Consolidated Segmentation and Churn Analysis of Bank Clients

By: Tamjid Ahsan

As capstone project of Flatiron Data Science Bootcamp.

- Student pace: Full Time
- Scheduled project review date/time: July 22, 2021, 04:00 PM [DST]
- Instructor name: James Irving

ABSTRACT

Attracting new customers is no longer a good strategy for mature businesses since the cost of retaining existing customers is much lower. For this reason, customer churn management becomes instrumental for any service industry.

This analysis is combining churn prediction and customer segmentation and aims to come up with an integrated customer analytics outline for churn management. There are six components in this analysis, starting with data pre-processing, exploratory data analysis, customer segmentation, customer characteristics analytics, churn prediction, and factor analysis. This analysis is adapting OESMiN framework for data science.

Customer data of a bank is used for this analysis. After preprocessing and exploratory data analysis, customer segmentation is carried out using K-means clustering. A Random Forest model is used focusing on optimizing f-1 score to validate the clustering and get feature importance. By using this model, customers are segmented into different groups, which sanctions marketers and decision makers to implement existing customer retention strategies more precisely. Then different machine learning models are used with the preprocessed data along with the segmentation prediction from the K-means clustering model. For this type of modeling, models were optimized for precision. To address class imbalance Synthetic Minority Oversampling Technique (SMOTE) is applied to the training set. For factor analysis feature importance of models are used.

OVERVIEW

Customer churn is a big issue that occurs when consumers abandon your products and go to another provider. Because of the direct impact on profit margins, firms are now focusing on identifying consumers who are at danger of churning and keeping them through tailored promotional offers. Customer churn analysis and customer turnover rates are frequently used as essential business indicators by banks, insurance firms, streaming service providers, and telecommunications service providers since the cost of maintaining existing customers is significantly less than the cost of obtaining a new one.

When it comes to customers, the financial crisis of 2008 changed the banking sector's strategy. Prior to the financial crisis, banks were mostly focused on acquiring more and more clients. However, once the market crashed after the market imploded, banks realized rapidly that the expense of attracting new clients is multiple times higher than holding existing ones, which means losing clients can be monetarily unfavorable. Fast forward to today, and the global banking sector has a market capitalization of $7.6 trillion, with technology and laws making things easier than ever to transfer assets and money between institutions. Furthermore, it has given rise to new forms of competition for banks, such as open banking, neo-banks, and fin-tech businesses (Banking as a Service (BaaS))[1]. Overall, today's consumers have more options than ever before, making it easier than ever to transfer or quit banks altogether. According to studies, repeat customers seem to be more likely to spend 67 percent more on a bank's products and services, emphasizing the necessity of knowing why clients churn and how it varies across different characteristics. Banking is one of those conventional sectors that has undergone continuous development throughout the years. Nonetheless, many banks today with a sizable client base expecting to gain a competitive advantage have not tapped into the huge amounts of data they have, particularly in tackling one of the most well-known challenges, customer turnover.

Churn can be expressed as a level of customer inactivity or disengagement seen over a specific period. This expresses itself in the data in a variety of ways e.g., frequent balance transfers to another account or unusual drop in average balance over time. But how can anyone look for churn indicators? Collecting detailed feedback on the customer's experience might be difficult. For one thing, surveys are both rare and costly. Furthermore, not all clients receive it, or bother to reply to it. So, where else can you look for indicators of future client dissatisfaction? The solution consists in identifying early warning indicators from existing data. Advanced machine learning and data science techniques can learn from previous customer behavior and external events that lead to churn and use this knowledge to anticipate the possibility of a churn-like event in the future.

Ref:

[1] Business Insider

[2] Stock images from PEXELS

BUSINESS PROBLEM

While everyone recognizes the importance of maintaining existing customers and therefore improving their lifetime value, there is very little banks can do about customer churn when they don't anticipate it coming in the first place. Predicting attrition becomes critical in this situation, especially when unambiguous consumer feedback is lacking. Precise prediction enables advertisers and client experience groups to be imaginative and proactive in their offering to the client.

XYZ Bank (read: fictional) is a mature financial institution based in Eastern North America. Recent advance in technology and rise in BaaS is a real threat for them as they can lure away the existing clientele. The bank has existing data of their clients. Based on the data available, the bank wants to know whom of them are in risk of churning.

This analysis focuses on the behavior of bank clients who are more likely to leave the bank (**i.e. close their bank account**, churn).

IMPORTS

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
from imports_and_functions.packages import *
import imports_and_functions as fn
```

In [3]:
```
# notebook styling
try:
    from jupyterthemes import jtplot
except:
    !pip install jupyterthemes
    from jupyterthemes import jtplot
# jtplot.reset() # reset notebook styling
jtplot.style(theme='monokai', context='notebook', ticks='True', grid='False')
```

OBTAIN

The data for this analysis is obtained from *Kaggle*, titled **"Credit Card customers"** uploaded by Sakshi Goyal. Which can be found here, this dataset was originally obtained from LEAPS Analyttica. A copy of the data is in this repository at `/data/BankChurners.csv`.

This dataset contains data of more than 10000 credit card accounts with around 19 variables of different types as of a time point and their attrition indicator over the next 6 months.

Data description is as below:

Variable	Type	Description
Clientnum	Num	Client number. Unique identifier for the customer holding the account
Attrition_Flag	obj	Internal event (customer activity) variable - if the account is closed then 1 else 0
Customer_Age	Num	Demographic variable - Customer's Age in Years
Gender	obj	Demographic variable - M=Male, F=Female
Dependent_count	Num	Demographic variable - Number of dependents
Education_Level	obj	Demographic variable - Educational Qualification of the account holder (example: high school, college graduate, etc.)
Marital_Status	obj	Demographic variable - Married, Single, Divorced, Unknown
Income_Category	obj	Demographic variable - Annual Income Category of the account holder (< $40K, $40K - 60K, $60K - $80K, $80K-$120K, > $120K, Unknown)
Card_Category	obj	Product Variable - Type of Card (Blue, Silver, Gold, Platinum)
Months_on_book	Num	Months on book (Time of Relationship)
Total_Relationship_Count	Num	Total no. of products held by the customer

Months_Inactive_12_mon	Num	No. of months inactive in the last 12 months
Contacts_Count_12_mon	Num	No. of Contacts in the last 12 months
Credit_Limit	Num	Credit Limit on the Credit Card
Total_Revolving_Bal	Num	Total Revolving Balance on the Credit Card
Avg_Open_To_Buy	Num	Open to Buy Credit Line (Average of last 12 months)
Total_Amt_Chng_Q4_Q1	Num	Change in Transaction Amount (Q4 over Q1)
Total_Trans_Amt	Num	Total Transaction Amount (Last 12 months)
Total_Trans_Ct	Num	Total Transaction Count (Last 12 months)
Total_Ct_Chng_Q4_Q1	Num	Change in Transaction Count (Q4 over Q1)
Avg_Utilization_Ratio	Num	Average Card Utilization Ratio

There are three unknown category in Education Level, Marital Status, and Income Category. Imputing values for those features does not make sense. And it is understandable why those are unknown in the first place. Information about Education and Marital status is often complicated and confidential and customers are reluctant to share those information. Same for the income level. It is best for the model to be able to handle when those information is not available and still produce prediction.

For this reason those are not imputed in any way for this analysis.

In [4]:
```
# loading dataset
raw_df = pd.read_csv('./data/BankChurners.csv')
# first view of the dataset
raw_df
```

Out[4]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category
0	768805383	Existing Customer	45	M	3	High School	Married	$60K - $80K	Blue
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K - $120K	Blue
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue
4	709106358	Existing Customer	40	M	3	Uneducated	Married	$60K - $80K	Blue
...	...	...	...	...	...	...	...	...	...
10122	772366833	Existing Customer	50	M	2	Graduate	Single	$40K - $60K	Blue
10123	710638233	Attrited Customer	41	M	2	Unknown	Divorced	$40K - $60K	Blue
10124	716506083	Attrited Customer	44	F	1	High School	Married	Less than $40K	Blue
10125	717406983	Attrited Customer	30	M	2	Graduate	Unknown	$40K - $60K	Blue
10126	714337233	Attrited Customer	43	F	2	Graduate	Married	Less than $40K	Silver

10127 rows × 23 columns

In [5]:
```
# columns of the dataset
raw_df.columns
```

Out[5]:
```
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_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',
       
'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months
_Inactive_12_mon_1',
       
'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months
_Inactive_12_mon_2'],
      dtype='object')
```

In [6]:
```python
# no duplicates in the dataset
raw_df.CLIENTNUM.duplicated().value_counts()
```

Out[6]:
```
False    10127
Name: CLIENTNUM, dtype: int64
```

In [7]:
```python
# dropping customer identifier and unnecessary feature
raw_df.drop(columns=[
    'CLIENTNUM',

'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_


'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_

],
        inplace=True)
raw_df
```

Out[7]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_b
0	Existing Customer	45	M	3	High School	Married	$60K - $80K	Blue	
1	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	
2	Existing Customer	51	M	3	Graduate	Married	$80K - $120K	Blue	
3	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	
4	Existing Customer	40	M	3	Uneducated	Married	$60K - $80K	Blue	
...	...	...	...	...	...	...	...	...	
10122	Existing Customer	50	M	2	Graduate	Single	$40K - $60K	Blue	
10123	Attrited Customer	41	M	2	Unknown	Divorced	$40K - $60K	Blue	
10124	Attrited Customer	44	F	1	High School	Married	Less than $40K	Blue	
10125	Attrited Customer	30	M	2	Graduate	Unknown	$40K - $60K	Blue	
10126	Attrited Customer	43	F	2	Graduate	Married	Less than $40K	Silver	

10127 rows × 20 columns

In [8]:
```python
# looking at the distribution for changing labels to more notebook friendly description
raw_df['Income_Category'].value_counts()
```

Out[8]:
```
Less than $40K    3561
$40K - $60K       1790
$80K - $120K      1535
$60K - $80K       1402
Unknown           1112
$120K +            727
Name: Income_Category, dtype: int64
```

In [9]:
```python
# cleaning text in 'Income_Category'
raw_df['Income_Category'] = raw_df['Income_Category'].apply(
    lambda x: x.replace("$", "")).apply(
        lambda x: x.replace(" - ", "_to_")).apply(
```

```
            lambda x: x.replace("120K +", "Above_120K")).apply(
                lambda x: x.replace("Less than 40K", "Less_than_40K"))
raw_df
```

Out[9]:

	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_l
0	Existing Customer	45	M	3	High School	Married	60K_to_80K	Blue	
1	Existing Customer	49	F	5	Graduate	Single	Less_than_40K	Blue	
2	Existing Customer	51	M	3	Graduate	Married	80K_to_120K	Blue	
3	Existing Customer	40	F	4	High School	Unknown	Less_than_40K	Blue	
4	Existing Customer	40	M	3	Uneducated	Married	60K_to_80K	Blue	
...	...	...	...	...	...	...	...	...	
10122	Existing Customer	50	M	2	Graduate	Single	40K_to_60K	Blue	
10123	Attrited Customer	41	M	2	Unknown	Divorced	40K_to_60K	Blue	
10124	Attrited Customer	44	F	1	High School	Married	Less_than_40K	Blue	
10125	Attrited Customer	30	M	2	Graduate	Unknown	40K_to_60K	Blue	
10126	Attrited Customer	43	F	2	Graduate	Married	Less_than_40K	Silver	

10127 rows × 20 columns

In [10]:

```
# distribution of target
(raw_df.Attrition_Flag.value_counts(1)*100).round(2)
```

Out[10]:

```
Existing Customer    83.93
Attrited Customer    16.07
Name: Attrition_Flag, dtype: float64
```

There is major class imbalance spotted in the target column.

In [11]:

```
df = raw_df.copy()
print(f'"df" statistical description: \n{"+"*30}')
display(fn.describe_dataframe(df))
print(f'"df" feature details: \n{"+"*30}\n')
fn.check_duplicates(df, verbose=2, limit_num=50)
```

```
"df" statistical description:
++++++++++++++++++++++++++++++
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	dtype	nulls
Attrition_Flag	10127.0	2	Existing Customer	8500								object	0
Customer_Age	10127.0				46.33	8.02	26.0	41.0	46.0	52.0	73.0	int64	0
Gender	10127.0	2	F	5358								object	0
Dependent_count	10127.0				2.35	1.3	0.0	1.0	2.0	3.0	5.0	int64	0
Education_Level	10127.0	7	Graduate	3128								object	0
Marital_Status	10127.0	4	Married	4687								object	0
Income_Category	10127.0	6	Less_than_40K	3561								object	0
Card_Category	10127.0	4	Blue	9436								object	0
Months_on_book	10127.0				35.93	7.99	13.0	31.0	36.0	40.0	56.0	int64	0
Total_Relationship_Count	10127.0				3.81	1.55	1.0	3.0	4.0	5.0	6.0	int64	0
Months_Inactive_12_mon	10127.0				2.34	1.01	0.0	2.0	2.0	3.0	6.0	int64	0
Contacts_Count_12_mon	10127.0				2.46	1.11	0.0	2.0	2.0	3.0	6.0	int64	0
Credit_Limit	10127.0				8631.95	9088.78	1438.3	2555.0	4549.0	11067.5	34516.0	float64	0
Total_Revolving_Bal	10127.0				1162.81	814.99	0.0	359.0	1276.0	1784.0	2517.0	int64	0

Avg_Open_To_Buy	10127.0		7469.14	9090.69	3.0	1324.5	3474.0	9859.0	34516.0	float64	0
Total_Amt_Chng_Q4_Q1	10127.0		0.76	0.22	0.0	0.63	0.74	0.86	3.4	float64	0
Total_Trans_Amt	10127.0		4404.09	3397.13	510.0	2155.5	3899.0	4741.0	18484.0	int64	0
Total_Trans_Ct	10127.0		64.86	23.47	10.0	45.0	67.0	81.0	139.0	int64	0
Total_Ct_Chng_Q4_Q1	10127.0		0.71	0.24	0.0	0.58	0.7	0.82	3.71	float64	0
Avg_Utilization_Ratio	10127.0		0.27	0.28	0.0	0.02	0.18	0.5	1.0	float64	0

```
"df" feature details:
++++++++++++++++++++++++++++++

Attrition_Flag >> number of uniques: 2
Values:
['Existing Customer' 'Attrited Customer']
____________________________________________________________

Customer_Age >> number of uniques: 45
Values:
[45 49 51 40 44 32 37 48 42 65 56 35 57 41 61 47 62 54 59 63 53 58 55 66
 50 38 46 52 39 43 64 68 67 60 73 70 36 34 33 26 31 29 30 28 27]
____________________________________________________________

Gender >> number of uniques: 2
Values:
['M' 'F']
____________________________________________________________

Dependent_count >> number of uniques: 6
Values:
[3 5 4 2 0 1]
____________________________________________________________

Education_Level >> number of uniques: 7
Values:
['High School' 'Graduate' 'Uneducated' 'Unknown' 'College' 'Post-Graduate'
 'Doctorate']
____________________________________________________________

Marital_Status >> number of uniques: 4
Values:
['Married' 'Single' 'Unknown' 'Divorced']
____________________________________________________________

Income_Category >> number of uniques: 6
Values:
['60K_to_80K' 'Less_than_40K' '80K_to_120K' '40K_to_60K' 'Above_120K'
 'Unknown']
____________________________________________________________

Card_Category >> number of uniques: 4
Values:
['Blue' 'Gold' 'Silver' 'Platinum']
____________________________________________________________

Months_on_book >> number of uniques: 44
Values:
[39 44 36 34 21 46 27 31 54 30 48 37 56 42 49 33 28 38 41 43 45 52 40 50
 35 47 32 20 29 25 53 24 55 23 22 26 13 51 19 15 17 18 16 14]
____________________________________________________________

Total_Relationship_Count >> number of uniques: 6
Values:
[5 6 4 3 2 1]
____________________________________________________________

Months_Inactive_12_mon >> number of uniques: 7
Values:
[1 4 2 3 6 0 5]
____________________________________________________________

Contacts_Count_12_mon >> number of uniques: 7
Values:
[3 2 0 1 4 5 6]
____________________________________________________________

Credit_Limit >> number of uniques: 6205, showing top 50 values
Top 50 Values:
[12691.   8256.   3418.   3313.   4716.   4010.  34516.  29081.  22352.
 11656.   6748.   9095.  11751.   8547.   2436.   4234.  30367.  13535.
  3193.  14470.  20979.   1438.3  4470.   2492.  12217.   7768.  14784.
 10215.  10100.   4785.   2753.   2451.   8923.   2650.  12555.   3520.
  3035.  15433.   3672.   7882.  32426.   6205.  17304.   3906.   9830.
```

```
  2283.   2548.  19458.   4745.   2622. ]

____________________________________________________________

Total_Revolving_Bal >> number of uniques: 1974, showing top 50 values
Top 50 Values:
[ 777  864    0 2517 1247 2264 1396 1677 1467 1587 1666  680  972 2362
 1291 1157 1800 1560 1669 1374 1010 1362 1811 1690 1490 1696 1914 2298
  886  605  578 2204 2055 1430 2020 1435 1227 1549  808 2179 2200 2363
 1880  978 1753 2016 1251 2102 1634 1515]

____________________________________________________________

Avg_Open_To_Buy >> number of uniques: 6813, showing top 50 values
Top 50 Values:
[11914.   7392.   3418.    796.   4716.   2763.  32252.  27685.  19835.
  9979.   5281.   7508.  11751.   6881.   1756.   3262.  28005.  12244.
   676.  13313.  19179.   1438.3  3790.    932.  12217.   6099.  13410.
  9205.  10100.   3423.    942.    761.   6406.   1160.  10859.   1606.
   737.  15433.   2786.   7277.  31848.   4001.  14787.   3906.   7775.
 34516.    853.    528.  18023.   3518. ]

____________________________________________________________

Total_Amt_Chng_Q4_Q1 >> number of uniques: 1158, showing top 50 values
Top 50 Values:
[1.335 1.541 2.594 1.405 2.175 1.376 1.975 2.204 3.355 1.524 0.831 1.433
 3.397 1.163 1.19  1.707 1.708 0.653 1.831 0.966 0.906 1.047 1.608 0.573
 1.075 0.797 0.921 0.843 0.525 0.739 0.977 1.323 1.726 1.75  0.519 0.51
 1.724 0.865 1.32  1.052 1.042 0.803 1.449 1.214 1.621 2.316 2.357 0.787
 0.624 1.321]

____________________________________________________________

Total_Trans_Amt >> number of uniques: 5033, showing top 50 values
Top 50 Values:
[1144 1291 1887 1171  816 1088 1330 1538 1350 1441 1201 1314 1539 1311
 1570 1348 1671 1028 1336 1207 1178  692  931 1126 1110 1051 1197 1904
 1052 1045 1038 1596 1589 1411 1407 1877  966 1464  704 1109 1347 1756
 1042 1444 1741 1719 1217 1140 1878  705]

____________________________________________________________

Total_Trans_Ct >> number of uniques: 126, showing top 50 values
Top 50 Values:
[42 33 20 28 24 31 36 32 26 17 29 27 21 30 16 18 23 22 40 38 25 43 37 19
 35 15 41 57 12 14 34 44 13 47 10 39 53 50 52 48 49 45 11 55 46 54 60 51
 63 58]

____________________________________________________________

Total_Ct_Chng_Q4_Q1 >> number of uniques: 830, showing top 50 values
Top 50 Values:
[1.625 3.714 2.333 2.5   0.846 0.722 0.714 1.182 0.882 0.68  1.364 3.25
 2.    0.611 1.7   0.929 1.143 0.909 0.6   1.571 0.353 0.75  0.833 1.3
 1.    0.9   2.571 1.6   1.667 0.483 1.176 1.2   0.556 0.143 0.474 0.917
 1.333 0.588 0.8   1.923 0.25  0.364 1.417 1.083 1.25  0.5   1.154 0.733
 0.667 2.4  ]

____________________________________________________________

Avg_Utilization_Ratio >> number of uniques: 964, showing top 50 values
Top 50 Values:
[0.061 0.105 0.    0.76  0.311 0.066 0.048 0.113 0.144 0.217 0.174 0.195
 0.279 0.23  0.078 0.095 0.788 0.08  0.086 0.152 0.626 0.215 0.093 0.099
 0.285 0.658 0.69  0.282 0.562 0.135 0.544 0.757 0.241 0.077 0.018 0.355
 0.145 0.209 0.793 0.074 0.259 0.591 0.687 0.127 0.667 0.843 0.422 0.156
 0.525 0.587]

____________________________________________________________
```

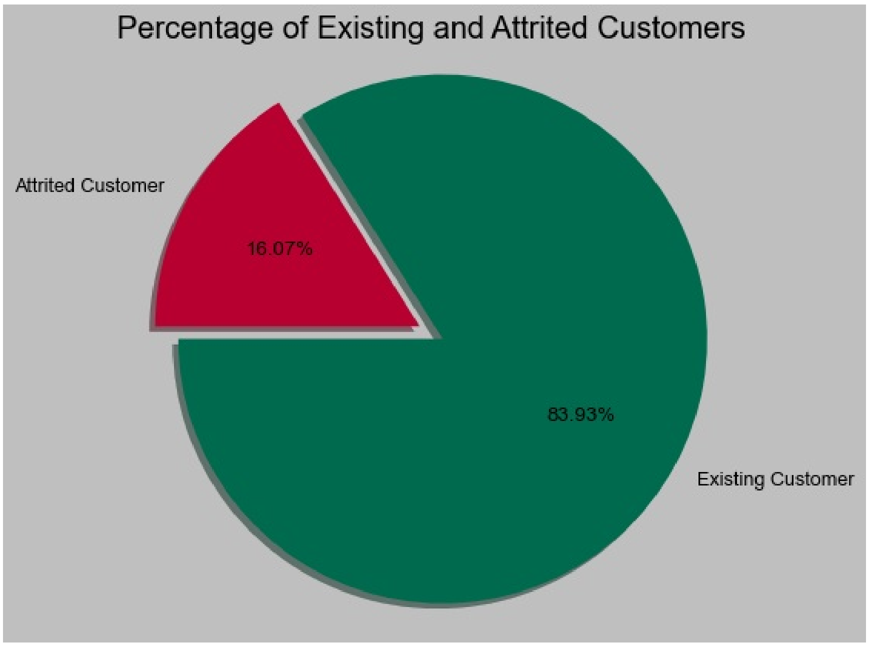
No null values to deal with. Features have the correct data type.

comment

EDA

In [12]:

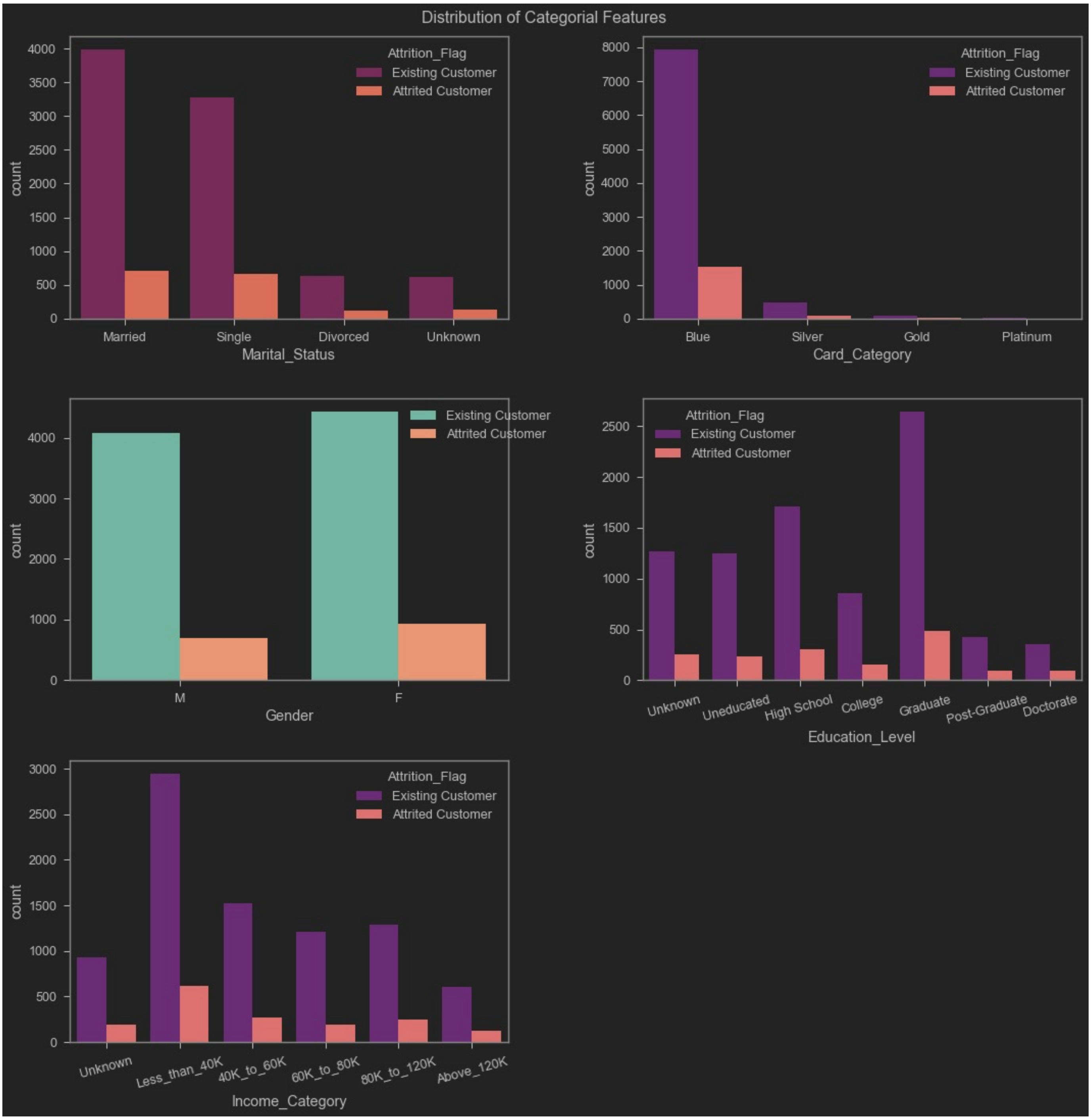
```python
with plt.style.context('grayscale'): # seaborn-deep
    plt.pie([
        df.Attrition_Flag[df.Attrition_Flag == 'Existing Customer'].count(),
        df.Attrition_Flag[df.Attrition_Flag == 'Attrited Customer'].count()
    ],
            labels=['Existing Customer', 'Attrited Customer'],
            colors=['#006a4e', '#b70030'],
            explode=[0.1, 0],
            autopct='%1.2f%%',
            shadow=True,
            startangle=180)
    plt.title("Percentage of Existing and Attrited Customers", size=20)
    plt.axis('equal')
    plt.show()
```



