Credit Card Customers Segmentation

The dataset for this project originates from the [Credit Card Dataset for Clustering](#).

Background:

Not all customers are alike. Consumers usually show a wide variety of behaviors. A lot of times, Segments that are used in businesses are threshold based. With growing number of features and a general theme of personlized products, there is a need for a scietific based methodology to group customers together. Clustering based on the behavioral data comes to the rescue. The aim of this analysis is to group credit card holders in appropriate groups to better understand their needs and behaviors and to serve them better with appropriate marketing offers.

Problem Statement:

In this project, we need to extract segments of customers depending on their behaviour patterns provided in the dataset, to focus marketing strategy of the company on a particular segment.

Attribute Information:

1) *CUSTID:* Identification of Credit Card holder (Categorical)

2) *BALANCE:* Balance amount left in their account to make purchases

3) *BALANCE_FREQUENCY:* How frequently the Balance is updated, score between 0 and 1

4) *PURCHASES:* Amount of purchases made from account

5) *ONEOFF_PURCHASES:* Maximum purchase amount done in one-go

6) *INSTALLMENTS_PURCHASES:* Amount of purchase done in installment

7) *CASH_ADVANCE:* Cash in advance given by the user

8) *PURCHASES_FREQUENCY:* How frequently the Purchases are being made, score between 0 and 1

9) *ONEOFF_PURCHASES_FREQUENCY:* How frequently Purchases are happening in one-go

10) *PURCHASES_INSTALLMENTS_FREQUENCY:* How frequently purchases in installments are being done

11) *CASH_ADVANCE_FREQUENCY:* How frequently the cash in advance being paid

12) *CASH_ADVANCE_TRX:* Number of Transactions made with "Cash in Advanced"

13) *PURCHASES_TRX:* Number of purchase transactions made

14) *CREDIT_LIMIT:* Limit of Credit Card for user

15) *PAYMENTS:* Amount of Payment done by user

16) *MINIMUM_PAYMENTS:* Minimum amount of payments made by user

17) *PRC_FULL_PAYMENT:* Percent of full payment paid by user

18) *TENURE:* Tenure of credit card service for user

Table of Contents

1. Environment Setup

goto toc

1.1. Install Packages

Install required packages

goto toc

In [1]:

```
# Install pandas
! pip install pandas

# Install matplotlib
! pip install matplotlib

# Install seaborn
! pip install seaborn

# Install sklearn
! pip install sklearn

# Install tqdm to visualize iterations
! pip install tqdm
```

```
Requirement already satisfied: pandas in c:\users\arun\anaconda3\envs\data_science\l
ib\site-packages (1.2.4)
Requirement already satisfied: pytz>=2017.3 in c:\users\arun\anaconda3\envs\data_sci
```

```
ence\lib\site-packages (from pandas) (2021.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\arun\anaconda3\env
s\data_science\lib\site-packages (from pandas) (2.8.1)
Requirement already satisfied: numpy>=1.16.5 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from pandas) (1.20.1)
Requirement already satisfied: six>=1.5 in c:\users\arun\anaconda3\envs\data_science
\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
Requirement already satisfied: matplotlib in c:\users\arun\anaconda3\envs\data_scien
ce\lib\site-packages (3.3.4)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users
\arun\anaconda3\envs\data_science\lib\site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\arun\anaconda3\envs\dat
a_science\lib\site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from matplotlib) (8.2.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\arun\anaconda3\envs
\data_science\lib\site-packages (from matplotlib) (2.8.1)
Requirement already satisfied: numpy>=1.15 in c:\users\arun\anaconda3\envs\data_scie
nce\lib\site-packages (from matplotlib) (1.20.1)
Requirement already satisfied: cycler>=0.10 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: six in c:\users\arun\anaconda3\envs\data_science\lib
\site-packages (from cycler>=0.10->matplotlib) (1.15.0)
Requirement already satisfied: seaborn in c:\users\arun\anaconda3\envs\data_science
\lib\site-packages (0.11.1)
Requirement already satisfied: matplotlib>=2.2 in c:\users\arun\anaconda3\envs\data_
science\lib\site-packages (from seaborn) (3.3.4)
Requirement already satisfied: scipy>=1.0 in c:\users\arun\anaconda3\envs\data_scien
ce\lib\site-packages (from seaborn) (1.6.2)
Requirement already satisfied: numpy>=1.15 in c:\users\arun\anaconda3\envs\data_scie
nce\lib\site-packages (from seaborn) (1.20.1)
Requirement already satisfied: pandas>=0.23 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from seaborn) (1.2.4)
Requirement already satisfied: pillow>=6.2.0 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from matplotlib>=2.2->seaborn) (8.2.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\arun\anaconda3\envs\dat
a_science\lib\site-packages (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\arun\anaconda3\envs
\data_science\lib\site-packages (from matplotlib>=2.2->seaborn) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users
\arun\anaconda3\envs\data_science\lib\site-packages (from matplotlib>=2.2->seaborn)
(2.4.7)
Requirement already satisfied: six in c:\users\arun\anaconda3\envs\data_science\lib
\site-packages (from cycler>=0.10->matplotlib>=2.2->seaborn) (1.15.0)
Requirement already satisfied: pytz>=2017.3 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from pandas>=0.23->seaborn) (2021.1)
Requirement already satisfied: sklearn in c:\users\arun\anaconda3\envs\data_science
\lib\site-packages (0.0)
Requirement already satisfied: scikit-learn in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from sklearn) (0.24.1)
Requirement already satisfied: numpy>=1.13.3 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from scikit-learn->sklearn) (1.20.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\arun\anaconda3\envs
\data_science\lib\site-packages (from scikit-learn->sklearn) (2.1.0)
Requirement already satisfied: scipy>=0.19.1 in c:\users\arun\anaconda3\envs\data_sc
ience\lib\site-packages (from scikit-learn->sklearn) (1.6.2)
Requirement already satisfied: joblib>=0.11 in c:\users\arun\anaconda3\envs\data_sci
ence\lib\site-packages (from scikit-learn->sklearn) (1.0.1)
Requirement already satisfied: tqdm in c:\users\arun\anaconda3\envs\data_science\lib
\site-packages (4.59.0)
```

1.2. Load Dependencies

Import required packages

goto toc

In [2]:
```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from tqdm import tqdm

# Pretty display for notebooks
%matplotlib inline

import seaborn as sns

# Set default setting of seaborn
sns.set()
```

2. Load dataset

Read data from credit_card.csv file using pandas method read_csv().

goto toc

In [3]:
```
# read the data
raw_data = pd.read_csv("data/credit_card.csv")

# print the first five rows of the data
raw_data.head()
```

Out[3]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	

3. Data Types and Dimensions

goto toc

In [4]:
```
print("Credit Card Data Set has \033[4m\033[1m{}\033[0m\033[0m data points with \033
```

```
Credit Card Data Set has 8950 data points with 18 variables each.
```

In [5]:
```
# check the data types of the features
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   CUST_ID                           8950 non-null   object 
 1   BALANCE                           8950 non-null   float64
 2   BALANCE_FREQUENCY                 8950 non-null   float64
 3   PURCHASES                         8950 non-null   float64
 4   ONEOFF_PURCHASES                  8950 non-null   float64
 5   INSTALLMENTS_PURCHASES            8950 non-null   float64
 6   CASH_ADVANCE                      8950 non-null   float64
 7   PURCHASES_FREQUENCY               8950 non-null   float64
 8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 11  CASH_ADVANCE_TRX                  8950 non-null   int64  
 12  PURCHASES_TRX                     8950 non-null   int64  
 13  CREDIT_LIMIT                      8949 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8637 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64  
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

4. Data Preprocessing

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.

...goto toc

4.1. Data Cleaning

Data cleaning refers to preparing data for analysis by removing or modifying data that is incomplete, irrelevant, duplicated, or improperly formatted.

...goto toc

Missing Data Treatment

If the missing values are not handled properly we may end up drawing an inaccurate inference about the data. Due to improper handling, the result obtained will differ from the ones where the missing values are present.

In [229…

```
# Create the dataframe
missing_values = pd.DataFrame()

# Get list of all columns
missing_values['Features'] = raw_data.columns.values

# get the count of missing values
missing_values['Count'] = raw_data.isnull().sum().values

# Calculate percentage of missing values
percentage = raw_data.isna().mean()*100
missing_values['Percentange'] = percentage.values
```

```python
# print the dataframe
missing_values.sort_values(ascending = False, by = 'Count')
```

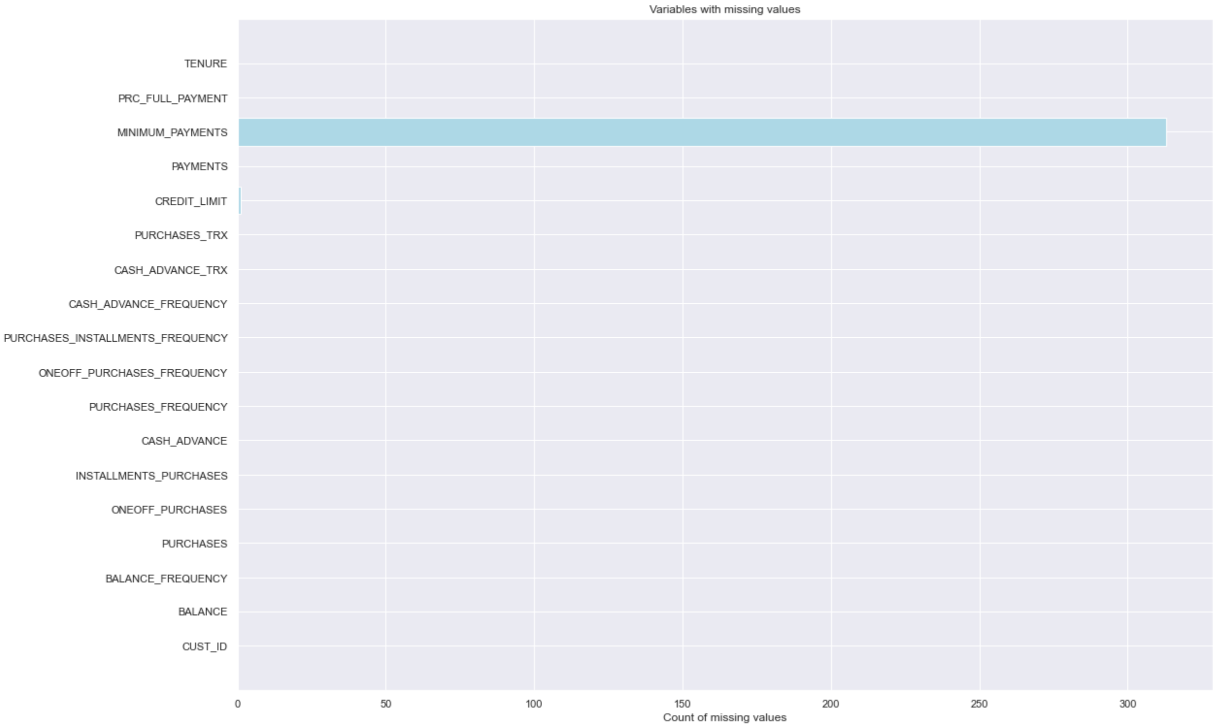
Out[229…

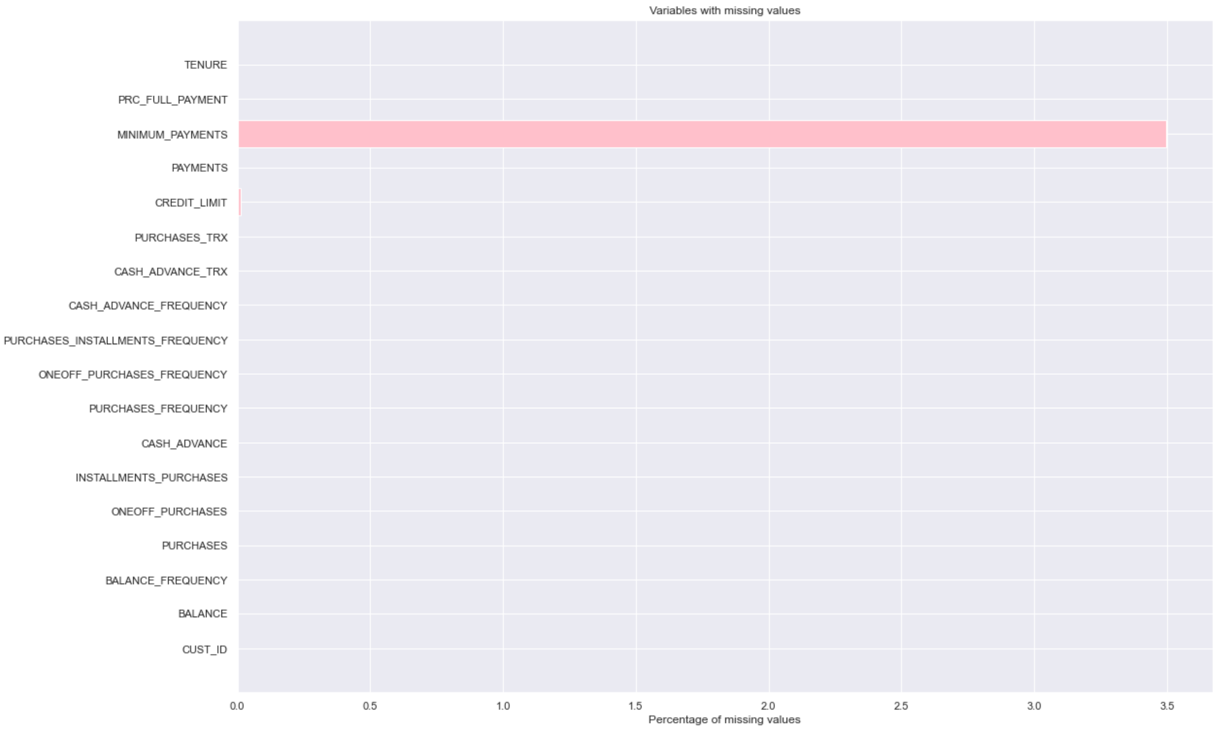
	Features	Count	Percentange
15	MINIMUM_PAYMENTS	313	3.497207
13	CREDIT_LIMIT	1	0.011173
0	CUST_ID	0	0.000000
1	BALANCE	0	0.000000
16	PRC_FULL_PAYMENT	0	0.000000
14	PAYMENTS	0	0.000000
12	PURCHASES_TRX	0	0.000000
11	CASH_ADVANCE_TRX	0	0.000000
10	CASH_ADVANCE_FREQUENCY	0	0.000000
9	PURCHASES_INSTALLMENTS_FREQUENCY	0	0.000000
8	ONEOFF_PURCHASES_FREQUENCY	0	0.000000
7	PURCHASES_FREQUENCY	0	0.000000
6	CASH_ADVANCE	0	0.000000
5	INSTALLMENTS_PURCHASES	0	0.000000
4	ONEOFF_PURCHASES	0	0.000000
3	PURCHASES	0	0.000000
2	BALANCE_FREQUENCY	0	0.000000
17	TENURE	0	0.000000

In [225…

```python
# Plot missing values

# Get list of features
columns = missing_values.Features.values.tolist()

# Get index's
ind = missing_values.index.to_list()

# Create subplots
fig, (ax1, ax2) = plt.subplots(2,1,figsize=(18, 28))

# Plot missing values based on count
rects = ax1.barh(ind, missing_values.Count.values.tolist(), color='lightblue')
ax1.set_yticks(ind)
ax1.set_yticklabels(columns, rotation='horizontal')
ax1.set_xlabel("Count of missing values")
ax1.set_title("Variables with missing values")

# Plot missing values based on percentage
rects = ax2.barh(ind, missing_values.Percentange.values.tolist(), color='pink')
ax2.set_yticks(ind)
ax2.set_yticklabels(columns, rotation='horizontal')
ax2.set_xlabel("Percentage of missing values")
ax2.set_title("Variables with missing values")
```

Out[225… Text(0.5, 1.0, 'Variables with missing values')





In [18]:

```
# We will drop Cust_id feature
data = raw_data.drop('CUST_ID', axis = 1)
```

1. Handling Missing values of Credit_limit feature

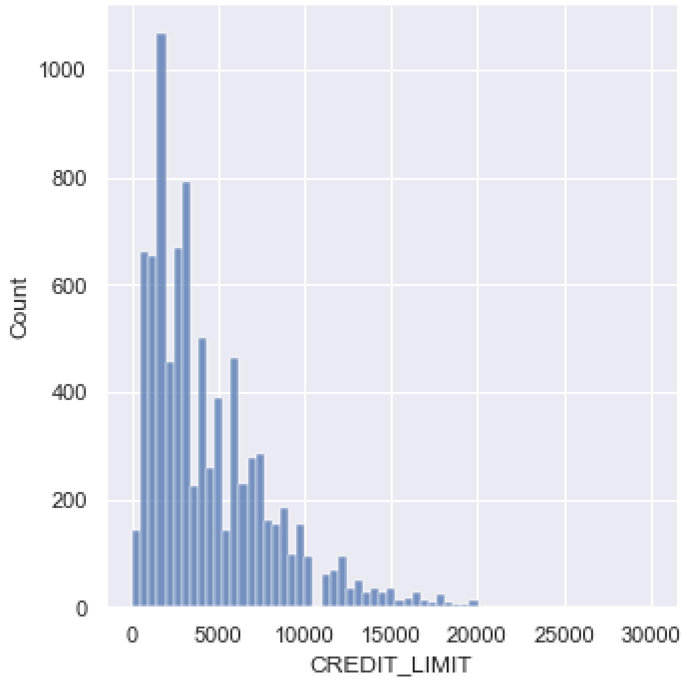
Since credit_limit has only one missing value so we Drop that row directly since it only has 1 missing value

In [19]:

```
sns.displot(x = 'CREDIT_LIMIT', data = raw_data.dropna())
```

<seaborn.axisgrid.FacetGrid at 0x24d5e5153a0>

Out[19]:



As credit_limit is skewed towards right (positive skewness) we will replace null value with median

In [20]:

```
# Replace with median
data['CREDIT_LIMIT'] = data.CREDIT_LIMIT.fillna(data['CREDIT_LIMIT'].median())
```

2. Handling Missing values of MINIMUM_PAYMENTS feature

Missing values are imputed for *KNN Imputer*

In [21]:

```
from sklearn.impute import KNNImputer
```

In [22]:

```
# Initilaize the imputer
imputer = KNNImputer(n_neighbors=2)

# fit and transform the data
no_missing = pd.DataFrame(imputer.fit_transform(data.iloc[:, :]), columns = data.col

# Get the shape
no_missing.shape
```

Out[22]: (8950, 17)

In [23]:

```
no_missing.isnull().sum()
```

Out[23]:

```
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
```

```
PURCHASES_TRX                    0
CREDIT_LIMIT                     0
PAYMENTS                         0
MINIMUM_PAYMENTS                 0
PRC_FULL_PAYMENT                 0
TENURE                           0
dtype: int64
```

Note: There are no missing values

Check for duplication

In [24]:

```
# Let's see if we have duplicated entries in the data
no_missing.duplicated().sum()
```

Out[24]:

```
0
```

Note: There are no duplicate values in the dataset

In [25]:

```
print("Credit Card Data Set has \033[4m\033[1m{}\033[0m\033[0m data points with \033
```

```
Credit Card Data Set has 8950 data points with 17 variables each.
```

4.2. Exploratory Data Analysis

The preliminary analysis of data to discover relationships between measures in the data and to gain an insight on the trends, patterns, and relationships among various entities present in the data set with the help of statistics and visualization tools is called Exploratory Data Analysis (EDA).

Exploratory data analysis is cross-classified in two different ways where each method is either graphical or non-graphical. And then, each method is either univariate, bivariate or multivariate.

...goto toc

4.2.1. Data Visualization

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps.

...goto toc

In [26]:

```
no_missing.describe()
```

Out[26]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PUR
count	8950.000000	8950.000000	8950.000000	8950.000000	895
mean	1564.474828	0.877271	1003.204834	592.437371	41
std	2081.531879	0.236904	2136.634782	1659.887917	90
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	8

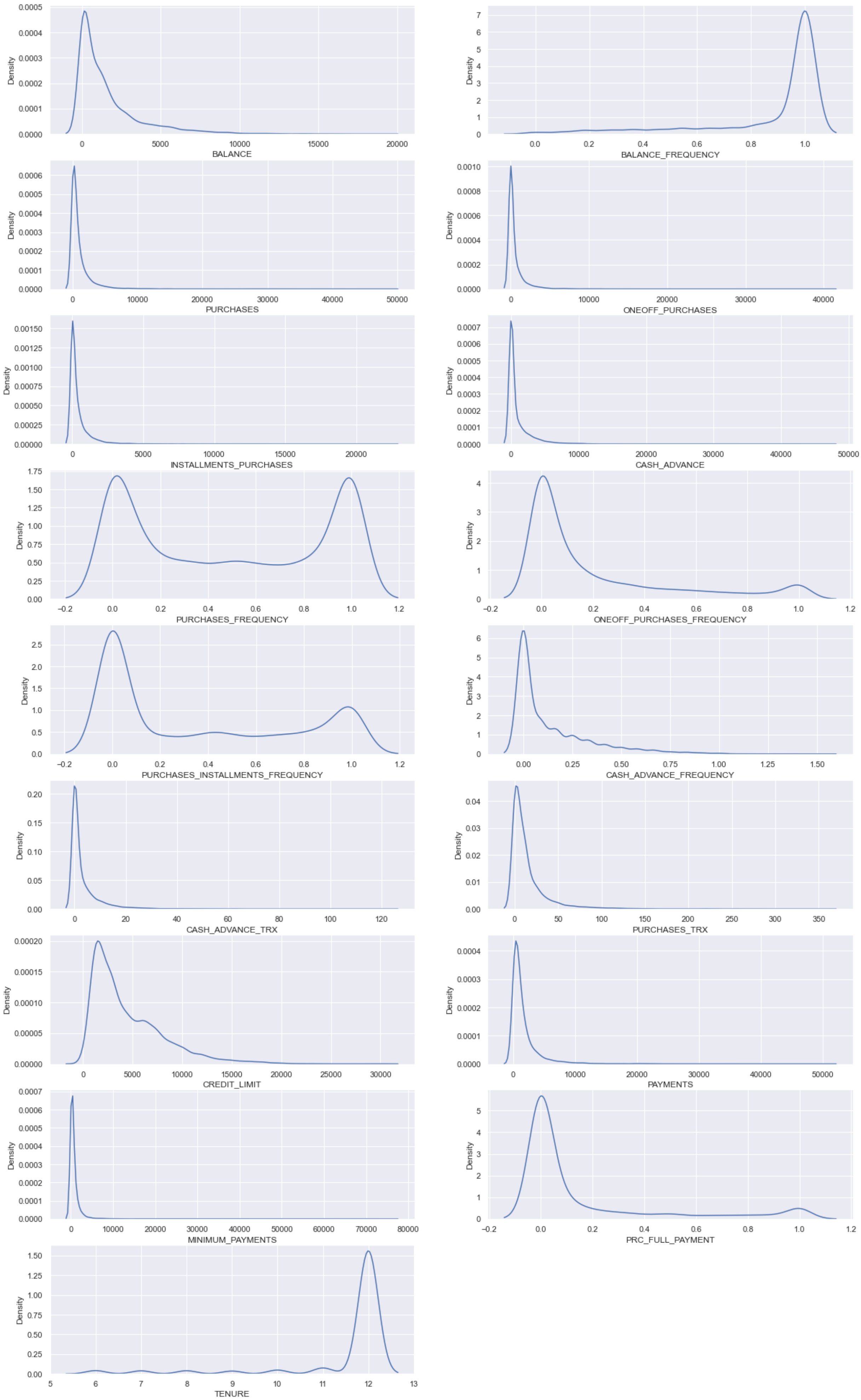
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PUF
75%	2054.140036	1.000000	1110.130000	577.405000	46
max	19043.138560	1.000000	49039.570000	40761.250000	2250

Kernel Density Plot

To understand data distribution

In [27]:

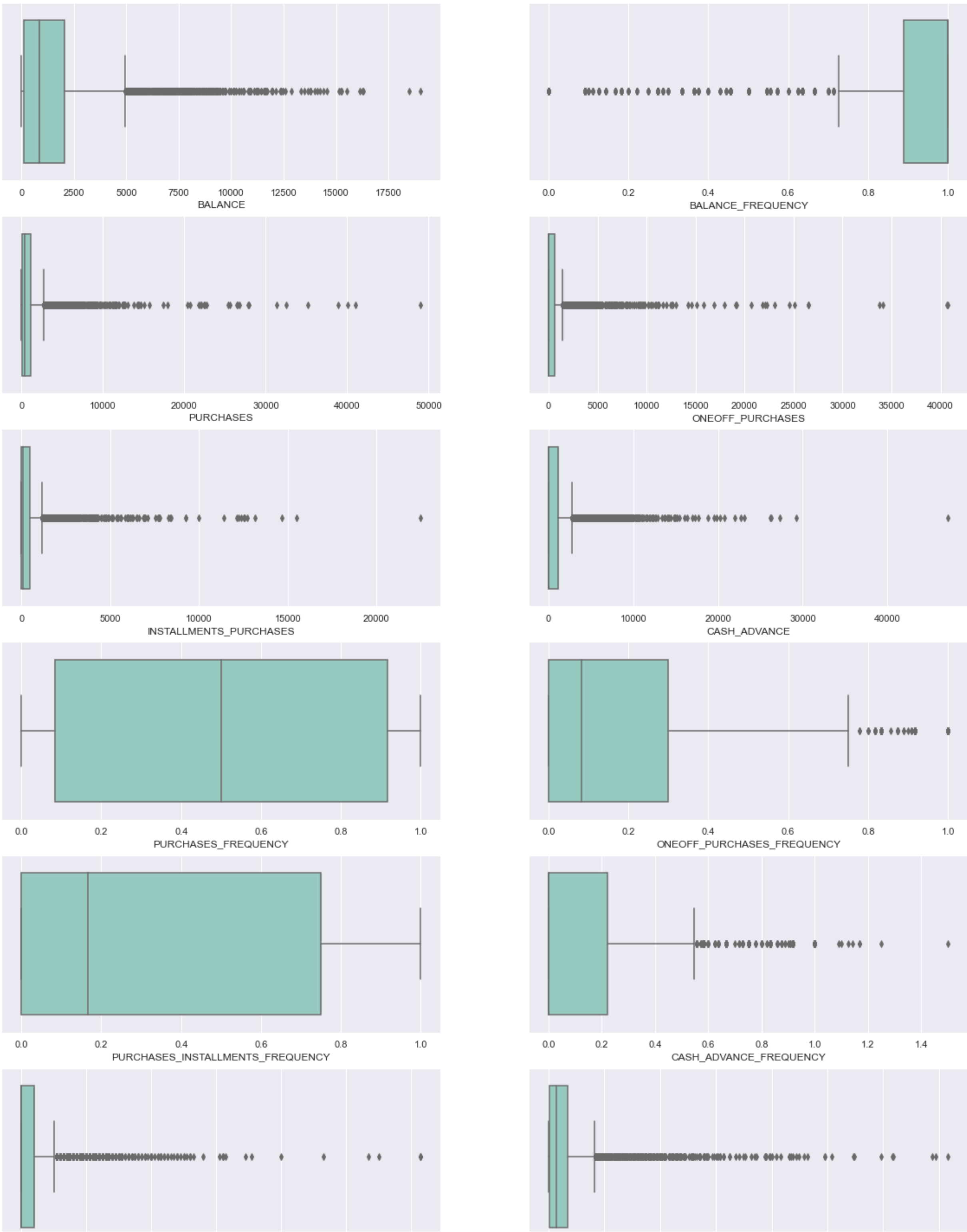
```
# PLot KDE for all features

# Set size of the figure
plt.figure(figsize=(20,35))

# Iterate on list of features
for i, col in enumerate(no_missing.columns):
    if no_missing[col].dtype != 'object':
        ax = plt.subplot(9, 2, i+1)
        kde = sns.kdeplot(no_missing[col], ax=ax)
        plt.xlabel(col)

plt.show()
```

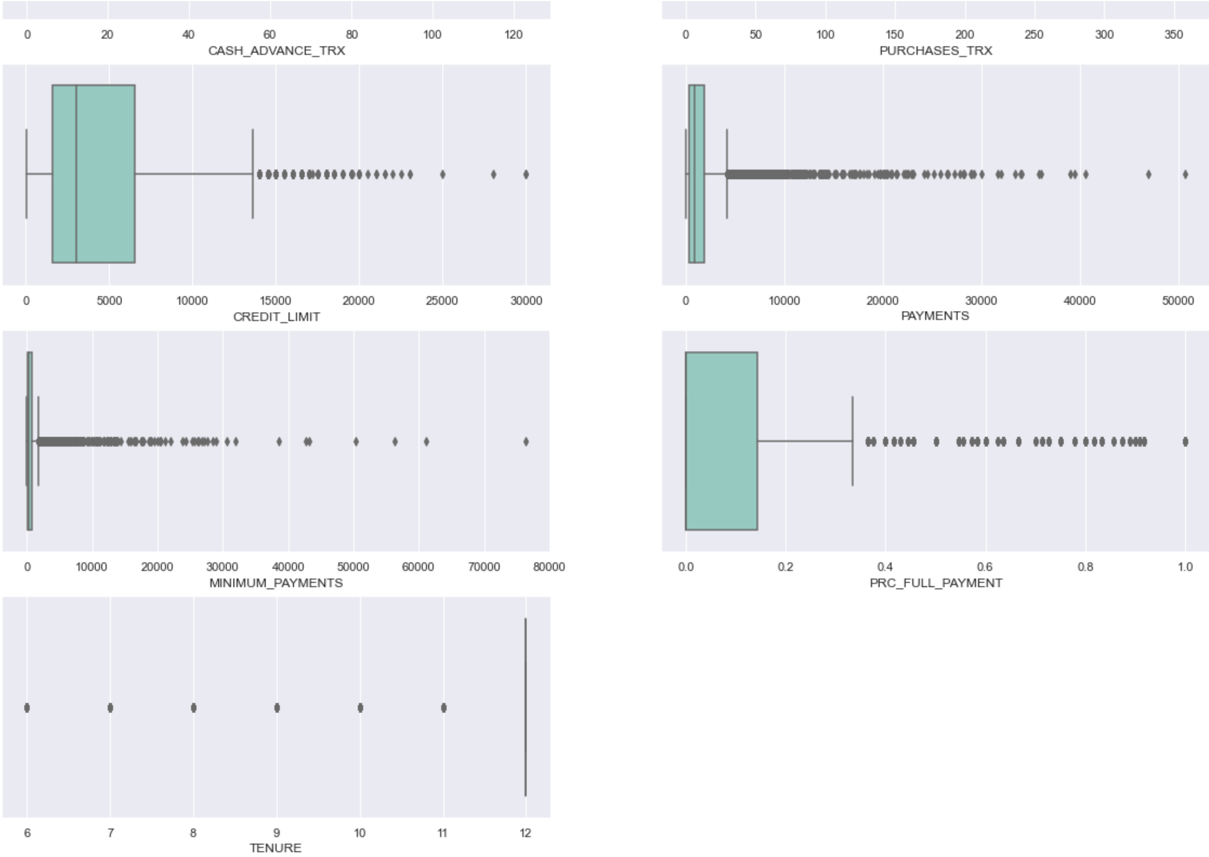

Density
BALANCE
BALANCE_FREQUENCY
PURCHASES
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS
PRC_FULL_PAYMENT
TENURE

Boxplot

To detect outliears

In [28]:
```
plt.figure(figsize = (20,40))

counter = 0

for i, col in enumerate(no_missing.columns):
    if no_missing[col].dtype == 'object':
        continue

    ax = plt.subplot(9, 2, i+1)
    sns.boxplot(x = col, data = no_missing, ax = ax, palette = "Set3")
    plt.xlabel(col)

plt.show()
```





Note: There are many ouliears in the data, by dropping them can result in loss of data adequate. So we will perform binning to handle them.

Additionaly features are either in the scale of thousands, units or decimals. So we will make intervals accordingly.

In [29]:

```
# create the copy of dataframe
data_1 = no_missing.copy()
```

In [30]:

```
# Get features having scale as thousands
columns = ['BALANCE', 'PURCHASES', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES',
            'CASH_ADVANCE', 'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS']

# Iterate through each column
for col in tqdm(columns):
    interval = col + "_interval"

    # 0
    data_1[interval] = 0

    # 1
    data_1.loc[((data_1[col] > 0) & (data_1[col] <= 500)), interval] = 1

    # 2
    data_1.loc[((data_1[col] > 500) & (data_1[col] <= 1000)), interval] = 2

    # 3
    data_1.loc[((data_1[col] > 1000) & (data_1[col] <= 3000)), interval] = 3

    # 4
    data_1.loc[((data_1[col] > 30000) & (data_1[col] <= 5000)), interval] = 4

    # 5
    data_1.loc[((data_1[col] > 5000) & (data_1[col] <= 10000)), interval] = 5
```

```
    # 6
    data_1.loc[(data_1[col] > 10000), interval] = 6

# drop the features
data_1.drop(columns, axis = 1, inplace = True)
```

100%|██████████| 8/8 [00:00<00:00, 135.69it/s]

In [31]:

```
# Get features having scale as ten's
columns=['PURCHASES_TRX', 'CASH_ADVANCE_TRX']

# Iterate through each column
for col in tqdm(columns):
    interval = col + "_interval"

    # 0
    data_1[interval] = 0

    # 1
    data_1.loc[((data_1[col] > 0) & (data_1[col] <= 5)), interval] = 1

    # 2
    data_1.loc[((data_1[col] > 5) & (data_1[col] <= 10)), interval] = 2

    # 3
    data_1.loc[((data_1[col] > 10) & (data_1[col] <= 15)), interval] = 3

    # 4
    data_1.loc[((data_1[col] > 15) & (data_1[col] <= 20)), interval] = 4

    # 5
    data_1.loc[((data_1[col] > 20) & (data_1[col] <= 30)), interval] = 5

    # 6
    data_1.loc[((data_1[col] > 50) & (data_1[col] <= 50)), interval] = 6

    # 7
    data_1.loc[((data_1[col] > 50) & (data_1[col] <= 100)), interval] = 7

    # 8
    data_1.loc[(data_1[col] > 100), interval] = 8

# drop the features
data_1.drop(columns, axis = 1, inplace = True)
```

100%|██████████| 2/2 [00:00<00:00, 76.98it/s]

In [32]:

```
# Get features having scale as decimal
columns=['BALANCE_FREQUENCY', 'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY', '
         'CASH_ADVANCE_FREQUENCY', 'PRC_FULL_PAYMENT']


# Iterate through each column
for col in tqdm(columns):
    interval = col + "_interval"

    # 0
    data_1[interval] = 0

    # 1
    data_1.loc[((data_1[col] > 0) & (data_1[col] <= 0.1)), interval] = 1
```

```
    # 2
    data_1.loc[((data_1[col] > 0.1) & (data_1[col] <= 0.2)), interval] = 2

    # 3
    data_1.loc[((data_1[col] > 0.2) & (data_1[col] <= 0.3)), interval] = 3

    # 4
    data_1.loc[((data_1[col] > 0.3) & (data_1[col] <= 0.4)), interval] = 4

    # 5
    data_1.loc[((data_1[col] > 0.4) & (data_1[col] <= 0.5)), interval] = 5

    # 6
    data_1.loc[((data_1[col] > 0.5) & (data_1[col] <= 0.6)), interval] = 6

    # 7
    data_1.loc[((data_1[col] > 0.6) & (data_1[col] <= 0.7)), interval] = 7

    # 8
    data_1.loc[((data_1[col] > 0.7) & (data_1[col] <= 0.8)), interval] = 8

    # 9
    data_1.loc[((data_1[col] > 0.8) & (data_1[col] <= 0.9)), interval] = 9

    # 10
    data_1.loc[((data_1[col] > 0.9) & (data_1[col] <= 1.0)), interval] = 10

# drop the features
data_1.drop(columns, axis = 1, inplace = True)
```

```
100%|██████████| 6/6 [00:00<00:00, 103.43it/s]
```

Correlation Analysis

In [33]:

```
# check correlation
corr = data_1.corr()
corr
```

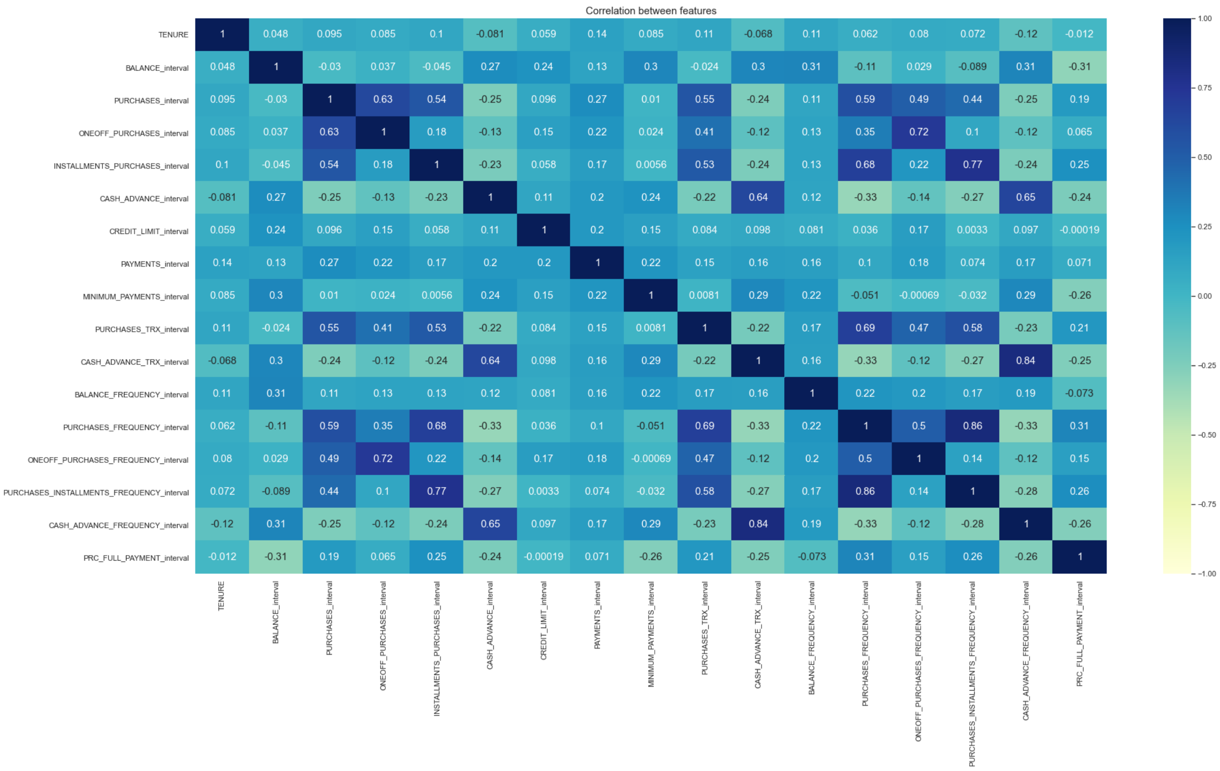
Out[33]:

	TENURE	BALANCE_interval	PURCHASES_interval
TENURE	1.000000	0.047742	0.094880
BALANCE_interval	0.047742	1.000000	-0.029776
PURCHASES_interval	0.094880	-0.029776	1.000000
ONEOFF_PURCHASES_interval	0.084740	0.036918	0.631275
INSTALLMENTS_PURCHASES_interval	0.103304	-0.044713	0.544998
CASH_ADVANCE_interval	-0.081488	0.267915	-0.248305
CREDIT_LIMIT_interval	0.059040	0.242117	0.096134
PAYMENTS_interval	0.144671	0.128015	0.268028
MINIMUM_PAYMENTS_interval	0.084878	0.301901	0.010438
PURCHASES_TRX_interval	0.112877	-0.024461	0.548951
CASH_ADVANCE_TRX_interval	-0.067906	0.302659	-0.243847
BALANCE_FREQUENCY_interval	0.112589	0.305288	0.112412
PURCHASES_FREQUENCY_interval	0.061732	-0.110793	0.593294

	TENURE	BALANCE_interval	PURCHASES_interval
ONEOFF_PURCHASES_FREQUENCY_interval	0.079818	0.028978	0.493775
PURCHASES_INSTALLMENTS_FREQUENCY_interval	0.071829	-0.088966	0.440815
CASH_ADVANCE_FREQUENCY_interval	-0.120745	0.309638	-0.248460
PRC_FULL_PAYMENT_interval	-0.012288	-0.305527	0.194169

In [34]:

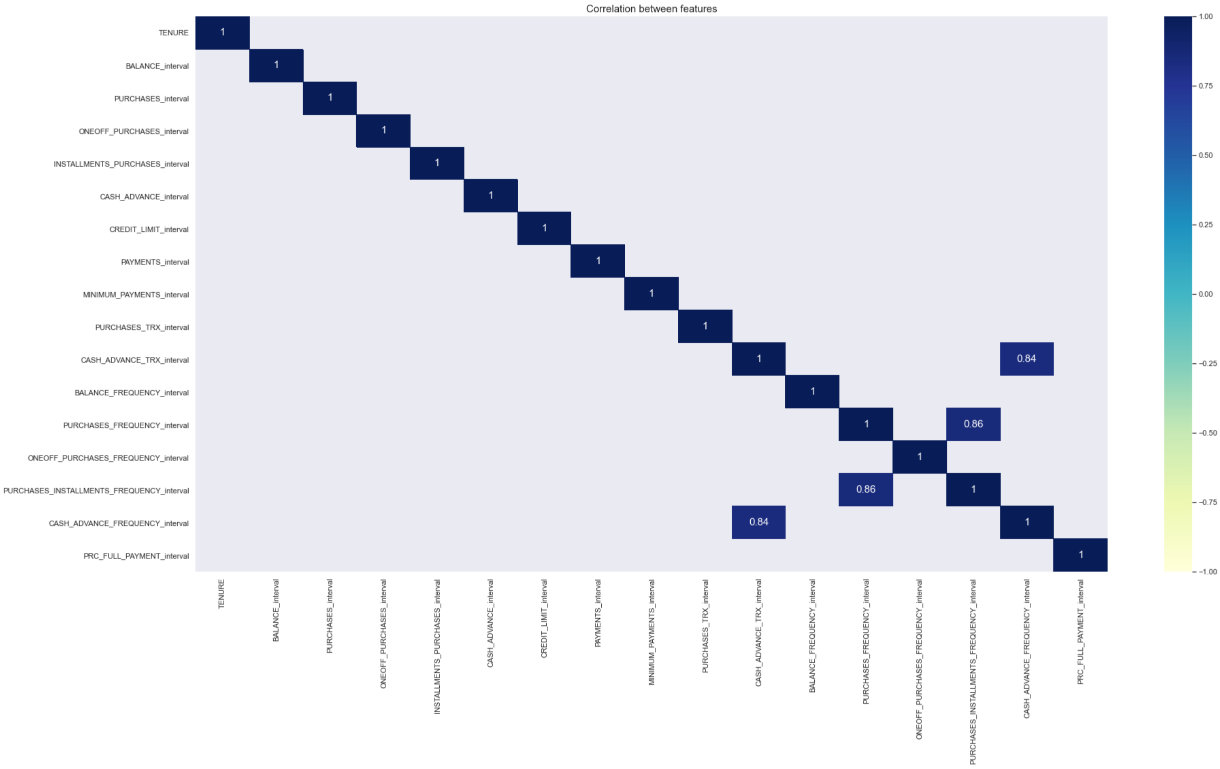
```
# ploting correlation plot

# set the figure size
plt.figure(figsize=(30, 15))

# plotting the heat map
sns.heatmap(corr,
            cmap='YlGnBu', vmax=1.0, vmin=-1.0,
            annot=True, annot_kws={"size": 15})

# set the title
# fontsize=30: set the font size of the title
plt.title('Correlation between features', fontsize=15)
# display the plot
plt.show()
```



In [40]:

```
# ploting correlation plot

# set the figure size
plt.figure(figsize=(30, 15))

# plotting the heat map
sns.heatmap(corr[(corr >= 0.8) | (corr <= -0.8)],
            cmap='YlGnBu', vmax=1.0, vmin=-1.0,
            annot=True, annot_kws={"size": 15})

# set the title
```

```python
plt.title('Correlation between features', fontsize=15)

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



... Goto TOC

Analysis Report

Type	Number of Instances	Number of Attributes	Numeric Features	Categorical Features	Missing Values
Unsupervised Learning	8950	17	17	0	Null

Data Types

Sr.No.	Column	Data type
1	TENURE	float64
2	BALANCE_interval	int64
3	PURCHASES_interval	int64
4	ONEOFF_PURCHASES_interval	int64
5	INSTALLMENTS_PURCHASES_interval	int64
6	CASH_ADVANCE_interval	int64
7	CREDIT_LIMIT_interval	int64

Sr.No.	Column	Data type
8	PAYMENTS_interval	int64
9	MINIMUM_PAYMENTS_interval	int64
10	PURCHASES_TRX_interval	int64
11	CASH_ADVANCE_TRX_interval	int64
12	BALANCE_FREQUENCY_interval	int64
13	PURCHASES_FREQUENCY_interval	int64
14	ONEOFF_PURCHASES_FREQUENCY_interval	int64
15	PURCHASES_INSTALLMENTS_FREQUENCY_interval	int64
16	CASH_ADVANCE_FREQUENCY_interval	int64
17	PRC_FULL_PAYMENT_interval	int64

Exploratory Data Analysis

- Mean of *balance* is 1564
- *Balance_Frequency* for most customers is updated frequently i.e *1*
- For *PURCHASES_FREQUENCY*, there are *two* distinct group of customers
- For *ONEOFF_PURCHASES_FREQUENCY* and *PURCHASES_INSTALLMENT_FREQUENCY* most users don't do one off puchases or installment purchases frequently
- Very small number of customers pay their balance in full *PRC_FULL_PAYMENT* i.e 0
- *Credit limit* average is around *4494.28*
- Most customers have *12* years *tenure*

Additionally,

- High Correlation between *PURCHASES_FREQUENCY* & *PURCHASES_INSTALLMENT_FREQUENCY* (0.86)

- When people use *one-off purchases*, purchase amount is higher than using installment purchases.

- More people use installment purchases (*CASHADVANCEFREQUENCY* & *CASHADVANCETRX*: 0.84)

4.3. Feature Scaling

Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization

...goto toc

In [45]:

```
# Import the required function
from sklearn.preprocessing import StandardScaler
```

In [46]:

```
# Initilize scaler
```

```
scaler = StandardScaler()

# fit the scaler
scaler.fit(data_1)
```

Out[46]: StandardScaler()

In [47]:

```
# Transform the dataset
X = scaler.transform(data_1)
```

5. Model Development

...goto toc

Since in our project we are focusing on understanding different customer groups so as to build marketing or other business strategies i.e **Customer Segementation**, it falls under **Unsupervised Machine Learning** use case.

For our project we will focus on implementing it via **KMeans**.

There are several methods to determine the optimal value of K in K-Means Clustering. But in our case we will be using

- **Elbow Method** - It consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use.
- **Silhouette Score** - It is a metric used to calculate the goodness of a clustering technique.
  - Its value ranges from *-1* to *1*
  - *1* means clusters are well apart from each other and clearly distinguished
- **Calinski Harabasz Score** - The Calinski-Harabasz index also known as the Variance Ratio Criterion, is the ratio of the sum of between-clusters dispersion and of inter-cluster dispersion for all clusters, the higher the score , the better the performances.
- **Davies Bouldin Score** - The score is defined as the average similarity measure of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances.

In [101…

```
# Import required packages
from sklearn.metrics import silhouette_score, calinski_harabasz_score, davies_bouldi
from sklearn.cluster import KMeans
```

In [98]:

```
# Function to calculate metrics
def compute_metrics(data, min_cluster = 2, max_cluster = 10, rand_state = 1):
    elbow_cost, sil_scores, ch_scores, db_scores = [], [], [], []

    for cluster in tqdm(range(min_cluster, max_cluster)):
        # Initialize KMeans with number of clusters
        kmeans = KMeans(n_clusters = cluster, random_state = rand_state)

        # Fit on data
        kmeans.fit(data)

        # Get labels assigned for the dataset
```

```python
        labels = kmeans.labels_

        # Calculate Inertia for Elbow Method
        elbow_cost.append(kmeans.inertia_)

        # Calculate Silhouette Score
        sil_scores.append(silhouette_score(data, labels))

        # Calculate Calinski Harabasz Score
        ch_scores.append(calinski_harabasz_score(data, labels))

        # Calculate Davies Bouldin Score
        db_scores.append(davies_bouldin_score(data, labels))

    return elbow_cost, sil_scores, ch_scores, db_scores
```

In [99]:

```python
# Function to plot metric scores to find optimal value of 'K'
def plot_metrics(elbow_cost, sil_scores, ch_scores, db_scores, min_cluster = 2, max_

    fig, axes = plt.subplots(2,2, figsize = (20,15))

    x_axis = list(range(min_cluster, max_cluster))

    # Plot Inertia for Elbow Method
    sns.lineplot(x = x_axis, y = elbow_cost, ax = axes[0,0])
    axes[0,0].set_title('Elbow Method', fontsize = 15)
    axes[0,0].set(xlabel = "Number of Clusters", ylabel = "Inertia")

    # Plot Silhouette Score
    sns.lineplot(x = x_axis, y = sil_scores, ax = axes[0,1])
    axes[0,1].set_title('Silhouette Method', fontsize = 15)
    axes[0,1].set(xlabel = "Number of Clusters", ylabel = "Silhouette Score")

    # Plot Calinski Harabasz Score
    sns.lineplot(x = x_axis, y = ch_scores, ax = axes[1,0])
    axes[1,0].set_title('Calinski Harabasz Method', fontsize = 15)
    axes[1,0].set(xlabel = "Number of Clusters", ylabel = "Calinski Harabasz Score")

    # Plot Davies Bouldin Score
    sns.lineplot(x = x_axis, y = db_scores, ax = axes[1,1])
    axes[1,1].set_title('Davies Bouldin Method', fontsize = 15)
    axes[1,1].set(xlabel = "Number of Clusters", ylabel = "Davies Bouldin Score")
```

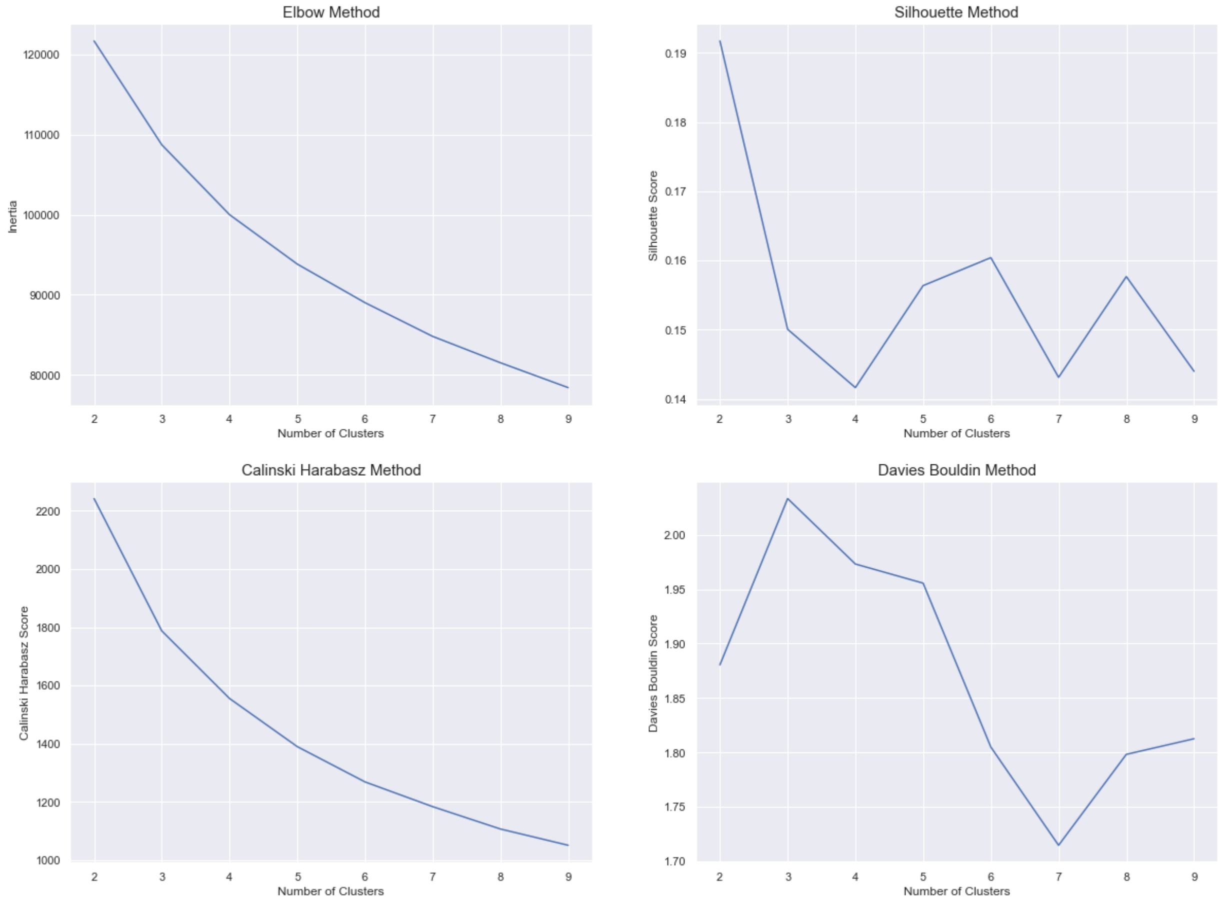
In [59]:

```python
# Computer metric scores
elbow_cost, sil_scores, ch_scores, db_scores = compute_metrics(X, min_cluster = 2, m
```

```
100%|██████████| 8/8 [00:14<00:00,  1.86s/it]
```

In [100…

```python
# Plot metrics
plot_metrics(elbow_cost, sil_scores, ch_scores, db_scores)
```



From analyzing different metrics we considered optimal value of **K** as **6**

In [106…

```
# Save optimal number of cluster
optimal_cluster = 6
```

In [108…

```
# Initialize KMeans with number of clusters
kmeans = KMeans(n_clusters = optimal_cluster)

# Fit on data
kmeans.fit(X)

# Get labels assigned for the dataset
labels = kmeans.labels_
```

In [110…

```
# Create copy of the dataframe
clusters = data_1.copy(deep = True)

# Assign clusters to customers
clusters['Cluster'] = labels

# Print cluster dataframe
clusters.head()
```

Out[110…

	TENURE	BALANCE_interval	PURCHASES_interval	ONEOFF_PURCHASES_interval	INSTALLMENTS_PU
0	12.0	1	1	0	
1	12.0	0	0	0	
2	12.0	3	2	2	
3	12.0	3	3	3	

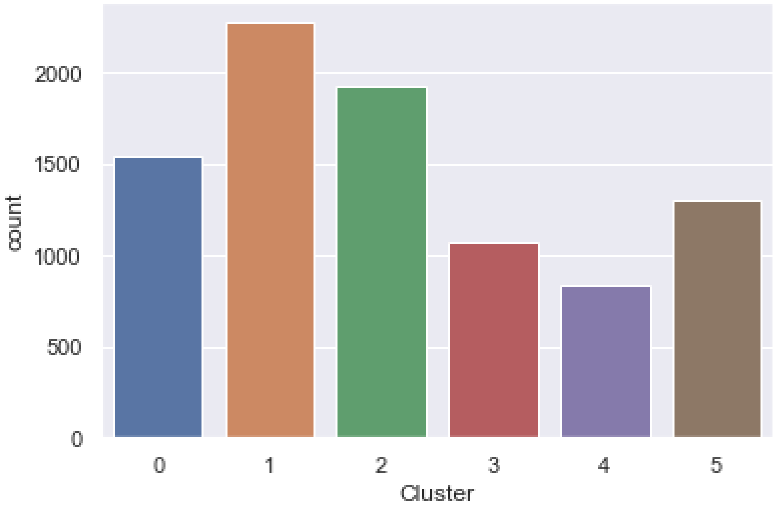
	TENURE	BALANCE_interval	PURCHASES_interval	ONEOFF_PURCHASES_interval	INSTALLMENTS_PI
4	12.0	2	1	1	

In [215…

```
# Get distribution of clusters
sns.countplot(x = "Cluster", data = clusters)
```

Out[215…

```
<AxesSubplot:xlabel='Cluster', ylabel='count'>
```

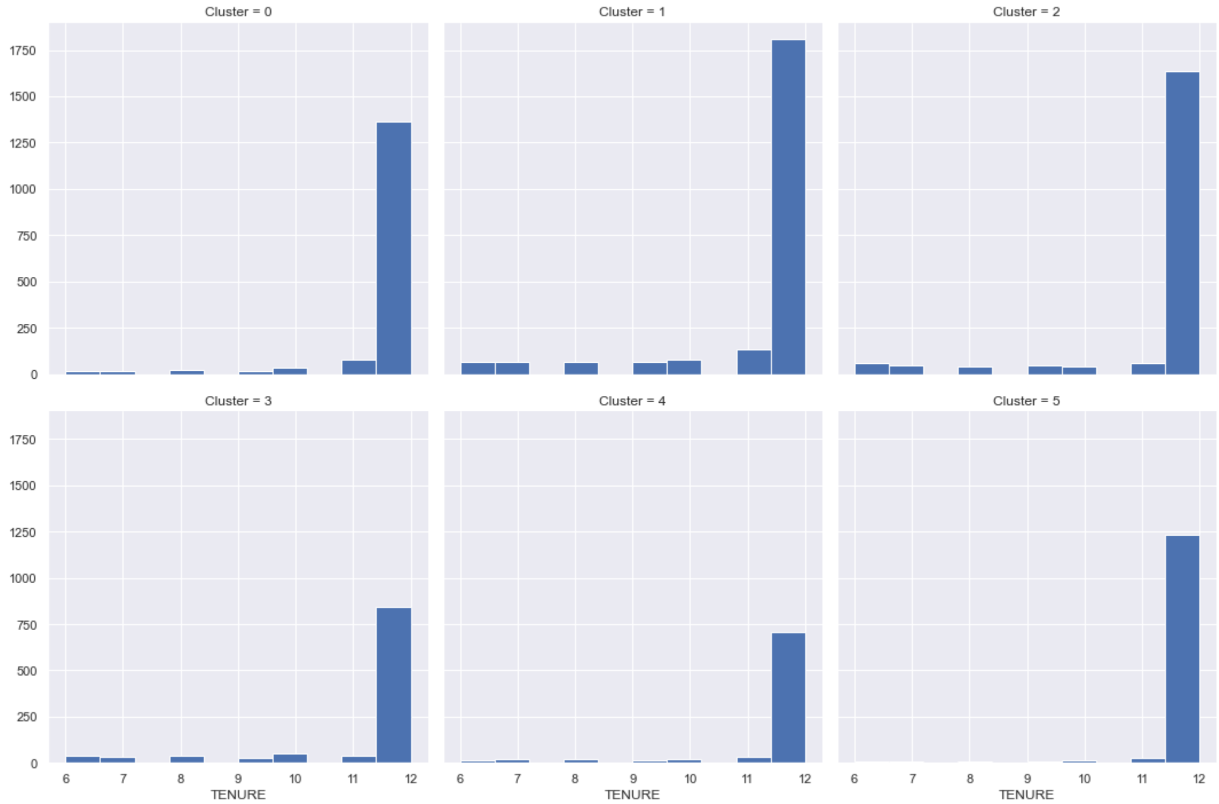


Visualize the featrues with respect to clusters

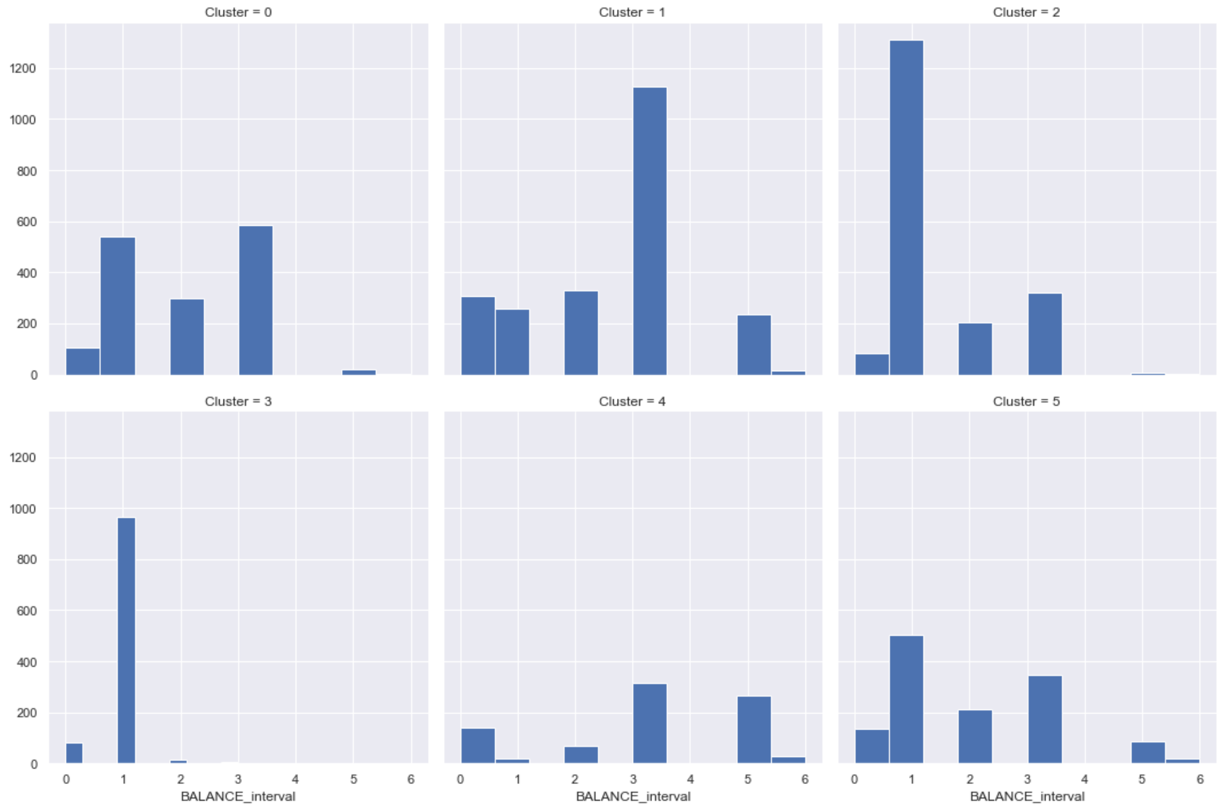
In [185…

```
# Iterate over each feature
for col in clusters:
    if col == "Cluster":
        continue

    print("-"*150)
    print(f"Feature : \033[4m\033[1m{col}\033[0m\033[0m")
    print("-"*150)

    # Plot histogram of a feature with respect to clusters
    grid = sns.FacetGrid(clusters, col='Cluster', col_wrap = 3, aspect = 1, height =
    grid.map(plt.hist, col)
    plt.show()
```

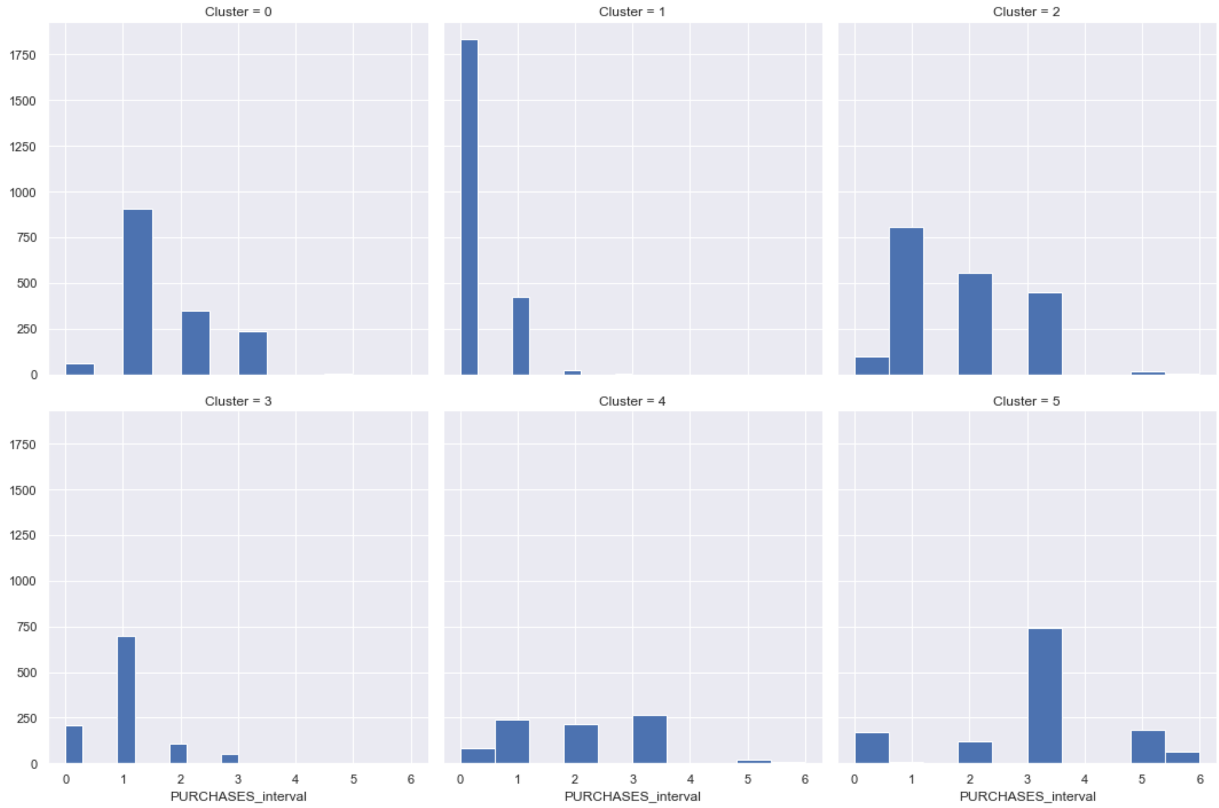
```
------------------------------------------------------------------------------------------------------------------------------------------------------
Feature : TENURE
------------------------------------------------------------------------------------------------------------------------------------------------------
```



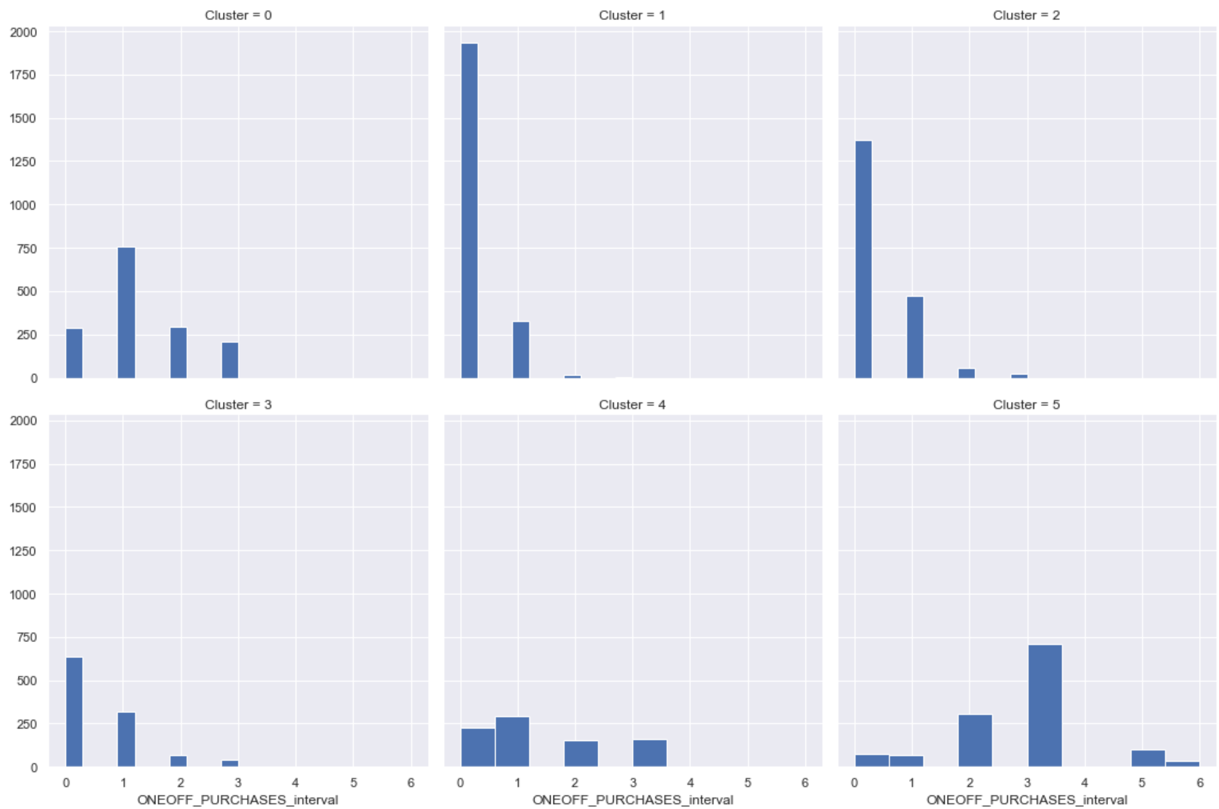
Feature : **BALANCE_interval**



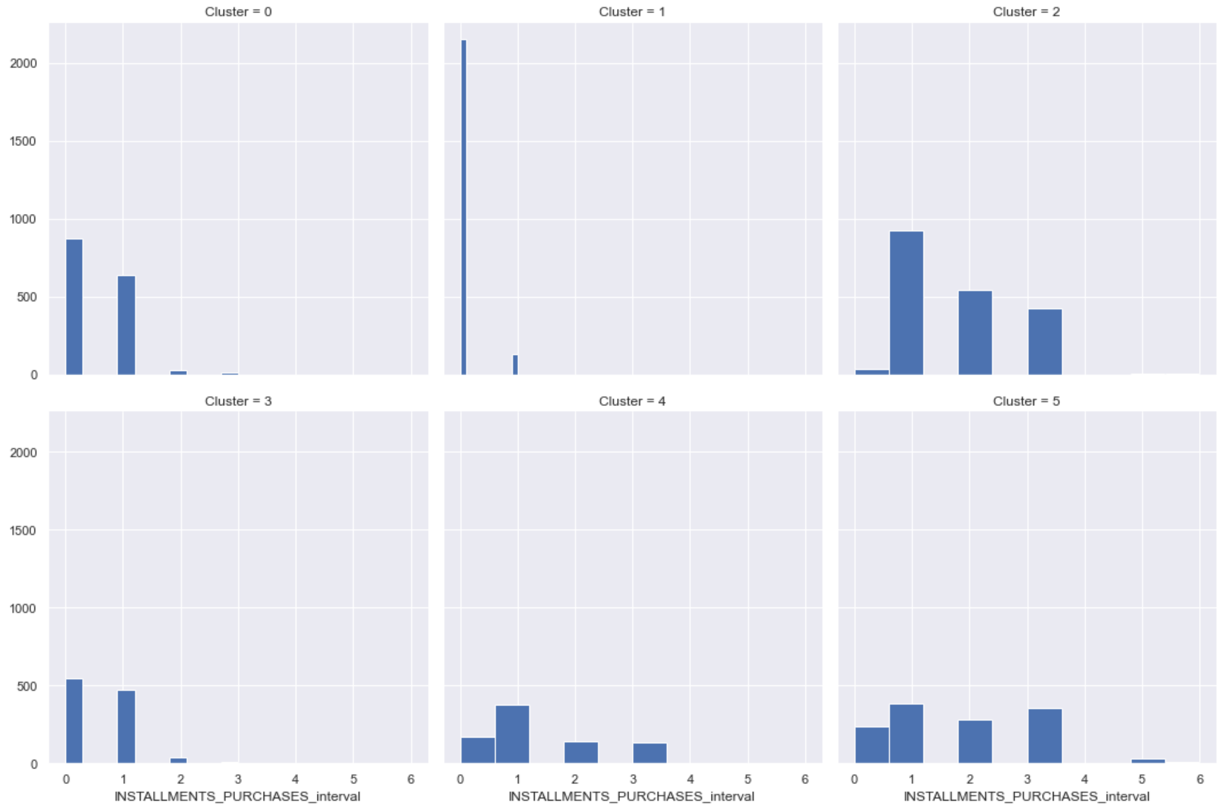
Feature : **PURCHASES_interval**



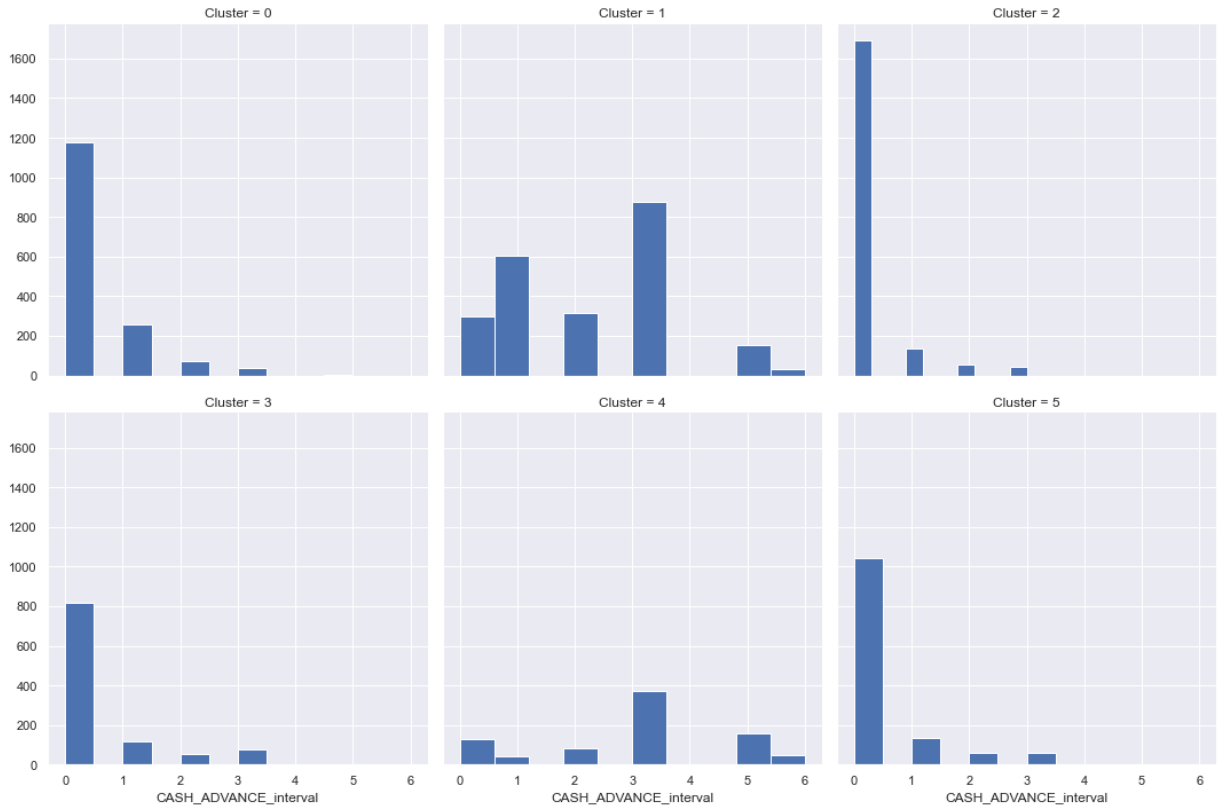
Feature : **ONEOFF_PURCHASES_interval**



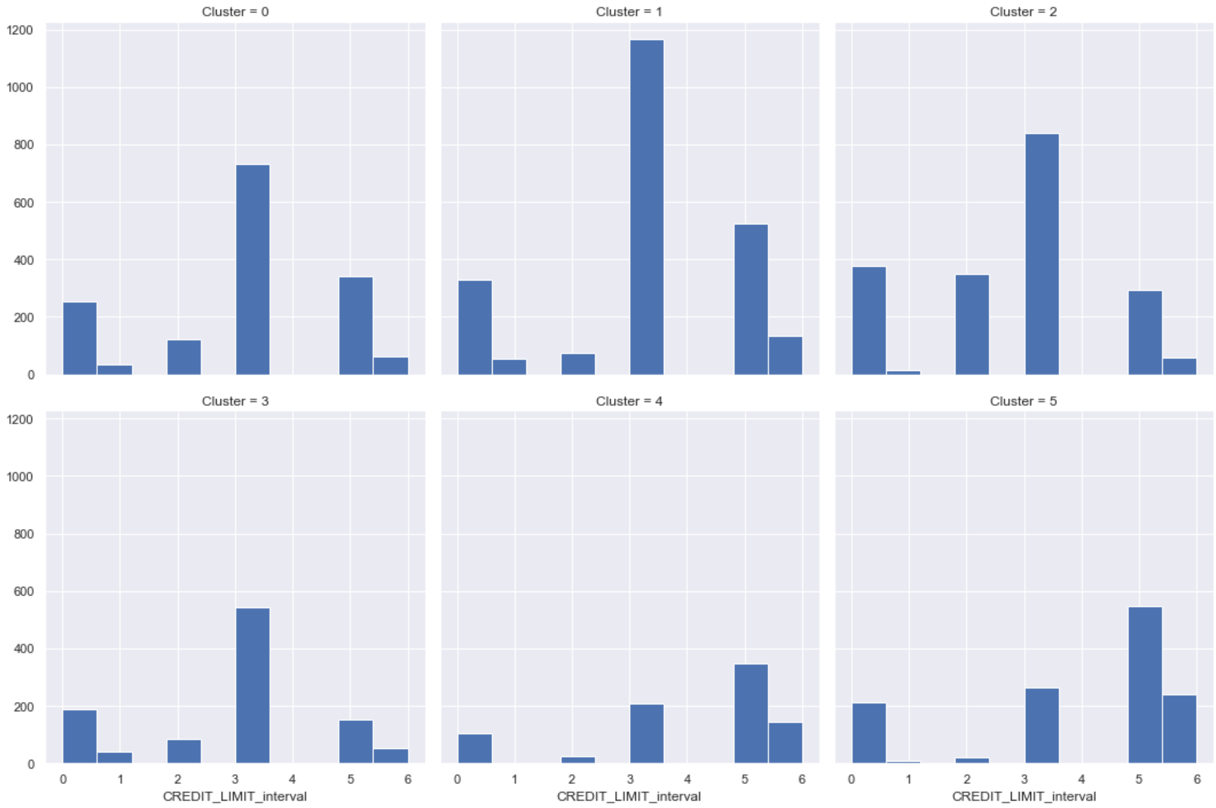
Feature : **INSTALLMENTS_PURCHASES_interval**



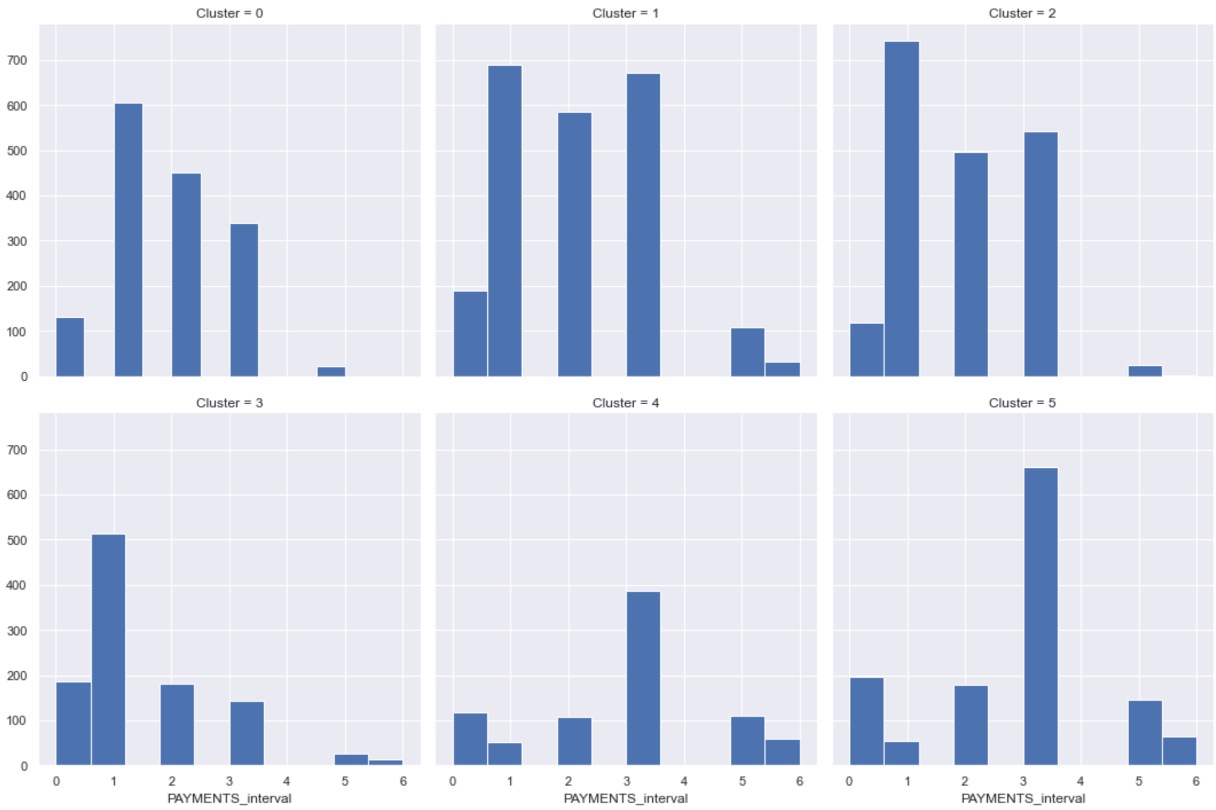
Feature : **CASH_ADVANCE_interval**



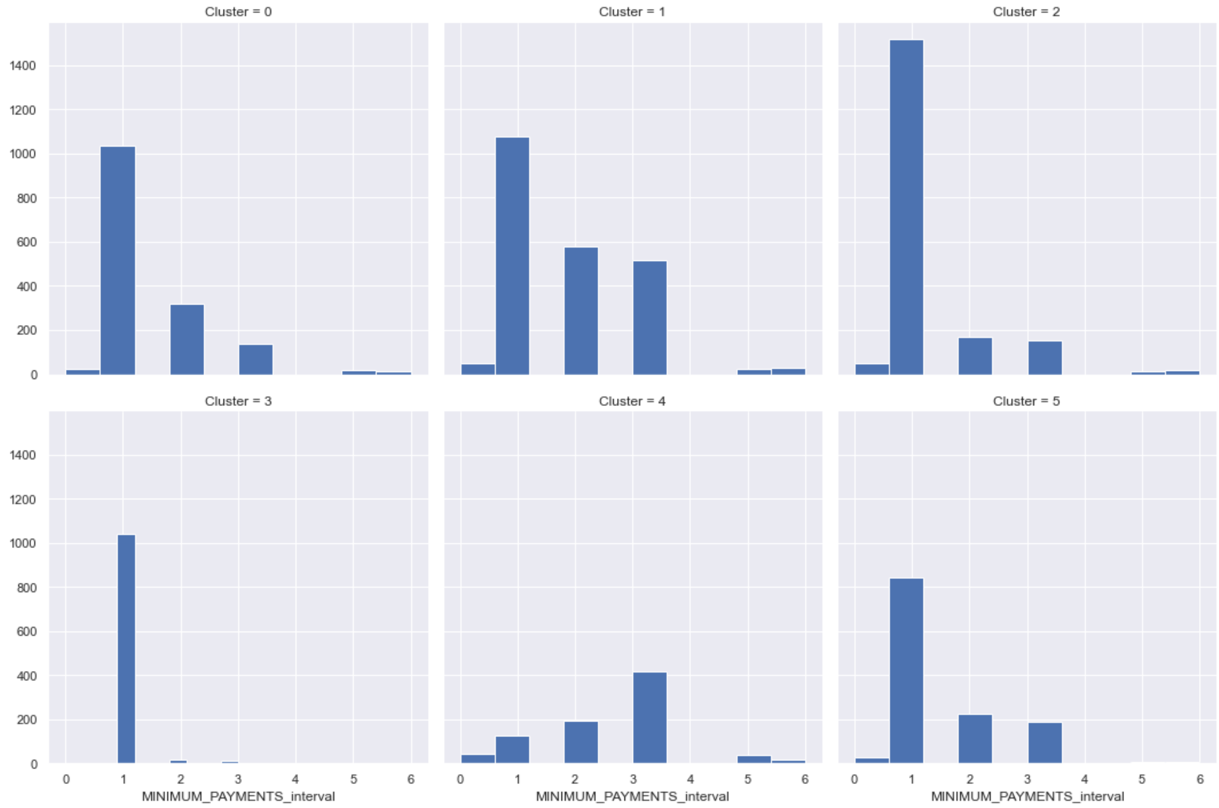
Feature : **CREDIT_LIMIT_interval**



Feature : **PAYMENTS_interval**



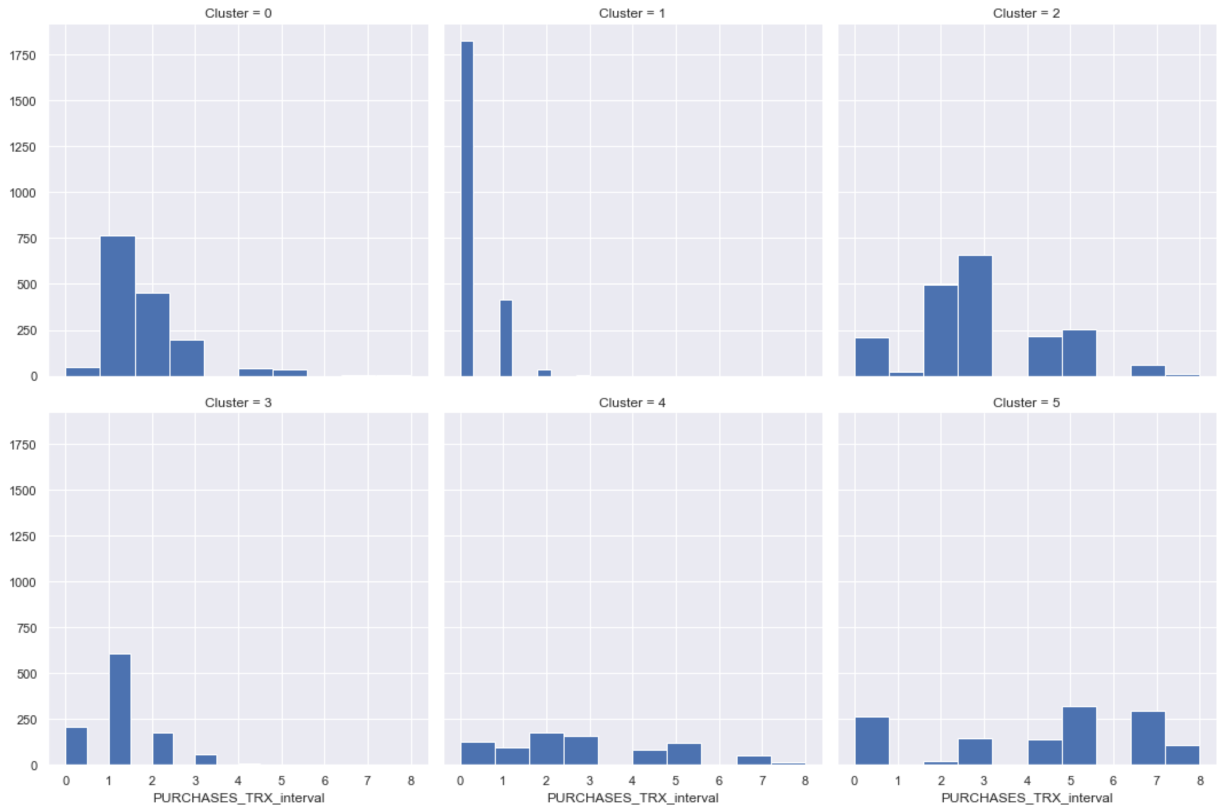
Feature : **MINIMUM_PAYMENTS_interval**




```
------------------------------------------------------------------------------------------
----------------------------------------------------------------------
```

Feature : **PURCHASES_TRX_interval**

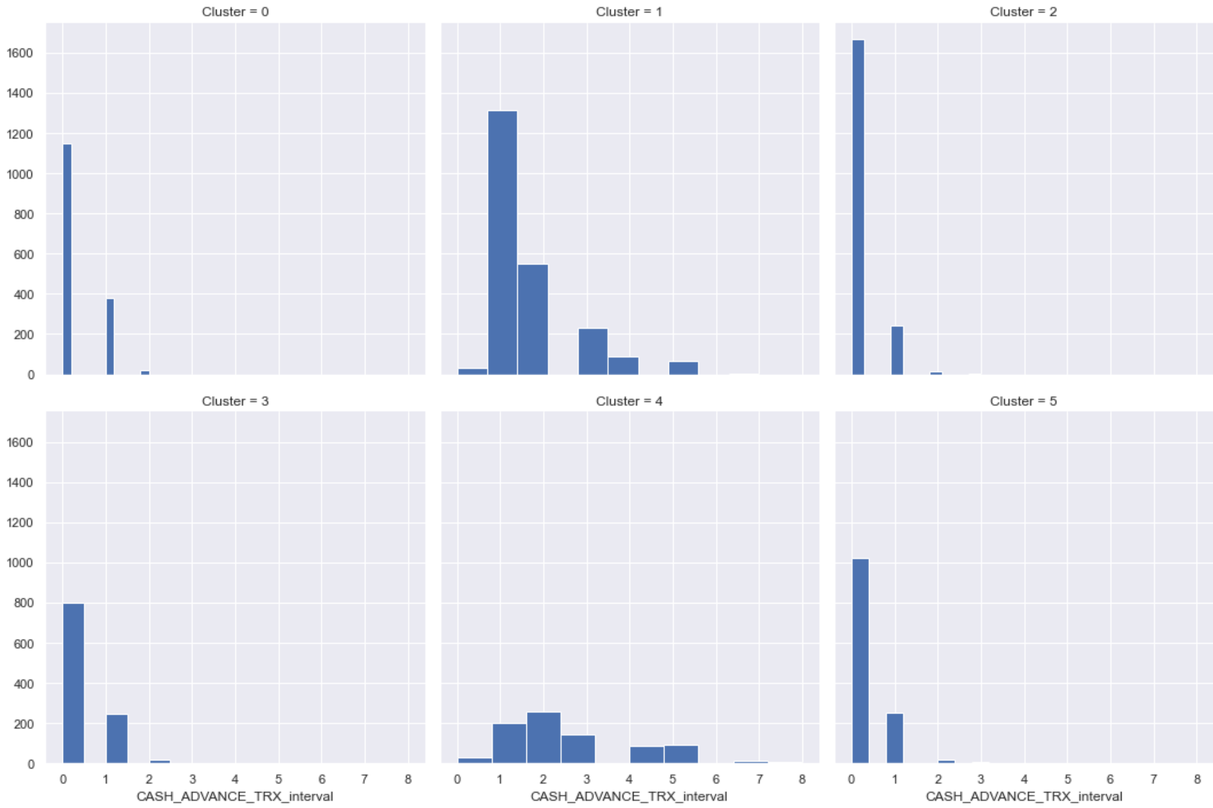
```
------------------------------------------------------------------------------------------
----------------------------------------------------------------------
```



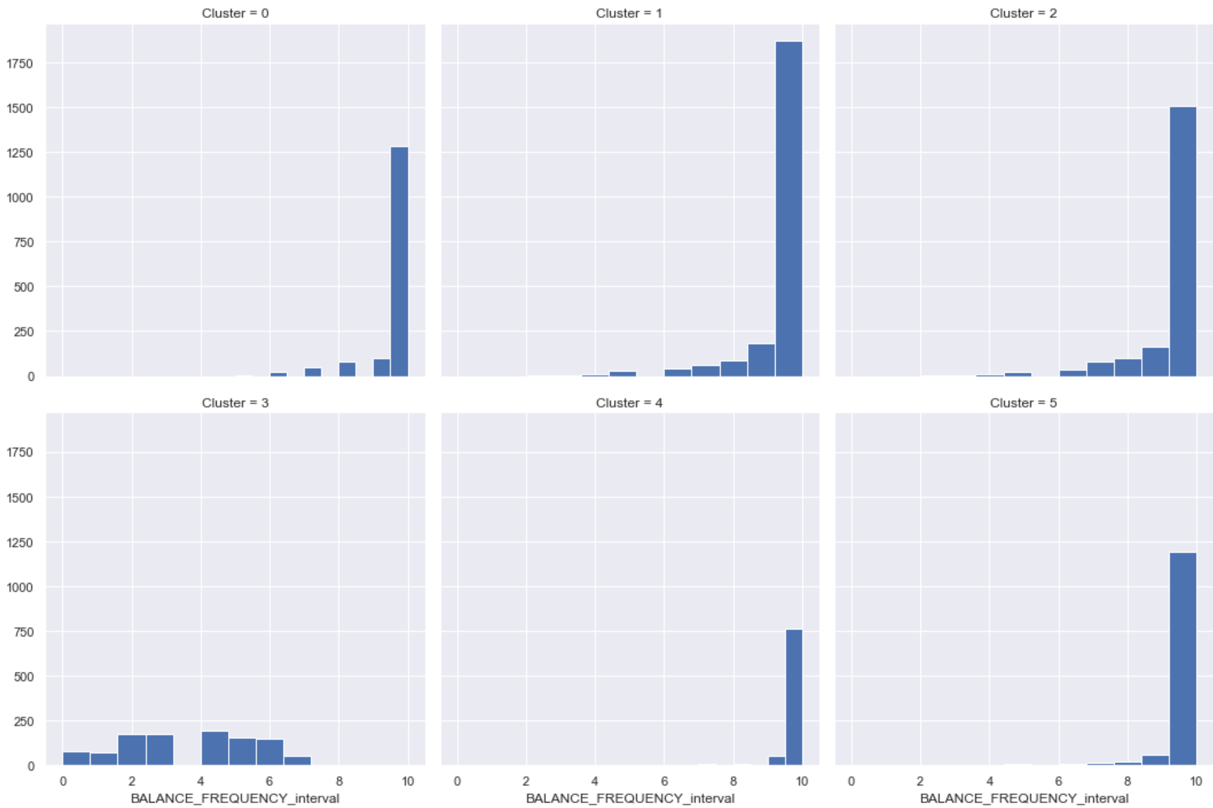

```
------------------------------------------------------------------------------------------
----------------------------------------------------------------------
```

Feature : **CASH_ADVANCE_TRX_interval**

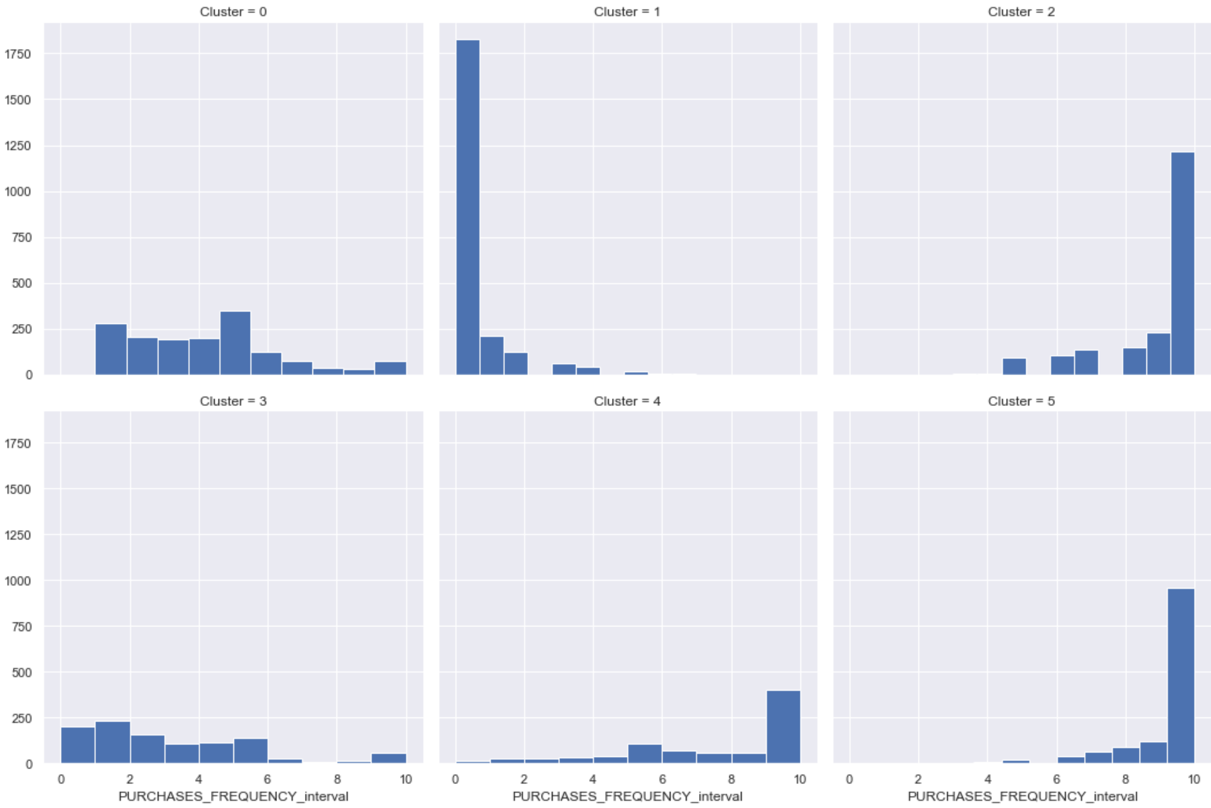
```
------------------------------------------------------------------------------------------
----------------------------------------------------------------------
```



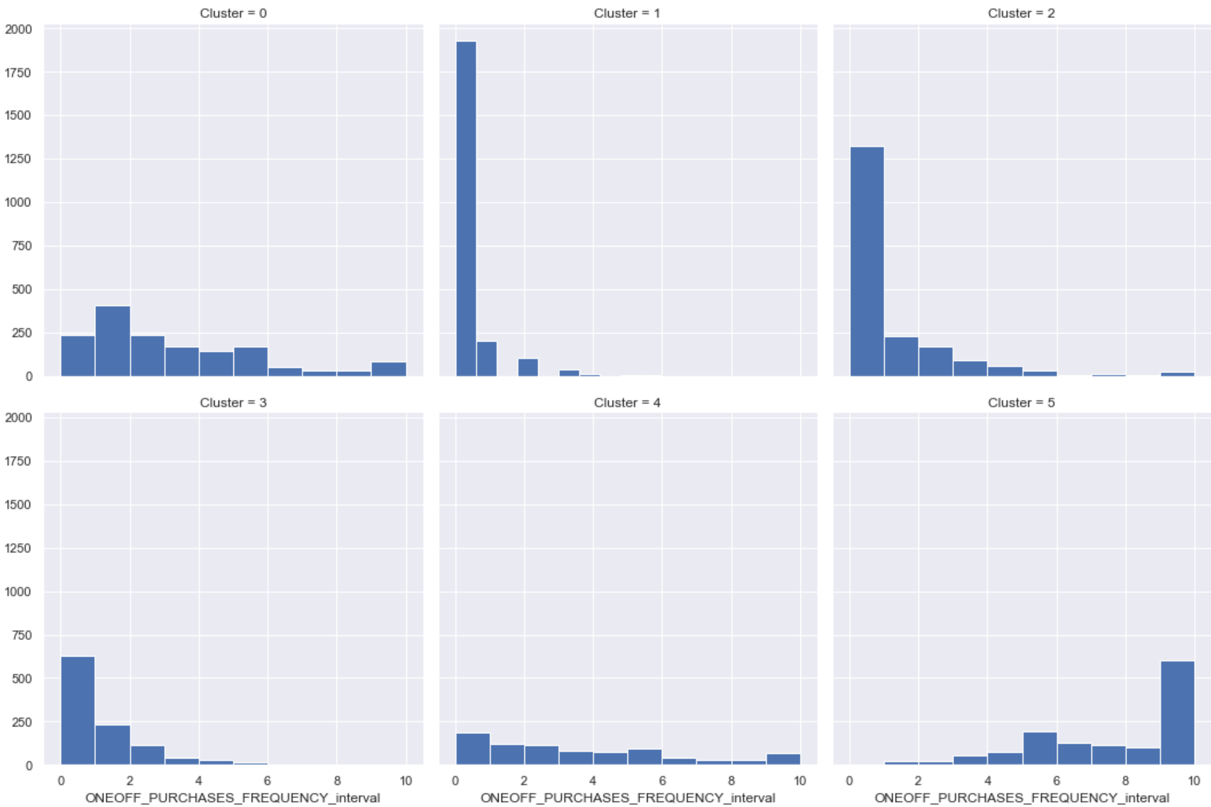
Feature : **BALANCE_FREQUENCY_interval**



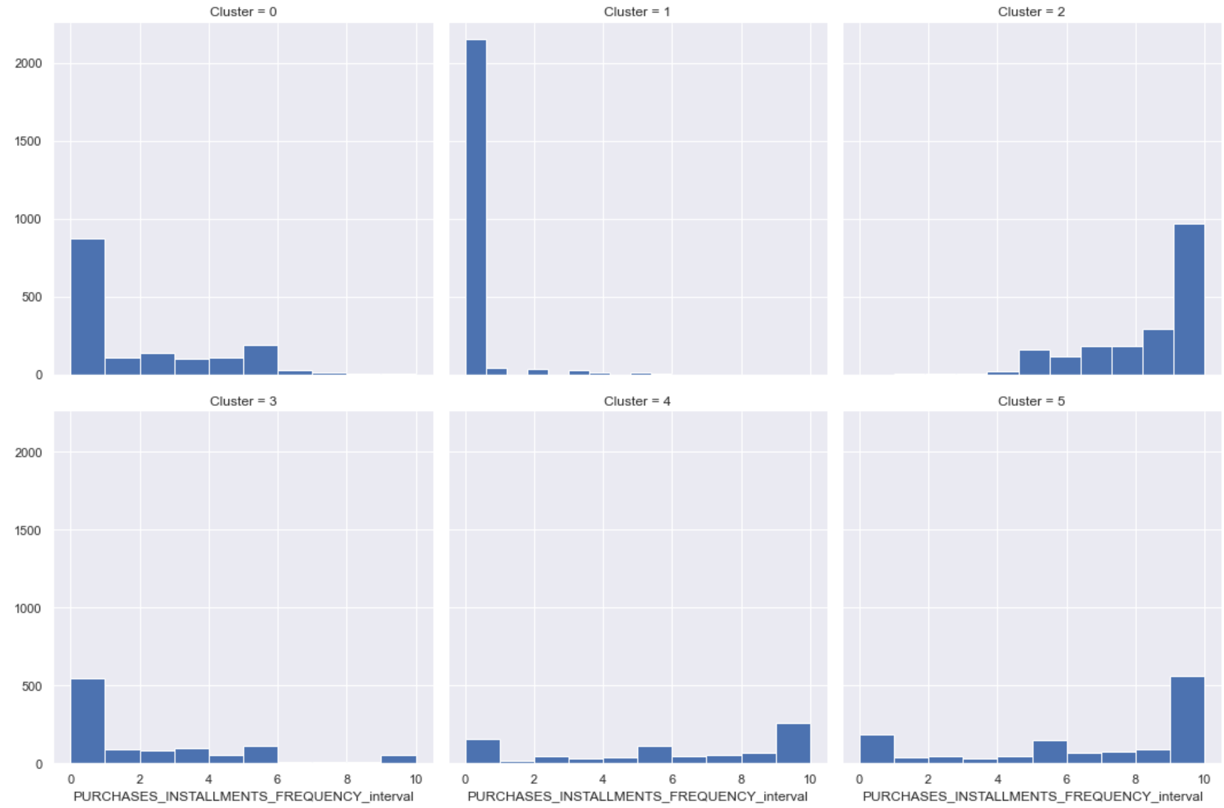
Feature : **PURCHASES_FREQUENCY_interval**



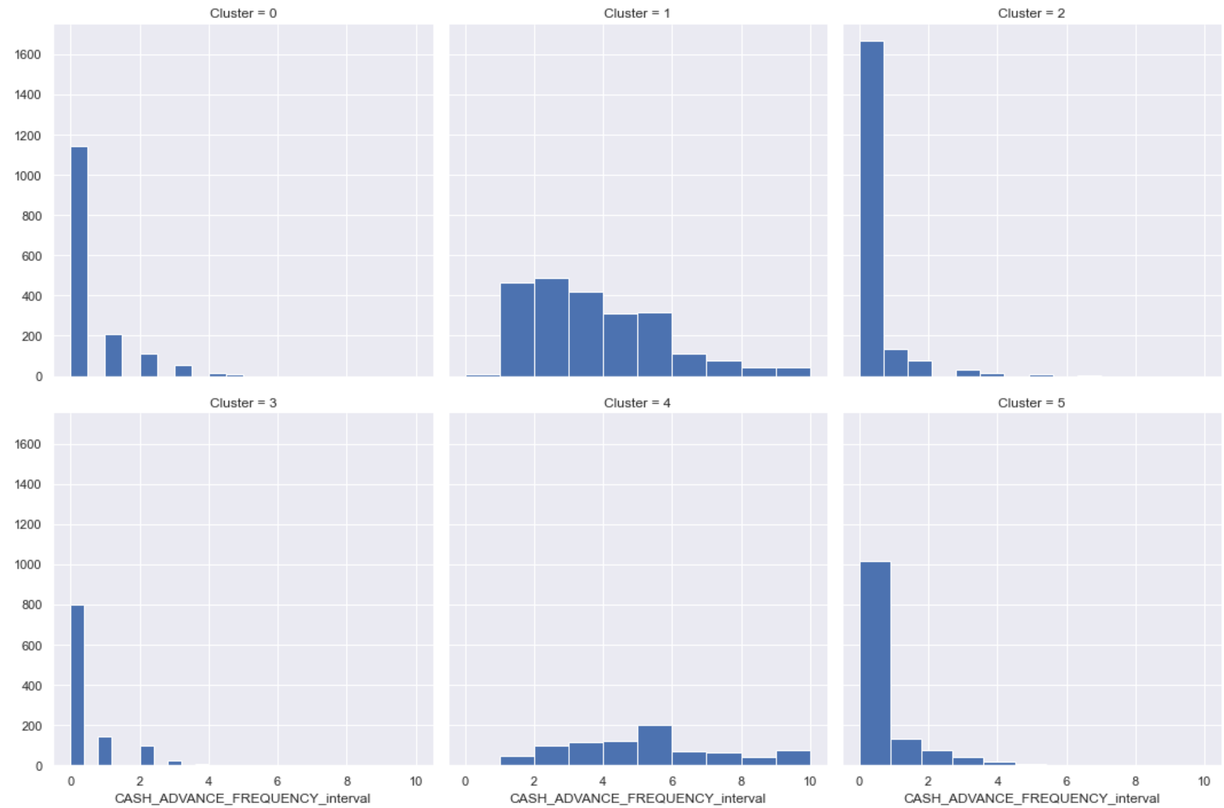
Feature : **ONEOFF_PURCHASES_FREQUENCY_interval**



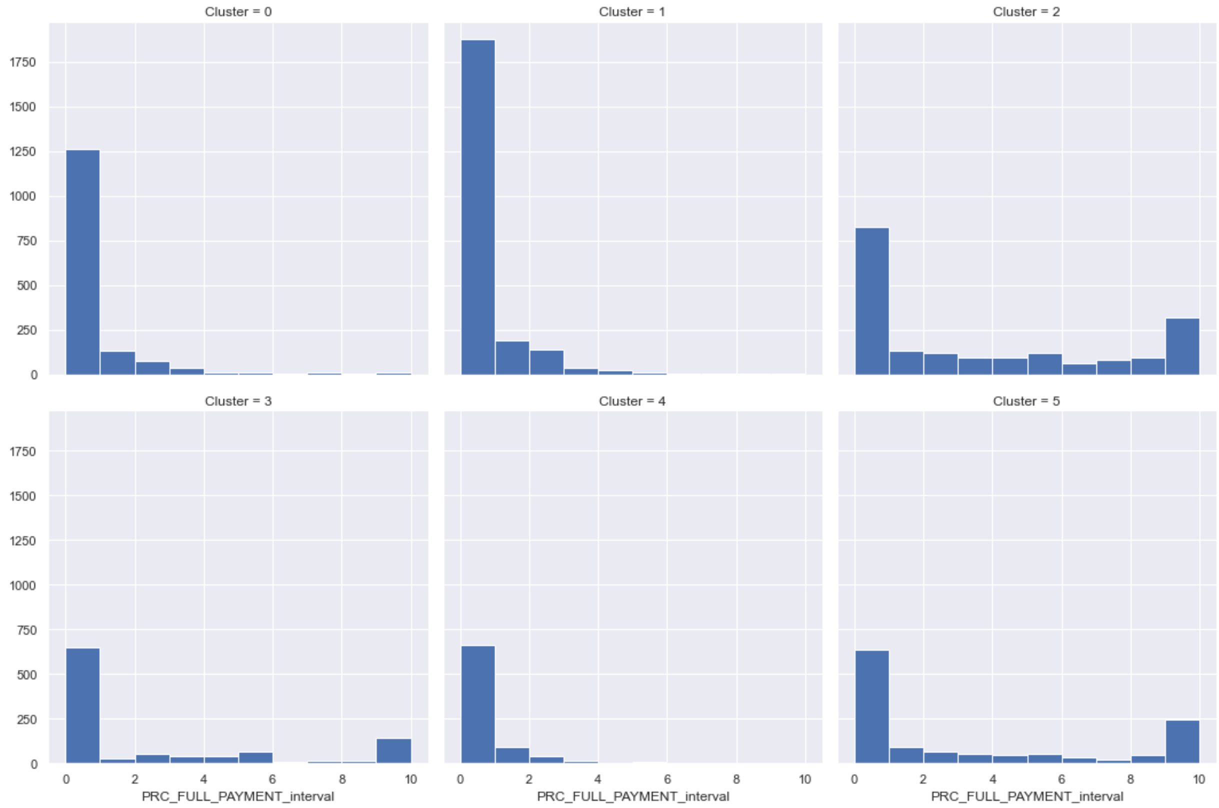
Feature : **PURCHASES_INSTALLMENTS_FREQUENCY_interval**



Feature : **CASH_ADVANCE_FREQUENCY_interval**



Feature : **PRC_FULL_PAYMENT_interval**

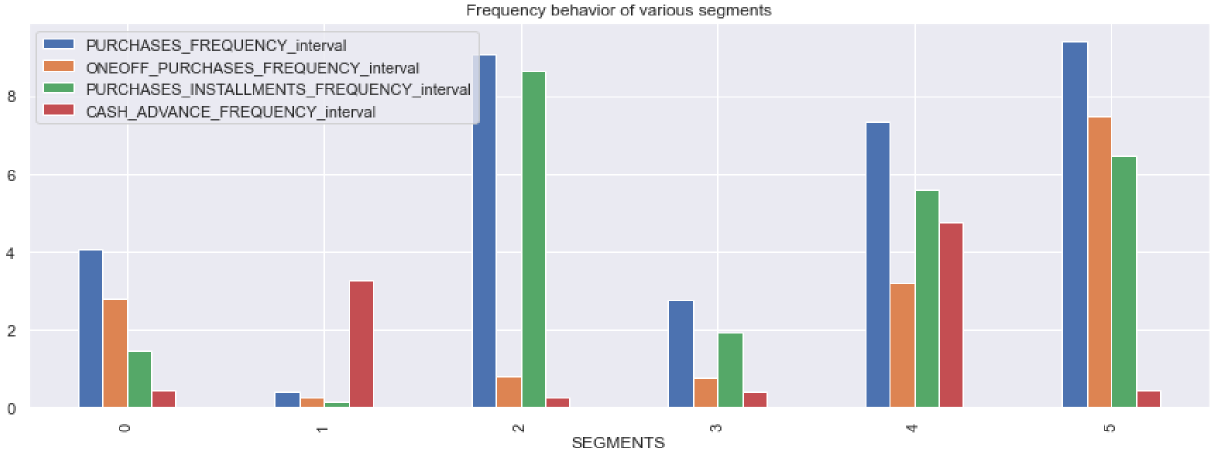


In [153…

```
# Explore features with respect to clusters
(clusters[['BALANCE_interval', 'PURCHASES_interval', 'CASH_ADVANCE_interval', 'CREDI
 .groupby('Cluster').mean().plot.bar(figsize=(15, 5)))
plt.title('Purchase Behavior of various segments of customers')
plt.xlabel('SEGMENTS');
```

In [162…

```
# Explore features with respect to clusters
(clusters[['PURCHASES_FREQUENCY_interval', 'ONEOFF_PURCHASES_FREQUENCY_interval', 'P
plt.title('Frequency behavior of various segments')
plt.xlabel('SEGMENTS');
```



Visualize clusters using PCA

Since we have very high dimensions it is not possible to plot them. To do so we will perform Principal Component Analysis which is a dimensionality reduction technique.

Principal component analysis (PCA) is a technique used to emphasize variation and bring out strong patterns in a dataset. It's often used to make data easy to explore and visualize. It is an unsupervised technique.

In [148…

```
# Import relevant functions to perform pca
from sklearn.decomposition import PCA
from sklearn.metrics.pairwise import cosine_similarity

# Perform PCA
dist = 1 - cosine_similarity(X)

# Initialize PCA object
# we will reduce dimensions to '2' for easy visualization of clusters
pca = PCA(2)

# fit on data
pca.fit(dist)

# transform the data into 2-Dimensional
X_PCA = pca.transform(dist)

X_PCA.shape
```

Out[148…

```
(8950, 2)
```

In [186…

```
print(X_PCA)
```

```
[[-11.7868436  -28.64167201]
 [-29.62154724   2.56322212]
 [ 15.75851456  13.38678885]
 ...
 [  9.2598699  -12.44331025]
 [-20.73443665 -10.8972464 ]
 [  6.09000857   7.82129244]]
```

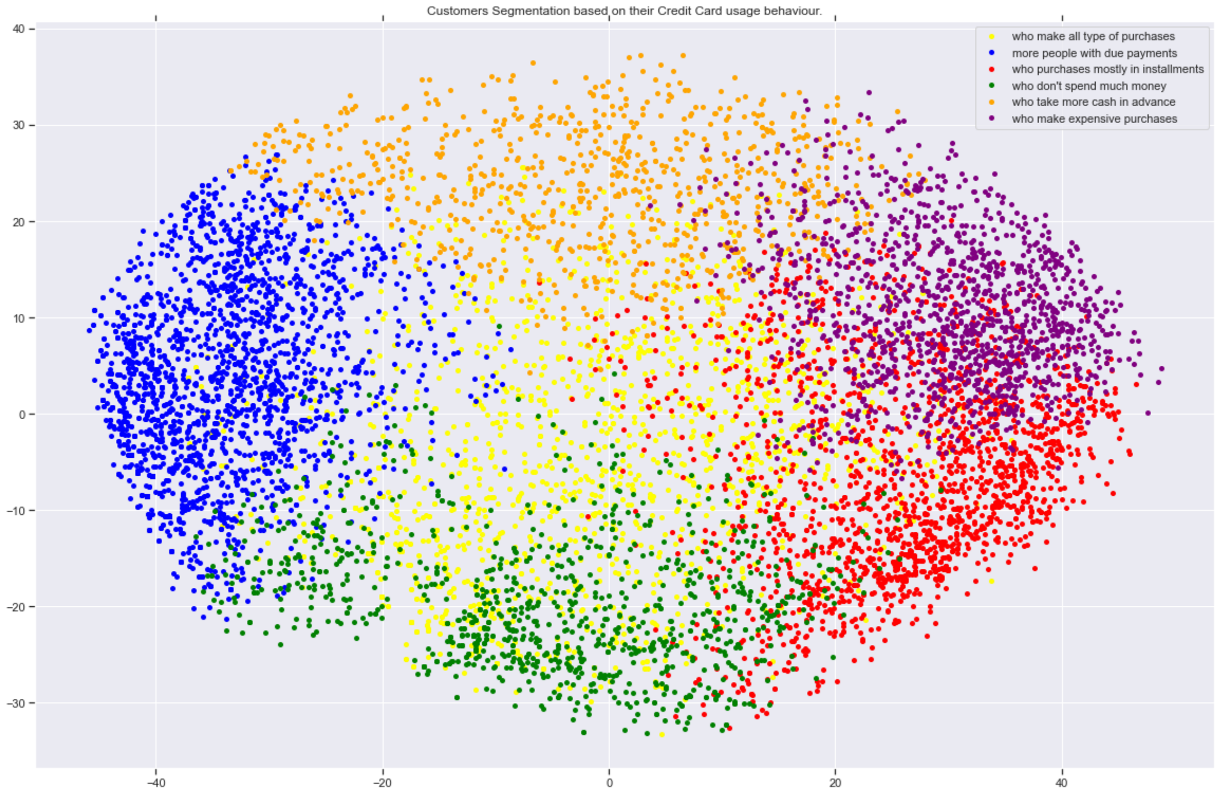
In [211…

```
import os

# Make output directory
output_path = os.getcwd() + "\\output"
```

```python
# Check if directory exist if not then create it
if not os.path.isdir(output_path):
    os.mkdir(output_path)
```

In [212…

```python
# Assign different colors for each cluster
colors = {0: 'yellow',
          1: 'blue',
          2: 'red',
          3: 'green',
          4: 'orange',
          5:'purple'}

# Assign names of clusters
names = {0: 'who make all type of purchases',
         1: 'more people with due payments',
         2: 'who purchases mostly in installments',
         3:'who don\'t spend much money',
         4: 'who take more cash in advance',
         5: 'who make expensive purchases'}

# Get feature 1 as x and feature 2 as y
x, y = X_PCA[:, 0], X_PCA[:, 1]

# Create a dataframe for grouping
df = pd.DataFrame({'x': x, 'y':y, 'label':labels})

# Group with respect to clusters
groups = df.groupby('label')

# Plot cluster
fig, ax = plt.subplots(figsize=(20, 13))

# Iterate over each cluster
for name, group in groups:
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=5,
            color=colors[name],label=names[name], mec='none')
    ax.set_aspect('auto')
    ax.tick_params(axis='x',which='both',bottom='off',top='off',labelbottom='off')
    ax.tick_params(axis= 'y',which='both',left='off',top='off',labelleft='off')

# add legend
ax.legend()

# add title
ax.set_title("Customers Segmentation based on their Credit Card usage behaviour.")

# Save cluster plot
plt.savefig(output_path + "\\cluster.png")

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



...goto toc

Conclusion

Large segments:

- **Cluster 1**: This group of customers on the other hand are not completely utilizing the credit line assigned to them. Additional investigations are needed to understand why this particular set of consumers are not utilizing their lines or if their credit lines could in the future be assigned to a different set of consumers.
- **Cluster 2**: This group of customers is in a dire need of a credit limit increase. They also have the highest activities among all the clusters.
- **Cluster 0**: This cluster belongs to customers with adequate activites and balance.
- **Cluster 5**: This cluster shows slightly higher balances and purchase activities, but higher one-off purchase behavior.

Small segments:

- **Cluster 3**: This cluster shows low balances but average activity. This cluster will be an approprite cluster for spend campaign targeting.
- **Cluster 4**: This cluster has the highest activity, balances, and purchases. This group of customers interestingly also have a higher set of credit lines, indicating that an increasing credit limit increases leads to an increase in the purchase activities. (A rigourous testing of this hypothesis should be carries out.)

In [221…

```
# Create final dataframe
final = pd.concat([raw_data.CUST_ID, clusters], axis = 1)

# The save the final dataframe
final.to_csv(output_path + '\\final.csv', index = False)
```