Credit Card Users Churn Prediction for Thera bank

Problem Statement:

The Thera bank recently saw a steep decline in the number of users of their credit card. 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.

Objective:

Provide insight that will help the bank improve its services, so that customers do not renounce their credit cards.

- Build a classification model to predict if the customer is going to churn.

Data Description:

- CLIENTNUM: Client number. Unique identifier for the customer holding the account
- Attrition_Flag: Internal event (customer activity) variable - if the account is closed then "Attrited Customer" else "Existing Customer"
- Customer_Age: Age in Years
- Gender: Gender of the account holder
- Dependent_count: Number of dependents
- Education_Level: Educational Qualification of the account holder - Graduate, High School, Unknown, Uneducated, College(refers to a college student), Post-Graduate, Doctorate.
- Marital_Status: Marital Status of the account holder
- Income_Category: Annual Income Category of the account holder
- Card_Category: Type of Card
- Months_on_book: Period of relationship with the bank
- Total_Relationship_Count: Total no. of products held by the customer
- Months_Inactive_12_mon: No. of months inactive in the last 12 months
- Contacts_Count_12_mon: No. of Contacts between the customer and bank in the last 12 months
- Credit_Limit: Credit Limit on the Credit Card
- Total_Revolving_Bal: The balance that carries over from one month to the next is the revolving balance
- Avg_Open_To_Buy: Open to Buy refers to the amount left on the credit card to use (Average of last 12 months)

- Total_Trans_Amt: Total Transaction Amount (Last 12 months)
- Total_Trans_Ct: Total Transaction Count (Last 12 months)
- Total_Ct_Chng_Q4_Q1: Ratio of the total transaction count in 4th quarter and the total transaction count in 1st quarter
- Total_Amt_Chng_Q4_Q1: Ratio of the total transaction amount in 4th quarter and the total transaction amount in 1st quarter
- Avg_Utilization_Ratio: Represents how much of the available credit the customer spent

Libraries

In [1]:
```
%load_ext nb_black
# Library to suppress warnings or deprecation notes
import warnings

warnings.filterwarnings("ignore")

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

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

%matplotlib inline
import seaborn as sns

# Libraries to split data, impute missing values
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer

# Libraries to import decision tree classifier and different ensemble classifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (
    AdaBoostClassifier,
    GradientBoostingClassifier,
    RandomForestClassifier,
    BaggingClassifier,
)
from xgboost import XGBClassifier

# Libtune to tune model, get different metric scores
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn import metrics
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
    plot_confusion_matrix,
)
```

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

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

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

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

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

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

Read Dataset

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

df = data.copy()
```

Data Info/Details

In [3]:
```
df.head()
```

Out[3]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital
0	768805383	Existing Customer	45	M	3	High School	
1	818770008	Existing Customer	49	F	5	Graduate	
2	713982108	Existing Customer	51	M	3	Graduate	
3	769911858	Existing Customer	40	F	4	High School	
4	709106358	Existing Customer	40	M	3	Uneducated	

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

Out[4]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	M
10122	772366833	Existing Customer	50	M	2	Graduate	

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	M
10123	710638233	Attrited Customer	41	M	2	NaN	
10124	716506083	Attrited Customer	44	F	1	High School	
10125	717406983	Attrited Customer	30	M	2	Graduate	
10126	714337233	Attrited Customer	43	F	2	Graduate	

In [5]:
```
np.random.seed(2)
df.sample(10)
```

Out[5]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Ma
7862	711589758	Existing Customer	42	M	5	High School	
3536	717886008	Existing Customer	63	M	1	High School	
9770	720852108	Existing Customer	46	M	4	High School	
8909	719785683	Existing Customer	41	M	4	Post-Graduate	
709	780054258	Existing Customer	40	M	5	Graduate	
975	717180633	Existing Customer	59	M	1	Post-Graduate	
32	709029408	Existing Customer	41	M	4	Graduate	
9454	708510858	Existing Customer	60	F	1	Uneducated	
4548	781297983	Existing Customer	58	M	2	NaN	
9351	789983133	Existing Customer	31	M	2	Graduate	

In [6]:
```
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns.")
```

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

In [7]:
```
df[data.duplicated()].count()
```

Out[7]:
```
CLIENTNUM                 0
Attrition_Flag            0
Customer_Age              0
Gender                    0
Dependent_count           0
Education_Level           0
Marital_Status            0
```

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

In [8]:
```
df.info()
```

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

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
CLIENTNUM                     0
Attrition_Flag                0
Customer_Age                  0
Gender                        0
Dependent_count               0
Education_Level            1519
Marital_Status              749
Income_Category               0
Card_Category                 0
Months_on_book                0
Total_Relationship_Count      0
Months_Inactive_12_mon        0
Contacts_Count_12_mon         0
Credit_Limit                  0
```

```
Total_Revolving_Bal         0
Avg_Open_To_Buy             0
Total_Amt_Chng_Q4_Q1        0
Total_Trans_Amt             0
Total_Trans_Ct              0
Total_Ct_Chng_Q4_Q1         0
Avg_Utilization_Ratio       0
dtype: int64
```

In [10]:
```
df.describe().T
```

Out[10]:

	count	mean	std	min	25%
CLIENTNUM	10127.000	739177606.334	36903783.450	708082083.000	713036770.500
Customer_Age	10127.000	46.326	8.017	26.000	41.000
Dependent_count	10127.000	2.346	1.299	0.000	1.000
Months_on_book	10127.000	35.928	7.986	13.000	31.000
Total_Relationship_Count	10127.000	3.813	1.554	1.000	3.000
Months_Inactive_12_mon	10127.000	2.341	1.011	0.000	2.000
Contacts_Count_12_mon	10127.000	2.455	1.106	0.000	2.000
Credit_Limit	10127.000	8631.954	9088.777	1438.300	2555.000
Total_Revolving_Bal	10127.000	1162.814	814.987	0.000	359.000
Avg_Open_To_Buy	10127.000	7469.140	9090.685	3.000	1324.500
Total_Amt_Chng_Q4_Q1	10127.000	0.760	0.219	0.000	0.631
Total_Trans_Amt	10127.000	4404.086	3397.129	510.000	2155.500
Total_Trans_Ct	10127.000	64.859	23.473	10.000	45.000
Total_Ct_Chng_Q4_Q1	10127.000	0.712	0.238	0.000	0.582
Avg_Utilization_Ratio	10127.000	0.275	0.276	0.000	0.023

Initial EDA

Univariate

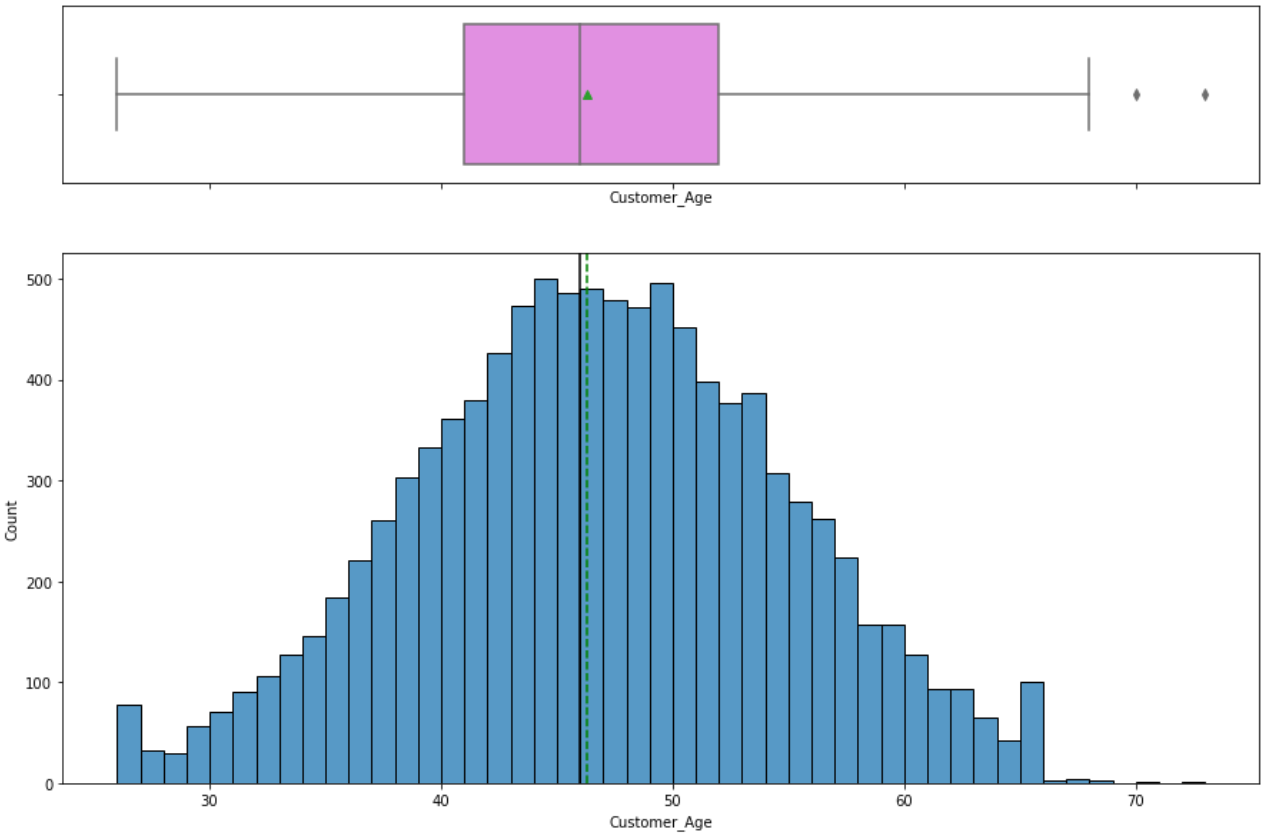
In [11]:
```
# Making a list of all categorical variables
cat_col = [
    "Attrition_Flag",
    "Gender",
    "Education_Level",
    "Marital_Status",
    "Income_Category",
    "Card_Category",
]

# Printing number of count of each unique value in each column
for column in cat_col:
```

```
    print(data[column].value_counts())
    print("-" * 40)
```

```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
----------------------------------------
F    5358
M    4769
Name: Gender, 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
----------------------------------------
```

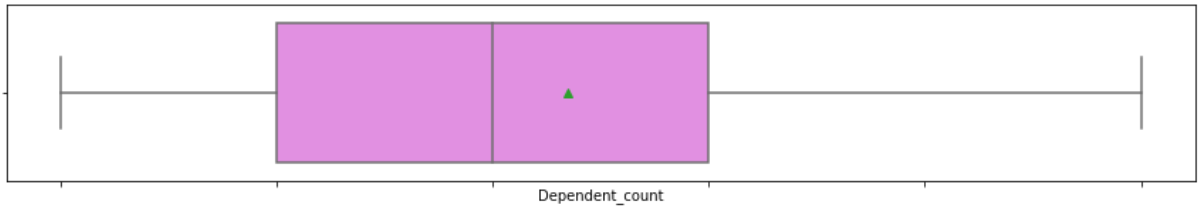
In [12]:
```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,
        sharex=True,
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )
    sns.boxplot(data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(data=data, x=feature, kde=kde, ax=ax_hist2)
    ax_hist2.axvline(data[feature].mean(), color="green", linestyle="--")
    ax_hist2.axvline(data[feature].median(), color="black", linestyle="-")
```

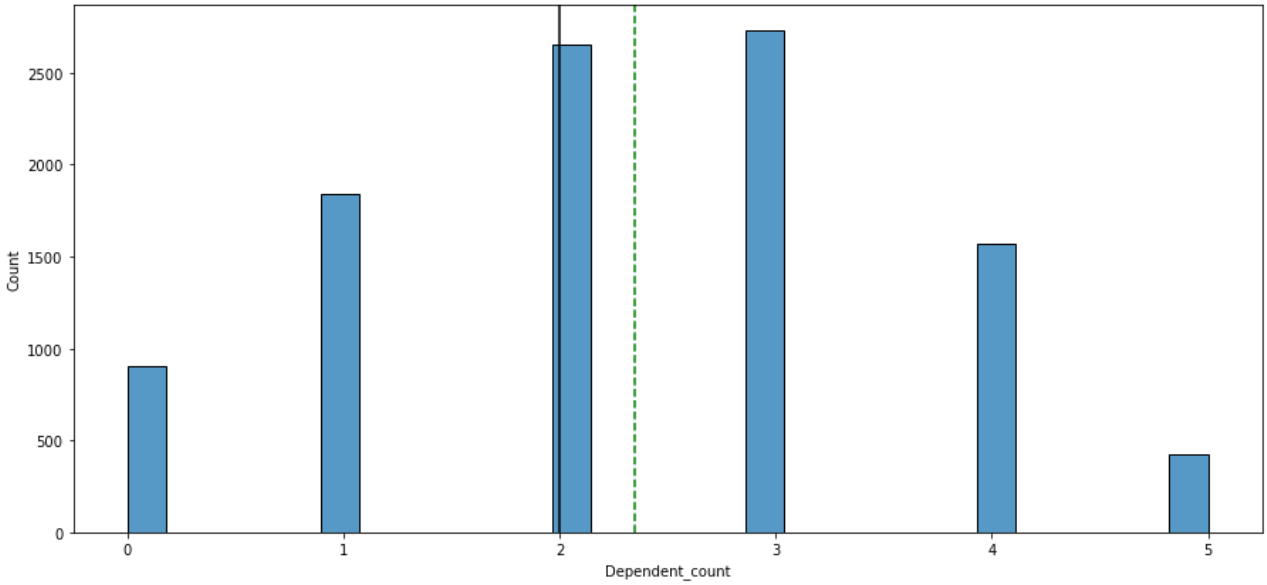
In [13]:
```
# Observation on Customer_Age
histogram_boxplot(data, "Customer_Age")
```



- Normal Distribution

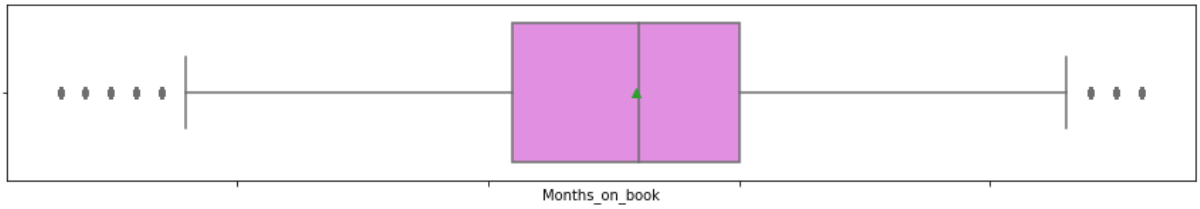
In [14]:
```
# Observation on Dependent_count
histogram_boxplot(data, "Dependent_count")
```

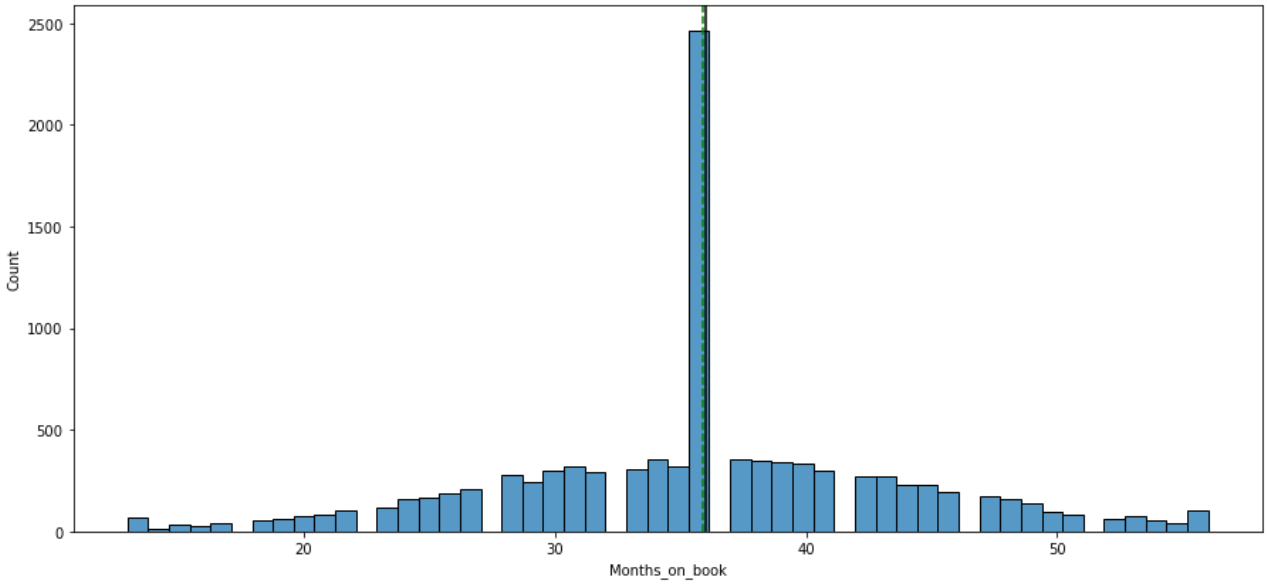




- Normal Distribution

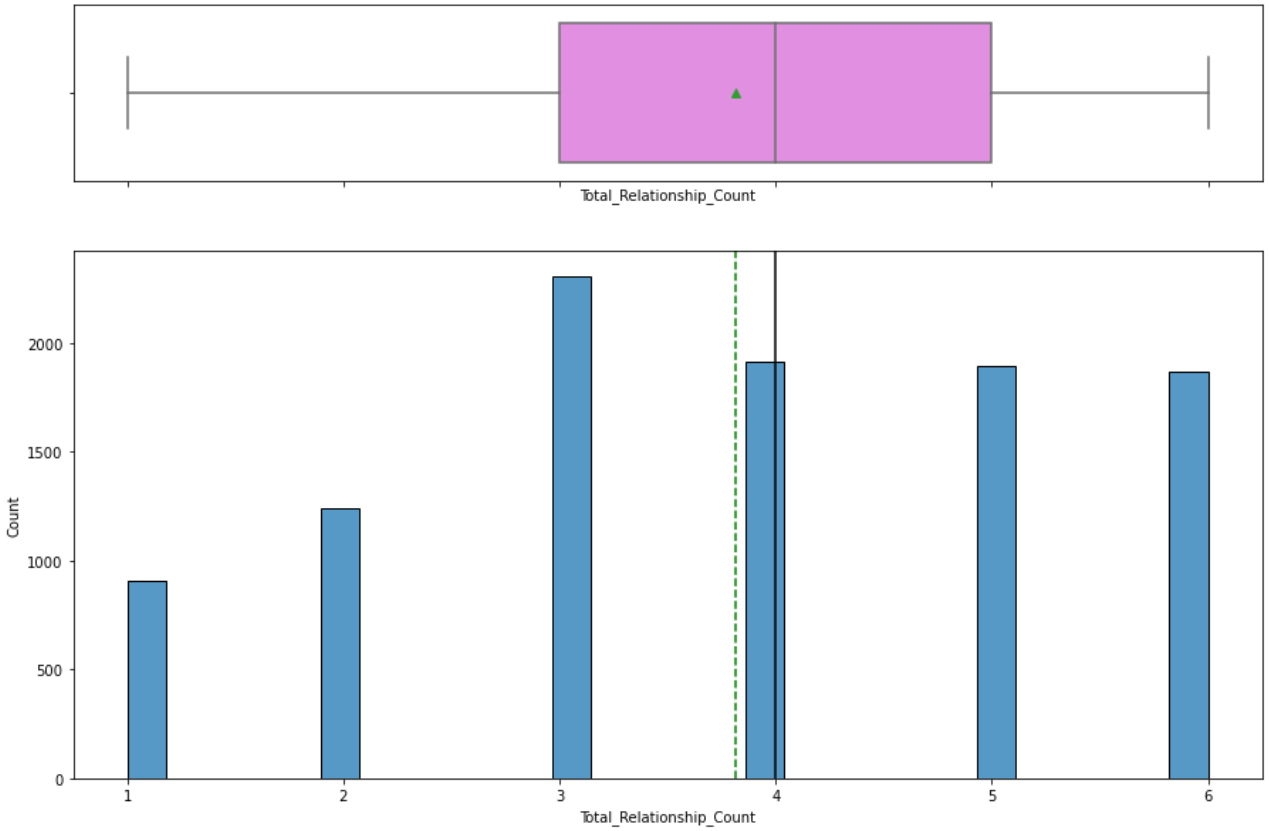
In [15]:
```
# Observation on Months_on_book
histogram_boxplot(data, "Months_on_book")
```