In [13]:

```python
plt.figure(figsize=(15, 15))
plt.subplot(3, 2, 1)
sns.countplot(x=df['Marital_Status'],
              hue=df['Attrition_Flag'],
              palette='rocket',
              order=['Married', 'Single', 'Divorced', 'Unknown'])
plt.subplot(3, 2, 2)
sns.countplot(x=df['Card_Category'],
              hue=df['Attrition_Flag'],
              palette='magma',
              order=['Blue', 'Silver', 'Gold', 'Platinum'])
plt.subplot(3, 2, 3)
sns.countplot(x=df['Gender'], hue=df['Attrition_Flag'], palette='Set2')
plt.legend(bbox_to_anchor=(.75, 1))
plt.subplot(3, 2, 4)
sns.countplot(x=df['Education_Level'],
              hue=df['Attrition_Flag'],
              palette='magma',
              order=[
                  'Unknown', 'Uneducated', 'High School', 'College',
                  'Graduate', 'Post-Graduate', 'Doctorate'
```

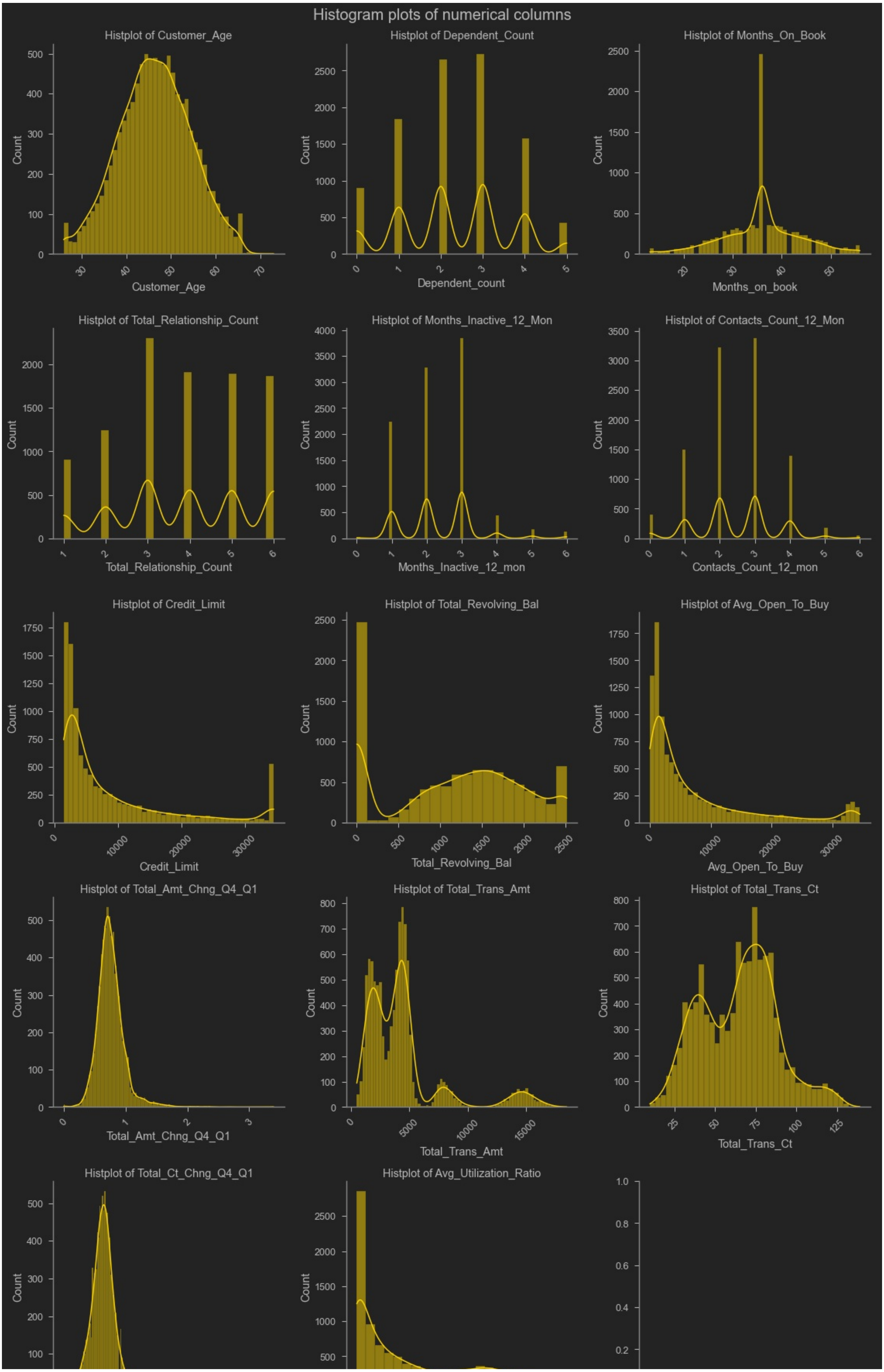
```
                ])
plt.xticks(rotation=15)
plt.subplot(3, 2, 5)
sns.countplot(x=df['Income_Category'],
              hue=df['Attrition_Flag'],
              palette='magma',
              order=[
                  'Unknown', 'Less_than_40K', '40K_to_60K', '60K_to_80K',
                  '80K_to_120K', 'Above_120K'
              ])
plt.xticks(rotation=15)
plt.tight_layout()
plt.suptitle(f'Distribution of Categorial Features \n', va='bottom')
plt.show()
```

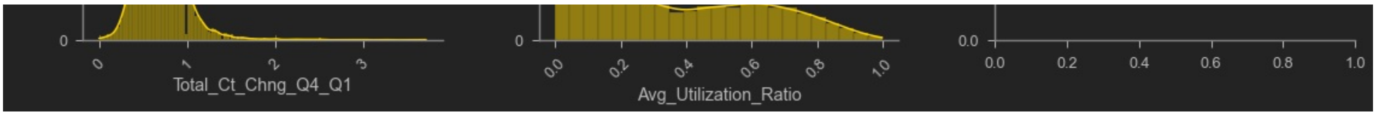


comment

In [14]:
```
fn.plot_distribution(df.select_dtypes('number'),
```

```
                               plot_title='Histogram plots of numerical columns')
```





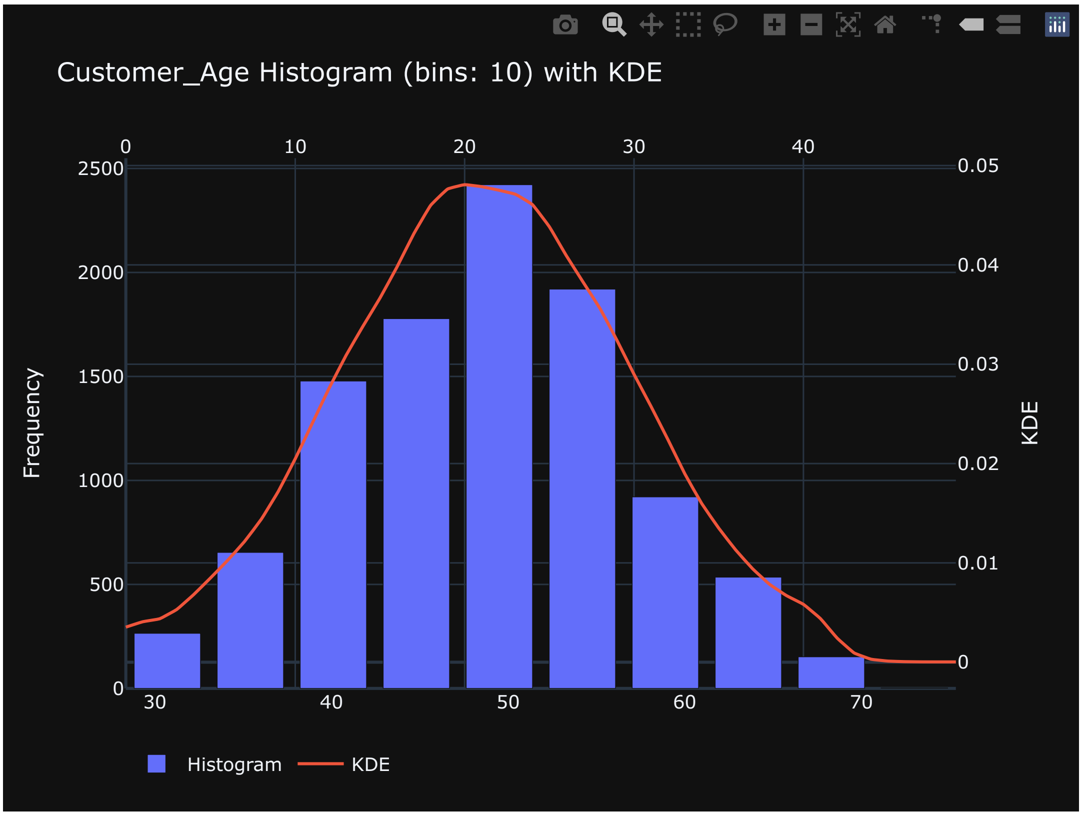
comment

In [15]:
```
print(f'Minimum customer age: {df.Customer_Age.unique().min()}')
print(f'Maximum customer age: {df.Customer_Age.unique().max()}')
```

```
Minimum customer age: 26
Maximum customer age: 73
```

In [16]:
```
s = df[~pd.isnull(df['Customer_Age'])][['Customer_Age']]
chart, labels = np.histogram(s, bins=10)
kde = sts.gaussian_kde(s['Customer_Age'])
kde_data = kde.pdf(np.linspace(labels.min(), labels.max()))
# main statistics
stats = df['Customer_Age'].describe().to_frame().T
charts = [
    go.Bar(x=labels[1:], y=chart, name='Histogram'),
    go.Scatter(x=list(range(len(kde_data))),
               y=kde_data,
               name='KDE',
               yaxis='y2',
               xaxis='x2',
               line={
                   'shape': 'spline',
                   'smoothing': 0.3
               },
               mode='lines')
]
figure = go.Figure(
    data=charts,
    layout=go.Layout(
        {
            'barmode': 'group',
            'legend': {
                'orientation': 'h'
            },
            'title': {
                'text': 'Customer_Age Histogram (bins: 10) with KDE'
            },
            'xaxis2': {
                'anchor': 'y',
                'overlaying': 'x',
                'side': 'top'
            },
            'yaxis': {
                'side': 'left',
                'title': {
                    'text': 'Frequency'
                }
            },
```

```
            'yaxis2': {
                'overlaying': 'y',
                'side': 'right',
                'title': {
                    'text': 'KDE'
                }
            }
        },
        template='plotly_dark'))
figure.show()
```



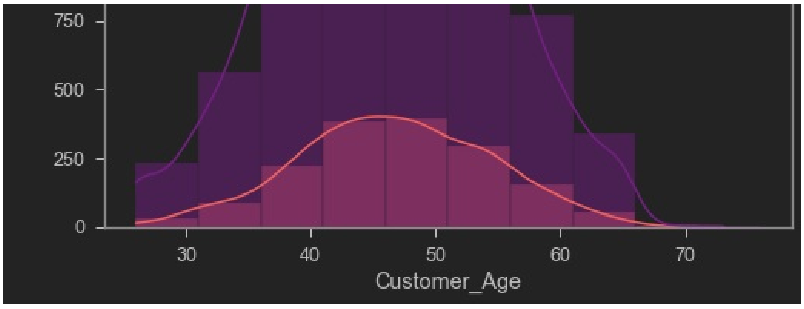
comment

In [17]:

```
# pairweise eda
```

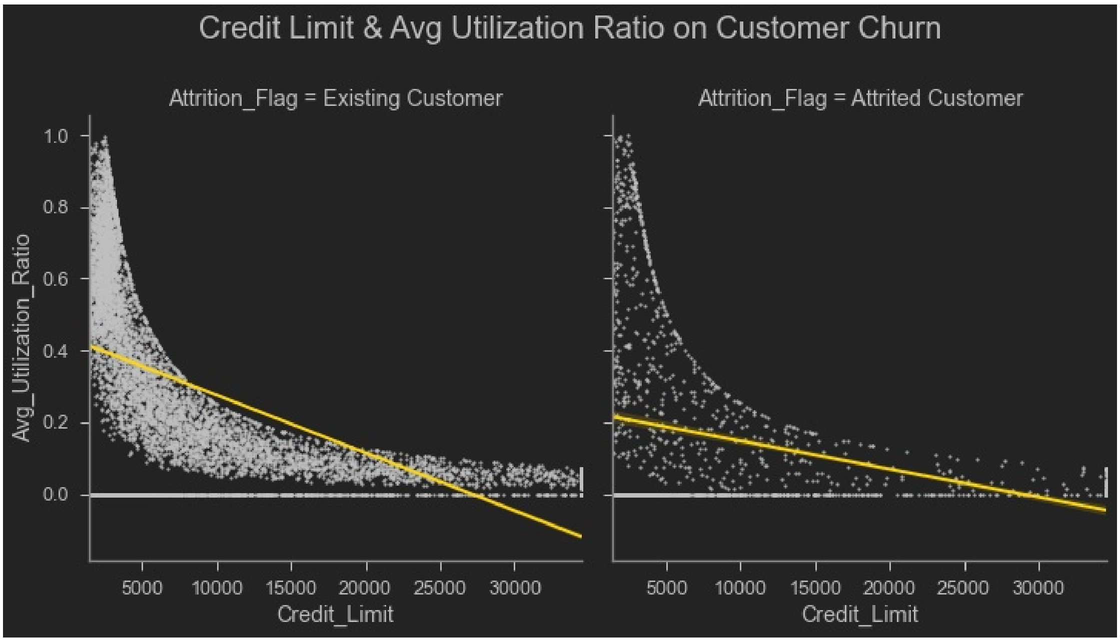
In [18]:

```
sns.histplot(x=df.Customer_Age,
             hue=df.Attrition_Flag,
             kde=True,
             binwidth=5,
             palette='magma')
plt.title('Customer Age & Churn')
plt.show()
```



In [19]:

```
sns.lmplot(x='Credit_Limit',
           y='Avg_Utilization_Ratio',
           data=df,
           col='Attrition_Flag',
           palette='Set2',
           scatter_kws={
               "s": 5,
               "color": 'silver'
           },
           line_kws={
               'lw': 2,
               'color': 'gold'
           })
plt.suptitle('Credit Limit & Avg Utilization Ratio on Customer Churn \n',
             va='bottom',
             fontsize=20)
plt.show()
```

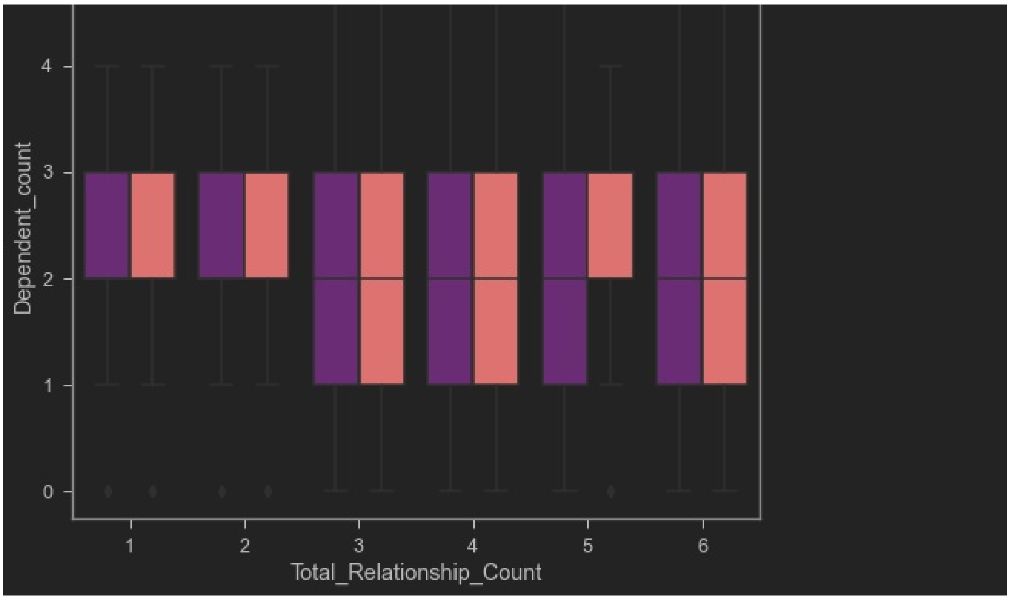


In [20]:

```
sns.boxplot(y='Dependent_count',
            x='Total_Relationship_Count',
            data=df,
            hue='Attrition_Flag',
            palette='magma')
plt.legend(bbox_to_anchor=(1, 1), loc="upper left")
```

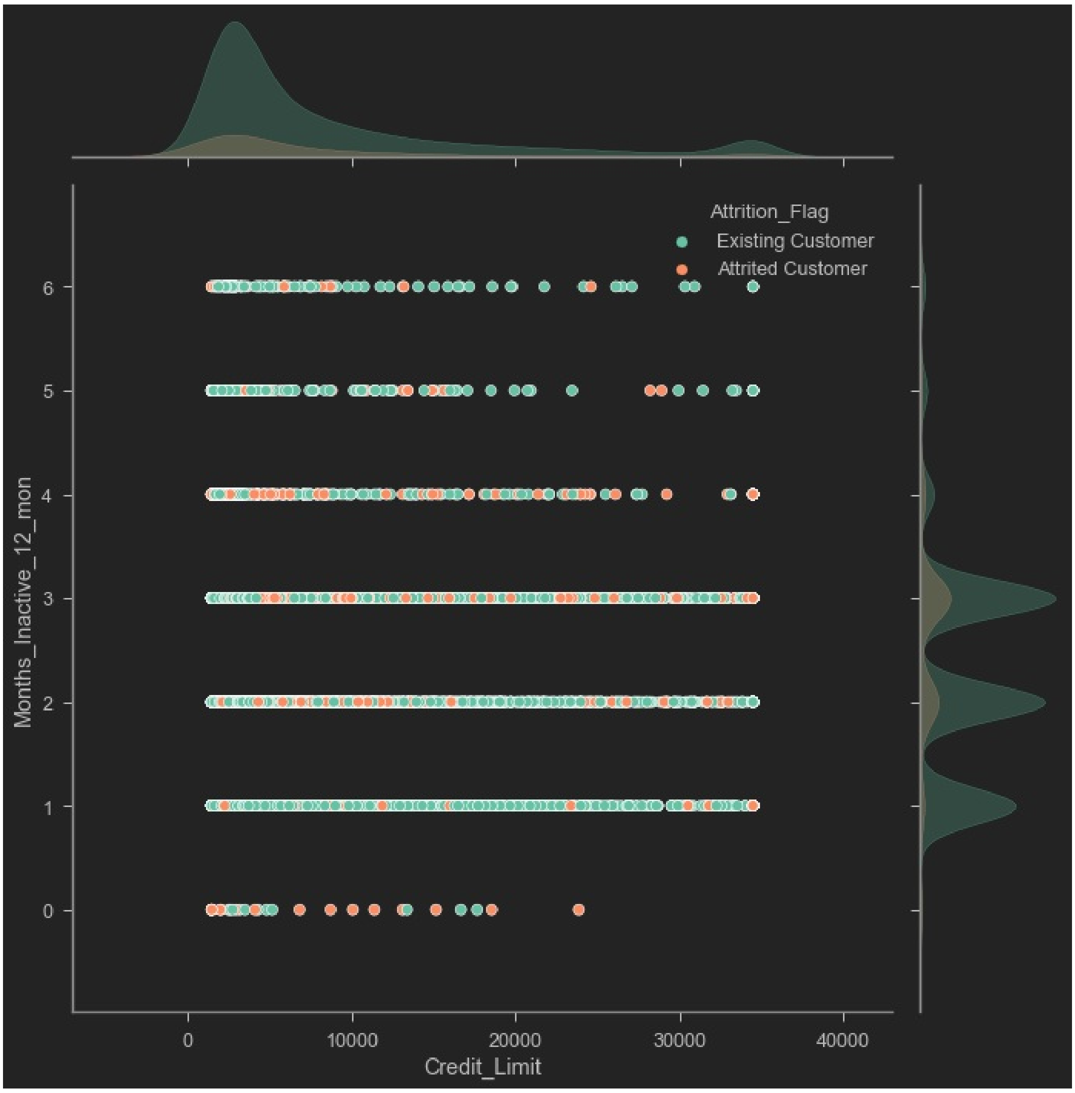
Out[20]:

```
<matplotlib.legend.Legend at 0x1ed60f3e8e0>
```



In [21]:

```
g = sns.jointplot(data=df,
                  x='Credit_Limit',
                  y='Months_Inactive_12_mon',
                  hue='Attrition_Flag',
                  palette='Set2',
                  height=10)
```



In [22]:

```
# sns.pairplot(df, hue="Attrition_Flag")
```

SCRUB

In [23]:
```
(df.Attrition_Flag.value_counts(1)*100).round(2)
```

Out[23]:
```
Existing Customer    83.93
Attrited Customer    16.07
Name: Attrition_Flag, dtype: float64
```

Class imbalance will be addressed by synthetic oversampling later in this section.

