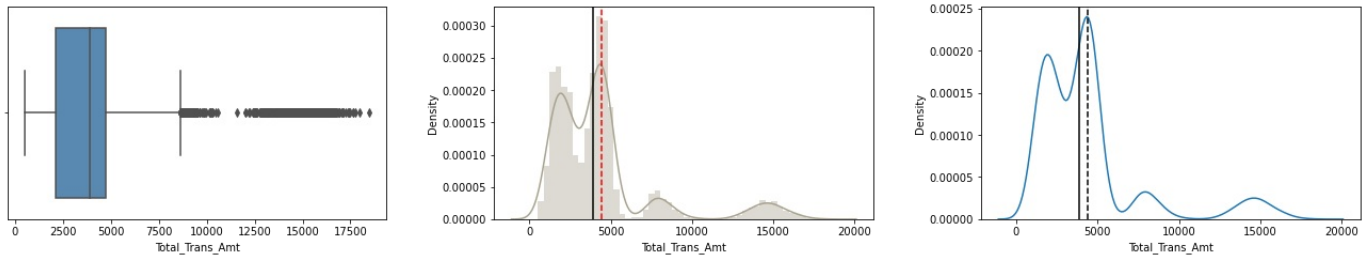


Observations:

1. Outliers visible both to the left and right of the boxplot.
2. Majority of outliers visible to the right of the boxplot
3. The graph is slighyly skewed to the right.
4. Large accumulation of the data from 0 to 1.

Observations on Total Trans Amount

In [25]:
```
histogram_boxplot(data['Total_Trans_Amt'])
```

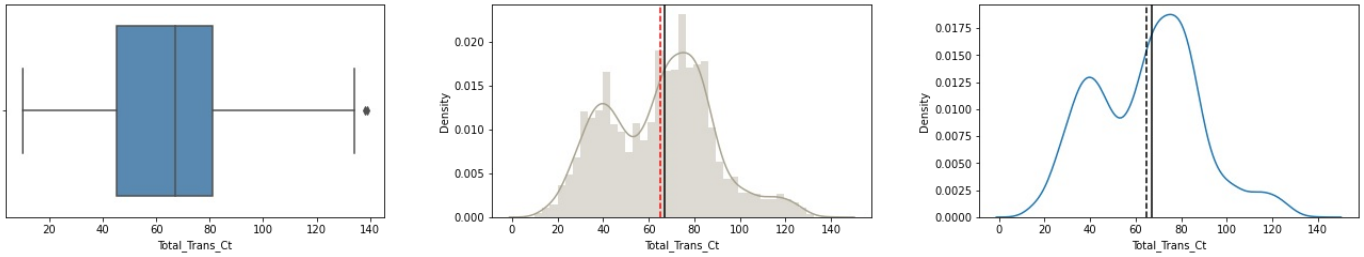


Observations:

1. Outliers primarily visible to the right of the boxplot.
2. The two peaks are visible between 6000 to 10000 and from 13000 to 15000.
3. Most of the data is accumulated from 0 to 5000.

Observations on Total Trans CT

In [26]:
```
histogram_boxplot(data['Total_Trans_Ct'])
```

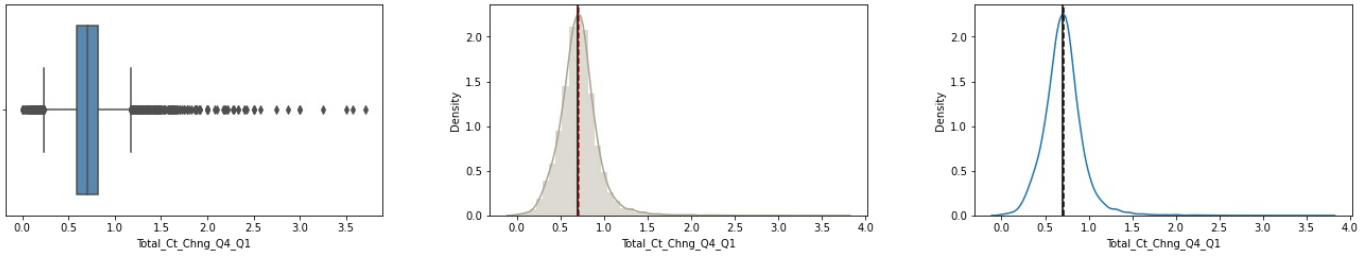


Observations:

1. One outlier value visible after the maximum value.
2. Two peaks visible at 40 and 80.
3. Most of the values are accumulated between 40 to 90.

Observations on Total_CT_Change Q4 to Q1

In [27]:
```
histogram_boxplot(data['Total_Ct_Chng_Q4_Q1'])
```

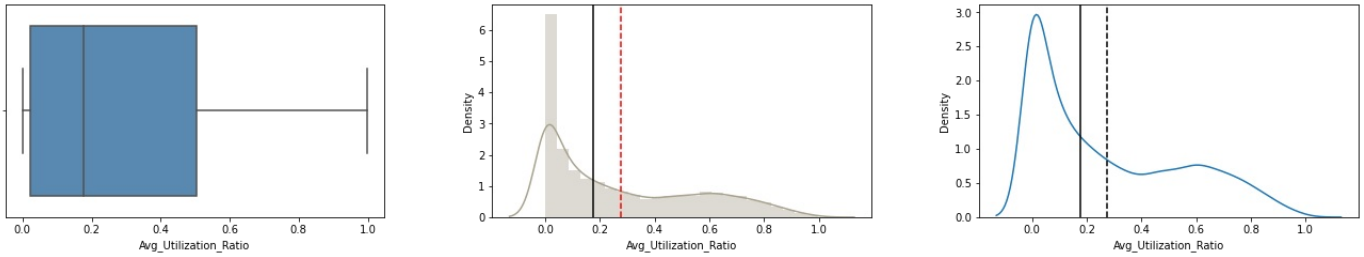


Observation:

1. There are many outliers visible.
2. The graph is skewed to the right.
3. This variable can be subjected to min, max scalar

Observations on Avg Utilization Ratio

In [28]:
```
histogram_boxplot(data['Avg_Utilization_Ratio'])
```



Observations:

1. No outliers visible.
2. mean of the avg utlization ratio is 0.27.

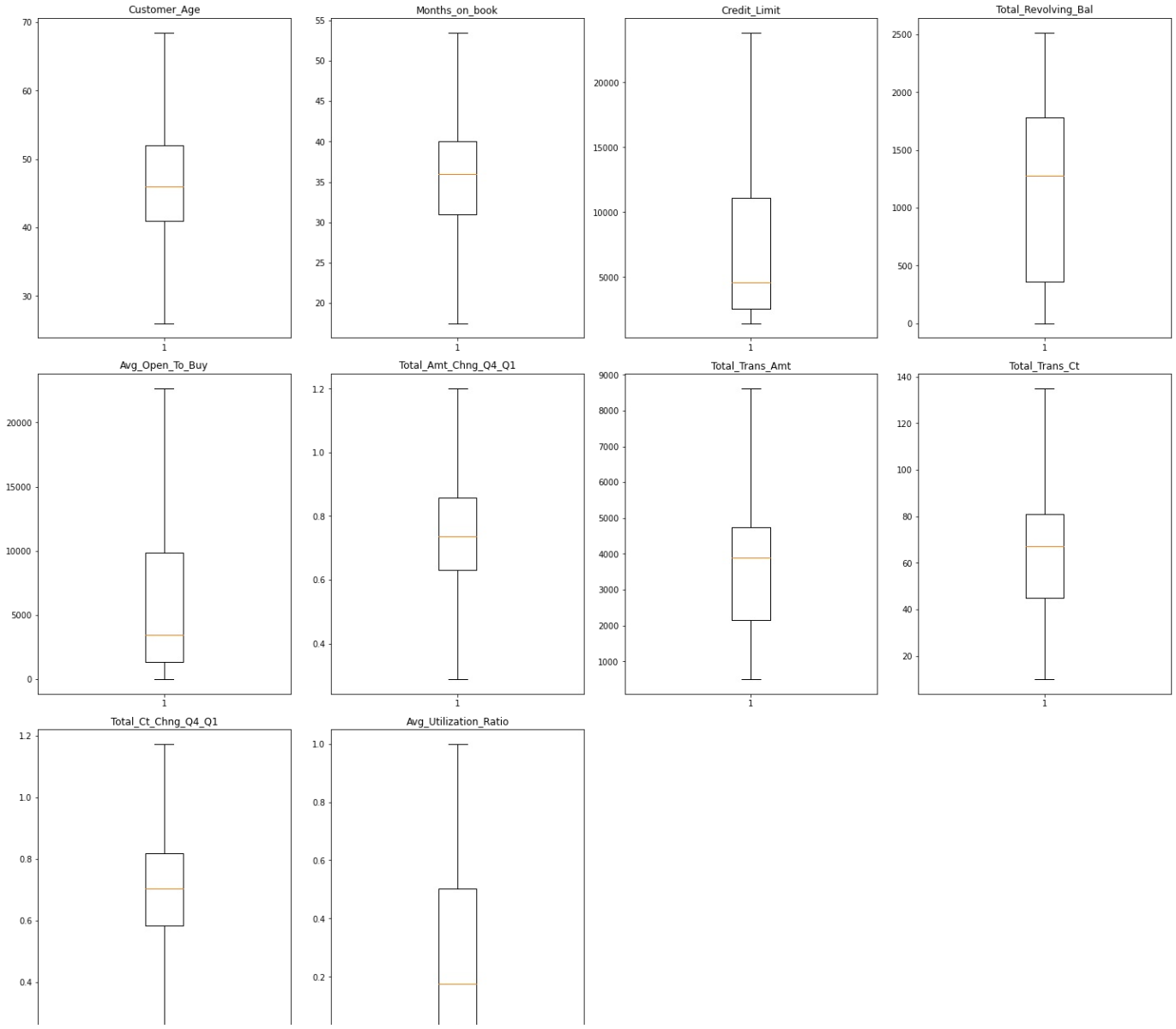
Outlier Treatment

In [29]:
```
# Let's treat outliers by flooring and capping
def treat_outliers(df, col):
    """
    treats outliers in a variable
    col: str, name of the numerical variable
```

```
    df: dataframe
    col: name of the column
    """
    Q1 = df[col].quantile(0.25)  # 25th quantile
    Q3 = df[col].quantile(0.75)  # 75th quantile
    IQR = Q3 - Q1
    Lower_Whisker = Q1 - 1.5 * IQR
    Upper_Whisker = Q3 + 1.5 * IQR

    # all the values smaller than Lower_Whisker will be assigned the value of Lower_Whisker
    # all the values greater than Upper_Whisker will be assigned the value of Upper_Whisker
    df[col] = np.clip(df[col], Lower_Whisker, Upper_Whisker)

    return df


def treat_outliers_all(df, col_list):
    """
    treat outlier in all numerical variables
    col_list: list of numerical variables
    df: data frame
    """
    for c in col_list:
        df = treat_outliers(df, c)

    return df
```

In [30]:
```
numerical_col = data.select_dtypes(include=np.number).columns.tolist() #converting numerical cols
data = treat_outliers_all(data, numerical_col) #treating outliers
```

In [31]:
```
plt.figure(figsize=(20, 30))

for i, variable in enumerate(numerical_col): #boxplots subplots
    plt.subplot(5, 4, i + 1)
    plt.boxplot(data[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```



Observations:

1. The outliers that were visible from the analysis of the continous vairables is no longer visible

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

In [32]:
```
def bar_perc(plot, feature):
    '''
    plot
    feature: 1-d categorical feature array
    '''
    total = len(feature) # length of the column
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total) # percentage of each class of the category
        x = p.get_x() + p.get_width() / 2 - 0.05 # width of the plot
        y = p.get_y() + p.get_height()           # hieght of the plot
        ax.annotate(percentage, (x, y), size = 12) # annotate the percantage
```

In [33]:
```
data.describe(include = 'category').T
```

Out[33]:

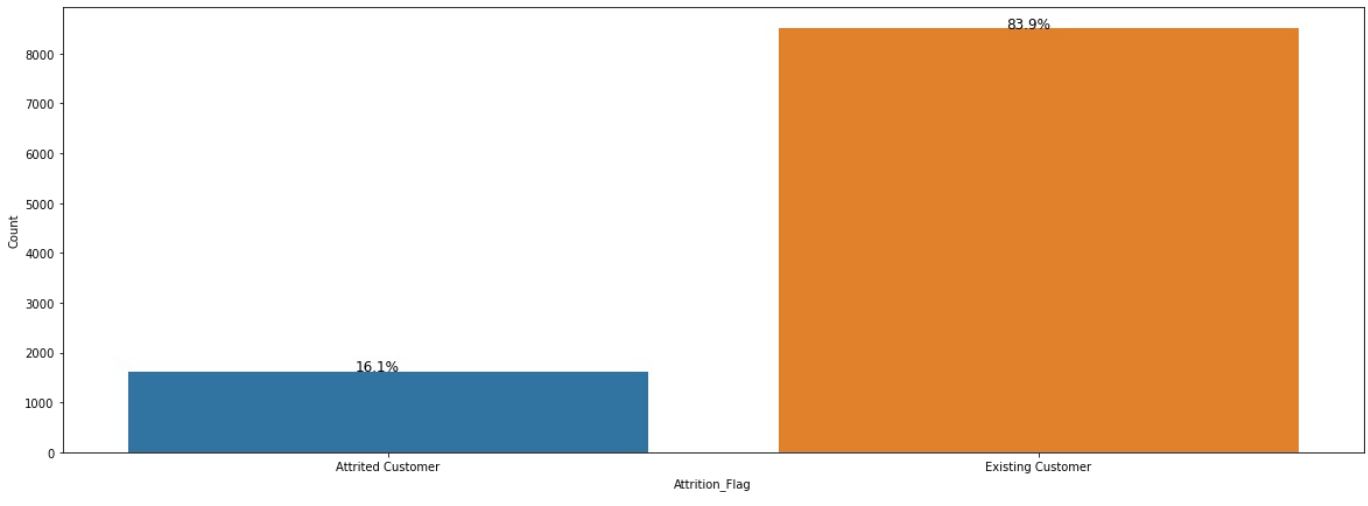
	count	unique	top	freq
Attrition_Flag	10127	2	Existing Customer	8500
Gender	10127	2	F	5358
Dependent_count	10127	6	3	2732
Education_Level	8608	6	Graduate	3128
Marital_Status	9378	3	Married	4687
Income_Category	9015	5	Less than $40K	3561
Card_Category	10127	4	Blue	9436
Total_Relationship_Count	10127	6	3	2305
Months_Inactive_12_mon	10127	7	3	3846
Contacts_Count_12_mon	10127	7	3	3380

Observations:

1. Existing customer is the most frequent.
2. Female customers are more frequent.
3. Most customers are graduate customers.
4. Married customers are more frequent.
5. Most customers use the blue card.
6. Total Relationship count, months inactive and contacts count are all 3.

Observations on Attrition FLag

In [34]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[0]]) #count plot for Name
plt.xlabel(cat_col[0])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[0]])
```