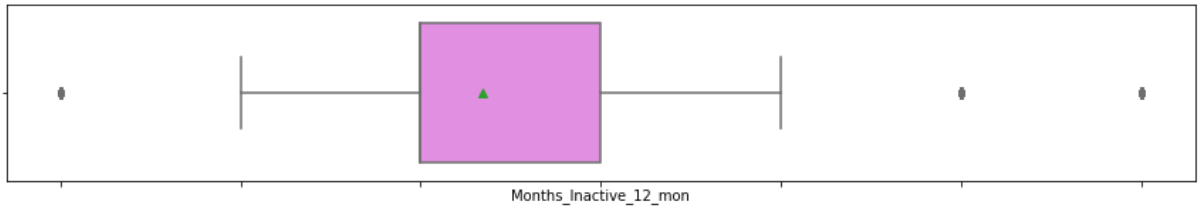
- Normal Distribution

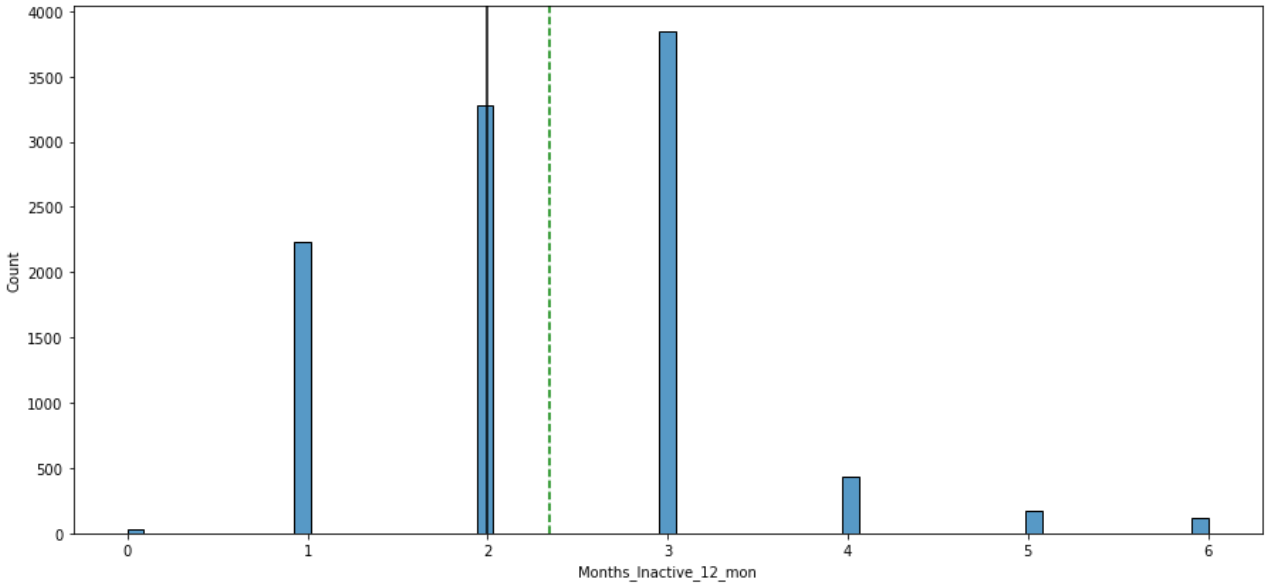
In [16]:
```
# Observation on Total_Relationship_Count
histogram_boxplot(data, "Total_Relationship_Count")
```



- Normal Distribution

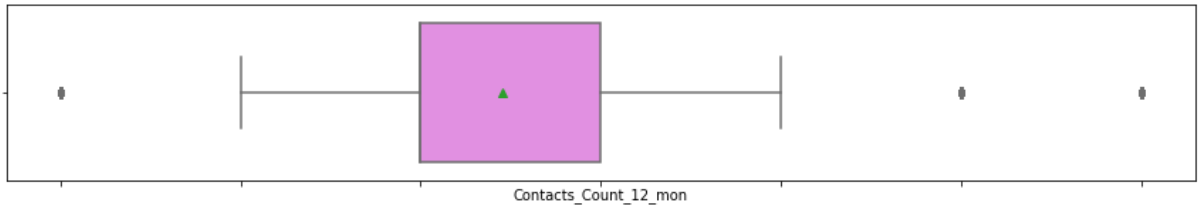
In [17]:
```
# Observation on Months_Inactive_12_mon
histogram_boxplot(data, "Months_Inactive_12_mon")
```

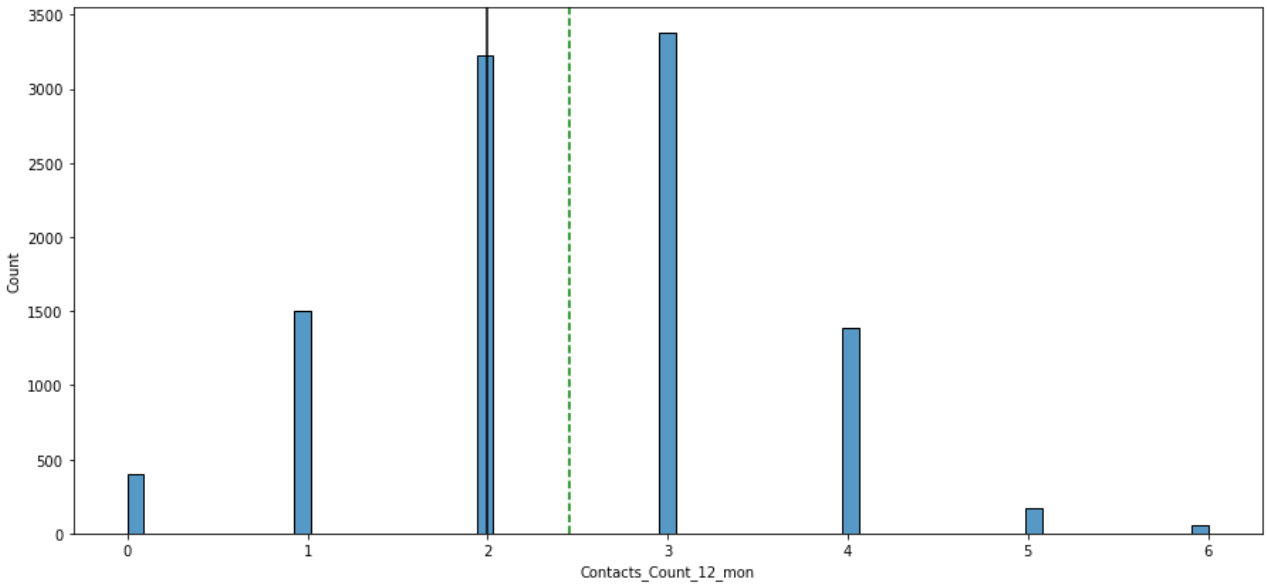




- Normal Distribution

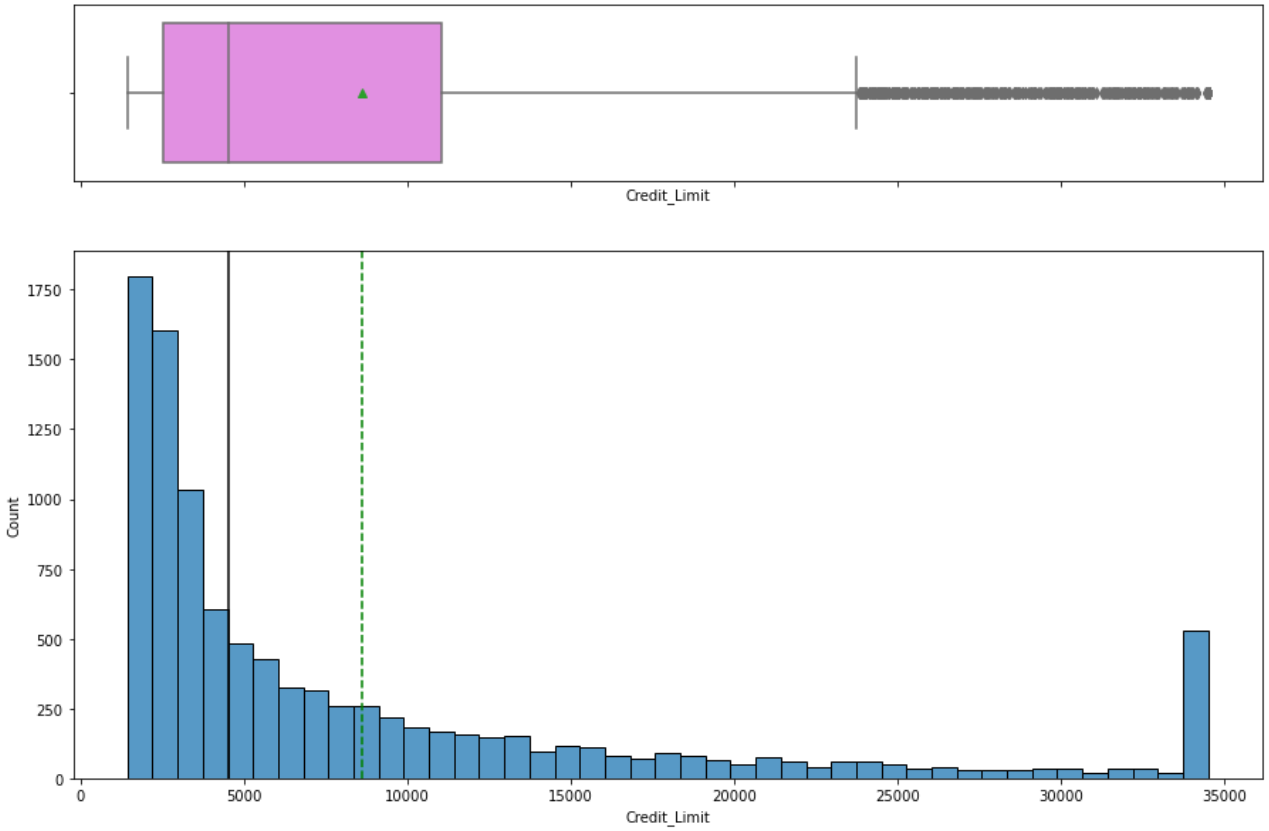
In [18]:
```
# Observation on Contacts_Count_12_mon
histogram_boxplot(data, "Contacts_Count_12_mon")
```





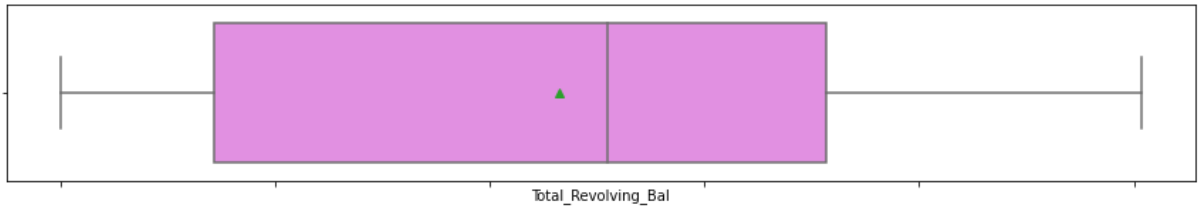
- Normal Distribution

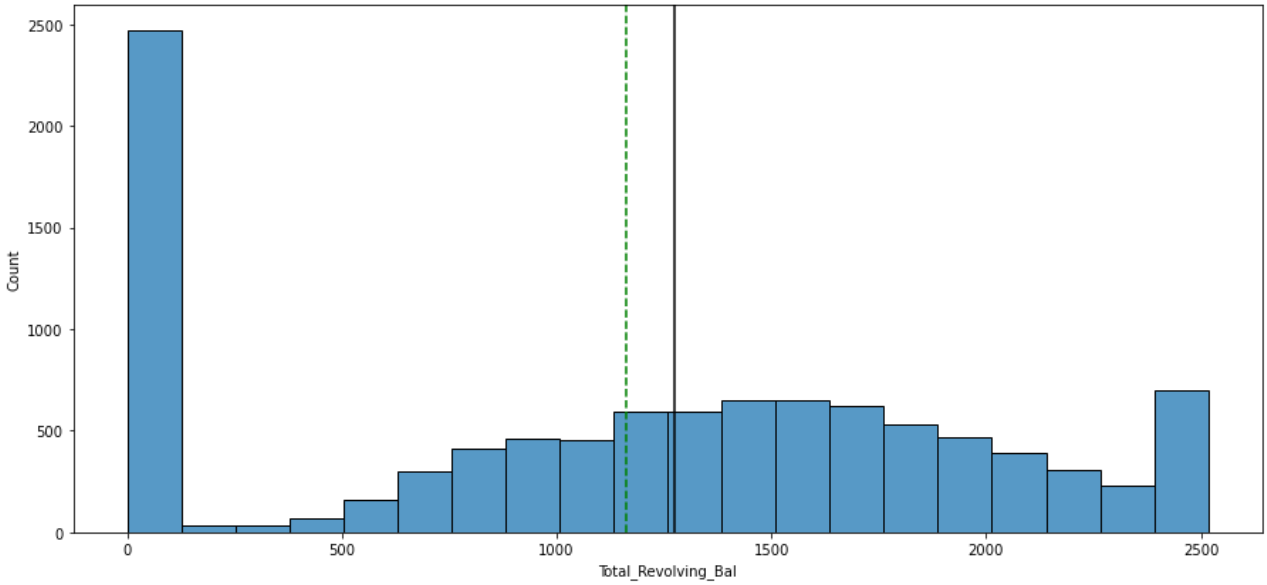
In [19]:
```
# Observation on Credit_Limit
histogram_boxplot(data, "Credit_Limit")
```



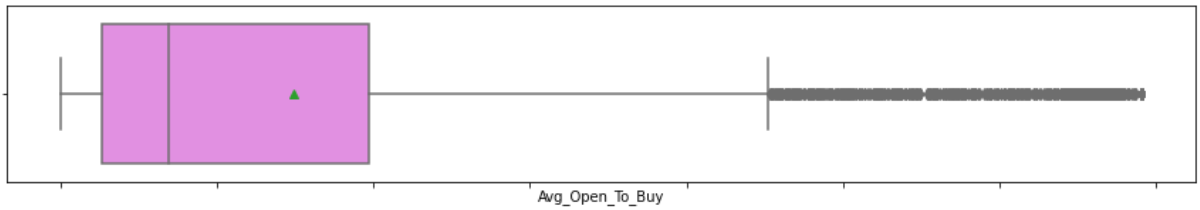
- Right-skewed

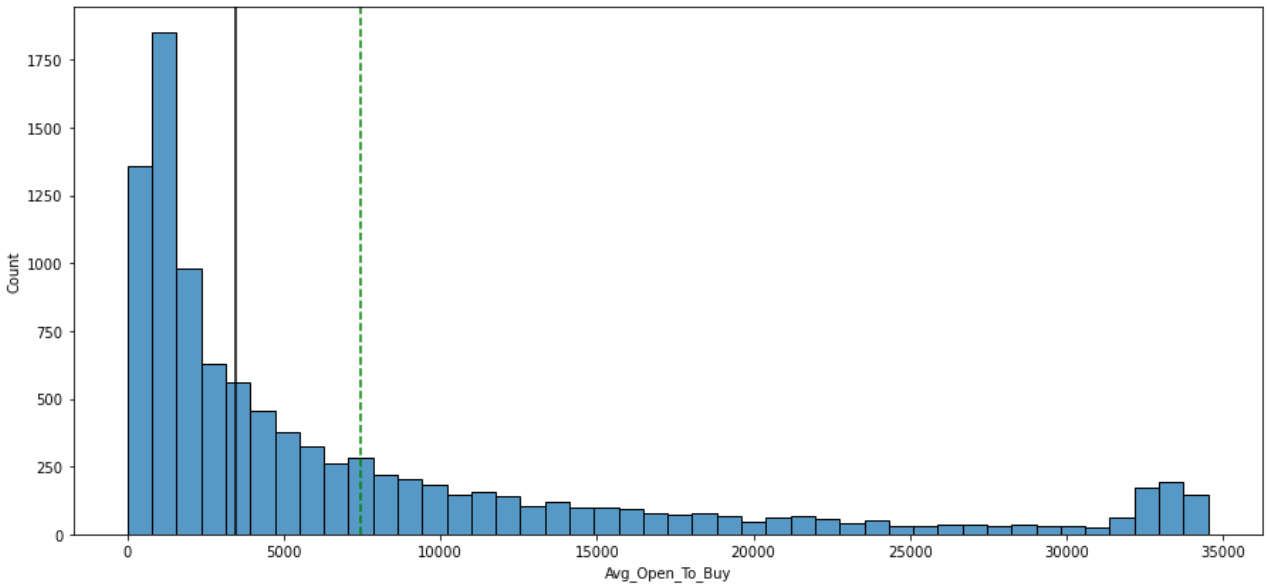
In [20]:
```
# Observation on Total_Revolving_Bal
histogram_boxplot(data, "Total_Revolving_Bal")
```





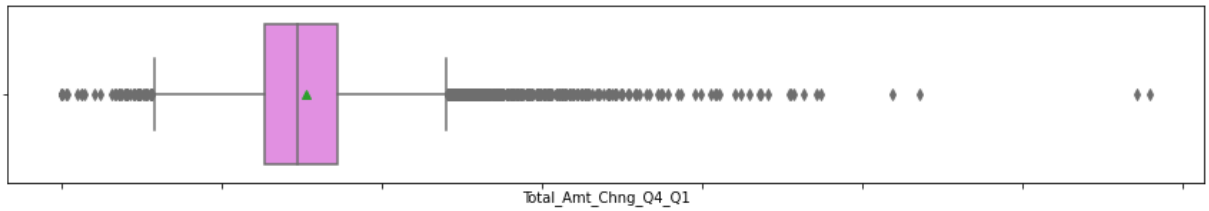
In [21]:
```
# Observation on Avg_Open_To_Buy
histogram_boxplot(data, "Avg_Open_To_Buy")
```

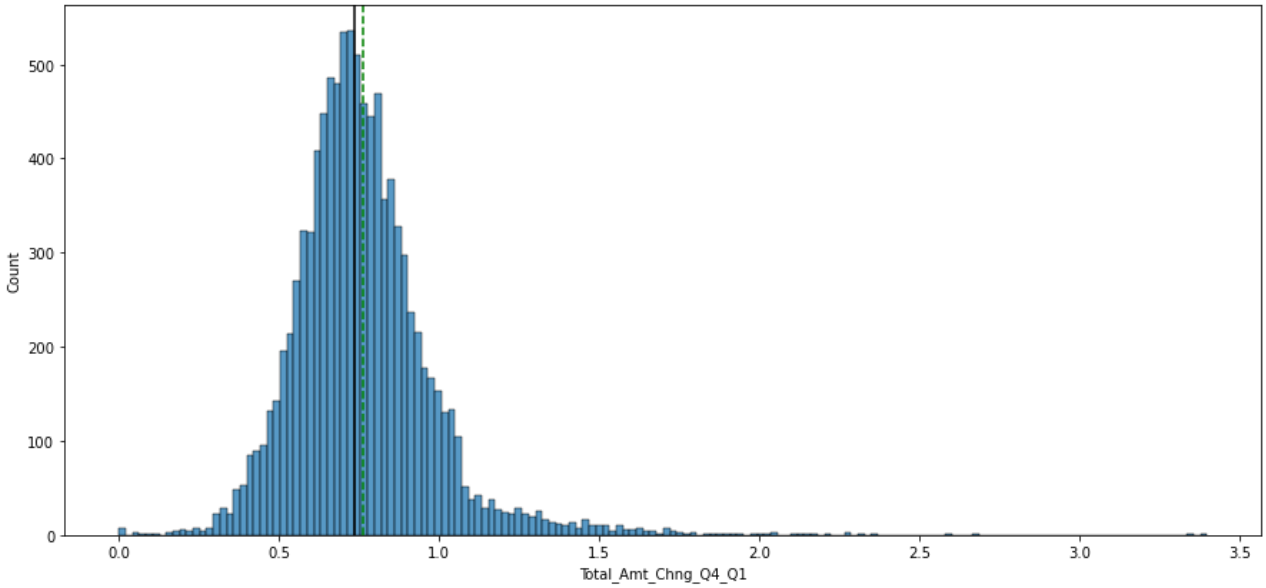




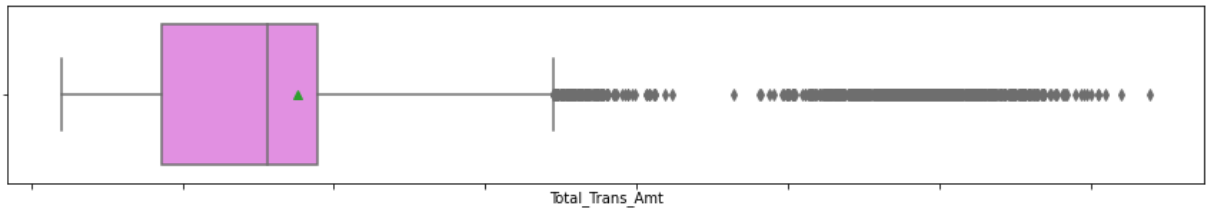
- Right-skewed

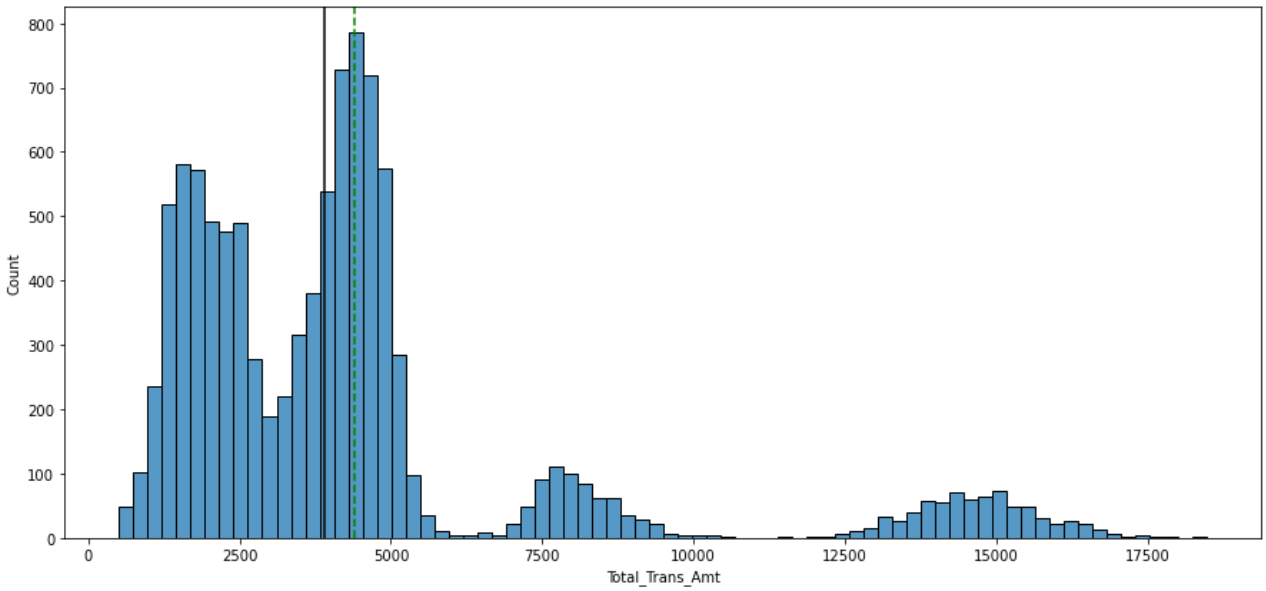
In [22]:
```
# Observation on Total_Amt_Chng_Q4_Q1
histogram_boxplot(data, "Total_Amt_Chng_Q4_Q1")
```