Label encoding

In [24]:
```
# ML friendly labels
churn_map = {'Existing Customer':0, 'Attrited Customer':1}
```

In [25]:
```
X = df.drop(columns='Attrition_Flag').copy()
y = df.Attrition_Flag.map(churn_map).copy()
```

Train-Test split

In [26]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.8)
```

Encoding & Scaling

Pipeline

In [27]:
```
# isolating numerical cols
nume_col = list(X.select_dtypes('number').columns)
# isolating categorical cols
cate_col = list(X.select_dtypes('object').columns)
# pipeline for processing categorical features
pipe_cate = Pipeline([('ohe', OneHotEncoder(sparse=False, drop=None))])
# pipeline for processing numerical features
pipe_nume = Pipeline([('scaler', StandardScaler())])
# transformer
preprocessor = ColumnTransformer([('nume_feat', pipe_nume, nume_col),
                                  ('cate_feat', pipe_cate, cate_col)])
# creating dataframes
# X_train
X_train_pr = pd.DataFrame(preprocessor.fit_transform(X_train),
                          columns=nume_col +
                          list(preprocessor.named_transformers_['cate_feat'].
                               named_steps['ohe'].get_feature_names(cate_col)))
# X_test
X_test_pr = pd.DataFrame(preprocessor.transform(X_test),
                         columns=nume_col +
                         list(preprocessor.named_transformers_['cate_feat'].
                              named_steps['ohe'].get_feature_names(cate_col)))
```

In [28]:
```
# # preprocessor, nume_col, cate_col are saved
# joblib.dump(preprocessor, filename='./model/preprocessor.joblib', compress=9)
# joblib.dump(nume_col, filename='./model/nume_col.joblib', compress=9)
# joblib.dump(cate_col, filename='./model/cate_col.joblib', compress=9)
```

SMOTENC

In [29]:
```
X_train_pr
```

Out[29]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_Li
0	-0.917719	-1.037085	-0.876565	-0.524060	-1.332205	-0.417338	0.104
1	-2.168045	-1.806447	-2.139626	0.768893	1.654565	1.397887	-0.631
2	-0.292556	0.501638	-0.245035	0.768893	0.658975	-1.324951	0.030
3	-1.792947	-1.037085	-2.139626	-1.170537	0.658975	2.305499	-0.551
4	1.207836	-1.806447	0.007577	0.768893	-0.336615	0.490274	-0.472
...	...	...	...	...	...	...	
8096	0.832738	-0.267724	1.649557	0.122416	-0.336615	0.490274	-0.288
8097	0.332607	-1.037085	-0.623953	0.768893	-0.336615	-1.324951	-0.327
8098	-1.542882	1.270999	-0.876565	1.415370	-0.336615	-0.417338	2.394
8099	0.082542	0.501638	0.639108	0.122416	-1.332205	1.397887	-0.230
8100	1.707966	-1.037085	0.007577	-1.170537	0.658975	-1.324951	-0.518

8101 rows × 37 columns

In [30]:
```
smotenc_features = [False] * len(nume_col) + [True] * (
    len(X_train_pr.columns) - len(nume_col))
```

In [31]:
```
oversampling = SMOTENC(categorical_features=smotenc_features, n_jobs=-1)
```

In [32]:
```
X_train_pr_os, y_train_encoded_os = oversampling.fit_sample(
                                                        X_train_pr, y_train)
```

MODEL

Segmentation

In [33]:
```
# # lable Encoding
# Education_Level_map = {
#     'High School': 2,
#     'Graduate': 4,
#     'Uneducated': 0,
#     'Unknown': 1,
#     'College': 3,
#     'Post-Graduate': 5,
#     'Doctorate': 6
# }
# Income_Category_map = {
#     '60K_to_80K': 3,
#     'Less_than_40K': 1,
#     '80K_to_120K': 4,
#     '40K_to_60K': 3,
#     'Above_120K': 5,
#     'Unknown': 0
# }
# Card_Category_map = {'Blue': 0, 'Gold': 2, 'Silver': 1, 'Platinum': 3}
```

```
# # OHE
# Marital_Status_map = {'Married': 2, 'Single': 1, 'Unknown': 0, 'Divorced': 3}
# Gender_map = {'M': 1, 'F': 0}

# X.Education_Level = X.Education_Level.map(Education_Level_map)
# X.Income_Category = X.Income_Category.map(Income_Category_map)
# X.Card_Category = X.Card_Category.map(Card_Category_map)

# X.Marital_Status = X.Marital_Status.map(Marital_Status_map)
# X.Gender = X.Gender.map(Gender_map)
# display("X",X)

# seg_scaler = StandardScaler()
# seg_scaler.fit(X)
# X_segmentation = pd.DataFrame(seg_scaler.transform(X),columns=X.columns)
# display("X_segmentation", X_segmentation)

# # NOTE: Tried diffrent versions of the dataset for modeling,
# # - Diffrent encoding
# # - Diffrent scaler
# # performance is mostly indiffrent
```

In [34]:

```
X_segmentation = fn.dataset_processor_segmentation(X, verbose=2)
```

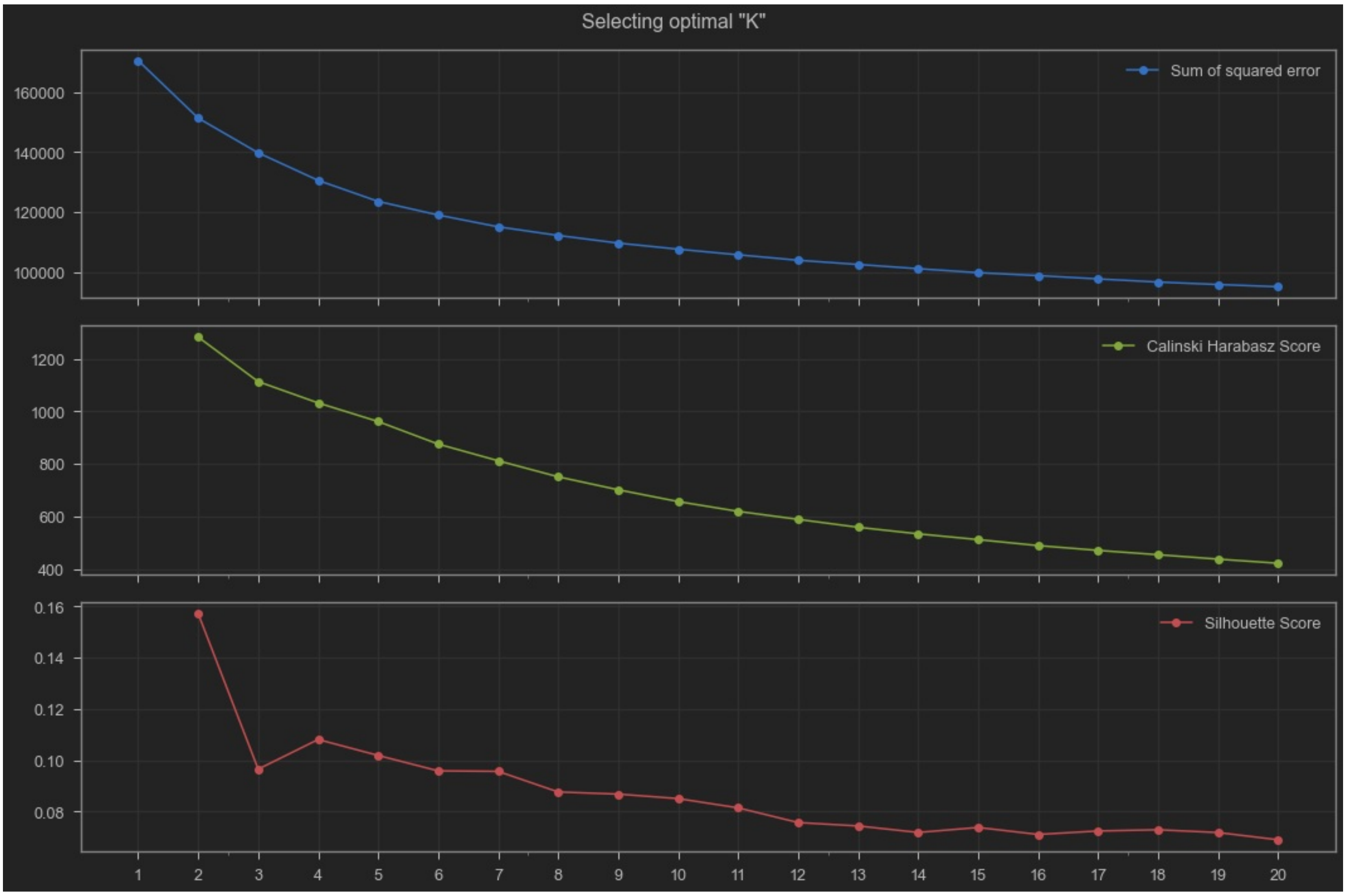
```
Numerical columns:
---------------------
 ['Customer_Age', 'Dependent_count', 'Months_on_book', 'Total_Relationship_Count', 'Months_Inactive_12_mon', 'Con
tacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal', 'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Tra
ns_Amt', 'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio']

Categorical columns:
---------------------
 ['Gender', 'Education_Level', 'Marital_Status', 'Income_Category', 'Card_Category']


------
Scaler: StandardScaler, settings: {'copy': True, 'with_mean': True, 'with_std': True}
Encoder: OneHotEncoder, settings: {'categories': 'auto', 'drop': None, 'dtype': <class 'numpy.float64'>, 'handle_
unknown': 'error', 'sparse': False}
------
```

Finding "K"

In [246…

```
search_range = range(1, 21)
report = {}
for k in search_range:
    temp_dict = {}
    kmeans = KMeans(init='k-means++',
                    algorithm='auto',
                    n_clusters=k,
                    max_iter=1000,
                    random_state=1,
                    verbose=0).fit(X_segmentation)
    inertia = kmeans.inertia_
    temp_dict['Sum of squared error'] = inertia
    try:
        cluster = kmeans.predict(X_segmentation)
        chs = metrics.calinski_harabasz_score(X_segmentation, cluster)
```

```
        ss = metrics.silhouette_score(X_segmentation, cluster)
        temp_dict['Calinski Harabasz Score'] = chs
        temp_dict['Silhouette Score'] = ss
        report[k] = temp_dict
    except:
        report[k] = temp_dict

report_df = pd.DataFrame(report).T
report_df.plot(figsize=(15, 10),
               xticks=search_range,
               grid=True,
               title=f'Selecting optimal "K"',
               subplots=True,
               marker='o',
               sharex=True)
plt.tight_layout()
```



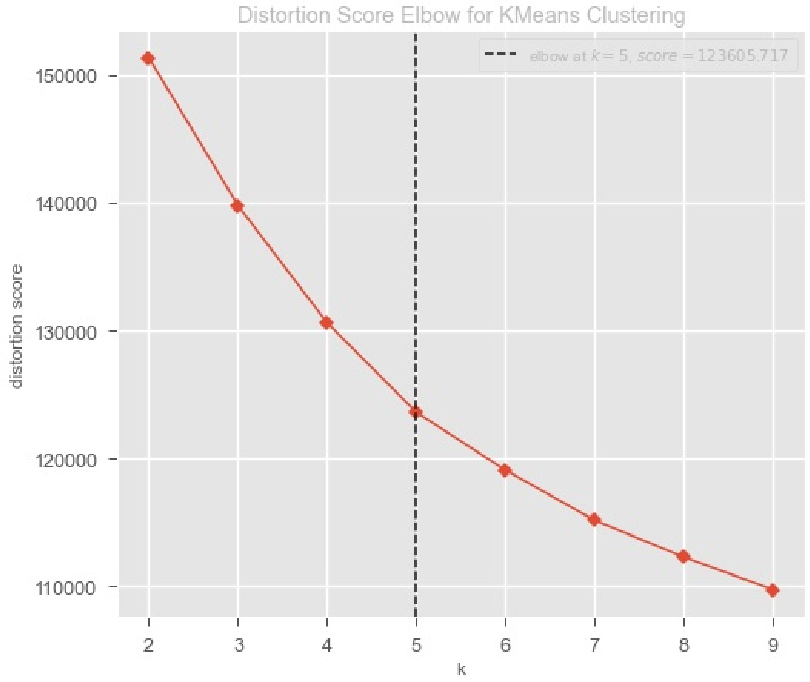
Higher Silhouette Coefficient score relates to a model with better defined clusters. And higher Calinski-Harabasz score relates to a model with better defined clusters.

Although by looking at the visual no obvious optimal K can not be spotted. Based on the `Silhouette Score` and `Sum of squared error` (a.k.a. Elbow plot), 5 segmentation seemed optimal for initial model. `Calinski Harabasz Score` also supports this segmentation.

Customers are segmented by 5 groups by their characteristics.

In [36]:

```
# using yellowbrick to get an idea about the choice of K=5
with plt.style.context('ggplot'):
    kelbow_visualizer(KMeans(random_state=1),
                      X_segmentation,
                      k=(2, 10),
                      timings=False)
```



Among models run for `K` from a range of 2 to 10, 5 is recommended by yellowbrick package.

In [248…

```
# using MeanShift to get an estimate
bandwidth = estimate_bandwidth(X_segmentation, quantile=0.3, n_jobs=-1)
ms = MeanShift(bandwidth=bandwidth, bin_seeding=False, n_jobs=-1, max_iter=500)
ms.fit(X_segmentation)
labels = ms.labels_
cluster_centers = ms.cluster_centers_
labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)
print(f"Number of estimated clusters : {n_clusters_}")
```

```
Number of estimated clusters : 5
```

Result of MeanShift supports the initial choice of K=5.

Selecting "K"

In [35]:

```
n_clusters=5
```

In [36]:

```
kmeans = KMeans(
    init='k-means++',
    algorithm='auto',
    n_clusters=n_clusters,
    max_iter=1000,
    random_state=1,  # selecting random_state=1 for reproducibility
    verbose=0).fit(X_segmentation)
```

In [37]:

```
clusters = kmeans.predict(X_segmentation)
cluster_df = X_segmentation.copy()
cluster_df['Clusters'] = clusters
cluster_df
```

Out[37]:

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_l
0	-0.165406	0.503368	0.384621	0.763943	-1.327136	0.492404	0.44

1	0.333570	2.043199	1.010715	1.407306	-1.327136	-0.411616	-0.04
2	0.583058	0.503368	0.008965	0.120579	-1.327136	-2.219655	-0.57
3	-0.789126	1.273283	-0.241473	-0.522785	1.641478	-1.315636	-0.58
4	-0.789126	0.503368	-1.869317	0.763943	-1.327136	-2.219655	-0.43
...	...	...	...	...	...	...	
10122	0.458314	-0.266547	0.509840	-0.522785	-0.337598	0.492404	-0.50
10123	-0.664382	-0.266547	-1.368442	0.120579	-0.337598	0.492404	-0.47
10124	-0.290150	-1.036462	0.008965	0.763943	0.651940	1.396424	-0.35
10125	-2.036565	-0.266547	0.008965	0.120579	0.651940	0.492404	-0.36
10126	-0.414894	-0.266547	-1.368442	1.407306	-0.337598	1.396424	0.19

10127 rows × 38 columns

In [38]:

```
# distribution of classes (%)
(cluster_df.Clusters.value_counts(1)*100).round(2)
```

Out[38]:

```
2    30.23
1    26.83
3    19.75
4    13.55
0     9.65
Name: Clusters, dtype: float64
```

In [44]:

```
@interact(df=fixed(cluster_df),
          x=cluster_df.columns,
          y=cluster_df.columns,
          z=cluster_df.columns)
def plot_segments(df=cluster_df,
                  x='Customer_Age',
                  y='Months_on_book',
                  z='Credit_Limit'):
    df['Clusters'] = df['Clusters'].astype('str')
    fig = px.scatter_3d(
        df,
        x=x,
        y=y,
        z=z,
        title=
        f'{x.replace("_", " ")}, {y.replace("_", " ")} and, {z.replace("_", " ")} by 
Clusters',
        color='Clusters',
        template='plotly_dark')
    fig.update_traces(marker=dict(size=2))
    df['Clusters'] = df['Clusters'].astype('int')
    fig.show()
```

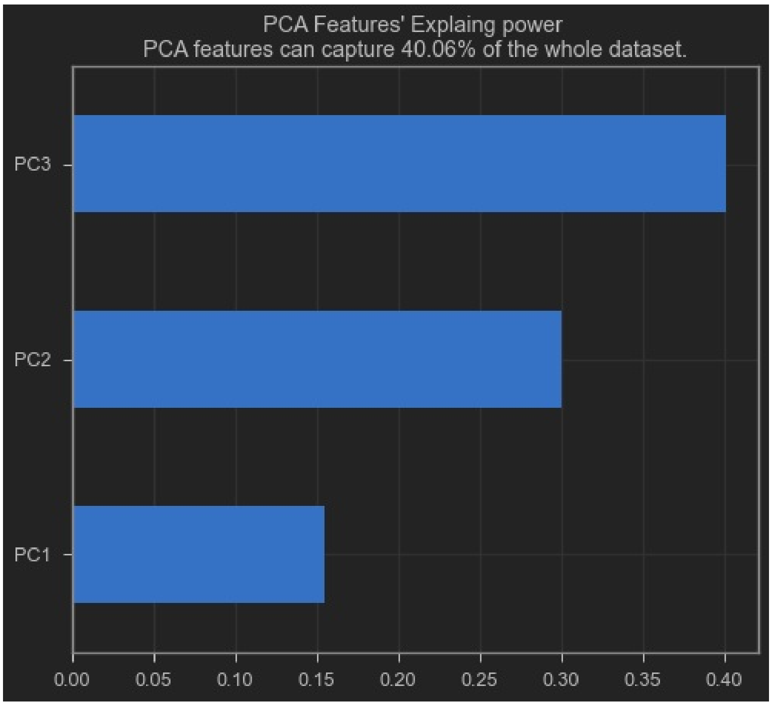
More insights on the segmentation is in the INTERPRET part of this analysis.

Using principal component analysis concept for reducing features to visualize the clusters in a three dimensional space.

In [40]:

```
pca = PCA(n_components=3)
pc_feature_names = [f"PC{x}" for x in range(1, pca.n_components + 1)]
pca_data = pca.fit_transform(cluster_df)
pca_df = pd.DataFrame(pca_data, columns=pc_feature_names)
pd.Series(pca.explained_variance_ratio_.cumsum(), index=pc_feature_names).plot(
```

```
    kind='barh',
    title=
    f"""PCA Features' Explaing power \nPCA features can capture
{((pca.explained_variance_ratio_.cumsum()[-1])*100).round(2)}% of the whole dataset."""
)
plt.grid()
plt.show()
pca_df['Clusters'] = clusters.astype('str')
fig = px.scatter_3d(pca_df,
                    x='PC1',
                    y='PC2',
                    z='PC3',
                    color='Clusters',
                    title='Cluster visualization with the help of PCA',
                    template='plotly_dark')
fig.update_traces(marker=dict(size=2))
fig.update_layout(width=700, height=500, bargap=0.05)
fig.show()
```

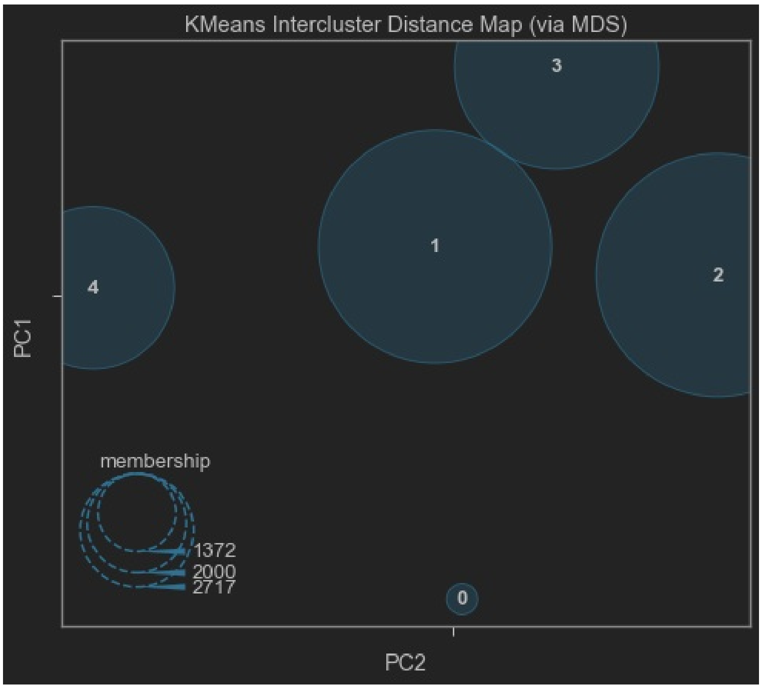


WebGL is not supported by your browser - visit https://get.webgl.org for more info

With only forty percent explainability of the entire dataset by PCA, the clusters exhibit a clear separation between them in a three dimensional space. I am content with the selected K of 5. This will be further evaluated when performing inter cluster exploration in later part.

In [255…

```
# Using two PC of 2
intercluster_distance(kmeans, X_segmentation, embedding='mds', random_state=12); # 'tsne'
```



Feature importance

Newly created `cluster_df` is used to get the feature importance to get insights which features were often used for determining the segmentation. A Random Forest model is used to get feature importance alongside a permutation importance analysis to get the most important features.

In [43]:

```
X_feat_imp = cluster_df.drop(columns='Clusters').copy()
y_feat_imp = cluster_df.Clusters.copy()
```

In [44]:

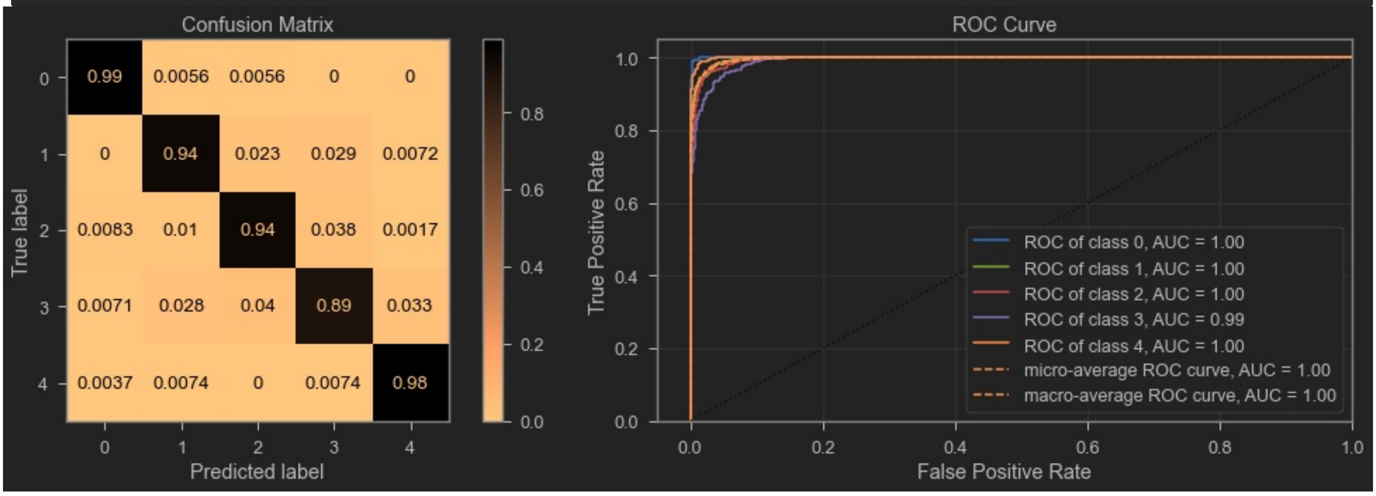
```
X_feat_imp_train, X_feat_imp_test, y_feat_imp_train, y_feat_imp_test = train_test_split(
    X_feat_imp, y_feat_imp, train_size=.8)
```

In [45]:

```
# Random Forest
clf_rf = RandomForestClassifier(
    n_jobs=-1,
    criterion='entropy',
    min_samples_leaf=5,
    min_samples_split=6,
    class_weight='balanced_subsample',
)
fn.model_report_multiclass(clf_rf,
                           X_feat_imp_train,
                           y_feat_imp_train,
                           X_feat_imp_test,
                           y_feat_imp_test,
                           show_train_report=False)
```

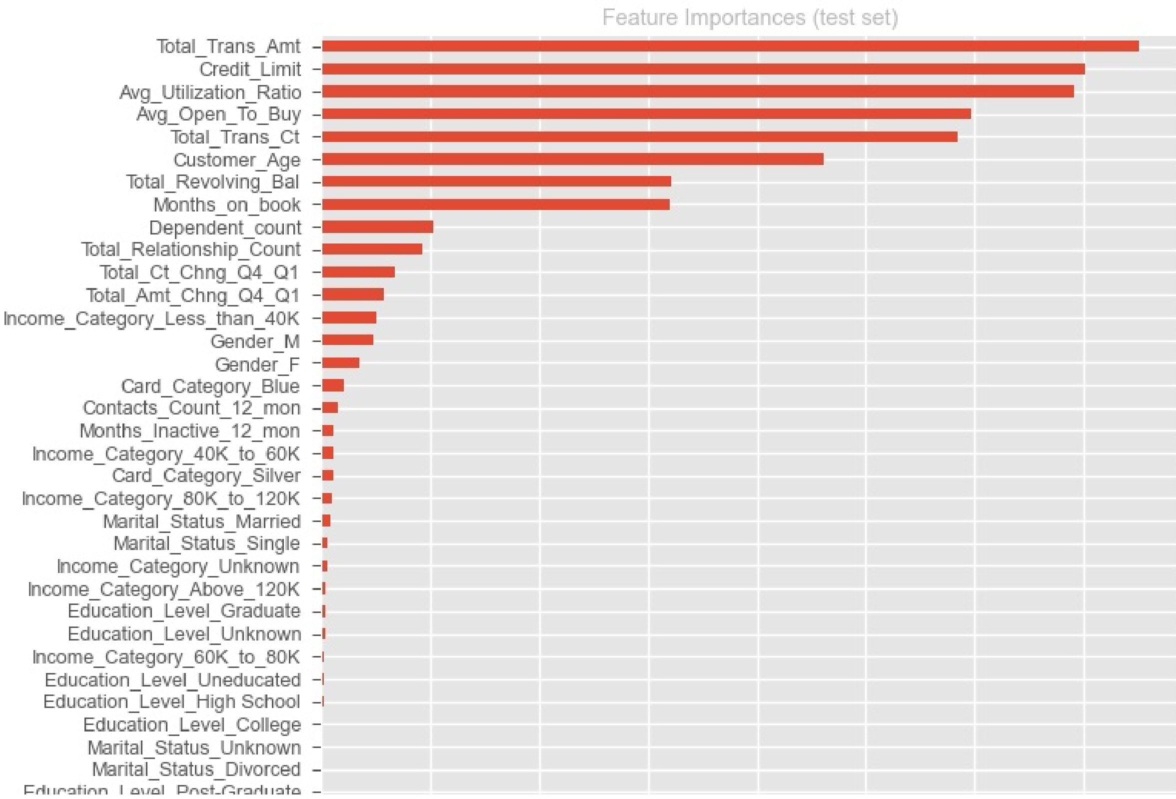
Report of RandomForestClassifier type model using train-test split dataset.

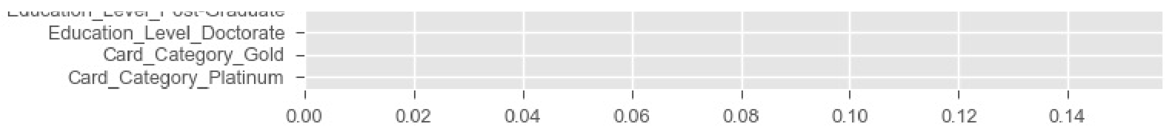
```
Train accuracy score: 0.9814
Test accuracy score: 0.9403
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
****************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       177
           1       0.96      0.94      0.95       554
           2       0.95      0.94      0.95       602
           3       0.90      0.89      0.90       423
           4       0.93      0.98      0.96       270

    accuracy                           0.94      2026
   macro avg       0.94      0.95      0.94      2026
weighted avg       0.94      0.94      0.94      2026

