ChurnPrediction

March 28, 2023

[5]:
```
%%latex
\tableofcontents
```

Contents

1 Introduction **2**

2 First Step: Data cleaning, analysis and visualization **2**

3 Data Analysis **6**

4 Data Visualization **8**

5 Second Step: Feature Importance or Selection **25**

6 Removing Colinear Features **30**

7 Balancing the Dataset **30**
7.0.1 Shuffling the training data 31

8 Using ExtraTreesClassifier to select the features **33**
8.1 Selecting the features for model building 34

9 Third Step: Machine Learning Models **36**

10 Hyperparameter Tuning for the models **36**
10.1 Classification of Churn rate 37
10.1.1 Classifiers 37

11 Adaboost model **38**

12 XGboost **39**

13 Undersampling **41**

14 weighted Average **43**

15 Conclusions **44**

1 Introduction

Kaggle data set

Business Problem Business case A business manager of a consumer credit card bank is facing the problem of customer attrition. They want to analyze the data to find out the reason behind this and leverage the same to predict customers who are more likely to churn.

Columns Description:

1. Attrition_Flag: Internal event (customer activity) variable - if the account is closed then "Attrited Customer" else "Existing Customer"
2. Customer_Age: Customers age
3. Gender: Gender of the account holder
4. Dependent_count: Number of dependents
5. Education_Level: Educational Qualification of the account holder - Graduate, High School, Unknown, Uneducated, College(refers to a college student), Post-Graduate, Doctorate
6. Marital_Status: Marital Status of the account holder
7. Income_Category: Annual Income Category of the account holder
8. Card_Category: Type of Card
9. Months_on_book: Period of relationship with the bank
10. Total_Relationship_Count: Total no. of products held by the customer
11. Months_Inactive_12_mon: No. of months inactive in the last 12 months
12. Contacts_Count_12_mon: No. of Contacts between the customer and bank in the last 12 months
13. Credit_Limit: Credit Limit on the Credit Card
14. Total_Revolving_Bal: The balance that carries over from one month to the next is the revolving balance
15. Avg_Open_To_Buy: Open to Buy refers to the amount left on the credit card to use (Average of last 12 months)
16. Total_Trans_Amt: Total Transaction Amount (Last 12 months)
17. Total_Trans_Ct: Total Transaction Count (Last 12 months)
18. Total_Ct_Chng_Q4_Q1: Ratio of the total transaction count in 4th quarter and the total transaction count in 1st quarter
19. Total_Amt_Chng_Q4_Q1: Ratio of the total transaction amount in 4th quarter and the total transaction amount in 1st quarter
20. Avg_Utilization_Ratio: Represents how much of the available credit the customer spent

About this file: This is a dataset aimed for data science use case project, specifically for classification model. Note: Please ignore the last 2 columns (Naive Bayes Classification), we suggest better delete it before doing anything.

2 First Step: Data cleaning, analysis and visualization

Packages and versions used for the project

[1]:
```
import sys
import sklearn
import xgboost
```

```
print("Python Version", sys.version)
print("Sklearn version",sklearn.__version__)
print("XGBoost version:", xgboost.__version__)
```

```
Python Version 3.9.12 (main, Apr  5 2022, 01:53:17)
[Clang 12.0.0 ]
Sklearn version 1.0.2
XGBoost version: 1.7.4
```

Importing the necessary libraries for data preprocessing

[2]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy
from scipy.stats import normaltest
from scipy.stats import anderson
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import␣
↪ExtraTreesClassifier,AdaBoostClassifier,RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
import scipy.stats as stats
from collections import Counter
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import OneHotEncoder,StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,roc_auc_score,classification_report
from sklearn.metrics import␣
↪ConfusionMatrixDisplay,confusion_matrix,roc_curve,auc,plot_confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
from imblearn.under_sampling import RandomUnderSampler

# displays the maximun columns in the dataframe
pd.set_option('display.max_columns', 40)

# ignoring runtime warnings
import warnings
warnings.filterwarnings('ignore')
```

Loading the data into dataframe

[3]:
```
data = pd.read_csv('credit_card_churn.csv')
```

Shape of the data

[4]:
```
data.shape
```

[4]:
```
(10127, 23)
```

[5]:
```
data.head()
```

[5]:
```
    CLIENTNUM     Attrition_Flag  Customer_Age Gender  Dependent_count  \
0  768805383  Existing Customer            45      M                3
1  818770008  Existing Customer            49      F                5
2  713982108  Existing Customer            51      M                3
3  769911858  Existing Customer            40      F                4
4  709106358  Existing Customer            40      M                3

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

   Months_on_book  Total_Relationship_Count  Months_Inactive_12_mon  \
0              39                         5                       1
1              44                         6                       1
2              36                         4                       1
3              34                         3                       4
4              21                         5                       1

   Contacts_Count_12_mon  Credit_Limit  Total_Revolving_Bal  Avg_Open_To_Buy  \
0                      3       12691.0                  777          11914.0
1                      2        8256.0                  864           7392.0
2                      0        3418.0                    0           3418.0
3                      1        3313.0                 2517            796.0
4                      0        4716.0                    0           4716.0

   Total_Amt_Chng_Q4_Q1  Total_Trans_Amt  Total_Trans_Ct  Total_Ct_Chng_Q4_Q1  \
0                 1.335             1144              42                1.625
1                 1.541             1291              33                3.714
2                 2.594             1887              20                2.333
3                 1.405             1171              20                2.333
4                 2.175              816              28                2.500

   Avg_Utilization_Ratio  \
0                  0.061
```

```
1                   0.105
2                   0.000
3                   0.760
4                   0.000

   Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dep
endent_count_Education_Level_Months_Inactive_12_mon_1  \
0                                            0.000093
1                                            0.000057
2                                            0.000021
3                                            0.000134
4                                            0.000022

   Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dep
endent_count_Education_Level_Months_Inactive_12_mon_2
0                                             0.99991
1                                             0.99994
2                                             0.99998
3                                             0.99987
4                                             0.99998
```

Dropping the first and last two columns it is not useful for the data analysis

[6]:
```
df = data.
 ↪drop(['CLIENTNUM','Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mo
```

[7]:
```
df.columns
```

[7]:
```
Index(['Attrition_Flag', 'Customer_Age', 'Gender', 'Dependent_count',
       'Education_Level', 'Marital_Status', 'Income_Category', 'Card_Category',
       'Months_on_book', 'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit', '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'],
      dtype='object')
```

Dataframe columns

1. Customer_Age,Credit_Limit,Total_Revolving_Bal,Avg_Open_To_Buy,Total_Amt_Chng_Q4_Q1,Total Total_Ct_Chng_Q4_Q1,Avg_Utilization_Ratio, are all continous varaibles, rest are categorical variables
2. Atttrition_flag is the target variable

Checking if null values exist in the dataframe

[8]:
```
df.isnull().sum()
```

[8]:
```
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
```

There are no missing values in the dataframe

3 Data Analysis

[9]:
```
df.describe()
```

[9]:
```
       Customer_Age  Dependent_count  Months_on_book  \
count  10127.000000     10127.000000    10127.000000
mean      46.325960         2.346203       35.928409
std        8.016814         1.298908        7.986416
min       26.000000         0.000000       13.000000
25%       41.000000         1.000000       31.000000
50%       46.000000         2.000000       36.000000
75%       52.000000         3.000000       40.000000
max       73.000000         5.000000       56.000000

       Total_Relationship_Count  Months_Inactive_12_mon  \
count              10127.000000            10127.000000
mean                   3.812580                2.341167
std                    1.554408                1.010622
min                    1.000000                0.000000
25%                    3.000000                2.000000
50%                    4.000000                2.000000
75%                    5.000000                3.000000
max                    6.000000                6.000000
```

```
       Contacts_Count_12_mon  Credit_Limit  Total_Revolving_Bal  \
count           10127.000000  10127.000000         10127.000000
mean                2.455317   8631.953698          1162.814061
std                 1.106225   9088.776650           814.987335
min                 0.000000   1438.300000             0.000000
25%                 2.000000   2555.000000           359.000000
50%                 2.000000   4549.000000          1276.000000
75%                 3.000000  11067.500000          1784.000000
max                 6.000000  34516.000000          2517.000000

       Avg_Open_To_Buy  Total_Amt_Chng_Q4_Q1  Total_Trans_Amt  Total_Trans_Ct  \
count     10127.000000          10127.000000     10127.000000    10127.000000
mean       7469.139637              0.759941      4404.086304       64.858695
std        9090.685324              0.219207      3397.129254       23.472570
min           3.000000              0.000000       510.000000       10.000000
25%        1324.500000              0.631000      2155.500000       45.000000
50%        3474.000000              0.736000      3899.000000       67.000000
75%        9859.000000              0.859000      4741.000000       81.000000
max       34516.000000              3.397000     18484.000000      139.000000

       Total_Ct_Chng_Q4_Q1  Avg_Utilization_Ratio
count         10127.000000           10127.000000
mean              0.712222               0.274894
std               0.238086               0.275691
min               0.000000               0.000000
25%               0.582000               0.023000
50%               0.702000               0.176000
75%               0.818000               0.503000
max               3.714000               0.999000
```

There are outliers in the data max is far off from the mean and median

1. Total_Trans_Amt
2. Credit_limit
3. Avg_Open_To_Buy

[10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Attrition_Flag            10127 non-null  object
 1   Customer_Age              10127 non-null  int64
 2   Gender                    10127 non-null  object
 3   Dependent_count           10127 non-null  int64
 4   Education_Level           10127 non-null  object
```

```
 5   Marital_Status            10127 non-null  object
 6   Income_Category           10127 non-null  object
 7   Card_Category             10127 non-null  object
 8   Months_on_book            10127 non-null  int64
 9   Total_Relationship_Count  10127 non-null  int64
 10  Months_Inactive_12_mon    10127 non-null  int64
 11  Contacts_Count_12_mon     10127 non-null  int64
 12  Credit_Limit              10127 non-null  float64
 13  Total_Revolving_Bal       10127 non-null  int64
 14  Avg_Open_To_Buy           10127 non-null  float64
 15  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 16  Total_Trans_Amt           10127 non-null  int64
 17  Total_Trans_Ct            10127 non-null  int64
 18  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 19  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(5), int64(9), object(6)
memory usage: 1.5+ MB
```

Numerical columns in the given dataframe

[11]:
```
df_numeric=␣
 ↪df[['Customer_Age','Dependent_count','Months_on_book','Total_Relationship_Count',
'Months_Inactive_12_mon','Contacts_Count_12_mon','Credit_Limit','Total_Revolving_Bal',
'Avg_Open_To_Buy','Total_Amt_Chng_Q4_Q1','Total_Trans_Amt','Total_Trans_Ct','Total_Ct_Chng_Q4_
'Avg_Utilization_Ratio']]
```

4 Data Visualization

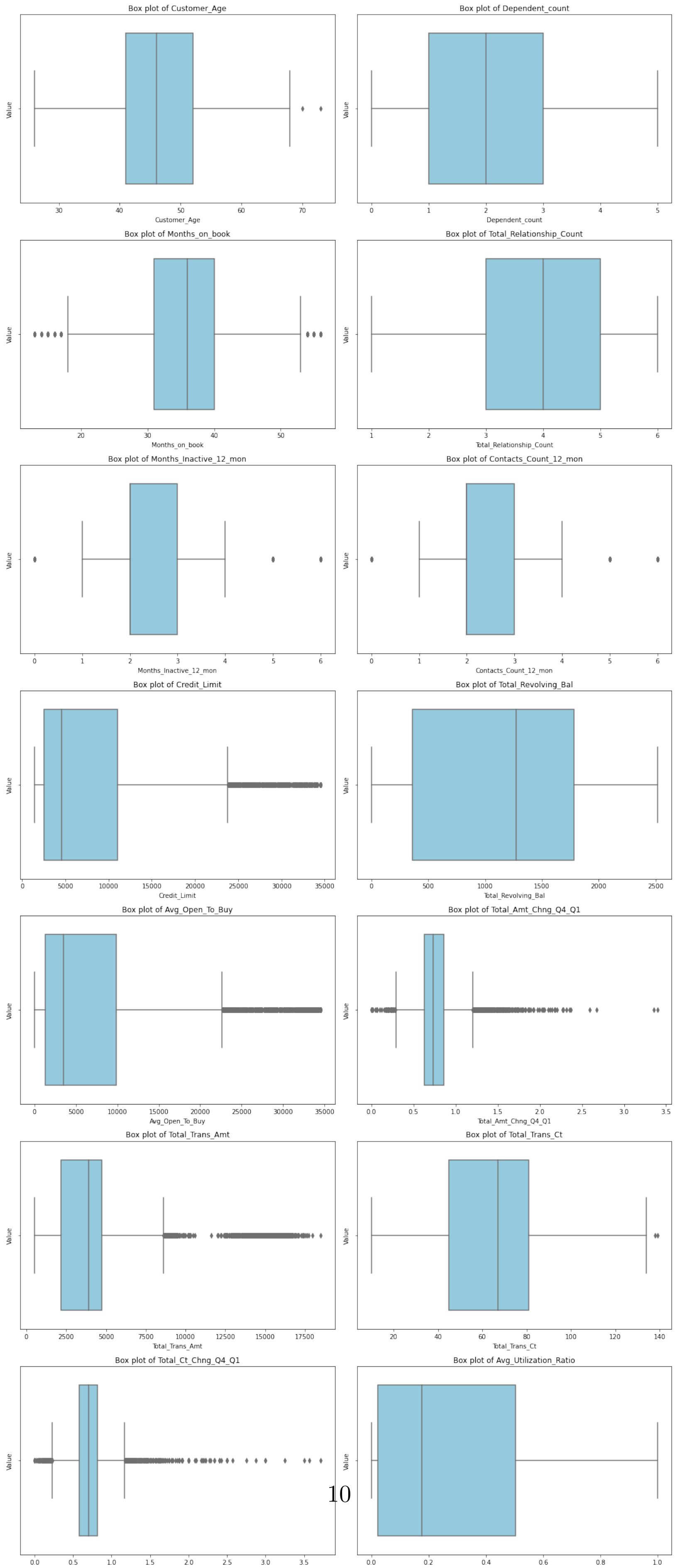
Checking the outliers on each feature

The below function would take the dataframe and crate a box sub plots for the numerical features