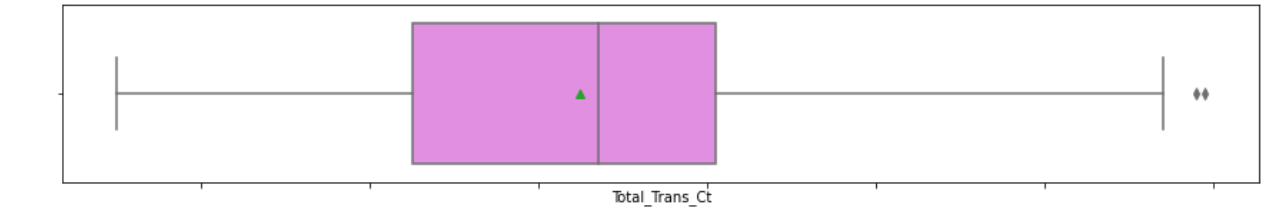
In [23]:
```
# Observation on Total_Trans_Amt
histogram_boxplot(data, "Total_Trans_Amt")
```

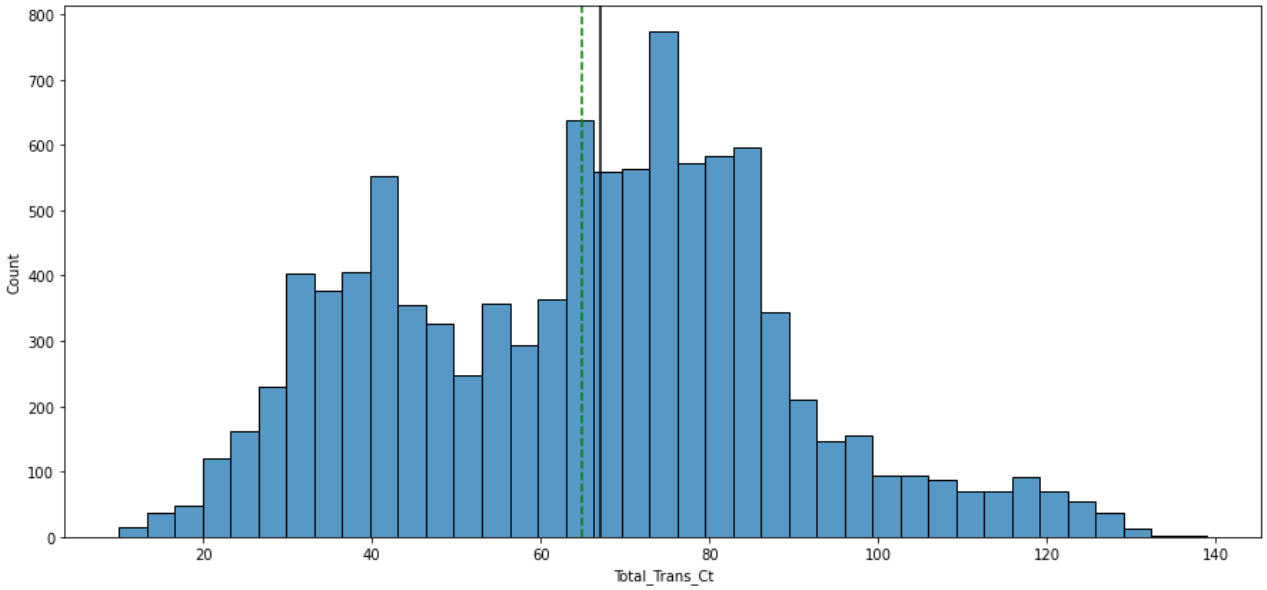




- Right-skewed

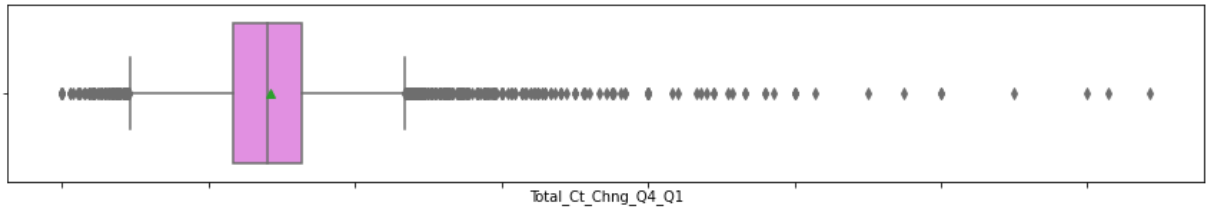
In [24]:
```
# Observation on Total_Trans_Ct
histogram_boxplot(data, "Total_Trans_Ct")
```

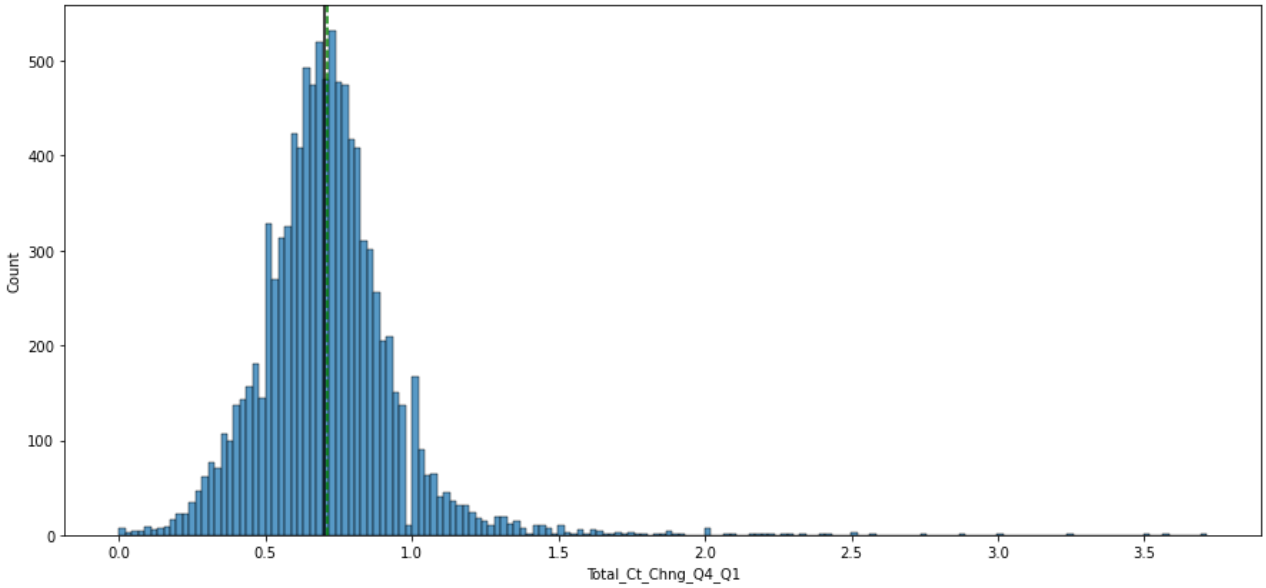




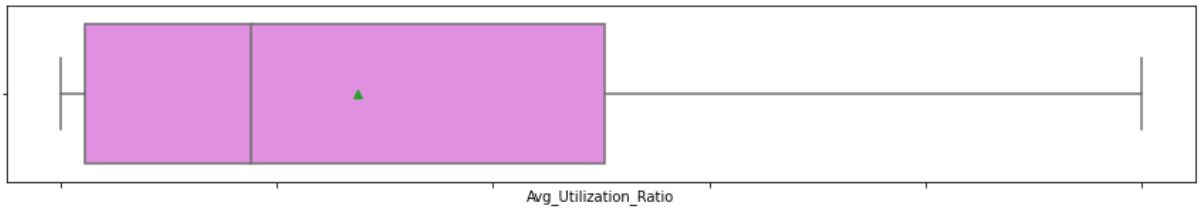
- Normal Distribution

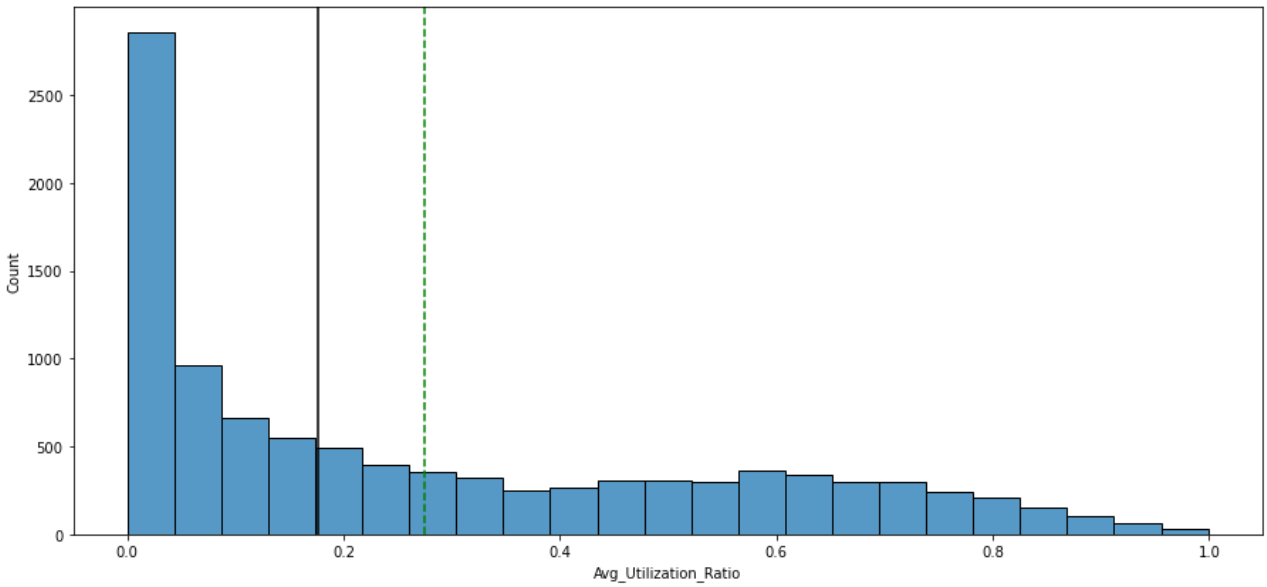
In [25]:
```
# Observation on Total_Ct_Chng_Q4_Q1
histogram_boxplot(data, "Total_Ct_Chng_Q4_Q1")
```





In [26]:
```
# Observation on Avg_Utilization_Ratio
histogram_boxplot(data, "Avg_Utilization_Ratio")
```