************************************************************
```



In [46]:

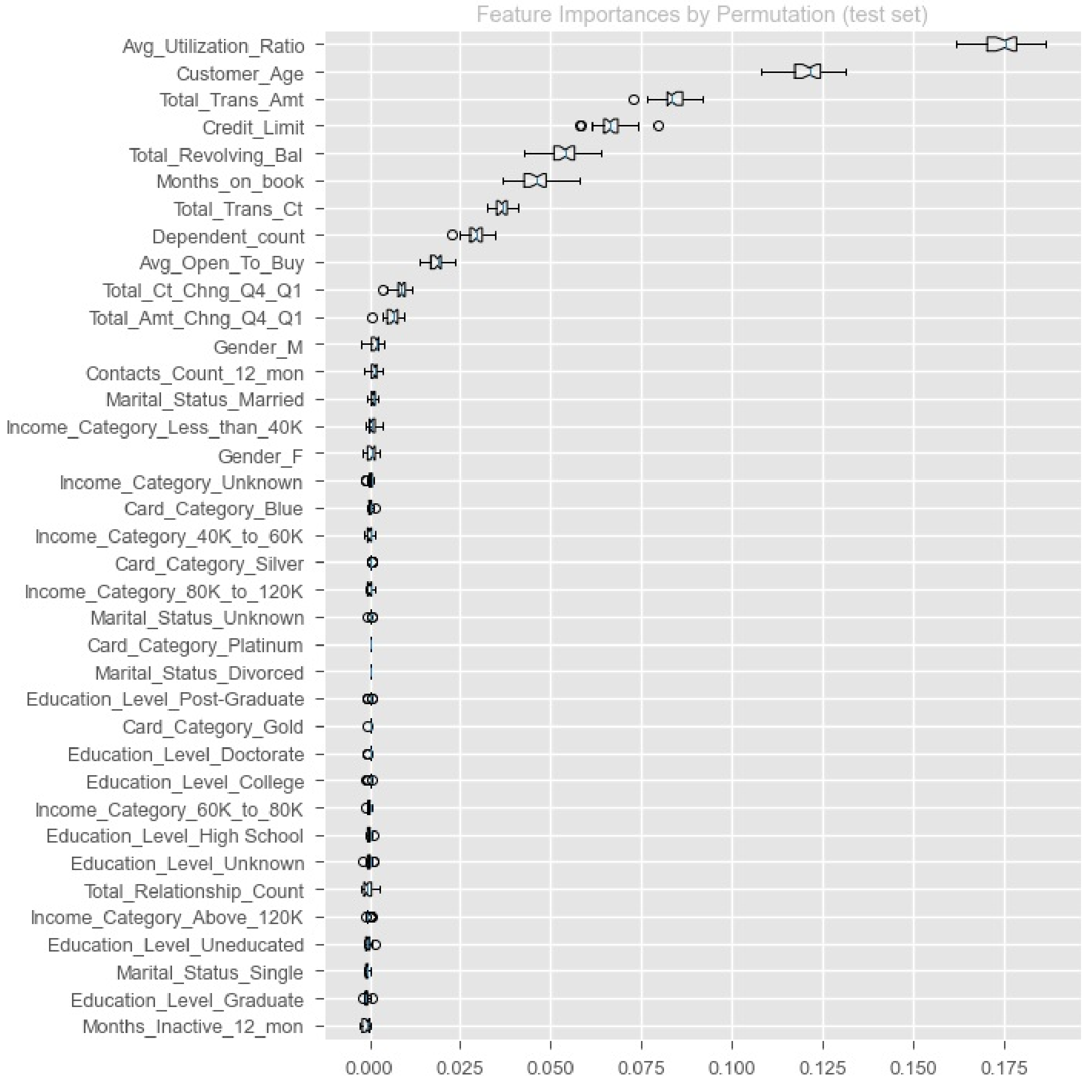
```python
# Feature Importance
with plt.style.context('ggplot'):
    pd.Series(clf_rf.feature_importances_,
          index=X_feat_imp_test.columns).sort_values().plot(kind='barh',
                                                          figsize=(10, 10))
    plt.title('Feature Importances (test set)')
```





In [47]:

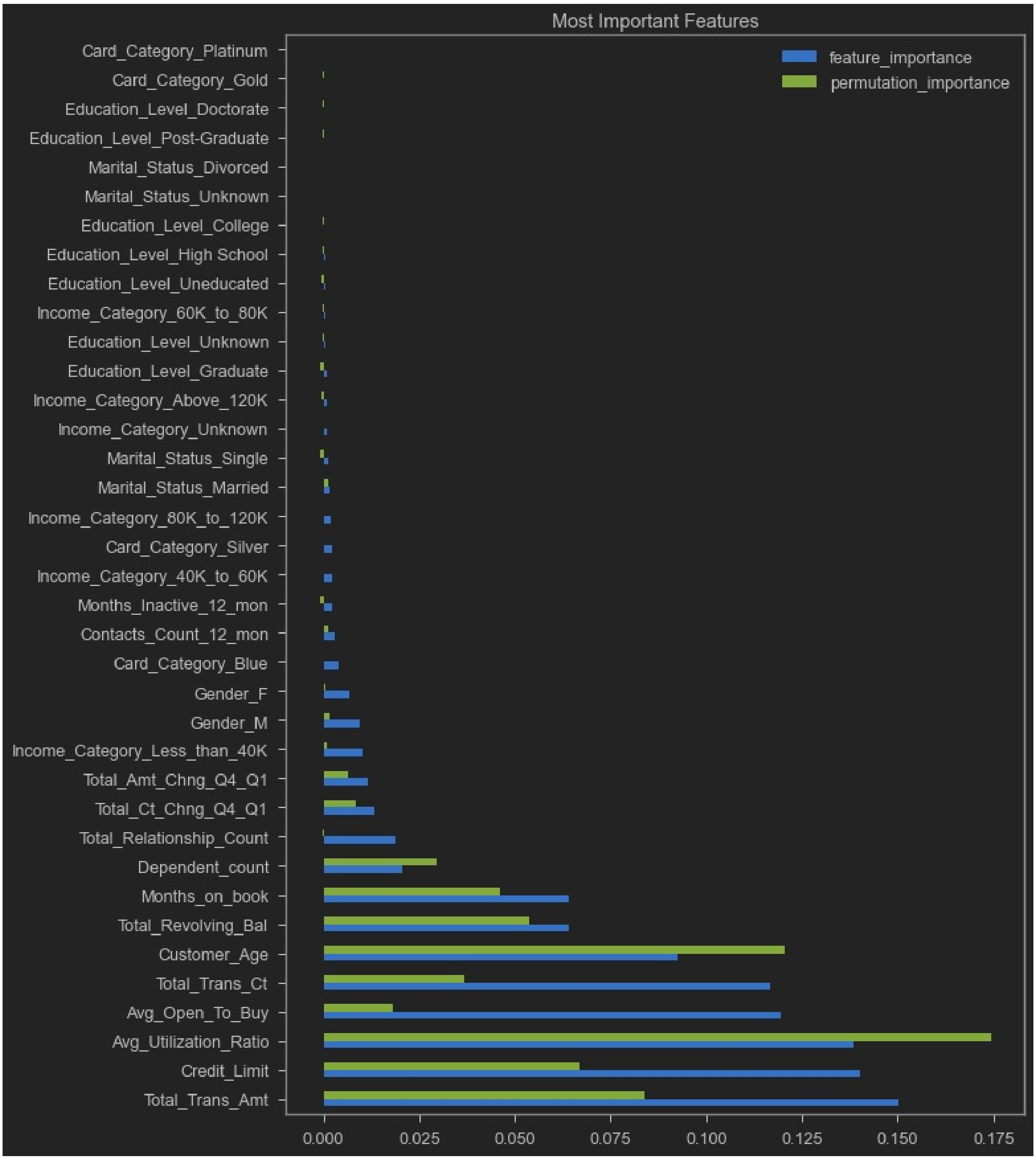
```
# Permutation Feature Importance
result = permutation_importance(clf_rf,
                                X_feat_imp_test,
                                y_feat_imp_test,
                                n_repeats=30,
                                random_state=42,
                                n_jobs=-1)
sorted_idx = result.importances_mean.argsort()
with plt.style.context('ggplot'):
    fig, ax = plt.subplots(figsize=(10, 10))
    ax.boxplot(result.importances[sorted_idx].T,
               notch=True,
               vert=False,
               labels=X_feat_imp_test.columns[sorted_idx])
    ax.set_title("Feature Importances by Permutation (test set)")
    fig.tight_layout()
    plt.show()
```



In [48]:

```
# fatures from the model
feature_importance = pd.Series(
```

```python
    clf_rf.feature_importances_,
    index=X_feat_imp_test.columns).sort_values(ascending=False)

permutation_importance = pd.DataFrame(
    result.importances[sorted_idx].T,
    columns=X_feat_imp_test.columns[sorted_idx]).mean().sort_values(
        ascending=False)

important_features = pd.DataFrame([feature_importance,
                                   permutation_importance]).T
important_features.columns = ['feature_importance', 'permutation_importance']
important_features.plot(kind='barh',
                        figsize=(10, 15),
                        title="Most Important Features")
plt.show()
```



In [49]:
```python
top_most_features = list(important_features[:10].index)
top_most_features
```

Out[49]:
```
['Total_Trans_Amt',
```

```
'Credit_Limit',
'Avg_Utilization_Ratio',
'Avg_Open_To_Buy',
'Total_Trans_Ct',
'Customer_Age',
'Total_Revolving_Bal',
'Months_on_book',
'Dependent_count',
'Total_Relationship_Count']
```

Segmentation Characteristics

In [46]:

```
characteristics_df = X.copy()
characteristics_df['target'] = y.copy()
characteristics_df['Clusters'] = cluster_df.Clusters
characteristics_df
```

Out[46]:

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Rel
0	45	M	3	High School	Married	60K_to_80K	Blue	39	
1	49	F	5	Graduate	Single	Less_than_40K	Blue	44	
2	51	M	3	Graduate	Married	80K_to_120K	Blue	36	
3	40	F	4	High School	Unknown	Less_than_40K	Blue	34	
4	40	M	3	Uneducated	Married	60K_to_80K	Blue	21	
...	...	...	...	...	...	...	...	...	
10122	50	M	2	Graduate	Single	40K_to_60K	Blue	40	
10123	41	M	2	Unknown	Divorced	40K_to_60K	Blue	25	
10124	44	F	1	High School	Married	Less_than_40K	Blue	36	
10125	30	M	2	Graduate	Unknown	40K_to_60K	Blue	36	
10126	43	F	2	Graduate	Married	Less_than_40K	Silver	25	

10127 rows × 21 columns

In [47]:

```
temp_char_dict = {}
for cluster in range(0, n_clusters):
    temp_df = characteristics_df.groupby(by='Clusters').get_group(cluster)
    temp_char_dict[cluster] = pd.Series(list(
        temp_df.value_counts().idxmax(axis=1)),
                                        index=list(temp_df.columns))
pd.DataFrame(temp_char_dict)
```

Out[47]:

	0	1	2	3	4
Customer_Age	27	26	26	44	26
Gender	F	F	F	F	F
Dependent_count	0	0	0	0	0
Education_Level	Graduate	Graduate	College	College	Uneducated
Marital_Status	Unknown	Single	Married	Married	Unknown
Income_Category	Less_than_40K	40K_to_60K	40K_to_60K	Less_than_40K	Unknown
Card_Category	Blue	Blue	Blue	Blue	Silver
Months_on_book	36	13	13	40	13
Total_Relationship_Count	1	5	2	5	1
Months_Inactive_12_mon	1	1	3	4	2
Contacts_Count_12_mon	2	2	3	2	3
Credit_Limit	4548.0	5655.0	2010.0	7499.0	34516.0
Total_Revolving_Bal	1450	0	1070	1083	2403
Avg_Open_To_Buy	3098.0	5655.0	940.0	6416.0	32113.0
Total_Amt_Chng_Q4_Q1	0.844	0.842	0.906	0.716	0.623
Total_Trans_Amt	14330	2312	3625	2478	4174

Total_Trans_Ct	131	61	85	45	59
Total_Ct_Chng_Q4_Q1	0.638	0.649	0.635	0.667	0.735
Avg_Utilization_Ratio	0.319	0.0	0.532	0.144	0.07
target	0	0	0	1	0
Clusters	0	1	2	3	4

In [48]:

```
cluster_names = [f"Cluster_{x}" for x in range(0, n_clusters)]
for cluster in cluster_names:
    print(f'{cluster}: \n Most occuring values:')
    cluster = characteristics_df.groupby(by='Clusters').get_group(
        int(cluster.split("_")[1]))
    print(cluster.value_counts().idxmax())
    display(cluster)
```

```
Cluster_0:
 Most occuring values:
(27, 'F', 0, 'Graduate', 'Unknown', 'Less_than_40K', 'Blue', 36, 1, 1, 2, 4548.0, 1450, 3098.0, 0.844, 14330, 131
, 0.638, 0.319, 0, 0)
```

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Rel
8271	40	M	2	College	Single	60K_to_80K	Blue	28	
8581	42	M	3	High School	Married	80K_to_120K	Blue	36	
8587	41	M	3	Graduate	Single	40K_to_60K	Blue	37	
8591	50	M	3	High School	Single	80K_to_120K	Blue	39	
8598	43	F	3	Unknown	Single	Unknown	Blue	37	
...	...	...	...	...	...	...	...	...	
10116	46	M	5	College	Single	80K_to_120K	Blue	36	
10117	57	M	2	Graduate	Married	80K_to_120K	Blue	40	
10120	54	M	1	High School	Single	60K_to_80K	Blue	34	
10121	56	F	1	Graduate	Single	Less_than_40K	Blue	50	
10122	50	M	2	Graduate	Single	40K_to_60K	Blue	40	

977 rows × 21 columns

```
Cluster_1:
 Most occuring values:
(26, 'F', 0, 'Graduate', 'Single', '40K_to_60K', 'Blue', 13, 5, 1, 2, 5655.0, 0, 5655.0, 0.842, 2312, 61, 0.649,
0.0, 0, 1)
```

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Rel
19	45	F	2	Graduate	Married	Unknown	Blue	37	
24	54	M	2	Unknown	Married	80K_to_120K	Blue	42	
28	44	F	3	Uneducated	Single	Unknown	Blue	34	
37	42	F	4	High School	Married	Less_than_40K	Gold	36	
43	49	M	3	High School	Married	60K_to_80K	Blue	37	
...	...	...	...	...	...	...	...	...	
10067	49	F	4	Uneducated	Married	40K_to_60K	Blue	36	
10089	52	F	5	Unknown	Married	Less_than_40K	Blue	36	
10118	50	M	1	Unknown	Unknown	80K_to_120K	Blue	36	
10124	44	F	1	High School	Married	Less_than_40K	Blue	36	
10125	30	M	2	Graduate	Unknown	40K_to_60K	Blue	36	

2717 rows × 21 columns

```
Cluster_2:
 Most occuring values:
(26, 'F', 0, 'College', 'Married', '40K_to_60K', 'Blue', 13, 2, 3, 3, 2010.0, 1070, 940.0, 0.906, 3625, 85, 0.635
, 0.532, 0, 2)
```

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Rel
1	49	F	5	Graduate	Single	Less_than_40K	Blue	44	

2	51	M	3	Graduate	Married	80K_to_120K	Blue	36	
3	40	F	4	High School	Unknown	Less_than_40K	Blue	34	
4	40	M	3	Uneducated	Married	60K_to_80K	Blue	21	
5	44	M	2	Graduate	Married	40K_to_60K	Blue	36	
...	...	...	...	...	...	...	...	...	
10054	33	F	1	Doctorate	Single	Less_than_40K	Blue	15	
10071	37	M	3	Unknown	Single	40K_to_60K	Blue	29	
10092	40	F	3	Graduate	Married	Unknown	Blue	25	
10123	41	M	2	Unknown	Divorced	40K_to_60K	Blue	25	
10126	43	F	2	Graduate	Married	Less_than_40K	Silver	25	

3061 rows × 21 columns

```
Cluster_3:
 Most occuring values:
(44, 'F', 0, 'College', 'Married', 'Less_than_40K', 'Blue', 40, 5, 4, 2, 7499.0, 1083, 6416.0, 0.716, 2478, 45, 0
.667, 0.144, 1, 3)
```

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Rel
9	48	M	2	Graduate	Single	80K_to_120K	Blue	36	
11	65	M	1	Unknown	Married	40K_to_60K	Blue	54	
12	56	M	1	College	Single	80K_to_120K	Blue	36	
14	57	F	2	Graduate	Married	Less_than_40K	Blue	48	
18	61	M	1	High School	Married	40K_to_60K	Blue	56	
...	...	...	...	...	...	...	...	...	
10013	52	F	0	Doctorate	Married	Less_than_40K	Blue	36	
10023	49	F	0	Unknown	Married	Less_than_40K	Blue	39	
10105	59	F	1	High School	Married	Less_than_40K	Blue	50	
10107	61	M	0	Graduate	Single	60K_to_80K	Blue	54	
10119	55	F	3	Uneducated	Single	Unknown	Blue	47	

2000 rows × 21 columns

```
Cluster_4:
 Most occuring values:
(26, 'F', 0, 'Uneducated', 'Unknown', 'Unknown', 'Silver', 13, 1, 2, 3, 34516.0, 2403, 32113.0, 0.623, 4174, 59,
0.735, 0.07, 0, 4)
```

	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	Total_Rel
0	45	M	3	High School	Married	60K_to_80K	Blue	39	
6	51	M	4	Unknown	Married	Above_120K	Gold	46	
7	32	M	0	High School	Unknown	60K_to_80K	Silver	27	
8	37	M	3	Uneducated	Single	60K_to_80K	Blue	36	
16	48	M	4	Post-Graduate	Single	80K_to_120K	Blue	36	
...	...	...	...	...	...	...	...	...	
10065	38	M	2	High School	Divorced	60K_to_80K	Silver	25	
10098	55	M	3	Graduate	Single	Above_120K	Silver	36	
10103	51	M	1	High School	Married	80K_to_120K	Blue	36	
10108	47	M	4	Graduate	Divorced	80K_to_120K	Blue	39	
10112	33	M	2	College	Married	Above_120K	Gold	20	

1372 rows × 21 columns

In [49]:

```
cluster_dict = dict(tuple(characteristics_df.groupby('Clusters')))
for i in range(n_clusters):
    print("Cluster " + str(i)+' description:')
    display(eval("cluster_dict[" + str(i)+"]").describe(include='all').T)
```

```
Cluster 0 description:
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Customer_Age	977.0	NaN	NaN	NaN	45.341863	7.637256	27.0	41.0	46.0	51.0	63.0
Gender	977	2	M	588	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Dependent_count	977.0	NaN	NaN	NaN	2.338792	1.291864	0.0	1.0	2.0	3.0	5.0
Education_Level	977	7	Graduate	312	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Marital_Status	977	4	Married	439	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Income_Category	977	6	Less_than_40K	272	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Card_Category	977	4	Blue	778	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Months_on_book	977.0	NaN	NaN	NaN	35.211873	7.655435	13.0	31.0	36.0	40.0	56.0
Total_Relationship_Count	977.0	NaN	NaN	NaN	2.183214	1.18659	1.0	1.0	2.0	3.0	6.0
Months_Inactive_12_mon	977.0	NaN	NaN	NaN	2.224156	0.984464	1.0	1.0	2.0	3.0	6.0
Contacts_Count_12_mon	977.0	NaN	NaN	NaN	2.175026	0.953354	0.0	1.0	2.0	3.0	6.0
Credit_Limit	977.0	NaN	NaN	NaN	13507.578301	9921.814347	2019.0	5282.0	10353.0	18341.0	34516.0
Total_Revolving_Bal	977.0	NaN	NaN	NaN	1402.448311	708.529891	0.0	1060.0	1481.0	1907.0	2517.0
Avg_Open_To_Buy	977.0	NaN	NaN	NaN	12105.12999	9935.945586	553.0	3936.0	9027.0	17328.0	34516.0
Total_Amt_Chng_Q4_Q1	977.0	NaN	NaN	NaN	0.775094	0.11037	0.507	0.699	0.759	0.845	1.232
Total_Trans_Amt	977.0	NaN	NaN	NaN	13144.038895	2954.380645	4957.0	12575.0	14242.0	15124.0	18484.0
Total_Trans_Ct	977.0	NaN	NaN	NaN	106.001024	13.032628	63.0	97.0	106.0	116.0	139.0
Total_Ct_Chng_Q4_Q1	977.0	NaN	NaN	NaN	0.729927	0.103683	0.412	0.662	0.731	0.797	1.148
Avg_Utilization_Ratio	977.0	NaN	NaN	NaN	0.178038	0.169453	0.0	0.057	0.127	0.248	0.802
target	977.0	NaN	NaN	NaN	0.016377	0.126984	0.0	0.0	0.0	0.0	1.0
Clusters	977.0	NaN	NaN	NaN	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
Cluster 1 description:
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Customer_Age	2717.0	NaN	NaN	NaN	44.327199	6.612517	26.0	40.0	45.0	49.0	63.0
Gender	2717	2	F	1583	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Dependent_count	2717.0	NaN	NaN	NaN	2.58042	1.216552	0.0	2.0	3.0	3.0	5.0
Education_Level	2717	7	Graduate	840	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Marital_Status	2717	4	Married	1207	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Income_Category	2717	6	Less_than_40K	1012	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Card_Category	2717	4	Blue	2646	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Months_on_book	2717.0	NaN	NaN	NaN	34.231137	6.574285	13.0	31.0	36.0	38.0	51.0
Total_Relationship_Count	2717.0	NaN	NaN	NaN	3.861612	1.490763	1.0	3.0	4.0	5.0	6.0
Months_Inactive_12_mon	2717.0	NaN	NaN	NaN	2.419212	0.979825	0.0	2.0	2.0	3.0	6.0
Contacts_Count_12_mon	2717.0	NaN	NaN	NaN	2.619801	1.121857	0.0	2.0	3.0	3.0	6.0
Credit_Limit	2717.0	NaN	NaN	NaN	5804.867685	4223.771992	1438.3	2054.0	4532.0	8621.0	18432.0
Total_Revolving_Bal	2717.0	NaN	NaN	NaN	310.391608	496.68118	0.0	0.0	0.0	672.0	2174.0
Avg_Open_To_Buy	2717.0	NaN	NaN	NaN	5494.476077	4069.772269	552.3	1950.0	4263.0	8017.0	18386.0
Total_Amt_Chng_Q4_Q1	2717.0	NaN	NaN	NaN	0.712263	0.190366	0.0	0.59	0.706	0.825	1.893
Total_Trans_Amt	2717.0	NaN	NaN	NaN	3391.271991	1616.655592	510.0	2131.0	3350.0	4447.0	10583.0
Total_Trans_Ct	2717.0	NaN	NaN	NaN	59.179978	19.362663	10.0	42.0	62.0	75.0	103.0
Total_Ct_Chng_Q4_Q1	2717.0	NaN	NaN	NaN	0.650925	0.212597	0.0	0.511	0.65	0.78	2.222
Avg_Utilization_Ratio	2717.0	NaN	NaN	NaN	0.050664	0.087924	0.0	0.0	0.0	0.091	0.616
target	2717.0	NaN	NaN	NaN	0.326831	0.469141	0.0	0.0	0.0	1.0	1.0
Clusters	2717.0	NaN	NaN	NaN	1.0	0.0	1.0	1.0	1.0	1.0	1.0

```
Cluster 2 description:
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Customer_Age	3061.0	NaN	NaN	NaN	42.119569	6.260736	26.0	38.0	43.0	47.0	58.0
Gender	3061	2	F	2090	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Dependent_count	3061.0	NaN	NaN	NaN	2.649788	1.250434	0.0	2.0	3.0	4.0	5.0
Education_Level	3061	7	Graduate	961	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Marital_Status	3061	4	Married	1412	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Income_Category	3061	6	Less_than_40K	1495	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Card_Category	3061	3	Blue	3039	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Months_on_book	3061.0	NaN	NaN	NaN	31.989546	6.509462	13.0	28.0	34.0	36.0	49.0
Total_Relationship_Count	3061.0	NaN	NaN	NaN	4.0049	1.491252	1.0	3.0	4.0	5.0	6.0
Months_Inactive_12_mon	3061.0	NaN	NaN	NaN	2.281281	1.007296	0.0	2.0	2.0	3.0	6.0
Contacts_Count_12_mon	3061.0	NaN	NaN	NaN	2.342372	1.084832	0.0	2.0	2.0	3.0	6.0
Credit_Limit	3061.0	NaN	NaN	NaN	3860.525417	2568.206148	1438.3	2289.0	2900.0	4502.0	16612.0
Total_Revolving_Bal	3061.0	NaN	NaN	NaN	1680.508004	503.868448	0.0	1305.0	1662.0	2053.0	2517.0
Avg_Open_To_Buy	3061.0	NaN	NaN	NaN	2180.017413	2447.879754	3.0	694.0	1102.0	2736.0	14424.0
Total_Amt_Chng_Q4_Q1	3061.0	NaN	NaN	NaN	0.797153	0.232706	0.0	0.657	0.76	0.886	2.594
Total_Trans_Amt	3061.0	NaN	NaN	NaN	3709.213002	1479.436364	643.0	2441.0	4074.0	4625.0	14257.0
Total_Trans_Ct	3061.0	NaN	NaN	NaN	64.94773	18.424033	12.0	51.0	69.0	79.0	109.0
Total_Ct_Chng_Q4_Q1	3061.0	NaN	NaN	NaN	0.76253	0.252604	0.0	0.628	0.738	0.86	3.714
Avg_Utilization_Ratio	3061.0	NaN	NaN	NaN	0.53748	0.212671	0.0	0.364	0.558	0.705	0.999
target	3061.0	NaN	NaN	NaN	0.080366	0.271903	0.0	0.0	0.0	0.0	1.0
Clusters	3061.0	NaN	NaN	NaN	2.0	0.0	2.0	2.0	2.0	2.0	2.0