[12]:
```
def plot_boxplot(df, color='skyblue', kde=False):
    num_columns = len(df.columns)
    num_rows = (num_columns + 1) // 2

    fig, axes = plt.subplots(num_rows, 2, figsize=(15, 5 * num_rows))
    axes = axes.flatten()

    for i, column_name in enumerate(df.columns):
        sns.boxplot(x=df[column_name], color=color, ax=axes[i])
        axes[i].set_xlabel(column_name)
        axes[i].set_ylabel('Value')
        axes[i].set_title(f'Box plot of {column_name}')

        if kde:
            sns.kdeplot(df[column_name], color=color, ax=axes[i], linewidth=3)
```

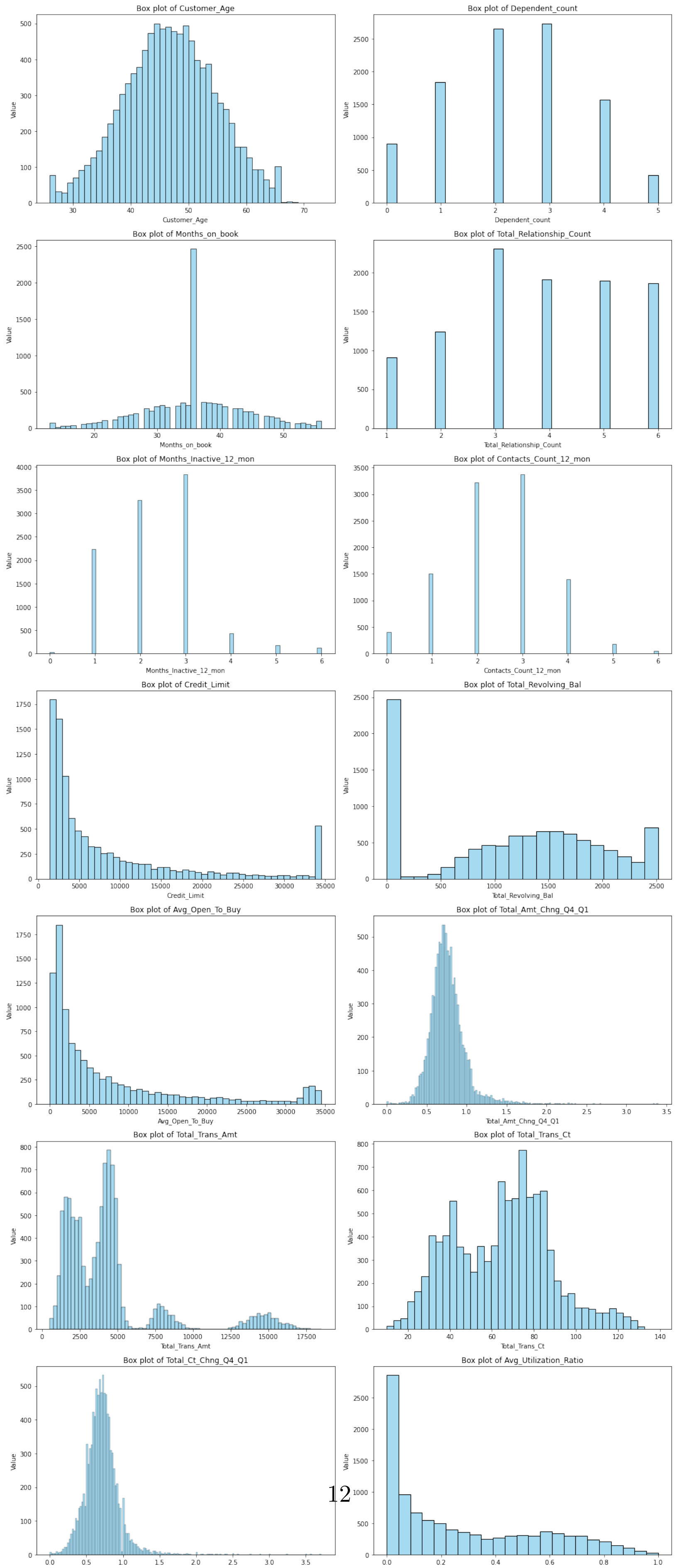
```
    # Remove the unused subplots (if any)
    for j in range(i+1, len(axes)):
        fig.delaxes(axes[j])

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

[13]:
```
plot_boxplot(df_numeric)
```


Box plot of Customer_Age
Customer_Age
Box plot of Dependent_count
Dependent_count
Box plot of Months_on_book
Months_on_book
Box plot of Total_Relationship_Count
Total_Relationship_Count
Box plot of Months_Inactive_12_mon
Months_Inactive_12_mon
Box plot of Contacts_Count_12_mon
Contacts_Count_12_mon
Box plot of Credit_Limit
Credit_Limit
Box plot of Total_Revolving_Bal
Total_Revolving_Bal
Box plot of Avg_Open_To_Buy
Avg_Open_To_Buy
Box plot of Total_Amt_Chng_Q4_Q1
Total_Amt_Chng_Q4_Q1
Box plot of Total_Trans_Amt
Total_Trans_Amt
Box plot of Total_Trans_Ct
Total_Trans_Ct
Box plot of Total_Ct_Chng_Q4_Q1
Total_Ct_Chng_Q4_Q1
Box plot of Avg_Utilization_Ratio
Avg_Utilization_Ratio
Value

[14]:
```
def plot_histogram(df, color='skyblue', kde=False):
    num_columns = len(df.columns)
    num_rows = (num_columns + 1) // 2

    fig, axes = plt.subplots(num_rows, 2, figsize=(15, 5 * num_rows))
    axes = axes.flatten()

    for i, column_name in enumerate(df.columns):
        sns.histplot(x=df[column_name], color=color, ax=axes[i])
        axes[i].set_xlabel(column_name)
        axes[i].set_ylabel('Value')
        axes[i].set_title(f'Box plot of {column_name}')

        if kde:
            sns.kdeplot(df[column_name], color=color, ax=axes[i], linewidth=3)

    # Remove the unused subplots (if any)
    for j in range(i+1, len(axes)):
        fig.delaxes(axes[j])

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

[15]:
```
plot_histogram(df_numeric)
```


Box plot of Customer_Age
Value
Customer_Age
Box plot of Dependent_count
Value
Dependent_count
Box plot of Months_on_book
Value
Months_on_book
Box plot of Total_Relationship_Count
Value
Total_Relationship_Count
Box plot of Months_Inactive_12_mon
Value
Months_Inactive_12_mon
Box plot of Contacts_Count_12_mon
Value
Contacts_Count_12_mon
Box plot of Credit_Limit
Value
Credit_Limit
Box plot of Total_Revolving_Bal
Value
Total_Revolving_Bal
Box plot of Avg_Open_To_Buy
Value
Avg_Open_To_Buy
Box plot of Total_Amt_Chng_Q4_Q1
Value
Total_Amt_Chng_Q4_Q1
Box plot of Total_Trans_Amt
Value
Total_Trans_Amt
Box plot of Total_Trans_Ct
Value
Total_Trans_Ct
Box plot of Total_Ct_Chng_Q4_Q1
Value
Total_Ct_Chng_Q4_Q1
Box plot of Avg_Utilization_Ratio
Value
Avg_Utilization_Ratio

Distribution: Seems like Customer age, Total_amt_Chang_q4_q1,Total_revolving balance, Total_Amt_Chng_Q4_Q1 are normally distributed and Total_Trans_amt have 4 distributions curve plots.

Checking all the unique values for the categorical variables and also checking which customers have churned or not

[16]:
```
df['Attrition_Flag'].unique()
```

[16]:
```
array(['Existing Customer', 'Attrited Customer'], dtype=object)
```

[17]:
```
df['Attrition_Flag'].value_counts()
```

[17]:
```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
```

[18]:
```
df['Education_Level'].unique()
```

[18]:
```
array(['High School', 'Graduate', 'Uneducated', 'Unknown', 'College',
       'Post-Graduate', 'Doctorate'], dtype=object)
```

[19]:
```
df['Marital_Status'].unique()
```

[19]:
```
array(['Married', 'Single', 'Unknown', 'Divorced'], dtype=object)
```

[20]:
```
df['Income_Category'].unique()
```

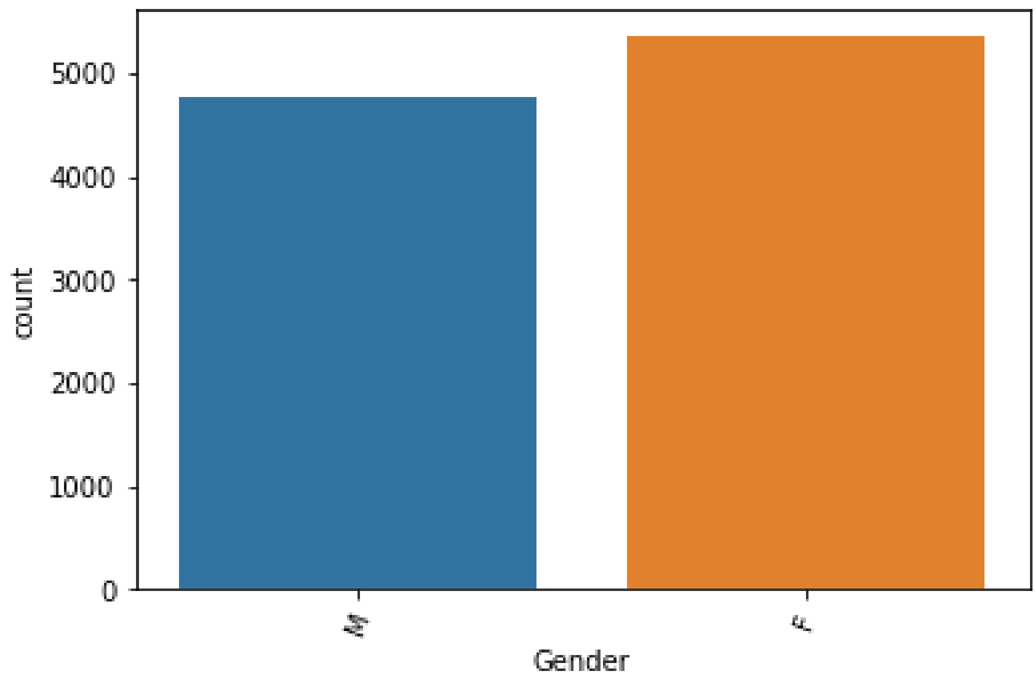
[20]:
```
array(['$60K - $80K', 'Less than $40K', '$80K - $120K', '$40K - $60K',
       '$120K +', 'Unknown'], dtype=object)
```

[21]:
```
df['Card_Category'].unique()
```

[21]:
```
array(['Blue', 'Gold', 'Silver', 'Platinum'], dtype=object)
```

Checking Gender feature

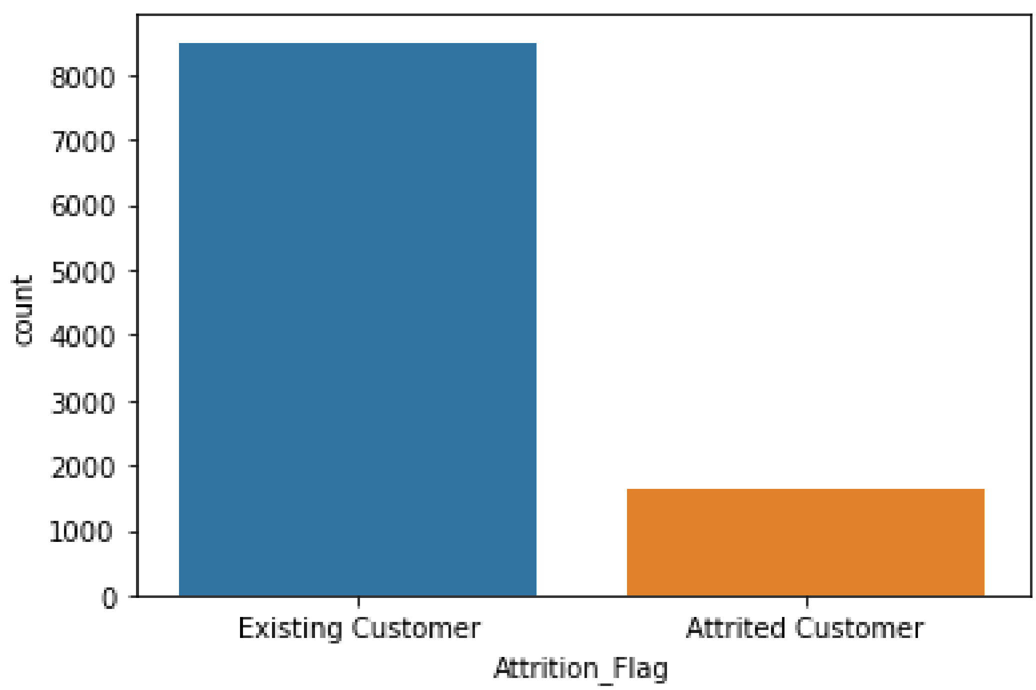
[22]:
```
sns.countplot(x='Gender', data=df)
plt.xticks(rotation=70)
plt.show()
```