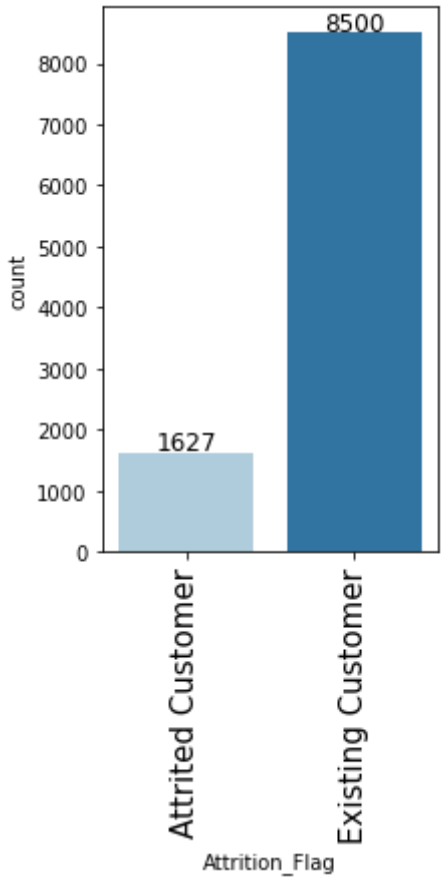




- Right-skewed

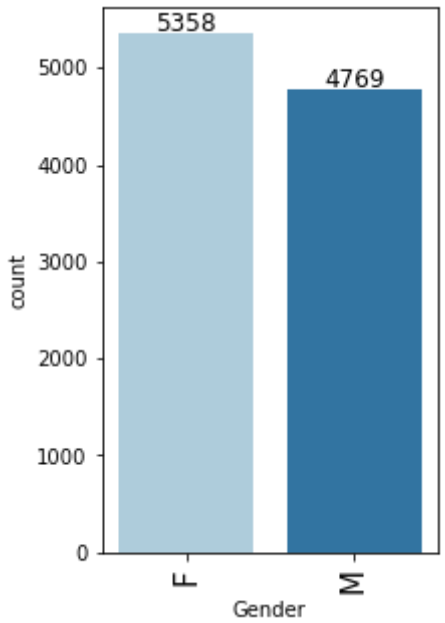
In [27]:
```
# function to create labeled barplots


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

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

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

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

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

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

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

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

In [28]:
```
# observations on Attrition_Flag
labeled_barplot(data, "Attrition_Flag")
```



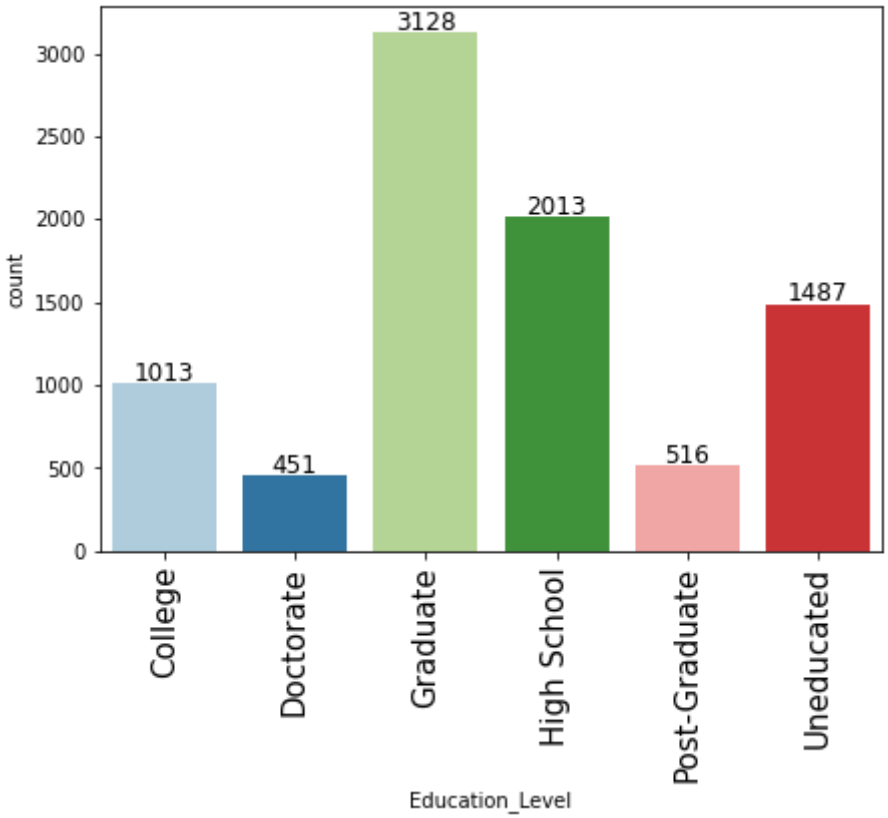
- There is ~5X more Existing Customers than Attrited Customers.

In [29]:
```
# observations on Gender
labeled_barplot(data, "Gender")
```



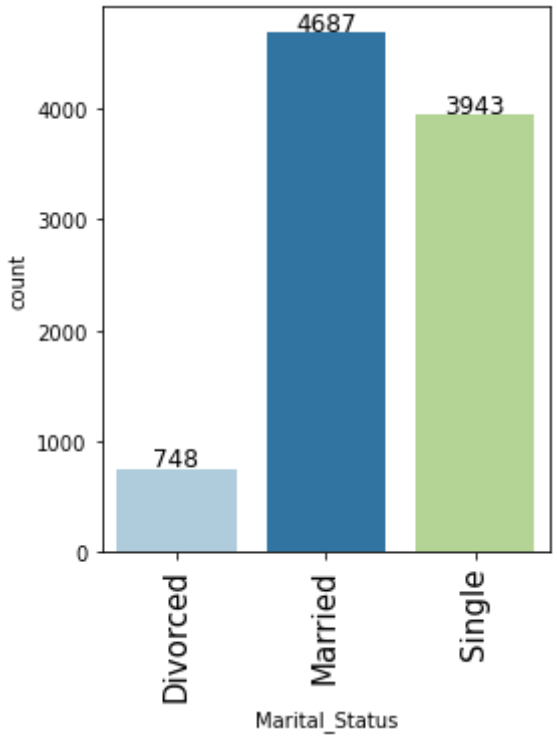
- There are slightly more Females than Males.

In [30]:
```
# observations on Education_Level
labeled_barplot(data, "Education_Level")
```



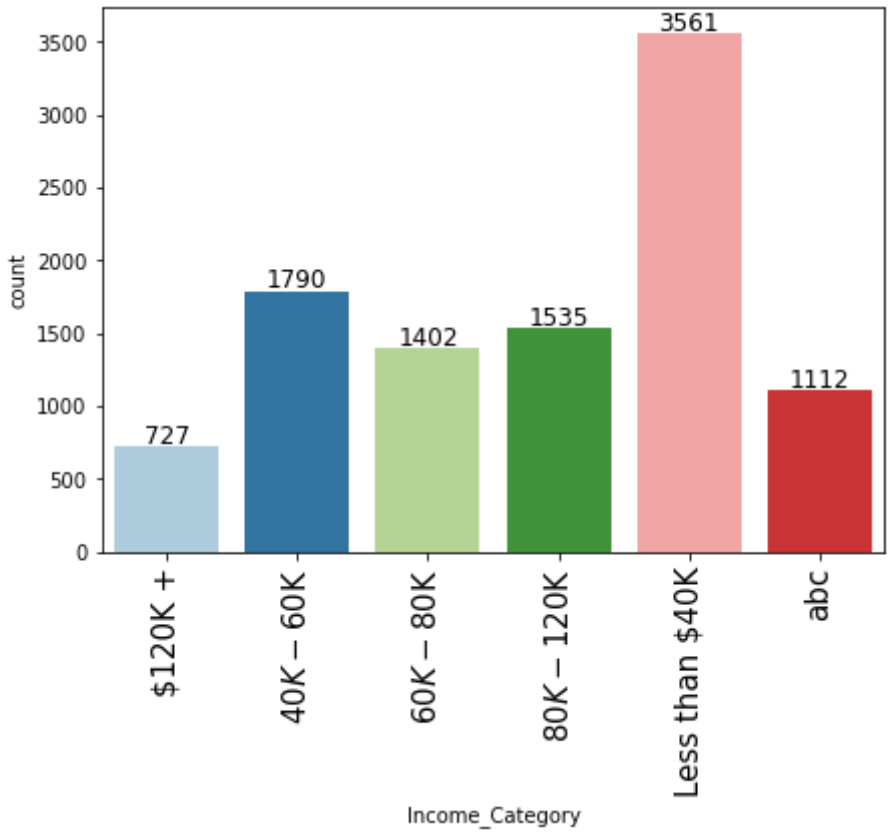
- Graduate and High School are the two most frequent education level.

In [31]:
```
# observations on Marital_Status
labeled_barplot(data, "Marital_Status")
```



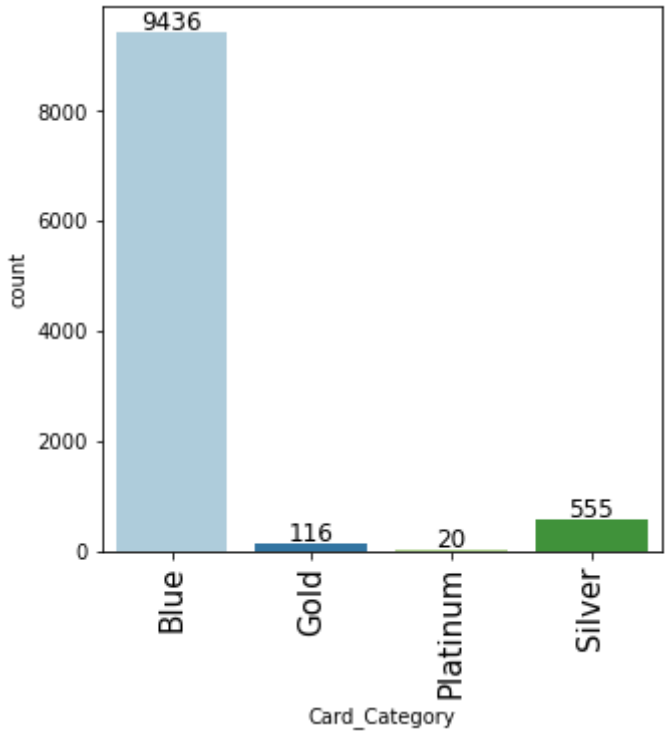
- Married is the most frequent, followed by Single.

In [32]:
```
# observations on Income_Category
labeled_barplot(data, "Income_Category")
```



- Most are in the 'Less than $40k' category.

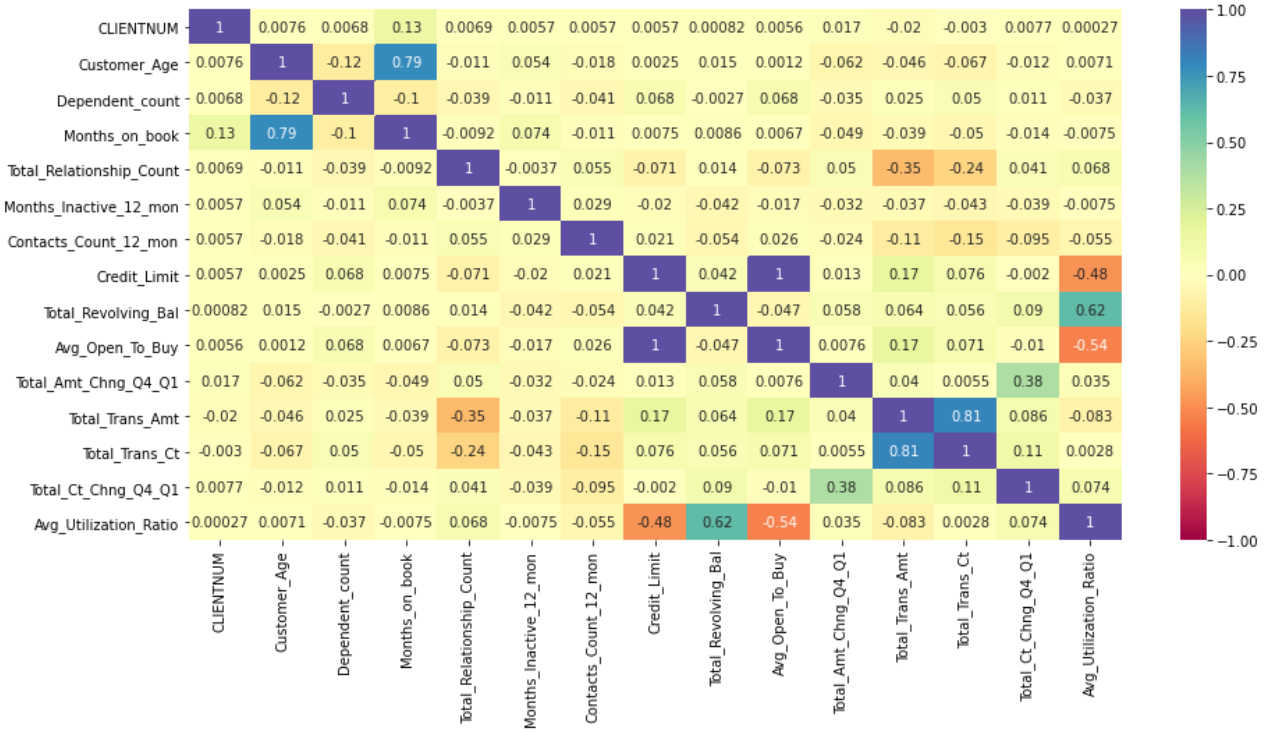
In [33]:
```
# observations on Card_Category
labeled_barplot(data, "Card_Category")
```



- The 'Blue' card is by far the most frequent with 'Platinum' being a tiny fraction.

Bivariate

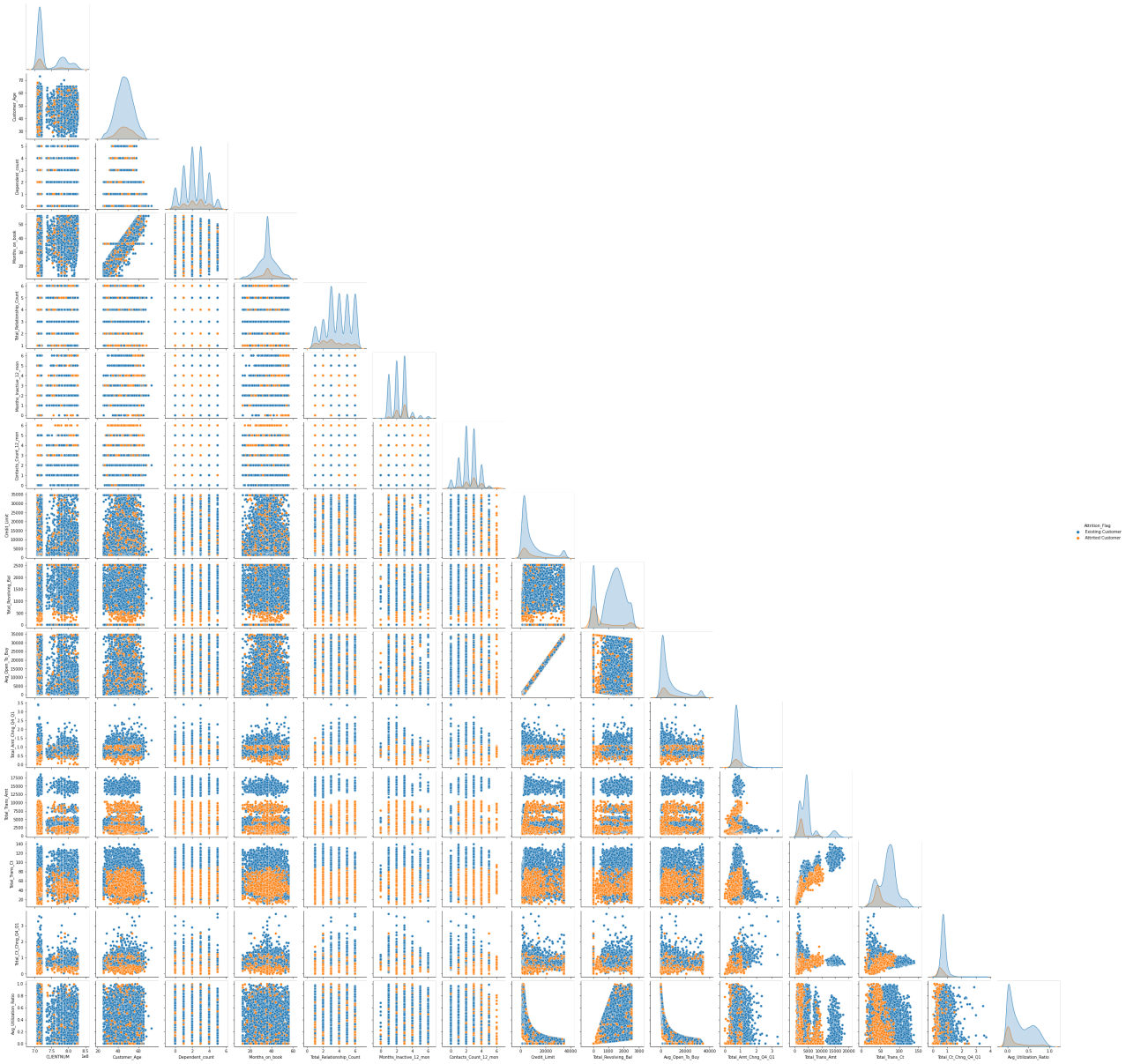
In [34]:
```
plt.figure(figsize=(15, 7))
sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, cmap="Spectral")
plt.show()
```



- Credit_Limit and Avg_Open_To_Buy are perfectly correlated, so will drop one of them.
- Avg_Utilization_Ratio has correlations with three other variables.

In [35]:
```
sns.pairplot(df, hue="Attrition_Flag", corner=True)
```

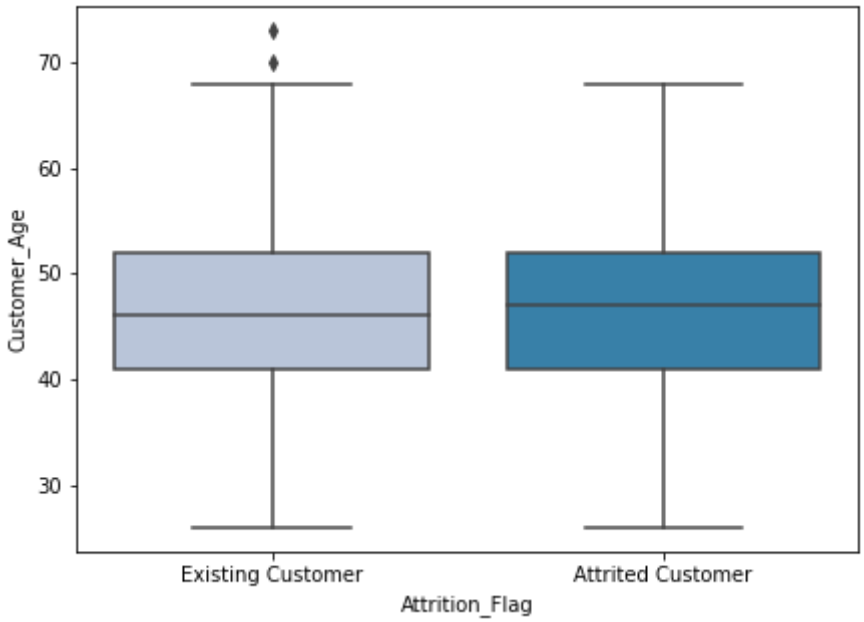
Out[35]:
```
<seaborn.axisgrid.PairGrid at 0x26a3101a280>
```



- There looks to be a clear distinction between existing customers and attrited customers in the Total_Trans_Amt, Total_Trans_Ct, and Total_Ct_Chng_Q4_Q1 variables.

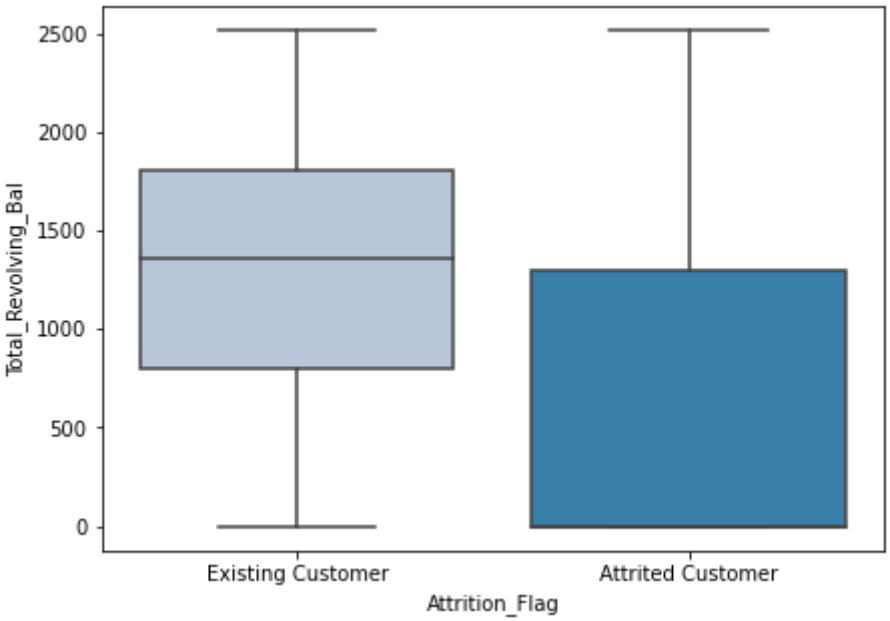
In [36]:
```
def boxplot(x):
    plt.figure(figsize=(7, 5))
    sns.boxplot(df["Attrition_Flag"], x, palette="PuBu")
    plt.show()
```

In [37]:
```
boxplot(df["Customer_Age"])
```



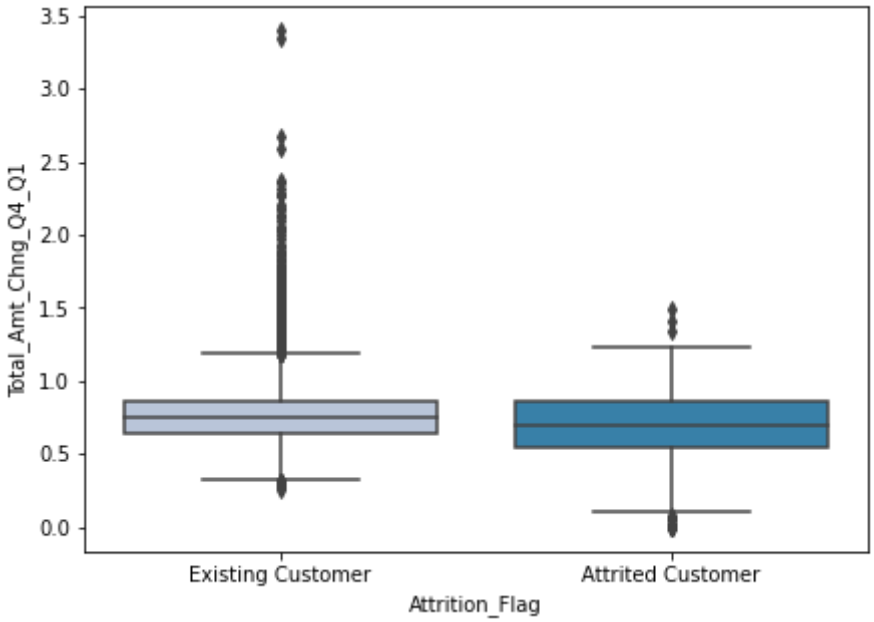
- Customer_Age seems to not factor in the type of customer.

In [38]:
```
boxplot(df["Total_Revolving_Bal"])
```

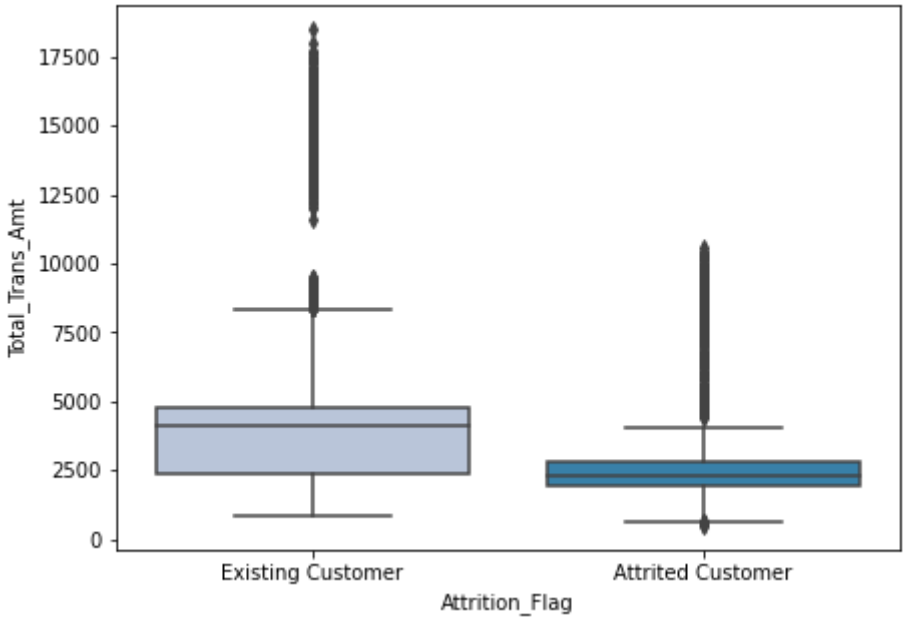


- The higher the balance that is transfered from month to month, the more likely the customer is still with the bank.

In [39]:
```
boxplot(df["Total_Amt_Chng_Q4_Q1"])
```

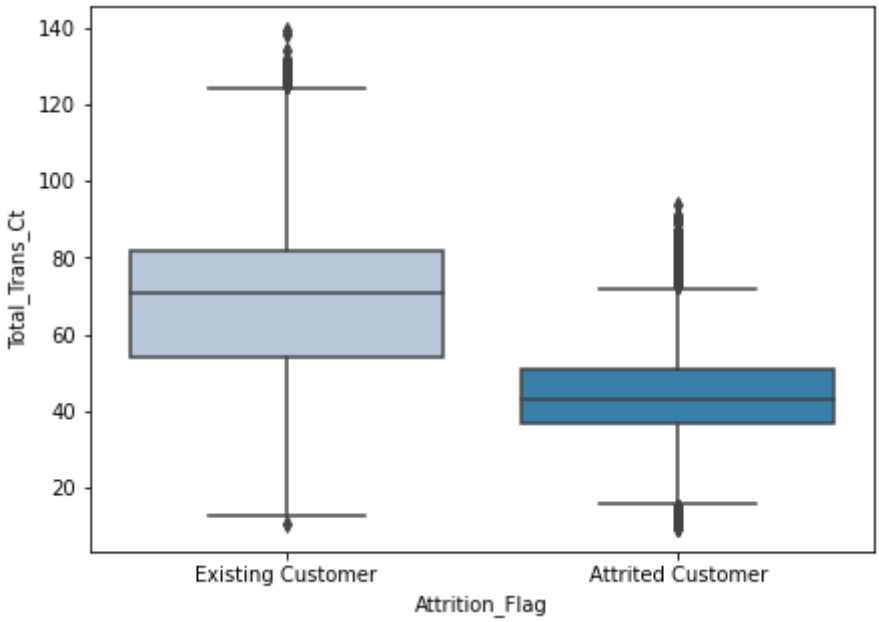


In [40]:
```
boxplot(df["Total_Trans_Amt"])
```



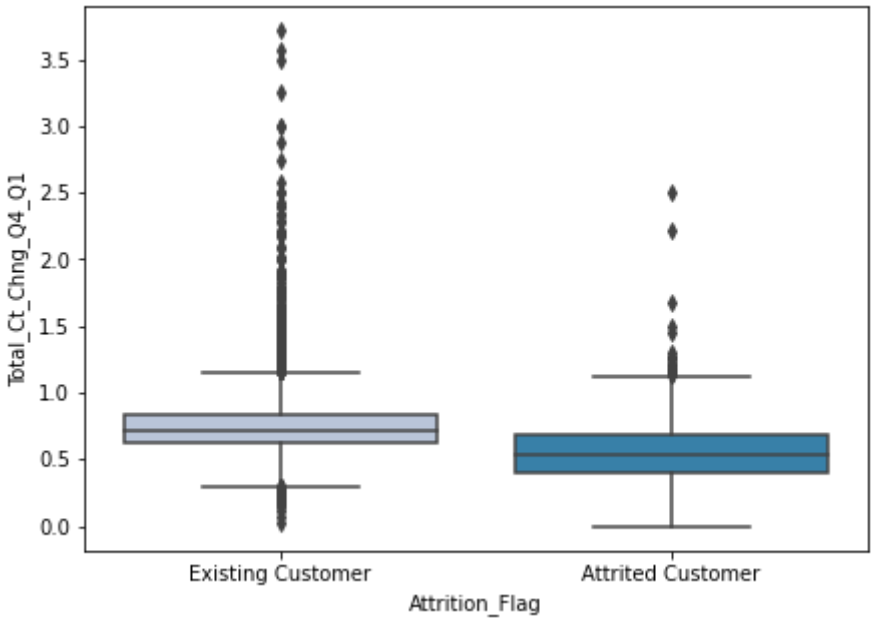
- Customers that make transactions of 2,500 or more, are likely still be with the bank.

In [41]:
```
boxplot(df["Total_Trans_Ct"])
```

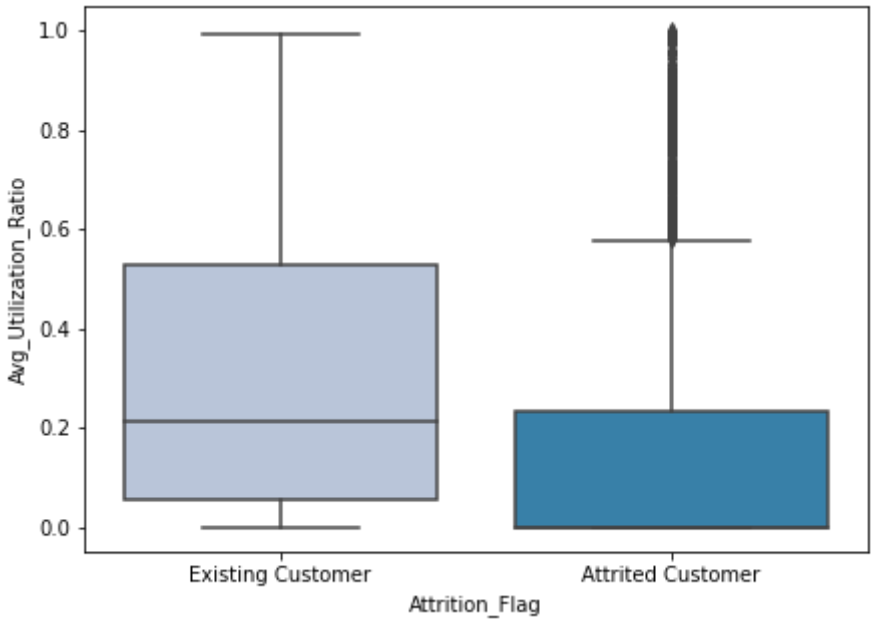


- The more transations made, the greater the chance the customer is still with the bank.

In [42]:
```
boxplot(df["Total_Ct_Chng_Q4_Q1"])
```



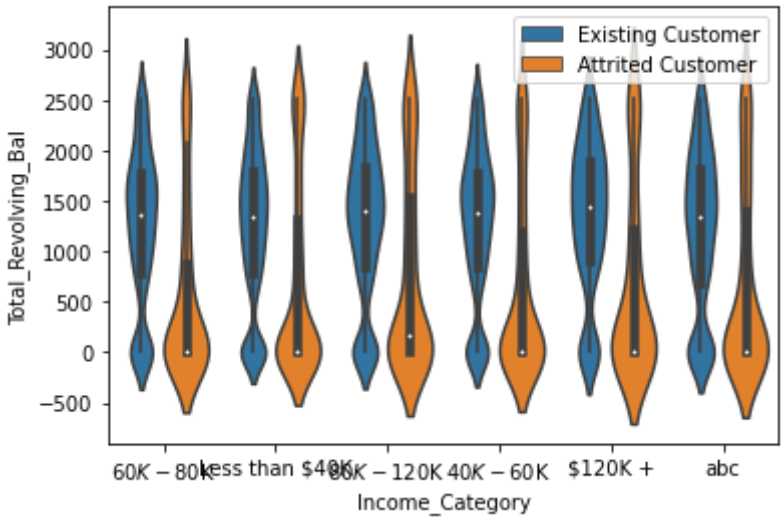
In [43]:
```
boxplot(df["Avg_Utilization_Ratio"])
```



- If the customer spends more of their available credit, the greater the chances the person is still with the bank.

In [44]:
```
sns.violinplot(
    df["Income_Category"], df["Total_Revolving_Bal"], hue=df["Attrition_Flag"]
)
plt.legend(bbox_to_anchor=(1, 1))
```

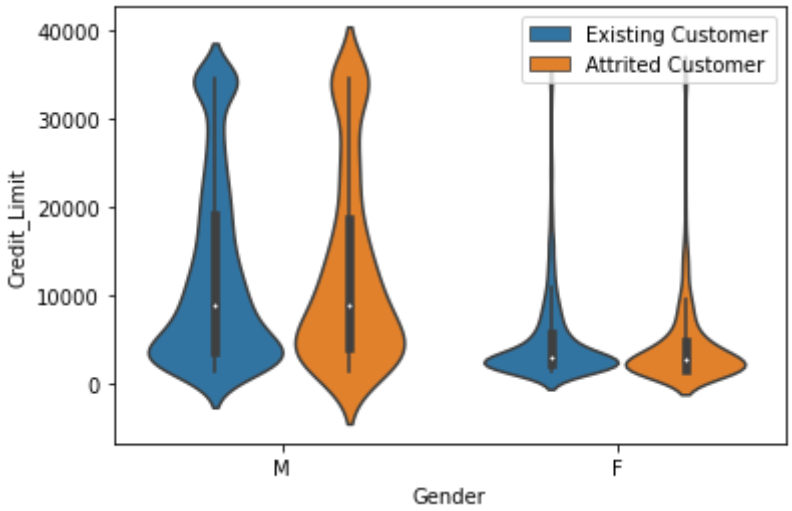
Out[44]: <matplotlib.legend.Legend at 0x26a312130a0>



- The balance transfer among all income categories and existing custmer all look compairable.

In [45]:
```
sns.violinplot(df["Gender"], df["Credit_Limit"], hue=df["Attrition_Flag"])
plt.legend(bbox_to_anchor=(1, 1))
```

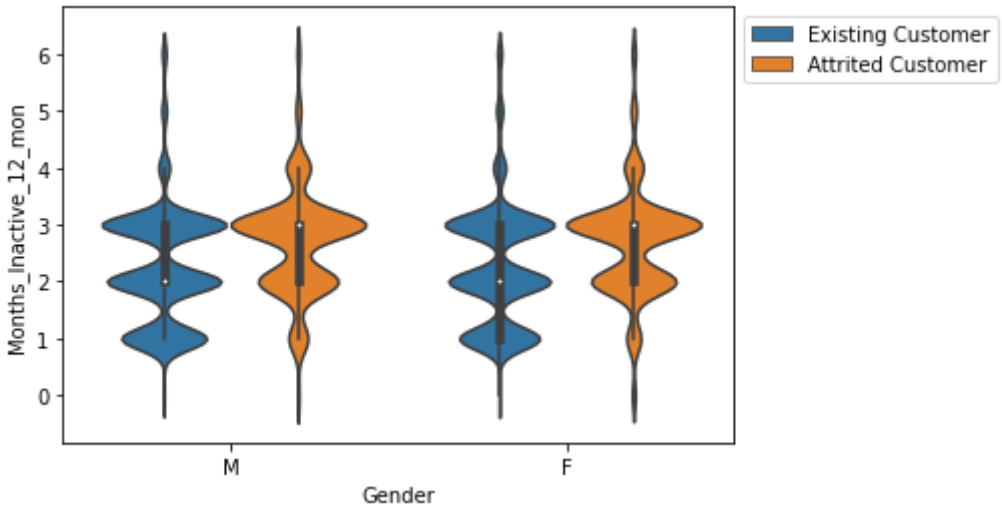
Out[45]: <matplotlib.legend.Legend at 0x26a3d47c9a0>



- Credit limit doesn't seems to be much of a factor in determining ones relation with the bank.

In [46]:
```
sns.violinplot(df["Gender"], df["Months_Inactive_12_mon"], hue=df["Attrition_Fla
plt.legend(bbox_to_anchor=(1, 1))
```

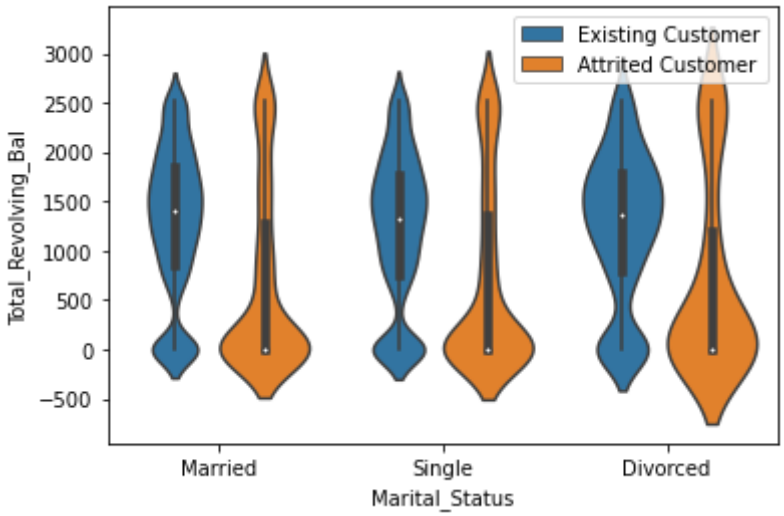
Out[46]:
```
<matplotlib.legend.Legend at 0x26a3d4a3b50>
```



- Months incactive also seems to be a weak factor in determining ones relation with the bank.

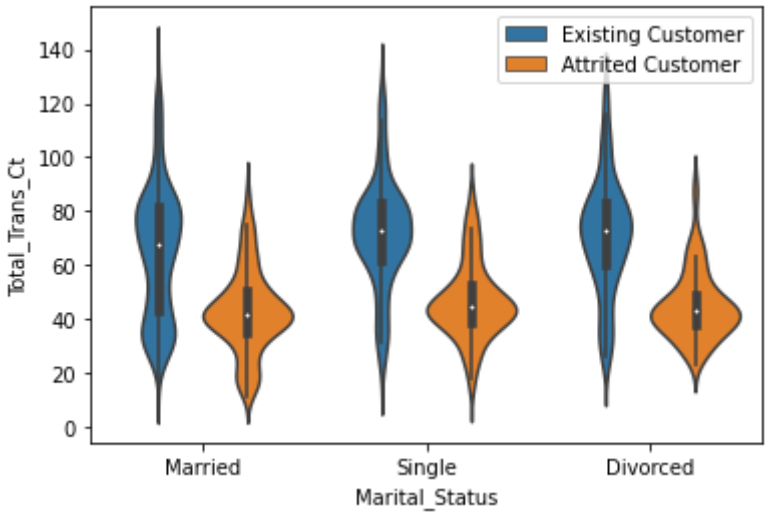
In [47]:
```
sns.violinplot(
    df["Marital_Status"], df["Total_Revolving_Bal"], hue=df["Attrition_Flag"]
)
plt.legend(bbox_to_anchor=(1, 1))
```

Out[47]:
```
<matplotlib.legend.Legend at 0x26a3d555070>
```



In [48]:
```
sns.violinplot(df["Marital_Status"], df["Total_Trans_Ct"], hue=df["Attrition_Fla
plt.legend(bbox_to_anchor=(1, 1))
```

Out[48]:
```
<matplotlib.legend.Legend at 0x26a3d5f3340>
```



- Marital status is a weak factor in determining bank relations.

Data pre-processing

Feature Engineering

In [49]:
```
df1 = df.copy()
```

In [50]:
```
df1.drop(["CLIENTNUM"], inplace=True, axis=1)
```

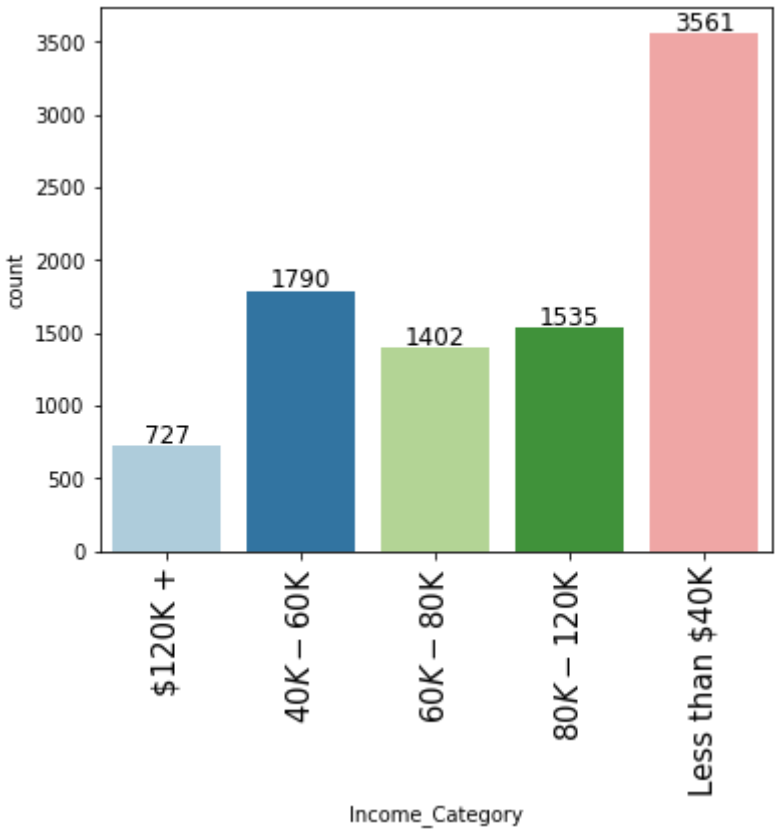
- Dropped "CLIENTNUM" as IDs are useless.

In [51]:
```
df1.drop(["Credit_Limit"], inplace=True, axis=1)
```

- Dropped "Credit_Limit" as it is perfectly correlated with "Avg_Open_To_Buy".

In [52]:
```
df1["Income_Category"].replace("abc", np.NaN, inplace=True)
```

In [53]:
```
labeled_barplot(df1, "Income_Category")
```



In [54]:
```
df1["Income_Category"].isnull().sum()
```

Out[54]:
```
1112
```

- Changed 'abc' values to np.NaN and will impute later.

Encoding Target Variable

In [55]:
```
def changer(x):
    if x == "Existing Customer":
        return 0
    elif x == "Attrited Customer":
        return 1