```
Cluster 3 description:
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Customer_Age	2000.0	NaN	NaN	NaN	55.952	4.702845	44.0	53.0	56.0	59.0	73.0
Gender	2000	2	F	1155	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Dependent_count	2000.0	NaN	NaN	NaN	1.405	1.080536	0.0	1.0	1.0	2.0	5.0
Education_Level	2000	7	Graduate	616	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Marital_Status	2000	4	Married	1072	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Income_Category	2000	6	Less_than_40K	781	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Card_Category	2000	3	Blue	1969	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Months_on_book	2000.0	NaN	NaN	NaN	44.593	6.076908	30.0	40.0	45.0	49.0	56.0
Total_Relationship_Count	2000.0	NaN	NaN	NaN	4.169	1.430896	1.0	3.0	4.0	5.0	6.0
Months_Inactive_12_mon	2000.0	NaN	NaN	NaN	2.4075	1.083527	0.0	2.0	2.0	3.0	6.0
Contacts_Count_12_mon	2000.0	NaN	NaN	NaN	2.485	1.112382	0.0	2.0	3.0	3.0	6.0
Credit_Limit	2000.0	NaN	NaN	NaN	5120.81665	3763.117208	1438.3	2422.5	3517.5	6909.25	23566.0
Total_Revolving_Bal	2000.0	NaN	NaN	NaN	1391.411	697.379879	0.0	975.0	1456.5	1892.0	2517.0
Avg_Open_To_Buy	2000.0	NaN	NaN	NaN	3729.40565	3725.139024	10.0	962.0	2160.0	5471.25	21896.0
Total_Amt_Chng_Q4_Q1	2000.0	NaN	NaN	NaN	0.757195	0.247699	0.0	0.609	0.726	0.8625	3.397
Total_Trans_Amt	2000.0	NaN	NaN	NaN	3108.5555	1588.814085	530.0	1618.75	3075.0	4363.75	10170.0
Total_Trans_Ct	2000.0	NaN	NaN	NaN	55.9635	20.742041	10.0	37.0	58.0	74.0	104.0
Total_Ct_Chng_Q4_Q1	2000.0	NaN	NaN	NaN	0.711277	0.263373	0.0	0.561	0.688	0.826	3.571
Avg_Utilization_Ratio	2000.0	NaN	NaN	NaN	0.379869	0.256527	0.0	0.167	0.352	0.59025	0.995
target	2000.0	NaN	NaN	NaN	0.133	0.33966	0.0	0.0	0.0	0.0	1.0
Clusters	2000.0	NaN	NaN	NaN	3.0	0.0	3.0	3.0	3.0	3.0	3.0

```
Cluster 4 description:
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Customer_Age	1372.0	NaN	NaN	NaN	46.337464	6.597126	26.0	42.0	46.0	51.0	65.0
Gender	1372	2	M	1231	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Dependent_count	1372.0	NaN	NaN	NaN	2.582362	1.219132	0.0	2.0	3.0	3.0	5.0
Education_Level	1372	7	Graduate	399	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Marital_Status	1372	4	Single	565	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Income_Category	1372	6	80K_to_120K	569	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Card_Category	1372	4	Blue	1004	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Months_on_book	1372.0	NaN	NaN	NaN	35.956997	6.715239	13.0	32.0	36.0	39.0	56.0
Total_Relationship_Count	1372.0	NaN	NaN	NaN	3.927114	1.521236	1.0	3.0	4.0	5.0	6.0
Months_Inactive_12_mon	1372.0	NaN	NaN	NaN	2.306851	0.968435	0.0	2.0	2.0	3.0	6.0
Contacts_Count_12_mon	1372.0	NaN	NaN	NaN	2.537901	1.149328	0.0	2.0	3.0	3.0	6.0
Credit_Limit	1372.0	NaN	NaN	NaN	26522.131195	6887.51212	12691.0	20229.75	26158.0	34516.0	34516.0
Total_Revolving_Bal	1372.0	NaN	NaN	NaN	1192.008017	792.140229	0.0	653.5	1291.0	1752.5	2517.0
Avg_Open_To_Buy	1372.0	NaN	NaN	NaN	25330.123178	6965.649001	10848.0	19084.0	25086.0	32606.0	34516.0
Total_Amt_Chng_Q4_Q1	1372.0	NaN	NaN	NaN	0.764548	0.237326	0.0	0.62075	0.738	0.88125	3.355

Total_Trans_Amt	1372.0	NaN	NaN	NaN	3624.896501	2162.814088	597.0	1809.0	3420.0	4367.25	14954.0
Total_Trans_Ct	1372.0	NaN	NaN	NaN	59.575073	20.179957	10.0	43.0	62.0	75.0	114.0
Total_Ct_Chng_Q4_Q1	1372.0	NaN	NaN	NaN	0.710143	0.254154	0.0	0.564	0.684	0.806	2.429
Avg_Utilization_Ratio	1372.0	NaN	NaN	NaN	0.049041	0.0363	0.0	0.022	0.049	0.07225	0.185
target	1372.0	NaN	NaN	NaN	0.15379	0.360879	0.0	0.0	0.0	0.0	1.0
Clusters	1372.0	NaN	NaN	NaN	4.0	0.0	4.0	4.0	4.0	4.0	4.0

intra cluster EDA

In [50]:
```
@interact(Cluster=cluster_dict.keys())
def show_clusters(Cluster):
    fn.cluster_insights(cluster_dict[Cluster])
```

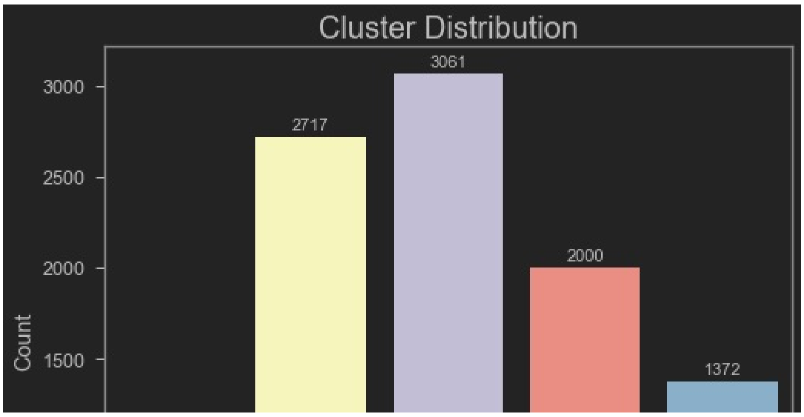
comment

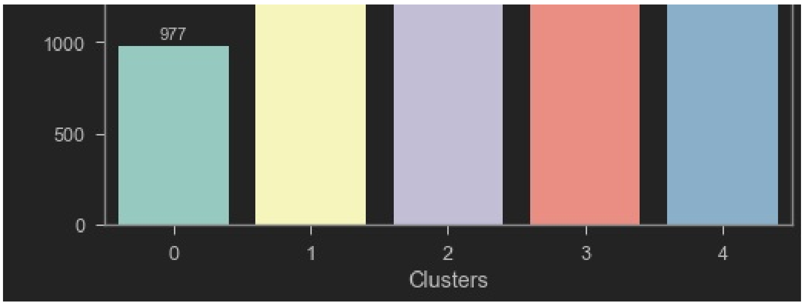
inter cluster EDA

Exploring features among clusters based on the insights from the feature importance from the previous part of the analysis. Most important features are explored.

Cluster Distribution

In [51]:
```
plot_data = characteristics_df.groupby(
    'Clusters').count()['target'].sort_index(ascending=False).reset_index()
plots = sns.barplot(y='target',
                    x='Clusters',
                    data=plot_data,
                    orient='v',
                    palette='Set3')
for bar in plots.patches:
    plots.annotate(format(bar.get_height(), '.0f'),
                   (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                   ha='center',
                   va='center',
                   size=11,
                   xytext=(0, 8),
                   textcoords='offset points')
plt.ylabel("Count")
plt.title("Cluster Distribution", size=20)
plt.show()
```

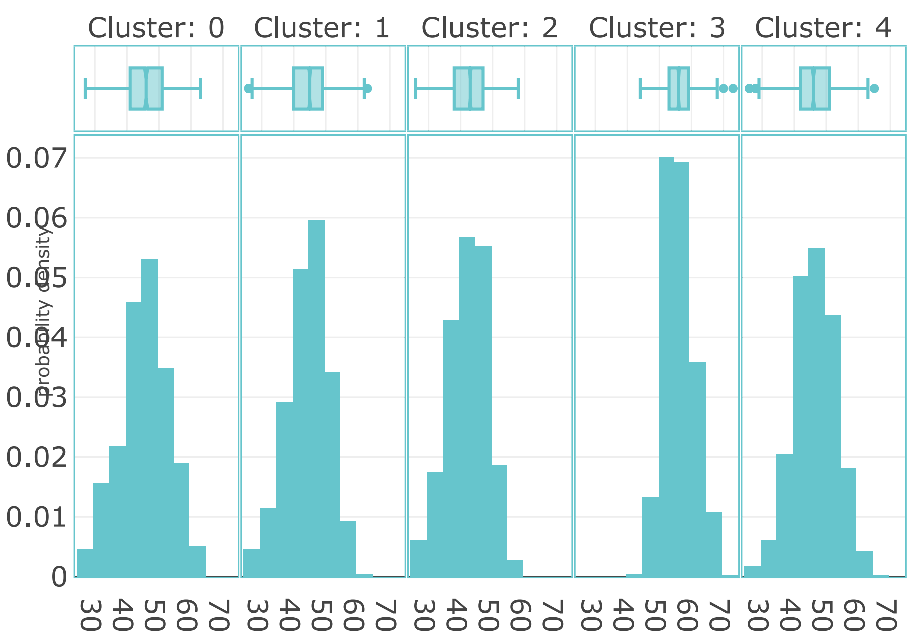




Customer Age

```
In [109…
fig = fn.feature_analysis_intracluster(data_frame=characteristics_df,
                                       x='Customer_Age',
                                       facet_col='Clusters',
                                       n_clusters=n_clusters,
                                       nbins=10)
fig.update_xaxes(tickmode='linear', tick0=20, dtick=10)
```

Customer Age

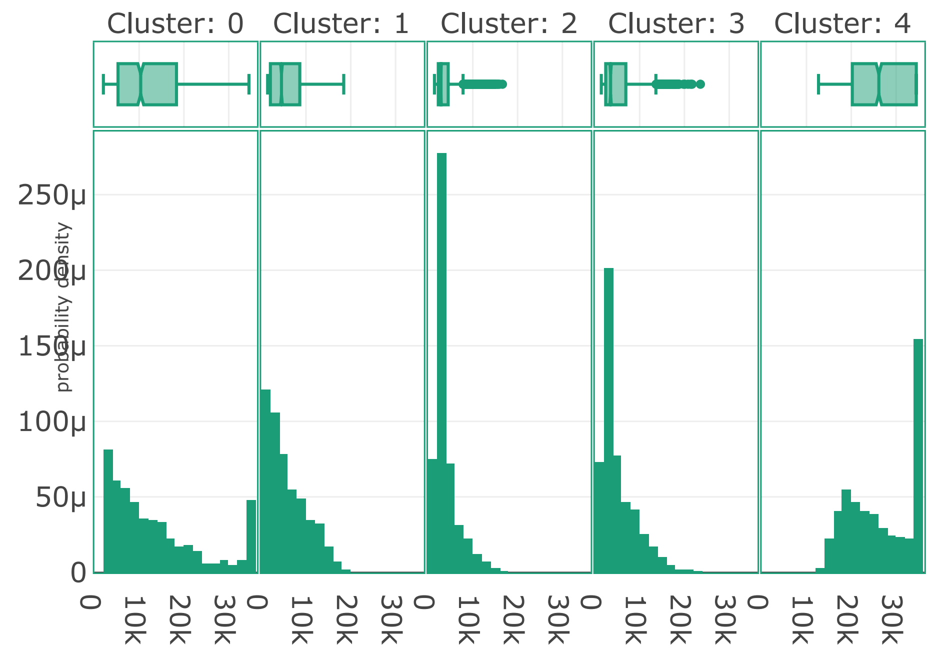


Cluster 4 and 1 has similar distribution. Cluster 0 is younger. Cluster 3 is distinct as it is mostly comprised of older clients. Others have similar distribution.

Credit Limit

```
In [111…
fig = fn.feature_analysis_intracluster(
    data_frame=characteristics_df,
    x='Credit_Limit',
    facet_col='Clusters',
    n_clusters=n_clusters,
    color_discrete_sequence=px.colors.qualitative.Dark2,
    nbins=25)
fig.show()
```

Credit Limit



- Cluster 0 has a well balanced distribution, it does not have lower credit limit clients.
- Cluster 1 has mostly lower credit limit clients.
- Cluster 2 and 3 has mostly same characteristics.
- Cluster 4 has the clients with mostly high credit limit.

Avg Utilization Ratio

In [113…

```
fig = fn.feature_analysis_intracluster(
    data_frame=characteristics_df,
    x='Avg_Utilization_Ratio',
    facet_col = characteristics_df.Clusters,
    n_clusters=n_clusters,
    color_discrete_sequence=['#ff0000'],
    nbins=10)
fig.update_xaxes(tickmode='linear', tick0=0, dtick=.20)
fig.show()
```

- Cluster 0 shows good utilization ratio, with some 0.
- Cluster 1 has mostly less utilization ratio.
- Cluster 2 and 3 has similar utilization. Cluster 2 does not have many 0's.
- Cluster 4 has low utilization of credit.

Months on book

In [115…

```
fig = fn.feature_analysis_intracluster(
    data_frame=characteristics_df,x='Months_on_book',
    facet_col='Clusters',
    n_clusters=n_clusters,
    color_discrete_sequence=['rgb(135, 197, 95)'])
fig.update_xaxes(tickmode='linear', tick0=10, dtick=10)
fig.show()
```

All of them show similar spread except Cluster 3, they are the most loyal clients.

Total_Trans_Amt

In [117…

```
fig = fn.feature_analysis_intracluster(data_frame=characteristics_df,
    x='Total_Trans_Amt',
    facet_col='Clusters',
    n_clusters=n_clusters,
    color_discrete_sequence=['rgb(201, 219, 116)'])
fig.show()
```

Cluster 0 has highest transaction amount. Rest of the has similar pattern.

Avg_Open_To_Buy

In [119…

```
fig = fn.feature_analysis_intracluster(data_frame=characteristics_df,
                                       x='Avg_Open_To_Buy',
                                       facet_col='Clusters',
                                       n_clusters=n_clusters,
                                       color_discrete_sequence=['chocolate'])
fig.show()
```

- Cluster 0 has a well spread.
- Cluster 1, 2, 3 are mostly similar.
- Cluster 4 has most open to buy available.

Total_Trans_Ct

In [121…

```
fig = fn.feature_analysis_intracluster(data_frame=characteristics_df,
                                       x='Total_Trans_Ct',
                                       facet_col='Clusters',
                                       n_clusters=n_clusters,
                                       color_discrete_sequence=['skyblue'])
```

```python
fig.update_xaxes(tickmode='linear', tick0=0, dtick=10)
fig.show()
```

- Cluster 0 is the most frequent user.
- rest of the clusters have similar spread.

Total_Revolving_Bal

In [124…

```python
fig = fn.feature_analysis_intracluster(data_frame=characteristics_df,
                                       x='Total_Revolving_Bal',
                                       facet_col='Clusters',histnorm='density',
                                       n_clusters=n_clusters,
                                       nbins=10,
                                       color_discrete_sequence=['tomato'])
fig.update_xaxes(tickmode='linear', tick0=0, dtick=500)
fig.show()
```

- Cluster 0 has even distribution.
- Cluster 1 has mostly low revolving balance.
- Cluster 2 does not include low revolving balance clients.
- Cluster 3 and 4 has similar distribution.

Total_Relationship_Count

In [126…

```
fig = fn.feature_analysis_intracluster(data_frame=characteristics_df,
                                       x='Total_Relationship_Count',
                                       facet_col='Clusters',
                                       n_clusters=n_clusters,
                                       color_discrete_sequence=['turquoise'])
fig.update_xaxes(tickmode='linear', tick0=0, dtick=1)
fig.show()
```

Cluster 0 mostly comprised of lower relationship count clients. Rest of the Clusters has similar distributions.

Dependent_count

In [128…

```
fig = fn.feature_analysis_intracluster(data_frame=characteristics_df,
                                       x='Dependent_count',
                                       facet_col='Clusters',
                                       n_clusters=n_clusters,
                                       color_discrete_sequence=['orangered'])
fig.update_xaxes(tickmode='linear', tick0=0, dtick=1)
fig.show()
```

All of them are mostly similar.

with churn info

All the features are explored with respect of churning.

In [136…

```
@interact(Cluster=fixed(characteristics_df),
          feature=characteristics_df.columns)
def show_clusters(Cluster, feature='Customer_Age'):
    fig = px.histogram(Cluster,
                       x=feature,
                       marginal="box",
                       template='presentation',
                       color='Clusters',
                       facet_col='target',
                       color_discrete_sequence=px.colors.qualitative.Dark2,
                       barmode='group',
                       title=f'"{feature}" seperated by Clusters',
                       hover_data=Cluster)
    fig.show()
    pass
```

In [156…

```
@interact(Cluster=fixed(characteristics_df),
          feature=characteristics_df.columns)
def show_clusters(Cluster, feature='Customer_Age'):
    fig = px.histogram(
        data_frame=Cluster,
        x=feature,
        marginal="box",
        template='presentation',
        color='target',
        facet_col='Clusters',
        color_discrete_sequence=px.colors.qualitative.Dark2,
        barmode='group',
        category_orders={'Clusters': list(np.arange(0, n_clusters))},
        title=f'"{feature.replace("_"," ")}" seperated by Clusters',
        hover_data=Cluster)
```

```
    fig.update_xaxes(showline=True,
                     linewidth=1,
                     linecolor='black', title={'text': ''})
#     fig.update_yaxes(title={'text': ''})
    fig.update_layout(annotations=list(fig.layout.annotations) + [
        go.layout.Annotation(x=0.5,
                             y=-0.22,
                             font=dict(size=14),
                             showarrow=False,
                             text=f"{feature}s",
                             textangle=0,
                             xref="paper",
                             yref="paper")
    ])
    fig.show()
    pass
```

comment

name clusters

Prediction

Prediction from the clustering model is used as a feature for modeling churn prediction model. Models without this feature was also experimented. Those models had a slightly worse performance. For the final modeling approach, dataset containing predictions from the kmeans model is used.

In [127…

```
# appending churn labels as 'target'
cluster_df['target'] = df.Attrition_Flag.map(churn_map).copy()
cluster_df
```

Out[127…

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon	Contacts_Count_12_mon	Credit_l
0	-0.165406	0.503368	0.384621	0.763943	-1.327136	0.492404	0.44
1	0.333570	2.043199	1.010715	1.407306	-1.327136	-0.411616	-0.04
2	0.583058	0.503368	0.008965	0.120579	-1.327136	-2.219655	-0.57
3	-0.789126	1.273283	-0.241473	-0.522785	1.641478	-1.315636	-0.58
4	-0.789126	0.503368	-1.869317	0.763943	-1.327136	-2.219655	-0.43
...	...	...	...	...	...	...	
10122	0.458314	-0.266547	0.509840	-0.522785	-0.337598	0.492404	-0.50
10123	-0.664382	-0.266547	-1.368442	0.120579	-0.337598	0.492404	-0.47
10124	-0.290150	-1.036462	0.008965	0.763943	0.651940	1.396424	-0.35
10125	-2.036565	-0.266547	0.008965	0.120579	0.651940	0.492404	-0.36
10126	-0.414894	-0.266547	-1.368442	1.407306	-0.337598	1.396424	0.19

10127 rows × 39 columns

In [123…

```
# preparing X (independent variable), and y (dependent) for the model
X_additional_col = cluster_df.drop(columns='target').copy()
y_additional_col = cluster_df.target.copy()
```

In [124…

```
# train test split size of 80%
X_train_pr, X_test_pr, y_train, y_test = train_test_split(X_additional_col,
                                                         y_additional_col,
                                                         train_size=.8)
```

In [125…

```python
# creating an instance of SMOTENC using feature list defined at the SCRUB section
oversampling1 = SMOTENC(categorical_features=smotenc_features, n_jobs=-1)
```

In [126…

```python
# oversampling X based on y
X_train_pr_os, y_train_encoded_os = oversampling1.fit_sample(X_train_pr, y_train)
```

Baseline model

In [59]:

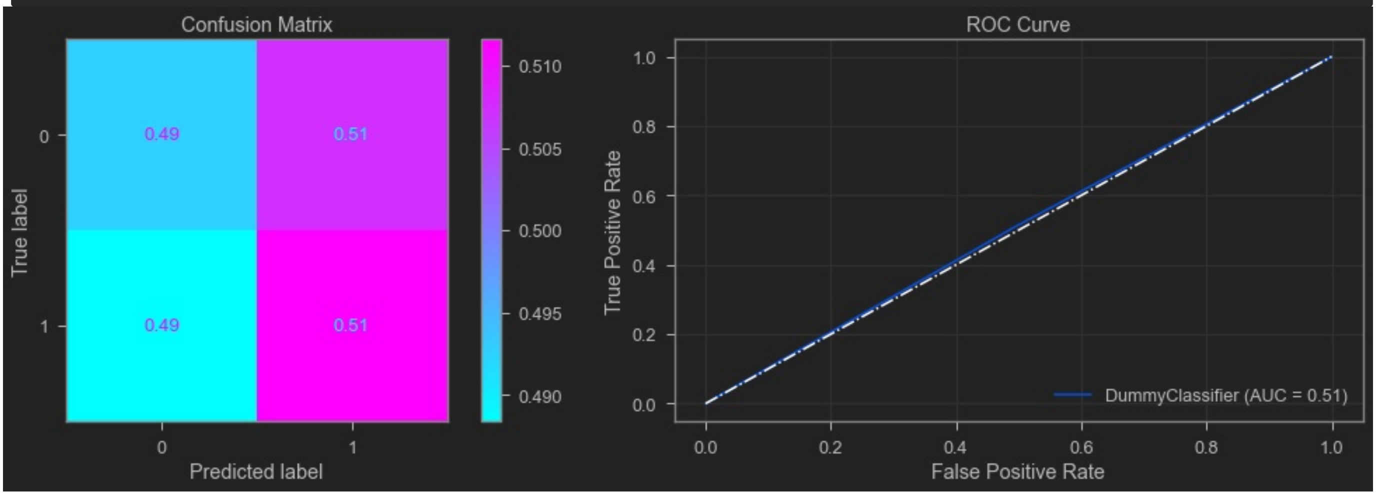
```python
base_model = DummyClassifier(strategy='stratified')
```

In [60]:

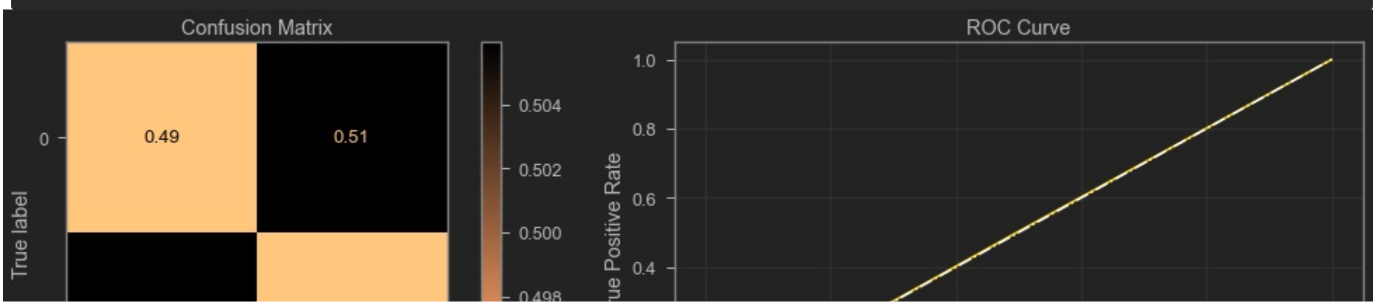
```python
# using oversampled data
fn.model_report(base_model, X_train_pr_os, y_train_encoded_os, X_test_pr,
                y_test)
```

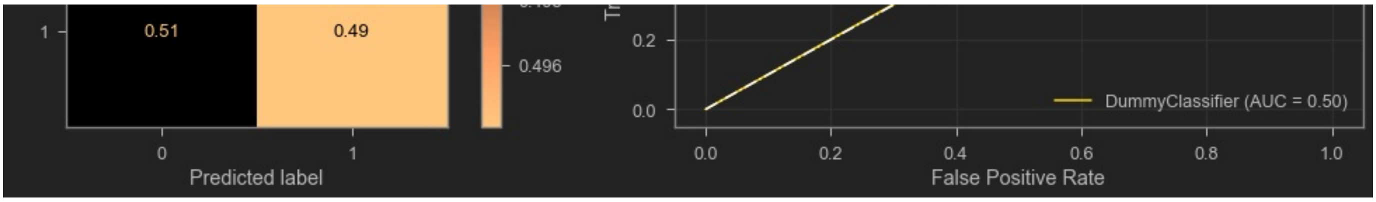
Report of DummyClassifier type model using train-test split dataset.

