Academia de Studii Economice, București
Facultatea de Cibernetică, Statistică și Informatică Economică
Specializarea: Informatică Economică

Practical Stage Project

Data Science – Machine Learning

Coordinator: Doinea Mihai

Student: **Rață Ștefania-Victoria**

București

2023

Table of contents

• Introduction

My practical stage was organized by Tap That Job project, 7th edition, which was hosted by the Cybernetics Students' Union in collaboration with BCR - Banca Comerciala Romana, one of the leading banks in Romania.

Tap That Job project aims to help students gain more insights about their field of interest and other opportunities on the labor market, which can improve the chances of further employment for the students within the partner companies.

In order to be accepted, I had to go through a selection process: sending my resume, supporting the technical interview that focused on programming principles such as Object Oriented or SQL and supporting the interview with the representatives of the HR department.

I enjoyed the courses held by BCR because of the fact that I was able to demonstrate my skills in programming, but also my knowledge related to statistics, finances and accounting. Nevertheless, it was an interesting experience because I got more insights about Data Science, I learned how Machine Learning actually works and I explored its applications in business.

It was not my first encounter with Python, as I studied Evolutive Programming and Genetic Algorithms as a subject at faculty, so it was not challenging to support my practical stage project using this programming language.

To sum up, my practical stage was really enjoyable, as I have met some wonderful people, and I have obtained some valuable skills which I am confident will lead to the further development of my career as a programmer.

- ## General presentation of the society

Banca Comercială Română (BCR), a member of Erste Group, is one of the most important financial groups in Romania, including universal banking operations (retail, corporate & investment banking, treasury and capital markets), as well as profile companies on the market of leasing, private pensions and housing banks.

I. Data Preparation and Analysis

A. Problem Statement

Delta Bank is a company that aims to develop a customer experience vision. Part of this vision, one important pillar is the customers retention. As a result, the business officers would want to strengthen the relationship with the clients for keeping them engaged and assessing their needs and pain points. Due to the costs involved, it is not possible to target all the clients of the company. Along with the marketing team, the officers would like to understand why customers are leaving the bank and find the most probable churners. The dataset consists in a list of the customers of this bank. As variables of interest, there are the credit limit, income category, age, attrition flag etc. The target is to identify the customers that are most likely leave the company and the drivers that determine their churn.

B. Import libraries

In [3]:
```
import numpy as np #for linear algebra
import pandas as pd #for data processing, dataset reading etc
import matplotlib.pyplot as plt #for plotting
import seaborn as sns #for various plot design
import missingno as msno #for outliers, plots
```

C. Read dataset

In [32]:
```
path = "C:\\Users\\user\\OneDrive\\Desktop\\TTJ\\TTJ project\\dataset\\dataset.csv"
```

In [33]:
```
data = pd.read_csv(path)
```

D. Exploratory data analysis

1. General

a) Visualize the dimensions of the dataset

```
In [5]: data.shape
```

```
Out[5]: (10127, 21)
```

b) Preview dataset

```
In [6]: data.head(10)
```

Out[6]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	...
0	768805383	Existing Customer	45.0	M	3.0	High School	Married	$60K-80K$	Blue	39	...
1	818770008	Existing Customer	49.0	F	5.0	Graduate	Single	Less than $40K	Blue	44	...
2	713982108	Existing Customer	51.0	M	3.0	Graduate	Married	$80K-120K$	Blue	36	...
3	769911858	Existing Customer	40.0	F	4.0	High School	NaN	Less than $40K	Blue	34	...
4	709106358	Existing Customer	40.0	M	3.0	Uneducated	Married	$60K-80K$	Blue	21	...
5	713061558	Existing Customer	44.0	M	2.0	Graduate	Married	$40K-60K$	Blue	36	...
6	810347208	Existing Customer	51.0	M	4.0	NaN	Married	$120K +	Gold	46	...
7	818906208	Existing Customer	32.0	M	0.0	High School	NaN	$60K-80K$	Silver	27	...
8	710930508	Existing Customer	37.0	M	3.0	Uneducated	Single	$60K-80K$	Blue	36	...
9	719661558	Existing Customer	48.0	M	2.0	Graduate	Single	$80K-120K$	Blue	36	...

10 rows × 21 columns

c) View column names

```
In [7]: print(data.columns)
```

```
Index(['CLIENTNUM', '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_Used_Bal',
       'Total_Unused_Bal', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio'],
      dtype='object')
```

d) View the types of the columns

```
In [8]: data.dtypes
```

```
Out[8]: CLIENTNUM                     int64
        Attrition_Flag               object
        Customer_Age                float64
        Gender                       object
        Dependent_count             float64
        Education_Level              object
        Marital_Status               object
        Income_Category              object
        Card_Category                object
        Months_on_book                int64
        Total_Relationship_Count      int64
        Months_Inactive_12_mon        int64
        Contacts_Count_12_mon         int64
        Credit_Limit                float64
        Total_Used_Bal                int64
        Total_Unused_Bal            float64
        Total_Amt_Chng_Q4_Q1        float64
        Total_Trans_Amt               int64
        Total_Trans_Ct                int64
        Total_Ct_Chng_Q4_Q1         float64
        Avg_Utilization_Ratio       float64
        dtype: object
```

e) View information about the dataset

```
In [9]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   CLIENTNUM                 10127 non-null  int64  
 1   Attrition_Flag            10127 non-null  object 
 2   Customer_Age              10124 non-null  float64
 3   Gender                    10127 non-null  object 
 4   Dependent_count           10122 non-null  float64
 5   Education_Level           8608 non-null   object 
 6   Marital_Status            9378 non-null   object 
 7   Income_Category           10125 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_Used_Bal            10127 non-null  int64  
 15  Total_Unused_Bal          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(7), int64(8), object(6)
memory usage: 1.6+ MB
```

Observations:

1.There are some columns that have a large number of missing values('Marital_Status', 'Education_Level' etc).

2.As variables of interest there are numerical ones('Customer_Age', 'Months_on_book', 'Credit_Limit' etc), as well as categorical ones('Gender', 'Education Level', 'Marital Status'etc).

f) View statistical properties of the dataset

In [10]:
```python
data.describe()
```

Out[10]:

	CLIENTNUM	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_L
count	1.012700e+04	10124.000000	10122.000000	10127.000000	10127.000000	10127.000000	10127.000000	10127.00(

unique	2	2	6	3	6	4
top	Existing Customer	F	Graduate	Married	Less than $40K	Blue
freq	8500	5358	3128	4687	3560	9436

Observation:

Here, the categorical columns(data type = 'object') are being described by the proper statistical indicators(count, number of unique values, frequency and the most frequent value).

2. Univariate Anaalysis

The target variable('Attrition Flag') is intended to be analysed.

a) Check for missing values

In [14]:
```python
data['Attrition_Flag'].isnull()
```

Out[14]:
```
0        False
1        False
2        False
3        False
4        False
```

```
10122    False
10123    False
10124    False
10125    False
10126    False
Name: Attrition_Flag, Length: 10127, dtype: bool
```

In [15]:
```
data['Attrition_Flag'].isnull().sum()
```

Out[15]:
```
0
```

Observation:

We do not have missing values on the target variable column, so it does not need to be processed.

b) Visualize unique values

In [16]:
```
data['Attrition_Flag'].nunique()
```

Out[16]:
```
2
```

This variable has 2 unique possible values.

In [17]:
```
data['Attrition_Flag'].unique()
```

Out[17]:
```
array(['Existing Customer', 'Attrited Customer'], dtype=object)
```

The values of this variable are: 'Existing Customer' and 'Attrited Customer'.

c) View frequency of values and percentage of frequency

In [19]:
```
data['Attrition_Flag'].value_counts()
```

Out[19]:
```
Existing Customer    8500
Attrited Customer    1627
Name: Attrition_Flag, dtype: int64
```

There are 8500 existing customers and 1627 attrited customers.

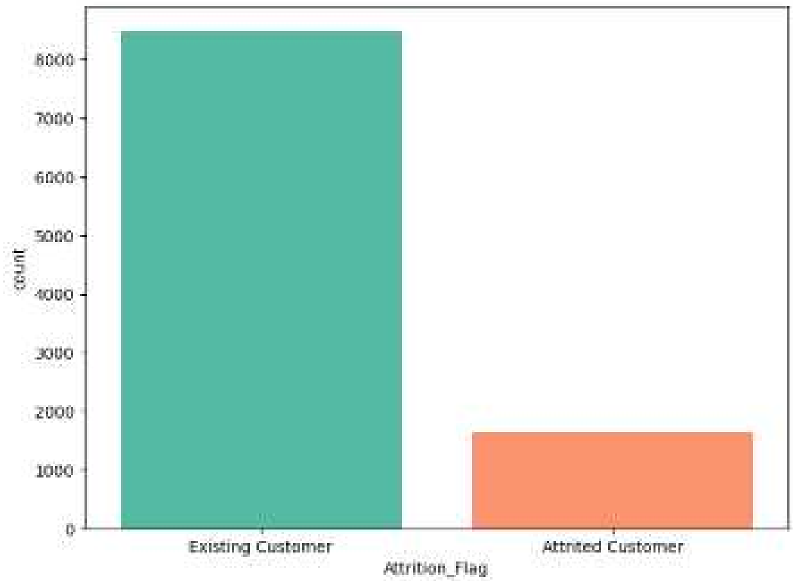
In [20]:
```
data['Attrition_Flag'].value_counts()/len(data)*100
```

Out[20]:
```
Existing Customer    83.934038
Attrited Customer    16.065962
Name: Attrition_Flag, dtype: float64
```

16% of the customers are attrited.

d) Visualize frequency distribution of the 'Attrition Flag' variable

In [23]:
```
fig, ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(data = data, x = 'Attrition_Flag', palette='Set2')
plt.show()
```



e) Change labels of the variable 'Attrition Flag' to numeric

In [34]:
```
data['Attrition_Flag'] = data['Attrition_Flag'].map({'Existing Customer':1, 'Attrited Customer':0})
```

In [35]:
```
data['Attrition_Flag'].value_counts()
```

Out[35]:
```
1    8500
0    1627
Name: Attrition_Flag, dtype: int64
```

The target variable shows the customers that have attrited(value = 0), and the ones who have not(value = 1).

Observation:

The target variable is independent.

3. Feature analysis

3.1 Categorical variables

a) Explore

Find the categorical variables.

In [26]:
```
data.dtypes
```

Out[26]:
```
CLIENTNUM                     int64
Attrition_Flag                int64
Customer_Age                float64
Gender                       object
Dependent_count             float64
Education_Level              object
Marital_Status               object
Income_Category              object
Card_Category                object
Months_on_book                int64
Total_Relationship_Count      int64
Months_Inactive_12_mon        int64
Contacts_Count_12_mon         int64
Credit_Limit                float64
Total_Used_Bal                int64
Total_Unused_Bal            float64
Total_Amt_Chng_Q4_Q1        float64
Total_Trans_Amt               int64
Total_Trans_Ct                int64
Total_Ct_Chng_Q4_Q1         float64
Avg_Utilization_Ratio       float64
dtype: object
```

The categorical variables have the type 'object', so they are selected.

In [36]:
```
categorical_columns = [col for col in data.columns if data[col].dtypes == 'object']
```

In [37]:
```
print('There are ', len(categorical_columns), ' categorical values. These are ', categorical_columns)
```

```
There are  5  categorical values. These are  ['Gender', 'Education_Level', 'Marital_Status', 'Income_Category', 'Card_Categor
y']
```

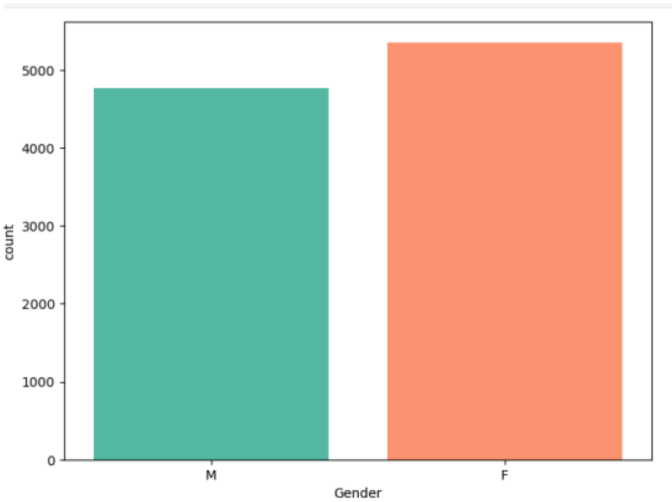
In [38]:
```
data[categorical_columns].head()
```

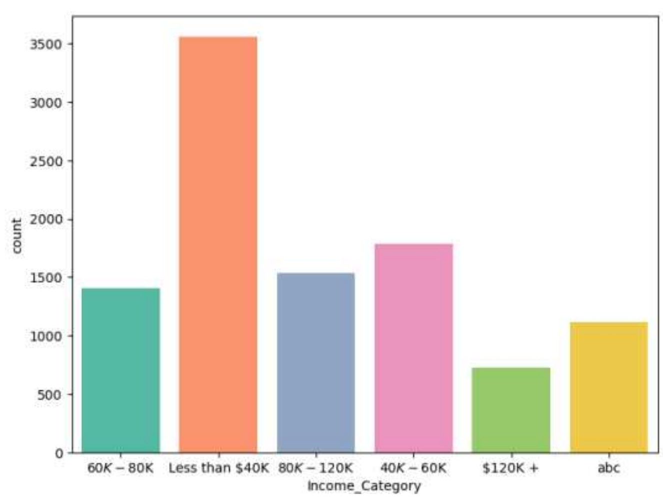
Out[38]:

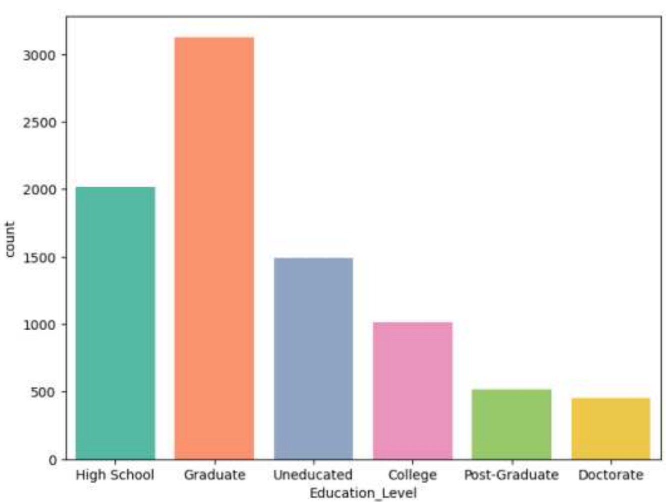
	Gender	Education_Level	Marital_Status	Income_Category	Card_Category
0	M	High School	Married	$60K-80$K	Blue
1	F	Graduate	Single	Less than $40K	Blue
2	M	Graduate	Married	$80K-120$K	Blue
3	F	High School	NaN	Less than $40K	Blue
4	M	Uneducated	Married	$60K-80$K	Blue

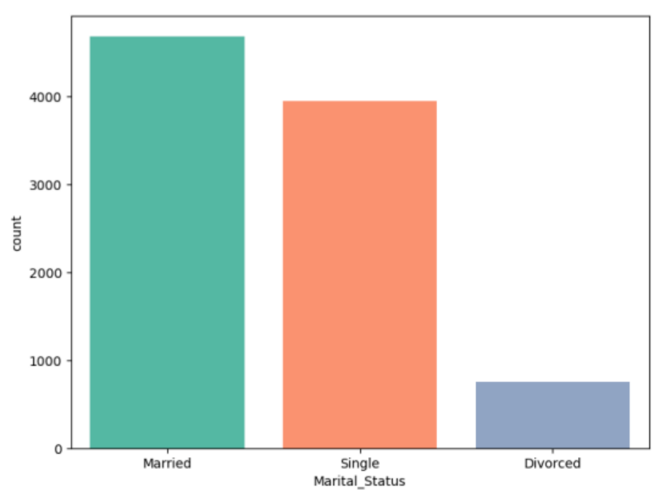
Check the categorical variables distribution in relationship with the target variable 'Attrition Flag'.

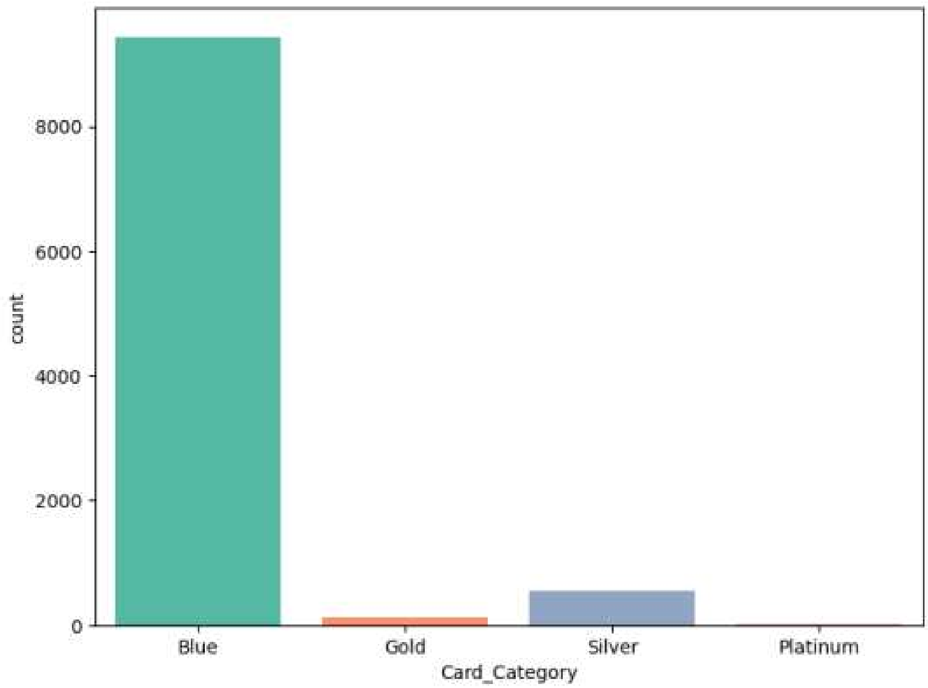
In [31]:
```
for col in categorical_columns:
    fig, ax = plt.subplots(figsize =(8,6))
    ax = sns.countplot(data = data, x = col, palette='Set2')
    plt.show()
```









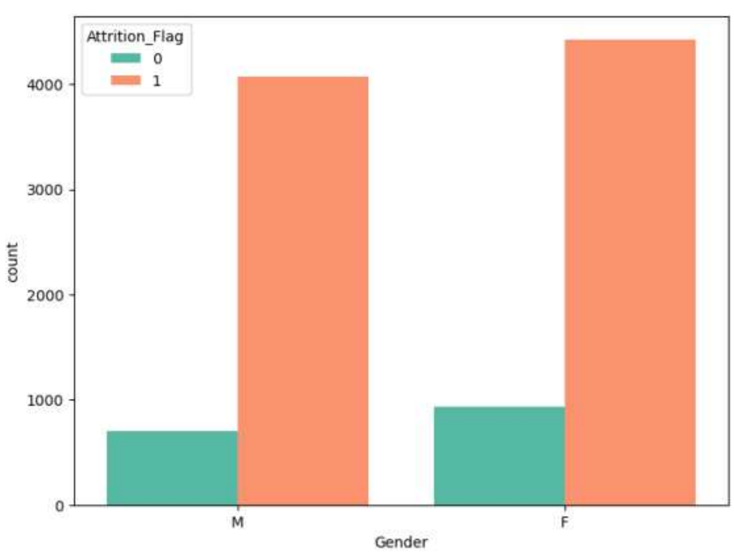


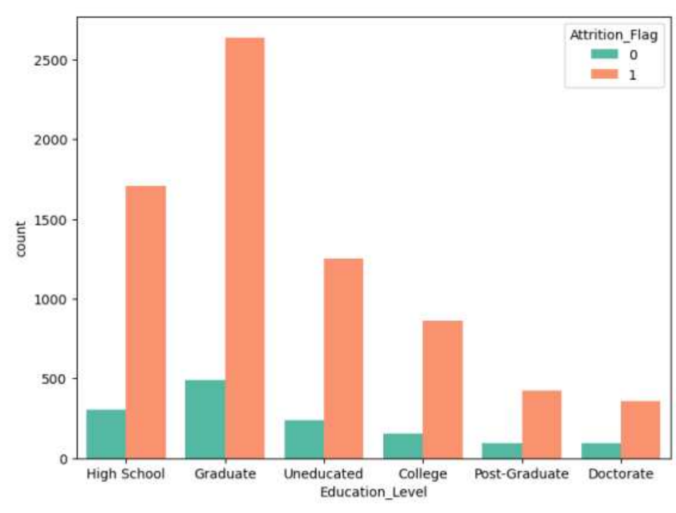
Observations:

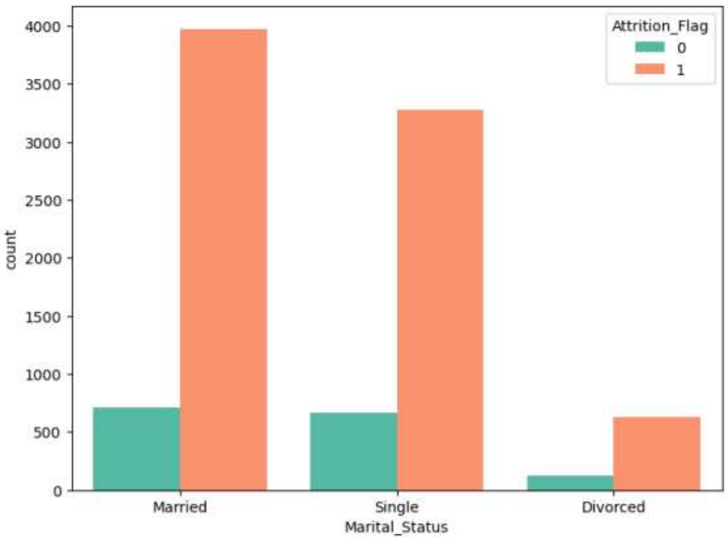
1. Most of the customers have graduated college or highschool.
2. Most of the customers are either married or single.
3. Most of the customers earn a low income: less than $40K.
4. Almost all of the customers own a Blue card.

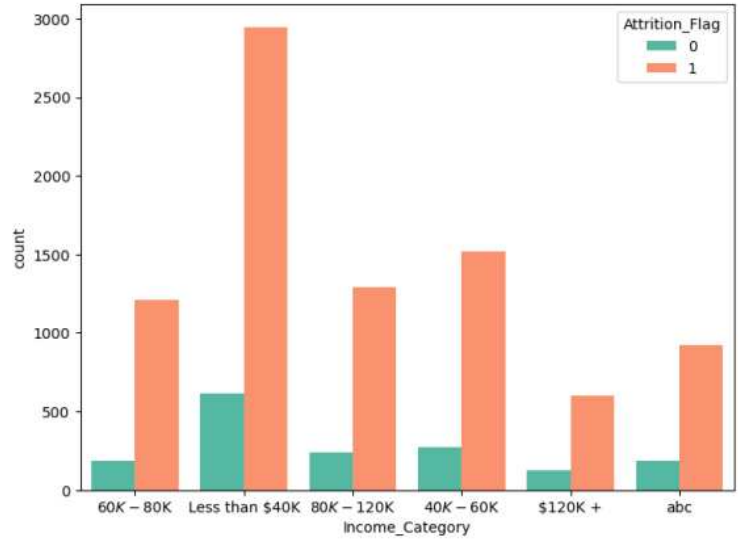
Emphasize the relationship between the target variable and the categorical variables.

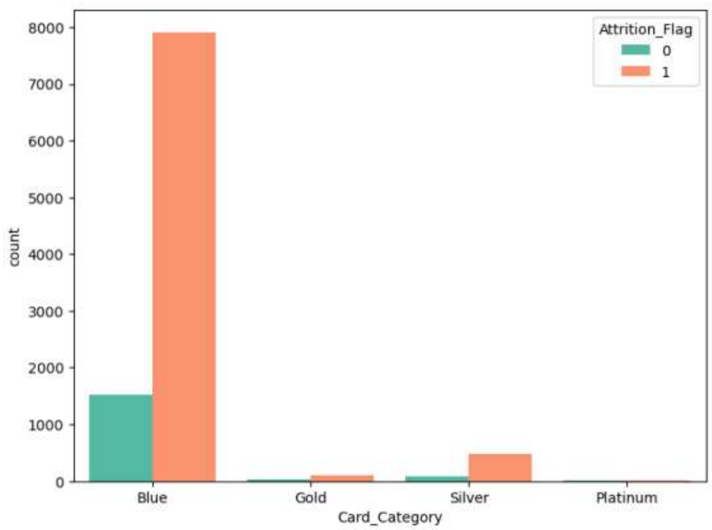
In [34]:
```
for col in categorical_columns:
    fig, ax = plt.subplots(figsize=(8,6))
    ax = sns.countplot(data=data, x=col,hue='Attrition_Flag',palette='Set2')
    plt.show()
```











Observations:

1.The customers who graduated college are more likely to attrit.

2.The married and single customers are almost equally likely to attrit, and more than the divordced ones.

3.The customers with a low income are very likely to attrit.

b) Missing value imputation

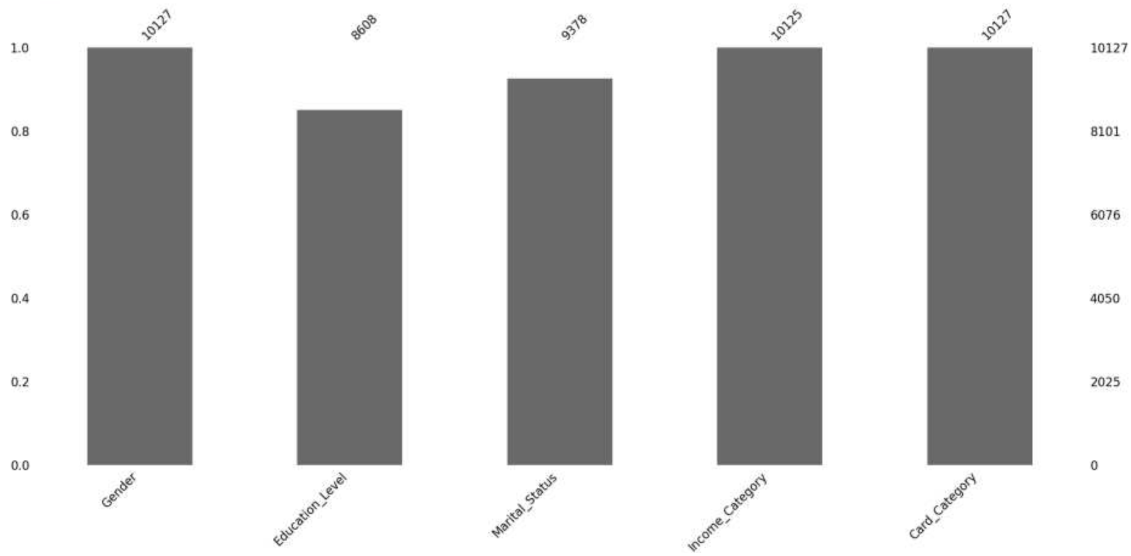
In [35]:
```
#Check which categorical variables have missing values
data[categorical_columns].isnull().sum()
```

Out[35]:
```
Gender                0
Education_Level    1519
Marital_Status      749
Income_Category       2
Card_Category         0
dtype: int64
```

The columns 'Education Level', 'Marital Status' and 'Income Category' have missing values.

In [36]:
```
#Visualize the missing values
msno.bar(data[categorical_columns])
```

Out[36]: <Axes: >



In [37]:
```
data['Education_Level'].value_counts()
```

Out[37]:
```
Graduate         3128
High School      2013
Uneducated       1487
College          1013
Post-Graduate     516
Doctorate         451
Name: Education_Level, dtype: int64
```

In [39]:
```
data['Education_Level'] = data['Education_Level'].fillna('Unknown')
```

In [40]:
```
data['Marital_Status'].value_counts()
```

Out[40]:
```
Married     4687
Single      3943
Divorced     748
Name: Marital_Status, dtype: int64
```

In [41]:
```
data['Marital_Status'] = data['Marital_Status'].fillna('Unknown')
```

Observation:

An Arbitrary Value Imputation has been performed for the columns 'Education Level' and 'Marital Status'.

In [42]:
```
data['Income_Category'].value_counts()
```

Out[42]:
```
Less than $40K    3560
$40K - $60K       1789
$80K - $120K      1535
$60K - $80K       1402
abc               1112
$120K +            727
Name: Income_Category, dtype: int64
```

In [43]:
```
import statistics
```

In [44]:
```
income_cat_mode = statistics.mode(data['Income_Category'])
print(income_cat_mode)
```

```
Less than $40K
```

In [45]:
```
data['Income_Category'] = data['Income_Category'].fillna(income_cat_mode)
```

Observation:

The missing values are replaced with the mode(the most frequent value). Since most of the customers have an income of less than 40K dollars and there are only 2 missing values, it is safe to assume that those are less than 40K dollars as well.

In [46]:
```
data[categorical_columns].isnull().sum()
```

Out[46]:
```
Gender             0
Education_Level    0
Marital_Status     0
Income_Category    0
Card_Category      0
dtype: int64
```

Now, there are no more missig values.

c) Feature variance

Check if there are columns with only 1 value.

In [47]:
```
data[categorical_columns].nunique()
```

Out[47]:
```
Gender             2
Education_Level    7
Marital_Status     4
Income_Category    6
Card_Category      4
dtype: int64
```

There are no categorical columns with only 1 value.

d) Categorical Encoding

Check for the number of different values for each column.

In [48]:
```
data[categorical_columns].nunique()
```

Out[48]:
```
Gender             2
Education_Level    7
Marital_Status     4
Income_Category    6
Card_Category      4
dtype: int64
```

The variable 'Card Category' should be checked because one value is very frequent and the others have a very low frequency.

In [49]:
```
data['Card_Category'].value_counts()/len(data)*100
```

Out[49]:
```
Blue        93.176656
Silver       5.480399
Gold         1.145453
Platinum     0.197492
Name: Card_Category, dtype: float64
```

Observation:

It can be stated that 93.18% own a Blue Card, 5.5% a Silver card and the rest are very low.

Rare Label Encoding

In [50]:
```
values = data['Card_Category'].value_counts()/len(data)*100
data['Card_Category'] = np.where(data['Card_Category'].isin(values.index[2:]),'Other Cards',data['Card_Category'])
```

In [51]:
```
data['Card_Category'].value_counts()/len(data)*100
```

Out[51]:
```
Blue           93.176656
Silver          5.480399
Other Cards     1.342945
Name: Card_Category, dtype: float64
```

3.2 Numerical Variables

a) Explore

In [52]:
```
numerical_columns = [col for col in data.columns if data[col].dtypes != 'object']
```

In [53]:
```
print('There are ', len(numerical_columns), ' numerical columns. These are: ', numerical_columns)
```

```
There are  16  numerical columns. These are:  ['CLIENTNUM', 'Attrition_Flag', 'Customer_Age', 'Dependent_count', 'Months_on_book', 'Total_Relations
hip_Count', 'Months_Inactive_12_mon', 'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Used_Bal', 'Total_Unused_Bal', 'Total_Amt_Chng_Q4_Q1', 'Total
_Trans_Amt', 'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio']
```

Observation:

The target variable does not have to be analysed because it will be predicted, so it is removed.

In [54]:
```
numerical_columns = [col for col in data.columns if data[col].dtypes != 'object' and col != 'Attrition_Flag']
```

In [55]:
```
print('There are ', len(numerical_columns), ' numerical columns. These are: ', numerical_columns)
```

```
There are  15  numerical columns. These are:  ['CLIENTNUM', 'Customer_Age', 'Dependent_count', 'Months_on_book', 'Total_Relationship_Count', 'Month
s_Inactive_12_mon', 'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Used_Bal', 'Total_Unused_Bal', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt', 'Tota
l_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio']
```

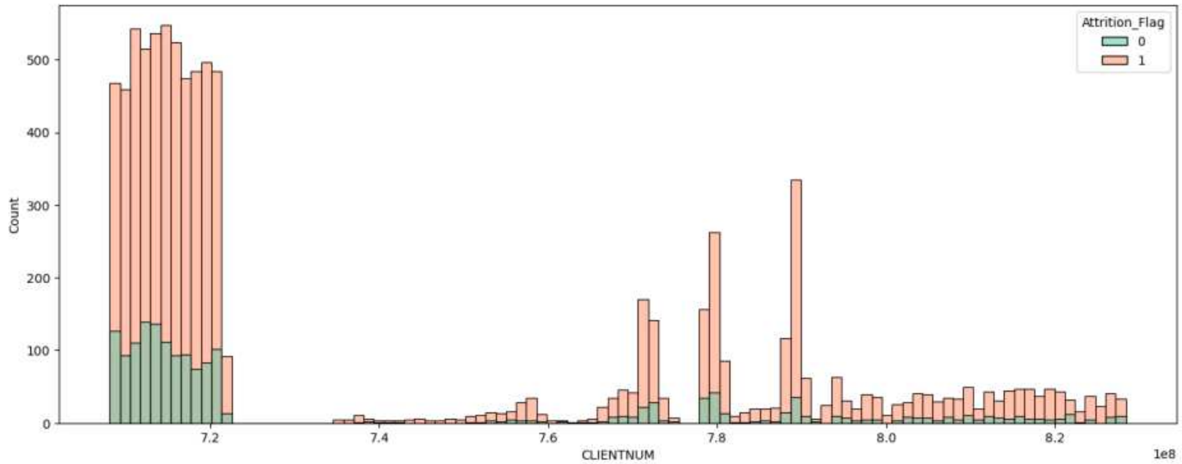
In [56]:
```
data[numerical_columns].head()
```

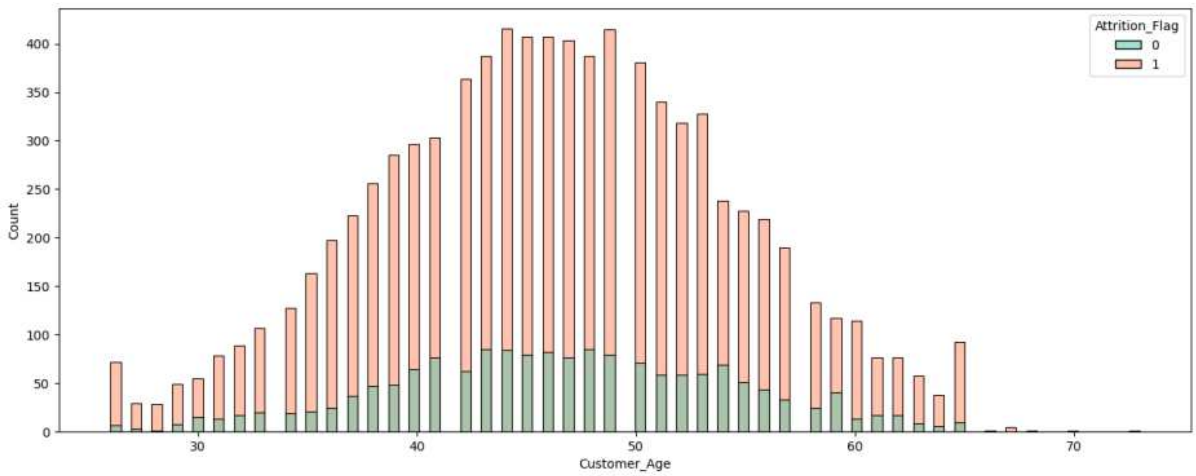
Out[56]:

	CLIENTNUM	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_Limit	Total_Used_I
0	768805383	45.0	3.0	39	5	1	3	12691.0	7
1	818770008	49.0	5.0	44	6	1	2	8256.0	8
2	713982108	51.0	3.0	36	4	1	0	3418.0	
3	769911858	40.0	4.0	34	3	4	1	3313.0	25
4	709106358	40.0	3.0	21	5	1	0	4716.0	

Relationship between the target variable and the numerical ones:

In [66]:
```
for col in numerical_columns:
    fig, ax = plt.subplots(figsize = (16, 6))
    sns.histplot(data = data, x = col, hue = 'Attrition_Flag', bins = 100, palette='Set2')
    plt.show()
```





b) Missing values

In [67]:
```
data[numerical_columns].isnull().sum()
```

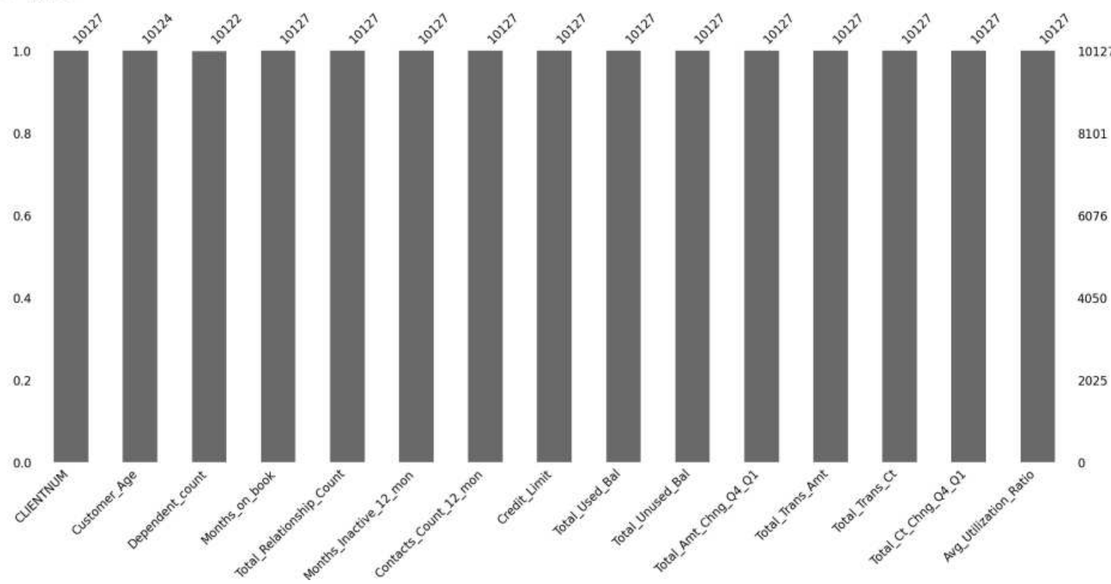
Out[67]:
```
CLIENTNUM                   0
Customer_Age                3
Dependent_count             5
Months_on_book              0
Total_Relationship_Count    0
Months_Inactive_12_mon      0
Contacts_Count_12_mon       0
Credit_Limit                0
Total_Used_Bal              0
Total_Unused_Bal            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 2 numerical variables that have missing values: Customer Age and Dependent count.

Visualize the missing values:

In [68]:
```
msno.bar(data[numerical_columns])
```

```
Out[68]: <Axes: >
```




```
In [69]: data['Customer_Age']
```

```
Out[69]: 0        45.0
         1        49.0
         2        51.0
         3        40.0
         4        40.0
                  ...
         10122    50.0
         10123    41.0
         10124    44.0
         10125    30.0
         10126    43.0
         Name: Customer_Age, Length: 10127, dtype: float64
```

Imputing the missing values by replacing them with the mean:

```
In [57]: data['Customer_Age'] = data['Customer_Age'].fillna(data['Customer_Age'].mean())
```

```
In [58]: data['Customer_Age'].isnull().sum()
```

```
Out[58]: 0
```

There are no more missing values.

```
In [59]: data['Dependent_count']
```

```
Out[59]: 0        3.0
         1        5.0
         2        3.0
         3        4.0
         4        3.0
                  ...
         10122    2.0
         10123    2.0
         10124    1.0
         10125    2.0
         10126    2.0
         Name: Dependent_count, Length: 10127, dtype: float64
```

```
In [60]: data['Dependent_count'] = data['Dependent_count'].fillna(data['Dependent_count'].mean())
```

```
In [61]: data['Dependent_count'].isnull().sum()
```

```
Out[61]: 0
```

c) Feature variance

Check for constant columns:

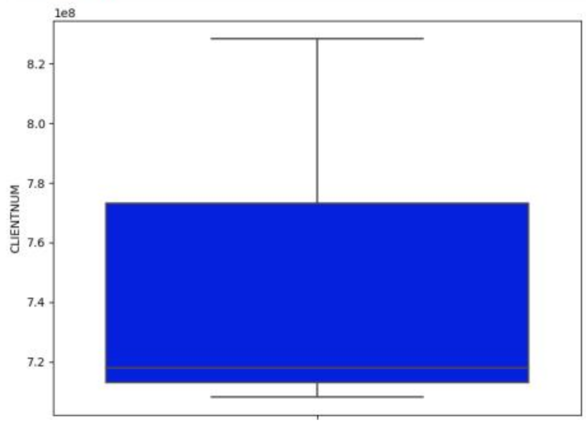
```
In [75]: data[numerical_columns].nunique()
```

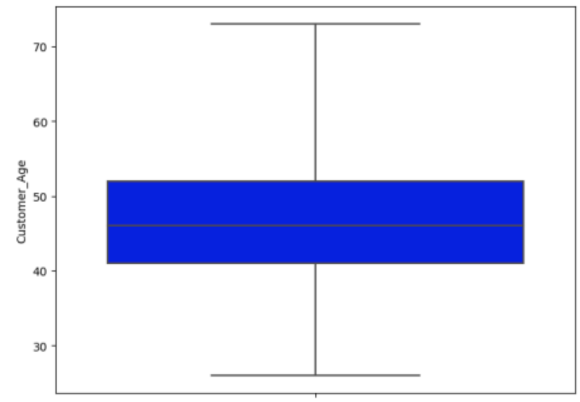
```
Out[75]: CLIENTNUM                   10127
         Customer_Age                   46
         Dependent_count                 7
         Months_on_book                 44
         Total_Relationship_Count        6
         Months_Inactive_12_mon          7
         Contacts_Count_12_mon           7
         Credit_Limit                 6205
         Total_Used_Bal               1974
         Total_Unused_Bal             6813
         Total_Amt_Chng_Q4_Q1         1158
         Total_Trans_Amt              5033
         Total_Trans_Ct                126
         Total_Ct_Chng_Q4_Q1           830
         Avg_Utilization_Ratio         964
         dtype: int64
```

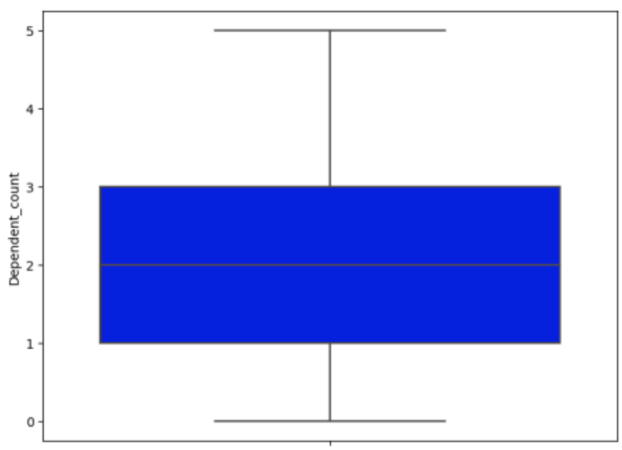
There are no columns that have constant values.

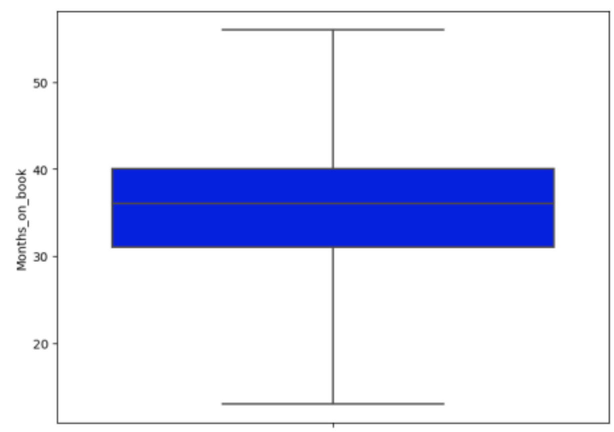
Check for outliers for each columns:

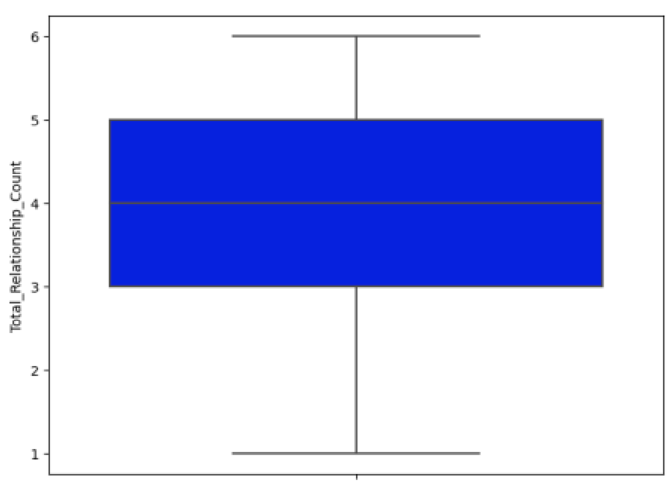
In [76]:
```python
for col in numerical_columns:
    fig, ax = plt.subplots(figsize=(8,6))
    sns.boxplot(y = data[col], color = 'blue', whis = 3)
    plt.show()
```

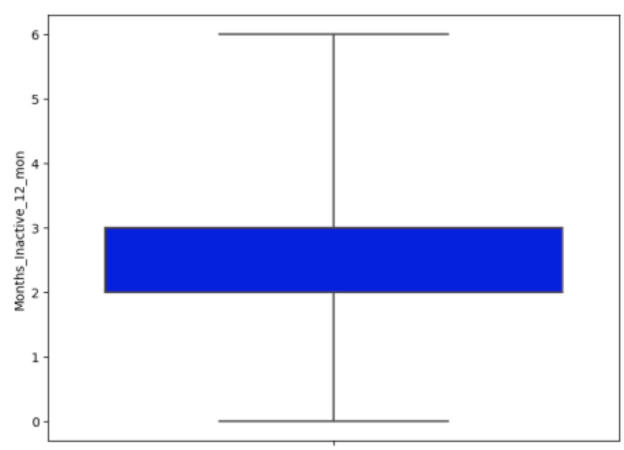












After analyzing the graphical representations, the variables heavily affected by outliers are identified.

In [62]:
```python
heavily_affected_by_outliers =['Credit_Limit', 'Total_Unused_Bal', 'Avg_Utilization_Ratio']
```

The outliers are eliminated only for the variables which are affected the most by them.

In [63]:
```python
def censoring_outliers(dataframe,column):
    q1 = dataframe[column].quantile(0.25)
    q3 = dataframe[column].quantile(0.75)
    IQR = q3 - q1
    lower_limit = q1-3*IQR
    upper_limit= q3+3*IQR
    dataframe[column]=np.where(dataframe[column] < lower_limit, lower_limit, np.where(dataframe[column] > upper_limit, upper_limit, dataframe[colu
```

In [64]:
```python
for variable in heavily_affected_by_outliers:
    censoring_outliers(data, variable)
```

e) Correlation

The Pearson correlation coefficient is being used because the variables are numerical.

In [65]:
```python
correlation = data[numerical_columns].corr()
```

In [66]:
```python
correlation
```

The Pearson correlation coefficient is being used because the variables are numerical.

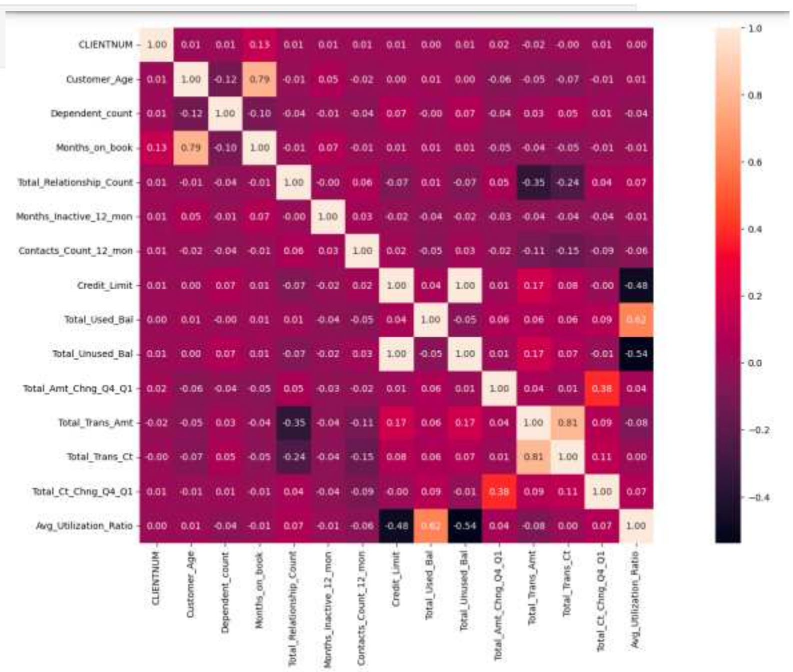
In [65]:
```python
correlation = data[numerical_columns].corr()
```

In [66]:
```python
correlation
```


Out[66]:

	CLIENTNUM	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Cr
CLIENTNUM	1.000000	0.007561	0.007049	0.134588	0.006907	0.005729	0.005694	
Customer_Age	0.007561	1.000000	-0.122321	0.788890	-0.010925	0.054321	-0.018468	
Dependent_count	0.007049	-0.122321	1.000000	-0.103255	-0.039189	-0.011408	-0.040632	
Months_on_book	0.134588	0.788890	-0.103255	1.000000	-0.009203	0.074164	-0.010774	
Total_Relationship_Count	0.006907	-0.010925	-0.039189	-0.009203	1.000000	-0.003675	0.055203	
Months_Inactive_12_mon	0.005729	0.054321	-0.011408	0.074164	-0.003675	1.000000	0.029493	
Contacts_Count_12_mon	0.005694	-0.018468	-0.040632	-0.010774	0.055203	0.029493	1.000000	
Credit_Limit	0.005708	0.002561	0.068150	0.007507	-0.071386	-0.020394	0.020817	
Total_Used_Bal	0.000825	0.014733	-0.002710	0.008623	0.013726	-0.042210	-0.053913	
Total_Unused_Bal	0.005633	0.001239	0.068379	0.006732	-0.072601	-0.016605	0.025646	
Total_Amt_Chng_Q4_Q1	0.017369	-0.062011	-0.035095	-0.048959	0.050119	-0.032247	-0.024445	
Total_Trans_Amt	-0.019692	-0.046455	0.025398	-0.038591	-0.347229	-0.036982	-0.112774	
Total_Trans_Ct	-0.002961	-0.067107	0.050590	-0.049819	-0.241891	-0.042787	-0.152213	
Total_Ct_Chng_Q4_Q1	0.007696	-0.012068	0.011016	-0.014072	0.040831	-0.038989	-0.094997	
Avg_Utilization_Ratio	0.000266	0.007023	-0.037063	-0.007541	0.067663	-0.007503	-0.055471	

In [84]:
```python
fig, ax = plt.subplots(figsize=(24,10))
sns.heatmap(correlation, annot = True, square=True, fmt='.2f')
plt.show()
```



Observation:

Variables that have a very high correlation coefficient, usually above 75%, are considered duplicates, so one of them has to be eliminated.

The variables that can be considered duplicates are 'Credit_Limit' and 'Total_Unused_Bal', respectively 'Total_Trans_Amt' and 'Total_Trans_Ct'.

In [67]:
```
columns_to_drop = ['Total_Unused_Bal', 'Total_Trans_Ct']
```

In [68]:
```
numerical_columns.remove('Total_Unused_Bal')
numerical_columns.remove('Total_Trans_Ct')
```

In [69]:
```
numerical_columns
```

Out[69]:
```
['CLIENTNUM',
 'Customer_Age',
 'Dependent_count',
 'Months_on_book',
 'Total_Relationship_Count',
 'Months_Inactive_12_mon',
 'Contacts_Count_12_mon',
 'Credit_Limit',
 'Total_Used_Bal',
 'Total_Amt_Chng_Q4_Q1',
 'Total_Trans_Amt',
 'Total_Ct_Chng_Q4_Q1',
 'Avg_Utilization_Ratio']
```

E. Final Dataset

In [70]:
```
print(columns_to_drop)
```

```
['Total_Unused_Bal', 'Total_Trans_Ct']
```

The columns are eliminated from the final dataset.

In [71]:
```
data = data.drop(columns = columns_to_drop)
```

In [72]:
```
data
```

Out[72]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Re
0	768805383	1	45.0	M	3.0	High School	Married	$60K - 80K$	Blue	39	
1	818770008	1	49.0	F	5.0	Graduate	Single	Less than $40K	Blue	44	
2	713982108	1	51.0	M	3.0	Graduate	Married	$80K - 120K$	Blue	36	
3	769911858	1	40.0	F	4.0	High School	Unknown	Less than $40K	Blue	34	
4	709106358	1	40.0	M	3.0	Uneducated	Married	$60K - 80K$	Blue	21	
...	...	...	...	...	...	...	...	...	...	...	
10122	772366833	1	50.0	M	2.0	Graduate	Single	$40K - 60K$	Blue	40	
10123	710638233	0	41.0	M	2.0	Unknown	Divorced	$40K - 60K$	Blue	25	
10124	716506083	0	44.0	F	1.0	High School	Married	Less than $40K	Blue	36	
10125	717406983	0	30.0	M	2.0	Graduate	Unknown	$40K - 60K$	Blue	36	
10126	714337233	0	43.0	F	2.0	Graduate	Married	Less than $40K	Silver	25	

10127 rows × 19 columns

Save the changes in the dataset to the file using the path.

In [73]:
```
data.to_csv(path, index=False)
```

II. MODEL DEVELOPMENT

A. Import libraries

In [46]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, roc_auc_score, roc_curve
```

B. Read the dataset

In [28]:
```
data=pd.read_csv("C:\\Users\\user\\OneDrive\\Desktop\\TTJ\\TTJ project\\dataset\\dataset.csv")
```

In [29]:
```
data.head()
```

Out[29]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Relatio
0	768805383	1	45.0	M	3.0	High School	Married	60K−80K	Blue	39	
1	818770008	1	49.0	F	5.0	Graduate	Single	Less than $40K	Blue	44	
2	713982108	1	51.0	M	3.0	Graduate	Married	80K−120K	Blue	36	
3	769911858	1	40.0	F	4.0	High School	Unknown	Less than $40K	Blue	34	
4	709106358	1	40.0	M	3.0	Uneducated	Married	60K−80K	Blue	21	

In [30]:
```
data.isnull()
```

Out[30]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Re
0	False	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	
10122	False	False	False	False	False	False	False	False	False	False	
10123	False	False	False	False	False	False	False	False	False	False	
10124	False	False	False	False	False	False	False	False	False	False	
10125	False	False	False	False	False	False	False	False	False	False	
10126	False	False	False	False	False	False	False	False	False	False	

C. Classification

1. Declare independent variables and target value

Observation:

The target variable is the dependent one, in this case - 'Attrition Flag'

In [64]:
```
#Independent variables:
X = data.drop(columns = ['Attrition_Flag'])

#Target variable:
y = data['Attrition_Flag']
```

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

Out[65]:

	CLIENTNUM	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Relationship_Count	I
0	768805383	45.0	M	3.0	High School	Married	60K−80K	Blue	39	5	
1	818770008	49.0	F	5.0	Graduate	Single	Less than $40K	Blue	44	6	
2	713982108	51.0	M	3.0	Graduate	Married	80K−120K	Blue	36	4	
3	769911858	40.0	F	4.0	High School	Unknown	Less than $40K	Blue	34	3	
4	709106358	40.0	M	3.0	Uneducated	Married	60K−80K	Blue	21	5	

2. Data preprocessing

a) Unnecessary data elimination

The variable 'CLIENTNUM' signifies the ID, which is irrelevant to this analysis, so it is removed.

In [66]:
```
X = X.drop(columns = ['CLIENTNUM'])
```

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

Out[67]:

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Relationship_Count	Months_Inactiv
0	45.0	M	3.0	High School	Married	60K−80K	Blue	39	5	
1	49.0	F	5.0	Graduate	Single	Less than $40K	Blue	44	6	
2	51.0	M	3.0	Graduate	Married	80K−120K	Blue	36	4	
3	40.0	F	4.0	High School	Unknown	Less than $40K	Blue	34	3	
4	40.0	M	3.0	Uneducated	Married	60K−80K	Blue	21	5	

b) Categorical encoding - One Hot Encoding

This type of encoding is used in order to transform the categorical variables based on the value of truth they hold for each customer. This operation helps because the algorithms can not be applied on categorical variables.

```
In [68]: categorical_columns = [col for col in X.columns if X[col].dtypes== 'object']
```

```
In [69]: categorical_columns
```

```
Out[69]: ['Gender',
          'Education_Level',
          'Marital_Status',
          'Income_Category',
          'Card_Category']
```

```
In [70]: X = pd.get_dummies(X, columns = categorical_columns)
```

```
In [71]: X.head()
```

Out[71]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_Limit	Total_Used_Bal	Total_Am
0	45.0	3.0	39	5	1	3	12691.0	777	
1	49.0	5.0	44	6	1	2	8256.0	864	
2	51.0	3.0	36	4	1	0	3418.0	0	
3	40.0	4.0	34	3	4	1	3313.0	2517	
4	40.0	3.0	21	5	1	0	4716.0	0	

5 rows × 34 columns

3. Split the data into train and test sets

```
In [72]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 10)
```

```
In [73]: X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

4. Random Forest

4.1 Train Algorithm

```
In [16]: help(RandomForestClassifier)
```

```
Help on class RandomForestClassifier in module sklearn.ensemble._forest:

class RandomForestClassifier(ForestClassifier)
 |  RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf
=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose
=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
 |
 |  A random forest classifier.
 |
 |  A random forest is a meta estimator that fits a number of decision tree
 |  classifiers on various sub-samples of the dataset and uses averaging to
 |  improve the predictive accuracy and control over-fitting.
 |  The sub-sample size is controlled with the `max_samples` parameter if
 |  `bootstrap=True` (default), otherwise the whole dataset is used to build
 |  each tree.
 |
 |  Read more in the :ref:`User Guide <forest>`.
 |
 |  Parameters
 |  ----------
 |  n_estimators : int, default=100
 |      The number of trees in the forest.
 |
 |      .. versionchanged:: 0.22
 |         The default value of ``n_estimators`` changed from 10 to 100
 |         in 0.22.
 |
 |  criterion : {"gini", "entropy", "log_loss"}, default="gini"
 |      The function to measure the quality of a split. Supported criteria are
 |      "gini" for the Gini impurity and "log_loss" and "entropy" both for the
 |      Shannon information gain, see :ref:`tree_mathematical_formulation`.
 |      Note: This parameter is tree-specific.
```

```
 |  max_depth : int, default=None
 |      The maximum depth of the tree. If None, then nodes are expanded until
 |      all leaves are pure or until all leaves contain less than
 |      min_samples_split samples.
 |
 |  min_samples_split : int or float, default=2
 |      The minimum number of samples required to split an internal node:
 |
 |      - If int, then consider `min_samples_split` as the minimum number.
 |      - If float, then `min_samples_split` is a fraction and
 |        `ceil(min_samples_split * n_samples)` are the minimum
 |        number of samples for each split.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  min_samples_leaf : int or float, default=1
 |      The minimum number of samples required to be at a leaf node.
 |      A split point at any depth will only be considered if it leaves at
 |      least ``min_samples_leaf`` training samples in each of the left and
 |      right branches.  This may have the effect of smoothing the model,
 |      especially in regression.
 |
 |      - If int, then consider `min_samples_leaf` as the minimum number.
 |      - If float, then `min_samples_leaf` is a fraction and
 |        `ceil(min_samples_leaf * n_samples)` are the minimum
 |        number of samples for each node.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  min_weight_fraction_leaf : float, default=0.0
 |      The minimum weighted fraction of the sum total of weights (of all
 |      the input samples) required to be at a leaf node. Samples have
 |      equal weight when sample_weight is not provided.
 |
 |  max_features : {"sqrt", "log2", None}, int or float, default="sqrt"
 |      The number of features to consider when looking for the best split:
```

Instantiate the model

```
In [41]: rf = RandomForestClassifier(n_estimators = 200, max_depth = 4, n_jobs = -1)
```

Observation:

n_estimators represents the number of trees and the n_jobs parameter requires the full capacity of the processor.

```
In [42]: rf.fit(X_train, y_train)
```

```
Out[42]: ▾ RandomForestClassifier
RandomForestClassifier(max_depth=4, n_estimators=200, n_jobs=-1)
```

4.2 Predict Results

```
In [43]: y_predict = rf.predict(X_test)
```

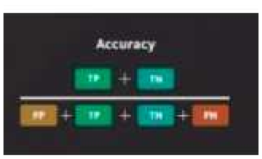
```
In [44]: print(y_predict)
```

```
[1 1 1 ... 0 1 1]
```

4.3 Performance metrics

a) Accuracy score

The accuracy score measures the number of correct classification predictions across both classes out of all predictions made.



In [21]:
```
accuracy = accuracy_score(y_test, y_predict)
print(accuracy)
```

```
0.8790720631786771
```

Observation:

88% of the observations are well predicted.

b) Confusion Matrix

The Confusion Matrix uses a Frequency Table to compare what it was predicted to the actual value.

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

In [22]:
```
cm = confusion_matrix(y_test, y_predict)
```

In [25]:
```
fig, ax = plt.subplots(figsize =(12,8))
sns.heatmap(cm, annot=True,fmt='d')
plt.show()
```

Observation:

It can be concluded that 9 observations were False Negative and 236 were False Positive.

c) Precision, Recall

The Precision, or Positive Predicted Value signifies the proportion of actual positive values out of all predicted positive values.

The Recall, or Sensitivity represents the number of predicted positive values out of all positive values.

In [47]:
```
precision = precision_score(y_test, y_predict)
recall = recall_score(y_test, y_predict)
print("Precision: ", precision)
print("Recall: ", recall)
```

```
Precision:  0.8746776689014956
Recall:  0.9947214076246335
```

Observation:

Out of all positive predicted values, 87% were actually positive. Out of all values that were actually positive, 99% were predicted corectly.

This algorithm seems to be suitable.

d) AUC score

The ROC (Receiver Operating Characteristics) is a probability curve that measures how well the model predicts the values of the target variable.

The AUC (Area Under the Curve) can be described as the degree as separability, namely how well the model can distinguish between classes. The higher the AUC score, the better the capacity of the model to predict positive values as positive and negative values as negative.

The Random Classifier is represented as a line that form a 45 degree angle with Ox and Oy axes on the graphical representation. It represents an AUC score of 0.5, which is equivalent with flipping a coin in order to predict the values. It is used to make a comparison to the ROC curve, therefore an AUC score closer to 0.5 means that the model is not very well trained to predict the correct values.

The Perfect Classifier has the value 1, which can be translated as 'the perfect model that can predict all the values accurately'.

In practice, an AUC greater than 0.7 means that the model is suitable.

In [49]:
```
auc_score = roc_auc_score(y_test, y_predict)
print('AUC score: ', auc_score)
```

```
AUC score:  0.6188560309151205
```

In [51]:
```
fpr, tpr, thershold = roc_curve(y_test, y_predict)
plt.plot(fpr, tpr)
```

Out[51]:
```
[<matplotlib.lines.Line2D at 0x240495b26e0>]
```

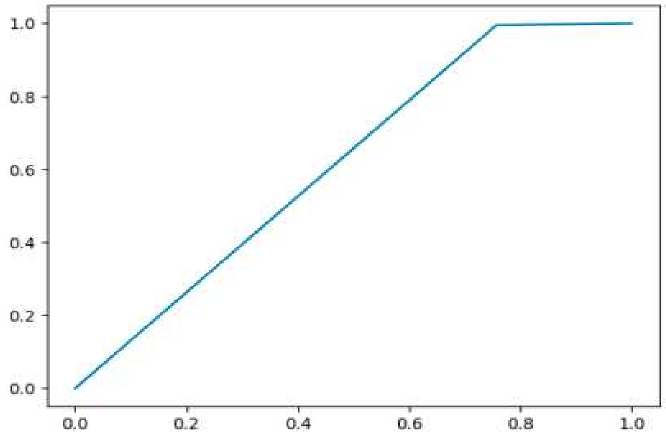
In practice, an AUC greater than 0.7 means that the model is suitable.

In [49]:
```
auc_score = roc_auc_score(y_test, y_predict)
print('AUC score: ', auc_score)
```

```
AUC score:  0.6188560309151205
```

In [51]:
```
fpr, tpr, thershold = roc_curve(y_test, y_predict)
plt.plot(fpr, tpr)
```

Out[51]:
```
[<matplotlib.lines.Line2D at 0x240495b26e0>]
```



This AUC score is not very high, so the model can not predict well the values.

In conclusion, even if other indicators have high values, the Random Forest model is not suitable for this dataset.¶

5.XGBoost (eXtreme Gradient Boosting)

5.1 Train Algorithm

In [52]:
```
help(XGBClassifier)
```

```
Help on class XGBClassifier in module xgboost.sklearn:

class XGBClassifier(XGBModel, sklearn.base.ClassifierMixin)
 |  XGBClassifier(*, objective: Union[str, Callable[[numpy.ndarray, numpy.ndarray], Tuple[numpy.ndarray, numpy.ndarray]], NoneType] = 'binary:logis
tic', use_label_encoder: Optional[bool] = None, **kwargs: Any) -> None
 |
 |  Implementation of the scikit-learn API for XGBoost classification.
 |
 |
 |  Parameters
 |  ----------
 |
 |      n_estimators : int
 |          Number of boosting rounds.
 |
 |      max_depth :  Optional[int]
 |          Maximum tree depth for base learners.
 |      max_leaves :
 |          Maximum number of leaves; 0 indicates no limit.
 |      max_bin :
 |          If using histogram-based algorithm, maximum number of bins per feature
 |      grow_policy :
 |          Tree growing policy. 0: favor splitting at nodes closest to the node, i.e. grow
 |          depth-wise. 1: favor splitting at nodes with highest loss change.
 |      learning_rate : Optional[float]
 |          Boosting learning rate (xgb's "eta")
 |      verbosity : Optional[int]
 |          The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
 |      objective : typing.Union[str, typing.Callable[[numpy.ndarray, numpy.ndarray], typing.Tuple[numpy.ndarray, numpy.ndarray]], NoneType]
 |          Specify the learning task and the corresponding learning objective or
 |          a custom objective function to be used (see note below).
 |      booster: Optional[str]
 |          Specify which booster to use: gbtree, gblinear or dart.
```

```
In [74]: xgb = XGBClassifier(n_estimators=200, max_depth=4, learning_rate=0.1,n_jobs=-1)
```

```
In [75]: xgb.fit(X_train, y_train)
```

```
Out[75]:                         XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

5.2 Predict Results

```
In [76]: y_predict = xgb.predict(X_test)
```

```
In [77]: print(y_predict)
```

```
[1 1 1 ... 0 1 1]
```

5.3 Performance Metrics

a) Accuracy score

```
In [79]: accuracy = accuracy_score(y_test, y_predict)
         print('Accuracy for XGBoost model is: ', accuracy)
```

```
Accuracy for XGBoost model is:  0.9590325765054294
```

The accuracy score is 95%, so very high, but this is not enough information to draw the conclusion that this model is actually suitable.

b) Confusion Matrix

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

```
In [80]: cm = confusion_matrix(y_test, y_predict)
```

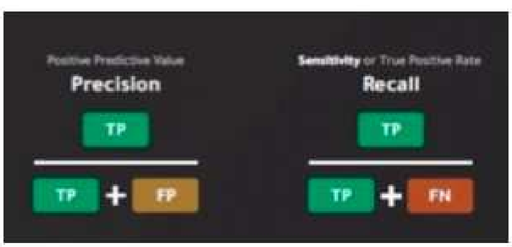
```
In [82]: fig, ax = plt.subplots(figsize=(12,8))
         sns.heatmap(cm, annot=True, fmt='d')
         plt.show()
```

Observation:

1. Out of all actually positive values, 266/(28+266)*100=90% were predicted corectly.
2. Out of all actually negatuve values, 1677/(1677+55)*100=96% were predicted corectly.

This algorithm seems to be good.

c) Precision, Recall




```
In [83]: precision = precision_score(y_test, y_predict)
         recall = recall_score(y_test, y_predict)
         print('Precision of the XGB model: ', precision)
         print('Recall of the XGBoost model: ', recall)

Precision of the XGB model:  0.9682448036951501
Recall of the XGBoost model:  0.9835777126099706
```

Observations:

1. Out of all the predicted positive values, 97% were predicted corectly.
2. Out of all actually positive values, 98% were predicted correctly.

The model seems to be a good one, considering the precision and the recall.

d) AUC score

Random Classifier = 0.5 Perfect Classifier = 1

n [84]:
```
auc_score = roc_auc_score(y_test, y_predict)
print('AUC for the XGBoost model: ', auc_score)
```

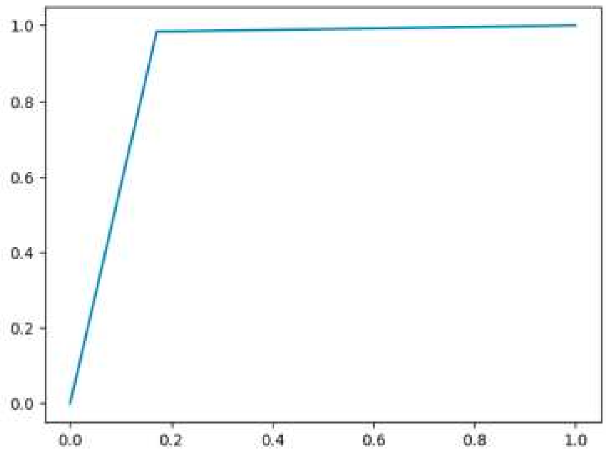
```
AUC for the XGBoost model:  0.906119074373521
```

A good AUC score has to be at least 0.7 and as closer to 1 as possible.

The AUC score is 90.61%, so it is a very good one.The XGBoost model can distinguish very well between the two classes, so it is a suitable choice.

n [85]:
```
fpr, tpr, thershold = roc_curve(y_test, y_predict)
plt.plot(fpr, tpr)
```

ut[85]:
```
[&lt;matplotlib.lines.Line2D at 0x2404db4f070&gt;]
```



5.4 Check for Overfit/Underfit

1.Overfitting is the situation in which a model fits very well the train set, but does not work very well on the test set.

2.Underfitting means that the modelis too simple: it does not work very well on the train set, but it does on the test set.

In [87]:
```
y_predict_train = xgb.predict(X_train)
y_predict_test = xgb.predict(X_test)
auc_score_train=roc_auc_score(y_train,y_predict_train)
auc_score_test=roc_auc_score(y_test, y_predict_test)
print('AUC score for the train set: ', auc_score_train)
print('AUC score for the test set: ', auc_score_test)
```

```
AUC score for the train set:  0.9623059699558386
AUC score for the test set:  0.906119074373521
```

There are no significant differneces between the train and the test set, which means there is no overfit, nor underfit.

6. Hyperparameters tunning

Hyperparameters are a particular kind of parameters that are not directly learned by the algorithm and are specified outside the training process.

Based on the different values they may take, the results of the model are changed. Therefore, they control the flexibility of the model, namely the capacity to fit the data.

These kind of parameters take euristic values, depending on the ones that are given, not necessarily the optimum one.

a) Declare possible hyperparameters

In [88]:
```
n_estimators = [200, 300]
max_depth = [3,4]
learning_rate=[0.05, 0.2]
```

b) Find the best hyperparameters

Observation:

Grid Search was used here: the hyperparameters were set manually and all their combinations were tested.

In [90]:
```python
results = []
for est in n_estimators:
    for md in max_depth:
        for lr in learning_rate:
            xgb = XGBClassifier(n_est = est, max_depth = md, learning_rate = lr, n_jobs = -1)
            xgb.fit(X_train,y_train)
            y_predict = xgb.predict(X_test)
            auc_score = roc_auc_score(y_test, y_predict)
            results.append(['n_estimators', est, 'max_depth', md, 'learning_rate', lr, 'AUC score', auc_score])
```

```
[15:14:34] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.
cc:767:
Parameters: { "n_est" } are not used.

[15:14:35] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.
cc:767:
Parameters: { "n_est" } are not used.

[15:14:36] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.
cc:767:
Parameters: { "n_est" } are not used.

[15:14:37] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.
cc:767:
Parameters: { "n_est" } are not used.

[15:14:38] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.
cc:767:
Parameters: { "n_est" } are not used.

[15:14:39] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.
cc:767:
Parameters: { "n_est" } are not used.
```

In [91]:
```python
print(results)
```

```
[['n_estimators', 200, 'max_depth', 3, 'learning_rate', 0.05, 'AUC score', 0.8262952101661779], ['n_estimators', 200, 'max_depth', 3, 'learning_rat
e', 0.2, 'AUC score', 0.9021240441801189], ['n_estimators', 200, 'max_depth', 4, 'learning_rate', 0.05, 'AUC score', 0.8687358968034276], ['n_estim
ators', 200, 'max_depth', 4, 'learning_rate', 0.2, 'AUC score', 0.9132293693644311], ['n_estimators', 300, 'max_depth', 3, 'learning_rate', 0.05,
'AUC score', 0.8262952101661779], ['n_estimators', 300, 'max_depth', 3, 'learning_rate', 0.2, 'AUC score', 0.9021240441801189], ['n_estimators', 30
0, 'max_depth', 4, 'learning_rate', 0.05, 'AUC score', 0.8687358968034276], ['n_estimators', 300, 'max_depth', 4, 'learning_rate', 0.2, 'AUC scor
e', 0.9132293693644311]]
```

Check for overfit

In [93]:
```python
best_xgb = XGBClassifier(n_estimators=300, max_depth=4, learnig_rate=0.2, n_jobs=-1)
best_xgb.fit(X_train, y_train)
y_predict_train = best_xgb.predict(X_train)
y_predict_test = best_xgb.predict(X_test)
auc_score_test = roc_auc_score(y_test, y_predict_test)
auc_score_train = roc_auc_score(y_train, y_predict_train)
print('AUC score for test is: ', auc_score_test)
print('AUC score for train is: ', auc_score_train)
```

```
[17:00:45] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.
cc:767:
Parameters: { "learnig_rate" } are not used.

AUC score for test is:  0.9205223778332009
AUC score for train is:  1.0
```

Since the AUC scores are not very differet, there is no overfit.

7. Save the model

In [94]:
```python
import pickle
with open('C:\\Users\\user\\OneDrive\\Desktop\\TTJ\\TTJ project\\project_best_model_xgb.pkl', 'wb') as file:
    pickle.dump(best_xgb, file)
```

III. Model Explainability

A. Import libraries

In [114…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
import shap
import pickle
```

B. Read Dataset

In [126…
```
data = pd.read_csv("C:\\Users\\user\\OneDrive\\Desktop\\TTJ\\TTJ project\\dataset\\dataset.csv")
```

In [127…
```
data.head()
```

Out[127]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Relati
0	768805383	1	45.0	M	3.0	High School	Married	$60K-80K$	Blue	39	
1	818770008	1	49.0	F	5.0	Graduate	Single	Less than $40K	Blue	44	
2	713982108	1	51.0	M	3.0	Graduate	Married	$80K-120K$	Blue	36	
3	769911858	1	40.0	F	4.0	High School	Unknown	Less than $40K	Blue	34	
4	709106358	1	40.0	M	3.0	Uneducated	Married	$60K-80K$	Blue	21	

In [128…
```
data.shape
```

Out[128]: (10127, 19)

C. Explainability

1. Import saved model

In [129…
```
with open ('C:\\Users\\user\\OneDrive\\Desktop\\TTJ\\TTJ project\\project_best_model_xgb.pkl', 'rb') as file:
    model = pickle.load(file)
```

2. Train test split

In [130…
```
#Independent variables
X = data.drop(columns = ['Attrition_Flag'])

#Target variable
y = data['Attrition_Flag']
```

In [131…
```
#Drop the unnecessary column: 'CLIENTNUM', since it does not bring any relevant information
X = X.drop(columns = ['CLIENTNUM'])
```

In [132…
```
#Categorical ecoding
categorical_columns = [col for col in X.columns if X[col].dtype == 'object']
X = pd.get_dummies(X, columns = categorical_columns)
```

In [133…
```
print(categorical_columns)
```

```
['Gender', 'Education_Level', 'Marital_Status', 'Income_Category', 'Card_Category']
```

In [134…
```
#Train test split
X_train, X_test, y_trai, y_test = train_test_split(X, y, test_size=0.2, random_state = 10)
```

In [135… `X.head()`

Out[135]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_Limit	Total_Used_Bal	Total_An
0	45.0	3.0	39	5	1	3	12691.0	777	
1	49.0	5.0	44	6	1	2	8256.0	864	
2	51.0	3.0	36	4	1	0	3418.0	0	
3	40.0	4.0	34	3	4	1	3313.0	2517	
4	40.0	3.0	21	5	1	0	4716.0	0	

5 rows × 34 columns

In [136… `X_train.shape, X_test.shape, y_train.shape, y_test.shape`

Out[136]: `((8101, 34), (2026, 34), (8101,), (2026,))`

3. Predict

In [137… `y_predict = model.predict(X_test)`

In [138… `y_predict`

Out[138]: `array([1, 1, 1, ..., 0, 1, 1])`

Predict likelyhood of the target variable

PROPENSITY PROBABILITIES = how much the current values of the target variable can help predict future values

In [139… `y_predict_proba = model.predict_proba(X_test)`

In [140… `y_predict_proba`

Out[140]:

```
array([[1.2612343e-04, 9.9987388e-01],
       [5.1736832e-05, 9.9994826e-01],
       [3.9249659e-04, 9.9960750e-01],
       ...,
       [9.9999464e-01, 5.3754893e-06],
       [7.6293945e-06, 9.9999237e-01],
       [3.1006336e-04, 9.9968994e-01]], dtype=float32)
```

Out of the 2 probabilities, the interest one is the one from class 0, which signifies the customers who have attrited.

In [141… `y_predict_proba_class0 = y_predict_proba[:,0]`

In [142… `y_predict_proba_class0`

Out[142]:

```
array([1.2612343e-04, 5.1736832e-05, 3.9249659e-04, ..., 9.9999464e-01,
       7.6293945e-06, 3.1006336e-04], dtype=float32)
```

4. Performace metrics

4.1 Lift and Gain analysis

Examining the cumulative gains and lift associated with the model means comparing its performance in prediction of the target variable with how accurate the same prediction would be without the added value offered by the model.

The main difference between these 2 techniques are the values to be compared: GAIN is expressed as a cumulative percentage of the number of positives out of the total number, whereas the LIFT is simply the CUMULATIVE GAIN/RADOM BASE, which means that the comparison is made at decile level and it emphasizes the difference between the predicted values without a model and the values predcited by the model.

In [144…

```
#Declare an empty DataFrame
lift_gain_report = pd.DataFrame()
print(lift_gain_report)
```

```
Empty DataFrame
Columns: []
Index: []
```

In [145…
```
#add y_test in the DataFrame
lift_gain_report['y_test'] = y_test
lift_gain_report
```

Out[145]:

	y_test
2286	1
4476	1
8935	1
9463	1
1478	0
...	...
394	1
1459	1
3320	0
5506	1
6714	1

2026 rows × 1 columns

Step 1: Predict probabilities for class 0 in the DataFrame

PROBABILITIES FOR CLASS 0 = how accurate the possibility that a customer will attrit can be predicted by the model

In [146…
```
lift_gain_report['Predicted Probabilities'] = y_predict_proba_class0
```

In [147…
```
lift_gain_report
```

Out[147]:

	y_test	Predicted Probabilities
2286	1	0.000126
4476	1	0.000052
8935	1	0.000392
9463	1	0.000036
1478	0	0.134543
...	...	...
394	1	0.000074
1459	1	0.000236
3320	0	0.999995
5506	1	0.000008
6714	1	0.000310

2026 rows × 2 columns

Step 2: Order the probabilities in ascendig order

In [149…
```
lift_gain_report['Probabilities Rank'] = lift_gain_report['Predicted Probabilities'].rank(method='first', ascending=True, pct=True)
```

Step 3: Calculate decile group

In [150…
```
lift_gain_report['Decile Group'] = np.floor((1-lift_gain_report['Probabilities Rank'])*10)+1
```

In [151…
```
lift_gain_report
```

Out[151]:

	y_test	Predicted Probabilities	Probabilities Rank	Decile Group
2286	1	0.000126	0.447187	6.0
4476	1	0.000052	0.352419	7.0
8935	1	0.000392	0.550839	5.0
9463	1	0.000036	0.311451	7.0
1478	0	0.134543	0.818855	2.0
...	...	...	...	...
394	1	0.000074	0.389931	7.0
1459	1	0.000236	0.506910	5.0
3320	0	0.999995	0.994571	1.0
5506	1	0.000008	0.149062	9.0
6714	1	0.000310	0.527641	5.0

2026 rows × 4 columns

Step 4: Group observations by deciles

In [152…
```
lift_gain_report['Number of observations'] = 1
lift_gain_report = lift_gain_report.groupby(['Decile Group']).sum().reset_index()
lift_gain_report
```

Out[152]:

	Decile Group	y_test	Predicted Probabilities	Probabilities Rank	Number of observations
0	1.0	8	202.007645	192.880059	203
1	2.0	94	105.991699	172.539980	203
2	3.0	188	3.728223	151.500000	202
3	4.0	202	0.369384	131.960020	203
4	5.0	201	0.087765	111.119941	202
5	6.0	202	0.027617	91.380059	203
6	7.0	203	0.010637	71.039980	203
7	8.0	202	0.004205	50.500000	202
8	9.0	203	0.001670	30.460020	203
9	10.0	202	0.000325	10.119941	202

In [153…
```
#Cumulative number of observations
lift_gain_report['Cumulative no. of observations'] = lift_gain_report['Number of observations'].cumsum()
lift_gain_report
```

Out[153]:

	Decile Group	y_test	Predicted Probabilities	Probabilities Rank	Number of observations	Cumulative no. of observations
0	1.0	8	202.007645	192.880059	203	203
1	2.0	94	105.991699	172.539980	203	406
2	3.0	188	3.728223	151.500000	202	608
3	4.0	202	0.369384	131.960020	203	811
4	5.0	201	0.087765	111.119941	202	1013
5	6.0	202	0.027617	91.380059	203	1216
6	7.0	203	0.010637	71.039980	203	1419
7	8.0	202	0.004205	50.500000	202	1621
8	9.0	203	0.001670	30.460020	203	1824
9	10.0	202	0.000325	10.119941	202	2026

In [154…
```
#Cumulative percentage of observations
lift_gain_report['Cumulative % of no. of observations'] = lift_gain_report['Cumulative no. of observations']/lift_gain_report['Cumulative no. of ol

lift_gain_report
```

Out[154]:

	Decile Group	y_test	Predicted Probabilities	Probabilities Rank	Number of observations	Cumulative no. of observations	Cumulative % of no. of observations
0	1.0	8	202.007645	192.880059	203	203	0.100197
1	2.0	94	105.991699	172.539980	203	406	0.200395
2	3.0	188	3.728223	151.500000	202	608	0.300099
3	4.0	202	0.369384	131.960020	203	811	0.400296
4	5.0	201	0.087765	111.119941	202	1013	0.500000
5	6.0	202	0.027617	91.380059	203	1216	0.600197
6	7.0	203	0.010637	71.039980	203	1419	0.700395
7	8.0	202	0.004205	50.500000	202	1621	0.800099
8	9.0	203	0.001670	30.460020	203	1824	0.900296
9	10.0	202	0.000325	10.119941	202	2026	1.000000

Step 5: Calculate the cumulative number of positive values

In [155]:
```
lift_gain_report['Cumulative no. of positives']=lift_gain_report['y_test'].cumsum()
lift_gain_report
```

Out[155]:

	Decile Group	y_test	Predicted Probabilities	Probabilities Rank	Number of observations	Cumulative no. of observations	Cumulative % of no. of observations	Cumulative no. of positives
0	1.0	8	202.007645	192.880059	203	203	0.100197	8
1	2.0	94	105.991699	172.539980	203	406	0.200395	102
2	3.0	188	3.728223	151.500000	202	608	0.300099	290
3	4.0	202	0.369384	131.960020	203	811	0.400296	492
4	5.0	201	0.087765	111.119941	202	1013	0.500000	693
5	6.0	202	0.027617	91.380059	203	1216	0.600197	895
6	7.0	203	0.010637	71.039980	203	1419	0.700395	1098
7	8.0	202	0.004205	50.500000	202	1621	0.800099	1300
8	9.0	203	0.001670	30.460020	203	1824	0.900296	1503
9	10.0	202	0.000325	10.119941	202	2026	1.000000	1705

Step 6: Calculate cumulative percentage of positives:GAIN

GAIN = the cumulative number of positive values out of the total number of positive values

In [156]:
```
lift_gain_report['Gain']=lift_gain_report['Cumulative no. of positives']/lift_gain_report['Cumulative no. of positives'].max()
lift_gain_report
```

Out[156]:

	Decile Group	y_test	Predicted Probabilities	Probabilities Rank	Number of observations	Cumulative no. of observations	Cumulative % of no. of observations	Cumulative no. of positives	Gain
0	1.0	8	202.007645	192.880059	203	203	0.100197	8	0.004692
1	2.0	94	105.991699	172.539980	203	406	0.200395	102	0.059824
2	3.0	188	3.728223	151.500000	202	608	0.300099	290	0.170088
3	4.0	202	0.369384	131.960020	203	811	0.400296	492	0.288563
4	5.0	201	0.087765	111.119941	202	1013	0.500000	693	0.406452
5	6.0	202	0.027617	91.380059	203	1216	0.600197	895	0.524927
6	7.0	203	0.010637	71.039980	203	1419	0.700395	1098	0.643988
7	8.0	202	0.004205	50.500000	202	1621	0.800099	1300	0.762463
8	9.0	203	0.001670	30.460020	203	1824	0.900296	1503	0.881525
9	10.0	202	0.000325	10.119941	202	2026	1.000000	1705	1.000000

Step 7: Calculate the Lift

LIFT = GAIN divided by the cumulative percentage of the number of observations, which is actually the decile group

In [157]:
```
lift_gain_report['Lift'] = lift_gain_report['Gain']/lift_gain_report['Cumulative % of no. of observations']
lift_gain_report
```

Out[157]:

	Decile Group	y_test	Predicted Probabilities	Probabilities Rank	Number of observations	Cumulative no. of observations	Cumulative % of no. of observations	Cumulative no. of positives	Gain	Lift
0	1.0	8	202.007645	192.880059	203	203	0.100197	8	0.004692	0.046828
1	2.0	94	105.991699	172.539980	203	406	0.200395	102	0.059824	0.298531
2	3.0	188	3.728223	151.500000	202	608	0.300099	290	0.170088	0.566773
3	4.0	202	0.369384	131.960020	203	811	0.400296	492	0.288563	0.720874
4	5.0	201	0.087765	111.119941	202	1013	0.500000	693	0.406452	0.812903
5	6.0	202	0.027617	91.380059	203	1216	0.600197	895	0.524927	0.874590
6	7.0	203	0.010637	71.039980	203	1419	0.700395	1098	0.643988	0.919465
7	8.0	202	0.004205	50.500000	202	1621	0.800099	1300	0.762463	0.952962
8	9.0	203	0.001670	30.460020	203	1824	0.900296	1503	0.881525	0.979150
9	10.0	202	0.000325	10.119941	202	2026	1.000000	1705	1.000000	1.000000

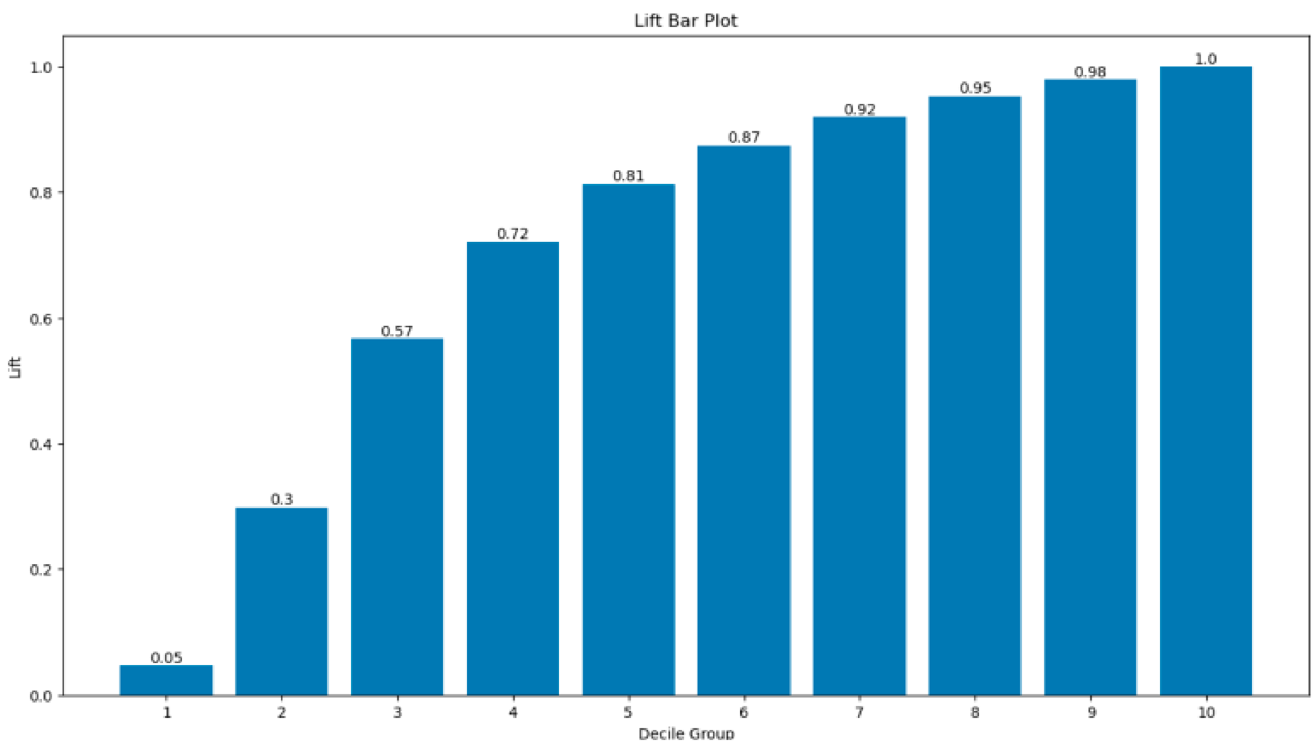
4.3 Lift and Gain charts

a) Lift chart

```
fig, ax = plt.subplots(figsize = (15, 8))
barplot = plt.bar(lift_gain_report['Decile Group'], lift_gain_report['Lift'])
plt.title("Lift Bar Plot")
plt.xlabel('Decile Group')
plt.ylabel('Lift')
plt.xticks(lift_gain_report['Decile Group'])

#Add text above bars from the chart
for b in barplot:
    plt.text(b.get_x() + b.get_width()/2, b.get_height()+0.005, round(b.get_height(),2), ha = 'center')


plt.show()
```



From this bar plot, an example of conclusion that can be drawn is that the first 30% of the observations are by 57% more accurately predicted by the model than they would have been without it.

b) Gain Chart

```
lift_gain_report['Random Selection'] = lift_gain_report['Decile Group']/lift_gain_report['Decile Group'].max()
```

Observation:

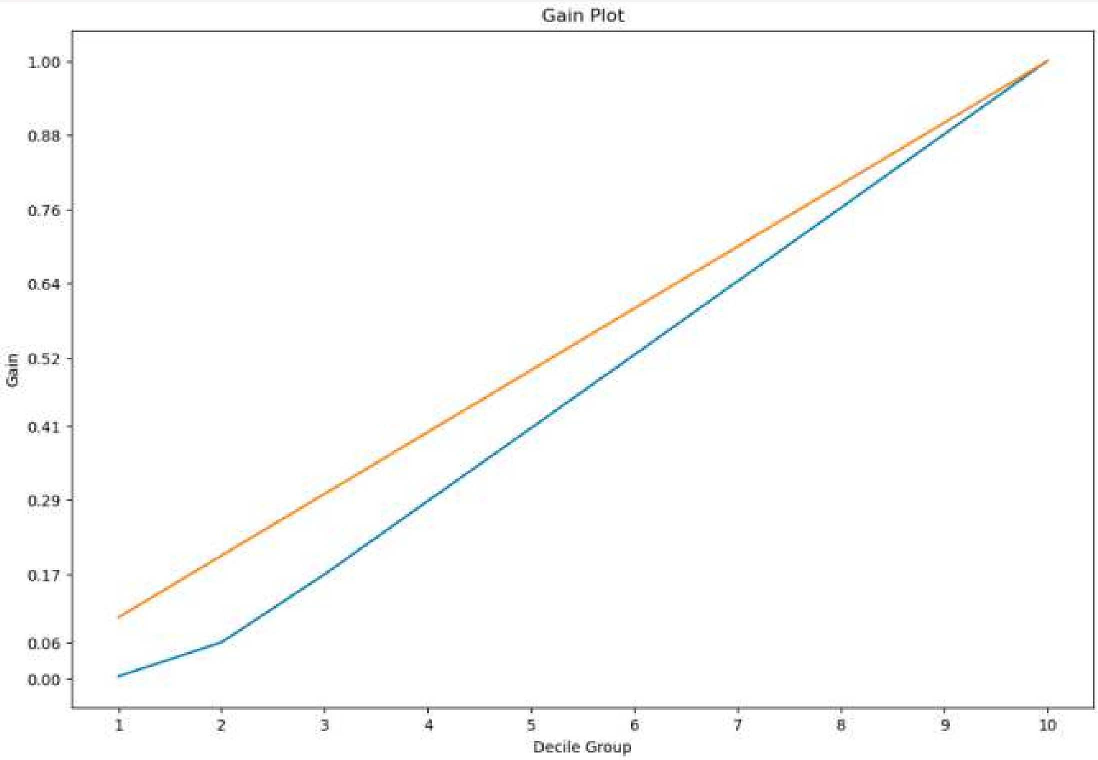
Since the GAIN represets the LIFT divided by the RANDOM BASE, which means by the decile group, the 'Random Selection' column actually represents the decile group or the cumulative percentage of the number of observations.

```
lift_gain_report
```

	Decile Group	y_test	Predicted Probabilities	Probabilities Rank	Number of observations	Cumulative no. of observations	Cumulative % of no. of observations	Cumulative no. of positives	Gain	Lift	Random Selection
0	1.0	8	202.007645	192.880059	203	203	0.100197	8	0.004692	0.046828	0.1
1	2.0	94	105.991699	172.539980	203	406	0.200395	102	0.059824	0.298531	0.2
2	3.0	188	3.728223	151.500000	202	608	0.300099	290	0.170088	0.566773	0.3
3	4.0	202	0.369384	131.960020	203	811	0.400296	492	0.288563	0.720874	0.4
4	5.0	201	0.087765	111.119941	202	1013	0.500000	693	0.406452	0.812903	0.5
5	6.0	202	0.027617	91.380059	203	1216	0.600197	895	0.524927	0.874590	0.6
6	7.0	203	0.010637	71.039980	203	1419	0.700395	1098	0.643988	0.919465	0.7
7	8.0	202	0.004205	50.500000	202	1621	0.800099	1300	0.762463	0.952962	0.8
8	9.0	203	0.001670	30.460020	203	1824	0.900296	1503	0.881525	0.979150	0.9
9	10.0	202	0.000325	10.119941	202	2026	1.000000	1705	1.000000	1.000000	1.0

In [167…

```
fig, ax = plt.subplots(figsize=(12, 8))
sns.lineplot(data = lift_gain_report, x = lift_gain_report['Decile Group'], y = lift_gain_report['Gain'])
sns.lineplot(data=lift_gain_report, x = lift_gain_report['Decile Group'], y = lift_gain_report['Random Selection'])
plt.title("Gain Plot")
plt.xticks(lift_gain_report['Decile Group'])
plt.yticks(round(lift_gain_report['Gain'],2))
plt.show()
```



5. Feature Importance

5.1 Feature Importance Analysis

In [168…

```
feat_imp = model.get_booster().get_score(importance_type='total_gain')
```

In [169…

```
feat_imp
```

Out[169]:

```
{'Customer_Age': 656.4432373046875,
 'Dependent_count': 31.986974716186523,
 'Months_on_book': 322.0689697265625,
 'Total_Relationship_Count': 1134.186279296875,
 'Months_Inactive_12_mon': 318.8481140136719,
 'Contacts_Count_12_mon': 271.1667175292969,
 'Credit_Limit': 445.6159973144531,
 'Total_Used_Bal': 1607.1461181640625,
 'Total_Amt_Chng_Q4_Q1': 1097.7427978515625,
 'Total_Trans_Amt': 4601.90087890625,
 'Total_Ct_Chng_Q4_Q1': 1870.585693359375,
 'Avg_Utilization_Ratio': 175.61996459960938,
 'Gender_F': 96.19579315185547,
 'Education_Level_College': 11.898409843444824,
 'Education_Level_Doctorate': 6.0171284675598145,
 'Education_Level_Graduate': 23.625629425048828,
 'Education_Level_High School': 10.309598922729492,
 'Education_Level_Post-Graduate': 11.70179271697998,
 'Education_Level_Uneducated': 10.471820831298828,
 'Education_Level_Unknown': 9.477663040161133,
 'Marital_Status_Divorced': 4.086399078369141,
 'Marital_Status_Married': 26.310712814331055,
 'Marital_Status_Single': 46.43423843383789,
 'Marital_Status_Unknown': 5.9292497634887695,
 'Income_Category_$120K +': 11.155954360961914,
 'Income_Category_$40K - $60K': 5.383142948150635,
 'Income_Category_$60K - $80K': 15.538938522338867,
 'Income_Category_$80K - $120K': 11.048712730407715,
 'Income_Category_Less than $40K': 8.374650955200195,
 'Income_Category_abc': 3.0575735569000244,
 'Card_Category_Blue': 9.089627265930176,
 'Card_Category_Other Cards': 2.931818962097168,
 'Card_Category_Silver': 0.652199387550354}
```

Observation:

The values are sorted according to their importance in predictig the values of the target variable based on the GAIN analysis and the percentage of their contribution to the model predictability is computed.

In [170]:
```
feature_importance= pd.DataFrame()
feature_importance['Variable'] = feat_imp.keys()
feature_importance['Importance Value']=feat_imp.values()
feature_importance['% Importance Value'] = feature_importance['Importance Value']/feature_importance['Importance Value'].sum()*100
feature_importance.sort_values(by=['Importance Value'], ascending=False)
```

Out[170]:

	Variable	Importance Value	% Importance Value
9	Total_Trans_Amt	4601.900879	35.776257
10	Total_Ct_Chng_Q4_Q1	1870.585693	14.542372
7	Total_Used_Bal	1607.146118	12.494331
3	Total_Relationship_Count	1134.186279	8.817430
8	Total_Amt_Chng_Q4_Q1	1097.742798	8.534110
0	Customer_Age	656.443237	5.103344
6	Credit_Limit	445.615997	3.464323
2	Months_on_book	322.068970	2.503840
4	Months_Inactive_12_mon	318.848114	2.478800
5	Contacts_Count_12_mon	271.166718	2.108114
11	Avg_Utilization_Ratio	175.619965	1.365311
12	Gender_F	96.195793	0.747849
22	Marital_Status_Single	46.434238	0.360991
1	Dependent_count	31.986975	0.248674
21	Marital_Status_Married	26.310713	0.204546
15	Education_Level_Graduate	23.625629	0.183671
26	Income_Category_$60K-$80K	15.538939	0.120803
27	Income_Category_$80K-$120K	11.048713	0.085895
18	Education_Level_Uneducated	10.471821	0.081410
16	Education_Level_High School	10.309599	0.080149
19	Education_Level_Unknown	9.477663	0.073682
30	Card_Category_Blue	9.089627	0.070665
28	Income_Category_Less than $40K	8.374651	0.065107
14	Education_Level_Doctorate	6.017128	0.046779
23	Marital_Status_Unknown	5.929250	0.046095
25	Income_Category_$40K-$60K	5.383143	0.041850
20	Marital_Status_Divorced	4.086399	0.031769
29	Income_Category_abc	3.057574	0.023770
31	Card_Category_Other Cards	2.931819	0.022793
32	Card_Category_Silver	0.652199	0.005070

Observation:

It can be stated that the total amount of payments made in the last 12 moths is the most important factor that determines whether the customer will attrit or not.

5.2 SHAP Chart

SHAP stands for SHapley Additive exPlanations and emphasizes the contribution of each feature to the predictions made by the model. It conducts a local, as well as global analysis and it is derived from the Game Theory. The 'game' is represented by reproducing the outcome of the model and the 'players' are the features included.

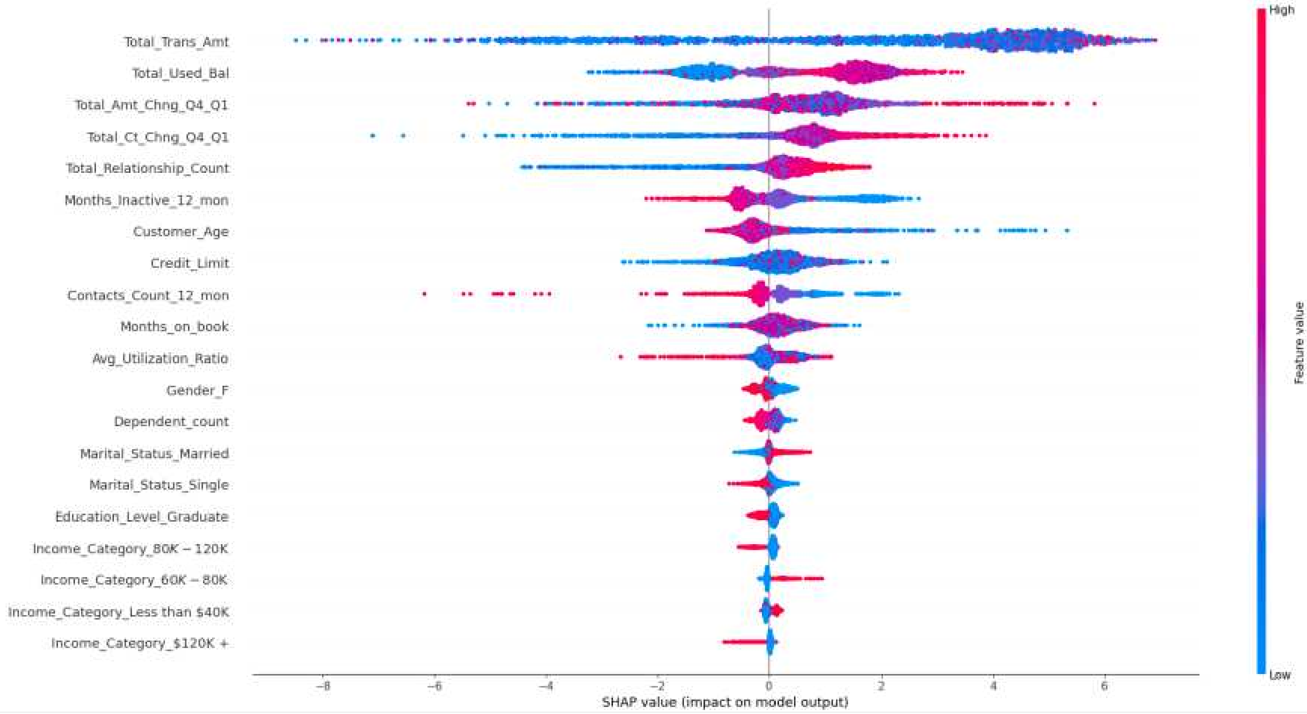
The variables are sorted according to their impact on the model and their scale is important in analyzig their impact. SHAP analysis is based on decision trees and its main objective is to describe the relationship between each feature ad the target variable in a manner which is easy to comprehend.

In [171]:
```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
#SHAP always done on set test
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

```
In [172… shap.summary_plot(shap_values,X_test,plot_size=(20,10))
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored
```



Observations:

1. The total amount of payments made in last 12 moths has the biggest impact on the target variable. The points which correspond to the negative segment of the Ox axis represent a negative impact on the target variable, most of them being low values as well (their color is blue). A conclusion that could be drawn is that the lower the number of payments made in the last 12 months, the lower the probability of this customer to attrit - since the money have not been withdrawn, he does not seem to be considering this decision.
2. Same analysis can be applied on the total used amount from credit card limit as well - as the second most important variable for this model, the higher its values (points that are red), the higher the value on the Ox axis, which means that the bigger the amount of money used from an account, the bigger the chance that the respective customer intends to attrit - perhaps he considers liquidating the accounts and depositing the money in another bank.
3. On the other hand, when it comes to the number of months with inactivity out of last 12 (no payments made), for example, the situation is reversed. The higher the number of months of inactivity, the lower the probability of the customer to attrit, due to the fact that there can possibly be a big amount of money accumulated in the account and the customer does not seem to intend to liquidate it.

Thank you!

Bibliography

1. Data Science 101 | Free Courses in Data Science, AI, Cloud Computing, Containers, Kubernetes, Blockchain and more. (cognitiveclass.ai): https://cognitiveclass.ai/courses/data-science-101
2. What is data science? (Coursera): https://www.coursera.org/learn/what-is-datascience?irclickid=TY0XfVW08xyIT0wQfeSStWSMUkAWugyi2y7AS00&irgwc=1&utm_medium=partners&utm_source=impact&utm_campaign=3294490&utm_content=b2c