df1["Attrition_Flag"] = df1.Attrition_Flag.apply(changer)
```

In [56]:
```
df1.head()
```

Out[56]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Inco
0	0	45	M	3	High School	Married	
1	0	49	F	5	Graduate	Single	L
2	0	51	M	3	Graduate	Married	
3	0	40	F	4	High School	NaN	L
4	0	40	M	3	Uneducated	Married	

Spliting Data

In [57]:
```
X = df1.drop("Attrition_Flag", axis=1)
y = df1["Attrition_Flag"]
```

In [58]:
```
# Splitting data into training, validation and test set:
# split data into 2 parts, 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 split the temporary set into train and validation

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

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

Missing-Value Treatment

In [59]:
```
df1.isnull().sum()
```

Out[59]:
```
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
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 [60]:
```
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              928
Marital_Status               457
Income_Category              654
Card_Category                  0
Months_on_book                 0
Total_Relationship_Count       0
Months_Inactive_12_mon         0
Contacts_Count_12_mon          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              294
Marital_Status               140
Income_Category              221
Card_Category                  0
Months_on_book                 0
Total_Relationship_Count       0
Months_Inactive_12_mon         0
Contacts_Count_12_mon          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              297
Marital_Status               152
Income_Category              237
Card_Category                  0
Months_on_book                 0
Total_Relationship_Count       0
Months_Inactive_12_mon         0
Contacts_Count_12_mon          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 [61]:
```
# defining a list with names of columns that will be used for imputation
reqd_col_for_impute = [
    "Education_Level",
    "Marital_Status",
    "Income_Category",
]
```

In [62]:
```
imputer = SimpleImputer(strategy="most_frequent")

# 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 [63]:
```
# 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
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
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
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
```

Encoding Categorical Variables

In [64]:
```
X_train = pd.get_dummies(X_train, drop_first=True)
X_val = pd.get_dummies(X_val, drop_first=True)
X_test = pd.get_dummies(X_test, drop_first=True)
print(X_train.shape, X_val.shape, X_test.shape)
```

```
(6075, 28) (2026, 28) (2026, 28)
```

In [65]:
```
X_train.head()
```

Out[65]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inact
800	40	2	21	6	
498	44	1	34	6	
4356	48	4	36	5	
407	41	2	36	6	
8728	46	4	36	2	

Building Models

Model evaluation criterion:

Model Prediction Errors

1. Predicting someone as an attrited customer, but isn't (FP) (Attrited customer, but is actually existing customer)
2. Predicting someone that is not an attrited customer, but is. (FN) (Existing customer, but is actually attrited customer)

Which is more important?

We want to create a model that accurately predicts the attrited customers, so we want to lower false-negatives.

Which metric to optimize?

Since we want to lower false-negatives, we will want to emphasize the recall score.

Oversampling train data using SMOTE

In [66]:
```
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 ==


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

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

After UpSampling, the shape of train_X: (10198, 28)
After UpSampling, the shape of train_y: (10198,)
```

Undersampling train data using Random Under Sampler

In [67]:
```
rus = RandomUnderSampler(random_state=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

In [68]:
```
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 ==
```

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

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

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

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

After Under Sampling, the shape of train_X: (1952, 28)
After Under Sampling, the shape of train_y: (1952,)
```

Functions for scoring and matrix

In [69]:
```
# defining a function to compute different metrics to check performance of a cla
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performa

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

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

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

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1": f1,
        },
        index=[0],
    )

    return df_perf
```

In [70]:
```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
```

```
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().su
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

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

Bagging

In [71]:
```
# Fitting the model
bagging_classifier = BaggingClassifier(random_state=1)
bagging_classifier.fit(X_train, y_train)
```

Out[71]:
```
BaggingClassifier(random_state=1)
```

In [72]:
```
# Calculating different metrics
bagging_classifier_model_train_perf = model_performance_classification_sklearn(
    bagging_classifier, X_train, y_train
)
print("Training performance:")
bagging_classifier_model_train_perf
```

```
Training performance:
```

Out[72]:

	Accuracy	Recall	Precision	F1
0	0.997	0.986	0.996	0.991

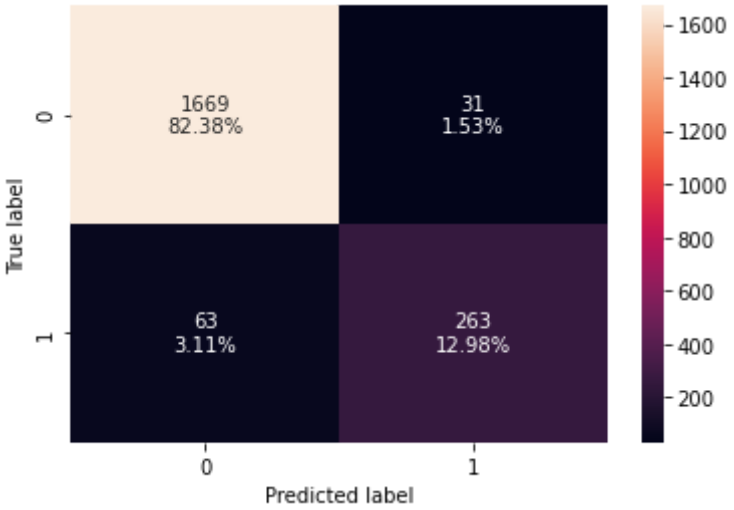
In [73]:
```
bagging_classifier_model_val_perf = model_performance_classification_sklearn(
    bagging_classifier, X_val, y_val
)
print("Validation performance:")
bagging_classifier_model_val_perf
```

```
Validation performance:
```

Out[73]:

	Accuracy	Recall	Precision	F1
0	0.954	0.807	0.895	0.848

In [74]:
```
# creating confusion matrix
confusion_matrix_sklearn(bagging_classifier, X_val, y_val)
```



- The model is overfitting.

Bagging on over-sampled train data

In [75]:
```
# Bagging on over-sampled train data
bagging_over = BaggingClassifier(random_state=1)
bagging_over.fit(X_train_over, y_train_over)
```

Out[75]:
```
BaggingClassifier(random_state=1)
```

In [76]:
```
# Calculating different metrics
bagging_over_model_over_train_perf = model_performance_classification_sklearn(
    bagging_over, X_train_over, y_train_over
)
print("Training performance:")
bagging_over_model_over_train_perf
```

```
Training performance:
```

Out[76]:

	Accuracy	Recall	Precision	F1
0	0.998	0.997	0.999	0.998

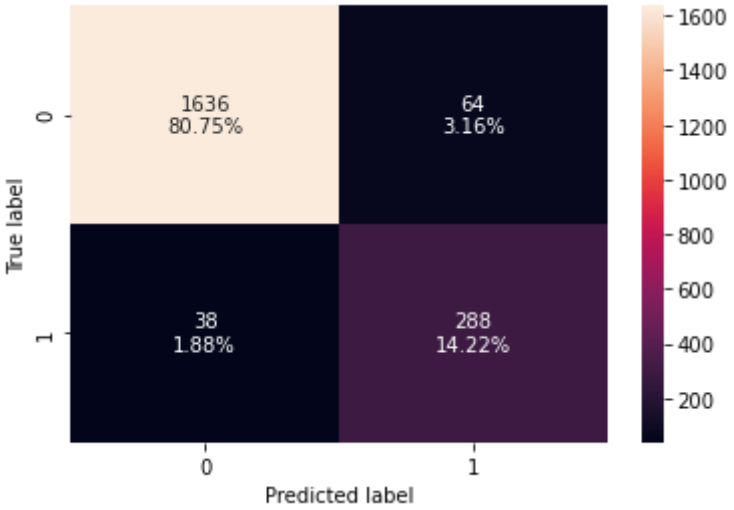
In [77]:
```
bagging_over_model_val_perf = model_performance_classification_sklearn(
    bagging_over, X_val, y_val
)
print("Validation performance:")
bagging_over_model_val_perf
```

```
Validation performance:
```

Out[77]:

	Accuracy	Recall	Precision	F1
0	0.950	0.883	0.818	0.850

In [78]:
```
# creating confusion matrix
confusion_matrix_sklearn(bagging_over, X_val, y_val)
```



- Performance is better, but still overfitting.

Bagging on under-sampled train data

In [79]:
```
# Bagging on under-sampled train data
bagging_under = BaggingClassifier(random_state=1)
bagging_under.fit(X_train_un, y_train_un)
```

Out[79]:
```
BaggingClassifier(random_state=1)
```

In [80]:
```
# Calculating different metrics
bagging_under_model_under_train_perf = model_performance_classification_sklearn(
    bagging_under, X_train_un, y_train_un
)
print("Training performance:")
bagging_under_model_under_train_perf
```

Out[80]:
```
Training performance:
```

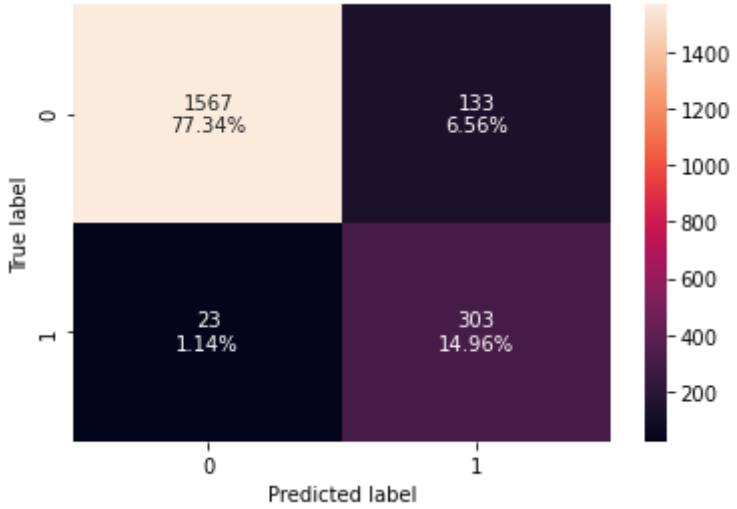
	Accuracy	Recall	Precision	F1
0	0.996	0.993	0.999	0.996

In [81]:
```
bagging_under_model_val_perf = model_performance_classification_sklearn(
    bagging_under, X_val, y_val
)
print("Validation performance:")
bagging_under_model_val_perf
```

Out[81]:
```
Validation performance:
```

	Accuracy	Recall	Precision	F1
0	0.923	0.929	0.695	0.795

In [82]:
```
# creating confusion matrix
confusion_matrix_sklearn(bagging_under, X_val, y_val)
```



- Recall and accuracy are better, but still overfitting a bit.