```
*****************************************************************************************
Train accuracy score: 0.5109
Test accuracy score: 0.4985
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
*****************************************************************************************

Train Report:
************************************************************
              precision    recall  f1-score   support

           0       0.50      0.51      0.51      6814
           1       0.50      0.50      0.50      6814

    accuracy                           0.50     13628
   macro avg       0.50      0.50      0.50     13628
weighted avg       0.50      0.50      0.50     13628

************************************************************
```




```
Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.85      0.52      0.65      1686
           1       0.18      0.53      0.27       340

    accuracy                           0.52      2026
   macro avg       0.51      0.53      0.46      2026
weighted avg       0.74      0.52      0.58      2026

************************************************************
```





The baseline model is performing as par as random chance of flipping a coin for prediction.

Logistic Regression

In [85]:
```
fn.heatmap_of_features(X_additional_col);
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-85-90626af915ca> in <module>
----> 1 fn.heatmap_of_features(X_additional_col);

NameError: name 'X_additional_col' is not defined
```

`'Avg_Open_To_Buy'` with `Credit_limit`, `'Card_Category_Silver'` with `'Card_Category_Blue`, `'Gender_M'` with `'Gender_F`, `'Months_on_book'` with `'Customer_Age'`, `'Total_Trans_Ct'` with `'Total_Trans_Amt` features are showing high multicollinearity. Those are expected by the nature of those features.

- `'Avg_Open_To_Buy'` with `Credit_limit`: Credit limit has a direct impact on a clients ability to spend. It is a positive relationship.
- `'Card_Category_Silver'` with `'Card_Category_Blue`: This is interesting. Blue and Silver cards are the two most common type of credit card. There is a strong negative relationship.
- `'Gender_M'` with `'Gender_F`: Binary category.
- `'Months_on_book'` with `'Customer_Age'`: Customers age has a impact on how long they can be a customer of the bank. Older they are, more time they have to be a customer. - `'Total_Trans_Ct'` with `'Total_Trans_Amt`: Very closely related feature. 80% correlation is not that horrible.

In [221…
```
fn.drop_features_based_on_correlation(X_additional_col)
```

Out[221…
```
{'Avg_Open_To_Buy', 'Months_on_book'}
```

Multicollinearity undermines the statistical significance of an independent variable. Here it is important to point out that multicollinearity does not affect the model's predictive accuracy. Choosing not to deal with this issue right now.

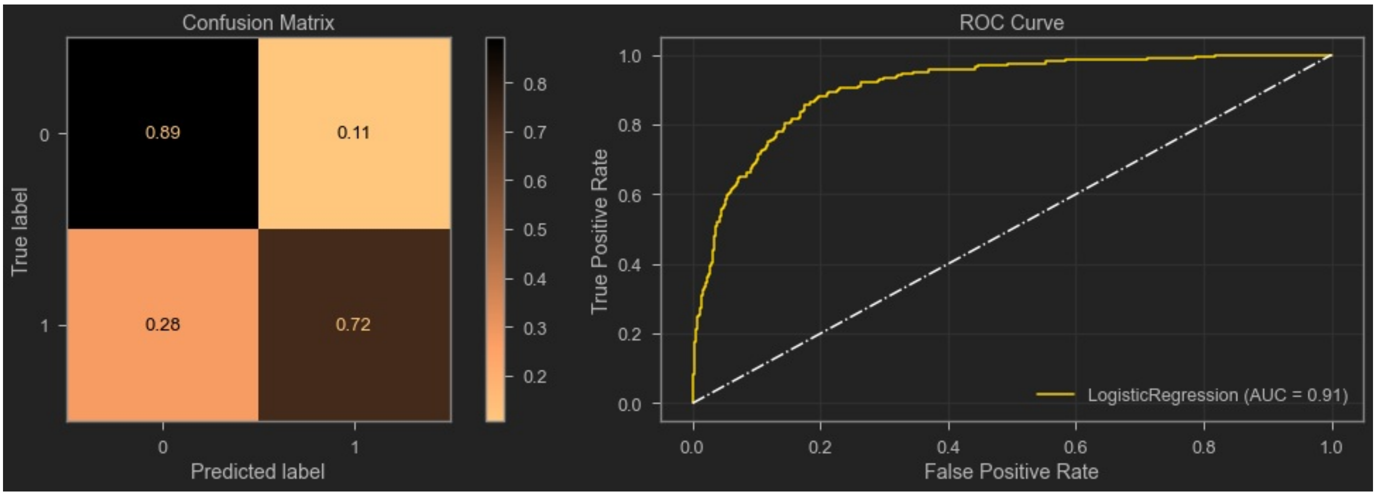
In [86]:
```
# dropped first from OHE using cluster prediction
X_log_reg = X.copy()
X_log_reg['cluster'] = clusters
X_log_reg['cluster'] = X_log_reg['cluster'].astype('str')

X_train_log_reg, y_train_log_reg, X_test_log_reg, y_test_log_reg = fn.dataset_processor(
    X_log_reg, y, verbose=0, OHE_drop_option='first')
```

In [88]:
```
# with all data
logreg = LogisticRegression(max_iter=1000, class_weight='balanced')
# score of logistic regression classifier
fn.model_report(logreg,
                X_train_log_reg,
                y_train_log_reg,
                X_test_log_reg,
                y_test_log_reg,
```

```
                show_train_report=False)
```

Report of LogisticRegression type model using train-test split dataset.

```
*****************************************************************************************
Train accuracy score: 0.9052
Test accuracy score: 0.8653
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
*****************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.94      0.89      0.92      1702
           1       0.56      0.72      0.63       324

    accuracy                           0.87      2026
   macro avg       0.75      0.81      0.77      2026
weighted avg       0.88      0.87      0.87      2026

************************************************************
```



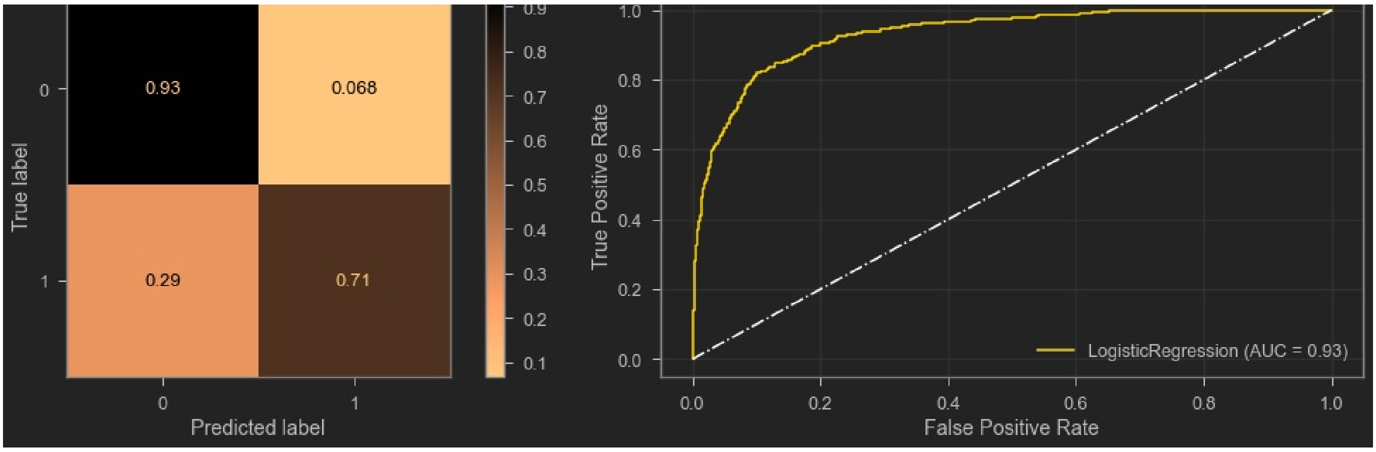
Model is not good enough to predict target class 1, churned customer. Although accuracy is good.

In [155…

```
# dropping all the correlated features
logreg_1 = LogisticRegression(max_iter=1000, class_weight='balanced')
# score of logistic regression classifier
fn.model_report(logreg_1,
                X_train_pr_os.drop(columns=['Gender_M', 'Months_on_book']),
                y_train_encoded_os,
                X_test_pr.drop(columns=['Gender_M', 'Months_on_book']),
                y_test,
                show_train_report=False)
```

Report of LogisticRegression type model using train-test split dataset.

```
*****************************************************************************************
Train accuracy score: 0.9229
Test accuracy score: 0.8963
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
*****************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.94      0.93      0.94      1699
           1       0.67      0.71      0.69       327

    accuracy                           0.90      2026
   macro avg       0.81      0.82      0.81      2026
weighted avg       0.90      0.90      0.90      2026

************************************************************
```



The accuracy is good enough. But the the residual must be crazy as indicated by the f-1 and precision values. Supports my previous point about model performance. Outlier removal is next. Not pursuing that because data loss will be very high as there are lots of recurring values for the numeric values (lots of zeros) for both IQR and Z-score based approach for outlier removal.

Critical features for churning:

Odds ratios are used to measure the relative odds of the occurrence of the outcome, given a factor of interest [Bland JM, Altman DG. (2000), The odds ratio]. The odds ratio is used to determine whether a particular attribute is a risk factor or protective factor for a particular class and the magnitude of percentage effect is used to compare the various risk factors for that class. The positive percentage effect means that the factor is positively correlated with churn and vice versa.

The odds ratio and percentage effect of each feature are estimated as $\mathbf{OddsRatio} = e^{\Theta }$ and $\mathbf{Effect (\%)} = 100 * (OddsRatio - 1)$, where Θ is the value of weight of each feature in Logistic Regression model. If the effect is positive, the greater the factor, the likely that the client will churn, those factors are considered as risk factors. While if the effect is negative, the greater the factor, the greater the possibility that the customer will not churn, and can be considered as protective factors. This is a Bayesian approach for identifying feature importance.

In [163…

```
churn_feature = pd.DataFrame(
    logreg.coef_,columns=X_train_pr_os.columns).T
churn_feature.columns = ['weights']
churn_feature['odds_ratio'] = np.exp(churn_feature['weights'])
churn_feature['effect'] = 100 * (churn_feature['odds_ratio'] - 1)
churn_feature
```

Out[163…

	weights	odds_ratio	effect
Customer_Age	-0.117767	0.888903	-11.109700
Dependent_count	0.238381	1.269192	26.919208
Months_on_book	0.000782	1.000783	0.078276
Total_Relationship_Count	-0.623083	0.536289	-46.371148
Months_Inactive_12_mon	0.561289	1.752930	75.293033
Contacts_Count_12_mon	0.557224	1.745820	74.581995
Credit_Limit	-0.032555	0.967970	-3.203032
Total_Revolving_Bal	-0.686817	0.503175	-49.682492
Avg_Open_To_Buy	0.029026	1.029451	2.945139
Total_Amt_Chng_Q4_Q1	-0.093160	0.911048	-8.895218
Total_Trans_Amt	1.734070	5.663656	466.365614
Total_Trans_Ct	-2.874869	0.056424	-94.357647
Total_Ct_Chng_Q4_Q1	-0.685138	0.504021	-49.597937
Avg_Utilization_Ratio	-0.097252	0.907327	-9.267299
Gender_F	0.569285	1.767003	76.700272
Gender_M	-0.569882	0.565592	-43.440755
Education_Level_College	-4.639986	0.009658	-99.034216
Education_Level_Doctorate	-4.591258	0.010140	-98.985991
Education_Level_Graduate	-4.100097	0.016571	-98.342893
Education_Level_High School	-4.690399	0.009183	-99.081698

Education_Level_Post-Graduate	-4.408546	0.012173	-98.782714
Education_Level_Uneducated	-4.656669	0.009498	-99.050195
Education_Level_Unknown	-4.506989	0.011032	-98.896838
Marital_Status_Divorced	-3.291290	0.037206	-96.279417
Marital_Status_Married	-3.106604	0.044753	-95.524732
Marital_Status_Single	-2.784195	0.061779	-93.822121
Marital_Status_Unknown	-3.196119	0.040921	-95.907929
Income_Category_40K_to_60K	-3.683680	0.025130	-97.486967
Income_Category_60K_to_80K	-3.400740	0.033349	-96.665141
Income_Category_80K_to_120K	-3.134136	0.043537	-95.646266
Income_Category_Above_120K	-2.982712	0.050655	-94.934471
Income_Category_Less_than_40K	-3.226716	0.039688	-96.031240
Income_Category_Unknown	-4.192490	0.015109	-98.489138
Card_Category_Blue	-0.806310	0.446502	-55.349751
Card_Category_Gold	-0.755450	0.469799	-53.020094
Card_Category_Platinum	-0.239052	0.787374	-21.262607
Card_Category_Silver	-1.022591	0.359662	-64.033803
Clusters	-0.004602	0.995409	-0.459118

In [164…

```
churn_feature = pd.DataFrame(
    logreg_1.coef_,
    columns=X_train_pr_os.drop(
        columns=['Gender_M', 'Months_on_book']).columns).T
churn_feature.columns = ['weights']
churn_feature['odds_ratio'] = np.exp(churn_feature['weights'])
churn_feature['effect'] = 100 * (churn_feature['odds_ratio'] - 1)
churn_feature
```

Out[164…

	weights	odds_ratio	effect
Customer_Age	-0.117033	0.889556	-11.044441
Dependent_count	0.238621	1.269497	26.949731
Total_Relationship_Count	-0.623010	0.536327	-46.367261
Months_Inactive_12_mon	0.561235	1.752837	75.283679
Contacts_Count_12_mon	0.557002	1.745431	74.543104
Credit_Limit	-0.032587	0.967938	-3.206218
Total_Revolving_Bal	-0.686723	0.503222	-49.677767
Avg_Open_To_Buy	0.028985	1.029409	2.940885
Total_Amt_Chng_Q4_Q1	-0.093182	0.911028	-8.897190
Total_Trans_Amt	1.733501	5.660436	466.043581
Total_Trans_Ct	-2.874071	0.056469	-94.353140
Total_Ct_Chng_Q4_Q1	-0.685043	0.504068	-49.593160
Avg_Utilization_Ratio	-0.097189	0.907384	-9.261572
Gender_F	1.128712	3.091673	209.167261
Education_Level_College	-4.638750	0.009670	-99.033022
Education_Level_Doctorate	-4.589374	0.010159	-98.984078
Education_Level_Graduate	-4.099040	0.016589	-98.341141
Education_Level_High School	-4.689351	0.009193	-99.080735
Education_Level_Post-Graduate	-4.407496	0.012186	-98.781434
Education_Level_Uneducated	-4.655568	0.009509	-99.049149
Education_Level_Unknown	-4.505731	0.011046	-98.895449
Marital_Status_Divorced	-3.293851	0.037111	-96.288933
Marital_Status_Married	-3.108819	0.044654	-95.534634
Marital_Status_Single	-2.786307	0.061648	-93.835153
Marital_Status_Unknown	-3.198776	0.040812	-95.918789

Income_Category_40K_to_60K	-3.681545	0.025184	-97.481597
Income_Category_60K_to_80K	-3.404742	0.033215	-96.678460
Income_Category_80K_to_120K	-3.138012	0.043369	-95.663105
Income_Category_Above_120K	-2.986736	0.050452	-94.954818
Income_Category_Less_than_40K	-3.221047	0.039913	-96.008675
Income_Category_Unknown	-4.186600	0.015198	-98.480212
Card_Category_Blue	-0.803436	0.447788	-55.221229
Card_Category_Gold	-0.756667	0.469228	-53.077243
Card_Category_Platinum	-0.240929	0.785897	-21.410283
Card_Category_Silver	-1.019312	0.360843	-63.915684
Clusters	-0.004510	0.995500	-0.450029

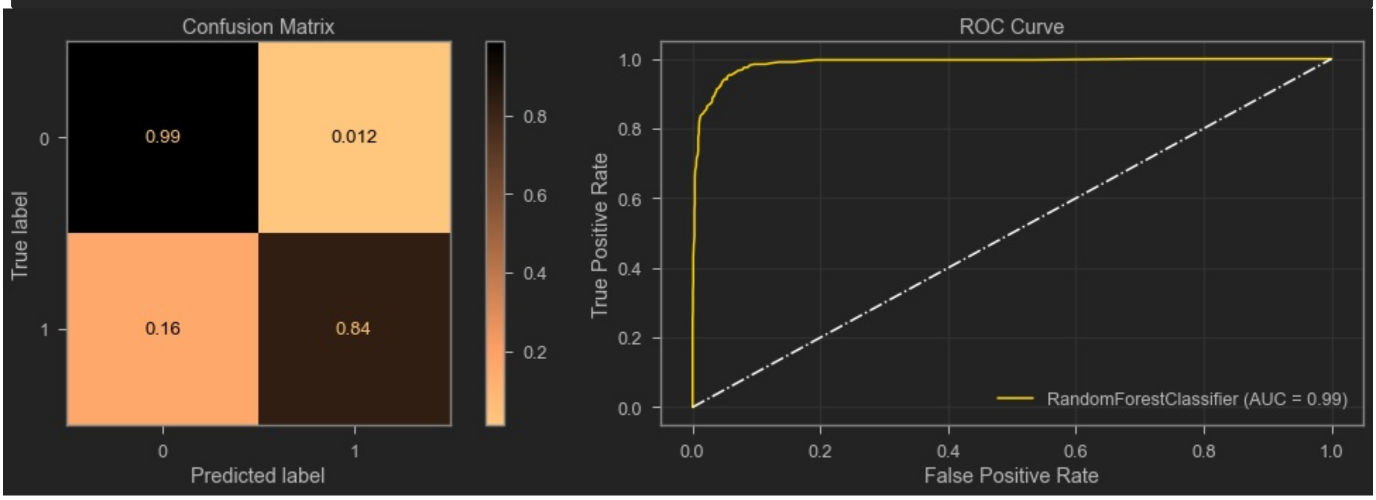
Random Forest

OG data

In [65]:
```
clf_rf = RandomForestClassifier(n_jobs=-1)
fn.model_report(clf_rf,
             X_train_pr,
             y_train,
             X_test_pr,
             y_test,
             show_train_report=False)
```

Report of RandomForestClassifier type model using train-test split dataset.

```
******************************************************************************************
Train accuracy score: 1.0
Test accuracy score: 0.9635
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
******************************************************************************************

Test Report: 
************************************************************
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1702
           1       0.93      0.84      0.88       324

    accuracy                           0.96      2026
   macro avg       0.95      0.91      0.93      2026
weighted avg       0.96      0.96      0.96      2026

************************************************************
```



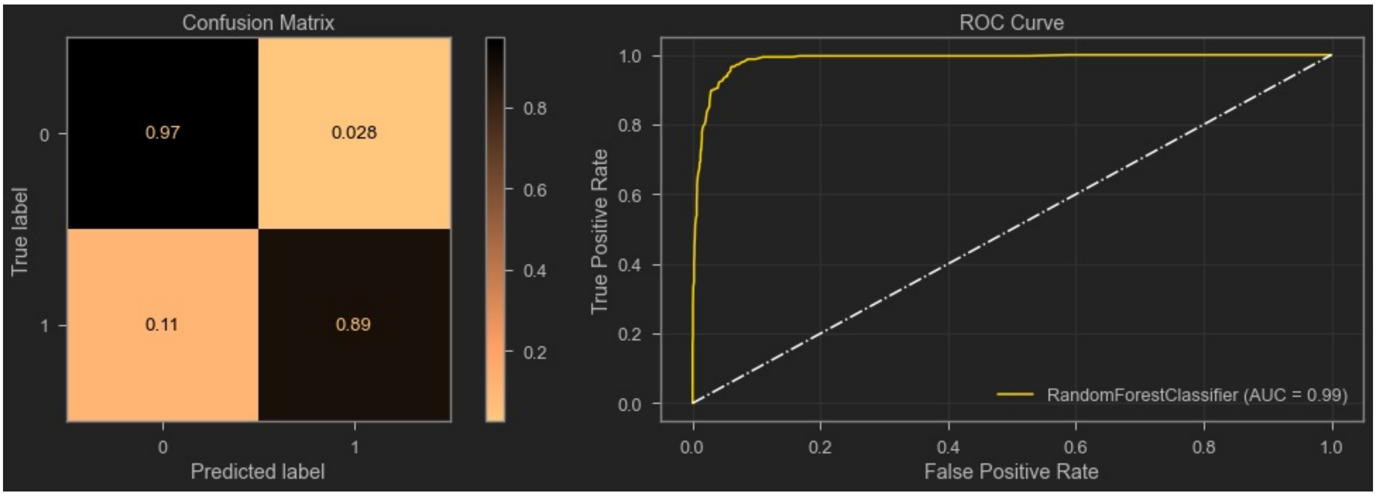
OS data

In [66]:
```
clf_rf = RandomForestClassifier(n_jobs=-1)
```

```
fn.model_report(clf_rf,
                X_train_pr_os,
                y_train_encoded_os,
                X_test_pr,
                y_test,
                show_train_report=False)
```

Report of RandomForestClassifier type model using train-test split dataset.

```
******************************************************************************************
Train accuracy score: 1.0
Test accuracy score: 0.959
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
******************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1702
           1       0.86      0.89      0.87       324

    accuracy                           0.96      2026
   macro avg       0.92      0.93      0.92      2026
weighted avg       0.96      0.96      0.96      2026

************************************************************
```



Grid Search

In [67]:

```
rf_clf_gs = RandomForestClassifier(n_jobs=-1, verbose=0)
params = {
    'criterion': ["gini", "entropy"],
    'max_depth': [5, 6, 7, 8, 9, 10],
    'min_samples_leaf': [2, 3, 4],
#      'class_weight': ["balanced", "balanced_subsample"]
}
gridsearch_rf_clf = GridSearchCV(estimator=rf_clf_gs,
                                 param_grid=params,
                                 n_jobs=-1,
                                 scoring='f1_macro')
gridsearch_rf_clf
```

Out[67]:

```
GridSearchCV(estimator=RandomForestClassifier(n_jobs=-1), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [5, 6, 7, 8, 9, 10],
                         'min_samples_leaf': [2, 3, 4]},
             scoring='f1_macro')
```

In [68]:

```
with warnings.catch_warnings():
```

```python
    warnings.simplefilter("ignore")
    gridsearch_rf_clf.fit(X_train_pr_os, y_train_encoded_os)
print(f"Best Parameters by gridsearch:\t{gridsearch_rf_clf.best_params_}")
print(f"Best Estimator by gridsearch:\t{gridsearch_rf_clf.best_estimator_}")

rf_clf_gs_best = gridsearch_rf_clf.best_estimator_
```

```
Best Parameters by gridsearch: {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2}
Best Estimator by gridsearch: RandomForestClassifier(max_depth=10, min_samples_leaf=2, n_jobs=-1)
```

In [69]:

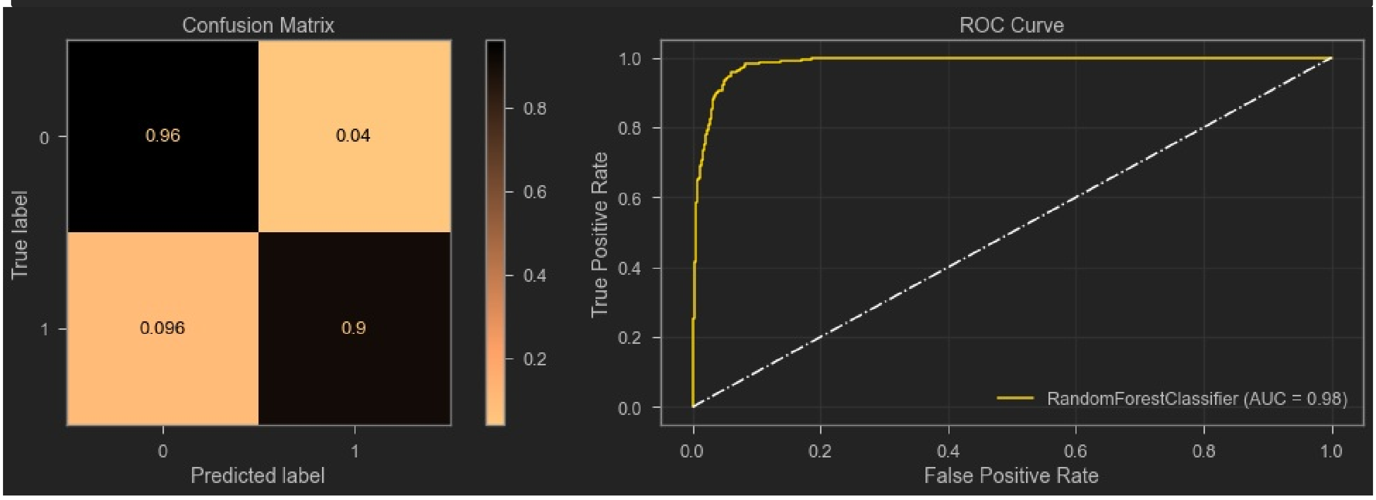
```python
# import sklearn
# sorted(sklearn.metrics.SCORERS.keys())
```

In [70]:

```python
fn.model_report(rf_clf_gs_best, X_train_pr_os, y_train_encoded_os, X_test_pr,
                y_test,
                show_train_report=False)
```

Report of RandomForestClassifier type model using train-test split dataset.

```
*****************************************************************************************
Train accuracy score: 0.9848
Test accuracy score: 0.9511
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
*****************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      1702
           1       0.81      0.90      0.86       324

    accuracy                           0.95      2026
   macro avg       0.90      0.93      0.91      2026
weighted avg       0.95      0.95      0.95      2026

************************************************************
```



XGBoost

XGBClassifier

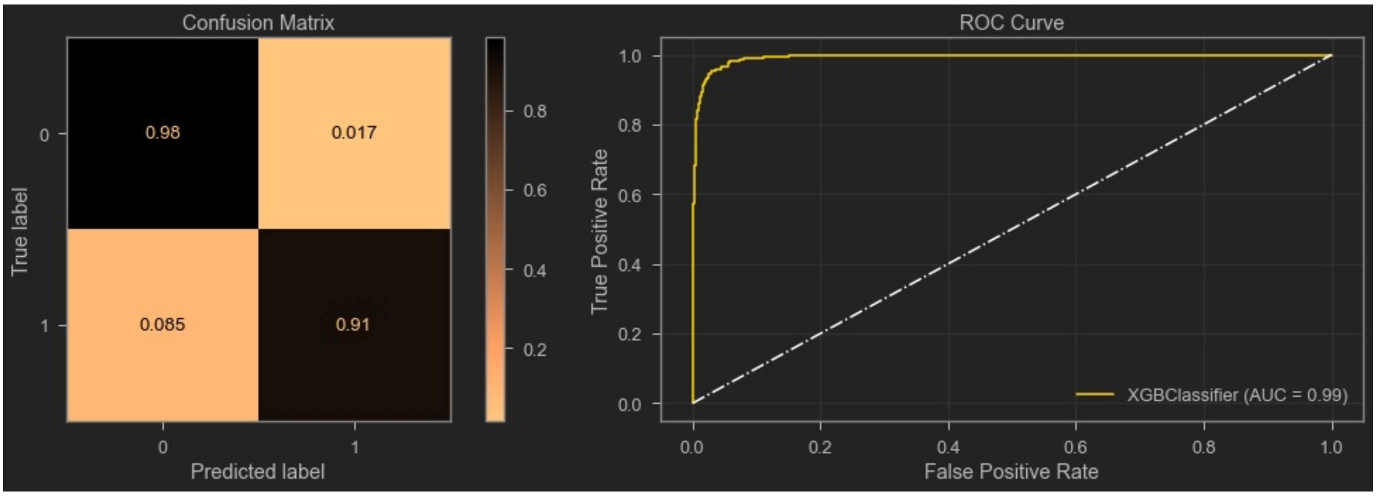
In [80]:

```python
clf_xg = XGBClassifier(n_jobs=-1)
fn.model_report(clf_xg, X_train_pr_os, y_train_encoded_os, X_test_pr, y_test,
                show_train_report=False)
```

Report of XGBClassifier type model using train-test split dataset.