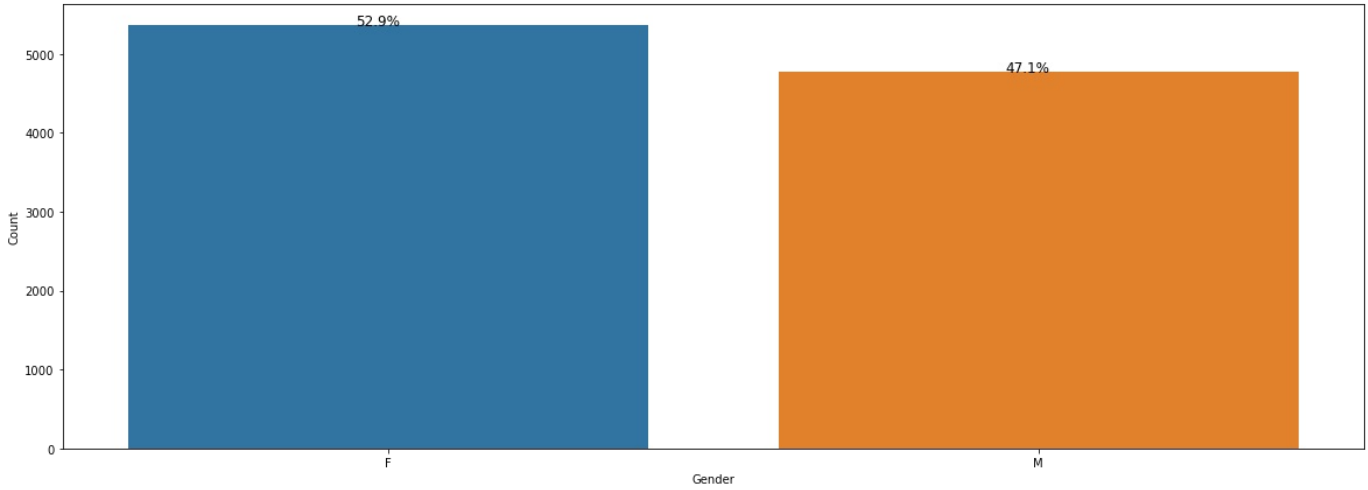


Observations:

1. Most common set of customers are existing customers with 83.9%
2. Attried Customers are 16.1%

Observations on Gender

In [35]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[1]]) #count plot for Name
plt.xlabel(cat_col[1])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[1]])
```

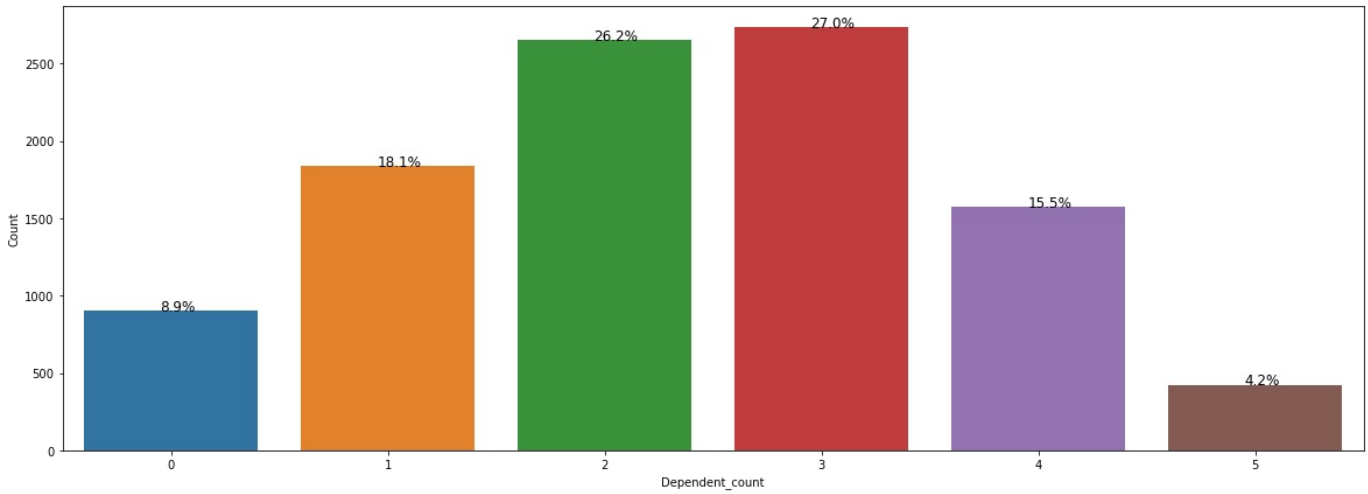


Observations:

1. Females are most common customers with 52.9%.
2. Males have 47.1% visibily

Observations on Dependent Count

In [36]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[2]]) #count plot for Name
plt.xlabel(cat_col[2])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[2]])
```

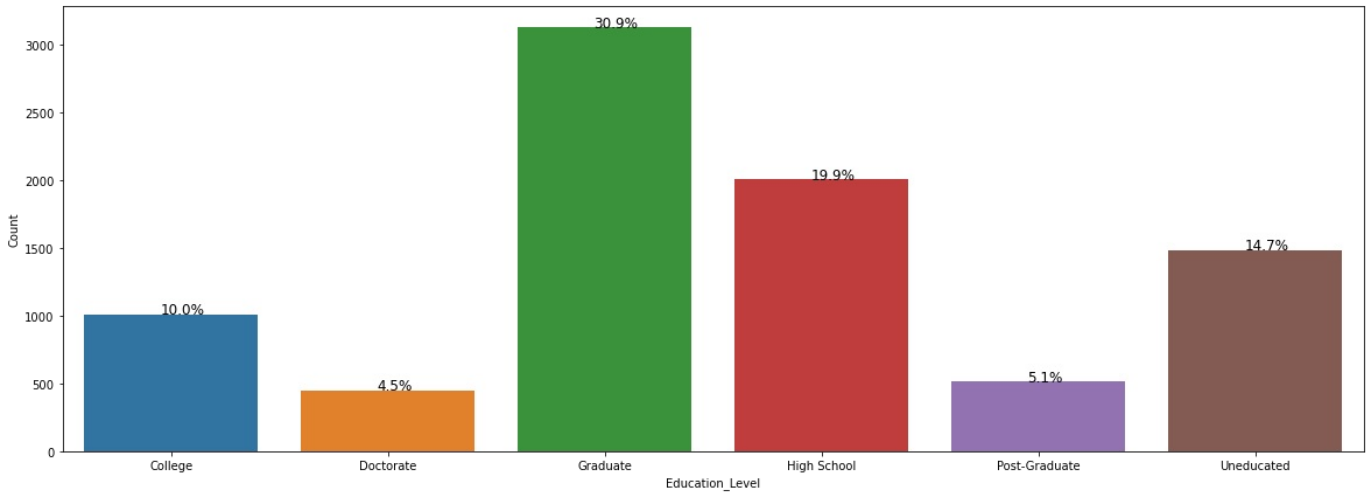


Observations:

1. 2 and 3 dependents are more common with 26.2% and 27.0%
2. Least common dependents are 5 ith 4.2%.

Observations on Education Level

In [37]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[3]]) #count plot for Name
plt.xlabel(cat_col[3])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[3]])
```

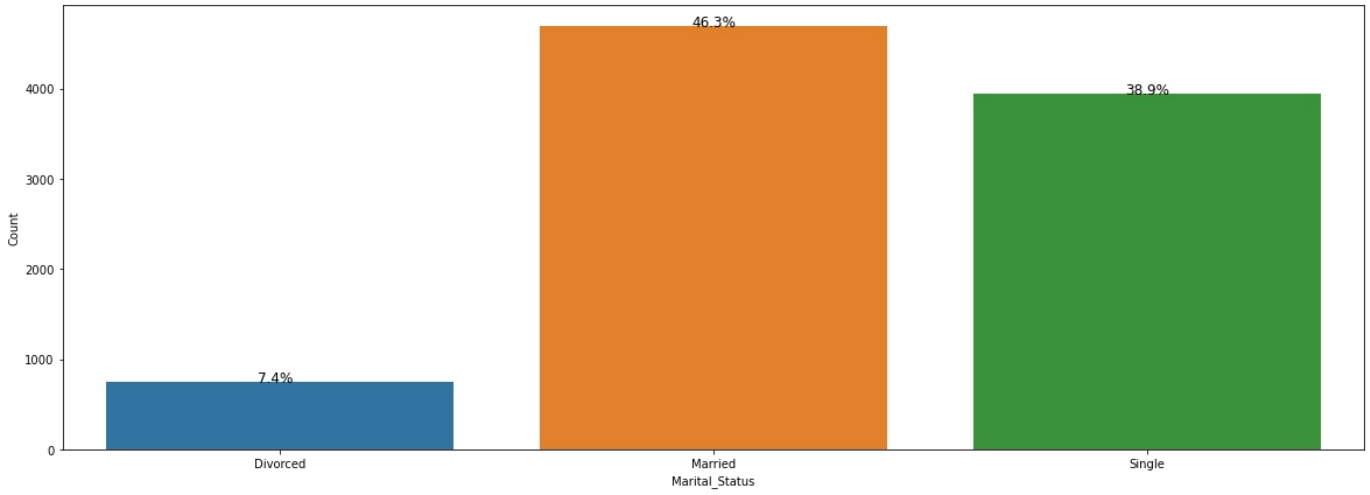


Observations:

1. Most common customers are graduate students with 30.9%
2. Least common are doctorates with 4.5% and post-graduates with 5.19%

Observations on Marital Status

In [38]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[4]]) #count plot for Name
plt.xlabel(cat_col[4])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[4]])
```

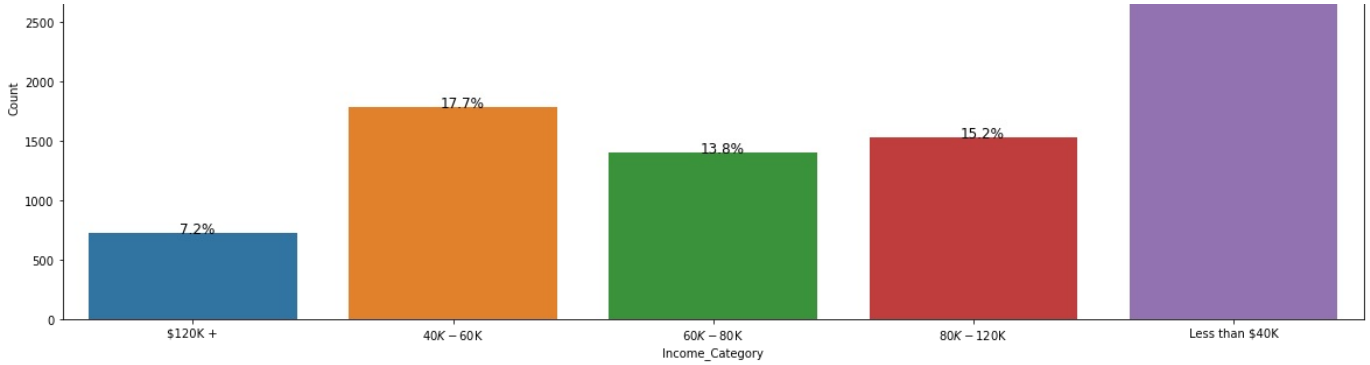


Observations:

1. Divorced are the least common customers with 7.4%.
2. Married customers are most common with 46.3% folloed by single customers with 38.9%.

Observations on Income Category

In [39]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[5]]) #count plot for Name
plt.xlabel(cat_col[5])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[5]])
```

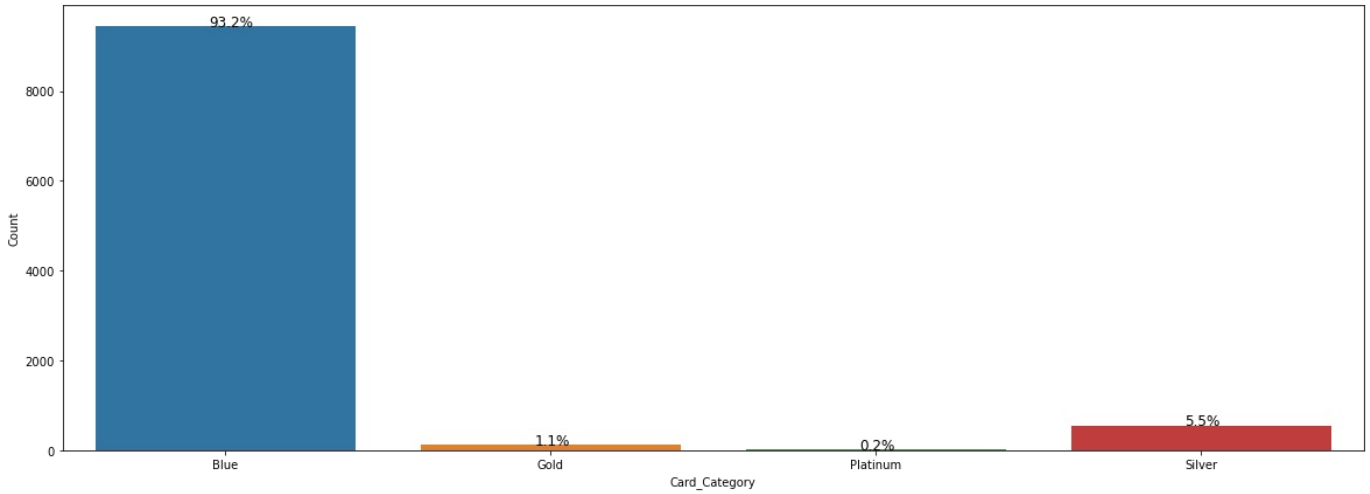


Observations:

1. Most common customers are those that make less than 40K.
2. Least common customers make more than $120K

Observations on Card Category

In [40]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[6]]) #count plot for Name
plt.xlabel(cat_col[6])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[6]])
```

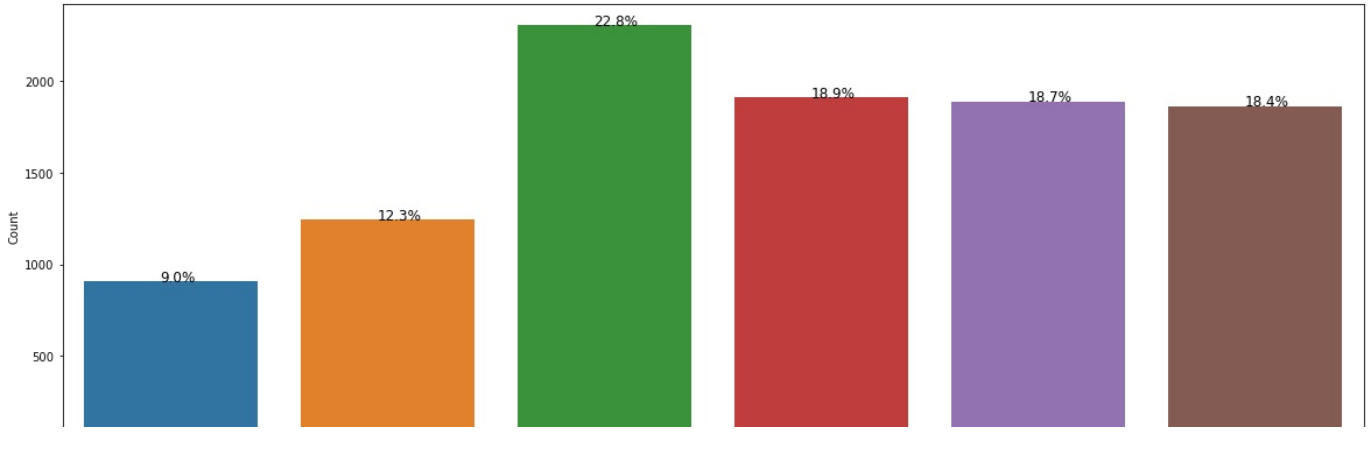


Observations:

1. Most common used card is blue with 93.2%
2. Least common used card type is platinum wiht 0.2% and gold with 1.1%.

Observations on Total Relationship Count

In [41]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[7]]) #count plot for Name
plt.xlabel(cat_col[7])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[7]])
```

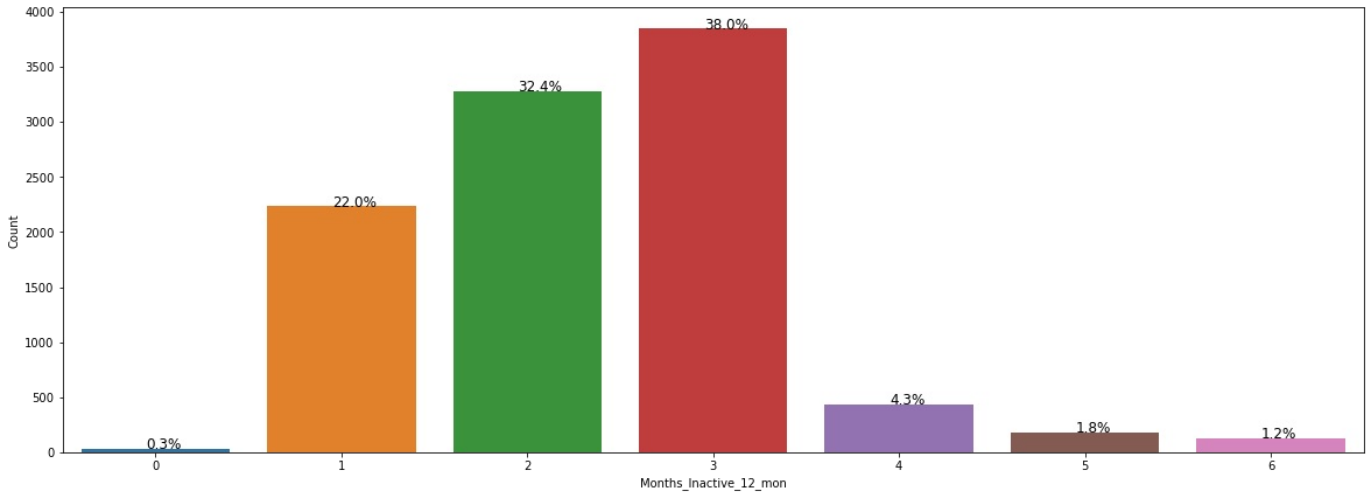


Observations:

1. 3 is the most total relationship count that is common.
2. Least common total relationship count is 1 with 9%.

Observations on Months inactive - 12 month period

In [42]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[8]]) #count plot for Name
plt.xlabel(cat_col[8])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[8]])
```

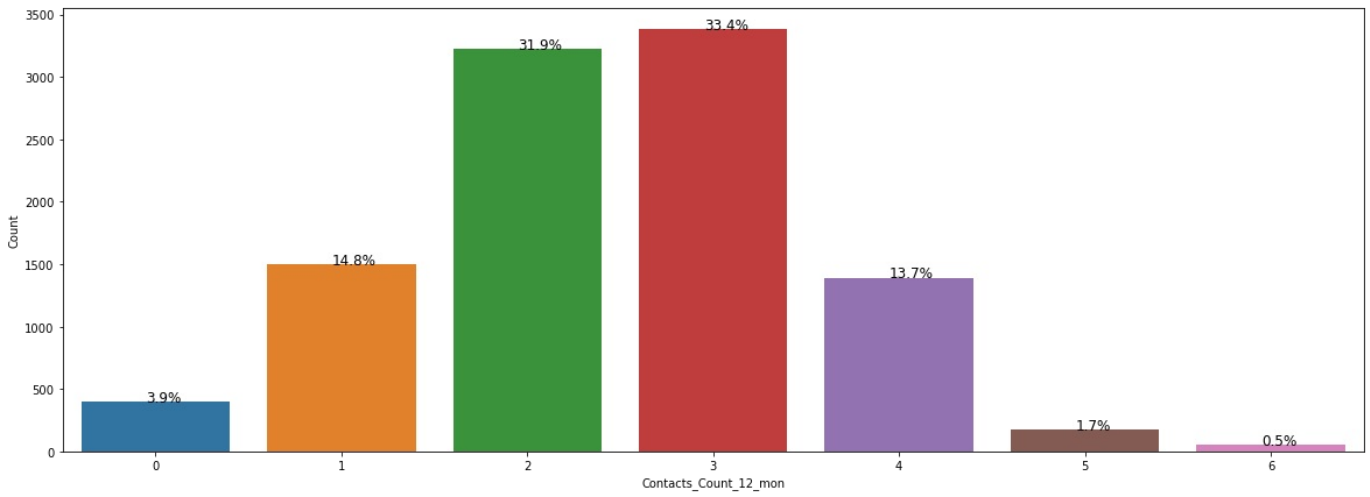


Observations:

1. 3 is the most common number of months inactive followed by 2 and 1.
2. Least number of months inactive is 0 months with 0.3%.

Observations on Conctacts count 12 month period

In [43]:
```
plt.figure(figsize=(20,7))
ax = sns.countplot(data[cat_col[9]]) #count plot for Name
plt.xlabel(cat_col[9])
plt.ylabel('Count')
bar_perc(ax,data[cat_col[9]])
```



Observations:

1. The most frequent contacts count is 3 followed by 2, 4 and 1 with 33.4%, 31.9%, 13.7% and 14.8% respectively.
2. Least frequent cotancts count is 6 with 0.5% and 0 with 3.9%.

Bivariate Analysis

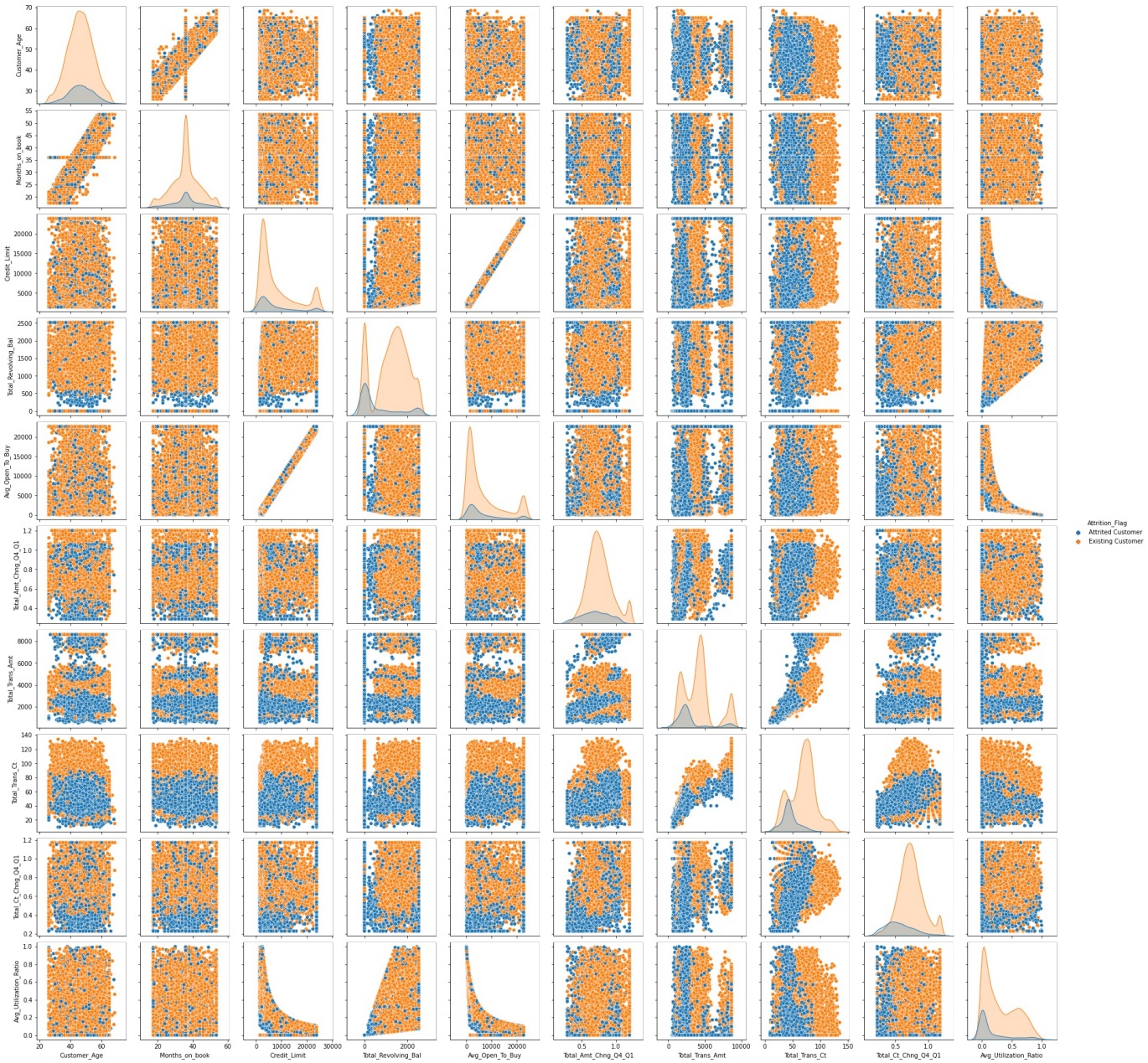
In [44]:
```
data.corr()
```

Out[44]:

	Customer_Age	Months_on_book	Credit_Limit	Total_Revolving_Bal	Avg_Open_To_Buy	Total_Amt_Chng_Q4_Q1	Total_
Customer_Age	1.000000	0.787989	0.002699	0.014778	0.000851	-0.070923	
Months_on_book	0.787989	1.000000	0.007977	0.008326	0.006756	-0.055963	
Credit_Limit	0.002699	0.007977	1.000000	0.046411	0.994238	0.015127	
Total_Revolving_Bal	0.014778	0.008326	0.046411	1.000000	-0.055839	0.051160	
Avg_Open_To_Buy	0.000851	0.006756	0.994238	-0.055839	1.000000	0.009522	
Total_Amt_Chng_Q4_Q1	-0.070923	-0.055963	0.015127	0.051160	0.009522	1.000000	
Total_Trans_Amt	-0.040467	-0.033587	0.139704	0.048189	0.135763	0.092683	
Total_Trans_Ct	-0.067010	-0.049957	0.068896	0.056055	0.064078	0.057027	
Total_Ct_Chng_Q4_Q1	-0.027871	-0.026631	-0.007484	0.093643	-0.017100	0.347408	
Avg_Utilization_Ratio	0.007108	-0.007056	-0.518439	0.624022	-0.587529	0.031923	

In [45]:
```
plt.figure(figsize=(30,7))
sns.pairplot(data = data, hue = 'Attrition_Flag')
```

Out[45]:
```
<seaborn.axisgrid.PairGrid at 0x1b75eb194f0>
<Figure size 2160x504 with 0 Axes>
```



Observations:

1. Customer Age and months on book have a positive correlation with 0.77 correlation, indicating a strong correlation.
2. Credit limit and Avgerage of credit limit have a near perfect correaltion with 0.99 which makes sense as these two variables are complementary to each other.
3. Avg open ration and Avg utilization ratio have strong negative correatlion of -0.6.
4. Total trans CT and total trans Amt have strong correlation of 0.86, which makes sense as the number of transactions goes up the amount of amount taken will likely increase, over a duration of period. 5.Average utilization ratio and Total revolving balance have strong correlation with 0.7.

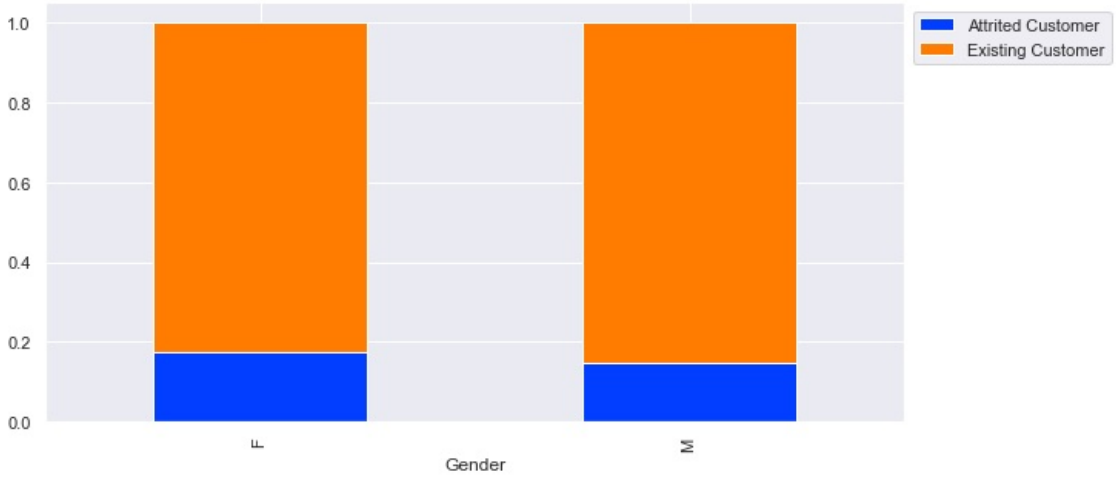
Let's define one more function to plot stacked bar charts

In [46]:
```
### Function to plot stacked bar charts for categorical columns
def stacked_plot(x,flag=True):
    sns.set(palette='bright')
    tab1 = pd.crosstab(x,data['Attrition_Flag'],margins=True)
    if flag==True:
        print(tab1)
        print('-'*120)
    tab = pd.crosstab(x,data['Attrition_Flag'],normalize='index')
    tab.plot(kind='bar',stacked=True,figsize=(10,5))
    plt.legend(loc='lower left', frameon=False)
    plt.legend(loc="upper left", bbox_to_anchor=(1,1))
    plt.show()
```

Attrition FLag vs Gender

In [47]:
```
stacked_plot(data[cat_col[1]])
```

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



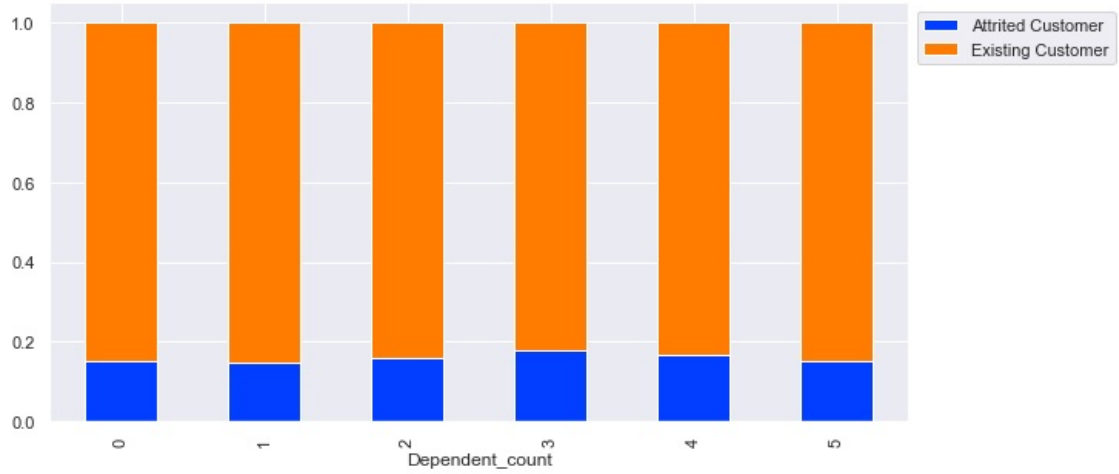
Observations:

1. The distribution between M and F attried customer vs Existing customer is about the same ratio

Attrition Flag vs Dependent Count

In [48]:
```
stacked_plot(data[cat_col[2]])
```

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



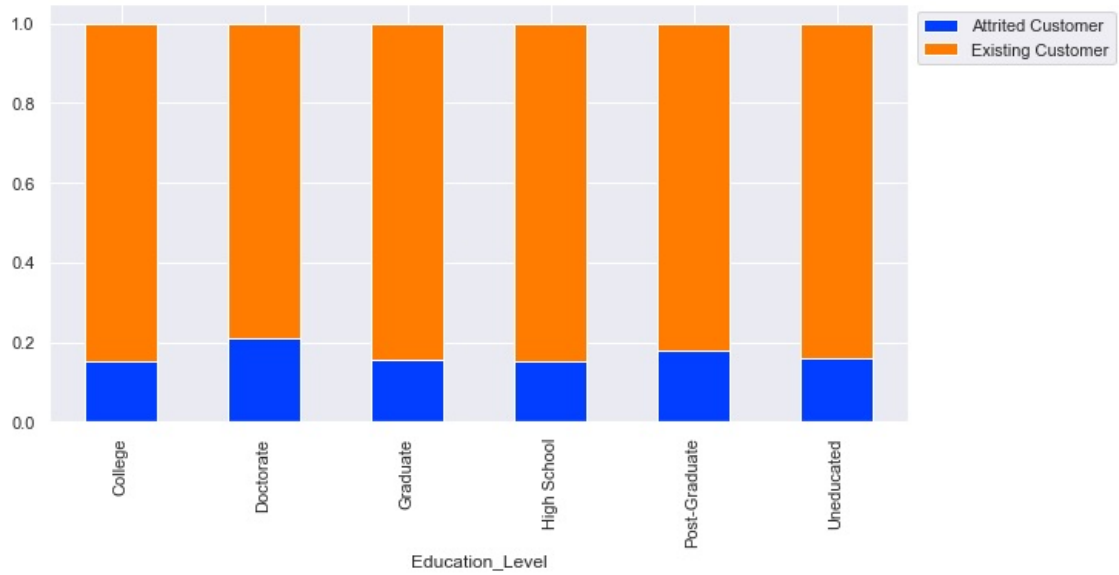
Observations:

1. The ratio of attried customer to existing customer is most same accros each value of dependent counts

Attrition Flag vs Education Level

In [49]:
```
stacked_plot(data[cat_col[3]])
```

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



Observations:

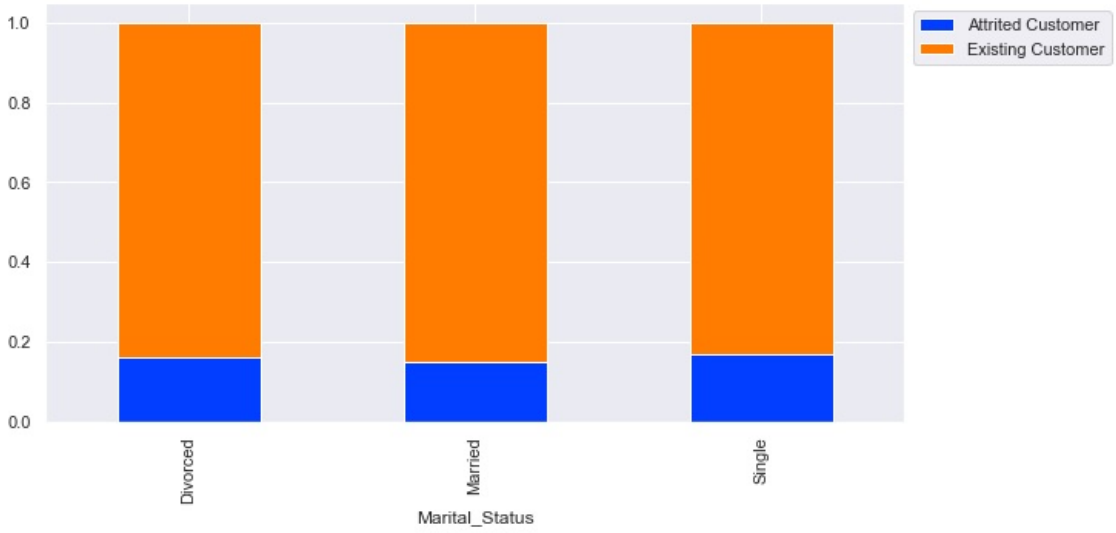
1. The ratio fo attried customer to existing customer is maintained equally among all education levels.
2. There are more existing customers than attried customers accross all education levels.

Attrition Flag vs Martial Status

In [50]:
```
stacked_plot(data[cat_col[4]])
```

```
Attrition_Flag  Attrited Customer  Existing Customer   All
Marital_Status
```

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



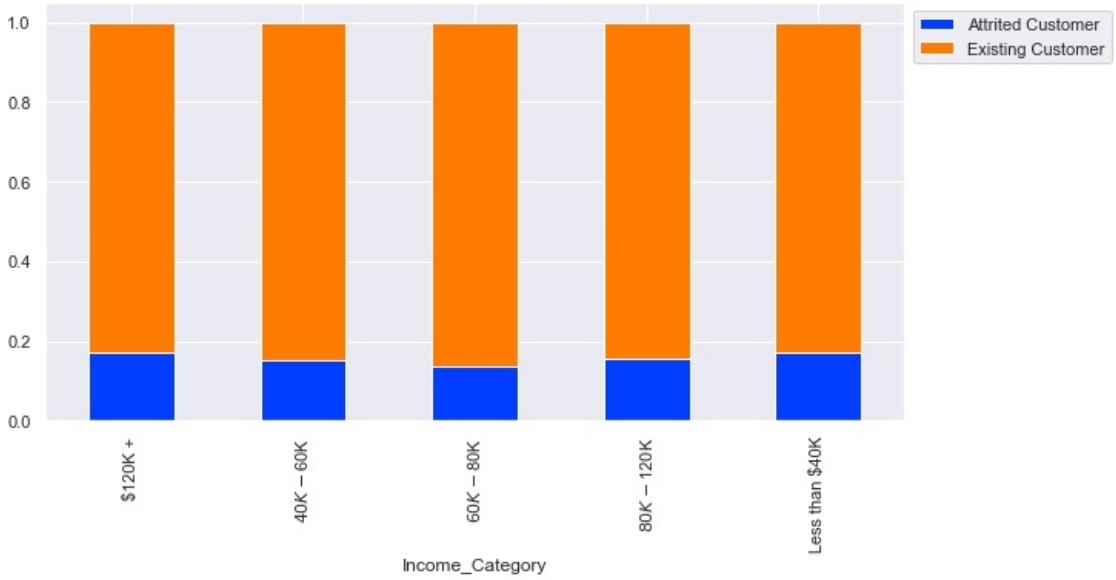
Observations:

1. There are more existing customers than attried customers accross all martial status.
2. The existing customer to attried customer is maintained uniform accorss all martial status.

Attrition flag vs Income Category

In [51]:
```
stacked_plot(data[cat_col[5]])
```

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



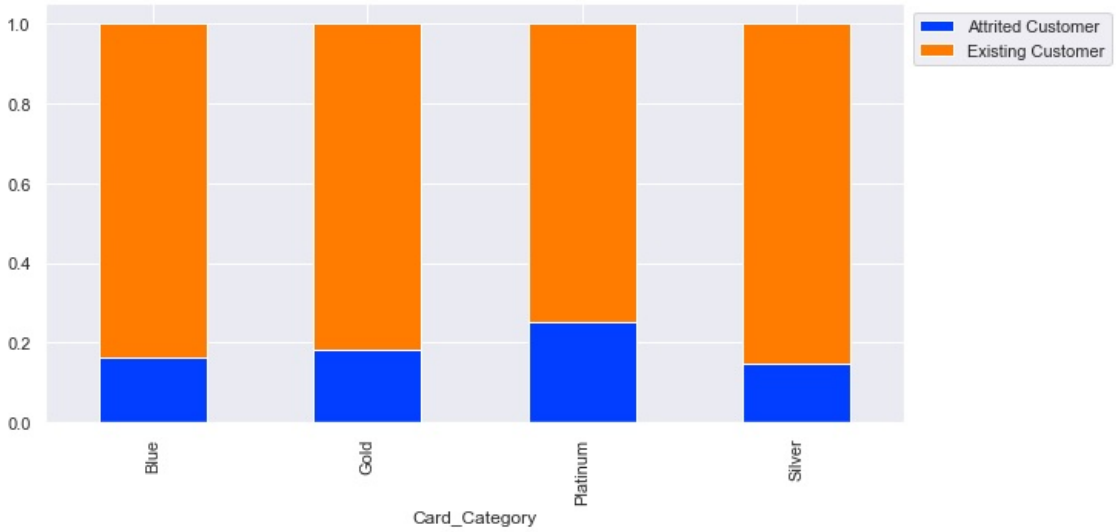
Observation:

1. The attried customer to existing customer ratio is uniformly mainted against all categories.
2. Existing customers are more prelevant compared to attried customers.

Attrition Flag vs Card Category

In [52]:
```
stacked_plot(data[cat_col[6]])
```

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



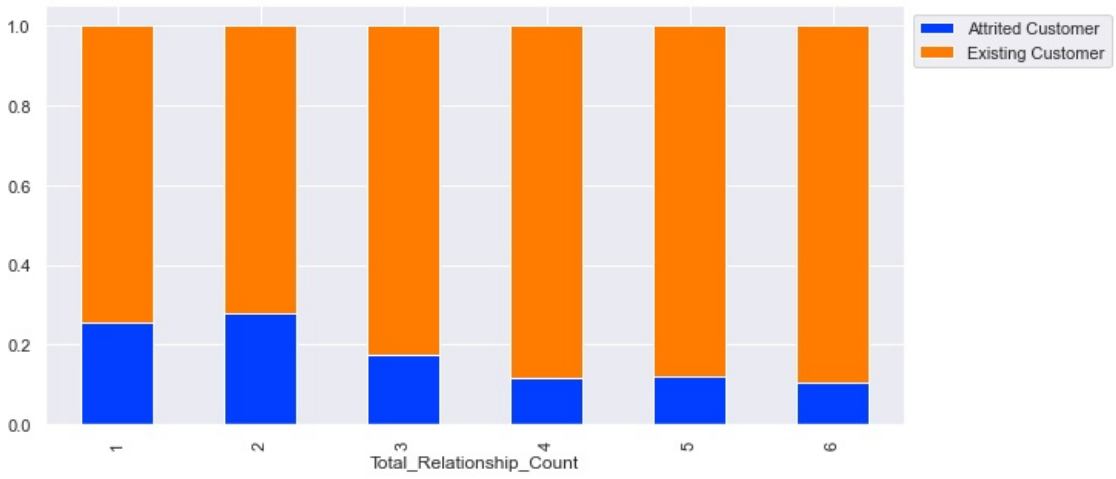
Observations:

1. The ratio of exisitng to attried customers is around 0.8 to 0.2.
2. More existing cusomters compared to attried customers

Attrition Flag vs Total Relationship Count

In [53]:
```
stacked_plot(data[cat_col[7]])
```

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



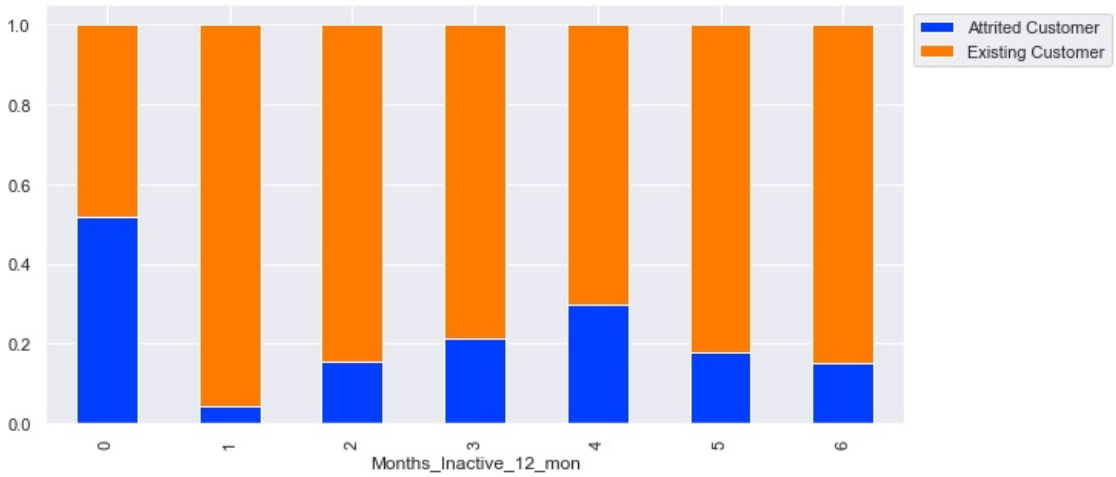
Observations:

1. The ratio of attried customers to existing customers is around 0.75 to 0.25
2. There are more exisitng customers compared to attried customers.

Attrition Flag vs Months Inactive 12 month period

In [54]:
```
stacked_plot(data[cat_col[8]])
```

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



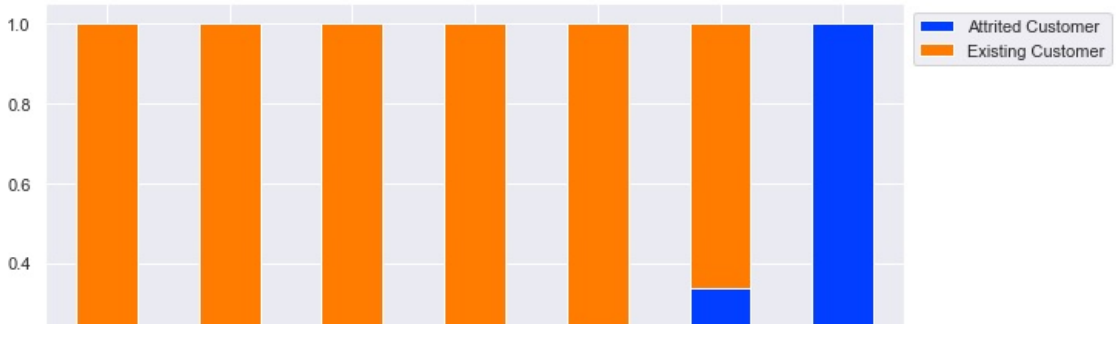
Observations:

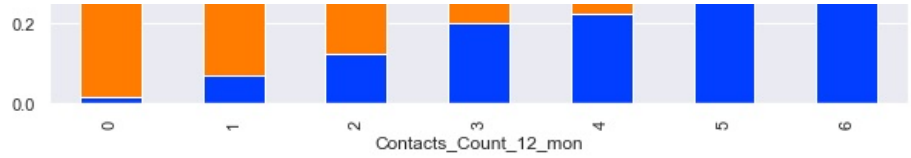
1. There are equal distubtion of attired customes to existings customers at 0.5 each at 0 months inactive in a 12 month period.
2. There are significatly more existing customers than attrted customers at 1 month inactive.

Attrition Flag vs Contacts count 12 month period

In [55]:
```
stacked_plot(data[cat_col[9]])
```

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



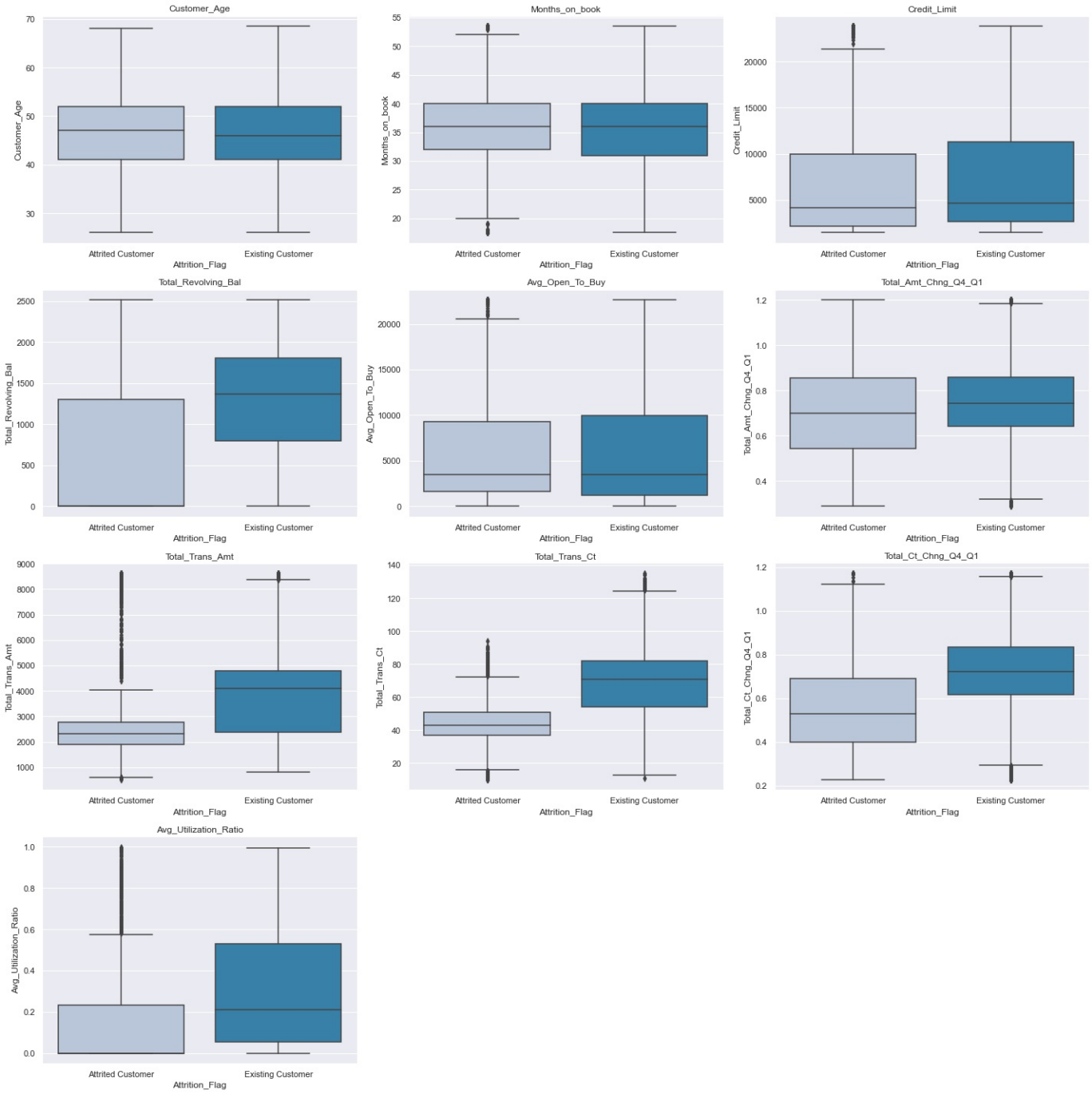


Observations:

1. There is nearly only existing customers in 0 contact months.
2. While there are only attritied customers in 6 contact months.
3. The ratio of attried customers to existings customers increase from 1 cotacts count to 5.

Attrition Flag vs Continous Variables

In [56]:
```
cols = data[['Customer_Age', 'Months_on_book', 'Credit_Limit', 'Total_Revolving_Bal', 'Avg_Open_To_Buy', 'Total_A
plt.figure(figsize=(20,20))

for i, variable in enumerate(cols):
                    plt.subplot(4,3,i+1)
                    sns.boxplot(data['Attrition_Flag'], data[variable],palette="PuBu")
                    plt.tight_layout()
                    plt.title(variable)
plt.show()
```



Observations:

1. For customer age on average the age remains same accross attrited and existing customers.
2. Months on book appears on average be equal for attritied and existing customers.
3. For Total transmited amount, ct and CT chang between Q4 and Q1 exising customers are higher than attrited customers on average.

In [57]:
```
data[(data['Attrition_Flag']== 'Existing Customer')].describe(include='all').T
```

Out[57]:

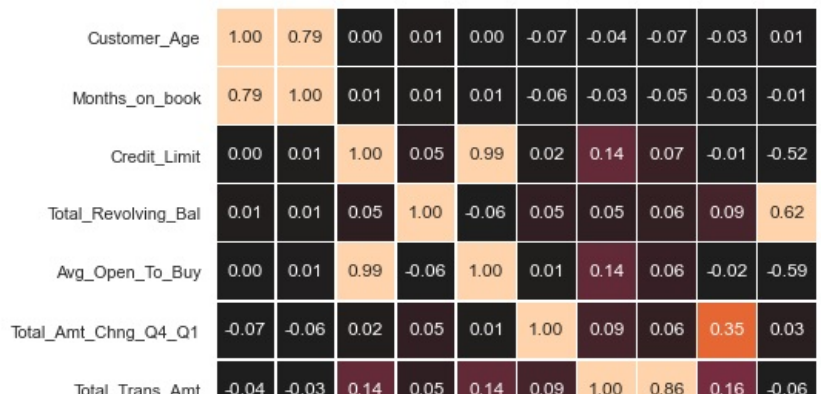
	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Attrition_Flag	8500	1	Existing Customer	8500	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Customer_Age	8500	NaN	NaN	NaN	46.2614	8.07905	26	41	46	52	68.5
Gender	8500	2	F	4428	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Dependent_count	8500	6	3	2250	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Education_Level	7237	6	Graduate	2641	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Marital_Status	7880	3	Married	3978	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Income_Category	7575	5	Less than $40K	2949	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Card_Category	8500	4	Blue	7917	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Months_on_book	8500	NaN	NaN	NaN	35.8994	7.80276	17.5	31	36	40	53.5
Total_Relationship_Count	8500	6	3	1905	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Months_Inactive_12_mon	8500	7	3	3020	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Contacts_Count_12_mon	8500	6	2	2824	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Credit_Limit	8500	NaN	NaN	NaN	7981.04	7234.72	1438.3	2602	4643.5	11252.8	23836.2
Total_Revolving_Bal	8500	NaN	NaN	NaN	1256.6	757.745	0	800	1364	1807	2517
Avg_Open_To_Buy	8500	NaN	NaN	NaN	6735.48	7264.9	15	1184.5	3469.5	9978.25	22660.8
Total_Amt_Chng_Q4_Q1	8500	NaN	NaN	NaN	0.761765	0.178332	0.289	0.643	0.743	0.86	1.201
Total_Trans_Amt	8500	NaN	NaN	NaN	4118.34	2109.58	816	2384.75	4100	4781.25	8619.25
Total_Trans_Ct	8500	NaN	NaN	NaN	68.6718	22.9166	11	54	71	82	135
Total_Ct_Chng_Q4_Q1	8500	NaN	NaN	NaN	0.731716	0.18199	0.228	0.617	0.721	0.833	1.172
Avg_Utilization_Ratio	8500	NaN	NaN	NaN	0.296412	0.272568	0	0.055	0.211	0.52925	0.994

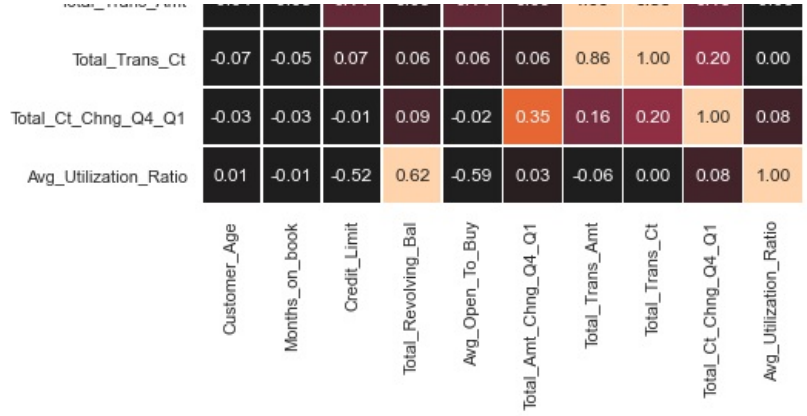
Observations on customer profile for Existing customers:

1. On average the customer age is around 46 years old.
2. Females are prelavant
3. Number of dependents are about 3.
4. Graduate, married and thsoe who make below 40K are visible.
5. Blue card is mostly used, with the total relationship count, months inactiv and contacts count being aroun 3.
6. Credit limit on average is arbout $8000. 7.The total amount changed and total Ct changed is about 0.73
7. Average amount left on credit is about 6735.

Correlation Heatmap

In [58]:
```
sns.set(rc={'figure.figsize':(7,7)})
sns.heatmap(data.corr(),
            annot=True,
            linewidths=.5,
            center=0,
            cbar=False,
            vmin=0, vmax=0.5,
            fmt='0.2f')
plt.show()
```





Observations:

1. Customer Age and months on book are positivly correlated with 0.79, this makes sense as the longer one has an account the older the person will be.
2. Credit limit as near perfect correaltion with Avg Open to Buy at 0.99
3. Total Revolving balance and Average utlization ratio have about 0.62 correlation.
4. Total trans ct and Total trans amount have 0.86 correlation as the number of transactions increase the overall total amount transacted should also be big.

Splitting the dataset into Train, Valdiation and Test set

In [59]:
```
df = data.copy() #copying the dataset
```

In [60]:
```
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  category
 1   Customer_Age              10127 non-null  float64
 2   Gender                    10127 non-null  category
 3   Dependent_count           10127 non-null  category
 4   Education_Level           8608 non-null   category
 5   Marital_Status            9378 non-null   category
 6   Income_Category           9015 non-null   category
 7   Card_Category             10127 non-null  category
 8   Months_on_book            10127 non-null  float64
 9   Total_Relationship_Count  10127 non-null  category
 10  Months_Inactive_12_mon    10127 non-null  category
 11  Contacts_Count_12_mon     10127 non-null  category
 12  Credit_Limit              10127 non-null  float64
 13  Total_Revolving_Bal       10127 non-null  int64
 14  Avg_Open_To_Buy           10127 non-null  float64
 15  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 16  Total_Trans_Amt           10127 non-null  float64
 17  Total_Trans_Ct            10127 non-null  int64
 18  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 19  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: category(10), float64(8), int64(2)
memory usage: 892.2 KB
```

In [61]:
```
df.head()
```

Out[61]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book
0	Existing Customer	45.0	M	3	High School	Married	$60K-80K$	Blue	39.0
1	Existing Customer	49.0	F	5	Graduate	Single	Less than $40K	Blue	44.0
2	Existing Customer	51.0	M	3	Graduate	Married	$80K-120K$	Blue	36.0
3	Existing Customer	40.0	F	4	High School	NaN	Less than $40K	Blue	34.0
4	Existing Customer	40.0	M	3	Uneducated	Married	$60K-80K$	Blue	21.0

In [62]:
```
for i in cat_col:
    print(data[i].value_counts()) #printing the unique values of categorical variables
    print('-'*50)
```

```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, 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
$120K +            727
Name: Income_Category, dtype: int64
--------------------------------------------------
Blue        9436
Silver       555
Gold         116
Platinum      20
Name: Card_Category, 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
--------------------------------------------------
```

In [63]:
```
df.isnull().sum()
```

Out[63]:
```
Attrition_Flag                0
Customer_Age                  0
Gender                        0
Dependent_count               0
Education_Level            1519
```

```
Marital_Status               749
Income_Category             1112
Card_Category                  0
Months_on_book                 0
Total_Relationship_Count       0
Months_Inactive_12_mon         0
Contacts_Count_12_mon          0
Credit_Limit                   0
Total_Revolving_Bal            0
Avg_Open_To_Buy                0
Total_Amt_Chng_Q4_Q1           0
Total_Trans_Amt                0
Total_Trans_Ct                 0
Total_Ct_Chng_Q4_Q1            0
Avg_Utilization_Ratio          0
dtype: int64
```

Missing-Value Treatment

- We will use KNN imputer to impute missing values.
- `KNNImputer` : Each sample's missing values are imputed by looking at the n_neighbors nearest neighbors found in the training set. Default value for n_neighbors=5.
- KNN imputer replaces missing values using the average of k nearest non-missing feature values.
- Nearest points are found based on euclidean distance.

In [64]:
```
imputer = KNNImputer(n_neighbors=5) #imputer
```

In [65]:
```
# defining a list with names of columns that will be used for imputation
reqd_col_for_impute = [
    "Education_Level",
    "Marital_Status",
    "Income_Category",
]
df[reqd_col_for_impute].head()
```

Out[65]:

	Education_Level	Marital_Status	Income_Category
0	High School	Married	$60K-80K$
1	Graduate	Single	Less than $40K
2	Graduate	Married	$80K-120K$
3	High School	NaN	Less than $40K
4	Uneducated	Married	$60K-80K$

In [66]:
```
# we need to pass numerical values for each categorical column for KNN imputation so we will label encode them
Gender = {"F": 0, "M": 1}
df["Gender"] = df["Gender"].map(Gender)

Education_Level= {"Uneducated": 0, "High School": 1, "College": 2, "Post-Graduate": 3, "Graduate" : 4, "Doctorate
df["Education_Level"] = df["Education_Level"].map(Education_Level)

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

Income_Category = {
    "Less than $40K": 0,
    "$40K - $60K": 1,
    "$60K - $80K": 2,
    "$80K - $120K": 3,
    "$120K +": 4,
}
df["Income_Category"] = df["Income_Category"].map(Income_Category)

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

In [67]:
```
df.head()
```

Out[67]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book
0	Existing Customer	45.0	1	3	1	1	2	0	39.0
1	Existing Customer	49.0	0	5	4	0	0	0	44.0
2	Existing Customer	51.0	1	3	4	1	3	0	36.0
3	Existing Customer	40.0	0	4	1	NaN	0	0	34.0
4	Existing Customer	40.0	1	3	0	1	2	0	21.0

In [68]:
```
X = df.drop(["Attrition_Flag"], axis=1)
y = df["Attrition_Flag"].apply(lambda x: 1 if x == "Existing Customer" else 0)
```

In [69]:
```
# Splitting data into training, validation and test set:
# first we split data into 2 parts, say temporary and test

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

# then we split the temporary set into train and validation

X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, random_state=1, stratify=y_temp
)
print(X_train.shape, X_val.shape, X_test.shape)
```

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

In [70]:
```
print("Number of rows in train data =", X_train.shape[0])
print("Number of rows in validation data =", X_val.shape[0])
print("Number of rows in test data =", X_test.shape[0])
```

```
Number of rows in train data = 6075
Number of rows in validation data = 2026
Number of rows in test data = 2026
```

Imputing Missing Values

In [71]:
```
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  category
 1   Customer_Age              10127 non-null  float64
 2   Gender                    10127 non-null  category
 3   Dependent_count           10127 non-null  category
 4   Education_Level           8608 non-null   category
 5   Marital_Status            9378 non-null   category
 6   Income_Category           9015 non-null   category
 7   Card_Category             10127 non-null  category
 8   Months_on_book            10127 non-null  float64
 9   Total_Relationship_Count  10127 non-null  category
 10  Months_Inactive_12_mon    10127 non-null  category
 11  Contacts_Count_12_mon     10127 non-null  category
 12  Credit_Limit              10127 non-null  float64
 13  Total_Revolving_Bal       10127 non-null  int64
 14  Avg_Open_To_Buy           10127 non-null  float64
 15  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 16  Total_Trans_Amt           10127 non-null  float64
 17  Total_Trans_Ct            10127 non-null  int64
 18  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 19  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: category(10), float64(8), int64(2)
memory usage: 892.2 KB
```

In [72]:
```
# Fit and transform the train data
X_train[reqd_col_for_impute] = imputer.fit_transform(X_train[reqd_col_for_impute])
```

```
# Transform the train data
X_val[reqd_col_for_impute] = imputer.fit_transform(X_val[reqd_col_for_impute])

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

In [73]:
```
# Checking that no column has missing values in train, validation 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 [74]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6075 entries, 9501 to 703
Data columns (total 19 columns):
```

```
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Customer_Age             6075 non-null   float64
 1   Gender                   6075 non-null   category
 2   Dependent_count          6075 non-null   category
 3   Education_Level          6075 non-null   float64
 4   Marital_Status           6075 non-null   float64
 5   Income_Category          6075 non-null   float64
 6   Card_Category            6075 non-null   category
 7   Months_on_book           6075 non-null   float64
 8   Total_Relationship_Count  6075 non-null   category
 9   Months_Inactive_12_mon   6075 non-null   category
 10  Contacts_Count_12_mon    6075 non-null   category
 11  Credit_Limit             6075 non-null   float64
 12  Total_Revolving_Bal      6075 non-null   int64
 13  Avg_Open_To_Buy          6075 non-null   float64
 14  Total_Amt_Chng_Q4_Q1     6075 non-null   float64
 15  Total_Trans_Amt          6075 non-null   float64
 16  Total_Trans_Ct           6075 non-null   int64
 17  Total_Ct_Chng_Q4_Q1      6075 non-null   float64
 18  Avg_Utilization_Ratio    6075 non-null   float64
dtypes: category(6), float64(11), int64(2)
memory usage: 701.5 KB
```

In [75]:
```
cat_col.remove('Attrition_Flag')
```

In [76]:
```
X_train = pd.get_dummies(X_train, columns = cat_col, drop_first=True)#creating dummy variables
X_val = pd.get_dummies(X_val,columns = cat_col, drop_first=True)
X_test = pd.get_dummies(X_test,columns = cat_col, drop_first=True)
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(6075, 50) (2026, 50) (2026, 50)
```

In [77]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6075 entries, 9501 to 703
Data columns (total 50 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Customer_Age               6075 non-null   float64
 1   Months_on_book             6075 non-null   float64
 2   Credit_Limit               6075 non-null   float64
 3   Total_Revolving_Bal        6075 non-null   int64
 4   Avg_Open_To_Buy            6075 non-null   float64
 5   Total_Amt_Chng_Q4_Q1       6075 non-null   float64
 6   Total_Trans_Amt            6075 non-null   float64
 7   Total_Trans_Ct             6075 non-null   int64
 8   Total_Ct_Chng_Q4_Q1        6075 non-null   float64
 9   Avg_Utilization_Ratio      6075 non-null   float64
 10  Gender_1                   6075 non-null   uint8
 11  Dependent_count_1          6075 non-null   uint8
 12  Dependent_count_2          6075 non-null   uint8
 13  Dependent_count_3          6075 non-null   uint8
 14  Dependent_count_4          6075 non-null   uint8
 15  Dependent_count_5          6075 non-null   uint8
 16  Education_Level_0.0        6075 non-null   uint8
 17  Education_Level_1.0        6075 non-null   uint8
 18  Education_Level_2.0        6075 non-null   uint8
 19  Education_Level_3.0        6075 non-null   uint8
 20  Education_Level_4.0        6075 non-null   uint8
 21  Education_Level_5.0        6075 non-null   uint8
 22  Marital_Status_0.0         6075 non-null   uint8
 23  Marital_Status_1.0         6075 non-null   uint8
 24  Marital_Status_2.0         6075 non-null   uint8
 25  Income_Category_0.0        6075 non-null   uint8
 26  Income_Category_1.0        6075 non-null   uint8
 27  Income_Category_2.0        6075 non-null   uint8
 28  Income_Category_3.0        6075 non-null   uint8
 29  Income_Category_4.0        6075 non-null   uint8
 30  Card_Category_2            6075 non-null   uint8
 31  Card_Category_3            6075 non-null   uint8
 32  Card_Category_1            6075 non-null   uint8
 33  Total_Relationship_Count_2  6075 non-null   uint8
 34  Total_Relationship_Count_3  6075 non-null   uint8
 35  Total_Relationship_Count_4  6075 non-null   uint8
 36  Total_Relationship_Count_5  6075 non-null   uint8
 37  Total_Relationship_Count_6  6075 non-null   uint8
 38  Months_Inactive_12_mon_1   6075 non-null   uint8
```

```
 39  Months_Inactive_12_mon_2    6075 non-null   uint8
 40  Months_Inactive_12_mon_3    6075 non-null   uint8
 41  Months_Inactive_12_mon_4    6075 non-null   uint8
 42  Months_Inactive_12_mon_5    6075 non-null   uint8
 43  Months_Inactive_12_mon_6    6075 non-null   uint8
 44  Contacts_Count_12_mon_1     6075 non-null   uint8
 45  Contacts_Count_12_mon_2     6075 non-null   uint8
 46  Contacts_Count_12_mon_3     6075 non-null   uint8
 47  Contacts_Count_12_mon_4     6075 non-null   uint8
 48  Contacts_Count_12_mon_5     6075 non-null   uint8
 49  Contacts_Count_12_mon_6     6075 non-null   uint8
dtypes: float64(8), int64(2), uint8(40)
memory usage: 759.4 KB
```

In [78]:
```
y_train.value_counts()
```

Out[78]:
```
1    5099
0     976
Name: Attrition_Flag, dtype: int64
```

Model evaluation criterion

We will be using Recall as a metric for our model performance because here company could face 2 types of losses

- Company wants recall to be maximized i.e. we need to reduce the number of false negatives.

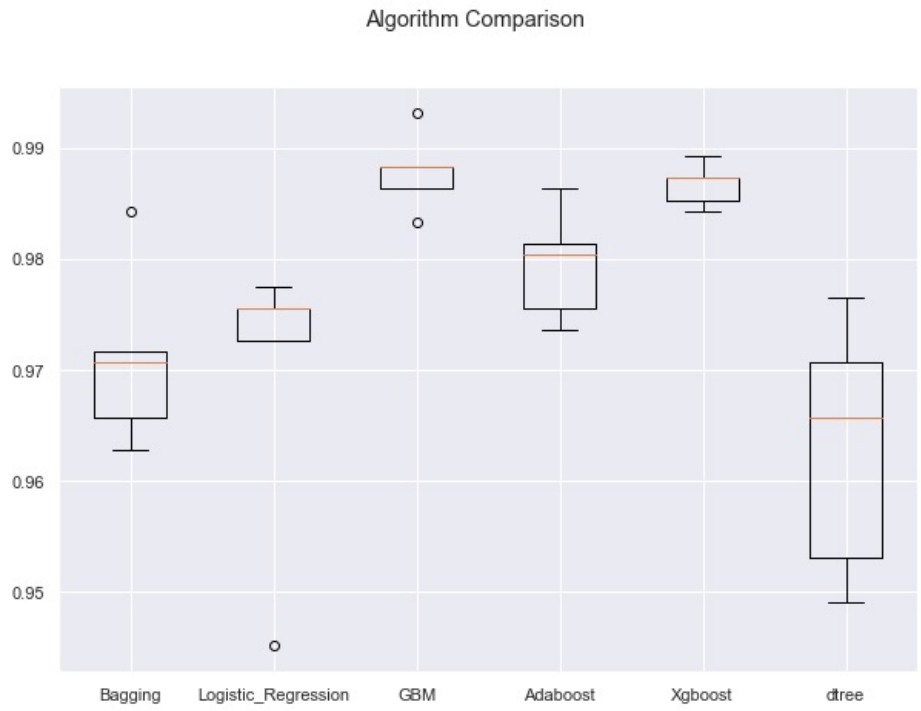
For model building we will be first building 6 models from Logistic regression, Bagging, GBM, AdaBoost, XGBoost and decision tree, we will run this with K-cross validation and then on the validation data set and we will measure the recall score of each of the models. From the 6 models, we will choose the 3 best models with the best recall score on the validation set.

In [79]:
```
models = []  # Empty list to store all the models

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

results = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models
score = []
# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation Performance:" "\n")
for name, model in models:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Setting number of splits equal to 5
    cv_result = cross_val_score(
        estimator=model, X=X_train, y=y_train, scoring=scoring, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean() * 100))

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

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

```
Cross-Validation Performance:

Bagging: 97.0977313398372
Logistic_Regression: 96.92102984471511
GBM: 98.78408281860341
Adaboost: 97.94081086801748
Xgboost: 98.66637803305818
dtree: 96.29365583328521

Validation Performance:

Bagging: 0.9788235294117648
```

```
Logistic_Regression: 0.97
GBM: 0.9894117647058823
Adaboost: 0.9823529411764705
Xgboost: 0.99
dtree: 0.9594117647058824
```

In [80]:
```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

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

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

plt.show()
```



Observations:

1. From the bar plots other all the models have good cross-validation scores and performed well on the validation set.
2. The recall score on the validation set for all of the is about 95% and above.
3. We will choose the 3 best models from here, which are: Gradient boosting with 99.9%, XGboost with 98.8% and Adaboost with 98.2%

Hyperparameter Tuning

We will tune Adaboost, Gradient boosting and xgboost models using RandomizedSearchCV.

First, let's create two functions to calculate different metrics and confusion matrix so that we don't have to use the same code repeatedly for each model.

In [81]:
```
# defining a function to compute different metrics to check performance of a classification model built using skl
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

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

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

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

    # creating a dataframe of metrics
```

```
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1": f1,
        },
        index=[0],
    )

    return df_perf
```

In [82]:
```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

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

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

AdaBoost

In [83]:
```
# defining model
model = AdaBoostClassifier(random_state=1)

# Parameter grid to pass in GridSearchCV

param_grid = {
    "n_estimators": np.arange(10, 110, 10),
    "learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "base_estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}

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

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_jobs = -1, n_iter=50, scori

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

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

```
Best parameters are {'n_estimators': 50, 'learning_rate': 0.01, 'base_estimator': DecisionTreeClassifier(max_dept
h=1, random_state=1)} with CV score=1.0:
```

In [84]:
```
# building model with best parameters
adb_tuned2 = AdaBoostClassifier(
    n_estimators=50,
    learning_rate= 0.01,
    random_state=1,
    base_estimator=DecisionTreeClassifier(max_depth=1, random_state=1),
)

# Fit the model on training data
adb_tuned2.fit(X_train, y_train)
```

Out[84]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1,
                                                         random_state=1),
                   learning_rate=0.01, random_state=1)
```

In [85]:
```
# Calculating different metrics on train set
```

```
Adaboost_random_train = model_performance_classification_sklearn(
    adb_tuned2, X_train, y_train
)
print("Training performance:")
Adaboost_random_train
```

```
Training performance:
```

Out[85]:

	Accuracy	Recall	Precision	F1
0	0.839342	1.0	0.839342	0.912654

In [86]:

```
# Calculating different metrics on validation set
Adaboost_random_val = model_performance_classification_sklearn(adb_tuned2, X_val, y_val)
print("Validation performance:")
Adaboost_random_val
```

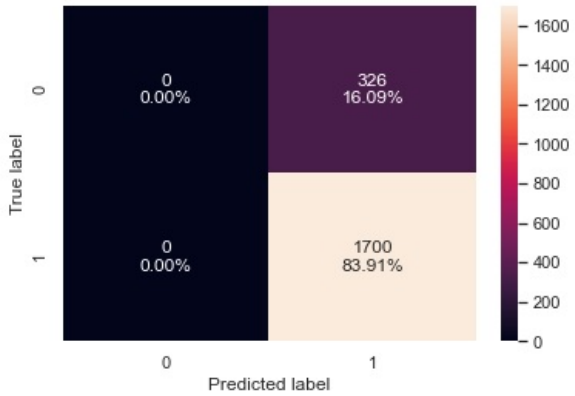
```
Validation performance:
```

Out[86]:

	Accuracy	Recall	Precision	F1
0	0.839092	1.0	0.839092	0.912507

In [87]:

```
# creating confusion matrix
confusion_matrix_sklearn(adb_tuned2, X_val, y_val)
```



Observations:

1. There appears to be no overfitting in the data sets.
2. We get perfect recall score of 1, and good accuracy scoes as both accuracy scores are at 0.83.
3. Need to test this with oversampled and undersampled data.

Gradient Boosting

In [88]:

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

# 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 = metrics.make_scorer(metrics.recall_score)

# Run the grid search
gbc_tuned_rc = RandomizedSearchCV(estimator = gbc_tuned,param_distributions =  parameters, n_jobs = -1, n_iter =
gbc_tuned_rc.fit(X_train, y_train)

# Set the clf to the best combination of parameters
print(gbc_tuned_rc.best_estimator_)

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

```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=1, random_state=1, subsample=0.8)
Best parameters are {'subsample': 0.8, 'n_estimators': 100, 'max_features': 1} with CV score=0.9925470953837865:
```

In [89]:
```
gbc_tuned_rc = GradientBoostingClassifier(
    random_state = 1,
    subsample = 0.9,
    n_estimators = 100,
    max_features = 1,
)
gbc_tuned_rc.fit(X_train, y_train)
```

Out[89]:
```
GradientBoostingClassifier(max_features=1, random_state=1, subsample=0.9)
```

In [90]:
```
# Calculating different metrics on train set
gbc_tuned_rc_random_train = model_performance_classification_sklearn(
    gbc_tuned_rc, X_train, y_train
)
print("Training performance:")
gbc_tuned_rc_random_train
```

```
Training performance:
```

Out[90]:

	Accuracy	Recall	Precision	F1
0	0.907325	0.993528	0.905289	0.947359

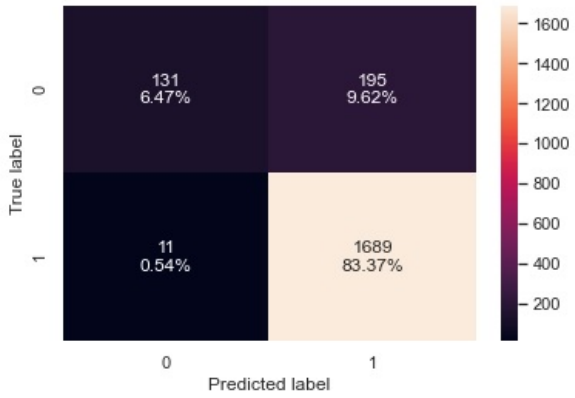
In [91]:
```
# Calculating different metrics on validation set
gbc_random_val = model_performance_classification_sklearn(gbc_tuned_rc, X_val, y_val)
print("Validation performance:")
gbc_random_val
```

```
Validation performance:
```

Out[91]:

	Accuracy	Recall	Precision	F1
0	0.898322	0.993529	0.896497	0.942522

In [92]:
```
# creating confusion matrix
confusion_matrix_sklearn(gbc_tuned_rc, X_val, y_val)
```



Observations:

1. No overfitting is observed.
2. Good scores across both testing and validation data sets.
3. All score metrics are above 90% suggesting a good generalized model.

XGBoost

In [93]:
```
# defining model
model = XGBClassifier(random_state=1,eval_metric='logloss')

# Parameter grid to pass in RandomizedSearchCV
param_grid={'n_estimators':np.arange(10,150,20),
            'scale_pos_weight':[2,5,10],
            'learning_rate':[0.01,0.1,0.2,0.05],
            'gamma':[0,1,3,5],
```

```
            'subsample':[0.5, 0.7, 0.8,0.9,1],
            'max_depth':np.arange(1,5,1),
            'reg_lambda':[5,10],
            "colsample_bytree":[0.5,0.7,0.9,1],
            "colsample_bylevel":[0.5,0.7,0.9,1]
           }

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

#Calling RandomizedSearchCV
xgb_tuned2 = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=50, scoring=scorer, cv=5,

#Fitting parameters in RandomizedSearchCV
xgb_tuned2.fit(X_train,y_train)

print("Best parameters are {} with CV score={}:" .format(xgb_tuned2.best_params_,xgb_tuned2.best_score_))
```

```
Best parameters are {'subsample': 1, 'scale_pos_weight': 10, 'reg_lambda': 10, 'n_estimators': 30, 'max_depth': 1
, 'learning_rate': 0.2, 'gamma': 0, 'colsample_bytree': 0.5, 'colsample_bylevel': 0.9} with CV score=1.0:
```

In [94]:
```
# building model with best parameters
xgb_tuned2 = XGBClassifier(
    random_state=1,
    n_estimators=90,
    scale_pos_weight=10,
    gamma=3,
    subsample=0.8,
    learning_rate=0.01,
    max_depth=2,
    reg_lambda=10,
    colsample_bytree = 0.7,
    colsample_bylevel = 0.9,
    eval_metric='logloss'
)
# Fit the model on training data
xgb_tuned2.fit(X_train, y_train)
```

Out[94]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
              colsample_bynode=1, colsample_bytree=0.7, eval_metric='logloss',
              gamma=3, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.01, max_delta_step=0,
              max_depth=2, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=90, n_jobs=8,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=10,
              scale_pos_weight=10, subsample=0.8, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [95]:
```
# Calculating different metrics on train set
xgboost_random_train = model_performance_classification_sklearn(
    xgb_tuned2, X_train, y_train
)
print("Training performance:")
xgboost_random_train
```

```
Training performance:
```

Out[95]:

	Accuracy	Recall	Precision	F1
0	0.84856	1.0	0.847151	0.917251

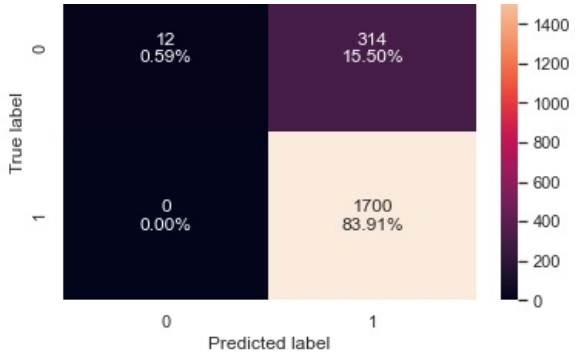
In [96]:
```
# Calculating different metrics on validation set
xgboost_random_val = model_performance_classification_sklearn(xgb_tuned2, X_val, y_val)
print("Validation performance:")
xgboost_random_val
```

```
Validation performance:
```

Out[96]:

	Accuracy	Recall	Precision	F1
0	0.845015	1.0	0.844091	0.915455

In [97]:
```
# creating confusion matrix
confusion_matrix_sklearn(xgb_tuned2, X_val, y_val)
```



Observation:

1. No overfitting of data observed in the data sets.
2. Perfect recall score on both train and validation data set.
3. All other score metrics also perform well as all scores are at 0.84 and above.

Changing the sampling size of data by oversampling and undersampling

Oversampling train data using SMOTE

In [98]:
```
print("Before UpSampling, counts of label 'Yes': {}".format(sum(y_train == 1)))
print("Before UpSampling, 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 UpSampling, counts of label 'Yes': {}".format(sum(y_train_over == 1)))
print("After UpSampling, counts of label 'No': {} \n".format(sum(y_train_over == 0)))


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

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

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

After UpSampling, the shape of train_X: (10198, 50)
After UpSampling, the shape of train_y: (10198,)
```

- The data has been scaled up from 976 to 5099, giving an even data set in train data set
- Now all the 3 models tuned before will be fit on the oversampled data set

AdaBoost on Oversampled Data

In [99]:
```
adb_tuned2_over = AdaBoostClassifier(
    n_estimators=50,
    learning_rate= 0.01,
    random_state=1,
    base_estimator=DecisionTreeClassifier(max_depth=1, random_state=1),
)

# Fit the model on training data
adb_tuned2_over.fit(X_train_over, y_train_over)
```

Out[99]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1,
                                                         random_state=1),
                   learning_rate=0.01, random_state=1)
```

In [100…
```
# Calculating different metrics on train set
Adaboost_random_train_over = model_performance_classification_sklearn(
    adb_tuned2_over, X_train_over, y_train_over
)
print("Training performance for oversampled data:")
Adaboost_random_train_over
```

```
Training performance for oversampled data:
```

Out[100…

	Accuracy	Recall	Precision	F1
0	0.791332	0.743479	0.822164	0.780844

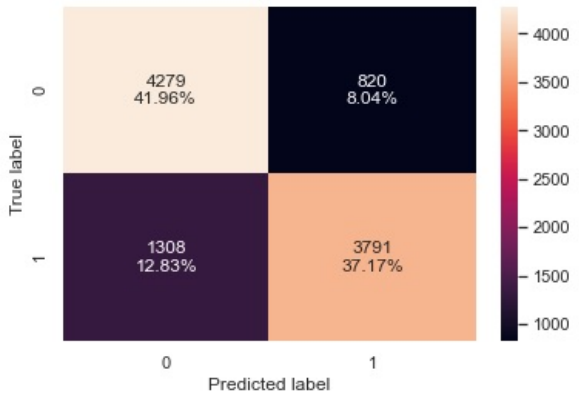
In [101…
```
# Calculating different metrics on validation set
Adaboost_random_val_over = model_performance_classification_sklearn(adb_tuned2_over, X_val, y_val)
print("Validation performance:")
Adaboost_random_val_over
```

```
Validation performance:
```

Out[101…

	Accuracy	Recall	Precision	F1
0	0.753702	0.748235	0.947133	0.836017

In [102…
```
confusion_matrix_sklearn(adb_tuned2_over, X_train_over, y_train_over)
```



Observations:

1. There is a slight variance in accuracy scores between the train and validation test set, but no big indication of overfitting.
2. THe recall score is slighly lower than the previous Adaboost.
3. Good performance in precision, but since that isn't our evaluation metric we will look at other models.

Gradient Boosting on Oversampled data

In [103…
```
gbc_tuned_rc_over = GradientBoostingClassifier(
    random_state = 1,
    subsample = 0.9,
    n_estimators = 100,
    max_features = 1,
)
gbc_tuned_rc_over.fit(X_train_over, y_train_over)
```

Out[103…
```
GradientBoostingClassifier(max_features=1, random_state=1, subsample=0.9)
```

In [104…
```
# Calculating different metrics on train set
gbc_tuned_rc_random_train_over = model_performance_classification_sklearn(
    gbc_tuned_rc_over, X_train_over, y_train_over
)
print("Training performance on Oversampled data:")
gbc_tuned_rc_random_train_over
```

```
Training performance on Oversampled data:
```

Out[104…

	Accuracy	Recall	Precision	F1
0	0.937635	0.958423	0.920166	0.938905

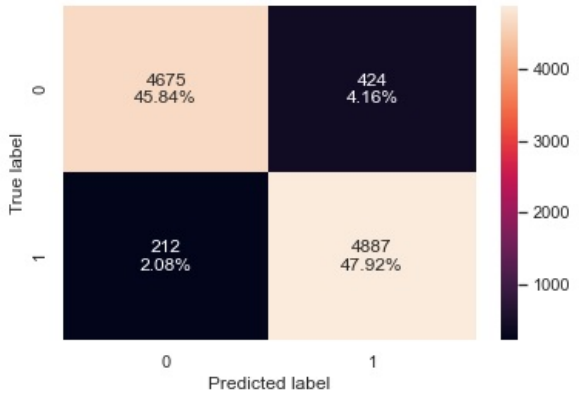
In [105…
```
# Calculating different metrics on validation set
gbc_random_val_over = model_performance_classification_sklearn(gbc_tuned_rc_over, X_val, y_val)
print("Validation performance:")
gbc_random_val_over
```

```
Validation performance:
```

Out[105…

	Accuracy	Recall	Precision	F1
0	0.894373	0.955294	0.92168	0.938186

In [106…
```
confusion_matrix_sklearn(gbc_tuned_rc_over, X_train_over, y_train_over)
```



Observations:

1. Good scores in all of metrics.
2. No overfitting visible.
3. Similarly to GBC tuned, GBC on oversampled data has performed well in all metrics.

XGBoost on oversampled data

In [107…
```
# building model with best parameters
xgb_tuned2_over = XGBClassifier(
    random_state=1,
    n_estimators=90,
    scale_pos_weight=10,
    gamma=3,
    subsample=0.8,
    learning_rate=0.01,
    max_depth=2,
    reg_lambda=10,
    colsample_bytree = 0.7,
    colsample_bylevel = 0.9,
    eval_metric='logloss'
)
# Fit the model on training data
xgb_tuned2_over.fit(X_train_over, y_train_over)
```

Out[107…
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
              colsample_bynode=1, colsample_bytree=0.7, eval_metric='logloss',
              gamma=3, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.01, max_delta_step=0,
              max_depth=2, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=90, n_jobs=8,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=10,
              scale_pos_weight=10, subsample=0.8, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [108…
```
# Calculating different metrics on train set
xgboost_random_train_over = model_performance_classification_sklearn(
    xgb_tuned2_over, X_train_over, y_train_over
)
print("Training performance on Oversampled data:")
xgboost_random_train_over
```

```
Training performance on Oversampled data:
```

Out[108…

	Accuracy	Recall	Precision	F1
0	0.671995	0.991371	0.604955	0.751394

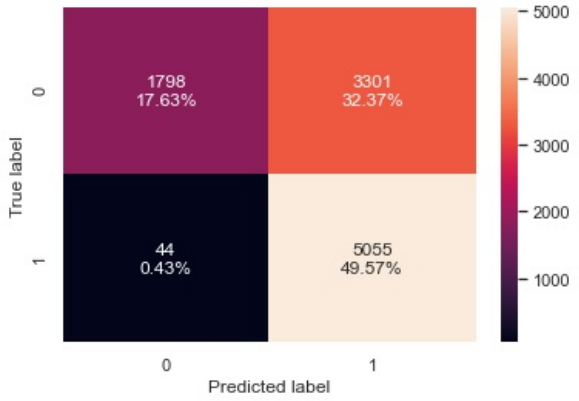
In [109…
```
# Calculating different metrics on validation set
xgboost_random_val_over = model_performance_classification_sklearn(xgb_tuned2_over, X_val, y_val)
print("Validation performance:")
xgboost_random_val_over
```

```
Validation performance:
```

Out[109…

	Accuracy	Recall	Precision	F1
0	0.883514	0.992353	0.883246	0.934626

In [110…
```
confusion_matrix_sklearn(xgb_tuned2_over, X_train_over, y_train_over)
```



Observations:

1. Overfitting is visible with XGBoost on oversampled data. There is disparity in the accuracy scores.
2. The recall score is good on both the train and validation set, as the recall score is perfect 1, suggesting a really good model.
3. Other metrics are also performing well.

Undersampling train data using Random Under Sampler

In [111…
```
rus = RandomUnderSampler(random_state=1)
X_train_under, y_train_under = rus.fit_resample(X_train, y_train)
```

In [112…
```
print("Before Under Sampling, counts of label 'Yes': {}".format(sum(y_train == 1)))
print("Before Under Sampling, counts of label 'No': {} \n".format(sum(y_train == 0)))

print("After Under Sampling, counts of label 'Yes': {}".format(sum(y_train_under == 1)))
print("After Under Sampling, counts of label 'No': {} \n".format(sum(y_train_under == 0)))

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

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

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

After Under Sampling, the shape of train_X: (1952, 50)
After Under Sampling, the shape of train_y: (1952,)
```

Adaboost on undersampled data

In [113…
```
adb_tuned2_under = AdaBoostClassifier(
    n_estimators=50,
    learning_rate= 0.01,
    random_state=1,
    base_estimator=DecisionTreeClassifier(max_depth=1, random_state=1),
```

```
)

# Fit the model on training data
adb_tuned2_under.fit(X_train_under, y_train_under)
```

Out[113…
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1,
                                                         random_state=1),
                   learning_rate=0.01, random_state=1)
```

In [114…
```
# Calculating different metrics on train set
Adaboost_random_train_under = model_performance_classification_sklearn(
    adb_tuned2_under, X_train_under, y_train_under
)
print("Training performance for Undersampled data:")
Adaboost_random_train_under
```

```
Training performance for Undersampled data:
```

Out[114…

	Accuracy	Recall	Precision	F1
0	0.779201	0.695697	0.835178	0.759083

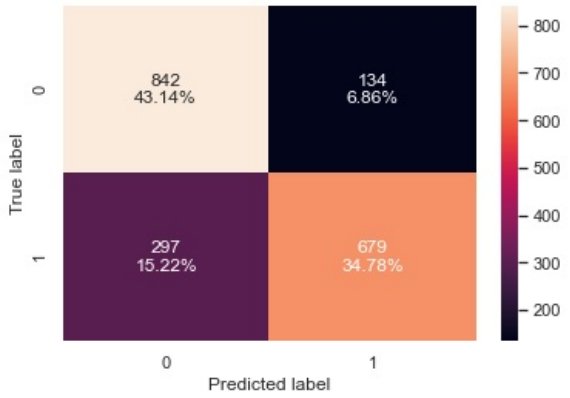
In [115…
```
# Calculating different metrics on validation set
Adaboost_random_val_under = model_performance_classification_sklearn(adb_tuned2_under, X_val, y_val)
print("Validation performance for Undersampled Data:")
Adaboost_random_val_under
```

```
Validation performance for Undersampled Data:
```

Out[115…

	Accuracy	Recall	Precision	F1
0	0.740375	0.721765	0.958594	0.82349

In [116…
```
confusion_matrix_sklearn(adb_tuned2_under, X_train_under, y_train_under)
```



Observations:

1. Overall a good accuracy scores on both train and validation data sets. No over fitting visible.
2. Ada boost hasn't performed well in oversampled, undersampled or the tuned model.
3. Not getting the best recall score on either train or validation set.

Gradient boosting on Undersampled data

In [117…
```
gbc_tuned_rc_under = GradientBoostingClassifier(
    random_state = 1,
    subsample = 0.9,
    n_estimators = 100,
    max_features = 1,
)
gbc_tuned_rc_under.fit(X_train_under, y_train_under)
```

Out[117…
```
GradientBoostingClassifier(max_features=1, random_state=1, subsample=0.9)
```

In [118…
```
# Calculating different metrics on train set
```

```
gbc_tuned_rc_random_train_under = model_performance_classification_sklearn(
    gbc_tuned_rc_under, X_train_under, y_train_under
)
print("Training performance on Undersampled data:")
gbc_tuned_rc_random_train_under
```

```
Training performance on Undersampled data:
```

Out[118…

	Accuracy	Recall	Precision	F1
0	0.876537	0.884221	0.870838	0.877478

In [119…

```
# Calculating different metrics on validation set
gbc_random_val_under = model_performance_classification_sklearn(gbc_tuned_rc_under, X_val, y_val)
print("Validation performance on Undersampled data:")
gbc_random_val_under
```

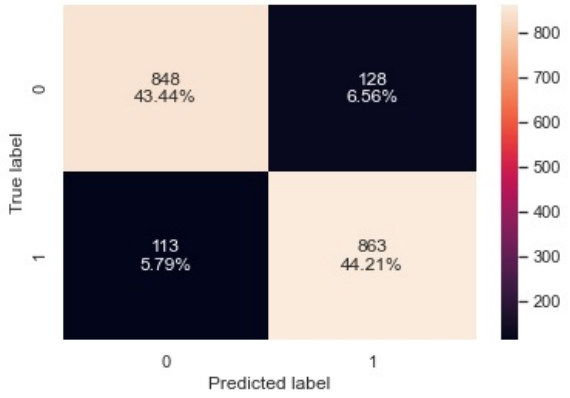
```
Validation performance on Undersampled data:
```

Out[119…

	Accuracy	Recall	Precision	F1
0	0.859329	0.871176	0.957337	0.912227

In [120…

```
confusion_matrix_sklearn(gbc_tuned_rc_under, X_train_under, y_train_under)
```



Observations:

1. The accuracy scores are a good indication of a good model.
2. No overfitting visible.
3. Good recall scores, at 87%.

XGBoost on undersampled data

In [121…

```
# building model with best parameters
xgb_tuned2_under = XGBClassifier(
    random_state=1,
    n_estimators=90,
    scale_pos_weight=10,
    gamma=3,
    subsample=0.8,
    learning_rate=0.01,
    max_depth=2,
    reg_lambda=10,
    colsample_bytree = 0.7,
    colsample_bylevel = 0.9,
    eval_metric='logloss'
)
# Fit the model on training data
xgb_tuned2_under.fit(X_train_under, y_train_under)
```

Out[121…

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
              colsample_bynode=1, colsample_bytree=0.7, eval_metric='logloss',
              gamma=3, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.01, max_delta_step=0,
              max_depth=2, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=90, n_jobs=8,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=10,
              scale_pos_weight=10, subsample=0.8, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [122…
```
# Calculating different metrics on train set
xgboost_random_train_under = model_performance_classification_sklearn(
    xgb_tuned2_under, X_train_under, y_train_under
)
print("Training performance on Undersampled data:")
xgboost_random_train_under
```

```
Training performance on Undersampled data:
```

Out[122…

	Accuracy	Recall	Precision	F1
0	0.625512	0.989754	0.572614	0.725498

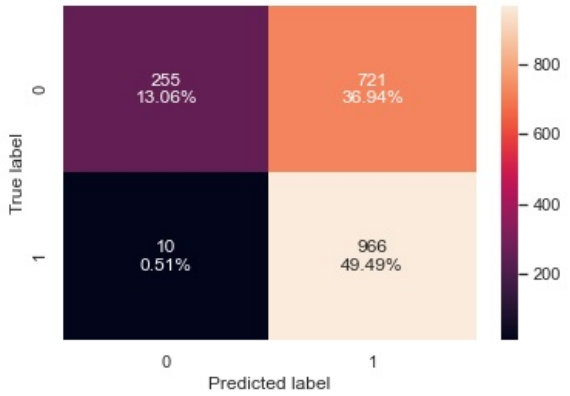
In [123…
```
# Calculating different metrics on validation set
xgboost_random_val_under = model_performance_classification_sklearn(xgb_tuned2_under, X_val, y_val)
print("Validation performance on Undersampled data:")
xgboost_random_val_under
```

```
Validation performance on Undersampled data:
```

Out[123…

	Accuracy	Recall	Precision	F1
0	0.871175	0.992941	0.871451	0.928238

In [124…
```
confusion_matrix_sklearn(xgb_tuned2_under, X_train_under, y_train_under)
```



Observations:

1. The train set performance is slightly poor for both accuracy and percision.
2. The recall scores for both train and validation data set is really good as it hits near 99%.
3. Overall a good indication of a good model.

Comparing all models

In [125…
```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        Adaboost_random_train.T,
        Adaboost_random_train_over.T,
        Adaboost_random_train_under.T,
        gbc_tuned_rc_random_train.T,
        gbc_tuned_rc_random_train_over.T,
        gbc_tuned_rc_random_train_under.T,
        xgboost_random_train.T,
        xgboost_random_train_over.T,
        xgboost_random_train_under.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "AdaBoost Tuned with Random search",
    "AdaBoost on Oversampled data",
    "AdaBoost on Undersampled data",
    "GBC Tuned with Random Search",
    "GBC on Oversampled data",
```

```
    "GBC on Undersampled data",
    "Xgboost Tuned with Random Search",
    "Xgboost on Oversampled data",
    "Xgboost on Undersampled data",
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[125…

	AdaBoost Tuned with Random search	AdaBoost on Oversampled data	AdaBoost on Undersampled data	GBC Tuned with Random Search	GBC on Oversampled data	GBC on Undersampled data	Xgboost Tuned with Random Search	Xgboost on Oversampled data	Xgboost on Undersampled data
Accuracy	0.839342	0.791332	0.779201	0.907325	0.937635	0.876537	0.848560	0.671995	0.625512
Recall	1.000000	0.743479	0.695697	0.993528	0.958423	0.884221	1.000000	0.991371	0.989754
Precision	0.839342	0.822164	0.835178	0.905289	0.920166	0.870838	0.847151	0.604955	0.572614
F1	0.912654	0.780844	0.759083	0.947359	0.938905	0.877478	0.917251	0.751394	0.725498

In [126…

```
# validation performance comparison

models_val_comp_df = pd.concat(
    [
        Adaboost_random_val.T,
        Adaboost_random_val_over.T,
        Adaboost_random_val_under.T,
        gbc_random_val.T,
        gbc_random_val_over.T,
        gbc_random_val_under.T,
        xgboost_random_val.T,
        xgboost_random_val_over.T,
        xgboost_random_val_under.T,
    ],
    axis=1,
)
models_val_comp_df.columns = [
    "AdaBoost Tuned with Random search",
    "AdaBoost on Oversampled data",
    "AdaBoost on Undersampled data",
    "GBC Tuned with Random Search",
    "GBC on Oversampled data",
    "GBC on Undersampled data",
    "Xgboost Tuned with Random Search",
    "Xgboost on Oversampled data",
    "Xgboost on Undersampled data",
]
print("Validation data set performance comparison:")
models_val_comp_df
```

```
Validation data set performance comparison:
```

Out[126…

	AdaBoost Tuned with Random search	AdaBoost on Oversampled data	AdaBoost on Undersampled data	GBC Tuned with Random Search	GBC on Oversampled data	GBC on Undersampled data	Xgboost Tuned with Random Search	Xgboost on Oversampled data	Xgboost on Undersampled data
Accuracy	0.839092	0.753702	0.740375	0.898322	0.894373	0.859329	0.845015	0.883514	0.871175
Recall	1.000000	0.748235	0.721765	0.993529	0.955294	0.871176	1.000000	0.992353	0.992941
Precision	0.839092	0.947133	0.958594	0.896497	0.921680	0.957337	0.844091	0.883246	0.871451
F1	0.912507	0.836017	0.823490	0.942522	0.938186	0.912227	0.915455	0.934626	0.928238

Observations:

- From the data sets, in training data set the best performed models are GBC tuned and and GBC on oversampled data with accuracy, precision and recall scores all above 0.89
- For the validation data set, the two best models are from GBC tuned and GBC oversampled data, with all metrics hitting above 0.89
- The best model that performed well on all metrics and that had no overfitting was the GBC tuned with Accuracy, recall and precision scores of 0.89, 0.99 and 0.89 respectively.

Performance on Test data using GBC tuned with random search

In [127…

```
# Calculating different metrics on the test set
gbc_random_val_test = model_performance_classification_sklearn(gbc_tuned_rc, X_test, y_test)
print("Test performance")
```

```
gbc_random_val_test
```

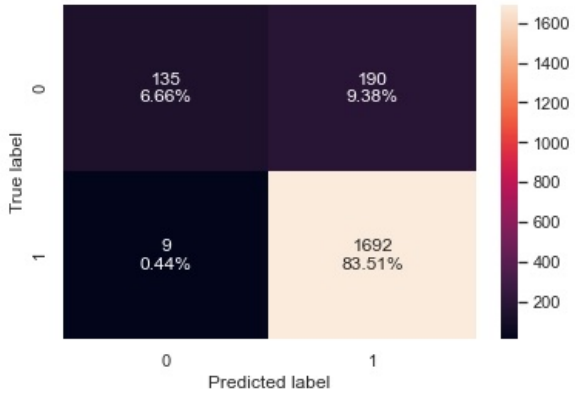
Test performance

Out[127…

	Accuracy	Recall	Precision	F1
0	0.901777	0.994709	0.899044	0.94446

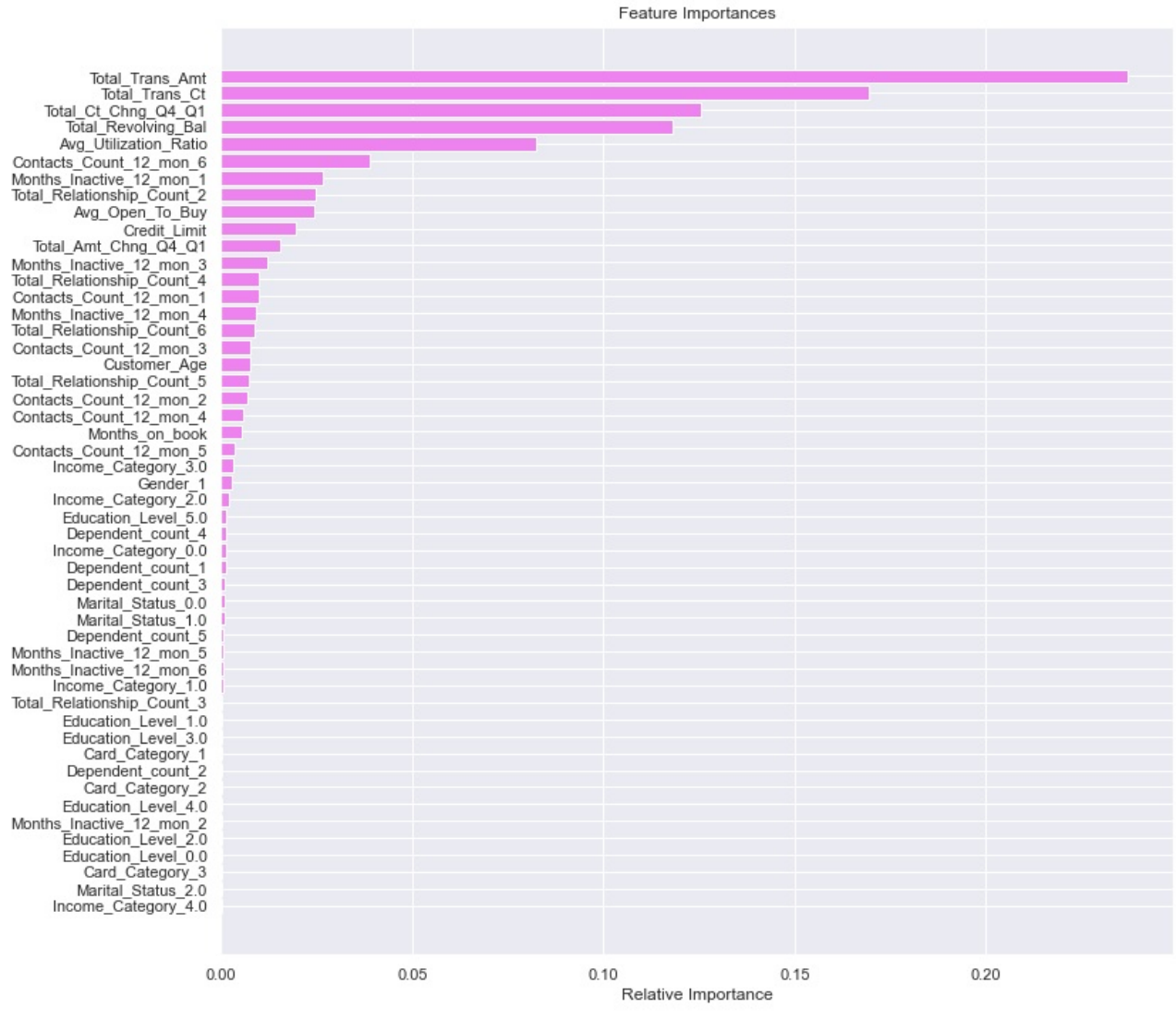
In [128…

```
# creating confusion matrix
confusion_matrix_sklearn(gbc_tuned_rc, X_test, y_test)
```



In [129…

```
feature_names = X_test.columns
importances = gbc_tuned_rc.feature_importances_
indices = np.argsort(importances)

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



Observations:

1. From the model performance on the test set we can confirm that this is a good generalized model.
2. The accuracy, precision and recall scores are all above 0.9.
3. The recall score of 0.99 indicates that this is a really good fit model and it caters well towards the data set.
4. From the future importances, it is visible that Total transaction amount in the past 12 months, is the most important variable.
5. Total Ct change Q4 to Q1 and Total trans CT are also very important variables.

Pipelines for productionizing the model

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

Column Transformer

- We know that we can use pipelines to standardize the model building, but the steps in a pipeline are applied to each and every variable - how can we personalize the pipeline to perform different processing on different columns
- Column transformer allows different columns or column subsets of the input to be transformed separately and the features generated by each transformer will be concatenated to form a single feature space. This is useful for heterogeneous or columnar data, to combine several feature extraction mechanisms or transformations into a single transformer.

In [130…
```
# creating a list of categorical variables
categorical_features = ["Gender", 'Dependent_count', "Education_Level", "Marital_Status", "Income_Category", "Tot



# creating a transformer for categorical variables, which will first apply simple imputer and
#then do one hot encoding for categorical variables
categorical_transformer = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("onehot", OneHotEncoder(handle_unknown="ignore"))
    ]
)
# handle_unknown = "ignore", allows model to handle any unknown category in the test data

# combining categorical transformer and numerical transformer using a column transformer

preprocessor = ColumnTransformer(
    transformers=[
        ("cat", categorical_transformer, categorical_features),
    ],
    remainder="passthrough",
)
# remainder = "passthrough" has been used, it will allow variables that are present in original data
# but not in "numerical_columns" and "categorical_columns" to pass through the column transformer without any cha
```

In [131…
```
# Separating target variable and other variables
X = df.drop(columns="Attrition_Flag")
Y = df["Attrition_Flag"]
```

In [132…
```
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.30, random_state=1, stratify=Y
)
print(X_train.shape, X_test.shape)
```

```
(7088, 19) (3039, 19)
```

In [133…
```
# Creating new pipeline with best parameters
model = Pipeline(steps=[("pre", preprocessor),("GBC",GradientBoostingClassifier(random_state = 1,subsample = 0.9,
# Fit the model on training data
model.fit(X_train, y_train)
```

Out[133…
```
Pipeline(steps=[('pre',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImputer(strategy='most_frequent')),
                                                                  ('onehot',
                                                                   OneHotEncoder(handle_unknown='ignore'))]),
                                                  ['Gender', 'Dependent_count',
                                                   'Education_Level',
                                                   'Marital_Status',
```

```
                                          'Income_Category',
                                          'Total_Relationship_Count',
                                          'Months_Inactive_12_mon',
                                          'Contacts_Count_12_mon'])])),
                ('GBC',
                 GradientBoostingClassifier(max_features=1, random_state=1,
                                            subsample=0.9))])
```

In [134…
```
y_predict = model.predict(X_test)
model_score = model.score(X_test, y_test)
print("The model score for the Pipeline is: {:.2f}".format(model_score))
```

```
The model score for the Pipeline is: 0.90
```

- Good overall moddel score for the pipleine at 0.9 indicating a good pipeline model.

Recommendation

- From the differnet models build and the EDA, we can establish that the total transmitted amount from an account over the duration of 12 months is a key indicator when it comes to determining if an individual will continue using their credit cards or not.
- If an indvidual has no money to transmit to other accounts for either personal/business reasons, then the client will discontinue their credit card service.
- Total transmitted amount and total trasactions count had a good positive correlation between each other which indicates that the more a person uses their account the more they are likely to transact bigger amounts of money.
- On average customers that completed had more trasmitted amounts were more likley to remain as existing customers. And from the data analysis it was seen that the existing customers had on average higher transmitted amounts.
- Customers with higher revolving balance are more likely to continue using credit cards.
- Higher ratio in the total trasaction count and total amount change in 4th quarter and thte total trasaction count in 1st quarter also meant that the customer is likely to keep using their credit cards.
- The more the customer is using their remaining available credit, the more likely the customer is to keep using their credit cards.
- As the customers age increases the months on book also increases which also increased the duration of which a user continued using their credit cards.

In [ ]:
```
```