Random Forest

In [83]:
```
# Fitting the model
rf_estimator = RandomForestClassifier(random_state=1)
rf_estimator.fit(X_train, y_train)
```

Out[83]:
```
RandomForestClassifier(random_state=1)
```

In [84]:
```
# Calculating different metrics
rf_estimator_model_train_perf = model_performance_classification_sklearn(
    rf_estimator, X_train, y_train
)
print("Training performance:")
rf_estimator_model_train_perf
```

```
Training performance:
```

Out[84]:

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

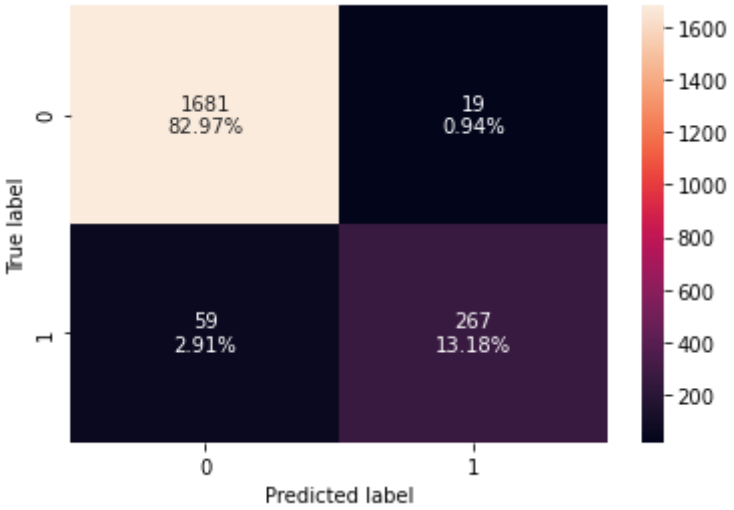
In [85]:
```
rf_estimator_model_val_perf = model_performance_classification_sklearn(
    rf_estimator, X_val, y_val
)
print("Validation performance:")
rf_estimator_model_val_perf
```

```
Validation performance:
```

Out[85]:

	Accuracy	Recall	Precision	F1
0	0.962	0.819	0.934	0.873

In [86]:
```
# creating confusion matrix
confusion_matrix_sklearn(rf_estimator, X_val, y_val)
```



- The model is overfitting.

Random Forest on over-sampled train data

In [87]:
```
# Random Forest on over-sampled train data
rf_over = RandomForestClassifier(random_state=1)
rf_over.fit(X_train_over, y_train_over)
```

Out[87]:
```
RandomForestClassifier(random_state=1)
```

In [88]:
```
# Calculating different metrics
rf_over_model_over_train_perf = model_performance_classification_sklearn(
    rf_over, X_train_over, y_train_over
)
print("Training performance:")
rf_over_model_over_train_perf
```

```
Training performance:
```

Out[88]:

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

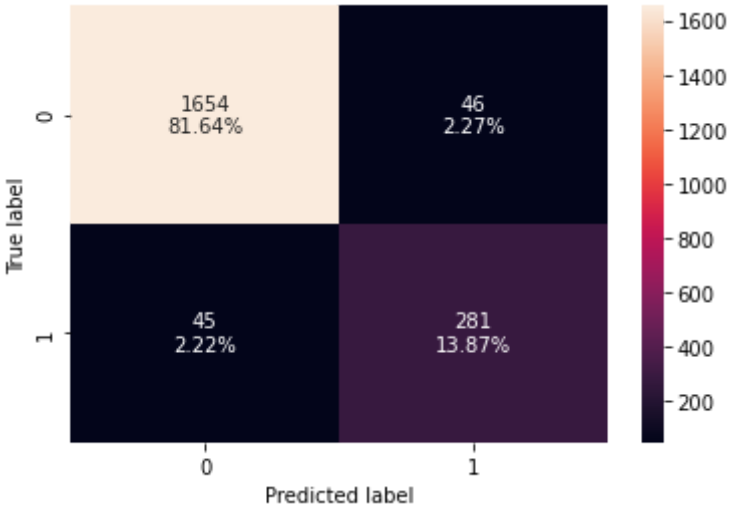
In [89]:
```
rf_over_model_val_perf = model_performance_classification_sklearn(rf_over, X_val
print("Validation performance:")
rf_over_model_val_perf
```

```
Validation performance:
```

Out[89]:

	Accuracy	Recall	Precision	F1
0	0.955	0.862	0.859	0.861

In [90]:
```
# creating confusion matrix
confusion_matrix_sklearn(rf_over, X_val, y_val)
```



- The model is still overfitting.

Random Forest on under-sampled train data

In [91]:
```
# Random Forest on over-sampled train data
rf_under = RandomForestClassifier(random_state=1)
rf_under.fit(X_train_un, y_train_un)
```

Out[91]:
```
RandomForestClassifier(random_state=1)
```

In [92]:
```
# Calculating different metrics
rf_under_model_under_train_perf = model_performance_classification_sklearn(
    rf_under, X_train_un, y_train_un
)
print("Training performance:")
rf_under_model_under_train_perf
```

Out[92]:
```
Training performance:
```

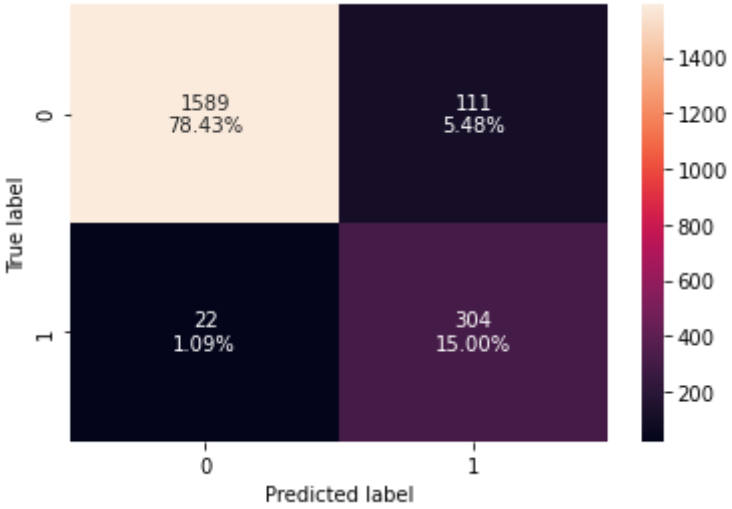
	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

In [93]:
```
rf_under_model_val_perf = model_performance_classification_sklearn(
    rf_under, X_val, y_val
)
print("Validation performance:")
rf_under_model_val_perf
```

Out[93]:
```
Validation performance:
```

	Accuracy	Recall	Precision	F1
0	0.934	0.933	0.733	0.821

In [94]:
```
# creating confusion matrix
confusion_matrix_sklearn(rf_under, X_val, y_val)
```



- The accuracy and recall are better, but the model is still overfitting.

Gradient Boost

In [95]:
```
# Fitting the model
gb_classifier = GradientBoostingClassifier(random_state=1)
gb_classifier.fit(X_train, y_train)
```

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

In [96]:
```
# Calculating different metrics
gb_classifier_model_train_perf = model_performance_classification_sklearn(
    gb_classifier, X_train, y_train
)
print("Training performance:")
gb_classifier_model_train_perf
```

Out[96]:
```
Training performance:
```

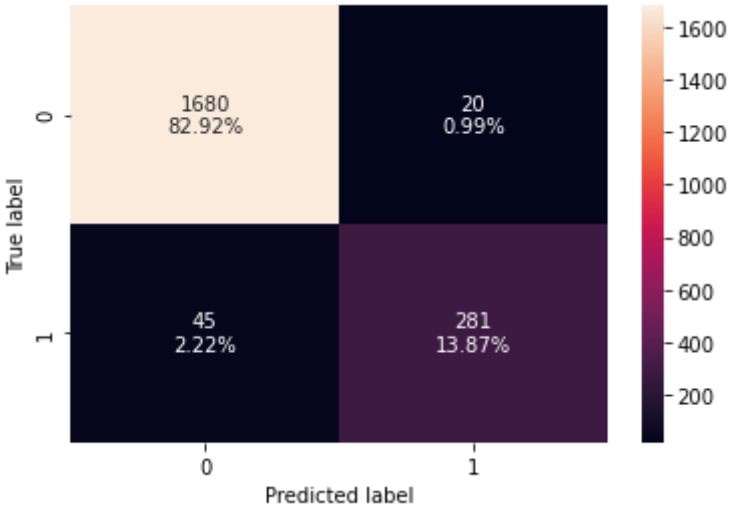
	Accuracy	Recall	Precision	F1
0	0.973	0.875	0.950	0.911

In [97]:
```
gb_classifier_model_val_perf = model_performance_classification_sklearn(
    gb_classifier, X_val, y_val
)
print("Validation performance:")
gb_classifier_model_val_perf
```

Out[97]:
```
Validation performance:
```

	Accuracy	Recall	Precision	F1
0	0.968	0.862	0.934	0.896

In [98]:
```
# creating confusion matrix
confusion_matrix_sklearn(gb_classifier, X_val, y_val)
```



- The model is generalizing well, but I would like the recall score to be higher.

Gradient Boost on over-sampled train data

In [99]:
```
# Gradient Boost on over-sampled train data
gb_over = GradientBoostingClassifier(random_state=1)
gb_over.fit(X_train_over, y_train_over)
```

Out[99]:
```
GradientBoostingClassifier(random_state=1)
```

In [100…
```
# Calculating different metrics
gb_over_model_over_train_perf = model_performance_classification_sklearn(
    gb_over, X_train_over, y_train_over
)
print("Training performance:")
gb_over_model_over_train_perf
```

```
Training performance:
```

Out[100…

	Accuracy	Recall	Precision	F1
0	0.978	0.981	0.975	0.978

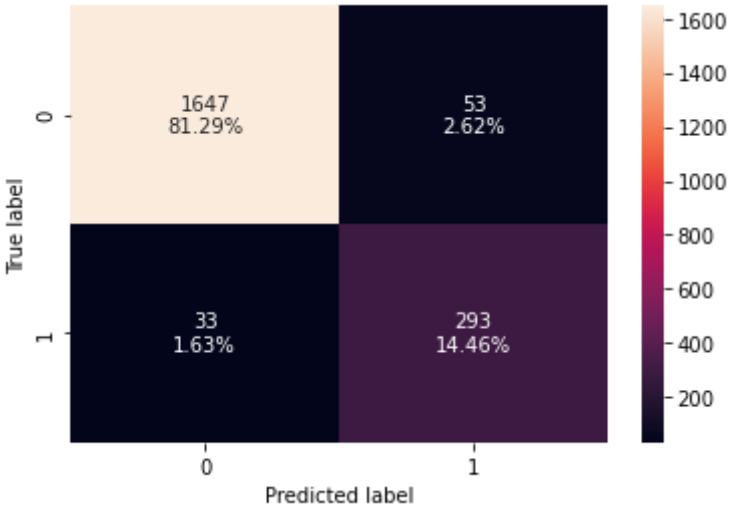
In [101…
```
gb_over_model_val_perf = model_performance_classification_sklearn(gb_over, X_val
print("Validation performance:")
gb_over_model_val_perf
```

```
Validation performance:
```

Out[101…

	Accuracy	Recall	Precision	F1
0	0.958	0.899	0.847	0.872

In [102…
```
# creating confusion matrix
confusion_matrix_sklearn(gb_over, X_val, y_val)
```



- Recall has improved slightly, but now the model is overfitting.

Gradient Boost on under-sampled train data

In [103…
```
# Gradient Boost on under-sampled train data
gb_under = GradientBoostingClassifier(random_state=1)
gb_under.fit(X_train_un, y_train_un)
```

Out[103… `GradientBoostingClassifier(random_state=1)`

In [104…
```
# Calculating different metrics
gb_under_model_under_train_perf = model_performance_classification_sklearn(
    gb_under, X_train_un, y_train_un
)
print("Training performance:")
gb_under_model_under_train_perf
```

Out[104…

Training performance:

	Accuracy	Recall	Precision	F1
0	0.973	0.980	0.967	0.973

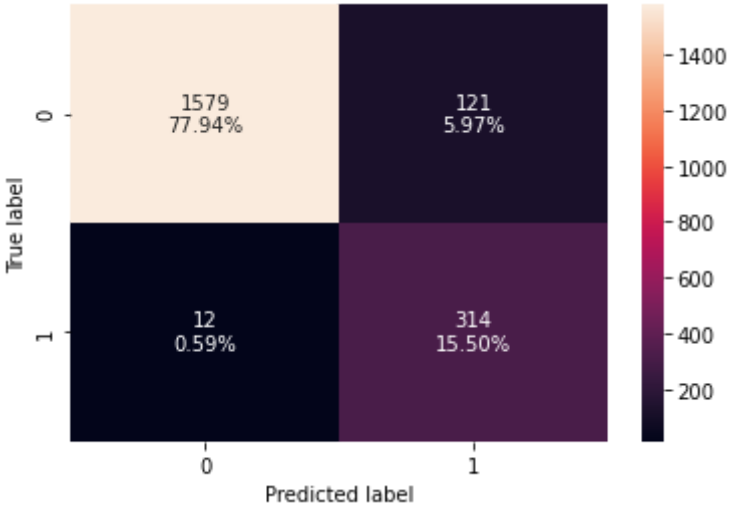
In [105…
```
gb_under_model_val_perf = model_performance_classification_sklearn(
    gb_under, X_val, y_val
)
print("Validation performance:")
gb_under_model_val_perf
```

Out[105…

Validation performance:

	Accuracy	Recall	Precision	F1
0	0.934	0.963	0.722	0.825

In [106…
```
# creating confusion matrix
confusion_matrix_sklearn(gb_under, X_val, y_val)
```



- The model is more what I'm looking for, as the recall is high and is generalizing.

AdaBoost

In [107…
```
# Fitting the model
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train, y_train)
```

Out[107… `AdaBoostClassifier(random_state=1)`

In [108…
```
# Calculating different metrics
ab_classifier_model_train_perf = model_performance_classification_sklearn(
    ab_classifier, X_train, y_train
)
print("Training performance:")
ab_classifier_model_train_perf
```

Training performance:

Out[108…

	Accuracy	Recall	Precision	F1
0	0.957	0.827	0.900	0.862

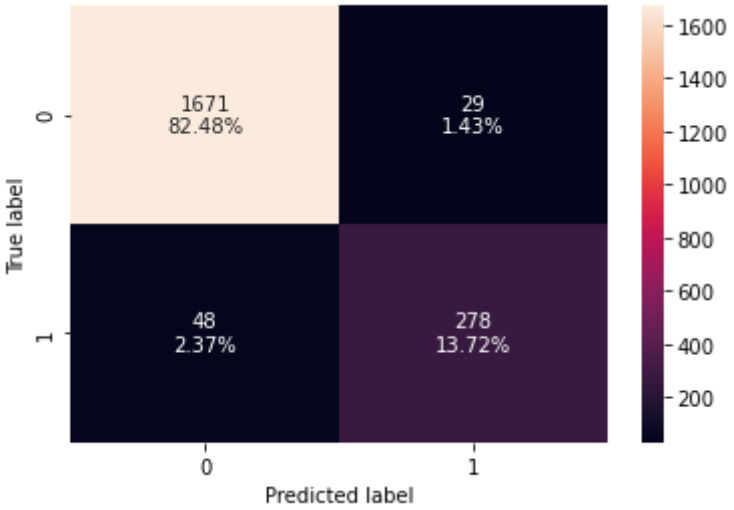
In [109…
```
ab_classifier_model_val_perf = model_performance_classification_sklearn(
    ab_classifier, X_val, y_val
)
print("Validation performance:")
ab_classifier_model_val_perf
```

Validation performance:

Out[109…

	Accuracy	Recall	Precision	F1
0	0.962	0.853	0.906	0.878

In [110…
```
# creating confusion matrix
confusion_matrix_sklearn(ab_classifier, X_val, y_val)
```



- The model is generalizing well, but I would like a higher recall score.

AdaBoost on over-sampled train data

In [111…
```
# AdaBoost on over-sampled train data
ab_over = AdaBoostClassifier(random_state=1)
ab_over.fit(X_train_over, y_train_over)
```

Out[111…
```
AdaBoostClassifier(random_state=1)
```

In [112…
```
# Calculating different metrics
ab_over_model_over_train_perf = model_performance_classification_sklearn(
    ab_over, X_train_over, y_train_over
)
print("Training performance:")
ab_over_model_over_train_perf
```

Out[112…

Training performance:

	Accuracy	Recall	Precision	F1
0	0.963	0.968	0.959	0.963

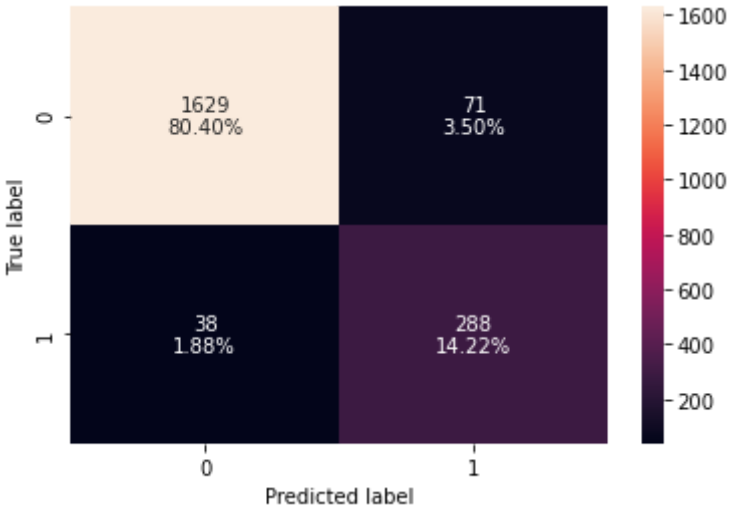
In [113…
```
ab_over_model_val_perf = model_performance_classification_sklearn(ab_over, X_val
print("Validation performance:")
ab_over_model_val_perf
```

Out[113…

Validation performance:

	Accuracy	Recall	Precision	F1
0	0.946	0.883	0.802	0.841

In [114…
```
# creating confusion matrix
confusion_matrix_sklearn(ab_over, X_val, y_val)
```



- Recall has improved slightly, but now the model is overfitting.

AdaBoost on under-sampled train data

In [115…
```
# AdaBoost on under-sampled train data
ab_under = AdaBoostClassifier(random_state=1)
ab_under.fit(X_train_un, y_train_un)
```

Out[115… `AdaBoostClassifier(random_state=1)`

In [116…
```
# Calculating different metrics
ab_under_model_under_train_perf = model_performance_classification_sklearn(
    ab_under, X_train_un, y_train_un
)
print("Training performance:")
ab_under_model_under_train_perf
```

Training performance:

Out[116…

	Accuracy	Recall	Precision	F1
0	0.949	0.955	0.944	0.950

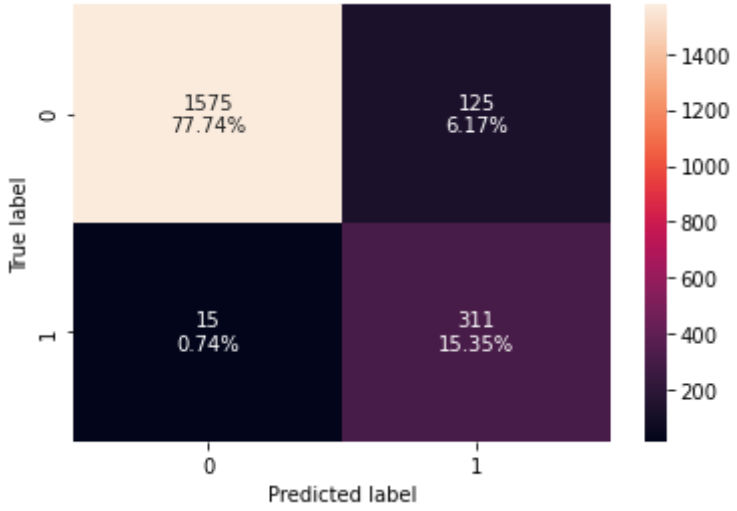
In [117…
```
ab_under_model_val_perf = model_performance_classification_sklearn(
    ab_under, X_val, y_val
)
print("Validation performance:")
ab_under_model_val_perf
```

Validation performance:

Out[117…

	Accuracy	Recall	Precision	F1
0	0.931	0.954	0.713	0.816

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



- The model is more what I'm looking for, as the recall is high and is generalizing.

XGBoost

In [119…
```
# Fitting the model
xgb_classifier = XGBClassifier(random_state=1, eval_metric="logloss")
xgb_classifier.fit(X_train, y_train)
```

Out[119…
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=12,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [120…
```
# Calculating different metrics
xgb_classifier_model_train_perf = model_performance_classification_sklearn(
    xgb_classifier, X_train, y_train
)
print("Training performance:")
xgb_classifier_model_train_perf
```

```
Training performance:
```

Out[120…

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

In [121…
```
xgb_classifier_model_val_perf = model_performance_classification_sklearn(
    xgb_classifier, X_val, y_val
)
print("Validation performance:")
xgb_classifier_model_val_perf
```

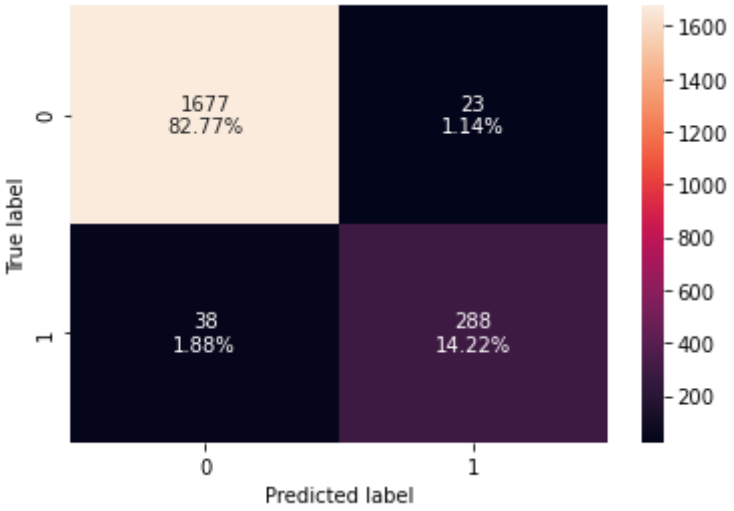
```
Validation performance:
```

Out[121…

	Accuracy	Recall	Precision	F1
0	0.970	0.883	0.926	0.904

In [122…

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



- The model is overfitting.