```
*****************************************************************************************
Train accuracy score: 1.0
Test accuracy score: 0.9714
```

```
   No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
*****************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1686
           1       0.91      0.91      0.91       340

    accuracy                           0.97      2026
   macro avg       0.95      0.95      0.95      2026
weighted avg       0.97      0.97      0.97      2026

************************************************************
```



In [72]:

```
## Grid search
```

In [73]:

```
xgg_clf_gs = XGBClassifier(
    n_jobs=-1, verbosity=0, objective='binary:logistic',
    eval_metric='error')  #“rank:pairwise”,“count:poisson” #'logloss','auc'
params = {
    'criterion': ["gini", "entropy"],
    'max_depth': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'class_weight': ["balanced", "balanced_subsample"],
    'ccp_alpha': [0.0, 0.05, 0.1, 0.2, 0.3],
    'importance_type':
    ["gain", "weight", "cover", "total_gain", "total_cover"],
}
gridsearch_xgg_clf_gs = GridSearchCV(
    estimator=xgg_clf_gs, param_grid=params, n_jobs=-1,
    scoring='precision')  #'roc_auc_ovr_weighted'
gridsearch_xgg_clf_gs
```

Out[73]:

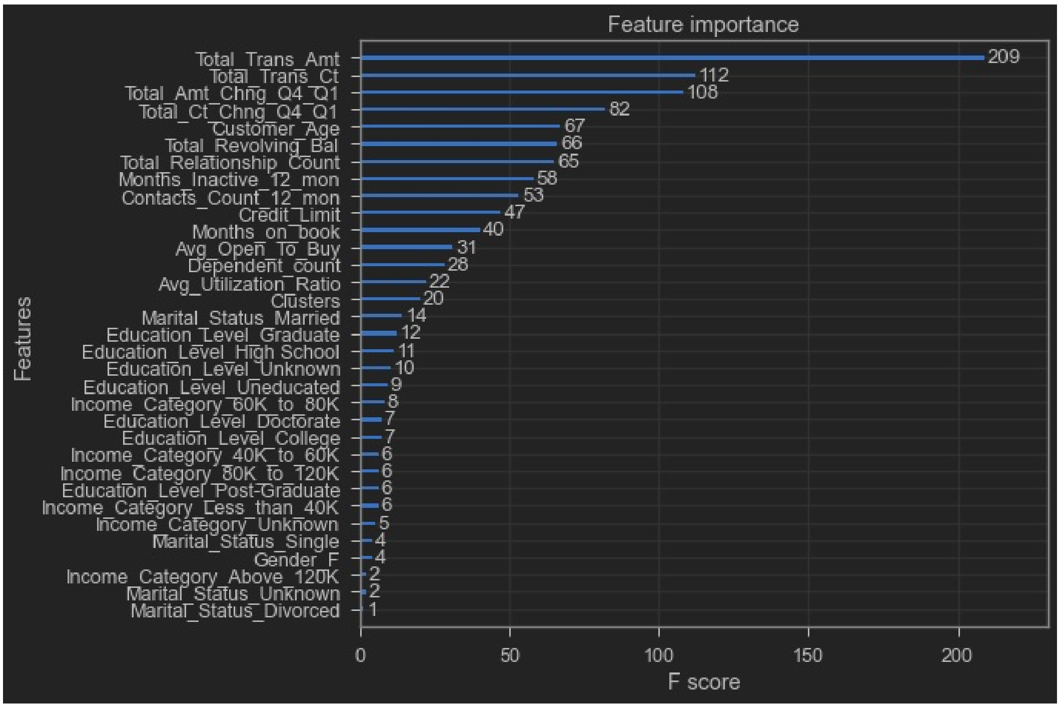
```
GridSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, eval_metric='error',
                                     gamma=None, gpu_id=None,
                                     importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estima...
                                     scale_pos_weight=None, subsample=None,
                                     tree_method=None, validate_parameters=None,
                                     verbosity=0),
             n_jobs=-1,
```

```
            param_grid={'ccp_alpha': [0.0, 0.05, 0.1, 0.2, 0.3],
                        'class_weight': ['balanced', 'balanced_subsample'],
                        'criterion': ['gini', 'entropy'],
                        'importance_type': ['gain', 'weight', 'cover',
                                            'total_gain', 'total_cover'],
                        'max_depth': [2, 3, 4],
                        'min_samples_leaf': [1, 2, 3, 4]},
            scoring='precision')
```

In [ ]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    gridsearch_xgg_clf_gs.fit(X_train_pr_os, y_train_encoded_os)

xgg_clf_gs_best = gridsearch_xgg_clf_gs.best_estimator_
```

In [76]:
```
xgb.plot_importance(xgg_clf_gs_best);
```

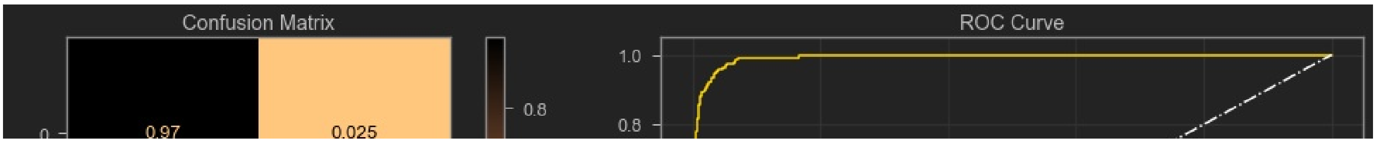
Out[76]:
```
<AxesSubplot:title={'center':'Feature importance'}, xlabel='F score', ylabel='Features'>
```

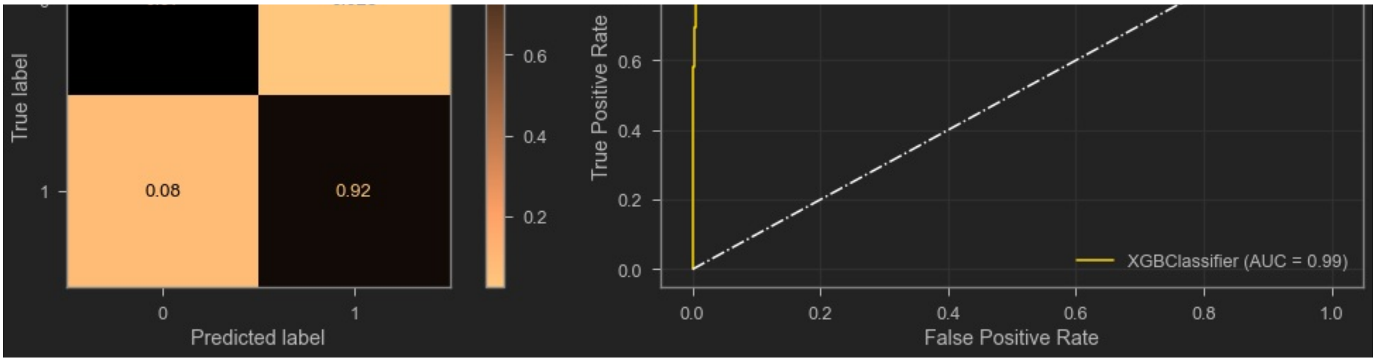


In [75]:
```
fn.model_report(xgg_clf_gs_best, X_train_pr_os, y_train_encoded_os, X_test_pr, y_test,
                show_train_report=False)
```

Report of XGBClassifier type model using train-test split dataset.

```
*****************************************************************************************
Train accuracy score: 0.9971
Test accuracy score: 0.9659
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
*****************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1702
           1       0.87      0.92      0.90       324

    accuracy                           0.97      2026
   macro avg       0.93      0.95      0.94      2026
weighted avg       0.97      0.97      0.97      2026

************************************************************
```



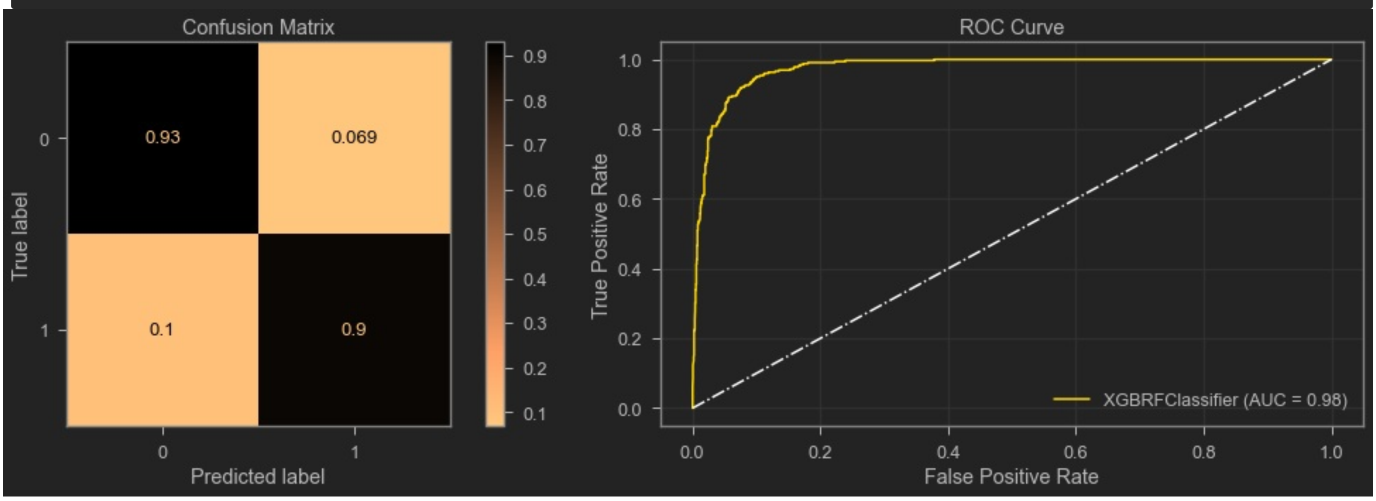


XGBRFClassifier

In [77]:
```
clf_xg_rf = XGBRFClassifier(n_jobs=-1)
fn.model_report(clf_xg_rf, X_train_pr_os, y_train_encoded_os, X_test_pr, y_test,
                show_train_report=False)
```

Report of XGBRFClassifier type model using train-test split dataset.

```
*******************************************************************************************
Train accuracy score: 0.9518
Test accuracy score: 0.9255
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
*******************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.98      0.93      0.95      1702
           1       0.71      0.90      0.79       324

    accuracy                           0.93      2026
   macro avg       0.85      0.91      0.87      2026
weighted avg       0.94      0.93      0.93      2026

************************************************************
```



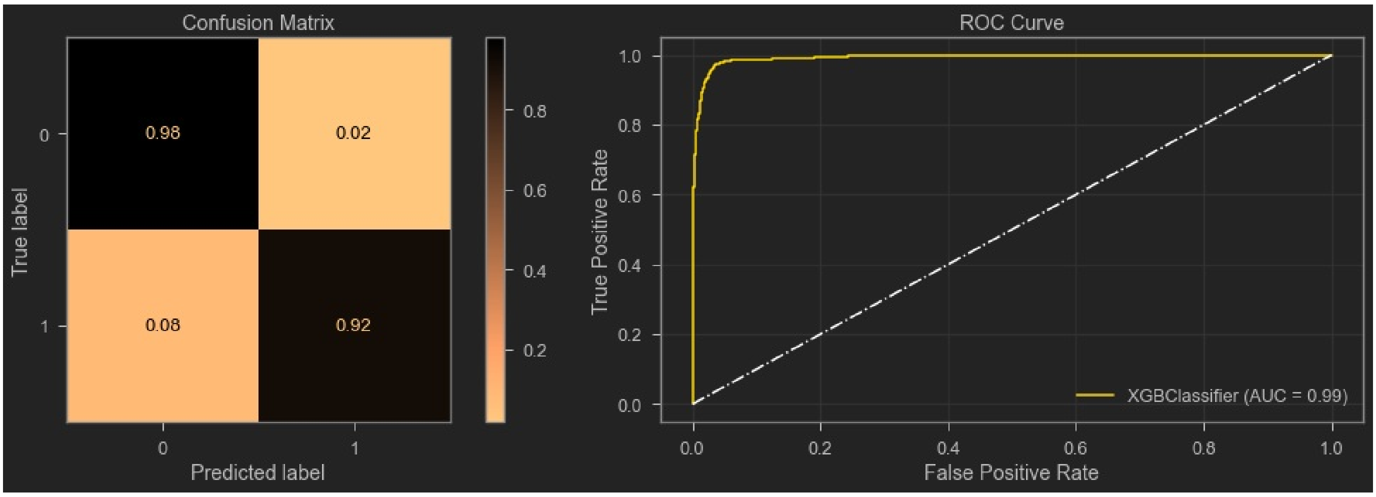
Best model

In [78]:
```
fn.model_report(clf_xg,
                X_train_pr_os,
                y_train_encoded_os,
                X_test_pr,
                y_test,
                show_train_report=False,
                fitted_model=True)
```

Report of XGBClassifier type model using train-test split dataset.

```
*******************************************************************************************
```

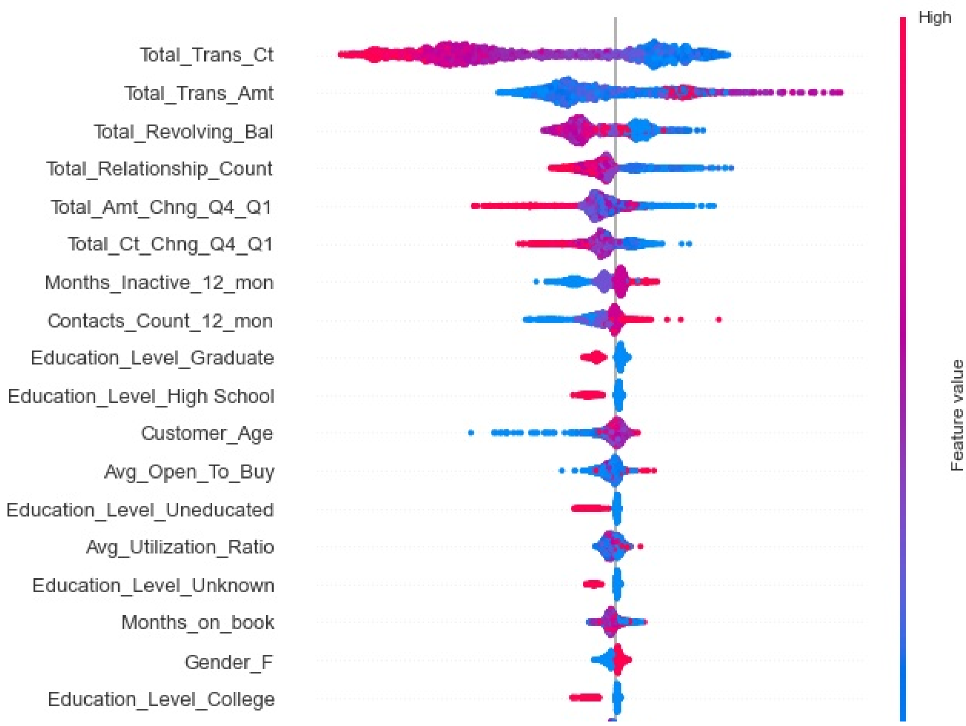
```
Train accuracy score: 1.0
Test accuracy score: 0.9704
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
****************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1702
           1       0.90      0.92      0.91       324

    accuracy                           0.97      2026
   macro avg       0.94      0.95      0.95      2026
weighted avg       0.97      0.97      0.97      2026

************************************************************
```

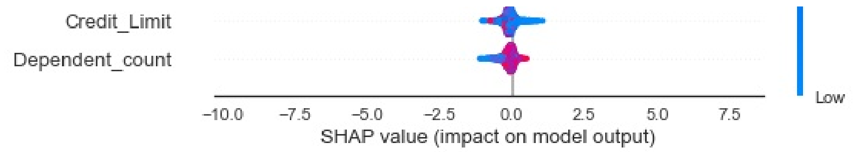


In [78]:
```python
# init shap
shap.initjs()
```

In [81]:
```python
explainer = shap.TreeExplainer(clf_xg)
shap_values = explainer.shap_values(X_test_pr)
```

In [83]:
```python
with plt.style.context('seaborn-white'):
    shap.summary_plot(shap_values, X_test_pr)
```





In [85]:
```
eli5.format_as_dataframe(eli5.explain_weights(
    clf_xg, feature_names=list(X_test_pr.columns)))
```

Out[85]:

	feature	weight
0	Total_Trans_Ct	0.261081
1	Total_Revolving_Bal	0.079481
2	Total_Relationship_Count	0.072611
3	Total_Trans_Amt	0.054719
4	Gender_F	0.046512
5	Months_Inactive_12_mon	0.039577
6	Education_Level_College	0.038613
7	Contacts_Count_12_mon	0.032720
8	Total_Ct_Chng_Q4_Q1	0.030708
9	Income_Category_60K_to_80K	0.030558
10	Education_Level_Unknown	0.030028
11	Education_Level_Uneducated	0.027573
12	Total_Amt_Chng_Q4_Q1	0.021398
13	Customer_Age	0.019571
14	Education_Level_Doctorate	0.018397
15	Avg_Open_To_Buy	0.017202
16	Education_Level_High School	0.016947
17	Income_Category_40K_to_60K	0.015005
18	Marital_Status_Unknown	0.014051
19	Income_Category_Unknown	0.013676

In [ ]:
```
# Save segmentation model
# get params of best model
# save model after fitting on entire dataset
```

INTERPRET

Customer Segmentation model

In [ ]:
```
# with churn
```

Churn Prediction model

In [ ]:
```

```

RECOMMENDATION

In [ ]:
```
# Reflection on interpretion
```

CONCLUSION

In [ ]:
```
# caviats
```

In [ ]:

```

```

NEXT STEPS

Modeling aspect: Gaussian Mixture Models for segmentation modeling, and Neural Network based approach for prediction model.

Business need aspect: A part of the business challenge is determining how soon you want the model to forecast. A prediction that is made too long in advance may be less accurate. A narrow prediction horizon, on the other hand, may perform better in terms of accuracy, but it may be too late to act after the consumer has made her decision.

Finally, it is critical to establish whether churn should be characterized at the product level (customers who are likely to discontinue using a certain product, such as a credit card) or at the relationship level (client likely to extricate from the bank itself). When data is evaluated at the relationship level, you gain a wider insight of the customer's perspective. Excessive withdrawals from a savings account, for example, may be used to pay for a deposit on a house or education costs. Such insights into client life events are extremely effective not just for preventing churn, but also for cross-selling complementary items that may enhance the engagement even further.

APPENDIX

all functions and imports from the `functions.py` and `packages.py`

In [120…

```python
# functions used in various steps of this analysis
fn.show_py_file_content(file='./imports_and_functions/functions.py')
```

```python
# imports
import matplotlib.pyplot as plt
from sklearn import metrics
from IPython.display import display, HTML, Markdown
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler, StandardScaler
from sklearn.compose import ColumnTransformer
from yellowbrick.classifier.rocauc import roc_auc
import seaborn as sns
import numpy as np
import plotly.express as px

# functions


def model_report(model,
                 X_train,
                 y_train,
                 X_test,
                 y_test,
                 show_train_report=True,
                 show_test_report=True,
                 fitted_model=False,
                 cmap=['cool', 'copper_r'],
                 normalize='true',
                 figsize=(15, 5)):
    """
    Dispalys model report.


    """
    if fitted_model is False:
        model.fit(X_train, y_train)
    train = model.score(X_train, y_train)
    test = model.score(X_test, y_test)

    def str_model_(model):
        """Helper function to get model class display statement, this text conversion breaks code if
        performed in ``model_report`` function's local space. This function is to isolate from the
        previous function's local space."""
        str_model = str(model.__class__).split('.')[-1][:-2]
        display(
            HTML(
                f"""<strong>Report of {str_model} type model using train-test split dataset.</strong>"
```

```
""
            ))

    str_model_(model)
    print(f"{'*'*90}")
    print(f"""Train accuracy score: {train.round(4)}""")
    print(f"""Test accuracy score: {test.round(4)}""")
    if abs(train - test) <= .05:
        print(
            f"    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold."
        )
    elif (train - test) > .05:
        print(
            f"    Possible Overfitting, diffrence of scores {round(abs(train-test)*100,2)}% crossed 5%
thresh hold."
        )
    elif (train - test) < -.05:
        print(
            f"    Possible Underfitting, diffrence of scores {round(abs(train-test)*100,2)}% crossed 5
% thresh hold."
        )
    print(f"{'*'*90}")
    print("")

    if show_train_report:
        print(f'Train Report: ')
        print(f"{'*'*60}")
        # train report
        # classification report
        print(
            metrics.classification_report(y_train,
                                          model.predict(X_train)))
        print(f"{'*'*60}")
        # Confusion matrix
        fig, ax = plt.subplots(ncols=2, figsize=figsize)
        metrics.plot_confusion_matrix(model,
                                      X_train,
                                      y_train,
                                      cmap='cool',
                                      normalize='true',
                                      ax=ax[0])
        ax[0].title.set_text('Confusion Matrix')
        # ROC curve
        metrics.plot_roc_curve(model,
                               X_train,
                               y_train,
                               color='#0450E7',
                               ax=ax[1])
        ax[1].plot([0, 1], [0, 1], ls='-.', color='white')
        ax[1].title.set_text('ROC Curve')
        plt.grid()
        plt.tight_layout()
        plt.show()

    if show_test_report:
        # train report
        # classification report
        print(f'Test Report: ')
        print(f"{'*'*60}")
        print(metrics.classification_report(y_test,
                                            model.predict(X_test)))
        print(f"{'*'*60}")
        # Confusion matrix
        fig, ax = plt.subplots(ncols=2, figsize=figsize)
        metrics.plot_confusion_matrix(model,
                                      X_test,
                                      y_test,
                                      cmap='copper_r',
                                      normalize='true',
                                      ax=ax[0])
        ax[0].title.set_text('Confusion Matrix')
        # ROC curve
        metrics.plot_roc_curve(model,
                               X_test,
                               y_test,
                               color='gold',
                               ax=ax[1])
        ax[1].plot([0, 1], [0, 1], ls='-.', color='white')
        ax[1].title.set_text('ROC Curve')
```

```python
        plt.grid()
        plt.tight_layout()
        plt.show()
    pass


def dataset_processor_segmentation(X, OHE_drop_option=None, verbose=0, scaler=None):
    """

    """
    # isolating numerical cols
    nume_col = list(X.select_dtypes('number').columns)
    if verbose > 0:
        print("Numerical columns: \n---------------------\n", nume_col)

    # isolating categorical cols
    cate_col = list(X.select_dtypes('object').columns)
    if verbose > 0:
        print('')
        print("Categorical columns: \n---------------------\n", cate_col)

    # pipeline for processing categorical features
    pipe_cate = Pipeline([('ohe',
                           OneHotEncoder(sparse=False, drop=OHE_drop_option))])
    # pipeline for processing numerical features
    if scaler is None:
        scaler = StandardScaler()
    pipe_nume = Pipeline([('scaler', scaler)])
    # transformer
    preprocessor = ColumnTransformer([('nume_feat', pipe_nume, nume_col),
                                      ('cate_feat', pipe_cate, cate_col)])
    # creating dataframes
    try:
        X_pr = pd.DataFrame(
            preprocessor.fit_transform(X),
            columns=nume_col +
            list(preprocessor.named_transformers_['cate_feat'].
                 named_steps['ohe'].get_feature_names(cate_col)))
        if verbose > 1:
            print("\n\n------")
            print(
                f"Scaler: {str(preprocessor.named_transformers_['nume_feat'].named_steps['scaler'].__c
lass__)[1:-2].split('.')[-1]}, settings: {preprocessor.named_transformers_['nume_feat'].named_steps['s
caler'].get_params()}"
            )
            print(
                f"Encoder: {str(preprocessor.named_transformers_['cate_feat'].named_steps['ohe'].__cla
ss__)[1:-2].split('.')[-1]}, settings: {preprocessor.named_transformers_['cate_feat'].named_steps['ohe
'].get_params()}"
            )
            print("------")
    except:
        if verbose > 1:
            print("\n\n------")
            print(
                f"Scaler: {str(preprocessor.named_transformers_['nume_feat'].named_steps['scaler'].__c
lass__)[1:-2].split('.')[-1]}, settings: {preprocessor.named_transformers_['nume_feat'].named_steps['s
caler'].get_params()}"
            )
            print(
                f"Encoder: {str(preprocessor.named_transformers_['cate_feat'].named_steps['ohe'].__cla
ss__)[1:-2].split('.')[-1]}, settings: {preprocessor.named_transformers_['cate_feat'].named_steps['ohe
'].get_params()}"
            )
            print("------")
            print("No Categorical columns found")
        X_pr = pd.DataFrame(preprocessor.fit_transform(X), columns=nume_col)
    return X_pr


def show_py_file_content(file='./imports_and_functions/functions.py'):
    """
    displays content of a py file output formatted as python code in jupyter notebook.

    Parameter:
    ==========
    file = `str`; default: './imports_and_functions/functions.py',
                path to the py file.
    """
```

```python
        with open(file, 'r', encoding="utf8") as f:
            x = f"""```python
{f.read()}
```"""
            display(Markdown(x))

def model_report_multiclass(model,
                            X_train,
                            y_train,
                            X_test,
                            y_test,
                            show_train_report=True,
                            show_test_report=True,
                            fitted_model=False,
                            cmap=['cool', 'copper_r'],
                            normalize='true',
                            figsize=(15, 5)):
    """
    Dispalys model report.
    """
    if fitted_model is False:
        model.fit(X_train, y_train)
    train = model.score(X_train, y_train)
    test = model.score(X_test, y_test)