Checking Attrition_flag count feature

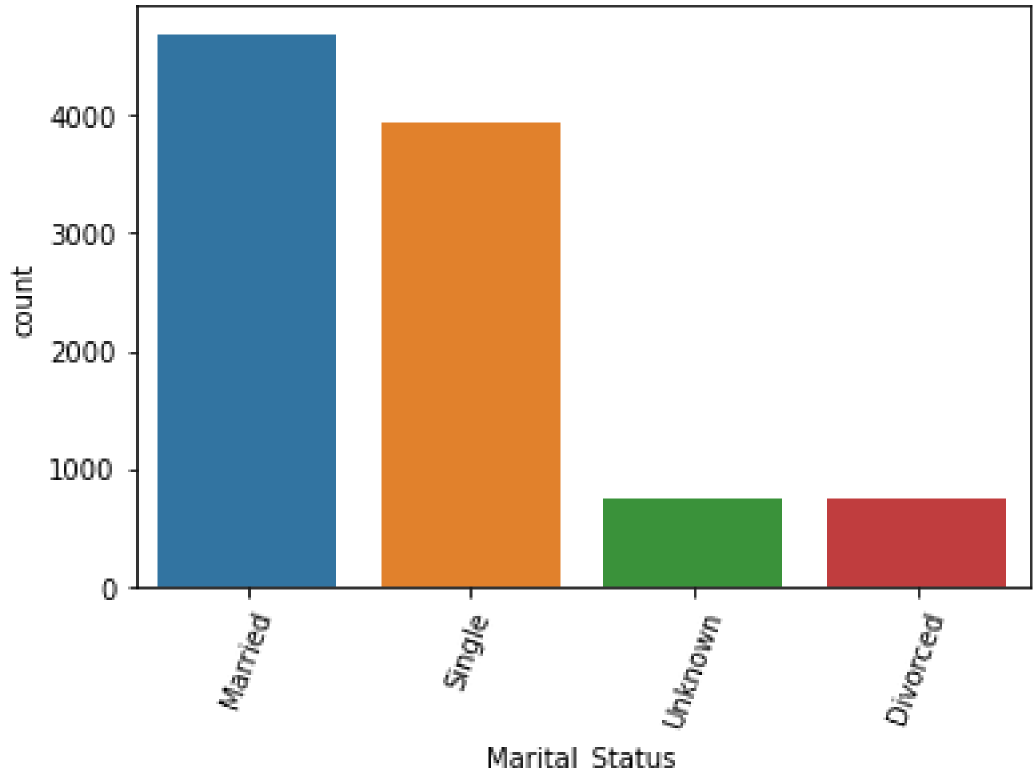
```
[23]: sns.countplot(x='Attrition_Flag', data=df)
```

```
[23]: <AxesSubplot:xlabel='Attrition_Flag', ylabel='count'>
```



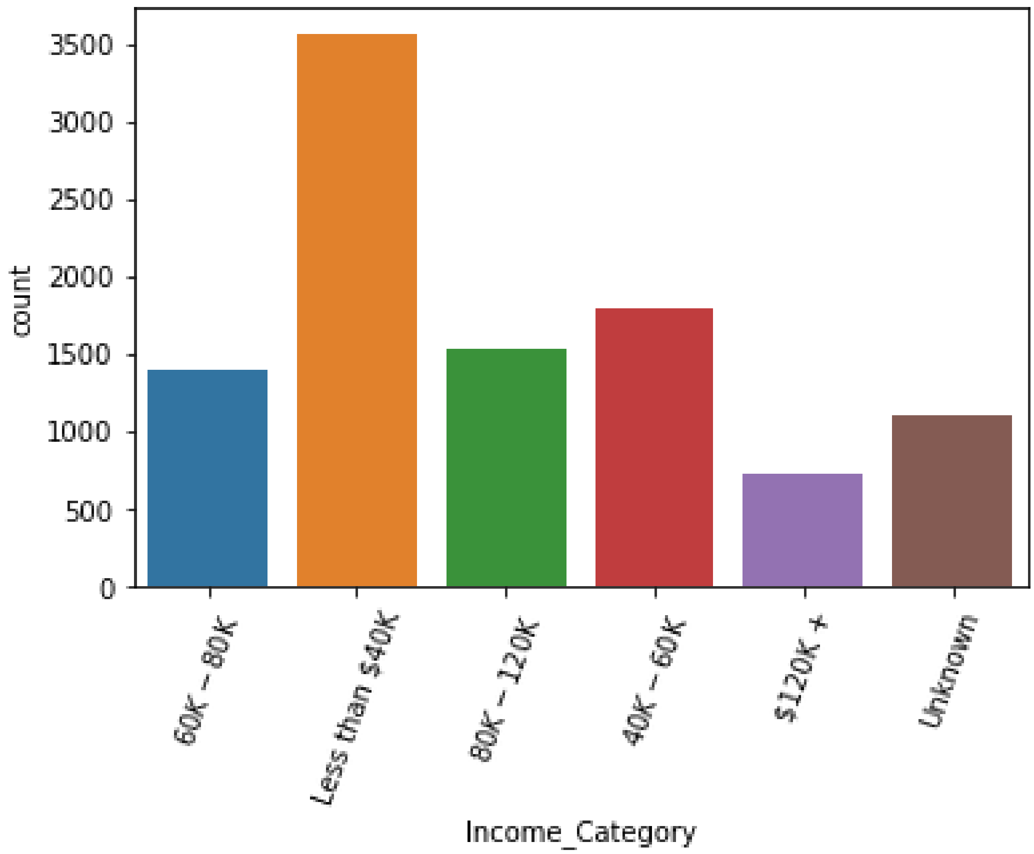
The target variable is higly imbalanced

[24]:
```
sns.countplot(x='Marital_Status', data=df)
plt.xticks(rotation=70)
plt.show()
```



Most of the customers are married and half are single and very less Unknown and divorced

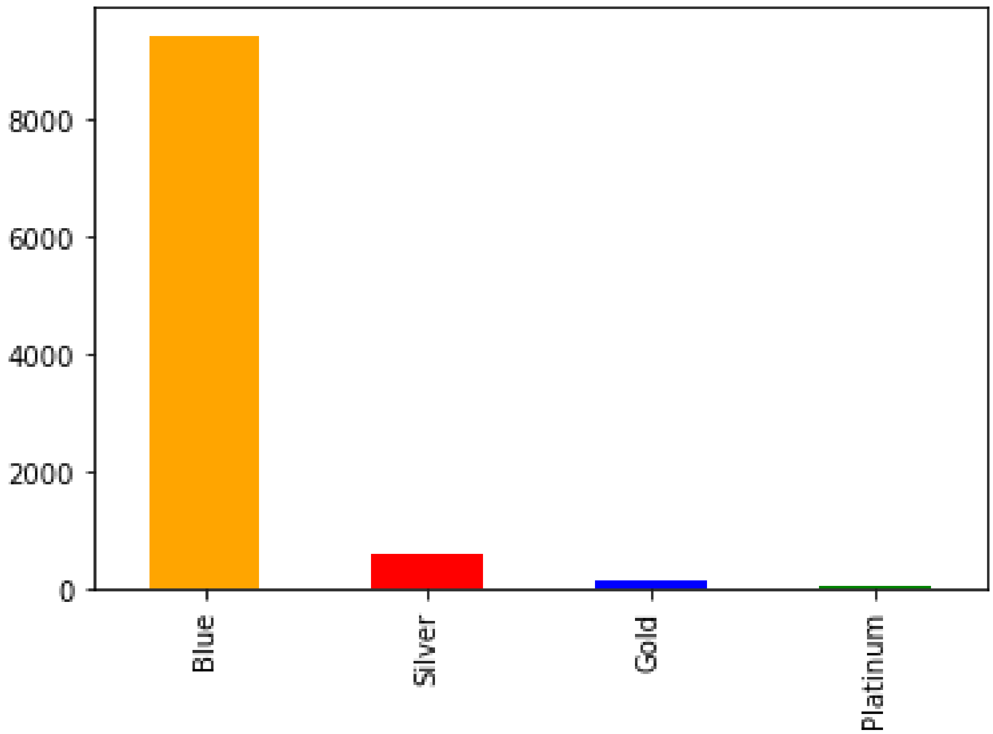
[25]:
```
sns.countplot(x='Income_Category', data=df)
plt.xticks(rotation=70)
plt.show()
```



More customers are less than 40k income and 40-60k are around 25%, less on 120K income level and more than 12% are unknown income levels

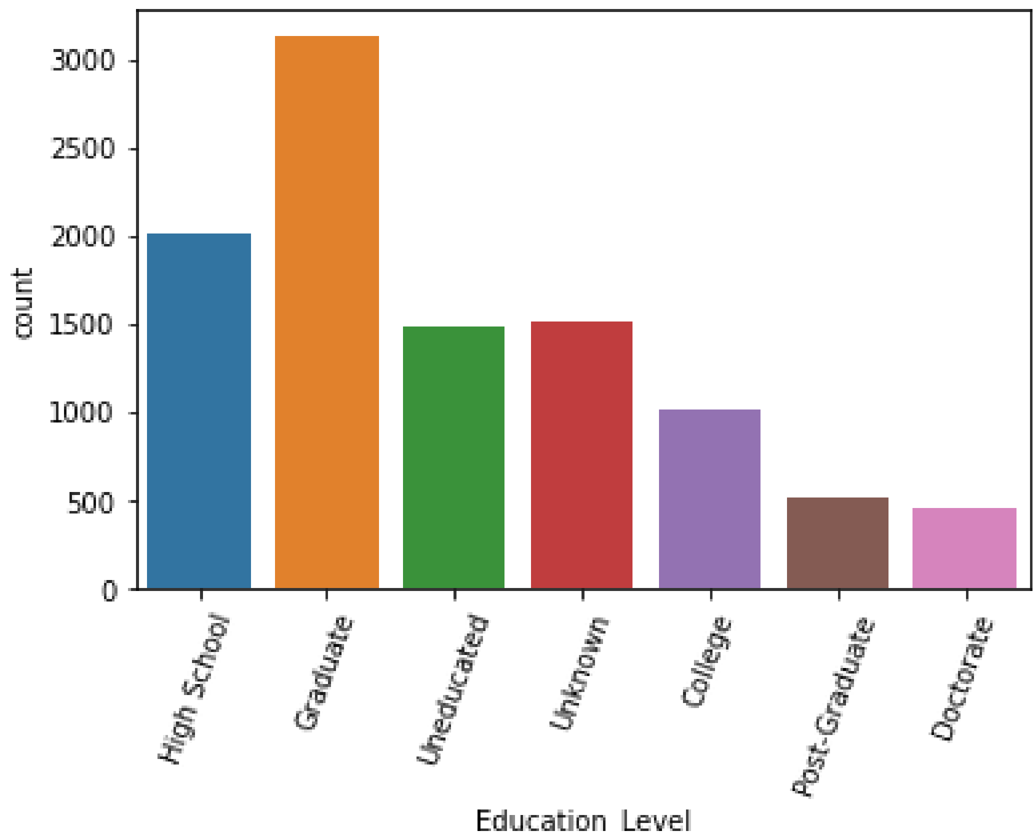
[26]:
```
df['Card_Category'].value_counts().plot.
↪bar(color=['orange','red','blue','green'])
```

[26]: <AxesSubplot:>



90% percent of the customer are held Blue card

[27]:
```
sns.countplot(x='Education_Level', data=df)
plt.xticks(rotation=70)
plt.show()
```



Most of the customers are graduates and have high school education. There are 30% population have unknown education

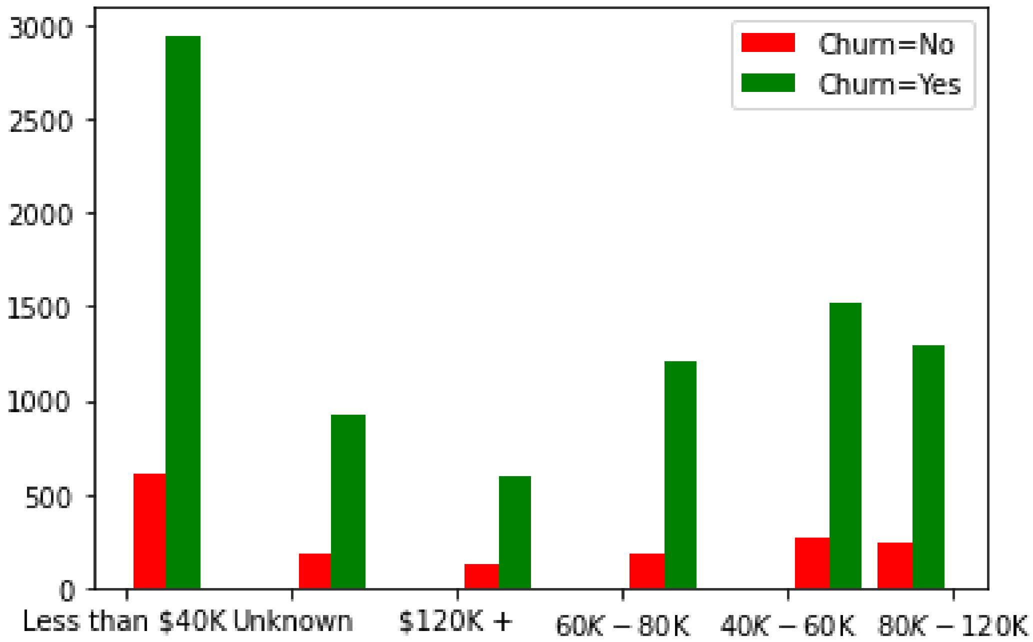
Setting attrition categorical feature value Existing Customer value 1 and Attrited Customer is 0

[28]:
```
target={'Existing Customer': 1,'Attrited Customer':0}
df['Attrition_Flag']= df['Attrition_Flag'].map(target)
```

Analyzing the which customers are more likely churning. Data Analysis

[29]:
```
atttrition_more_income = df[df.Attrition_Flag==1].Income_Category
atttrition_less_income = df[df.Attrition_Flag==0].Income_Category
plt.hist([atttrition_less_income,atttrition_more_income],color=['red','green'],␣
  ↪label=['Churn=No','Churn=Yes'])
plt.legend()
```

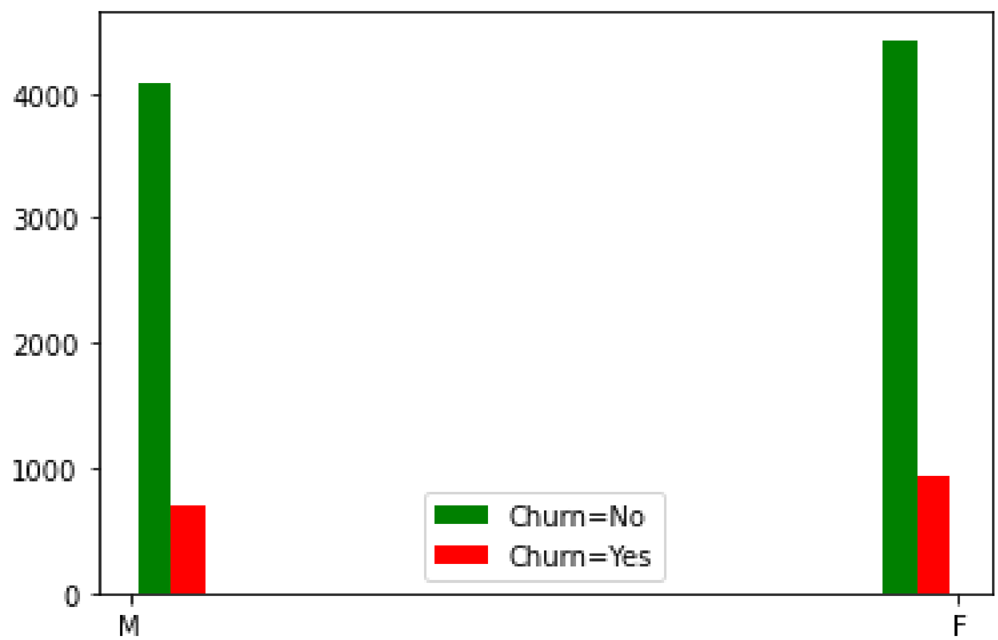
[29]:
```
<matplotlib.legend.Legend at 0x7fba58e835b0>
```



Less income people are have more churning rate

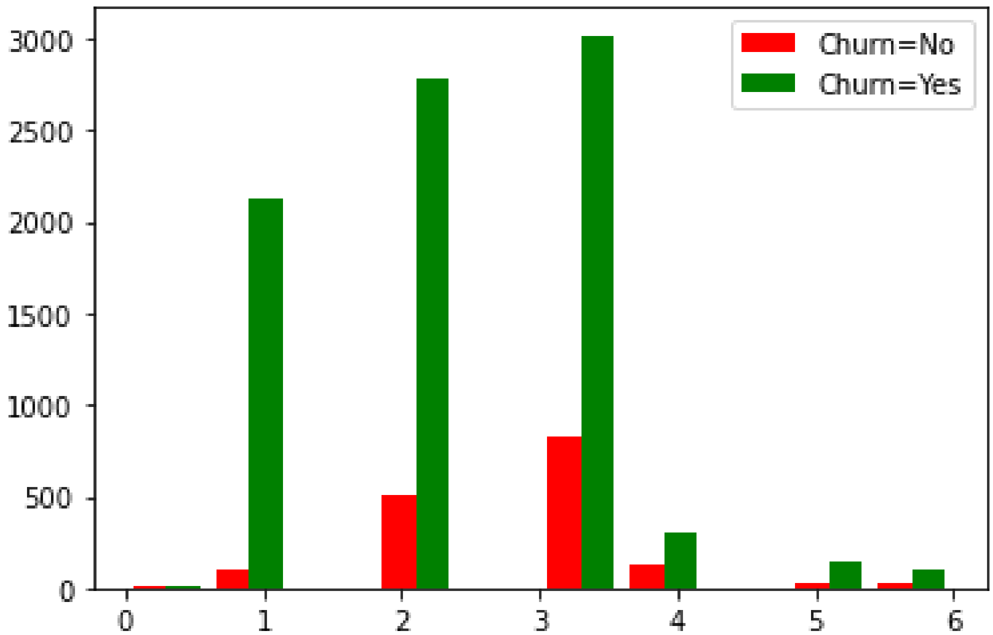
[30]:
```
not_churn_Gender = df[df.Attrition_Flag==1].Gender
churn_Gender = df[df.Attrition_Flag==0].Gender
plt.hist([not_churn_Gender,churn_Gender],color=['green','red'],␣
 ↪label=['Churn=No','Churn=Yes'])
plt.legend()
```

[30]: <matplotlib.legend.Legend at 0x7fba092dafd0>



[31]:
```
not_churn_total_inactive = df[df.Attrition_Flag==1].Months_Inactive_12_mon
churn_total_inactive = df[df.Attrition_Flag==0].Months_Inactive_12_mon
plt.hist([churn_total_inactive,not_churn_total_inactive],color=['red','green'],␣
  ↪label=['Churn=No','Churn=Yes'])
plt.legend()
```

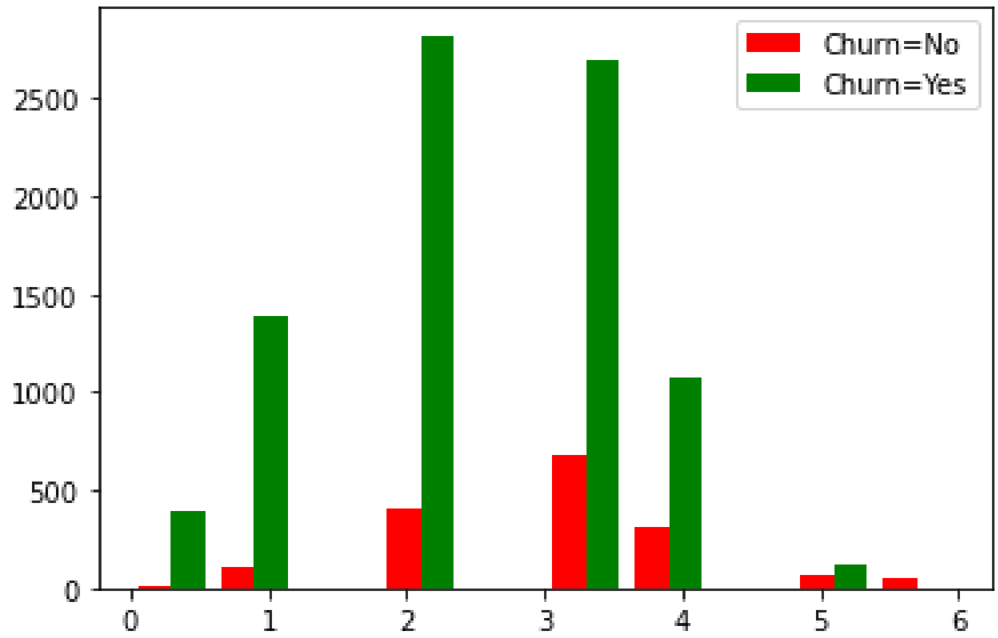
[31]:
```
<matplotlib.legend.Legend at 0x7fba39753070>
```



Not active card usage customers are more likely churning

[32]:
```
not_churn_contact_count = df[df.Attrition_Flag==1].Contacts_Count_12_mon
churn_contact_count = df[df.Attrition_Flag==0].Contacts_Count_12_mon
plt.hist([churn_contact_count,not_churn_contact_count],color=['red','green'],␣
 ↪label=['Churn=No','Churn=Yes'])
plt.legend()
```

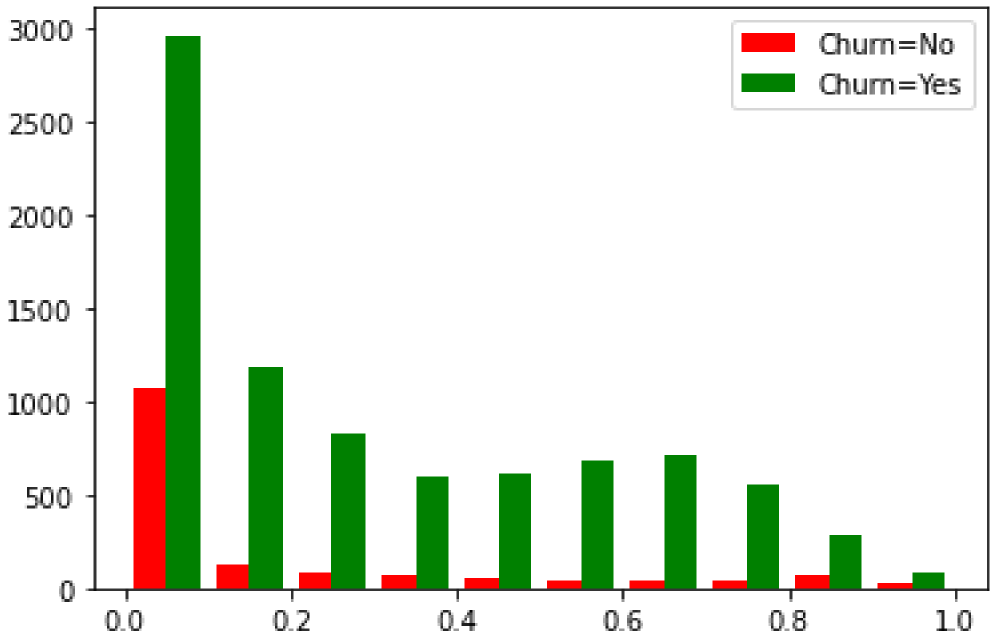
[32]: <matplotlib.legend.Legend at 0x7fba58f9c190>



Less communication to the bank are more likely to churn

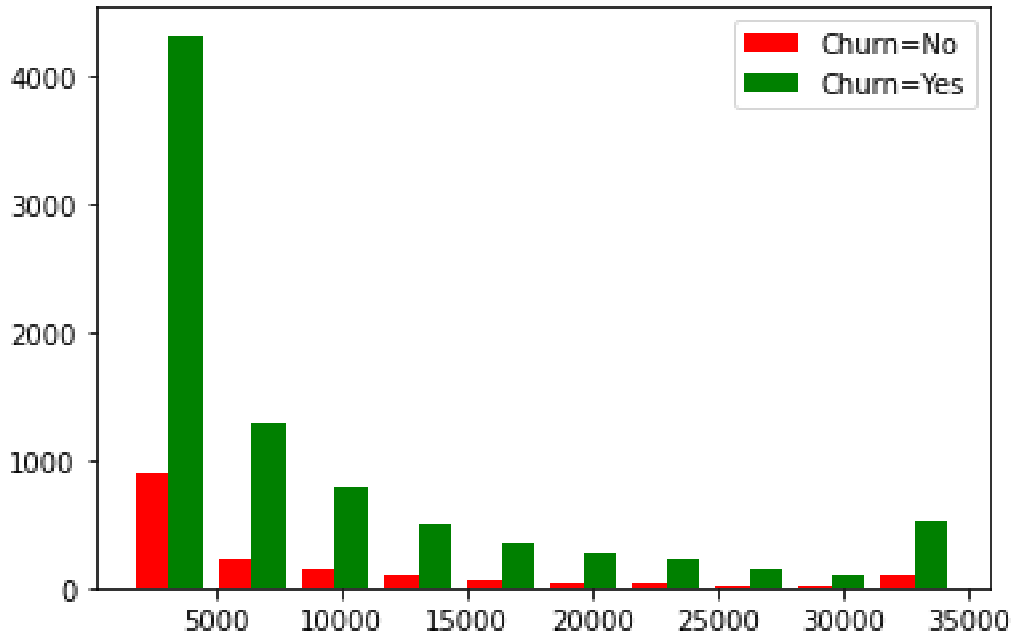
[33]:
```
not_churn_contact_count = df[df.Attrition_Flag==1].Avg_Utilization_Ratio
churn_contact_count = df[df.Attrition_Flag==0].Avg_Utilization_Ratio
plt.hist([churn_contact_count,not_churn_contact_count],color=['red','green'],␣
 ↪label=['Churn=No','Churn=Yes'])
plt.legend()
```

[33]: <matplotlib.legend.Legend at 0x7fba491a80d0>



[34]:
```
not_churn_credit_limit = df[df.Attrition_Flag==1].Credit_Limit
churn_credit_limit = df[df.Attrition_Flag==0].Credit_Limit
plt.hist([churn_credit_limit,not_churn_credit_limit],color=['red','green'],␣
    ↪label=['Churn=No','Churn=Yes'])
plt.legend()
```

[34]: <matplotlib.legend.Legend at 0x7fba18d7fca0>



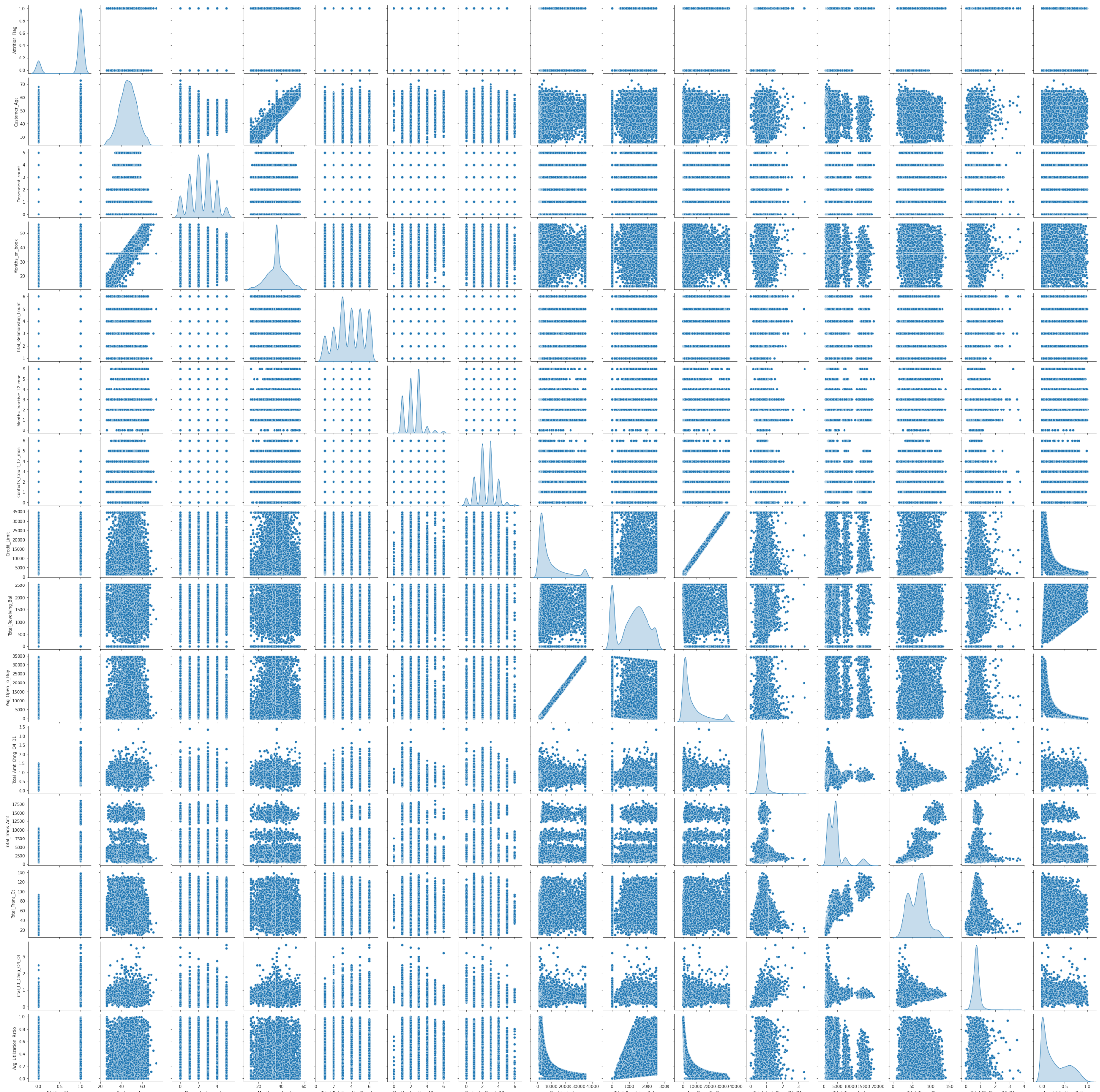
The attrition customers has a lower transaction count, revolving balance, average utilization ration and transaction amount compared to existing customers

Checking the distribution for the dataset

Creating pair plot from the dataset

[35]:
```
sns.pairplot(df, diag_kind='kde')
```

[35]: <seaborn.axisgrid.PairGrid at 0x7fba58eb7430>



Applying encodings to the Categorical variable

[36]:
```
gender={'M':0,'F':1}
df['Gender'] = df['Gender'].map(gender)
```

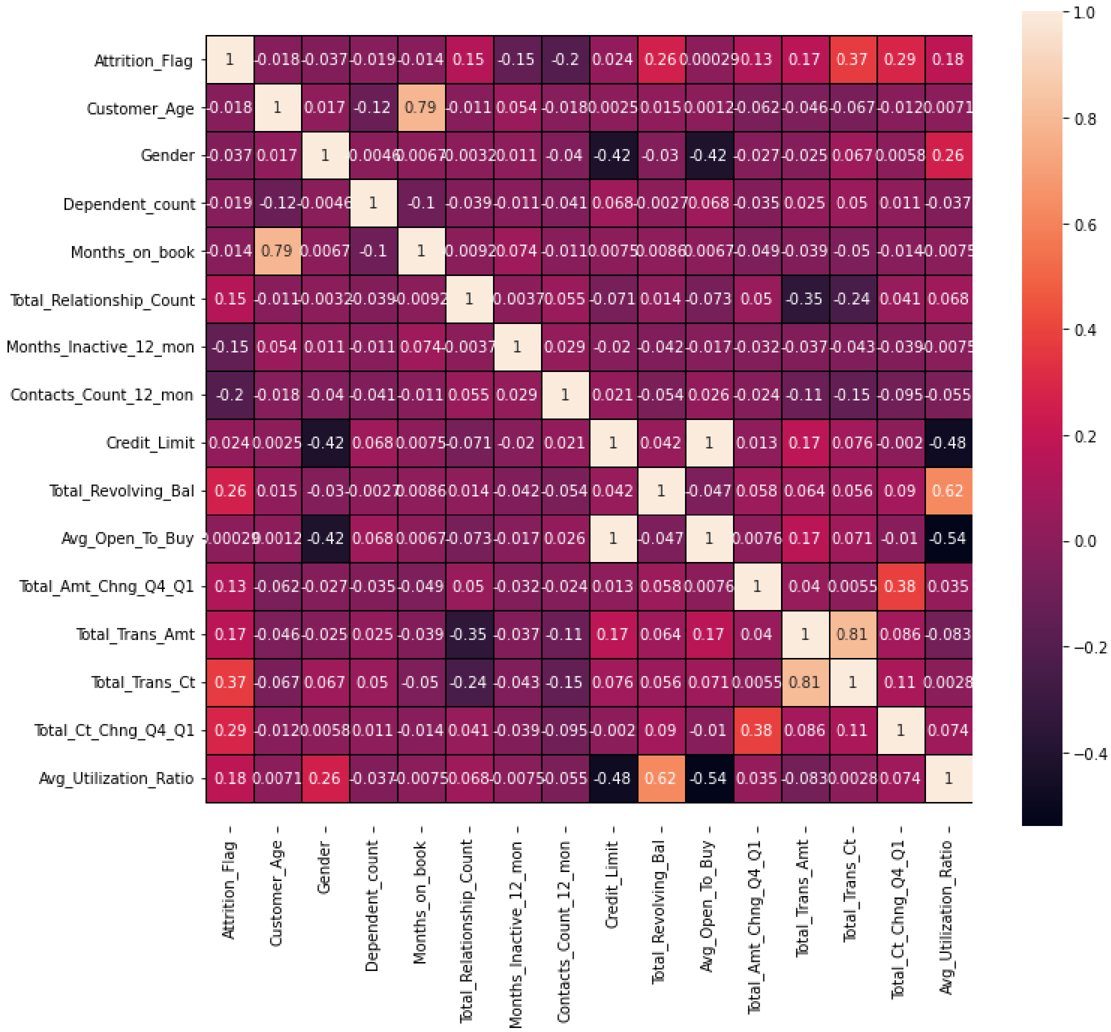
5 Second Step: Feature Importance or Selection

By Analysing the below correlation we can select what features can give more information and useful for the model predictions and also which features plays an important role. This will tell us which variables are more useful for to predict the chustomer churn.

Checking the corelation matrix

[37]:
```
matrix = np.triu(df.corr())
f, ax = plt.subplots(figsize=(12, 10))

ax = sns.heatmap(df.corr(), annot = True, square=True, \
            linewidths=1, linecolor='black')

bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[37]: (16.5, -0.5)



Findings from the correlation matrix

1. Average_open_to_buy highly correlated with credit_limit
2. Average_Utilization_ration is correlated with total_revolving_balance
3. Total_trans_amt is higly correlated with Total_Trans_ct

4. Total_Amt_Chang_Q4_Q1 is correlated to the Total_Ct_Chng_Q4_Q1

Will not select these features for model building Average_open_to_buy, Average_Utilization_ration, Total_trans_amt

Applying onehot encoding to the categorical features and creating a new dataframe

[38]:
```
cat_col = ['Income_Category','Education_Level','Marital_Status','Card_Category']
encoder = OneHotEncoder(sparse=False)
encoded_cols = encoder.fit_transform(df[cat_col])
encoded_cols_df = pd.DataFrame(encoded_cols, columns=encoder.
 ↪get_feature_names_out(cat_col))
new_df = pd.concat([df.drop(cat_col, axis=1), encoded_cols_df], axis=1)
```

[39]:
```
new_df.head()
```

[39]:
```
   Attrition_Flag  Customer_Age  Gender  Dependent_count  Months_on_book  \
0               1            45       0                3              39
1               1            49       1                5              44
2               1            51       0                3              36
3               1            40       1                4              34
4               1            40       0                3              21

   Total_Relationship_Count  Months_Inactive_12_mon  Contacts_Count_12_mon  \
0                         5                       1                      3
1                         6                       1                      2
2                         4                       1                      0
3                         3                       4                      1
4                         5                       1                      0

   Credit_Limit  Total_Revolving_Bal  Avg_Open_To_Buy  Total_Amt_Chng_Q4_Q1  \
0       12691.0                  777          11914.0                 1.335
1        8256.0                  864           7392.0                 1.541
2        3418.0                    0           3418.0                 2.594
3        3313.0                 2517            796.0                 1.405
4        4716.0                    0           4716.0                 2.175

   Total_Trans_Amt  Total_Trans_Ct  Total_Ct_Chng_Q4_Q1  \
0             1144              42                1.625
1             1291              33                3.714
2             1887              20                2.333
3             1171              20                2.333
4              816              28                2.500

   Avg_Utilization_Ratio  Income_Category_$120K +  \
0                  0.061                      0.0
1                  0.105                      0.0
2                  0.000                      0.0
```

```
3                0.760                        0.0
4                0.000                        0.0

   Income_Category_$40K - $60K  Income_Category_$60K - $80K  \
0                          0.0                          1.0
1                          0.0                          0.0
2                          0.0                          0.0
3                          0.0                          0.0
4                          0.0                          1.0

   Income_Category_$80K - $120K  Income_Category_Less than $40K  \
0                           0.0                             0.0
1                           0.0                             1.0
2                           1.0                             0.0
3                           0.0                             1.0
4                           0.0                             0.0

   Income_Category_Unknown  Education_Level_College  \
0                      0.0                      0.0
1                      0.0                      0.0
2                      0.0                      0.0
3                      0.0                      0.0
4                      0.0                      0.0

   Education_Level_Doctorate  Education_Level_Graduate  \
0                        0.0                       0.0
1                        0.0                       1.0
2                        0.0                       1.0
3                        0.0                       0.0
4                        0.0                       0.0

   Education_Level_High School  Education_Level_Post-Graduate  \
0                          1.0                            0.0
1                          0.0                            0.0
2                          0.0                            0.0
3                          1.0                            0.0
4                          0.0                            0.0

   Education_Level_Uneducated  Education_Level_Unknown  \
0                         0.0                      0.0
1                         0.0                      0.0
2                         0.0                      0.0
3                         0.0                      0.0
4                         1.0                      0.0

   Marital_Status_Divorced  Marital_Status_Married  Marital_Status_Single  \
0                      0.0                     1.0                    0.0
```

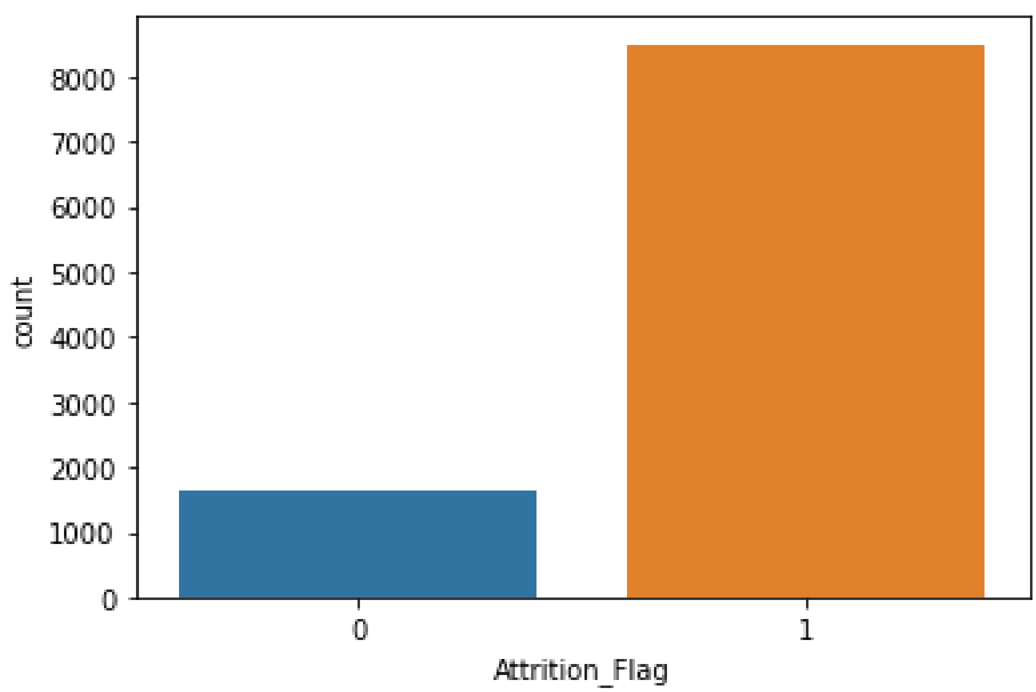
```
1                        0.0                      0.0                     1.0
2                        0.0                      1.0                     0.0
3                        0.0                      0.0                     0.0
4                        0.0                      1.0                     0.0

   Marital_Status_Unknown  Card_Category_Blue  Card_Category_Gold  \
0                     0.0                 1.0                 0.0
1                     0.0                 1.0                 0.0
2                     0.0                 1.0                 0.0
3                     1.0                 1.0                 0.0
4                     0.0                 1.0                 0.0

   Card_Category_Platinum  Card_Category_Silver
0                     0.0                   0.0
1                     0.0                   0.0
2                     0.0                   0.0
3                     0.0                   0.0
4                     0.0                   0.0
```

Dropping the unknown columns and card silver category from the features

[40]:
```
new_df.
 ↪drop(['Income_Category_Unknown','Education_Level_Unknown','Marital_Status_Unknown',
             'Card_Category_Silver'],axis=1,inplace=True)
```

[41]:
```
sns.countplot(x='Attrition_Flag', data=new_df)
```

[41]:
```
<AxesSubplot:xlabel='Attrition_Flag', ylabel='count'>
```



Here we can see that that the target cariable is highly imbalanced

6 Removing Colinear Features

Before we build a machine learning model we need to remove highly colinear with one another. We don't want to use multiple collinear features in our model. We are using corelation matrix to remove those features

Dropping these features

1. Avg_Open_To_Buy
2. Avg_Utilization_Ratio
3. Total_Trans_Amt

[42]:
```
new_df.
 ↪drop(['Avg_Open_To_Buy','Avg_Utilization_Ratio','Total_Trans_Amt'],axis=1,inplace=True)
```

7 Balancing the Dataset

The target variable is highley imbalanced Using SMOTE to make it balanced dataset

SMOTE: Synthetic Monority Oversampling Technique: SMOTE is an oversmapling technique where the synthetic sampleted are generated for the minority class. This helps to ovecome the overfitting problem by random oversampling the minority class. Before applying the SMOTE it is always best to split the data into train,test it avoid overfitting the data and for better performace of the model

[43]:
```
x= new_df.drop(['Attrition_Flag'],axis=1)
y= new_df['Attrition_Flag']
```

Spliting the data into train and test sets

[44]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪2,random_state=42)
```

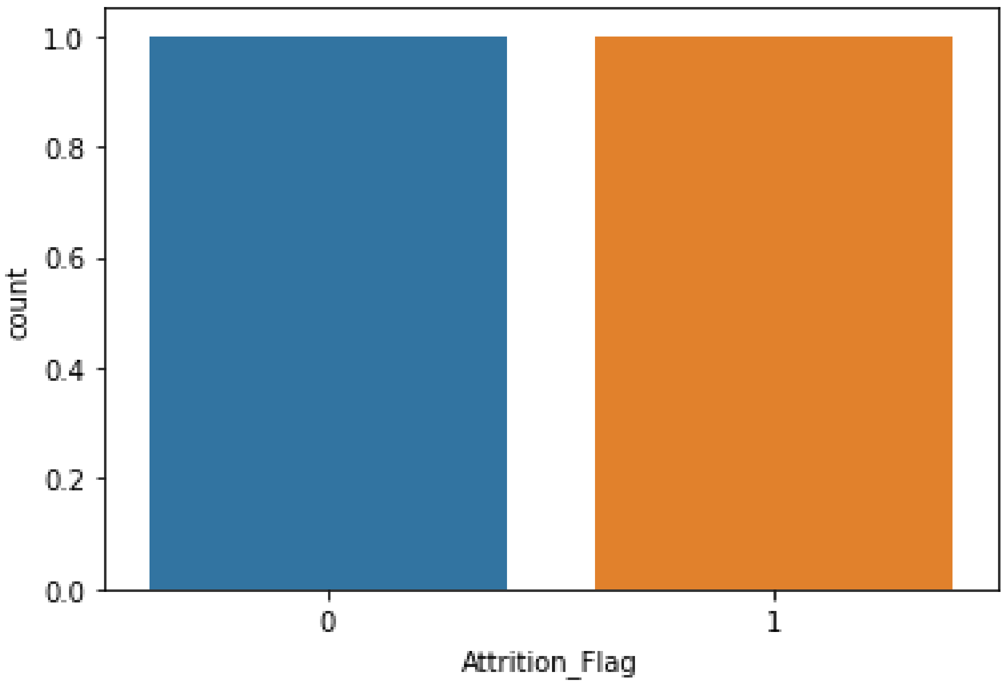
[45]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(8101, 29)
(2026, 29)
(8101,)
(2026,)
```

[46]:
```
smote = SMOTE()
x_train_over_sample, y_train_over_sample = smote.fit_resample(x_train, y_train)
```

```
counter_before = Counter(y)
counter_after = Counter(y_train_over_sample)
df_after = pd.DataFrame(list(counter_after.items()),␣
 ↪columns=["Attrition_Flag",'Count'])
sns.countplot(x='Attrition_Flag',data=df_after)
```

[46]: <AxesSubplot:xlabel='Attrition_Flag', ylabel='count'>



After SMOTE the data samples

[47]:
```
print(x_train_over_sample.shape)
print(x_test.shape)
print(y_train_over_sample.shape)
print(y_test.shape)
```

```
(13602, 29)
(2026, 29)
(13602,)
(2026,)
```

7.0.1 Shuffling the training data

The sample method shuffles the rows in a dataframe

[48]:

```
resampled_data = pd.concat([x_train_over_sample, y_train_over_sample], axis=1)
shuffled_resampled_data = resampled_data.sample(frac=1, random_state=42).
↪reset_index(drop=True)
x_over_train_shuffled = shuffled_resampled_data.drop('Attrition_Flag', axis=1)
y_over_train_shuffled = shuffled_resampled_data['Attrition_Flag']
```

Here the smote is added around 4000 samples to the target variable to balance the dataset

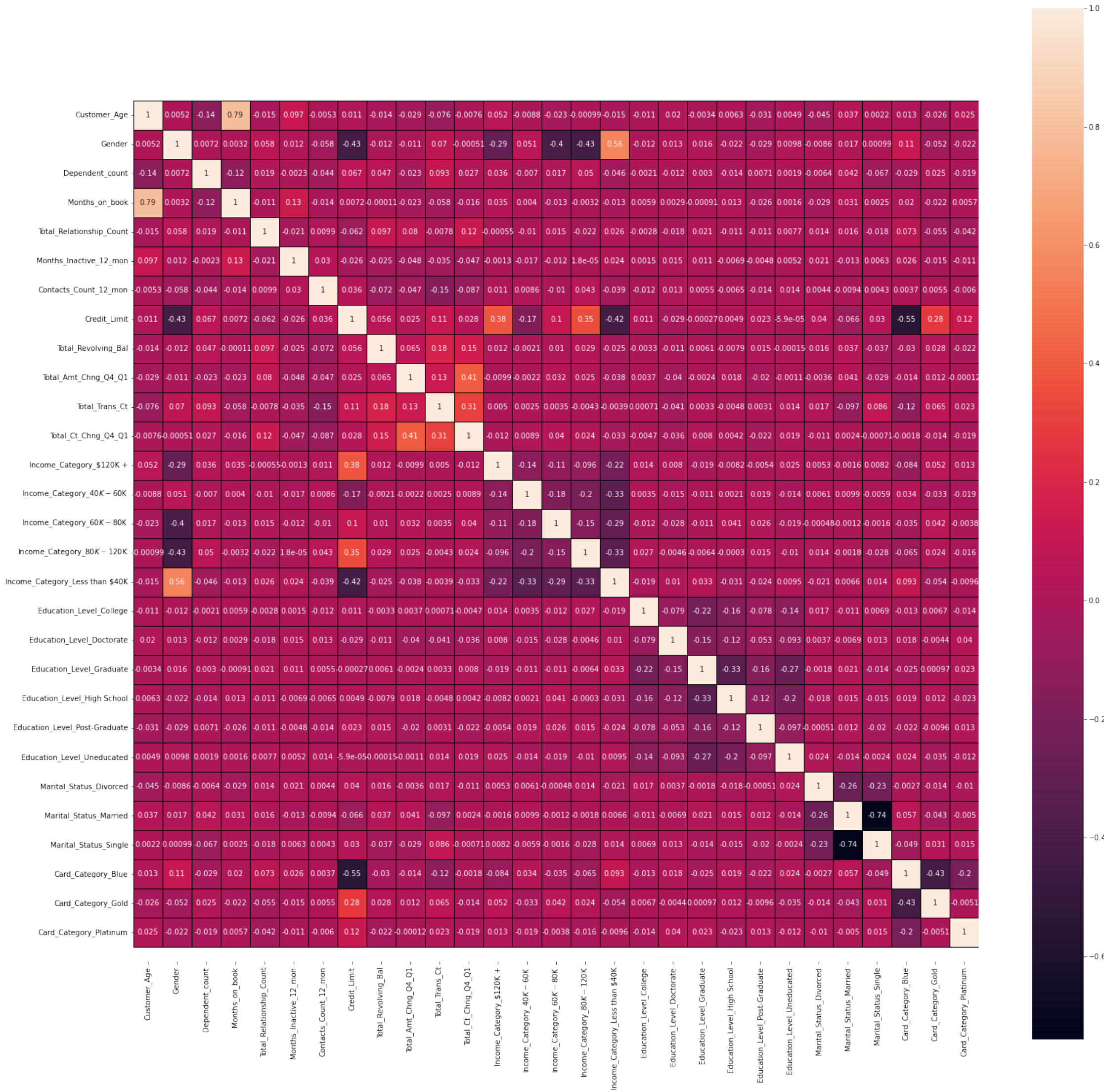
Plotting to see after dropping the highly correlated features with balanced training dataset

[49]:

```
matrix = np.triu(x_over_train_shuffled.corr())
f, ax = plt.subplots(figsize=(26, 26))

ax = sns.heatmap(x_over_train_shuffled.corr(), annot = True, square=True, \
            linewidths=1, linecolor='black')

bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

[49]:

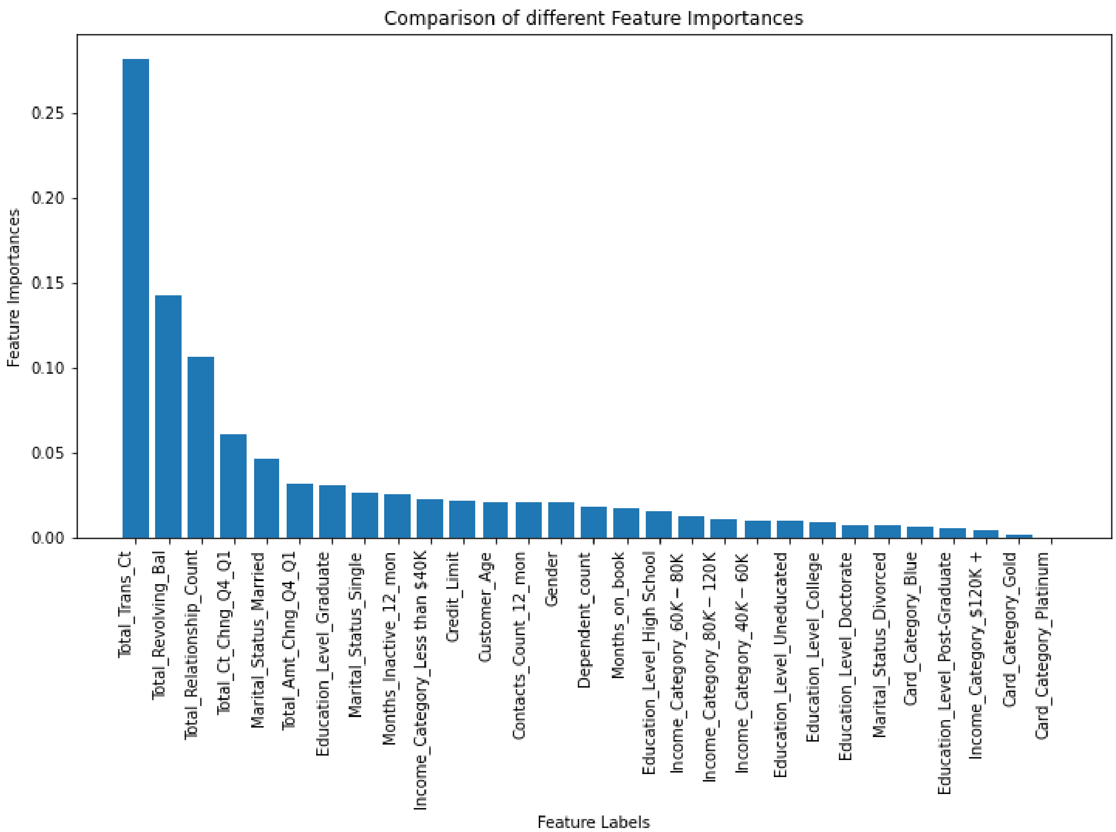
```
(29.5, -0.5)
```



8 Using ExtraTreesClassifier to select the features

[50]:
```
extra_tree_forest = ExtraTreesClassifier(n_estimators = 32, criterion
↪='entropy', max_features = 20)
extra_tree_forest.fit(x_over_train_shuffled, y_over_train_shuffled)

feature_importance = extra_tree_forest.feature_importances_

importance_df = pd.DataFrame({'feature': x_over_train_shuffled.columns,
↪'importance': feature_importance})

importance_df = importance_df.sort_values('importance', ascending=False)
```

```
plt.figure(figsize=(12, 6))

plt.bar(importance_df['feature'], importance_df['importance'])
plt.xticks(rotation=90, ha='right')
plt.xlabel('Feature Labels')
plt.ylabel('Feature Importances')
plt.title('Comparison of different Feature Importances')
plt.show()
```



Since I have dropped the multicolinear feature with dependent fetures it self. I am going with ExtraTreesClassifier to select the features. However we can also apply PCA to reduce the dimentionality reduction and features selection.

8.1 Selecting the features for model building

[51]:
```
x_train_over_sample_df=x_over_train_shuffled[['Total_Trans_Ct','Total_Revolving_Bal',
'Total_Relationship_Count','Total_Ct_Chng_Q4_Q1','Months_Inactive_12_mon']]
```

[52]:
```
x_test_over_sample_df=x_test[['Total_Trans_Ct','Total_Revolving_Bal','Total_Relationship_Count
                    'Total_Ct_Chng_Q4_Q1','Months_Inactive_12_mon']]
```

Checking the normal distribution for the numerical features and transform them

Below method would check numerical features are noamally distributed or not, based on that we are going to apply scalar to the features

[53]:
```
def test_normality(df):
    normality_results = {}

    for column_name in df.columns:
        reject_h0 = False

        data = df[column_name]
        nordistest = anderson(data)

        for i in range(len(nordistest.critical_values)):
            sl, cv = nordistest.significance_level[i], nordistest.
 ↪critical_values[i]
            if nordistest.statistic >= cv:
                reject_h0 = True
                break

        if reject_h0:
            conclusion = f"Data is not normally distributed (reject H0)"
        else:
            conclusion = f"Data is normally distributed (fail to reject H0)"

        normality_results[column_name] = conclusion

    return normality_results
```

[54]:
```
normality_results = test_normality(x_train_over_sample_df)
for column_name, result in normality_results.items():
    print(f"Normality: '{column_name}': {result}")
```

```
Normality: 'Total_Trans_Ct': Data is not normally distributed (reject H0)
Normality: 'Total_Revolving_Bal': Data is not normally distributed (reject H0)
Normality: 'Total_Relationship_Count': Data is not normally distributed (reject
H0)
Normality: 'Total_Ct_Chng_Q4_Q1': Data is not normally distributed (reject H0)
Normality: 'Months_Inactive_12_mon': Data is not normally distributed (reject
H0)
```

Applying the Standard scalar to the features before sending data to the model

[55]:
```
scaler = MinMaxScaler()
x_max_train_scaled = scaler.fit_transform(x_train_over_sample_df)
x_max_test_scaled = scaler.transform(x_test_over_sample_df)
```

9 Third Step: Machine Learning Models

Model Building Here I am building different models using RandomizedsearcgCV to find the best parameters and use these parameters to build a models

Running hyperparameters individually for the models

Optimization

10 Hyperparameter Tuning for the models

Random Forest tuning

[56]:
```
rfparameters= {
    'n_estimators':[200,400],
    'max_features':['auto','sqrt','log2'],
    'max_depth':[10,20,30,40],
    'criterion':['gini','entropy']}
rfclf= RandomForestClassifier()
rand_rfclf = RandomizedSearchCV(rfclf, rfparameters, cv = 5, verbose=True,␣
 ↪n_jobs=2)
rand_rfclf.fit(x_max_train_scaled,y_over_train_shuffled)
print(rand_rfclf.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 200, 'max_features': 'log2', 'max_depth': 40, 'criterion':
'gini'}
```

XGboost Tuning

[57]:
```
xgbparams = {
    "learning_rate": [0.01, 0.1, 0.2],
    "max_depth": [3, 5, 7, 9],
    "n_estimators": [50, 100, 150, 200],
    "min_child_weight": [1, 3, 5],
    "subsample": [0.8, 0.9, 1.0],
    "colsample_bytree": [0.8, 0.9, 1.0],
    "gamma": [0, 1, 5]
}

xgbclf= XGBClassifier()
xgb_clf = RandomizedSearchCV(xgbclf, xgbparams, cv = 5, verbose=True, n_jobs=2)
xgb_clf.fit(x_max_train_scaled,y_over_train_shuffled)
print(xgb_clf.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'subsample': 1.0, 'n_estimators': 150, 'min_child_weight': 3, 'max_depth': 7,
'learning_rate': 0.1, 'gamma': 1, 'colsample_bytree': 1.0}
```

Adaboost Tuning

[58]:
```
adaparams= {
            'base_estimator__max_depth': [10, 40, 50],
            'n_estimators': [10, 50, 100, 200,300],
            'learning_rate': [0.01, 0.1, 0.5, 1],
        }

adaclf=AdaBoostClassifier(base_estimator=DecisionTreeClassifier())
ada_boostclf = RandomizedSearchCV(adaclf, adaparams, cv = 5, verbose=True,␣
 ↪n_jobs=-1)
ada_boostclf.fit(x_max_train_scaled,y_over_train_shuffled)
print(ada_boostclf.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'n_estimators': 300, 'learning_rate': 0.01, 'base_estimator__max_depth': 10}
```

After applying the training dataset for the classification models Adaboost, Random forest and XGboost are giving the best results so using these 3 models to fir the data and make predictions

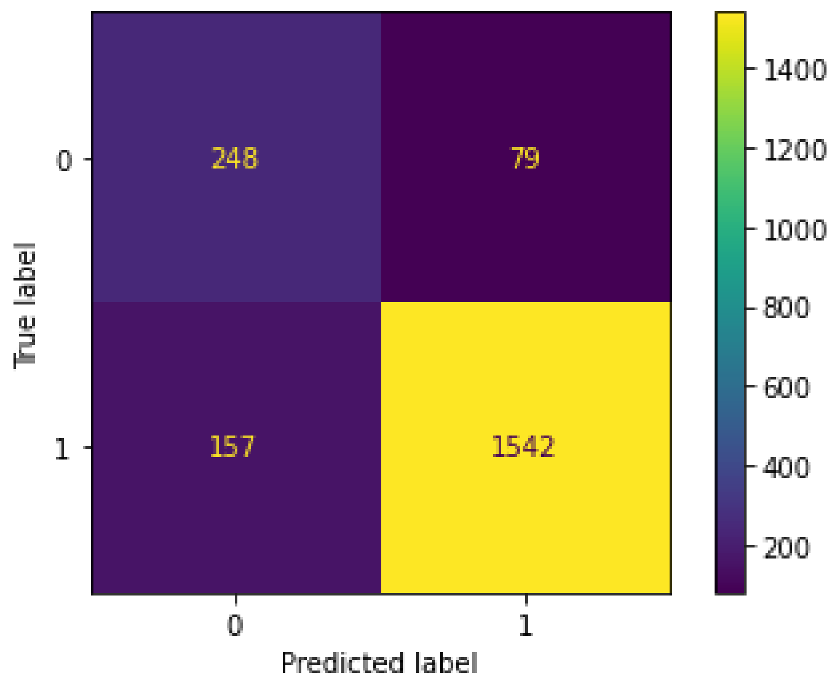
10.1 Classification of Churn rate

Evaluate each model and store it into dataframe The below method would take classifier, train and target variales and diaplys confusion matrix and roc curve

10.1.1 Classifiers

1. Random Forest Classifier
2. XGboost Classifier
3. Adaboost Classifier
   These are supervised models can use it for classification problems

[59]:
```
rand_clf=␣
 ↪RandomForestClassifier(max_features='log2',n_estimators=200,max_depth=40,criterion=␣
 ↪'gini')
rand_clf.fit(x_max_train_scaled,y_over_train_shuffled)
rand_rc_pred = rand_clf.predict(x_max_test_scaled)
cm = confusion_matrix(y_test, rand_rc_pred,labels=rand_clf.classes_)
ConfusionMatrixDisplay.from_estimator(rand_clf,x_max_test_scaled, y_test)
plt.show()
print(confusion_matrix(y_test,rand_rc_pred))
print(classification_report(y_test,rand_rc_pred))
print("Accuracy Score is " , (accuracy_score(y_test, rand_rc_pred)*100))
```

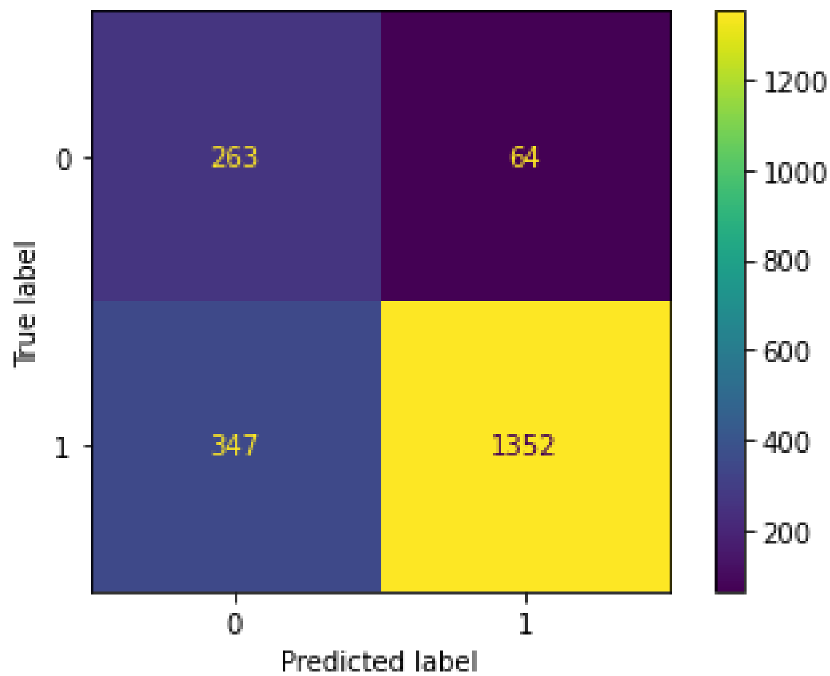



```
[[ 248   79]
 [ 157 1542]]
              precision    recall  f1-score   support

           0       0.61      0.76      0.68       327
           1       0.95      0.91      0.93      1699

    accuracy                           0.88      2026
   macro avg       0.78      0.83      0.80      2026
weighted avg       0.90      0.88      0.89      2026

Accuracy Score is  88.35143139190524
```

11 Adaboost model

[60]:
```
adaboostmodel = AdaBoostClassifier(n_estimators=300,learning_rate=0.01)
adaboostmodel.fit(x_max_train_scaled,y_over_train_shuffled)

adaboost_pred = adaboostmodel.predict(x_max_test_scaled)
cm = confusion_matrix(y_test, adaboost_pred,labels=adaboostmodel.classes_)
ConfusionMatrixDisplay.from_estimator(adaboostmodel,x_max_test_scaled, y_test)
plt.show()
print(confusion_matrix(y_test,adaboost_pred))
print(classification_report(y_test,adaboost_pred))
```

```
print("Accuracy Score is " , (accuracy_score(y_test, adaboost_pred)*100))
adaboost_roc_auc = roc_auc_score(y_test, adaboost_pred)
print("Adaboost AUC = %0.2f" %adaboost_roc_auc)
```

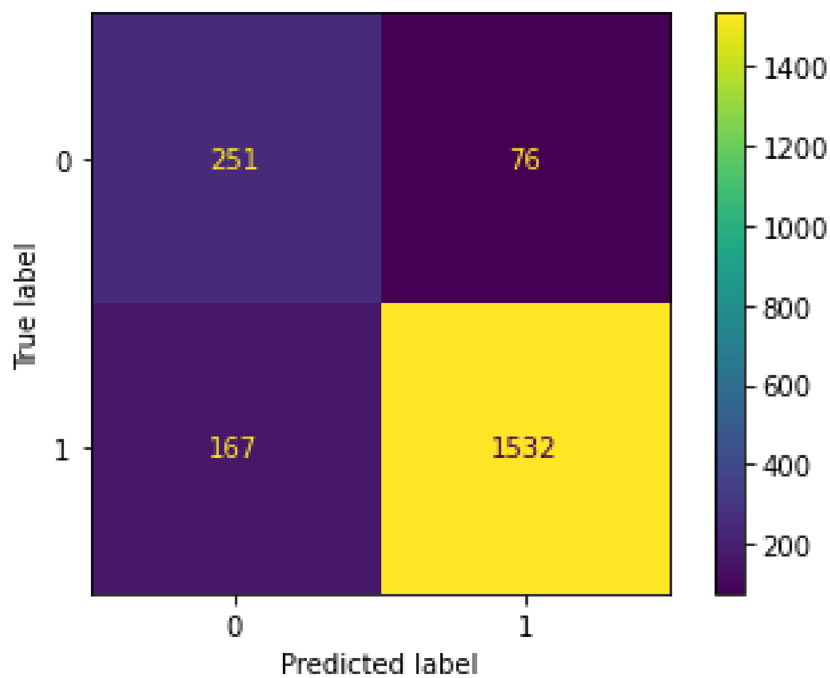



```
[[ 263   64]
 [ 347 1352]]
              precision    recall  f1-score   support

           0       0.43      0.80      0.56       327
           1       0.95      0.80      0.87      1699

    accuracy                           0.80      2026
   macro avg       0.69      0.80      0.71      2026
weighted avg       0.87      0.80      0.82      2026

Accuracy Score is  79.7137216189536
Adaboost AUC = 0.80
```

12 XGboost

XGboost documentation: XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework.

[61]:
```
xgb = XGBClassifier(subsample= 1.0, n_estimators= 150, min_child_weight= 3,␣
↪max_depth= 7,
                        learning_rate=0.1, gamma= 1, colsample_bytree= 1.0)
xgb.fit(x_max_train_scaled,y_over_train_shuffled)
xgboost_pred = xgb.predict(x_max_test_scaled)

ConfusionMatrixDisplay.from_estimator(xgb,x_max_test_scaled, y_test)
plt.show()
print("Accuracy Score is " , (accuracy_score(y_test, xgboost_pred)*100))
xgboost_roc_auc = roc_auc_score(y_test, xgboost_pred)
print(classification_report(y_test, xgboost_pred))
```

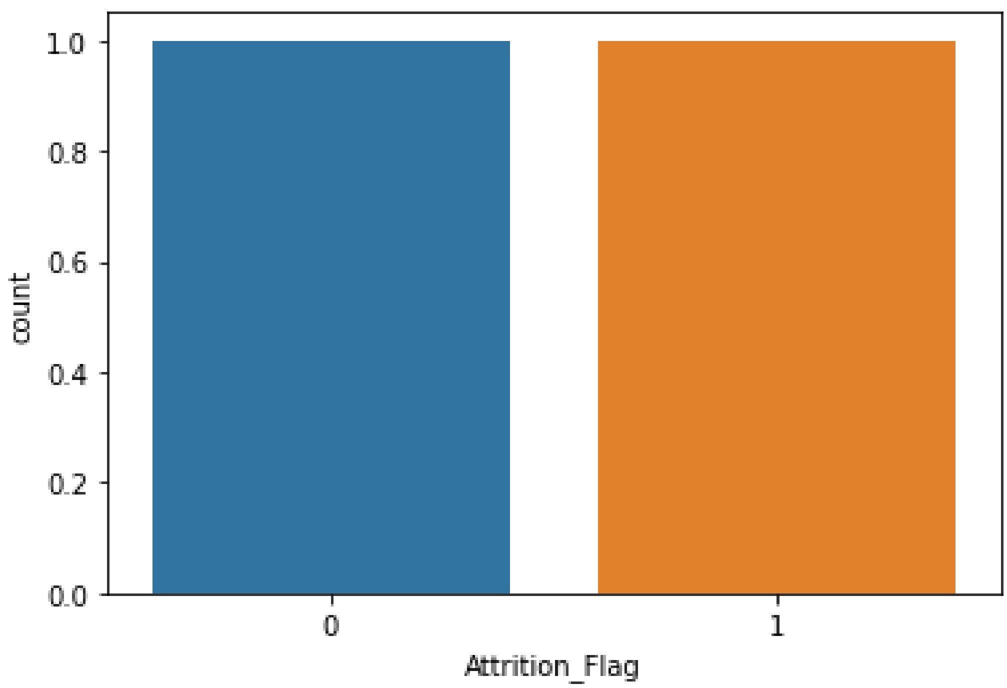



```
Accuracy Score is  88.00592300098717
              precision    recall  f1-score   support

           0       0.60      0.77      0.67       327
           1       0.95      0.90      0.93      1699

    accuracy                           0.88      2026
   macro avg       0.78      0.83      0.80      2026
weighted avg       0.90      0.88      0.89      2026
```

13 Undersampling

[62]:
```
undersample = RandomUnderSampler(random_state = 42)
x_train_under_sample, y_train_under_sample = undersample.fit_resample(x_train,
↪y_train)
counter_before = Counter(y_train)
counter_after = Counter(y_train_under_sample)
df_after = pd.DataFrame(list(counter_after.items()),
↪columns=["Attrition_Flag",'Count'])
sns.countplot(x='Attrition_Flag',data=df_after)
```

[62]: <AxesSubplot:xlabel='Attrition_Flag', ylabel='count'>



Shuffling after Undersampling the data

[63]:
```
resampled_data = pd.concat([x_train_under_sample, y_train_under_sample], axis=1)
shuffled_undersample_data = resampled_data.sample(frac=1, random_state=42).
↪reset_index(drop=True)
x_train_under_shuffled = shuffled_undersample_data.drop('Attrition_Flag',
↪axis=1)
y_train_under_shuffled = shuffled_undersample_data['Attrition_Flag']
```

[64]:
```
x_train_under_df=x_train_under_shuffled[['Total_Trans_Ct','Total_Revolving_Bal',
'Total_Relationship_Count','Total_Ct_Chng_Q4_Q1','Months_Inactive_12_mon']]
```

```
x_test_under_df=x_test[['Total_Trans_Ct','Total_Revolving_Bal','Total_Relationship_Count',
                  'Total_Ct_Chng_Q4_Q1','Months_Inactive_12_mon']]
```

[65]:

```
scaler = MinMaxScaler()
x_max_under_train_scaled = scaler.fit_transform(x_train_under_df)
x_max_under_test_scaled = scaler.transform(x_test_under_df)
```

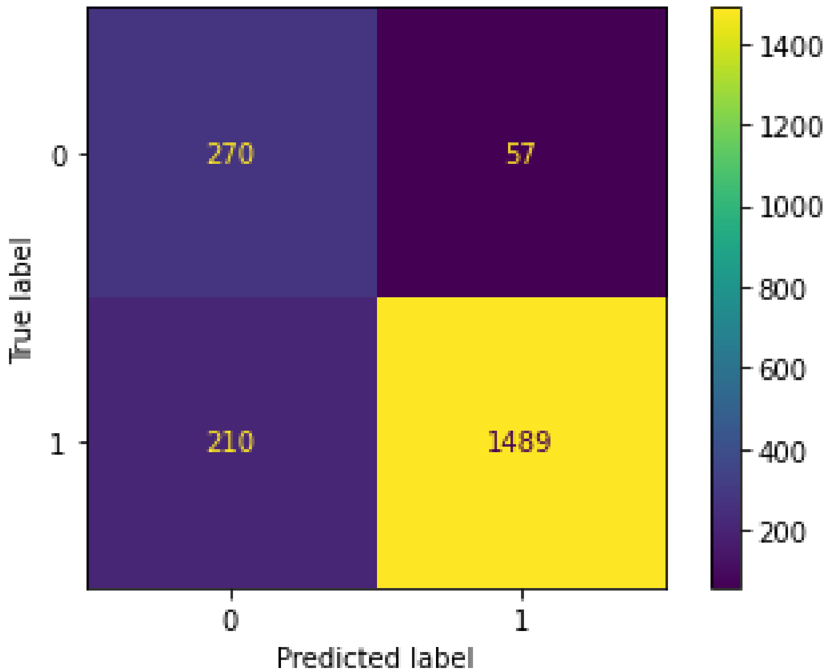
[66]:

```
xgbparams = {
    "learning_rate": [0.01, 0.1, 0.2],
    "max_depth": [3, 5, 7, 9],
    "n_estimators": [50, 100, 150, 200],
    "min_child_weight": [1, 3, 5],
    "subsample": [0.8, 0.9, 1.0],
    "colsample_bytree": [0.8, 0.9, 1.0],
    "gamma": [0, 1, 5]
}

xgbclf= XGBClassifier()
xgb_clf = RandomizedSearchCV(xgbclf, xgbparams, cv = 5, verbose=True, n_jobs=2)
xgb_clf.fit(x_max_under_train_scaled,y_train_under_shuffled)
print(xgb_clf.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'subsample': 0.8, 'n_estimators': 50, 'min_child_weight': 5, 'max_depth': 7,
'learning_rate': 0.2, 'gamma': 0, 'colsample_bytree': 0.9}
```

[67]:

```
xgb = XGBClassifier(subsample= 0.8, n_estimators= 50, min_child_weight= 5,␣
 ↪max_depth= 7,
                        learning_rate=0.2, gamma= 0, colsample_bytree= 0.9)
xgb.fit(x_max_under_train_scaled, y_train_under_shuffled)
xgboost_pred = xgb.predict(x_max_under_test_scaled)

ConfusionMatrixDisplay.from_estimator(xgb,x_max_under_test_scaled, y_test)
plt.show()
print("Accuracy Score is " , (accuracy_score(y_test, xgboost_pred)*100))
xgboost_roc_auc = roc_auc_score(y_test, xgboost_pred)
print(classification_report(y_test, xgboost_pred))
```




```
Accuracy Score is  86.8213228035538
              precision    recall  f1-score   support

           0       0.56      0.83      0.67       327
           1       0.96      0.88      0.92      1699

    accuracy                           0.87      2026
   macro avg       0.76      0.85      0.79      2026
weighted avg       0.90      0.87      0.88      2026
```

Here after undersampling the data recall rate got improved by the XGboost

14 weighted Average

Source : Online

A weighted average is an average in which each value in the dataset is assigned a weight according to its importance or frequency, making some values contribute more to the average than others. The weighted average is calculated by multiplying each value by its corresponding weight, summing the results, and then dividing the sum by the total of the weights.

15 Conclusions

1. Total transaction count, balance and No. of months inactive in the last 12 months features are giving more weight so the bank can monitor these features to prevent churn rate
2. Random Forest, XGboost classifier trained on the training data was able to achieve 93% weighted average precision and 93% recall rate. Even though model is predicting well, we can still improve recall rate on churn customers. Either by collecting the more quality data for churn customers, or we can dig deeper into other variables like total tansaction count by each quarter.
3. Here we need to consider F1score. f1 score is harmonic mean of the precision and recall. It represents both precision and recall in one metric. Recall would look for false negative which basically means churn customers.
4. Now the bank can able to use this models to predict the churn customers. Teh solution can help to identify which customers are more likely to leave the banl in the future.

```
[ ]:
```