XGBoost on over-sampled train data

In [123…

```
# XGBoost on over-sampled train data
xgb_over = XGBClassifier(random_state=1, eval_metric="logloss")
xgb_over.fit(X_train_over, y_train_over)
```

Out[123…

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

In [124…

```
# Calculating different metrics
xgb_over_model_over_train_perf = model_performance_classification_sklearn(
    xgb_over, X_train_over, y_train_over
)
print("Training performance:")
xgb_over_model_over_train_perf
```

```
Training performance:
```

Out[124…

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

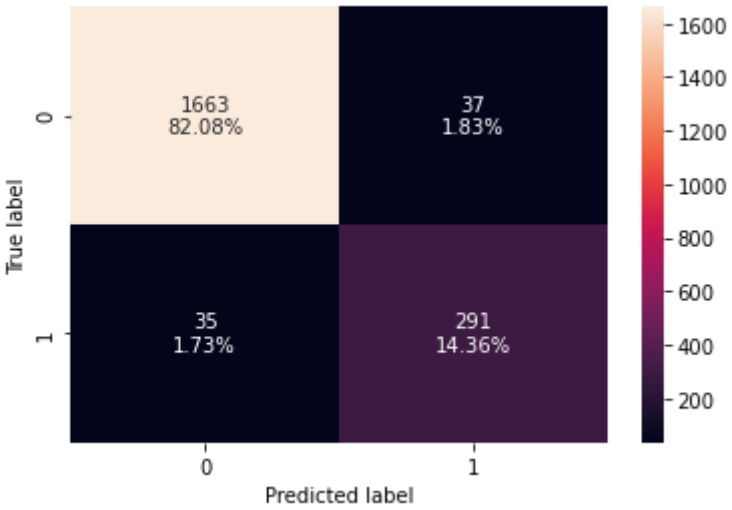
In [125…
```
xgb_over_model_val_perf = model_performance_classification_sklearn(
    xgb_over, X_val, y_val
)
print("Validation performance:")
xgb_over_model_val_perf
```

```
Validation performance:
```

Out[125…

	Accuracy	Recall	Precision	F1
0	0.964	0.893	0.887	0.890

In [126…
```
# creating confusion matrix
confusion_matrix_sklearn(xgb_over, X_val, y_val)
```



- The model is overfitting.

XGBoost on under-sampled train data

In [127…
```
# XGBoost on under-sampled train data
xgb_under = XGBClassifier(random_state=1, eval_metric="logloss")
xgb_under.fit(X_train_un, y_train_un)
```

Out[127…
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=12,
              num_parallel_tree=1, random_state=1, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [128…
```
# Calculating different metrics
xgb_under_model_under_train_perf = model_performance_classification_sklearn(
    xgb_under, X_train_un, y_train_un
)
print("Training performance:")
xgb_under_model_under_train_perf
```

Training performance:

Out[128…

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

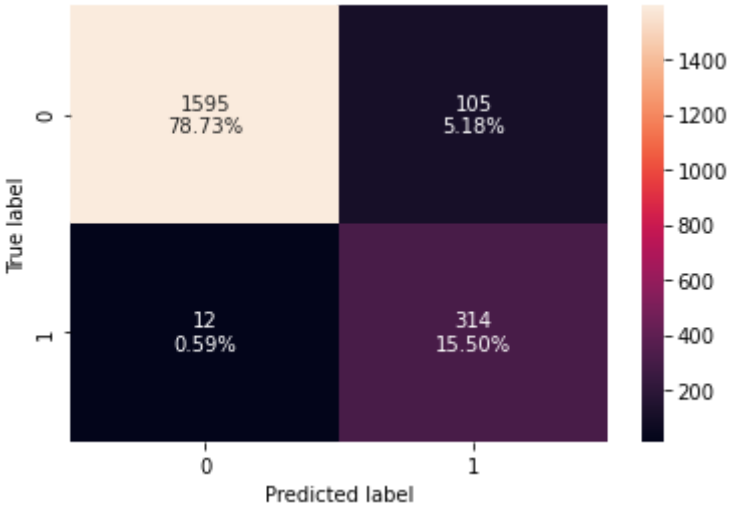
In [129…
```
xgb_under_model_val_perf = model_performance_classification_sklearn(
    xgb_under, X_val, y_val
)
print("Validation performance:")
xgb_under_model_val_perf
```

Validation performance:

Out[129…

	Accuracy	Recall	Precision	F1
0	0.942	0.963	0.749	0.843

In [130…
```
# creating confusion matrix
confusion_matrix_sklearn(xgb_under, X_val, y_val)
```



- The model is overfitting.

Decision Tree

In [131…
```
# Fitting the model
dtree = DecisionTreeClassifier(random_state=1)
dtree.fit(X_train, y_train)
```

Out[131… `DecisionTreeClassifier(random_state=1)`

In [132…
```
# Calculating different metrics
dtree_model_train_perf = model_performance_classification_sklearn(
    dtree, X_train, y_train
)
print("Training performance:")
dtree_model_train_perf
```

Training performance:

Out[132…

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

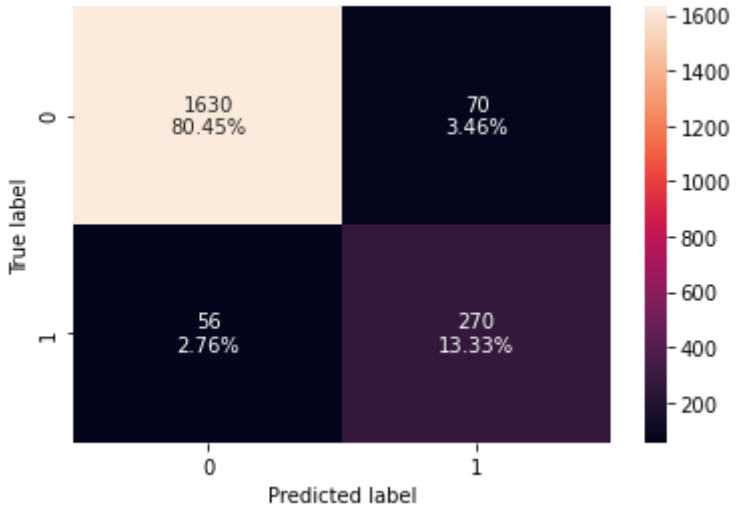
In [133…
```
dtree_model_val_perf = model_performance_classification_sklearn(dtree, X_val, y_
print("Validation performance:")
dtree_model_val_perf
```

```
Validation performance:
```

Out[133…

	Accuracy	Recall	Precision	F1
0	0.938	0.828	0.794	0.811

In [134…
```
# creating confusion matrix
confusion_matrix_sklearn(dtree, X_val, y_val)
```



- The model is overfitting.

Decision Tree on over-sampled train data

In [135…
```
# Decision Tree on over-sampled train data
dtree_over = DecisionTreeClassifier(random_state=1)
dtree_over.fit(X_train_over, y_train_over)
```

Out[135… `DecisionTreeClassifier(random_state=1)`

In [136…
```
# Calculating different metrics
dtree_over_model_over_train_perf = model_performance_classification_sklearn(
    dtree_over, X_train_over, y_train_over
)
print("Training performance:")
dtree_over_model_over_train_perf
```

```
Training performance:
```

Out[136…

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

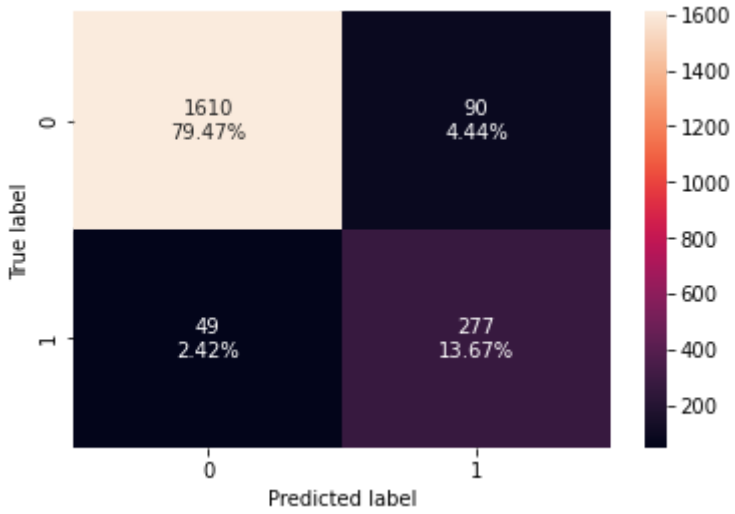
In [137…
```
dtree_over_model_val_perf = model_performance_classification_sklearn(
    dtree_over, X_val, y_val
)
print("Validation performance:")
dtree_over_model_val_perf
```

```
Validation performance:
```

Out[137…

	Accuracy	Recall	Precision	F1
0	0.931	0.850	0.755	0.799

In [138…
```
# creating confusion matrix
confusion_matrix_sklearn(dtree_over, X_val, y_val)
```



- The model is overfitting.

Decision Tree on under-sampled train data

In [139…
```
# Decision Tree on under-sampled train data
dtree_under = DecisionTreeClassifier(random_state=1)
dtree_under.fit(X_train_un, y_train_un)
```

Out[139… `DecisionTreeClassifier(random_state=1)`

In [140…
```
# Calculating different metrics
dtree_under_model_under_train_perf = model_performance_classification_sklearn(
    dtree_under, X_train_un, y_train_un
)
print("Training performance:")
dtree_under_model_under_train_perf
```

```
Training performance:
```

Out[140…

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

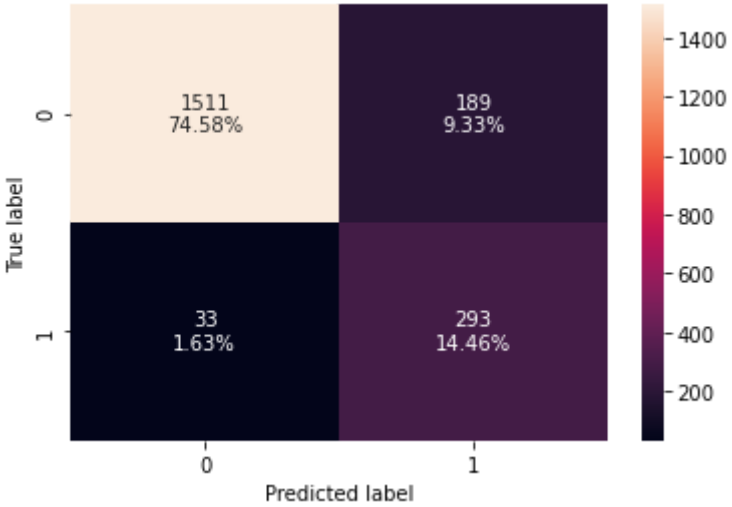
In [141…
```
dtree_under_model_val_perf = model_performance_classification_sklearn(
    dtree_under, X_val, y_val
)
print("Validation performance:")
dtree_under_model_val_perf
```

```
Validation performance:
```

Out[141…

	Accuracy	Recall	Precision	F1
0	0.890	0.899	0.608	0.725

In [142…
```
# creating confusion matrix
confusion_matrix_sklearn(dtree_under, X_val, y_val)
```



- The model is overfitting.

In [143…
```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        bagging_classifier_model_train_perf.T,
        bagging_over_model_over_train_perf.T,
        bagging_under_model_under_train_perf.T,
        rf_estimator_model_train_perf.T,
        rf_over_model_over_train_perf.T,
        rf_under_model_under_train_perf.T,
        gb_classifier_model_train_perf.T,
        gb_over_model_over_train_perf.T,
        gb_under_model_under_train_perf.T,
        ab_classifier_model_train_perf.T,
        ab_over_model_over_train_perf.T,
        ab_under_model_under_train_perf.T,
        xgb_classifier_model_train_perf.T,
        xgb_over_model_over_train_perf.T,
        xgb_under_model_under_train_perf.T,
        dtree_model_train_perf.T,
        dtree_over_model_over_train_perf.T,
        dtree_under_model_under_train_perf.T,
```

```
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Bagging",
    "Bagging Over",
    "Bagging Under",
    "Random Forest",
    "Random Forest Over",
    "Random Forest Under",
    "Gradient Boost",
    "Gradient Boost Over",
    "Gradient Boost Under",
    "AdaBoost",
    "AdaBoost Over",
    "AdaBoost Under",
    "XGBoost",
    "XGBoost Over",
    "XGBoost Under",
    "Decision Tree",
    "Decision Tree Over",
    "Decision Tree Under",
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[143…

	Bagging	Bagging Over	Bagging Under	Random Forest	Random Forest Over	Random Forest Under	Gradient Boost	Gradient Boost Over	Gradient Boost Under
Accuracy	0.997	0.998	0.996	1.000	1.000	1.000	0.973	0.978	0.973
Recall	0.986	0.997	0.993	1.000	1.000	1.000	0.875	0.981	0.980
Precision	0.996	0.999	0.999	1.000	1.000	1.000	0.950	0.975	0.967
F1	0.991	0.998	0.996	1.000	1.000	1.000	0.911	0.978	0.973

In [144…

```
# Validation performance comparison

models_val_comp_df = pd.concat(
    [
        bagging_classifier_model_val_perf.T,
        bagging_over_model_val_perf.T,
        bagging_under_model_val_perf.T,
        rf_estimator_model_val_perf.T,
        rf_over_model_val_perf.T,
        rf_under_model_val_perf.T,
        gb_classifier_model_val_perf.T,
        gb_over_model_val_perf.T,
        gb_under_model_val_perf.T,
        ab_classifier_model_val_perf.T,
        ab_over_model_val_perf.T,
        ab_under_model_val_perf.T,
        xgb_classifier_model_val_perf.T,
        xgb_over_model_val_perf.T,
        xgb_under_model_val_perf.T,
        dtree_model_val_perf.T,
        dtree_over_model_val_perf.T,
```

```
        dtree_under_model_val_perf.T,
    ],
    axis=1,
)
models_val_comp_df.columns = [
    "Bagging",
    "Bagging Over",
    "Bagging Under",
    "Random Forest",
    "Random Forest Over",
    "Random Forest Under",
    "Gradient Boost",
    "Gradient Boost Over",
    "Gradient Boost Under",
    "AdaBoost",
    "AdaBoost Over",
    "AdaBoost Under",
    "XGBoost",
    "XGBoost Over",
    "XGBoost Under",
    "Decision Tree",
    "Decision Tree Over",
    "Decision Tree Under",
]
print("Validation performance comparison:")
models_val_comp_df
```

```
Validation performance comparison:
```

Out[144…

	Bagging	Bagging Over	Bagging Under	Random Forest	Random Forest Over	Random Forest Under	Gradient Boost	Gradient Boost Over	Gradient Boost Under
Accuracy	0.954	0.950	0.923	0.962	0.955	0.934	0.968	0.958	0.934
Recall	0.807	0.883	0.929	0.819	0.862	0.933	0.862	0.899	0.963
Precision	0.895	0.818	0.695	0.934	0.859	0.733	0.934	0.847	0.722
F1	0.848	0.850	0.795	0.873	0.861	0.821	0.896	0.872	0.825

I like Gradient Boost Under, AdaBoost Under, and Bagging Under as the recall scores are high and are comparable between train and validation sets.

Tuning GBoost Under, AdaBoost Under, and Bagging Under with RandomizedSearchCV

Tuned Bagging Under

In [145…

```
bagging_params = BaggingClassifier(random_state=1)
bagging_params.get_params()
```

Out[145…

```
{'base_estimator': None,
 'bootstrap': True,
 'bootstrap_features': False,
 'max_features': 1.0,
 'max_samples': 1.0,
 'n_estimators': 10,
 'n_jobs': None,
```

```
'oob_score': False,
'random_state': 1,
'verbose': 0,
'warm_start': False}
```

In [146…

```
%%time

# defining model
bagging_under_tuned = BaggingClassifier(random_state=1)

# Parameter grid to pass in RandomizedSearchCV
param_grid={'n_estimators':np.arange(10,150,10),
            'max_features': [0.7, 0.8, 0.9, 1],
            'max_samples': [0.7, 0.8, 0.9, 1],}

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

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(bagging_under_tuned, param_distributions=para

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

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

```
Best parameters are {'n_estimators': 50, 'max_samples': 1, 'max_features': 0.8}
with CV score=1.0:
Wall time: 11.6 s
```

In [147…

```
# building model with best parameters
bagging_under_tuned = BaggingClassifier(
    random_state=1,
    n_estimators=50,
    max_features=0.8,
    max_samples=1,
)
# Fit the model on training data
bagging_under_tuned.fit(X_train_un, y_train_un)
```

Out[147…

```
BaggingClassifier(max_features=0.8, max_samples=1, n_estimators=50,
                  random_state=1)
```

In [148…

```
# Calculating different metrics on train set
bagging_random_Under_train = model_performance_classification_sklearn(
    bagging_under_tuned, X_train_un, y_train_un
)
print("Training performance:")
bagging_random_Under_train
```

```
Training performance:
```

Out[148…

	Accuracy	Recall	Precision	F1
0	0.500	1.000	0.500	0.667

In [149…

```
# Calculating different metrics on validation set
bagging_random_val = model_performance_classification_sklearn(
```

```
    bagging_under_tuned, X_val, y_val
)
print("Validation performance:")
bagging_random_val
```

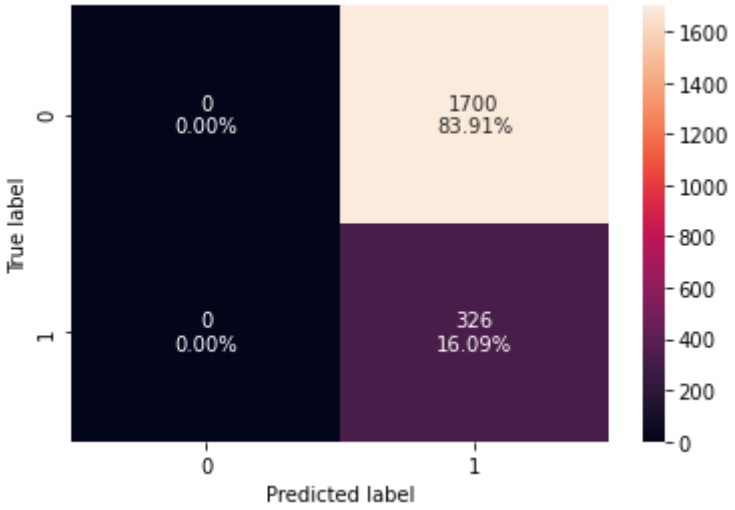
```
Validation performance:
```

Out[149…

	Accuracy	Recall	Precision	F1
0	0.161	1.000	0.161	0.277

In [150…

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



- The model's recall is perfect on both train and validation set, but the other metrics are really low.

Tuned Gradient Boosting Under

In [151…

```
gb_params = GradientBoostingClassifier(random_state=1)
gb_params.get_params()
```

Out[151…

```
{'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'deviance',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': 1,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
```

```
'verbose': 0,
'warm_start': False}
```

In [152…

```
%%time

# defining model
gb_under_tuned = GradientBoostingClassifier(random_state=1)

# Parameter grid to pass in RandomizedSearchCV
param_grid={'n_estimators':np.arange(10,150,10),
            'max_features': [0.7, 0.8, 0.9, 1],
            'subsample': [0.7, 0.8, 0.9, 1],
            'max_depth': [3, 4, 5, None],}

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

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(gb_under_tuned, param_distributions=param_gri

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

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

```
Best parameters are {'subsample': 0.7, 'n_estimators': 130, 'max_features': 0.8,
'max_depth': 5} with CV score=0.9600523286237573:
Wall time: 29.6 s
```

In [153…

```
# building model with best parameters
gb_under_tuned = GradientBoostingClassifier(
    random_state=1,
    n_estimators=130,
    max_features=0.8,
    max_depth=5,
    subsample=0.7,
)
# Fit the model on training data
gb_under_tuned.fit(X_train_un, y_train_un)
```

Out[153…

```
GradientBoostingClassifier(max_depth=5, max_features=0.8, n_estimators=130,
                           random_state=1, subsample=0.7)
```

In [154…

```
# Calculating different metrics on train set
gb_random_Under_train = model_performance_classification_sklearn(
    gb_under_tuned, X_train_un, y_train_un
)
print("Training performance:")
gb_random_Under_train
```

```
Training performance:
```

Out[154…

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

In [155…

```
# Calculating different metrics on validation set
gb_random_val = model_performance_classification_sklearn(gb_under_tuned, X_val,
```

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

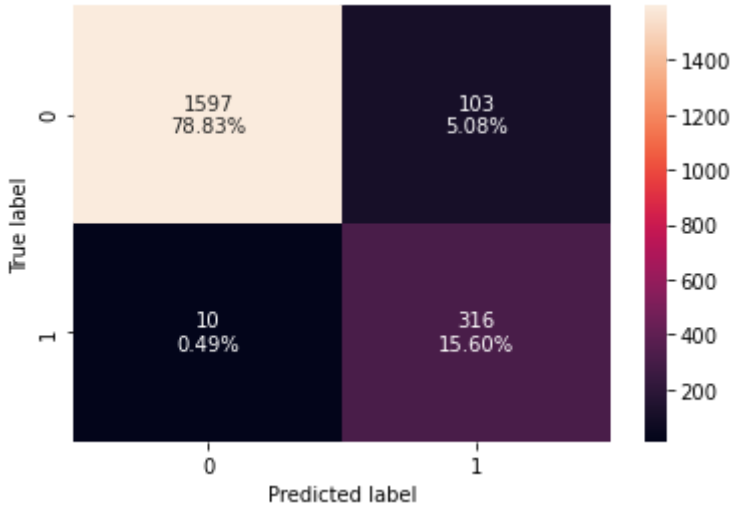
```
Validation performance:
```

Out[155…

	Accuracy	Recall	Precision	F1
0	0.944	0.969	0.754	0.848

In [156…

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



- The model is giving me the scores I want on the validation set, but the model is overfitted on the train set.

Tuned AdaBoost Under

In [157…

```
ab_params = AdaBoostClassifier(random_state=1)
ab_params.get_params()
```

Out[157…

```
{'algorithm': 'SAMME.R',
 'base_estimator': None,
 'learning_rate': 1.0,
 'n_estimators': 50,
 'random_state': 1}
```

In [158…

```
%%time

# defining model
ab_under_tuned = AdaBoostClassifier(random_state=1)

# Parameter grid to pass in RandomizedSearchCV
param_grid={"base_estimator": [
        DecisionTreeClassifier(max_depth=1),
        DecisionTreeClassifier(max_depth=2),
        DecisionTreeClassifier(max_depth=3),
        None,
    ],
    "n_estimators": np.arange(10, 150, 10),
    "learning_rate": np.arange(0.1, 2, 0.1),
```

```
}

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

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(ab_under_tuned, param_distributions=param_gri

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

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

```
Best parameters are {'n_estimators': 90, 'learning_rate': 0.8, 'base_estimator':
DecisionTreeClassifier(max_depth=3)} with CV score=0.95287284144427:
Wall time: 10.3 s
```

In [159…
```
# building model with best parameters
ab_under_tuned = AdaBoostClassifier(
    random_state=1,
    n_estimators=90,
    learning_rate=0.8,
    base_estimator=DecisionTreeClassifier(max_depth=3),
)
# Fit the model on training data
ab_under_tuned.fit(X_train_un, y_train_un)
```

Out[159…
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                   learning_rate=0.8, n_estimators=90, random_state=1)
```

In [160…
```
# Calculating different metrics on train set
ab_random_Under_train = model_performance_classification_sklearn(
    ab_under_tuned, X_train_un, y_train_un
)
print("Training performance:")
ab_random_Under_train
```

```
Training performance:
```

Out[160…

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

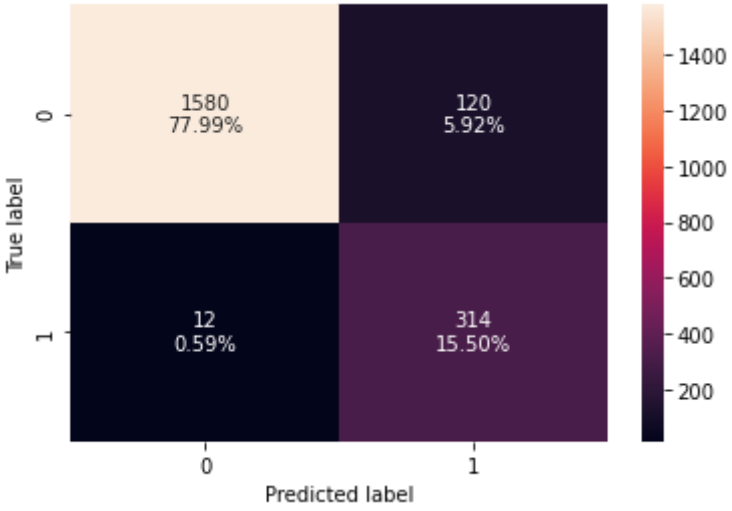
In [161…
```
# Calculating different metrics on validation set
ab_random_val = model_performance_classification_sklearn(ab_under_tuned, X_val,
print("Validation performance:")
ab_random_val
```

```
Validation performance:
```

Out[161…

	Accuracy	Recall	Precision	F1
0	0.935	0.963	0.724	0.826

In [162…
```
# creating confusion matrix
confusion_matrix_sklearn(ab_under_tuned, X_val, y_val)
```



- The model is giving me the scores I want on the validation set, but the model is overfitted on the train set.

In [163…
```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        bagging_random_Under_train.T,
        gb_random_Under_train.T,
        ab_random_Under_train.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Bagging Under Tuned",
    "Gradient Boost Under Tuned",
    "AdaBoost Under Tuned",
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[163…

	Bagging Under Tuned	Gradient Boost Under Tuned	AdaBoost Under Tuned
Accuracy	0.500	1.000	1.000
Recall	1.000	1.000	1.000
Precision	0.500	1.000	1.000
F1	0.667	1.000	1.000

In [164…
```
# Validation performance comparison

models_val_comp_df = pd.concat(
    [
        bagging_random_val.T,
        gb_random_val.T,
        ab_random_val.T,
    ],
```

```
    axis=1,
)
models_val_comp_df.columns = [
    "Bagging Under Tuned",
    "Gradient Boost Under Tuned",
    "AdaBoost Under Tuned",
]
print("Validation performance comparison:")
models_val_comp_df
```

```
Validation performance comparison:
```

Out[164…

	Bagging Under Tuned	Gradient Boost Under Tuned	AdaBoost Under Tuned
Accuracy	0.161	0.944	0.935
Recall	1.000	0.969	0.963
Precision	0.161	0.754	0.724
F1	0.277	0.848	0.826

The validation scores for Gradient Boost and AdaBoost have improved sligtly, but the tuned models are now overfitting the train data.

Performance on the test set

In [165…

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

```
Test performance:
```

Out[165…

	Accuracy	Recall	Precision	F1
0	0.919	0.942	0.678	0.789

In [166…

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

```
Test performance:
```

Out[166…

	Accuracy	Recall	Precision	F1
0	0.931	0.969	0.709	0.819

In [167…

```
# Calculating different metrics on the test set
ab_under_model_test_perf = model_performance_classification_sklearn(
    ab_under, X_test, y_test
)
```

```
print("Test performance:")
ab_under_model_test_perf
```

```
Test performance:
```

Out[167…

	Accuracy	Recall	Precision	F1
0	0.929	0.954	0.706	0.812

In [168…

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

```
Test performance:
```

Out[168…

	Accuracy	Recall	Precision	F1
0	0.160	1.000	0.160	0.276

In [169…

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

```
Test performance:
```

Out[169…

	Accuracy	Recall	Precision	F1
0	0.941	0.969	0.741	0.840

In [170…

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

```
Test performance:
```

Out[170…

	Accuracy	Recall	Precision	F1
0	0.940	0.963	0.742	0.838

In [171…

```
# Test performance comparison

models_test_comp_df = pd.concat(
    [
        bagging_under_model_test_perf.T,
        gb_under_model_test_perf.T,
        ab_under_model_test_perf.T,
        bagging_random_test_perf.T,
        gb_random_test_perf.T,
        ab_random_test_perf.T,
```

```
    ],
    axis=1,
)
models_test_comp_df.columns = [
    "Bagging Under",
    "Gradient Boost Under",
    "AdaBoost Under",
    "Bagging Under Tuned",
    "Gradient Boost Under Tuned",
    "AdaBoost Under Tuned",
]
print("Test performance comparison:")
models_test_comp_df
```

```
Test performance comparison:
```

Out[171…

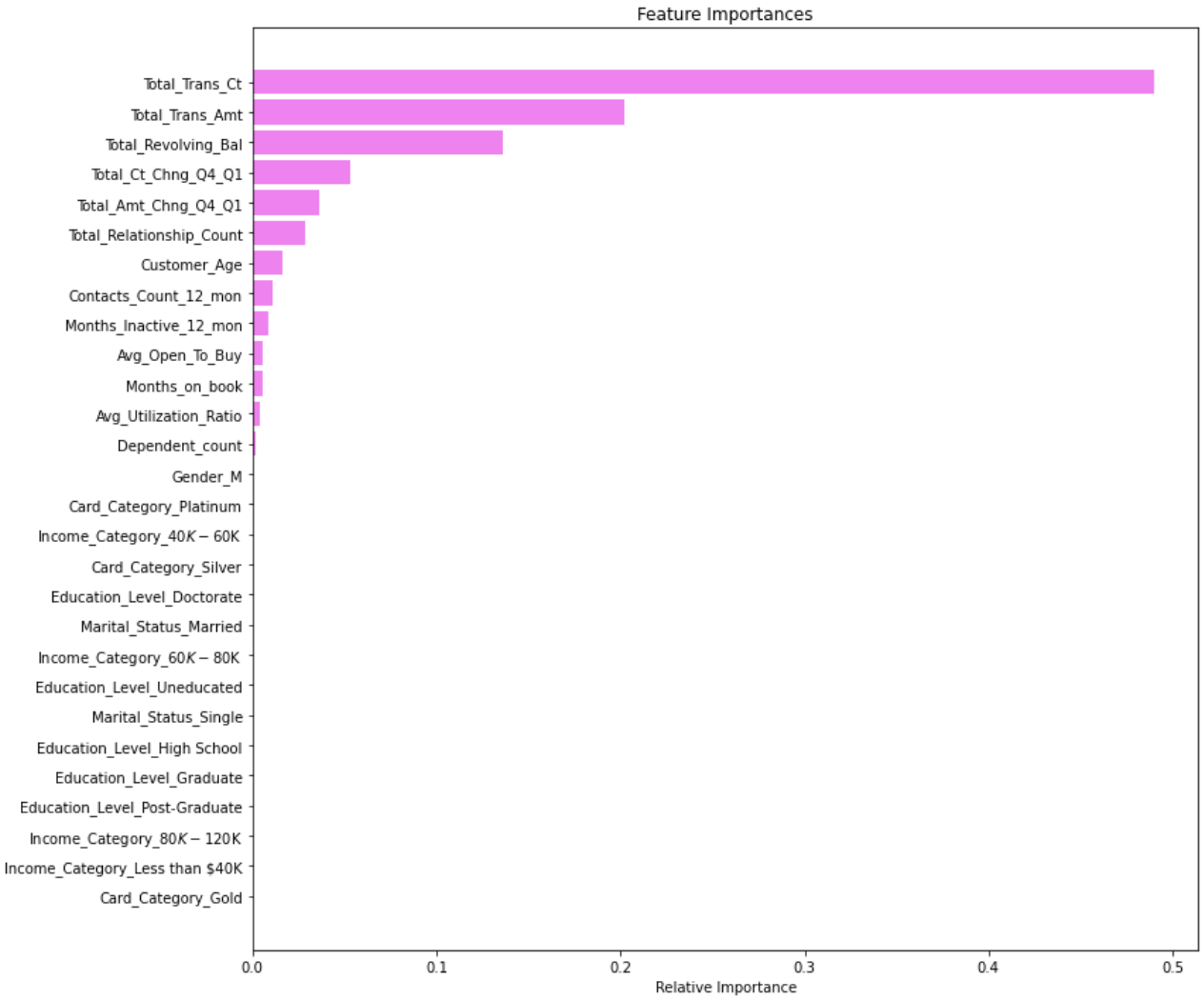
	Bagging Under	Gradient Boost Under	AdaBoost Under	Bagging Under Tuned	Gradient Boost Under Tuned	AdaBoost Under Tuned
Accuracy	0.919	0.931	0.929	0.160	0.941	0.940
Recall	0.942	0.969	0.954	1.000	0.969	0.963
Precision	0.678	0.709	0.706	0.160	0.741	0.742
F1	0.789	0.819	0.812	0.276	0.840	0.838

I will productionize the Gradient Boost model trained on the under-sampled data.

- The model generalizes well on train, validation, and test sets.

In [172…

```
feature_names = X_train_un.columns
importances = gb_under.feature_importances_
indices = np.argsort(importances)

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



- The three most important features are 'Total_Trans_Ct', 'Total_Trans_Amt', and 'Total_Revolving_Bal'.

Pipelines for productionizing the model

In [173…
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Attrition_Flag            10127 non-null  int64
 1   Customer_Age              10127 non-null  int64
 2   Gender                    10127 non-null  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           9015 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  Total_Revolving_Bal       10127 non-null  int64
 13  Avg_Open_To_Buy           10127 non-null  float64
 14  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 15  Total_Trans_Amt           10127 non-null  int64
 16  Total_Trans_Ct            10127 non-null  int64
 17  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 18  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(4), int64(10), object(5)
memory usage: 1.5+ MB
```

In [174…

```
# creating a list of numerical variables
numerical_features = [
    "Customer_Age",
    "Dependent_count",
    "Months_on_book",
    "Total_Relationship_Count",
    "Months_Inactive_12_mon",
    "Contacts_Count_12_mon",
    "Total_Revolving_Bal",
    "Avg_Open_To_Buy",
    "Total_Amt_Chng_Q4_Q1",
    "Total_Trans_Amt",
    "Total_Trans_Ct",
    "Total_Ct_Chng_Q4_Q1",
    "Avg_Utilization_Ratio",
]

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


# creating a list of categorical variables
categorical_features = [
    "Gender",
    "Education_Level",
    "Marital_Status",
    "Income_Category",
    "Card_Category",
]

# creating a transformer for categorical variables, which will first apply simpl
# 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

# combining categorical transformer and numerical transformer using a column tra

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numerical_features),
        ("cat", categorical_transformer, categorical_features),
    ],
    remainder="passthrough",
)
```

```
# remainder = "passthrough" has been used, it will allow variables that are pres
# but not in "numerical_columns" and "categorical_columns" to pass through the c
```

In [175…
```
# Separating target variable and other variables
X = df1.drop("Attrition_Flag", axis=1)
y = df1["Attrition_Flag"]
```

In [176…
```
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=1, stratify=y
)
print(X_train.shape, X_test.shape)
```

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

In [177…
```
# Under-sampling train set
rus = RandomUnderSampler(random_state=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

In [178…
```
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 ==

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

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

```
Before Under Sampling, counts of label 'Yes': 1139
Before Under Sampling, counts of label 'No': 5949

After Under Sampling, counts of label 'Yes': 1139
After Under Sampling, counts of label 'No': 1139

After Under Sampling, the shape of train_X: (2278, 18)
After Under Sampling, the shape of train_y: (2278,)
```

In [179…
```
# Creating new pipeline with best parameters
model = Pipeline(
    steps=[
        ("pre", preprocessor),
        (
            "GB",
            GradientBoostingClassifier(random_state=1),
        ),
    ]
)
# Fit the model on training data
model.fit(X_train_un, y_train_un)
```

Out[179…
```
Pipeline(steps=[('pre',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('num',
                                                  Pipeline(steps=[('imputer',
                                                                   SimpleImputer
```

```
(strategy='median'))]),
                                                 ['Customer_Age',
                                                  'Dependent_count',
                                                  'Months_on_book',
                                                  'Total_Relationship_Count',
                                                  'Months_Inactive_12_mon',
                                                  'Contacts_Count_12_mon',
                                                  'Total_Revolving_Bal',
                                                  'Avg_Open_To_Buy',
                                                  'Total_Amt_Chng_Q4_Q1',
                                                  'Total_Trans_Amt',
                                                  'Total_Trans_Ct',
                                                  'Total_Ct_Chng_Q4_Q1',
                                                  'Avg_Utilization_Ratio']),
                                                ('cat',
                                                 Pipeline(steps=[('imputer',
                                                                  SimpleImputer
(strategy='most_frequent')),
                                                                 ('onehot',
                                                                  OneHotEncoder
(handle_unknown='ignore'))]),
                                                 ['Gender', 'Education_Level',
                                                  'Marital_Status',
                                                  'Income_Category',
                                                  'Card_Category'])])),
                ('GB', GradientBoostingClassifier(random_state=1))])
```

In [180…

```
# Calculating different metrics
model_under_train_perf = model_performance_classification_sklearn(
    model, X_train_un, y_train_un
)
print("Training performance:")
model_under_train_perf
```

```
Training performance:
```

Out[180…

	Accuracy	Recall	Precision	F1
0	0.974	0.975	0.973	0.974

In [181…

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

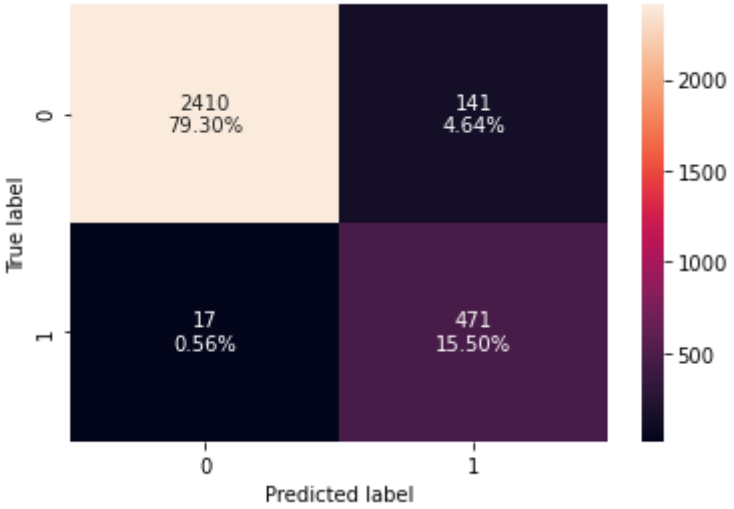
```
Test performance:
```

Out[181…

	Accuracy	Recall	Precision	F1
0	0.948	0.965	0.770	0.856

In [182…

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

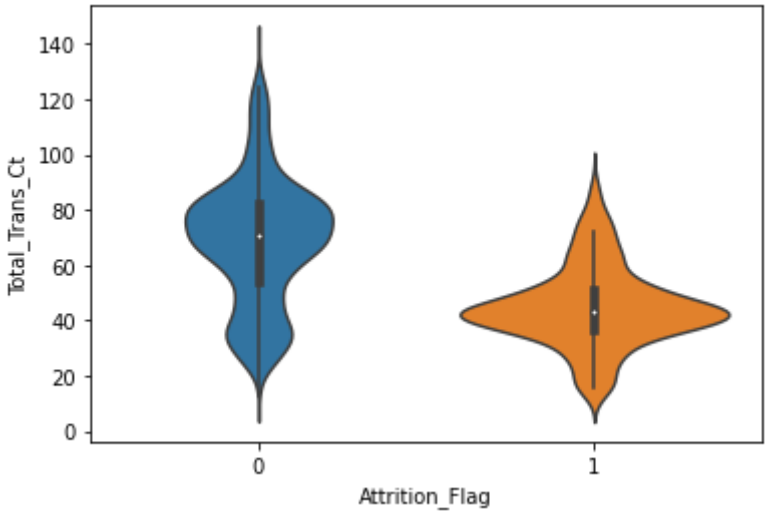


- I am getting a recall >0.95 and precision/accuracy >0.70

Actionable Insights & Recommendations

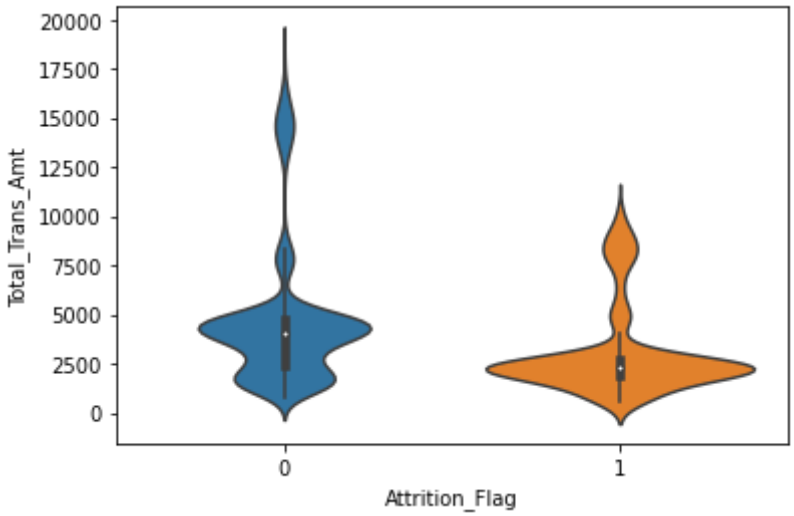
In [183…
```
sns.violinplot(df1["Attrition_Flag"], df1["Total_Trans_Ct"])
```

Out[183…
```
<AxesSubplot:xlabel='Attrition_Flag', ylabel='Total_Trans_Ct'>
```



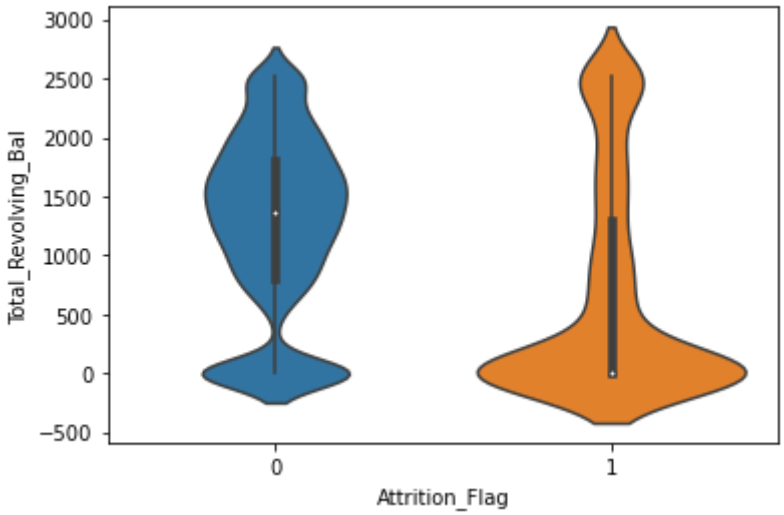
In [184…
```
sns.violinplot(df1["Attrition_Flag"], df1["Total_Trans_Amt"])
```

Out[184…
```
<AxesSubplot:xlabel='Attrition_Flag', ylabel='Total_Trans_Amt'>
```



In [185…
```
sns.violinplot(df1["Attrition_Flag"], df1["Total_Revolving_Bal"])
```

Out[185…
```
<AxesSubplot:xlabel='Attrition_Flag', ylabel='Total_Revolving_Bal'>
```



Once the total transaction count reaches ~75 within 12 months, the chance of the customer leaving the bank decreases.

- Maybe introduce a system that incentivizes customers to make more transactions.

Customers whose total transaction amount within a year, reaching ~4,000USD, are likely to still be customers with the bank.

The higher the customer's balance that carries over from month to month, the likelyhood of the customer staying with the bank increases.