    def str_model_(model):
        """Helper function to get model class display statement, this text conversion breaks code if
        performed in ``model_report`` function's local space. This function is to isolate from the
        previous function's local space."""
        str_model = str(model.__class__).split('.')[-1][:-2]
        display(
            HTML(
                f"""Report of {str_model} type model using train-test split dataset."
""
            ))

    str_model_(model)
    print(f"{'*'*90}")
    print(f"""Train accuracy score: {train.round(4)}""")
    print(f"""Test accuracy score: {test.round(4)}""")
    if abs(train - test) <= .05:
        print(
            f"   No over or underfitting detected, diffrence of scores did not cross 5% thresh hold."
)
    elif (train - test) > .05:
        print(
            f"   Possible Overfitting, diffrence of scores {round(abs(train-test)*100,2)}% crossed 5%
thresh hold."
)
    elif (train - test) < -.05:
        print(
            f"   Possible Underfitting, diffrence of scores {round(abs(train-test)*100,2)}% crossed 5
% thresh hold."
)
    print(f"{'*'*90}")
    print("")

    if show_train_report:
        print(f'Train Report: ')
        print(f"{'*'*60}")
        # train report
        # classification report
        print(
            metrics.classification_report(y_train,
                                          model.predict(X_train)))
        print(f"{'*'*60}")
        # Confusion matrix
        fig, ax = plt.subplots(ncols=2, figsize=figsize)
        metrics.plot_confusion_matrix(model,
                                      X_train,
                                      y_train,
                                      cmap='cool',
                                      normalize='true',
                                      ax=ax[0])
        ax[0].title.set_text('Confusion Matrix')
        # ROC curve
        _ = roc_auc(model,
                    X_train,
```
```

```
                    y_train,
                    classes=None,
                    is_fitted=True,
                    show=False,
                    ax=ax[1])

        ax[1].grid()
        ax[1].title.set_text('ROC Curve')
        plt.xlim([-.05, 1])
        plt.ylim([0, 1.05])
        plt.tight_layout()
        plt.show()

    if show_test_report:
        # train report
        # classification report
        print(f'Test Report: ')
        print(f"{'*'*60}")
        print(metrics.classification_report(y_test,
                                            model.predict(X_test)))
        print(f"{'*'*60}")
        # Confusion matrix
        fig, ax = plt.subplots(ncols=2, figsize=figsize)
        metrics.plot_confusion_matrix(model,
                                      X_test,
                                      y_test,
                                      cmap='copper_r',
                                      normalize='true',
                                      ax=ax[0])
        ax[0].title.set_text('Confusion Matrix')
        # ROC curve
        _ = roc_auc(model,
                    X_test,
                    y_test,
                    classes=None,
                    is_fitted=True,
                    show=False,
                    ax=ax[1])
        plt.xlim([-.05, 1])
        plt.ylim([0, 1.05])
        ax[1].grid()
        ax[1].title.set_text('ROC Curve')
        plt.tight_layout()
        plt.show()
    pass


def plot_distribution(df,
                      color='gold',
                      figsize=(16, 26),
                      fig_col=3,
                      labelrotation=45,
                      plot_title='Histogram plots of the dataset'):
    def num_col_for_plotting(row, col=fig_col):
        """
        +++ formatting helper function +++
        __________________________________
        Returns number of rows to plot

        Parameters:
        ===========
        row = int;
        col = int; default col: 3
        """
        if row % col != 0:
            return (row // col) + 1
        else:
            return row // col

    fig, axes = plt.subplots(nrows=num_col_for_plotting(len(df.columns),
                                                        col=fig_col),
                             ncols=fig_col,
                             figsize=figsize,
                             sharey=False)
    for ax, column in zip(axes.flatten(), df):
        sns.histplot(x=column, data=df, color=color, ax=ax, kde=True)
        ax.set_title(f'Histplot of {column.title()}')
        ax.tick_params('x', labelrotation=labelrotation)
        sns.despine()
```

```python
        plt.tight_layout()
        plt.suptitle(plot_title, fontsize=20, fontweight=3, va='bottom')
    plt.show()
    pass


def heatmap_of_features(df, figsize=(15, 15), annot_format='.1f'):
    """
    Return a masked heatmap of the given DataFrame

    Parameters:
    ===========
    df            = pandas.DataFrame object.
    annot_format  = str, for formatting; default: '.1f'

    Example of `annot_format`:
    --------------------------
    .1e = scientific notation with 1 decimal point (standard form)
    .2f = 2 decimal places
    .3g = 3 significant figures
    .4% = percentage with 4 decimal places

    Note:
    =====
    Rounding error can happen if '.1f' is used.

    -- version: 1.1 --
    """
    with plt.style.context('dark_background'):
        plt.figure(figsize=figsize, facecolor='k')
        mask = np.triu(np.ones_like(df.corr(), dtype=bool))
        cmap = sns.diverging_palette(3, 3, as_cmap=True)
        ax = sns.heatmap(df.corr(),
                         mask=mask,
                         cmap=cmap,
                         annot=True,
                         fmt=annot_format,
                         linecolor='k',
                         annot_kws={"size": 9},
                         square=False,
                         linewidths=.5,
                         cbar_kws={"shrink": .5})
        plt.title(f'Features heatmap', fontdict={"size": 20})
        plt.show()
        return ax


def drop_features_based_on_correlation(df, threshold=0.75):
    """
    Returns features with high collinearity.

    Parameters:
    ===========
    df = pandas.DataFrame; no default.
            data to work on.
    threshold = float; default: .75.
            Cut off value of check of collinearity.

    -- ver: 1.0 --
    """
    # Set of all the names of correlated columns
    feature_corr = set()
    corr_matrix = df.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            # absolute coeff value
            if abs(corr_matrix.iloc[i, j]) > threshold:
                # getting the name of column
                colname = corr_matrix.columns[i]
                feature_corr.add(colname)
    return feature_corr


def cluster_insights(df, color=px.colors.qualitative.Pastel):
    """

    """
    # fig 1 Age
    financials = [
```

```
        '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'
    ]
    fig = px.histogram(df,
                       x='Customer_Age',
                       marginal="box",
                       template='presentation',
                       nbins=10,
                       color='Gender',
                       barmode='group', color_discrete_sequence=color,
                       title='Customer Demographics',
                       hover_data=df)
    fig.update_traces(opacity=0.8)
    fig.update_layout(bargap=0.05)
    fig.show()
    # fig 2 Education
    fig = px.histogram(df,
                       color='Education_Level',
                       marginal="box",
                       template='presentation', color_discrete_sequence=color,
                       category_orders=dict(Income_Category=[
                           'Unknown', 'Less_than_40K', '40K_to_60K',
                           '60K_to_80K', '80K_to_120K', 'Above_120K'
                       ]),
                       title='Education Level & Income Category',
                       x='Income_Category',
                       barmode='group',
                       hover_data=df)
    # fig.update_layout(width=700, height=500, bargap=0.05)
    fig.show()
    # fig 4 dependent count
    fig = px.histogram(df,
                       x='Dependent_count',
                       marginal="box",
                       template='presentation', color_discrete_sequence=color,
                       title='Marital Status & Dependent count',
                       color='Marital_Status',
                       barmode='group',
                       hover_data=df)
    fig.update_traces(opacity=0.8)
    fig.update_layout(width=700, height=500, bargap=0.05)
    fig.show()
    # fig 5 Card category
    fig = px.bar(x='Card_Category',
                 color='Card_Category',
                 data_frame=df,
                 template='presentation',
                 title='Card Category',
                 color_discrete_sequence=["blue", "gold", "silver", "#c1beba"])
    fig.update_layout(width=700, height=500, bargap=0.05)
    fig.show()
    # fig 6
    plot_distribution(df[financials], color='silver', figsize=(
        16, 16), plot_title='Histogram of Numreical features')
    plt.show()
    pass


def describe_dataframe(df):
    """

    """
    left = df.describe(include='all').round(2).T
    right = pd.DataFrame(df.dtypes)
    right.columns = ['dtype']
    ret_df = pd.merge(left=left, right=right,
                      left_index=True, right_index=True)
    na_df = pd.DataFrame(df.isna().sum())
    na_df.columns = ['nulls']
    ret_df = pd.merge(left=ret_df, right=na_df,
                      left_index=True, right_index=True)
    ret_df.fillna('', inplace=True)
    return ret_df


def check_duplicates(df, verbose=0, limit_output=True, limit_num=150):
    """
```

```python
    Checks for duplicates in the pandas DataFrame and return a Dataframe of report.

    Parameters:
    ===========
    df = pandas.DataFrame
    verbose = `int` or `boolean`; default: `False`
    limit_output = `int` or `boolean`; default: `True`
                `True` limits featurs display to 150.
                `False` details of unique features.
    limit_num = `int`, limit number of uniques; default: 150,

    Returns:
    ========
    pandas.DataFrame, if verbose = 1.

    ---version 1.3---
    """
    dup_checking = []
    for column in df.columns:
        not_duplicated = df[column].duplicated().value_counts()[0]
        try:
            duplicated = df[column].duplicated().value_counts()[1]
        except:
            duplicated = 0
        temp_dict = {
            'name': column,
            'duplicated': duplicated,
            'not_duplicated': not_duplicated
        }
        dup_checking.append(temp_dict)
    df_ = pd.DataFrame(dup_checking)

    if verbose > 0:
        if limit_output:
            for col in df:
                if (len(df[col].unique())) <= limit_num:
                    print(
                        f"{col} >> number of uniques: {len(df[col].unique())}\nValues:\n{df[col].uniqu
e()}")
                else:
                    print(
                        f"{col} >> number of uniques: {len(df[col].unique())}, showing top {limit_num}
values\nTop {limit_num} Values:\n{df[col].unique()[:limit_num]}\n")
                print(f"{'_'*60}\n")
        else:
            for col in df:
                print(
                    f"{col} >> number of uniques: {len(df[col].unique())}\nValues:\n{df[col].unique()}
")
    if 1 > verbose >= 0:
        return df_


def unseen_data_processor(X, preprocessor, nume_col, cate_col):
    ret_df = pd.DataFrame(preprocessor.transform(X),
                          columns=nume_col +
                          list(preprocessor.named_transformers_['cate_feat'].
                               named_steps['ohe'].get_feature_names(cate_col)))
    return ret_df


def feature_analysis_intracluster(
        df, cluster_df, n_clusters, title=None,
        nbins=None, marginal='box', histnorm='probability density',
        color_discrete_sequence=px.colors.qualitative.Pastel,
        template='presentation'):
    if title is None:
        title = f'{df.name.replace("_"," ")}'
    fig = px.histogram(
        data_frame=df,
        facet_col=cluster_df,
        marginal=marginal,
        histnorm=histnorm,
        nbins=nbins,
        color_discrete_sequence=color_discrete_sequence,
        template=template,
        title=title,
        category_orders={
            'Clusters': [range(0, n_clusters)]
```

```
        }).update_layout(showlegend=False)
    fig.for_each_annotation(lambda a: a.update(
        text=f'Cluster: {a.text.split("=")[1]}'))
    return fig


def show_px_color_options(type='qualitative'):
    if type == 'qualitative':
        display(dir(px.colors.qualitative))
    elif type == 'sequential':
        display(dir(px.colors.sequential))
    pass
```

In [119…

```
# imports for this analysis
fn.show_py_file_content(file='./imports_and_functions/packages.py')
```

```
import pandas as pd
import scipy.stats as sts
import numpy as np
from ipywidgets import interact, fixed
import plotly.express as px
import plotly.graph_objs as go
import warnings
from IPython.display import display, HTML, Markdown
import eli5
import shap
from sklearn.preprocessing import OrdinalEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.cluster import KMeans
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTENC
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.compose import ColumnTransformer
from sklearn.cluster import MeanShift, estimate_bandwidth
from sklearn.inspection import permutation_importance
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler, StandardScaler
from sklearn import metrics
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier, XGBRFClassifier
import xgboost as xgb
from yellowbrick.cluster import intercluster_distance
from yellowbrick.cluster.elbow import kelbow_visualizer
import seaborn as sns
import matplotlib.pyplot as plt
import joblib
```

Dashboard

In [ ]:

```

```

Logistic regression with no category dropped for categorical columns

In [93]:

```
# dropped first from OHE using cluster prediction
X_log_reg = X.copy()
X_log_reg['cluster'] = clusters
X_log_reg['cluster'] = X_log_reg['cluster'].astype('str')

X_train_log_reg, y_train_log_reg, X_test_log_reg, y_test_log_reg = fn.dataset_processor(
    X_log_reg, y, verbose=3, oversample=False)
```

```
Numerical columns:
---------------------
```

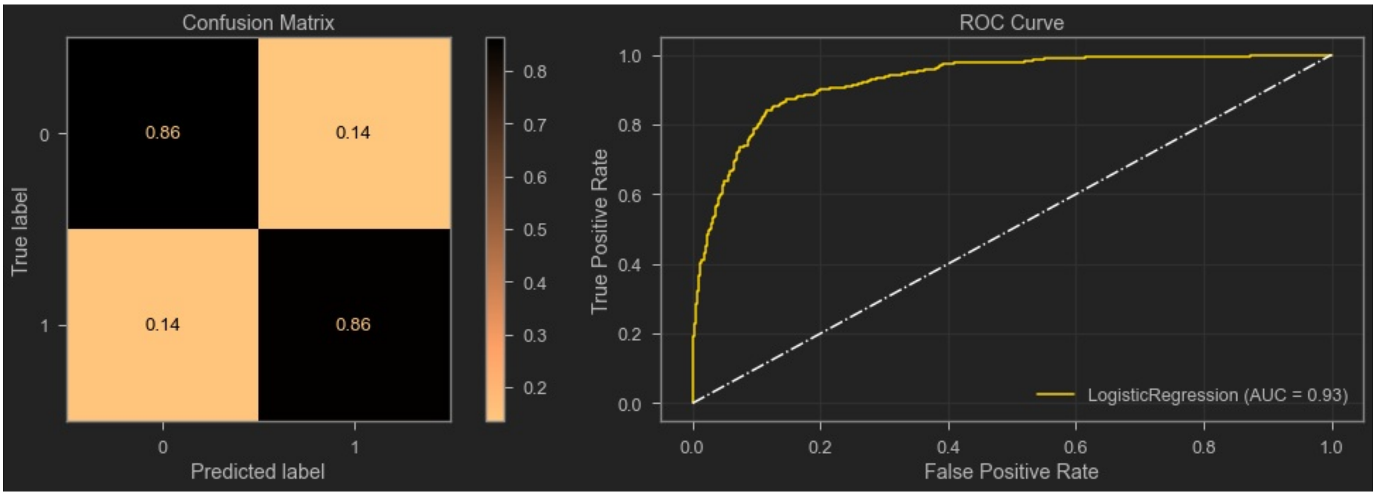
```
 ['Customer_Age', 'Dependent_count', 'Months_on_book', 'Total_Relationship_Count', 'Months_Inactive_12_mon', 'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal', 'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt', 'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio']

Categorical columns:
--------------------
 ['Gender', 'Education_Level', 'Marital_Status', 'Income_Category', 'Card_Category', 'cluster']


------
Scaler: StandardScaler, settings: {'copy': True, 'with_mean': True, 'with_std': True}
Encoder: OneHotEncoder, settings: {'categories': 'auto', 'drop': None, 'dtype': <class 'numpy.float64'>, 'handle_unknown': 'error', 'sparse': False}
------
```

In [90]:

```python
# with all data
logreg_clf = LogisticRegression(max_iter=1000, class_weight='balanced')
# score of logistic regression classifier
fn.model_report(logreg_clf,
                X_train_log_reg,
                y_train_log_reg,
                X_test_log_reg,
                y_test_log_reg,
                show_train_report=False)
```

Report of LogisticRegression type model using train-test split dataset.

```
*******************************************************************************************
Train accuracy score: 0.8631
Test accuracy score: 0.8623
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
*******************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.97      0.86      0.91      1677
           1       0.57      0.86      0.68       349

    accuracy                           0.86      2026
   macro avg       0.77      0.86      0.80      2026
weighted avg       0.90      0.86      0.87      2026

************************************************************
```



In [62]:

```python
fn.drop_features_based_on_correlation(X_train_log_reg, .8)
```

Out[62]:

```
{'Avg_Open_To_Buy', 'Total_Trans_Ct', 'cluster_4'}
```

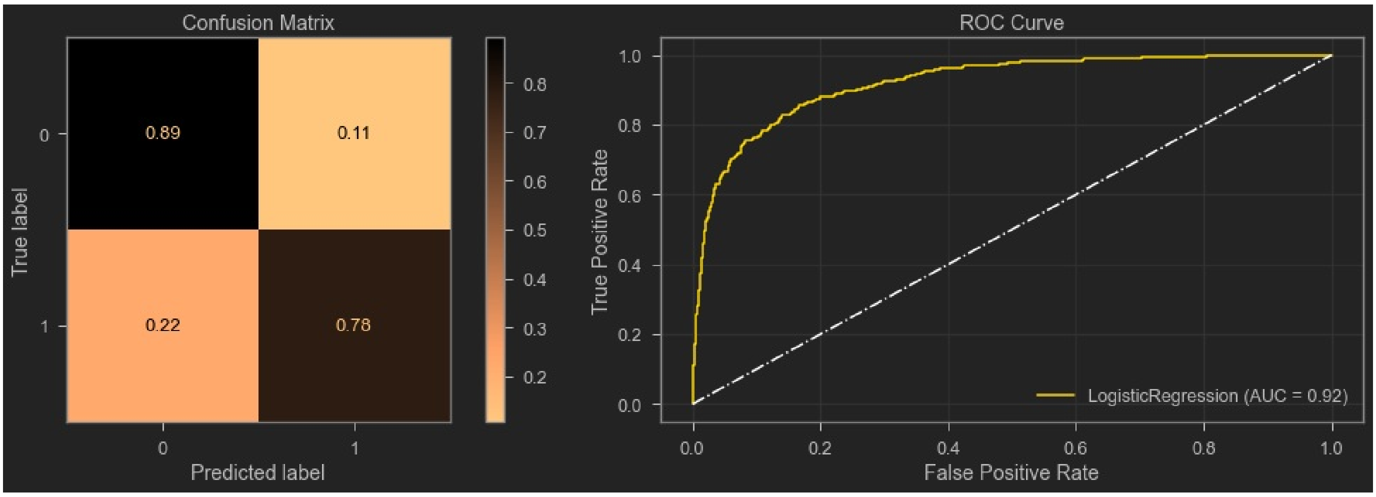
In [63]:

```python
logreg_clf = LogisticRegression(max_iter=1000, class_weight='balanced')
# score of logistic regression classifier
```

```python
fn.model_report(logreg_clf,
                X_train_log_reg.drop(columns=['Avg_Open_To_Buy']),
                y_train_log_reg,
                X_test_log_reg.drop(columns=['Avg_Open_To_Buy']),
                y_test_log_reg,
                show_train_report=False)
```

Report of LogisticRegression type model using train-test split dataset.

```
*****************************************************************************************
Train accuracy score: 0.8977
Test accuracy score: 0.8722
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
*****************************************************************************************


Test Report: 
************************************************************
              precision    recall  f1-score   support

           0       0.95      0.89      0.92      1668
           1       0.61      0.78      0.68       358

    accuracy                           0.87      2026
   macro avg       0.78      0.83      0.80      2026
weighted avg       0.89      0.87      0.88      2026

************************************************************
```



In [64]:

```python
churn_feature = pd.DataFrame(
    logreg_clf.coef_,columns=X_train_log_reg.drop(columns=['Avg_Open_To_Buy']).columns).T
churn_feature.columns = ['weights']
churn_feature['odds_ratio'] = np.exp(churn_feature['weights'])
churn_feature['effect'] = 100 * (churn_feature['odds_ratio'] - 1)
churn_feature
```

Out[64]:

	weights	odds_ratio	effect
Customer_Age	0.020807	1.021025	2.102500
Dependent_count	0.085503	1.089264	8.926430
Months_on_book	0.020645	1.020859	2.085932
Total_Relationship_Count	-0.717614	0.487915	-51.208477
Months_Inactive_12_mon	0.625913	1.869953	86.995261
Contacts_Count_12_mon	0.649716	1.914996	91.499614
Credit_Limit	0.066191	1.068431	6.843083
Total_Revolving_Bal	-0.525134	0.591476	-40.852387
Total_Amt_Chng_Q4_Q1	-0.112877	0.893260	-10.673977
Total_Trans_Amt	3.241114	25.562173	2456.217309
Total_Trans_Ct	-3.737465	0.023814	-97.618560
Total_Ct_Chng_Q4_Q1	-0.815076	0.442606	-55.739442

Avg_Utilization_Ratio	-0.113203	0.892970	-10.703048
Gender_M	-1.160080	0.313461	-68.653879
Education_Level_Doctorate	-2.582524	0.075583	-92.441702
Education_Level_Graduate	-1.927811	0.145466	-85.453367
Education_Level_High School	-2.228764	0.107661	-89.233857
Education_Level_Post-Graduate	-2.678308	0.068679	-93.132076
Education_Level_Uneducated	-2.785212	0.061716	-93.828400
Education_Level_Unknown	-1.649395	0.192166	-80.783394
Marital_Status_Married	-0.492168	0.611300	-38.870038
Marital_Status_Single	0.036915	1.037605	3.760530
Marital_Status_Unknown	-0.174750	0.839667	-16.033269
Income_Category_60K_to_80K	-0.051284	0.950009	-4.999094
Income_Category_80K_to_120K	0.433955	1.543350	54.335002
Income_Category_Above_120K	0.609378	1.839287	83.928713
Income_Category_Less_than_40K	-0.004265	0.995744	-0.425593
Income_Category_Unknown	-0.118802	0.887984	-11.201643
Card_Category_Gold	0.551665	1.736141	73.614127
Card_Category_Platinum	0.660023	1.934837	93.483698
Card_Category_Silver	0.266080	1.304840	30.483953
cluster_1	4.859941	129.016527	12801.652674
cluster_2	4.612288	100.714350	9971.435038
cluster_3	3.833848	46.240141	4524.014145
cluster_4	4.172268	64.862375	6386.237487

In [ ]:

```

```

SVC

In [234…

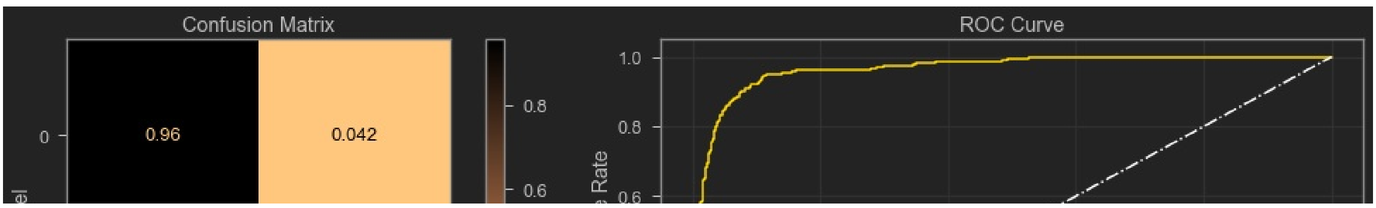
```
# clf_svc = SVC(kernel='linear', C=100, class_weight='balanced')
# clf_svc = SVC(kernel='rbf', C=1, gamma='auto', class_weight='balanced', tol=.8)
clf_svc = SVC(kernel='poly', degree=4, C=1, gamma='scale', class_weight='balanced')
# clf_svc = SVC(kernel='sigmoid', C=2, class_weight='balanced')
```

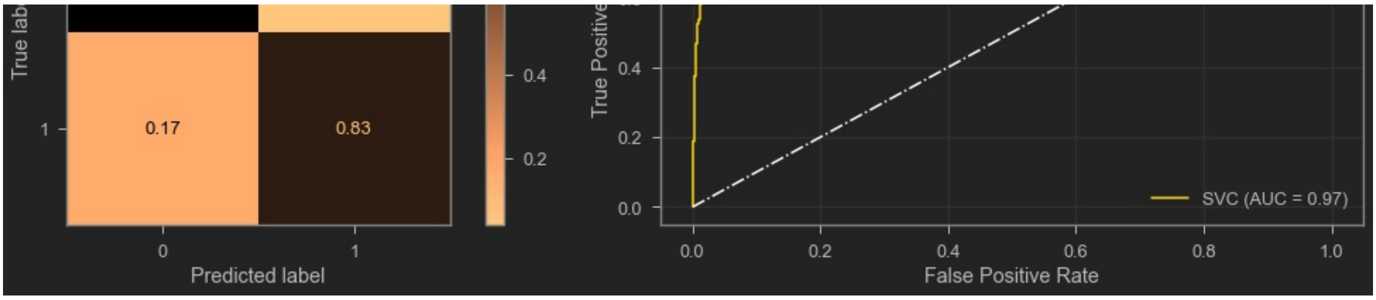
In [235…

```
fn.model_report(clf_svc, X_train_pr_os, y_train_encoded_os, X_test_pr, y_test,
                show_train_report=False)
```

Report of SVC type model using train-test split dataset.

```
****************************************************************************************
Train accuracy score: 0.9742
Test accuracy score: 0.9368
    No over or underfitting detected, diffrence of scores did not cross 5% thresh hold.
****************************************************************************************

Test Report:
************************************************************
              precision    recall  f1-score   support

           0       0.97      0.96      0.96      1699
           1       0.79      0.83      0.81       327

    accuracy                           0.94      2026
   macro avg       0.88      0.89      0.89      2026
weighted avg       0.94      0.94      0.94      2026

************************************************************
```




True label
1
0.17
0.83
0
1
Predicted label
0.4
0.2
True Positive
0.4
0.2
0.0
SVC (AUC = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate