Project - Credit Card Segmentation

NAME: VINAYAKA O

Reg. Number: 7019066576

Email: vinayonkar333@gmail.com

Organization: edWisor

www.edWisor.com

Problem Statement:

This case requires trainees to develop a customer segmentation to define marketing strategy. The sample dataset summarizes the usage behaviour of about 9000 active credit card holders during the last 6 months. The file is at a customer level with 18 behavioural variables.

Variables:

- CUST_ID: Credit card holder ID
- BALANCE: Monthly average balance (based on daily balance averages)
- BALANCE_FREQUENCY: Ratio of last 12 months with balance
- PURCHASES: Total purchase amount spent during last 12 months
- ONEOFF_PURCHASES: Total amount of one-off purchases
- INSTALLMENTS_PURCHASES: Total amount of installment purchases
- CASH_ADVANCE: Total cash-advance amount
- PURCHASES_ FREQUENCY: Frequency of purchases (Percent of months with at least one purchase)
- ONEOFF_PURCHASES_FREQUENCY: Frequency of one-off-purchases
- PURCHASES_INSTALLMENTS_FREQUENCY: Frequency of instalment purchases
- CASH_ADVANCE_ FREQUENCY: Cash-Advance frequency
- AVERAGE_PURCHASE_TRX: Average amount per purchase transaction
- CASH_ADVANCE_TRX: Average amount per cash-advance transaction

- PURCHASES_TRX: Average amount per purchase transaction
- CREDIT_LIMIT: Credit limit
- PAYMENTS: Total payments (due amount paid by the customer to decrease their statement balance) in the period
- MINIMUM_PAYMENTS: Total minimum payments due in the period.
- PRC_FULL_PAYMEN: Percentage of months with full payment of the due statement balance
- TENURE: Number of months as a customer

Customer Segmentation

Customer Segmentation is the process of division of customer base into several groups of individuals that share a similarity in different ways that are relevant to marketing such as gender, age, interests, and miscellaneous spending habits.

Companies that deploy customer segmentation are under the notion that every customer has different requirements and require a specific marketing effort to address them appropriately. Companies aim to gain a deeper approach of the customer they are targeting. Therefore, their aim has to be specific and should be tailored to address the requirements of each and every individual customer. Furthermore, through the data collected, companies can gain a deeper understanding of customer preferences as well as the requirements for discovering valuable segments that would reap them maximum profit. This way, they can strategize their marketing techniques more efficiently and minimize the possibility of risk to their investment.

The technique of customer segmentation is dependent on several key differentiators that divide customers into groups to be targeted. Data related to demographics, geography, economic status as well as behavioural patterns play a crucial role in determining the company direction towards addressing the various segments.

About the Given data:

The given data set is raw data set and this data set is **unsupervised data set**, so at very high end I am going to use **K-means Algorithm (clustering)**

Missing Value Analysis:

The Missing Value Analysis procedure performs three primary functions:

- Describes the pattern of missing data. Where are the missing values located? How extensive are they? Do pairs of variables tend to have values missing in multiple cases? Are data values extreme? Are values missing randomly?

- Estimates means, standard deviations, covariance's, and correlations for different missing value methods: list wise, pairwise, regression, or expectation-maximization. The pairwise method also displays counts of pairwise complete cases.

- Imputes missing values is done in 4 major methods
  - Actual value
  - Mean
  - Median
  - KNN imputation

Missing value analysis helps address several concerns caused by incomplete data. If cases with missing values are systematically different from cases without missing values, the results can be misleading. Also, missing data may reduce the precision of calculated statistics because there is less information than originally planned. Another concern is that the assumptions behind many statistical procedures are based on complete cases, and missing values can complicate the theory required.

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

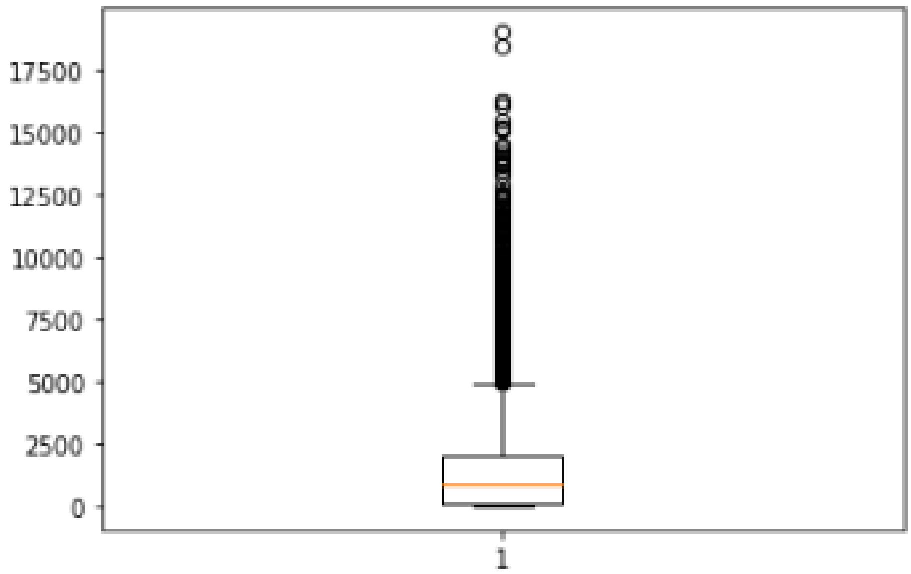
Outlier Analysis:

An outlier is an element of a data set that distinctly stands out from the rest of the data. In other words, outliers are those data points that lie outside the overall pattern of distribution.

The easiest way to detect outliers is to create a graph. Plots such as Box plots, Scatterplots and Histograms can help to detect outliers. Alternatively, we can use mean and standard deviation to list out the outliers. Interquartile Range and Quartiles can also be used to detect outliers.

Outlier data points can represent either a) items that are so far outside the norm that they need not be considered or b) the illustration of a very unique and singular category or variable that is worth exploring either to capitalize on a niche or find an area where an organization can offer a unique focus.

When considering the use of Outlier analysis, a business should first think about why they want to find the outliers and what they will do with that data. That focus will help the business to select the right method of analysis, graphing or plotting to reveal the results they need to see and understand.



Data Cleaning:

- Data cleaning involve different techniques based on the problem and the data type. Different methods can be applied with each has its own trade-offs.

- Overall, incorrect data is either removed, corrected, or imputed.

Factor Analysis:

Factor analysis is a technique that is used to reduce a large number of variables into fewer numbers of factors. This technique extracts maximum common variance from all variables and puts them into a common score. As an index of all variables, we can use this score for further analysis. Factor analysis is part of general linear model (GLM) and this method also assumes several assumptions: there is linear relationship, there is no multicollinearity, it includes relevant variables into analysis, and there is true correlation between variables and factors. Several methods are available, but principle component analysis is used most commonly.

Data Standardization:

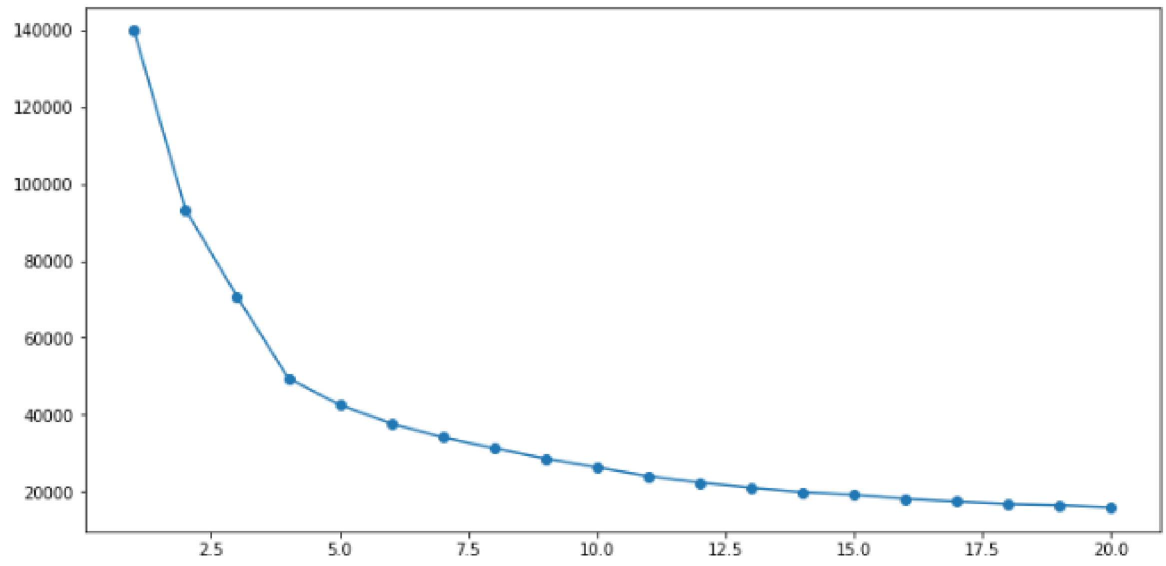
Data Standardization is a data processing workflow that converts the structure of disparate datasets into a Common Data Format. As part of the Data Preparation field, Data Standardization deals with the transformation of datasets after the data is pulled from source systems and before it's loaded into target systems. Because of that, Data Standardization can also be thought of as the transformation rules engine in Data Exchange operations.

Data Standardization enables the data consumer to analyse and use data in a consistent manner. Typically, when data is created and stored in the source system, it's structured in a particular way that is often unknown to the data consumer. Moreover, datasets that might be semantically related may be stored and represented differently, thereby making it difficult for a data consumer to aggregate or compare the datasets.

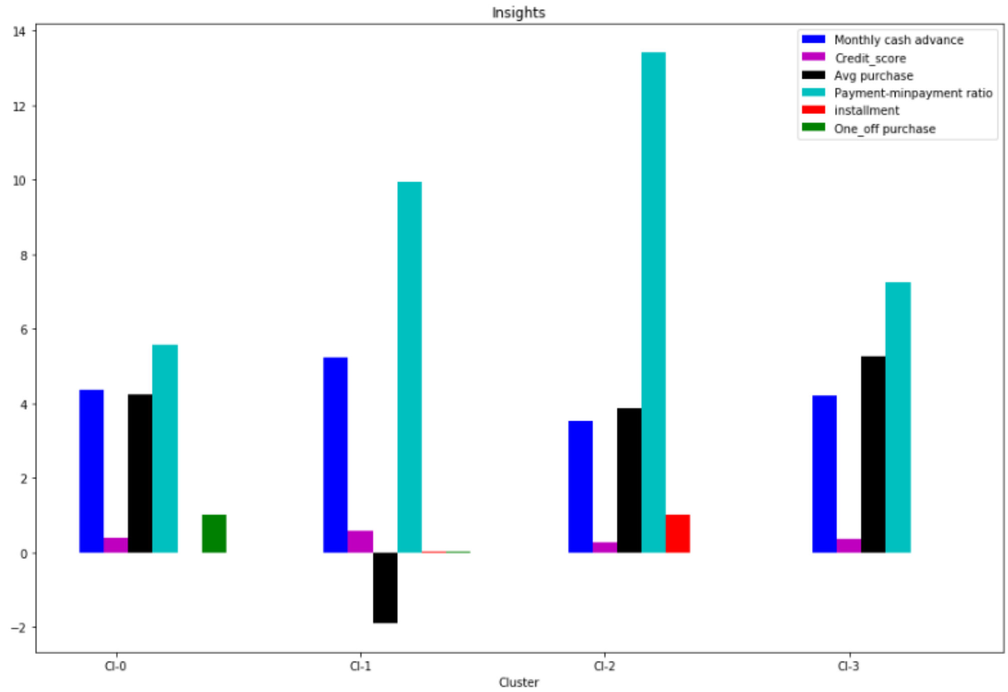
K-Means Clustering:

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.

A cluster refers to a collection of data points aggregated together because of certain similarities. We will define a target number k, which refers to the number of centroids we need in the dataset. A centroid is the imaginary or real location representing the centre of the cluster. Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. The ‘means’ in the K-means refers to averaging of the data that is finding the centroid.



With 4 cluster



Clusters are clearly distinguishing behaviour within customers

---

- Cluster 0 customers are doing maximum One_Off transactions and has least payment ratio amongst all the cluster.

- Cluster 1 is the group of customers who have highest monthly cash advance and doing both instalment as well as one_off purchases, have comparatively good credit score but have poor average purchase score.

- Cluster 2 customers have maximum Average Purchase and good Monthly cash advance but this cluster doesn't do instalment or one_off purchases.

- cluster 3 is doing maximum instalment, has maximum payment too min_payment ratio and doesn't do one-off purchases

Findings through clustering are validating Insights derived from KPI. (as shown above in Insights from KPI

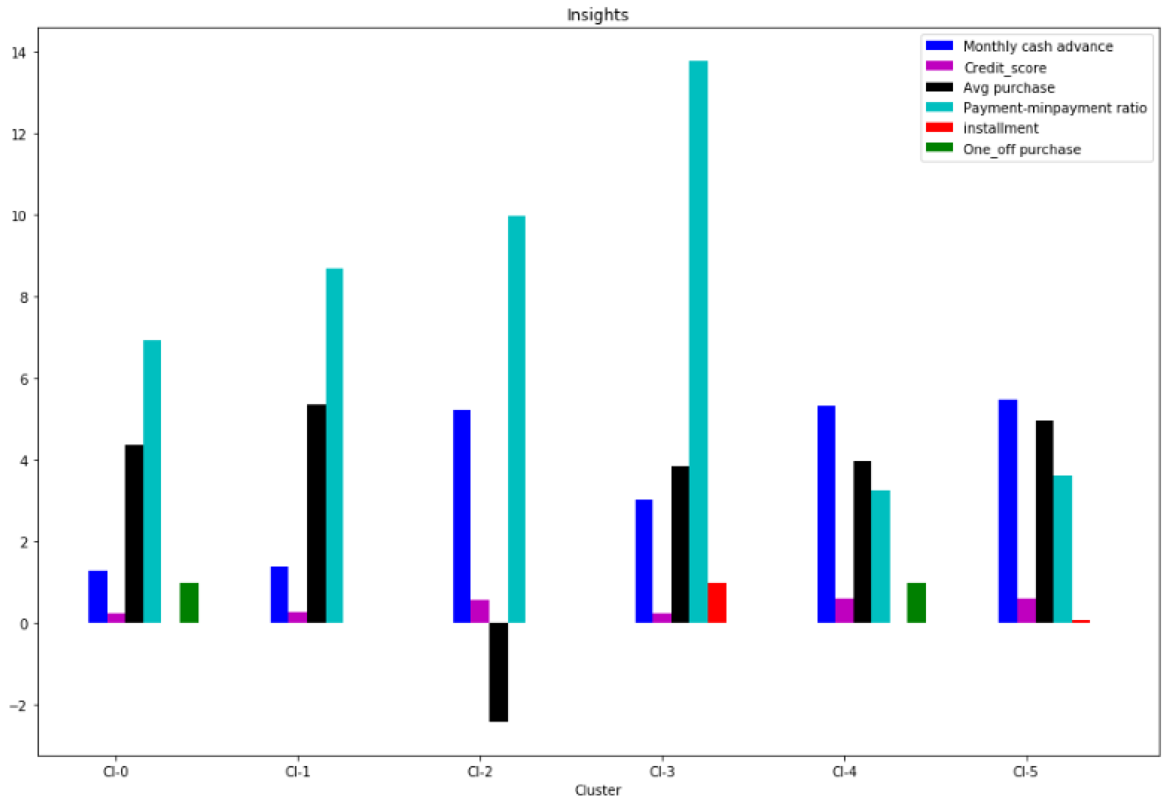
With 5 clusters:

- We have a group of customers having highest average purchases but there is Cluster 4 also having highest cash advance & second highest purchase behaviour but their type of purchases is same.
- Cluster 0 and Cluster 4 are behaving similar in terms of Credit limit and have cash transactions is on higher side

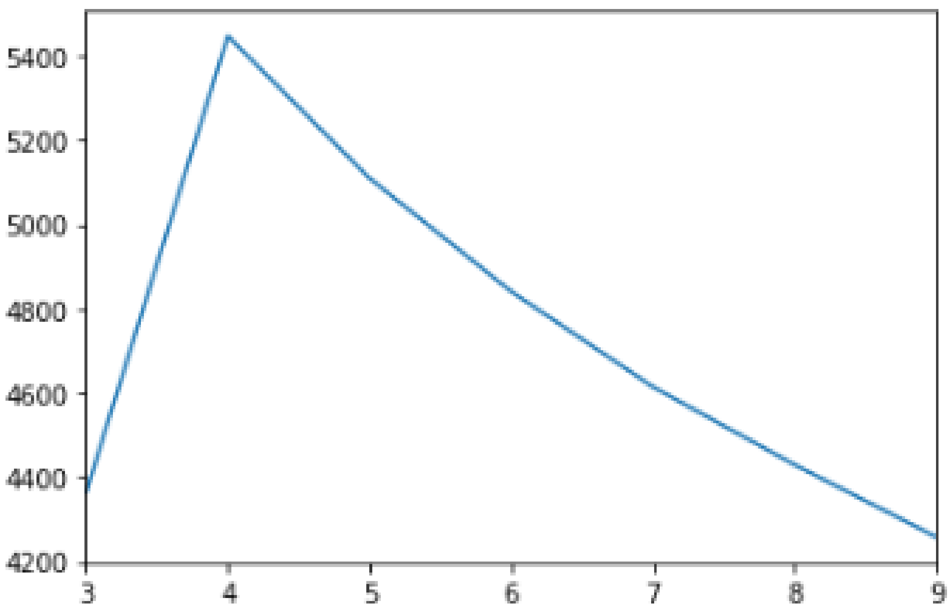
So we don't have quite distinguishable characteristics with 5 clusters.

Insights with 6 clusters

- Here also groups are overlapping.



Checking performance metrics for K means



Performance metrics also suggest that K-means with 4 clusters is able to show distinguished characteristics of each cluster.

- Cluster 0 customers are doing maximum One_Off transactions and have least payment ratio amongst the entire cluster. Low credit score.
- Cluster 1 is the group of customers who have highest Monthly cash advance and doing both instalment as well as one_off purchases, have comparatively good credit score but have poor average purchase score. poor credit score
- Cluster 2 customers have maximum Average Purchase and good Monthly cash advance but this cluster doesn't do instalment or one_off purchases and has good credit score.
- cluster 3 is doing maximum instalment, has maximum payment too min_payment ratio and doesn't do one-off purchases. Max credit score

Conclusion:

a. Group 2

- They are potential target customers who are paying dues and doing purchases and maintaining comparatively good credit score we can increase credit limit or can lower down interest rate Can be given premium card /loyalty cards to increase transactions

b. Group 1

- They have poor credit score and taking only cash on advance. We can target them by providing less interest rate on purchase transaction

c. Group 0

- This group is has minimum paying ratio and using card for just one off transactions (may be for utility bills only). This group seems to be risky group.

d. Group 3

- This group is performing best among all as customers are maintaining good credit score and paying dues on time. Giving rewards point will make them perform more purchases.

Profiling:

Data profiling is the process of examining the data available from an existing information source (e.g. a database or a file) and collecting statistics or informative summaries about that data.

Problems need to address:

Advanced data preparation: Build an 'enriched' customer profile by deriving "intelligent" KPIs such as:

- Monthly average purchase and cash advance amount
- Purchases by type (one-off, instalments)
- Average amount per purchase and cash advance transaction,
- Limit usage (balance to credit limit ratio),
- Payments to minimum payments ratio etc.

Advanced reporting:

- Use the derived KPIs to gain insight on the customer profiles.
- Identification of the relationships/ affinities between services.
- Clustering: Apply a data reduction technique factor analysis for variable reduction technique and a clustering algorithm to reveal the behavioural segments of credit card holders
- Identify cluster characteristics' of the cluster using detailed profiling.

- Provide the strategic insights and implementation of strategies for given set of cluster characteristics

DATA DICTIONARY:

- CUST_ID: Credit card holder ID
- BALANCE: Monthly average balance (based on daily balance averages)
- BALANCE_FREQUENCY: Ratio of last 12 months with balance
- PURCHASES: Total purchase amount spent during last 12 months
- ONEOFF_PURCHASES: Total amount of one-off purchases
- INSTALLMENTS_PURCHASES: Total amount of installment purchases
- CASH_ADVANCE: Total cash-advance amount
- PURCHASES_ FREQUENCY: Frequency of purchases (Percent of months with at least one purchase)
- ONEOFF_PURCHASES_FREQUENCY: Frequency of one-off-purchases
- PURCHASES_INSTALLMENTS_FREQUENCY: Frequency of instalment purchases
- CASH_ADVANCE_ FREQUENCY: Cash-Advance frequency
- AVERAGE_PURCHASE_TRX: Average amount per purchase transaction
- CASH_ADVANCE_TRX: Average amount per cash-advance transaction
- PURCHASES_TRX: Average amount per purchase transaction
- CREDIT_LIMIT: Credit limit

- PAYMENTS: Total payments (due amount paid by the customer to decrease their statement balance) in the period
- MINIMUM_PAYMENTS: Total minimum payments due in the period.
- PRC_FULL_PAYMEN: Percentage of months with full payment of the due statement balance

- TENURE: Number of months as a customer

Instruction to deploy and run code

To deploy and the code of python and R

- The attached python code file in jupyter notebook format (i.e. .ipynb format)
- We can run that whole code in jypyter notebook by by uploading attached python notebook.
- The attached R code file in R studio format.
- We can run that codes in R studio by opening the attached R file.

The file names are given below.

Python code: Credit_Card_Segmentation_FINAL...ipynb

R code file: Credit_Card_Segmentation_FINAL.R

Python code:

Credit Card Segmentation Problem

In [1]:

```
#importing required libraries
import os
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
import seaborn as sns
```

In [2]:

```
#Set working directory
os.chdir("C:/Users/Hp/Desktop/Project")
```

Data Pre-Processing

In [3]:

```
# reading data into dataframe
credit_card= pd.read_csv("credit_card_data.csv",sep=',')
```

In [4]:

```
credit_card.head()
```

Out[4]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083

In [6]:

```
credit_card.info()
```

Out [6]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
CUST_ID                             8950 non-null object
BALANCE                             8950 non-null float64
BALANCE_FREQUENCY                   8950 non-null float64
PURCHASES                           8950 non-null float64
ONEOFF_PURCHASES                    8950 non-null float64
INSTALLMENTS_PURCHASES              8950 non-null float64
CASH_ADVANCE                        8950 non-null float64
PURCHASES_FREQUENCY                 8950 non-null float64
ONEOFF_PURCHASES_FREQUENCY          8950 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY    8950 non-null float64
CASH_ADVANCE_FREQUENCY              8950 non-null float64
CASH_ADVANCE_TRX                    8950 non-null int64
PURCHASES_TRX                       8950 non-null int64
CREDIT_LIMIT                        8949 non-null float64
PAYMENTS                            8950 non-null float64
MINIMUM_PAYMENTS                    8637 non-null float64
PRC_FULL_PAYMENT                    8950 non-null float64
```

```
TENURE                              8950 non-null int64
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

In [7]:

```
credit_card.shape
```

Out [7]:

```
(8950, 18)
```

In [8]:

```
# Intital descriptive analysis of data.
credit_card.describe()
```

Out [8]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0.490351
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	0.083333
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000

In [8]:

```
credit_card['CREDIT_LIMIT'].isnull().value_counts()
```

Out [8]:

```
False    8949
True        1
Name: CREDIT_LIMIT, dtype: int64
```

In [9]:

```
print (credit_card['CREDIT_LIMIT'].describe())
```

Out [9]:

```
count     8949.000000
mean      4494.449450
std       3638.815725
min         50.000000
25%       1600.000000
50%       3000.000000
```

```
75%        6500.000000
max       30000.000000
Name: CREDIT_LIMIT, dtype: float64
```

In [10]:

```
credit_card[credit_card['CREDIT_LIMIT'].isnull()]
```

Out[10]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUE
5203	C15349	18.400472	0.166667	0.0	0.0	0.0	186.853063	

Missing Value Analysis

As of now we observe that from our Analysis in Given data, there are Missing value in the data.

In [11]:

```
#Create dataframe with missing percentage
missing_val = pd.DataFrame(credit_card.isnull().sum())
```

In [12]:

```
missing_val
```

Out [12]:

	0
CUST_ID	0
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

	0
CREDIT_LIMIT	1
PAYMENTS	0
MINIMUM_PAYMENTS	313
PRC_FULL_PAYMENT	0
TENURE	0

In [13]:

```
#Reset index
missing_val = missing_val.reset_index()
```

In [14]:

```
#Reset index
missing_val
```

Out [14]:

	index	0
0	CUST_ID	0
1	BALANCE	0
2	BALANCE_FREQUENCY	0
3	PURCHASES	0
4	ONEOFF_PURCHASES	0
5	INSTALLMENTS_PURCHASES	0
6	CASH_ADVANCE	0
7	PURCHASES_FREQUENCY	0
8	ONEOFF_PURCHASES_FREQUENCY	0
9	PURCHASES_INSTALLMENTS_FREQUENCY	0
10	CASH_ADVANCE_FREQUENCY	0
11	CASH_ADVANCE_TRX	0
12	PURCHASES_TRX	0
13	CREDIT_LIMIT	1
14	PAYMENTS	0

	index	0
15	MINIMUM_PAYMENTS	313
16	PRC_FULL_PAYMENT	0
17	TENURE	0

In [15]:

```
#Rename variable
missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missing_perce
ntage'})
```

In [16]:

```
missing_val
```

Out [16]:

	Variables	Missing_percentage
0	CUST_ID	0
1	BALANCE	0
2	BALANCE_FREQUENCY	0
3	PURCHASES	0
4	ONEOFF_PURCHASES	0
5	INSTALLMENTS_PURCHASES	0
6	CASH_ADVANCE	0
7	PURCHASES_FREQUENCY	0
8	ONEOFF_PURCHASES_FREQUENCY	0
9	PURCHASES_INSTALLMENTS_FREQUENCY	0
10	CASH_ADVANCE_FREQUENCY	0
11	CASH_ADVANCE_TRX	0
12	PURCHASES_TRX	0
13	CREDIT_LIMIT	1
14	PAYMENTS	0
15	MINIMUM_PAYMENTS	313
16	PRC_FULL_PAYMENT	0

	Variables	Missing_percentage
17	TENURE	0

In [17]:

```
#Calculate percentage
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(credit_c
ard))*100
```

In [18]:

```
missing_val
```

Out [18]:

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

In [19]:

```
#descending order

missing_val = missing_val.sort_values('Missing_percentage', ascending = False).rese
t_index(drop = True)
```

In [20]:

```
missing_val
```

Out [20]:

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

In [21]:

```
#save output results

missing_val.to_csv("Miising_perc.csv", index = False)
```

In missing value analysis, there two approches are there one is deleting methode and Impution methode

Here i am going to imputation approche. Very high level imputation, following methodes are used.

- Actual value
- Mean
- Median
- KNN

Based on accuracy between these methods. I am going to freeze that method for my farther missing value treatment.
Based on accuracy i am going to impute the given data with **median**

In [22]:

```
#Impute with median of missing value variable i.e CREDIT_LIMT & MINIMUM_PAYMENT
credit_card['CREDIT_LIMIT'].fillna(credit_card['CREDIT_LIMIT'].median(),inplace=True)
credit_card['MINIMUM_PAYMENTS'].fillna(credit_card['MINIMUM_PAYMENTS'].median(),inplace=True)
print (credit_card.isnull().sum())
```

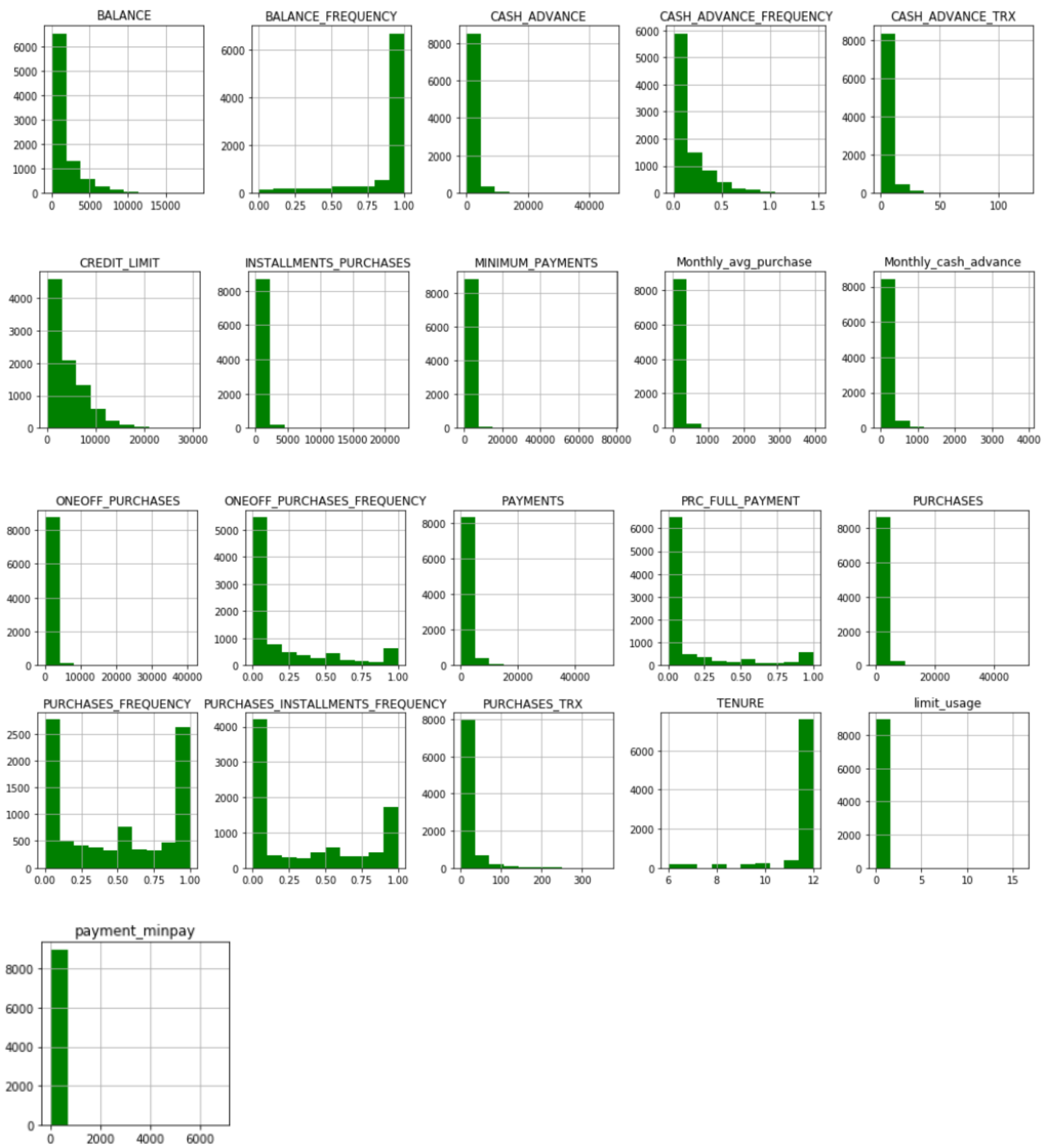
Out [22]:

```
CUST_ID                             0
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
```

In [23]:

```
# EXPORATORY DATA ANALYSIS
credit_card.hist(figsize=(18,18),color = "green" );
```

Out [23]:



As per given Problem Statement

1. Deriving New KPI's

a) Monthly_avg_purchase and Cash Advance Amount

In [24]:

```
#deriving KPI's for Monthly_avg_purchase and Cash_Advance
credit_card['Monthly_avg_purchase']=credit_card['PURCHASES']/credit_card['TENURE']
credit_card['Monthly_cash_advance']=credit_card['CASH_ADVANCE']/credit_card['TENURE
']
```

In [25]:

```
#view of top 5 observation of 'monthly_avg_purchase'
credit_card['Monthly_avg_purchase'].head()
```

Out[25]:

```
0      7.950000
1      0.000000
2     64.430833
3    124.916667
4      1.333333
Name: Monthly_avg_purchase, dtype: float64
```

In [26]:

```
#view of top 5 observation of 'monthly_cash_advance'
credit_card['Monthly_cash_advance'].head()
```

Out[26]:

```
0      0.000000
1    536.912124
2      0.000000
3     17.149001
4      0.000000
Name: Monthly_cash_advance, dtype: float64
```

In [27]:

```
credit_card[credit_card['ONEOFF_PURCHASES']==0]['ONEOFF_PURCHASES'].count()
```

Out[27]:

```
4302
```

b) Purchases by type

To find what type of purchases customers are making on credit card,lets explore the data.

In [28]:

```
#top 20 observation of onoff_purchase and Installment_purchases of given data
credit_card.loc[:,['ONEOFF_PURCHASES','INSTALLMENTS_PURCHASES']].head(20)
```

Out [28]:

	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES
0	0.00	95.40
1	0.00	0.00
2	773.17	0.00
3	1499.00	0.00
4	16.00	0.00
5	0.00	1333.28
6	6402.63	688.38
7	0.00	436.20
8	661.49	200.00
9	1281.60	0.00
10	0.00	920.12
11	1492.18	0.00
12	2500.23	717.76
13	419.96	1717.97
14	0.00	0.00
15	0.00	1611.70
16	0.00	0.00
17	0.00	519.00
18	166.00	338.35
19	0.00	398.64

In [29]:

```
#shape of onoff_purchases and installments_purchases in People who do none probebil
ity

credit_card[(credit_card['ONEOFF_PURCHASES']==0) & (credit_card['INSTALLMENTS_PURCH
ASES']==0)].shape
```

Out[29]:

```
(2042, 20)
```

In [30]:

```
#shape of onoff_purchases and installments_purchases in People who do both probebil
ity

credit_card[(credit_card['ONEOFF_PURCHASES']>0) & (credit_card['INSTALLMENTS_PURCHA
SES']>0)].shape
```

Out[30]:

```
(2774, 20)
```

In [31]:

```
#shape of onoff_purchases and installments_purchases in People who only do One-Off
Purchases probebility

credit_card[(credit_card['ONEOFF_PURCHASES']>0) & (credit_card['INSTALLMENTS_PURCHA
SES']==0)].shape
```

Out[31]:

```
(1874, 20)
```

In [32]:

```
#shape of onoff_purchases and installments_purchases in People who only do Installm
ents Purchases probebility.

credit_card[(credit_card['ONEOFF_PURCHASES']==0) & (credit_card['INSTALLMENTS_PURCH
ASES']>0)].shape
```

Out[32]:

```
(2260, 20)
```

I found out that there are 4 types of purchase behaviour in the data set.

1.People who only do One-Off Purchases.

2.People who only do Installments Purchases.

3.People who do both.

4.People who do none.

So deriving a categorical variable based on the behaviour.

In [33]:

```
def purchase(credit_card):

    if (credit_card['ONEOFF_PURCHASES']==0) & (credit_card['INSTALLMENTS_PURCHASES'
]==0):

        return 'none'

    if (credit_card['ONEOFF_PURCHASES']>0) & (credit_card['INSTALLMENTS_PURCHASES']
>0):

         return 'both_oneoff_installment'

    if (credit_card['ONEOFF_PURCHASES']>0) & (credit_card['INSTALLMENTS_PURCHASES']
==0):

        return 'one_off'

    if (credit_card['ONEOFF_PURCHASES']==0) & (credit_card['INSTALLMENTS_PURCHASES'
]>0):

        return 'istallment'
```

In [34]:

```
credit_card['purchase_type']=credit_card.apply(purchase,axis=1)
```

In [35]:

```
credit_card['purchase_type'].value_counts()
```

Out[35]:

```
both_oneoff_installment    2774
istallment                 2260
none                       2042
one_off                    1874
Name: purchase_type, dtype: int64
```

c) Limit_Usage (balance to credit limit ratio)

```
Lower value implies cutomers are maintaing thier balance properly.

as per data anaysis Lower value means good credit score
```

In [36]:

```
#formula used to calculate balance to credit limit ration

credit_card['limit_usage']=credit_card.apply(lambda x: x['BALANCE']/x['CREDIT_LIMIT
'], axis=1)
```

In [37]:

```
# top 5 observation of 'limit_usage'

credit_card['limit_usage'].head()
```

Out[37]:

```
0    0.040901
1    0.457495
2    0.332687
3    0.222223
4    0.681429
Name: limit_usage, dtype: float64
```

d) Payment to minimum payments Ratio

In [38]:

```
# formula used to caluculate payment to minimum payments Ratio

credit_card['payment_minpay']=credit_card.apply(lambda x:x['PAYMENTS']/x['MINIMUM_P
AYMENTS'],axis=1)

credit_card['payment_minpay'].describe()
```

Out[38]:

```
count    8950.000000
mean        9.059164
std       118.180526
min         0.000000
25%         0.913275
50%         2.032717
75%         6.052729
max      6840.528861
Name: payment_minpay, dtype: float64
```

In [39]:

```
credit_card.info()
```

Out [39]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 23 columns):
CUST_ID                            8950 non-null object
BALANCE                            8950 non-null float64
BALANCE_FREQUENCY                  8950 non-null float64
PURCHASES                          8950 non-null float64
ONEOFF_PURCHASES                   8950 non-null float64
INSTALLMENTS_PURCHASES             8950 non-null float64
CASH_ADVANCE                       8950 non-null float64
PURCHASES_FREQUENCY                8950 non-null float64
ONEOFF_PURCHASES_FREQUENCY         8950 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY   8950 non-null float64
CASH_ADVANCE_FREQUENCY             8950 non-null float64
CASH_ADVANCE_TRX                   8950 non-null int64
PURCHASES_TRX                      8950 non-null int64
CREDIT_LIMIT                       8950 non-null float64
PAYMENTS                           8950 non-null float64
MINIMUM_PAYMENTS                   8950 non-null float64
PRC_FULL_PAYMENT                   8950 non-null float64
TENURE                             8950 non-null int64
Monthly_avg_purchase               8950 non-null float64
Monthly_cash_advance               8950 non-null float64
purchase_type                      8950 non-null object
limit_usage                        8950 non-null float64
payment_minpay                     8950 non-null float64
dtypes: float64(18), int64(3), object(2)
memory usage: 1.6+ MB
```

Extreme value Treatment

- Since there are variables having extreme values, I am doing log-transformation on the dataset to remove outlier effect

In [40]:

```
# log tranformation
cr_log=credit_card.drop(['CUST_ID','purchase_type'],axis=1).applymap(lambda x: np.l
og(x+1))
```

In [41]:

```
cr_log.describe()
```

Out[40]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	O
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	
mean	6.161637	0.619940	4.899647	3.204274	3.352403	3.319086	0.361268	
std	2.013303	0.148590	2.916872	3.246365	3.082973	3.566298	0.277317	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	4.861995	0.635989	3.704627	0.000000	0.000000	0.000000	0.080042	
50%	6.773521	0.693147	5.892417	3.663562	4.499810	0.000000	0.405465	
75%	7.628099	0.693147	7.013133	6.360274	6.151961	7.016449	0.650588	
max	9.854515	0.693147	10.800403	10.615512	10.021315	10.760839	0.693147	

8 rows × 21 columns

In [42]:

```
col=['BALANCE','PURCHASES','CASH_ADVANCE','TENURE','PAYMENTS','MINIMUM_PAYMENTS','P
RC_FULL_PAYMENT','CREDIT_LIMIT']

cr_pre=cr_log[[x for x in cr_log.columns if x not in col ]]
```

2. Insights from derived KPI's on the Customer Profile

In [43]:

```
# Average payment_minpayment ratio for each purchse type.

x=credit_card.groupby('purchase_type').apply(lambda x: np.mean(x['payment_minpay'])
)

type(x)

x.values
```

Out[43]:

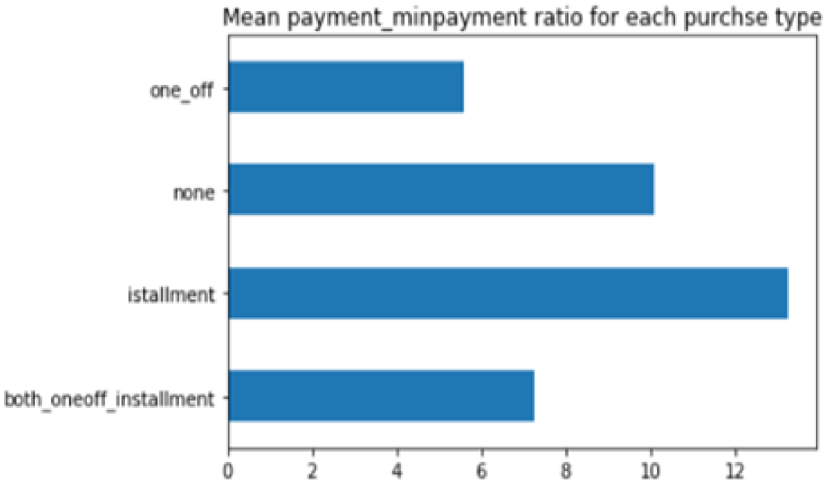
```
array([ 7.23698216, 13.2590037 , 10.08745106,  5.57108156])
```

In [44]:

```
credit_card.groupby('purchase_type').apply(lambda x: np.mean(x['payment_minpay'])).
plot.barh()


plt.title('Mean payment_minpayment ratio for each purchse type')
```

Out[44]:

```
Text(0.5, 1.0, 'Mean payment_minpayment ratio for each purchse type')
```



In [45]:

```
credit_card.describe()
```

Out [45]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000
mean	1564.474828	0.877271	1003.204834	592.437371	411.067645	978.871112	0.490351
std	2081.531879	0.236904	2136.634782	1659.887917	904.338115	2097.163877	0.401371
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	128.281915	0.888889	39.635000	0.000000	0.000000	0.000000	0.083333
50%	873.385231	1.000000	361.280000	38.000000	89.000000	0.000000	0.500000
75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.916667
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.000000

8 rows × 21 columns

Insight 1: Customers With Instalment Purchases are Paying Dues

In [46]:

```
credit_card[credit_card['purchase_type']=='n']
```

Out [46]:

CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY

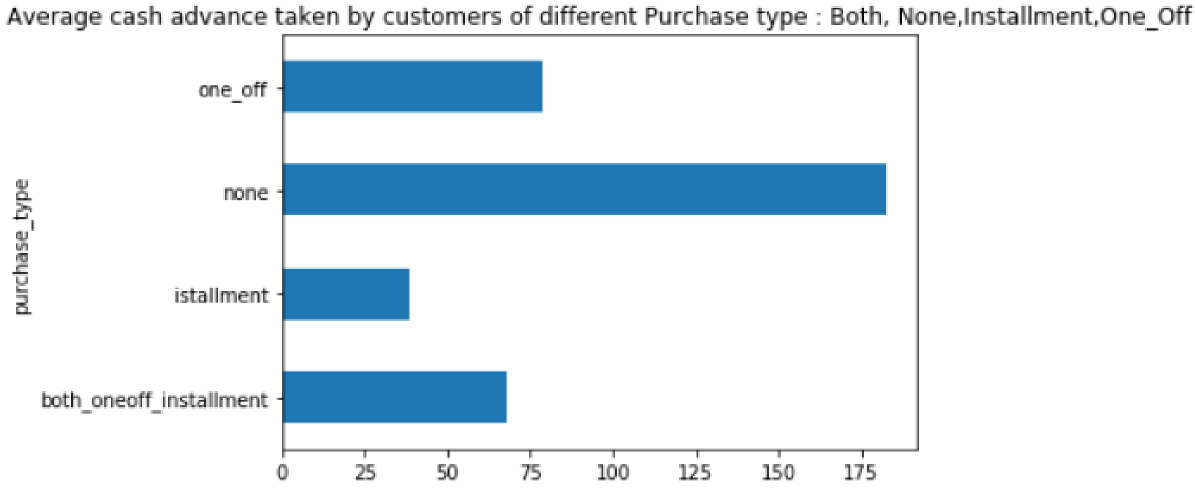
0 rows × 23 columns

In [47]:

```
credit_card.groupby('purchase_type').apply(lambda x: np.mean(x['Monthly_cash_advanc
e'])).plot.barh()


plt.title('Average cash advance taken by customers of different Purchase type : Bot
h, None,Installment,One_Off')
```

Out [47]:

```
Text(0.5, 1.0, 'Average cash advance taken by customers of different
Purchase type : Both, None,Installment,One_Off')
```



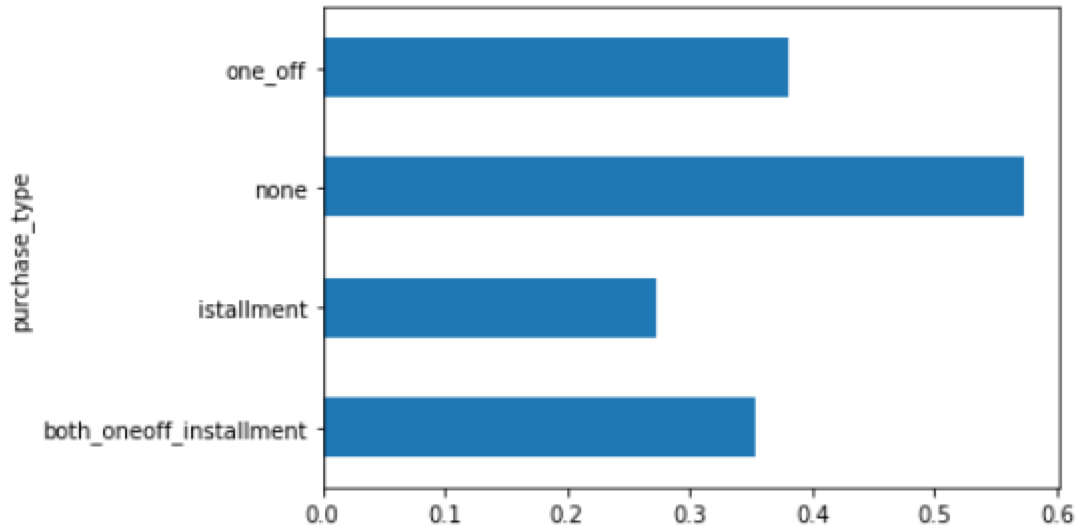
Insight 2: Customers who don't do either one-off or installment purchases take more cash on advance.

In [48]:

```
credit_card.groupby('purchase_type').apply(lambda x: np.mean(x['limit_usage'])).plo
t.barh()
```

Out [48]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd3f1bda88>
```



Insight 3: Customers with installment purchases have good credit score.

In [49]:

```
# Original dataset with categorical column converted to number type.
cre_original=pd.concat([credit_card,pd.get_dummies(credit_card['purchase_type'])],a
xis=1)
```

In [50]:

```
cre_original
```

Out [50]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQU
0	C10001	40.900749	0.818182	95.40	0.00	95.40	0.000000	0.
1	C10002	3202.467416	0.909091	0.00	0.00	0.00	6442.945483	0.
2	C10003	2495.148862	1.000000	773.17	773.17	0.00	0.000000	1.
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.00	205.788017	0.
4	C10005	817.714335	1.000000	16.00	16.00	0.00	0.000000	0.
...	...	...	...	...	...	...	...	
8945	C19186	28.493517	1.000000	291.12	0.00	291.12	0.000000	1.
8946	C19187	19.183215	1.000000	300.00	0.00	300.00	0.000000	1.
8947	C19188	23.398673	0.833333	144.40	0.00	144.40	0.000000	0.
8948	C19189	13.457564	0.833333	0.00	0.00	0.00	36.558778	0.
8949	C19190	372.708075	0.666667	1093.25	1093.25	0.00	127.040008	0.

8950 rows × 27 columns

3. Preparing for Machine learning

In [51]:

```
# creating Dummies for categorical variable
cr_pre['purchase_type']=credit_card.loc[:,'purchase_type']
pd.get_dummies(cr_pre['purchase_type']).head()
```

Out [51]:

	both_oneoff_installment	istallment	none	one_off
0	0	1	0	0
1	0	0	1	0
2	0	0	0	1
3	0	0	0	1
4	0	0	0	1

In [52]:

```
cr_dummy=pd.concat([cr_pre,pd.get_dummies(cr_pre['purchase_type'])],axis=1)
```

In [53]:

```
l=['purchase_type']
```

In [54]:

```
cr_dummy=cr_dummy.drop(l,axis=1)
cr_dummy.isnull().sum()
```

Out [54]:

```
BALANCE_FREQUENCY                   0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
Monthly_avg_purchase                0
Monthly_cash_advance                0
limit_usage                         0
payment_minpay                      0
both_oneoff_installment             0
istallment                          0
none                                0
one_off                             0
```

dtype: int64

In [55]:

```
cr_dummy.describe()
```

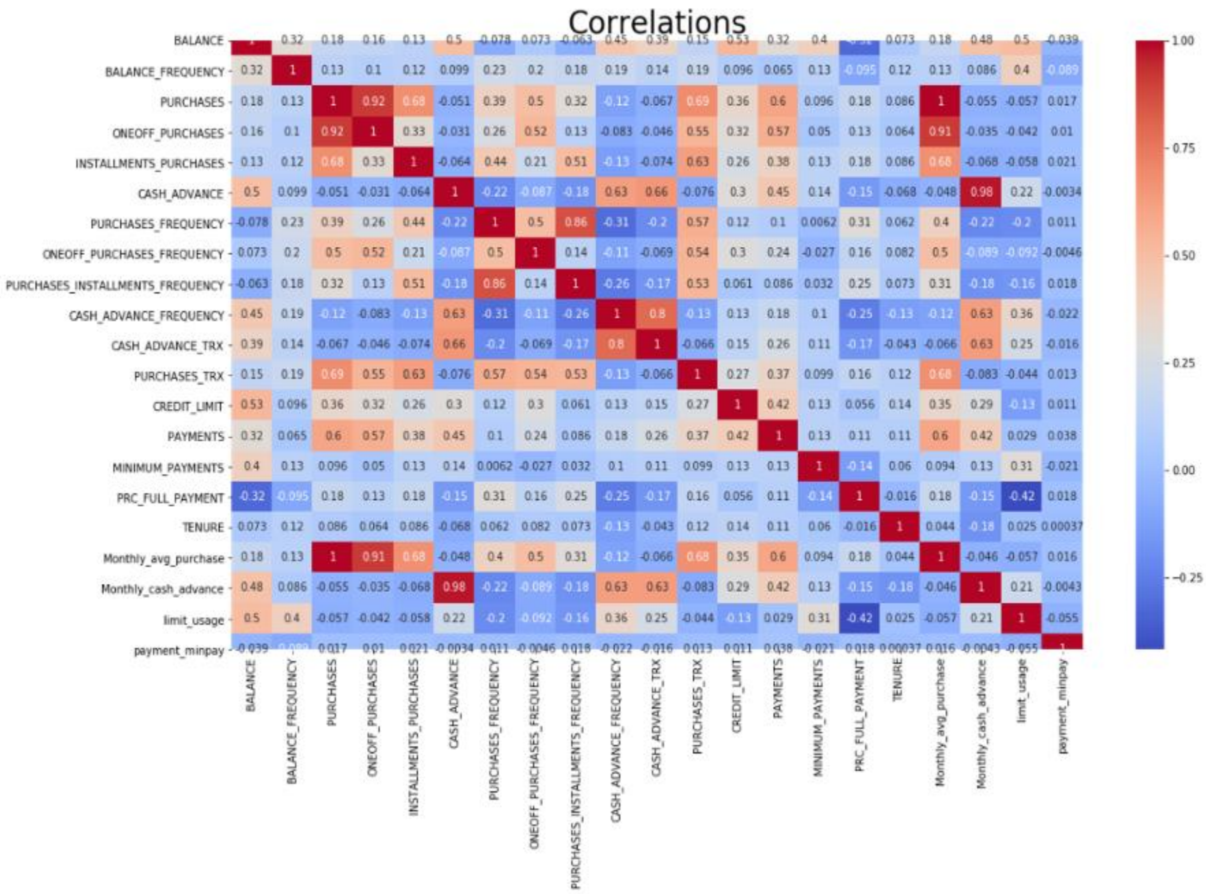
Out [55]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASE
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	
mean	0.619940	3.204274	3.352403	0.361268	0.158699	
std	0.148590	3.246365	3.082973	0.277317	0.216672	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.635989	0.000000	0.000000	0.080042	0.000000	
50%	0.693147	3.663562	4.499810	0.405465	0.080042	
75%	0.693147	6.360274	6.151961	0.650588	0.262364	
max	0.693147	10.615512	10.021315	0.693147	0.693147	

In [56]:

```
#creating heat map
plt.figure(figsize=(18,10))
sns.heatmap(credit_card.corr(),cmap='coolwarm',annot=True);
plt.title('Correlations', size = 28);
```

Out [56]:



Heat map shows that many features are co-related so applying dimensionality reduction will help negating multi-colinearity in data

- Before applying PCA we will standardize data to avoid effect of scale on our result. Cantering and Scaling will make all features with equal weight.

Standardizing data

- To put data on the same scale

In [57]:

```
#importing required libraries
from sklearn.preprocessing import  StandardScaler


#Standardrizing data
sc=StandardScaler()
cr_scaled=sc.fit_transform(cr_dummy)
```

Applying PCA

In [58]:

```
#importing PCA libraries
from sklearn.decomposition import PCA
```

In [59]:

```
cr_dummy.shape
```

Out [59]:

```
(8950, 17)
```

In [60]:

```
#We have 17 features so our n_component will be 17.
pc=PCA(n_components=17)
cr_pca=pc.fit(cr_scaled)
```

In [61]:

```
#Lets check if we will take 17 component then how much varience it explain. Ideally
it should be 1 i.e 100%
sum(cr_pca.explained_variance_ratio_)
```

Out [61]

```
1.0
```

In [62]:

```
var_ratio={}
for n in range(4,15):
    pc=PCA(n_components=n)
    cr_pca=pc.fit(cr_scaled)
    var_ratio[n]=sum(cr_pca.explained_variance_ratio_)
```

In [63]:

```
var_ratio
```

Out [63]:

```
{4: 0.8115442762351258,
 5: 0.8770555795291441,
 6: 0.9186492443512623,
 7: 0.9410925256030133,
 8: 0.9616114053683062,
 9: 0.9739787081990642,
 10: 0.9835896584630714,
 11: 0.9897248107341953,
 12: 0.9927550009135224,
 13: 0.9953907562385427,
 14: 0.9979616898169594}
```

Since 6 components are explaining about 90% variance so we select 6 components

In [64]:

```
pc=PCA(n_components=6)
```

In [65]:

```
p=pc.fit(cr_scaled)
```

In [66]:

```
cr_scaled.shape
```

Out [66]:

```
(8950, 17)
```

In [67]:

```
p.explained_variance_
```

Out [67]:

```
array([6.83574755, 3.07030693, 2.50427698, 1.38746289, 1.1138166 ,
       0.70717132])
```

In [68]:

```
np.sum(p.explained_variance_)
```

Out [68]:

```
15.618782269308797
```

In [69]:

```
var_ratio
```

Out [69]:

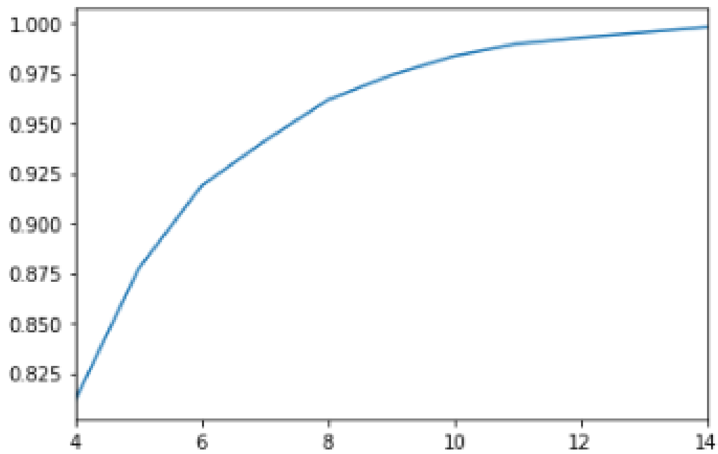
```
{4: 0.8115442762351258,
 5: 0.8770555795291441,
 6: 0.9186492443512623,
 7: 0.9410925256030133,
 8: 0.9616114053683062,
 9: 0.9739787081990642,
 10: 0.9835896584630714,
 11: 0.9897248107341953,
 12: 0.9927550009135224,
 13: 0.9953907562385427,
 14: 0.9979616898169594}
```

In [70]:

```
#plot of var_ratio
pd.Series(var_ratio).plot()
```

Out [70]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd44cfe808>
```



Since 6 components are explaining about 90% variance so we select 6 components

In [71]:

```
cr_scaled.shape
```

Out [71]:

```
(8950, 17)
```

In [72]:

```
pc_final=PCA(n_components=6).fit(cr_scaled)

reduced_cr=pc_final.fit_transform(cr_scaled)
```

In [73]:

```
dd=pd.DataFrame(reduced_cr)
```

In [74]:

```
dd.head()
```

Out [74]:

	0	1	2	3	4	5
0	-0.242841	-2.759668	0.343061	-0.417359	-0.007100	0.019755
1	-3.975652	0.144625	-0.542989	1.023832	-0.428929	-0.572463
2	1.287396	1.508938	2.709966	-1.892252	0.010809	-0.599932
3	-1.047613	0.673103	2.501794	-1.306784	0.761348	1.408986
4	-1.451586	-0.176336	2.286074	-1.624896	-0.561969	-0.675214

In [75]:

```
dd.shape
```

Out [75]:

```
(8950, 6)
```

In [76]:

```
col_list=cr_dummy.columns
```

In [77]:

```
col_list
```

Out [77]:

```
Index(['BALANCE_FREQUENCY', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'Monthly_avg_purchase',
       'Monthly_cash_advance', 'limit_usage', 'payment_minpay',
       'both_oneoff_installment', 'istallment', 'none', 'one_off'],
      dtype='object')
```

In [78]:

```
pd.DataFrame(pc_final.components_.T, columns=['PC_' +str(i) for i in range(6)],index=col_list)
```

Out [78]:

	PC_0	PC_1	PC_2	PC_3	PC_4	PC_5
BALANCE_FREQUENCY	0.029707	0.240072	-0.263140	-0.353549	-0.228681	-0.693816
ONEOFF_PURCHASES	0.214107	0.406078	0.239165	0.001520	-0.023197	0.129094
INSTALLMENTS_PURCHASES	0.312051	-0.098404	-0.315625	0.087983	-0.002181	0.115223
PURCHASES_FREQUENCY	0.345823	0.015813	-0.162843	-0.074617	0.115948	-0.081879
ONEOFF_PURCHASES_FREQUENCY	0.214702	0.362208	0.163222	0.036303	-0.051279	-0.097299
PURCHASES_INSTALLMENTS_FREQUENCY	0.295451	-0.112002	-0.330029	0.023502	0.025871	0.006731
CASH_ADVANCE_FREQUENCY	-0.214336	0.286074	-0.278586	0.096353	0.360132	0.066589
CASH_ADVANCE_TRX	-0.229393	0.291556	-0.285089	0.103484	0.332753	0.082307
PURCHASES_TRX	0.355503	0.106625	-0.102743	-0.054296	0.104971	-0.009402
Monthly_avg_purchase	0.345992	0.141635	0.023986	-0.079373	0.194147	0.015878

	PC_0	PC_1	PC_2	PC_3	PC_4	PC_5
Monthly_cash_advance	-0.243861	0.264318	-0.257427	0.135292	0.268026	0.058258
limit_usage	-0.146302	0.235710	-0.251278	-0.431682	-0.181885	0.024298
payment_minpay	0.119632	0.021328	0.136357	0.591561	0.215446	-0.572467
both_oneoff_installment	0.241392	0.273676	-0.131935	0.254710	-0.340849	0.294708
istallment	0.082209	-0.443375	-0.208683	-0.190829	0.353821	-0.086087
none	-0.310283	-0.005214	-0.096911	0.245104	-0.342222	-0.176809
one_off	-0.042138	0.167737	0.472749	-0.338549	0.362585	-0.060698

So above data gave us Eigen vector for each component we had all Eigen vector value very small we can remove that variable but in our case it’s not.

In [79]:

```
# Factor Analysis : variance explained by each component-
pd.Series(pc_final.explained_variance_ratio_,index=['PC_'+ str(i) for i in range(6)
])
```

Out [79]:

```
PC_0    0.402058
PC_1    0.180586
PC_2    0.147294
PC_3    0.081606
PC_4    0.065511
PC_5    0.041594
dtype: float64
```

In [80]:

```
type(cr_pca)
```

Out [80]:

```
sklearn.decomposition.pca.PCA
```

Clustering

Based on the intuition on type of purchases made by customers and their distinctive behavior exhibited based on the purchase_type (as visualized above in Insights from KPI), I am starting with 4 clusters.

In [81]:

```
# Importing clustering libraries
from sklearn.cluster import KMeans
```

In [82]:

```
#K means algorithm of 4 cluster
km_4=KMeans(n_clusters=4,random_state=123)
```

In [83]:

```
km_4.fit(reduced_cr)
```

Out [83]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=123, tol=0.0001, verbose=0)
```

In [84]:

```
km_4.labels_
```

Out [84]:

```
array([2, 1, 0, ..., 2, 1, 0])
```

In [85]:

```
pd.Series(km_4.labels_).value_counts()
```

Out [85]:

```
3    2769
2    2224
1    2088
```

```
0    1869
dtype: int64
```

Here we donot have known k value so we will find the K. To do that we need to take a cluster range between 1 and 21.

Identify cluster Error.

In [86]:

```
cluster_range = range( 1, 21 )
cluster_errors = []

for num_clusters in cluster_range:
    clusters = KMeans( num_clusters )
    clusters.fit( reduced_cr )
    cluster_errors.append( clusters.inertia_ )# clusters.inertia_ is basically clus
ter error here.
```

In [87]:

```
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors": clust
er_errors } )

clusters_df[0:21]
```

Out [87]:

	num_clusters	cluster_errors
0	1	139772.482528
1	2	93307.182042
2	3	70745.193400
3	4	49446.066485
4	5	42548.525149
5	6	37712.952211
6	7	34124.444666
7	8	31285.935392

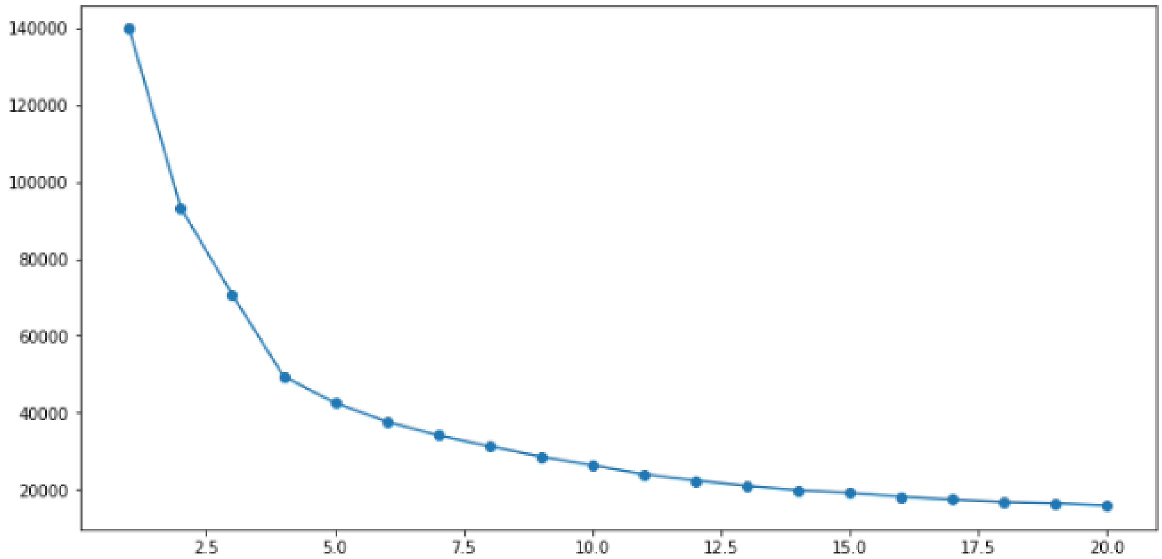
8	9	28601.707202
9	10	26318.569135
10	11	24020.651261
11	12	22363.730765
12	13	21006.553450
13	14	19857.335295
14	15	19184.021099
15	16	18126.843963
16	17	17383.533847
17	18	16782.281831
18	19	16454.483146
19	20	15919.967715

In [88]:

```
# allow plots to appear in the notebook
plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
```

Out [88]:

```
[<matplotlib.lines.Line2D at 0x1bd43ad7248>]
```



From above graph we will find elbow range. here it is 4,5,6

In [89]:

```
from sklearn import metrics
```

In [90]:

```
# calculate SC for K=3 through K=12
k_range = range(2, 21)
scores = []
for k in k_range:
    km = KMeans(n_clusters=k, random_state=1)
    km.fit(reduced_cr)
    scores.append(metrics.silhouette_score(reduced_cr, km.labels_))
```

In [91]:

```
scores
```

Out [91]:

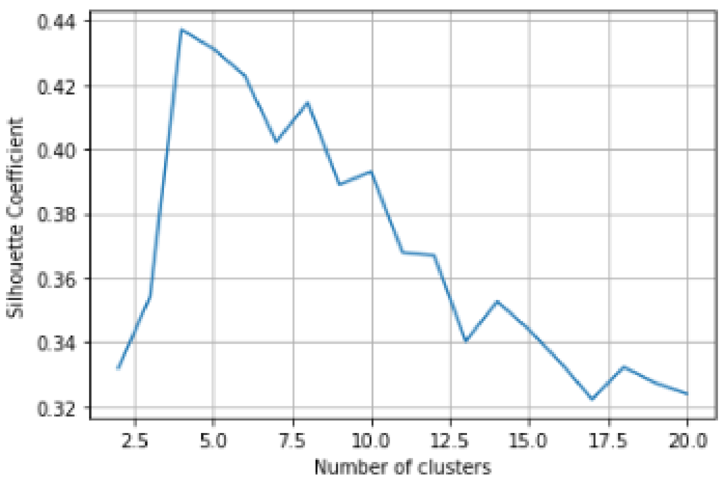
```
[0.3319452179234267,
 0.354031151189225,
 0.43708577439659485,
 0.43121145209717765,
 0.42281449146537453,
 0.4022440826179354,
 0.4144537298622617,
 0.3889240713086915,
 0.392999135547462,
 0.36787983742685676,
 0.3669766371659528,
 0.34034730550080816,
```

```
 0.3526966868656406,
 0.34392337889574587,
 0.3336956707851422,
 0.32231668200851954,
 0.33229025814273944,
 0.3274102528436649,
 0.3239792346891927]
```

In [92]:

```
# plot the results
plt.plot(k_range, scores)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.grid(True)
```

Out [92]:

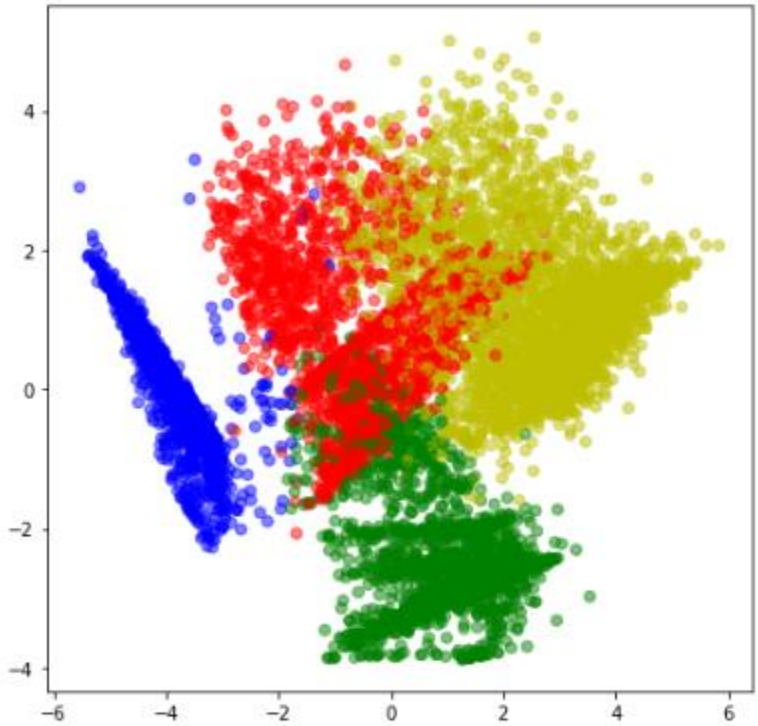


In [93]:

```
#maping cluster
color_map={0:'r',1:'b',2:'g',3:'y'}
label_color=[color_map[l] for l in km_4.labels_]
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=label_color,cmap='Spectral',alpha=0.5
)
```

Out [93]:

```
<matplotlib.collections.PathCollection at 0x1bd3fc94508>
```



It is very difficult to draw individual plot for cluster, so we will use pair plot which will provide us all graph in one shot. To do that we need to take following steps

In [94]:

```
df_pair_plot=pd.DataFrame(reduced_cr,columns=['PC_' +str(i) for i in range(6)])
```

In [95]:

```
df_pair_plot['Cluster']=km_4.labels_#Add cluster column in the data frame
```

In [96]:

```
df_pair_plot.head()
```

Out [96]:

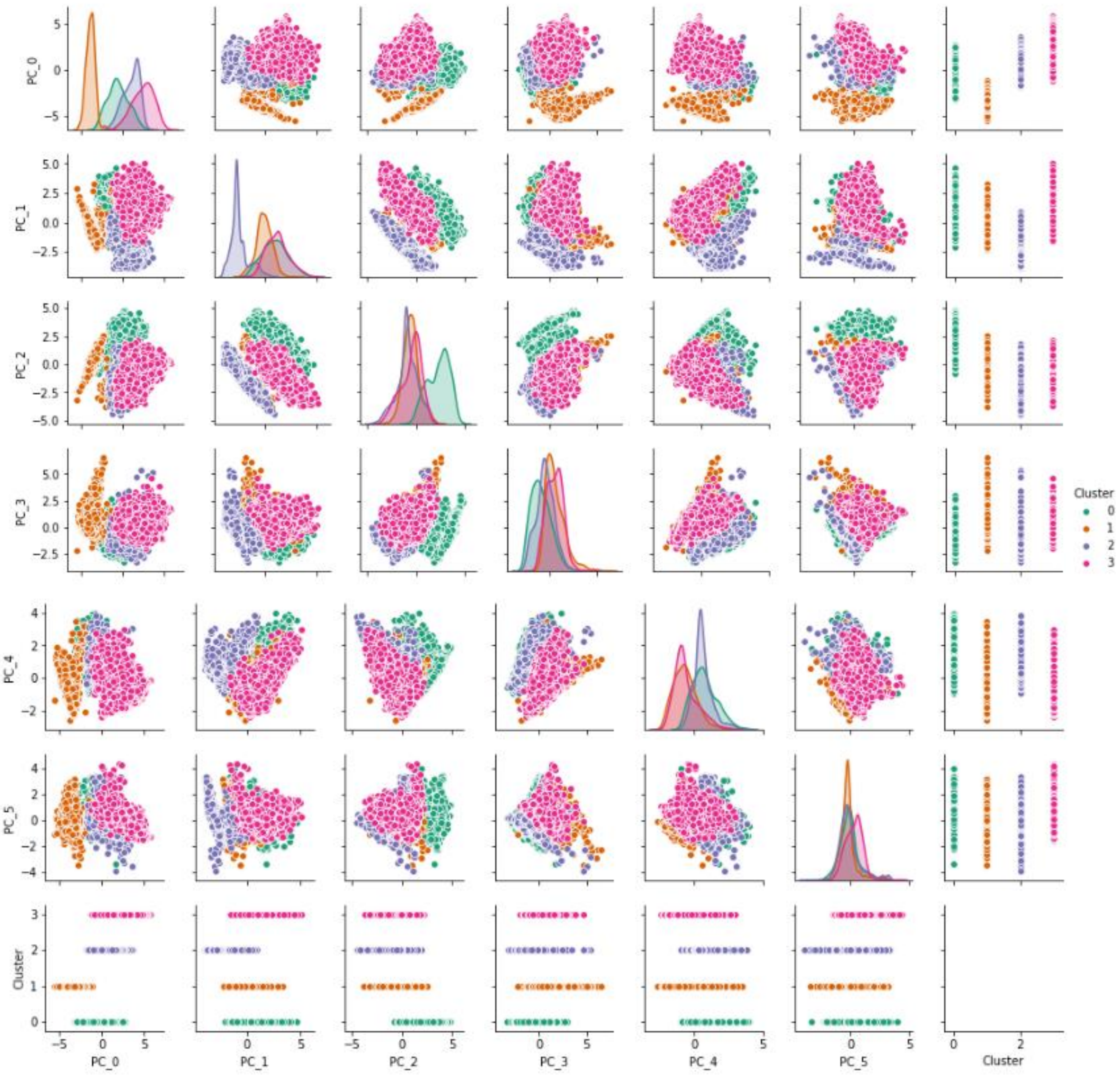
	PC_0	PC_1	PC_2	PC_3	PC_4	PC_5	Cluster
0	-0.242841	-2.759668	0.343061	-0.417359	-0.007100	0.019755	2
1	-3.975652	0.144625	-0.542989	1.023832	-0.428929	-0.572463	1
2	1.287396	1.508938	2.709966	-1.892252	0.010809	-0.599932	0
3	-1.047613	0.673103	2.501794	-1.306784	0.761348	1.408986	0
4	-1.451586	-0.176336	2.286074	-1.624896	-0.561969	-0.675214	0

In [97]:

```
#pairwise relationship of components on the data
sns.pairplot(df_pair_plot,hue='Cluster', palette= 'Dark2', diag_kind='kde',size=1.8
5)
```

Out [97]:

```
<seaborn.axisgrid.PairGrid at 0x1bd440e8e88>
```



It shows that first two components are able to identify clusters

Now we have done here with principle component now we need to come bring our original data frame and we will merge the cluster with them.

To interpreter result we need to use our data frame

In [98]:

```
# Key performace variable selection . here i am taking varibales which we will use
in derving new KPI.

#We can take all 17 variables but it will be difficult to interprate.So we are sele
cting less no of variables.


col_kpi=['PURCHASES_TRX','Monthly_avg_purchase','Monthly_cash_advance','limit_usage
','CASH_ADVANCE_TRX',
```

```
        'payment_minpay','both_oneoff_installment','istallment','one_off','none','
CREDIT_LIMIT']
```

In [99]:

```
cr_pre.describe()
```

Out [99]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PUR
count	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	
mean	0.619940	3.204274	3.352403	0.361268	0.158699	
std	0.148590	3.246365	3.082973	0.277317	0.216672	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.635989	0.000000	0.000000	0.080042	0.000000	
50%	0.693147	3.663562	4.499810	0.405465	0.080042	
75%	0.693147	6.360274	6.151961	0.650588	0.262364	
max	0.693147	10.615512	10.021315	0.693147	0.693147	

In [100]:

```
# Conactenating labels found through Kmeans with data
cluster_df_4=pd.concat([cre_original[col_kpi],pd.Series(km_4.labels_,name='Cluster_
4')],axis=1)
```

In [101]:

```
cluster_df_4.head()
```

Out [101]:

	PURCHASES_TRX	Monthly_avg_purchase	Monthly_cash_advance	limit_usage	CASH_ADVANCE_TRX	payment_minpay	both_oneoff_installment	istallm
0	2	7.950000	0.000000	0.040901	0	1.446508	0	
1	0	0.000000	536.912124	0.457495	4	3.826241	0	
2	12	64.430833	0.000000	0.332687	0	0.991682	0	
3	1	124.916667	17.149001	0.222223	1	0.000000	0	
4	1	1.333333	0.000000	0.681429	0	2.771075	0	

In [102]:

```
# Mean value gives a good indication of the distribution of data. So we are finding
mean value for each variable for each cluster
cluster_4=cluster_df_4.groupby('Cluster_4')\
.apply(lambda x: x[col_kpi].mean()).T
cluster_4
```

Out [102]:

Cluster_4	0	1	2	3
PURCHASES_TRX	7.127341	0.043582	12.062050	33.013723
Monthly_avg_purchase	69.875917	0.148297	47.626256	193.008043
Monthly_cash_advance	78.098613	186.281319	33.550080	67.466910
limit_usage	0.379761	0.576076	0.264745	0.353591
CASH_ADVANCE_TRX	2.881220	6.540230	1.021133	2.804261
payment_minpay	5.573672	9.936617	13.422420	7.245651
both_oneoff_installment	0.000535	0.001916	0.000000	1.000000
istallment	0.000000	0.017241	1.000000	0.000000
one_off	0.999465	0.002874	0.000000	0.000000
none	0.000000	0.977969	0.000000	0.000000
CREDIT_LIMIT	4519.708481	4055.156450	3338.270406	5736.732730

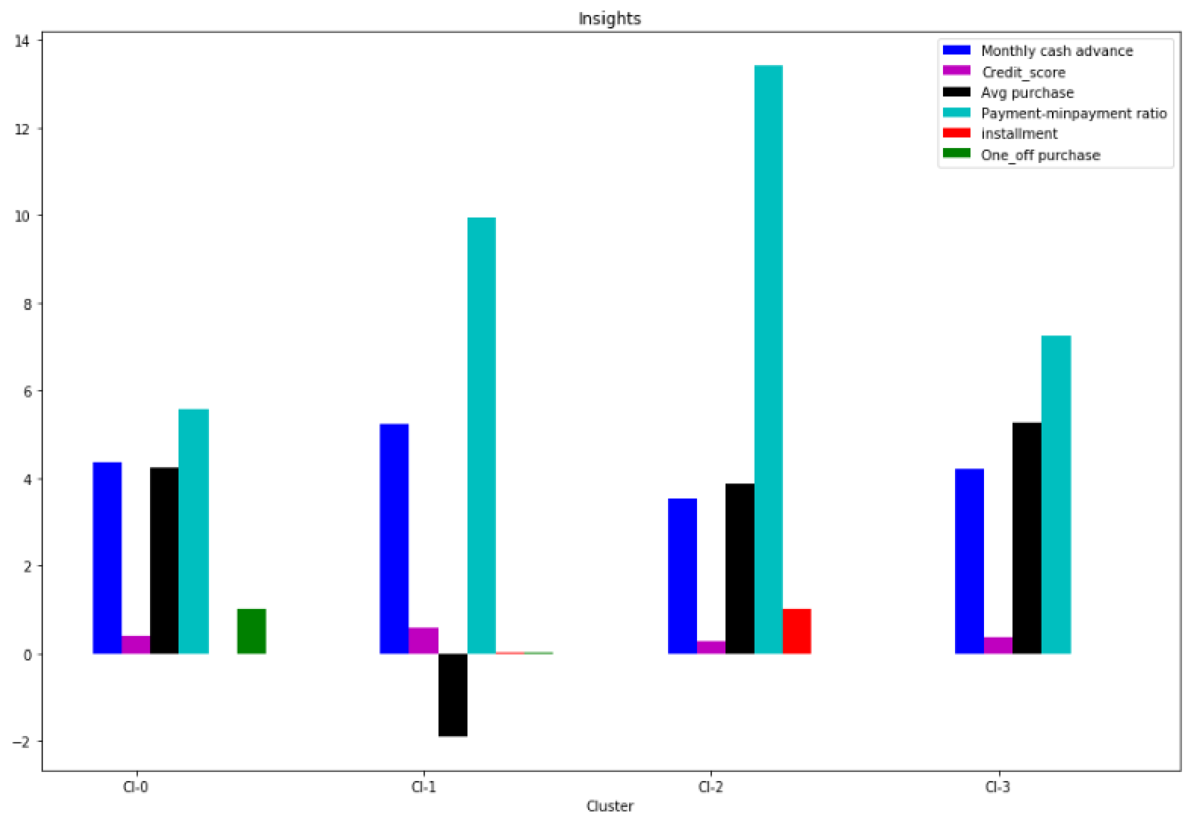
In [103]:

```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(cluster_4.columns))

cash_advance=np.log(cluster_4.loc['Monthly_cash_advance',:].values)
credit_score=(cluster_4.loc['limit_usage',:].values)
purchase= np.log(cluster_4.loc['Monthly_avg_purchase',:].values)
payment=cluster_4.loc['payment_minpay',:].values
installment=cluster_4.loc['istallment',:].values
one_off=cluster_4.loc['one_off',:].values



bar_width=.10
b1=plt.bar(index,cash_advance,color='b',label='Monthly cash advance',width=bar_width)
b2=plt.bar(index+bar_width,credit_score,color='m',label='Credit_score',width=bar_width)
b3=plt.bar(index+2*bar_width,purchase,color='k',label='Avg purchase',width=bar_width)
b4=plt.bar(index+3*bar_width,payment,color='c',label='Payment-minpayment ratio',width=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',width=bar_width)
b6=plt.bar(index+5*bar_width,one_off,color='g',label='One_off purchase',width=bar_width)


plt.xlabel("Cluster")
plt.title("Insights")
plt.xticks(index + bar_width, ('Cl-0', 'Cl-1', 'Cl-2', 'Cl-3'))
```

```
plt.legend()
```

Out [103]:

```
<matplotlib.legend.Legend at 0x1bd00dd3548>
```



Clusters are clearly distinguishing behaviour within customers

---

- Cluster 0 customers are doing maximum One_Off transactions and has least payment ratio amongst all the cluster.
- Cluster 1 is the group of customers who have highest monthly cash advance and doing both instalment as well as one_off purchases, have comparatively good credit score but have poor average purchase score.
- Cluster 2 customers have maximum Average Purchase and good Monthly cash advance but this cluster doesn't do installment or one_off purchases.

- cluster 3 is doing maximum installment, has maximum payment too min_payment ratio and doesn't do one-off purchases

Findings through clustering is validating Insights dervied from KPI. (as shown above in Insights from KPI

In [104]:

```
# Percentage of each cluster in the total customer base
s=cluster_df_4.groupby('Cluster_4').apply(lambda x: x['Cluster_4'].value_counts())
print (s,'\n')

per=pd.Series((s.values.astype('float')/ cluster_df_4.shape[0])*100,name='Percentag
e')
print ("Cluster -4 ",'\n')
print (pd.concat([pd.Series(s.values,name='Size'),per],axis=1),'\n')
```

Out [104]:

```
Cluster_4
0    0   1869
1    1   2088
2    2   2224
3    3   2769
Name: Cluster_4, dtype: int64


Cluster -4
   Size  Percentage
0  1869   20.882682
1  2088   23.329609
2  2224   24.849162
3  2769   30.938547
```

Finding behaviour with 5 Clusters:

In [105]:

```
km_5=KMeans(n_clusters=5,random_state=123)
km_5=km_5.fit(reduced_cr)
km_5.labels_
```

Out [105]:

```
array([4, 2, 0, ..., 4, 2, 0])
```

In [106]:

```
pd.Series(km_5.labels_).value_counts()
```

Out [106]:

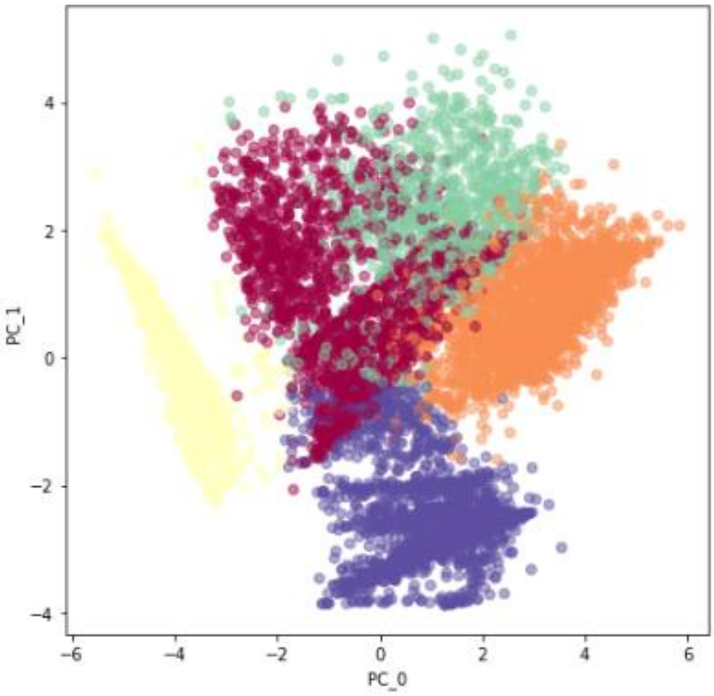
```
4    2149
2    2081
1    1977
0    1862
3     881
dtype: int64
```

In [107]:

```
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=km_5.labels_,cmap='Spectral',alpha=0.5)
plt.xlabel('PC_0')
plt.ylabel('PC_1')
```

Out [107]:

```
Text(0, 0.5, 'PC_1')
```



In [108]:

```
cluster_df_5=pd.concat([cre_original[col_kpi],pd.Series(km_5.labels_,name='Cluster_
5')],axis=1)
```

In [109]:

```
# Finding Mean of features for each cluster
cluster_df_5.groupby('Cluster_5')\
.apply(lambda x: x[col_kpi].mean()).T
```

Out [109]:

Cluster_5	0	1	2	3	4
PURCHASES_TRX	7.096670	34.587759	0.032196	27.703746	11.905537
Monthly_avg_purchase	68.917645	210.536468	0.086126	141.584086	47.369817
Monthly_cash_advance	74.517541	4.040708	185.038534	249.942101	20.636870
limit_usage	0.377959	0.258931	0.576110	0.600096	0.250011
CASH_ADVANCE_TRX	2.697637	0.152757	6.448823	10.384790	0.550489

Cluster_5	0	1	2	3	4
payment_minpay	5.562287	8.675499	9.963172	3.651686	13.783426
both_oneoff_installment	0.002148	1.000000	0.000000	0.900114	0.000000
istallment	0.000000	0.000000	0.015858	0.088536	1.000000
one_off	0.997852	0.000000	0.002883	0.011351	0.000000
none	0.000000	0.000000	0.981259	0.000000	0.000000
CREDIT_LIMIT	4497.951209	5722.970627	4046.692295	5873.041998	3228.949923

With 5 clusters:

- we have a group of customers (cluster 2) having highest average purchases but there is Cluster 4 also having highest cash advance & second highest purchase behaviour but their type of purchases are same.
- Cluster 0 and Cluster 4 are behaving similar in terms of Credit limit and have cash transactions is on higher side

So we don't have quite distinguishable characteristics with 5 clusters,

In [110]:

```
s1=cluster_df_5.groupby('Cluster_5').apply(lambda x: x['Cluster_5'].value_counts())
print (s1)
```

Out [110]:

```
Cluster_5
0          0    1862
1          1    1977
2          2    2081
3          3     881
4          4    2149
Name: Cluster_5, dtype: int64
```

In [111]:

```
# percentage of each cluster


print ("Cluster-5"),'\n'
per_5=pd.Series((s1.values.astype('float')/ cluster_df_5.shape[0])*100,name='Percentage')
print (pd.concat([pd.Series(s1.values,name='Size'),per_5],axis=1))
```

Out [111]:

```
Cluster-5
   Size  Percentage
0  1862   20.804469
1  1977   22.089385
2  2081   23.251397
3   881    9.843575
4  2149   24.011173
```

Finding behaviour with 6 clusters

In [112]:

```
km_6=KMeans(n_clusters=6).fit(reduced_cr)
km_6.labels_
```

Out [112]:

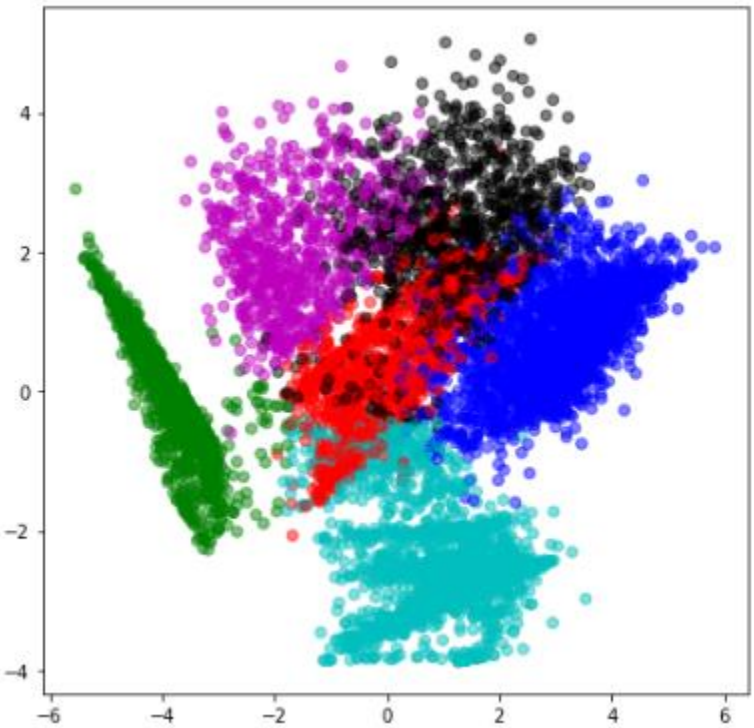
```
array([3, 2, 0, ..., 3, 2, 4])
```

In [113]:

```
color_map={0:'r',1:'b',2:'g',3:'c',4:'m',5:'k'}
label_color=[color_map[l] for l in km_6.labels_]
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=label_color,cmap='Spectral',alpha=0.5
)
```

Out [113]

```
<matplotlib.collections.PathCollection at 0x1bd009f7508>
```



In [114]:

```
cluster_df_6=pd.concat([cre_original[col_kpi],pd.Series(km_6.labels_,name='Cluster_
6')],axis=1)
```

In [115]:

```
six_cluster=cluster_df_6.groupby('Cluster_6').apply(lambda x: x[col_kpi].mean()).T
six_cluster
```

Out [115]:

Cluster_6	0	1	2	3	4	5
PURCHASES_TRX	7.760575	34.652439	0.030347	11.905537	5.967143	27.953143
Monthly_avg_purchase	78.585295	211.137209	0.088891	47.369817	54.091602	140.589204
Monthly_cash_advance	3.603272	4.007851	184.829434	20.636870	205.502536	242.628712
limit_usage	0.245772	0.258170	0.575724	0.250011	0.605930	0.600342
CASH_ADVANCE_TRX	0.125212	0.149898	6.434971	0.550489	7.642857	9.990857
payment_minpay	6.911822	8.706031	9.976487	13.783426	3.257979	3.615910
both_oneoff_installment	0.006768	1.000000	0.000000	0.000000	0.000000	0.912000
istallment	0.000000	0.000000	0.016378	1.000000	0.000000	0.088000
one_off	0.993232	0.000000	0.000000	0.000000	1.000000	0.000000
none	0.000000	0.000000	0.983622	0.000000	0.000000	0.000000
CREDIT_LIMIT	4471.701020	5733.126488	4047.527296	3228.949923	4577.649351	5839.371429

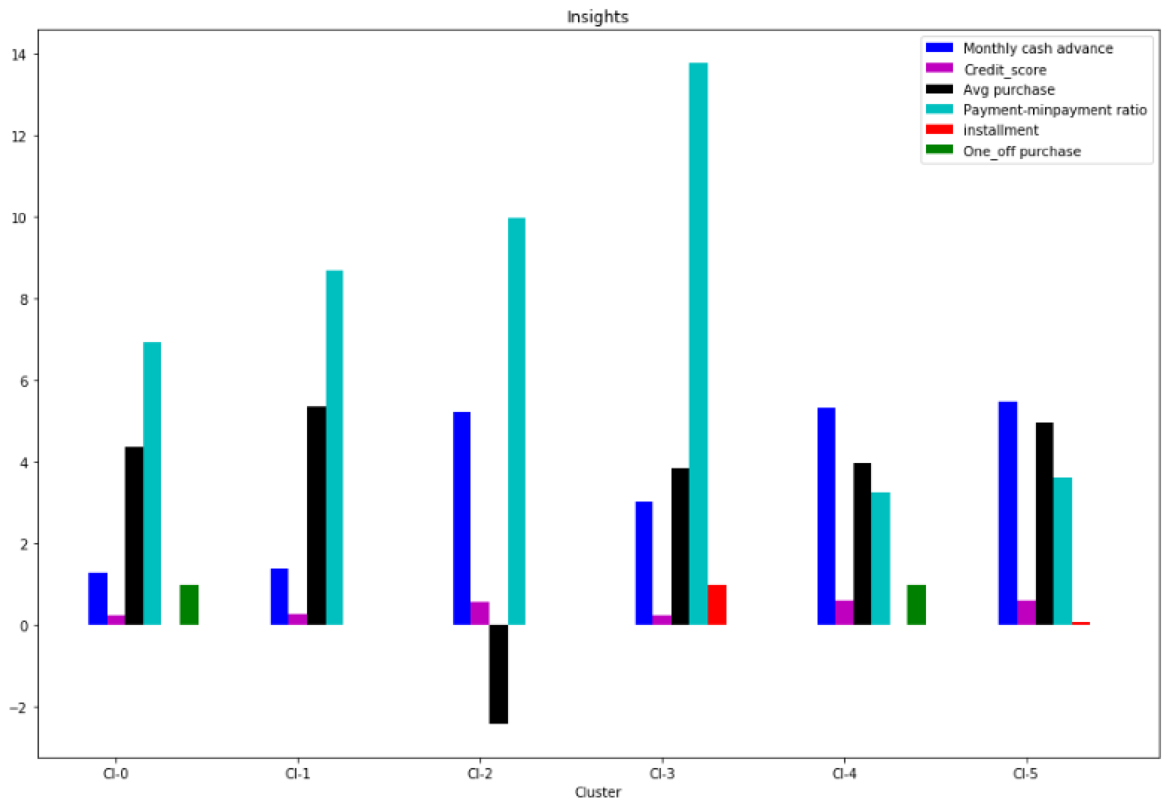
In [116]:

```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(six_cluster.columns))

cash_advance=np.log(six_cluster.loc['Monthly_cash_advance',:].values)
credit_score=(six_cluster.loc['limit_usage',:].values)
purchase= np.log(six_cluster.loc['Monthly_avg_purchase',:].values)
payment=six_cluster.loc['payment_minpay',:].values
installment=six_cluster.loc['istallment',:].values
one_off=six_cluster.loc['one_off',:].values
```

```
bar_width=.10
b1=plt.bar(index,cash_advance,color='b',label='Monthly cash advance',width=bar_widt
h)
b2=plt.bar(index+bar_width,credit_score,color='m',label='Credit_score',width=bar_wi
dth)
b3=plt.bar(index+2*bar_width,purchase,color='k',label='Avg purchase',width=bar_widt
h)
b4=plt.bar(index+3*bar_width,payment,color='c',label='Payment-minpayment ratio',wid
th=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',width=bar_wi
dth)
b6=plt.bar(index+5*bar_width,one_off,color='g',label='One_off purchase',width=bar_w
idth)


plt.xlabel("Cluster")
plt.title("Insights")
plt.xticks(index + bar_width, ('Cl-0', 'Cl-1', 'Cl-2', 'Cl-3','Cl-4','Cl-5'))


plt.legend()
```

Out [116]:

```
<matplotlib.legend.Legend at 0x1bd00a753c8>
```



In [117]:

```
cash_advance=np.log(six_cluster.loc['Monthly_cash_advance',:].values)
credit_score=list(six_cluster.loc['limit_usage',:].values)
cash_advance
```

Out [117]:

```
array ([1.28184245, 1.38825514, 5.21943342, 3.02707927, 5.32545837,
5.49153234])
```

Insights with 6 clusters

- Here also groups are overlapping.

Checking performance metrics for Kmeans

- I am validating performance with 2 metrics Calinski harabaz and Silhouette score

In [118]:

```
from sklearn.metrics import calinski_harabaz_score,silhouette_score
```

In [119]:

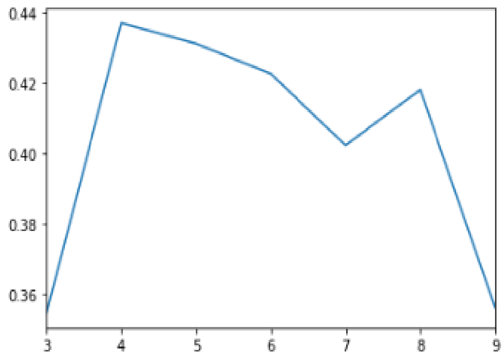
```
#K means algorithm
score={}
score_c={}
for n in range(3,10):
    km_score=KMeans(n_clusters=n)
    km_score.fit(reduced_cr)
    score_c[n]=calinski_harabaz_score(reduced_cr,km_score.labels_)
    score[n]=silhouette_score(reduced_cr,km_score.labels_)
```

In [120]:

```
pd.Series(score).plot()
```

Out [120]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd00cb7088>
```

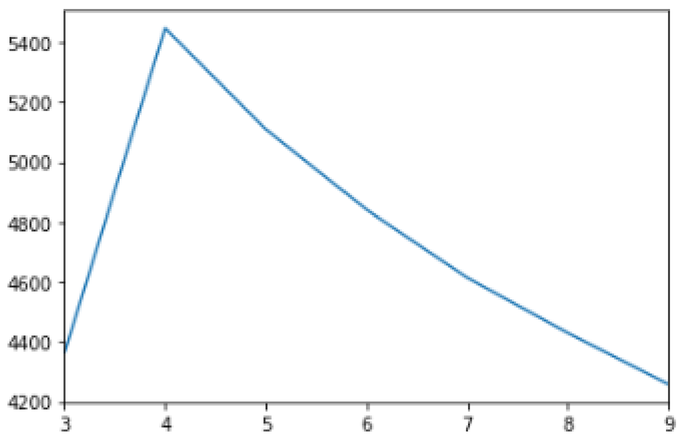


In [121]:

```
pd.Series(score_c).plot()
```

Out [121]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd011b3ec8>
```



Performance metrics also suggest that K-means with 4 clusters is able to show distinguished characteristics of each cluster.

- Cluster 0 customers are doing maximum One_Off transactions and have least payment ratio amongst the entire cluster. Low credit score.

- Cluster 1 is the group of customers who have highest Monthly cash advance and doing both instalment as well as one_off purchases, have comparatively good credit score but have poor average purchase score. poor credit score
- Cluster 2 customers have maximum Average Purchase and good Monthly cash advance but this cluster doesn't do instalment or one_off purchases.and has good credit score.
- cluster 3 is doing maximum instalment, has maximum payment too min_payment ratio and doesn't do one-off purchases. Max credit score

Conclusion with python code:

a. Group 2

- They are potential target customers who are paying dues and doing purchases and maintaining comparatively good credit score we can increase credit limit or can lower down interest rate Can be given premium card /loyalty cards to increase transactions

b. Group 1

- They have poor credit score and taking only cash on advance. We can target them by providing less interest rate on purchase transaction

c. Group 0

- This group is has minimum paying ratio and using card for just one off transactions (may be for utility bills only). This group seems to be risky group.

d. Group 3

- This group is performing best among all as customers are maintaining good credit score and paying dues on time. Giving rewards point will make them perform more purchases.

R CODE:

```
#remove all the objects stored

rm(list=ls())


#set current working directory

setwd("C:/Users/Hp/Desktop/Project")


#Current working directory

getwd()

Output : "C:/Users/Hp/Desktop/Project"


##Load data in R

#reading CSV

credit_card= read.csv("credit_card_data.csv", header = T)


View(credit_card)
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEN
1	C10001	40.900749	0.818182	95.40	0.00	95.40	0.00000	0.166667
2	C10002	3202.467416	0.909091	0.00	0.00	0.00	6442.94548	0.000000
3	C10003	2495.148862	1.000000	773.17	773.17	0.00	0.00000	1.000000
4	C10004	1666.670542	0.636364	1499.00	1499.00	0.00	205.78802	0.083333
5	C10005	817.714335	1.000000	16.00	16.00	0.00	0.00000	0.083333
6	C10006	1809.828751	1.000000	1333.28	0.00	1333.28	0.00000	0.666667
7	C10007	627.260806	1.000000	7091.01	6402.63	688.38	0.00000	1.000000
8	C10008	1823.652743	1.000000	436.20	0.00	436.20	0.00000	1.000000

#Here i am going to remove one variable from given data set

credit = credit_card[,-1]

View(credit)

Output:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEO
1	40.900749	0.818182	95.40	0.00	95.40	0.00000	0.166667	0.0
2	3202.467416	0.909091	0.00	0.00	0.00	6442.94548	0.000000	0.0
3	2495.148862	1.000000	773.17	773.17	0.00	0.00000	1.000000	1.0
4	1666.670542	0.636364	1499.00	1499.00	0.00	205.78802	0.083333	0.0
5	817.714335	1.000000	16.00	16.00	0.00	0.00000	0.083333	0.0
6	1809.828751	1.000000	1333.28	0.00	1333.28	0.00000	0.666667	0.0
7	627.260806	1.000000	7091.01	6402.63	688.38	0.00000	1.000000	1.0
8	1823.652743	1.000000	436.20	0.00	436.20	0.00000	1.000000	0.0

Showing 1 to 10 of 8,950 entries, 17 total columns

dim(credit)

Output: [1] 8950 17

colnames(credit)

Output:

```
> colnames(credit)
 [1] "BALANCE"                  "BALANCE_FREQUENCY"                "PURCHASES"
 [4] "ONEOFF_PURCHASES"   "INSTALLMENTS_PURCHASES"           "CASH_ADVANCE"
 [7] "PURCHASES_FREQUENCY"   "ONEOFF_PURCHASES_FREQUENCY"        "PURCHASES_
INSTALLMENTS_FREQUENCY"
[10] "CASH_ADVANCE_FREQUENCY"  "CASH_ADVANCE_TRX"     "PURCHASES_TRX"
[13] "CREDIT_LIMIT"          "PAYMENTS"      "MINIMUM_PAYMENTS"
[16] "PRC_FULL_PAYMENT"                  "TENURE"
```

str(credit)

Output:

```
> str(credit)
'data.frame':   8950 obs. of  17 variables:
 $ BALANCE                         : num  40.9 3202.5 2495.1 1666.7 817.7
...
 $ BALANCE_FREQUENCY               : num  0.818 0.909 1 0.636 1 ...
 $ PURCHASES                       : num  95.4 0 773.2 1499 16 ...
 $ ONEOFF_PURCHASES                : num  0 0 773 1499 16 ...
 $ INSTALLMENTS_PURCHASES          : num  95.4 0 0 0 0 ...
 $ CASH_ADVANCE                    : num  0 6443 0 206 0 ...
 $ PURCHASES_FREQUENCY             : num  0.1667 0 1 0.0833 0.0833 ...
 $ ONEOFF_PURCHASES_FREQUENCY      : num  0 0 1 0.0833 0.0833 ...
 $ PURCHASES_INSTALLMENTS_FREQUENCY: num  0.0833 0 0 0 0 ...
 $ CASH_ADVANCE_FREQUENCY          : num  0 0.25 0 0.0833 0 ...
 $ CASH_ADVANCE_TRX                : int  0 4 0 1 0 0 0 0 0 0 ...
 $ PURCHASES_TRX                   : int  2 0 12 1 1 8 64 12 5 3 ...
 $ CREDIT_LIMIT                    : num  1000 7000 7500 7500 1200 1800 13
500 2300 7000 11000 ...
 $ PAYMENTS                        : num  202 4103 622 0 678 ...
 $ MINIMUM_PAYMENTS                : num  140 1072 627 NA 245 ...
 $ PRC_FULL_PAYMENT                : num  0 0.222 0 0 0 ...
 $ TENURE                          : int  12 12 12 12 12 12 12 12 12 12 ..
.
```

Identifying Outliers

mystats <- function(x) {

 nmiss<-sum(is.na(x))

 a <- x[!is.na(x)]

 m <- mean(a)

 n <- length(a)

 s <- sd(a)

 min <- min(a)

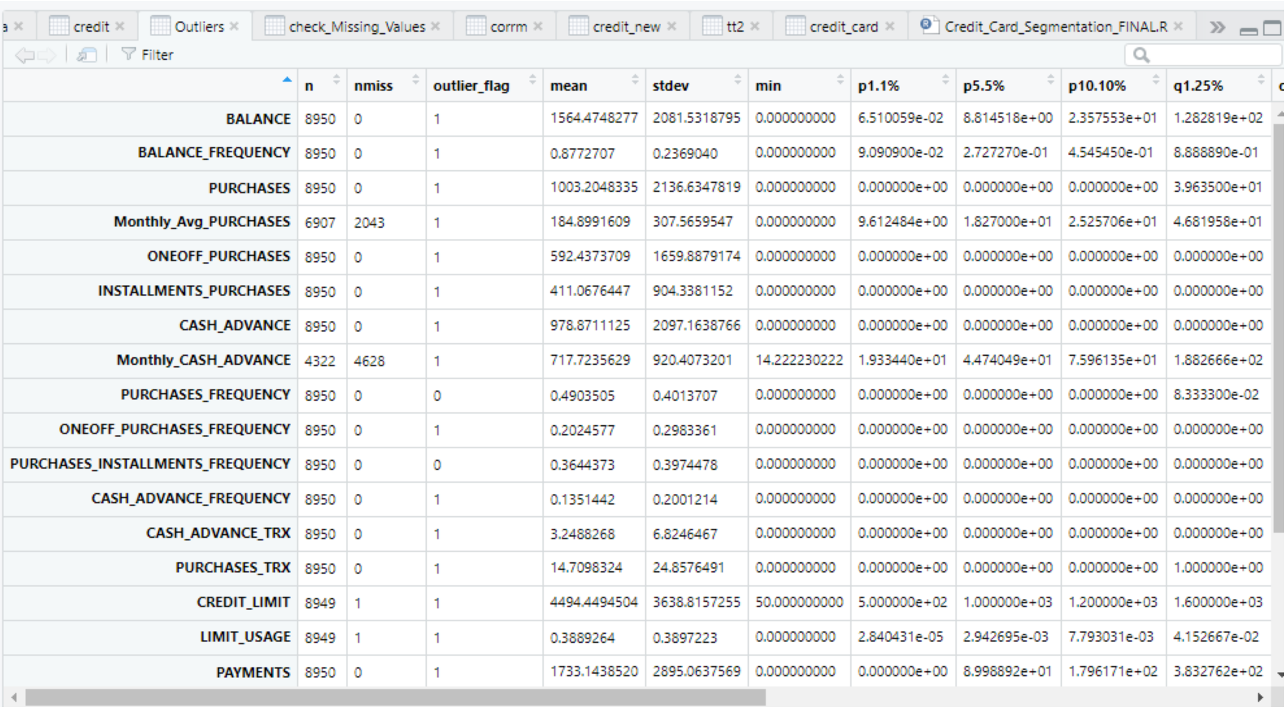
```
  p1<-quantile(a,0.01)

  p5<-quantile(a,0.05)

  p10<-quantile(a,0.10)

  q1<-quantile(a,0.25)

  q2<-quantile(a,0.5)

  q3<-quantile(a,0.75)

  p90<-quantile(a,0.90)

  p95<-quantile(a,0.95)

  p99<-quantile(a,0.99)

  max <- max(a)

  UC <- m+2*s

  LC <- m-2*s

  outlier_flag<- max>UC | min<LC

  return(c(n=n, nmiss=nmiss, outlier_flag=outlier_flag, mean=m, stdev=s,min = min,
p1=p1,p5=p5,p10=p10,q1=q1,q2=q2,q3=q3,p90=p90,p95=p95,p99=p99,max=max, UC=UC,
LC=LC ))

}


#New Variables creation#



credit$Monthly_Avg_PURCHASES <-
credit$PURCHASES/(credit$PURCHASES_FREQUENCY*credit$TENURE)



credit$Monthly_CASH_ADVANCE <-
credit$CASH_ADVANCE/(credit$CASH_ADVANCE_FREQUENCY*credit$TENURE)



credit$LIMIT_USAGE <- credit$BALANCE/credit$CREDIT_LIMIT
```

```
credit$MIN_PAYMENTS_RATIO <- credit$PAYMENTS/credit$MINIMUM_PAYMENTS

write.csv(credit,"New_variables_creation.csv")




Num_Vars <- c(
  "BALANCE",
  "BALANCE_FREQUENCY",
  "PURCHASES",
  "Monthly_Avg_PURCHASES",
  "ONEOFF_PURCHASES",
  "INSTALLMENTS_PURCHASES",
  "CASH_ADVANCE",
  "Monthly_CASH_ADVANCE",
  "PURCHASES_FREQUENCY",
  "ONEOFF_PURCHASES_FREQUENCY",
  "PURCHASES_INSTALLMENTS_FREQUENCY",
  "CASH_ADVANCE_FREQUENCY",
  "CASH_ADVANCE_TRX",
  "PURCHASES_TRX",
  "CREDIT_LIMIT",
  "LIMIT_USAGE",
```

```
 "PAYMENTS",
 "MINIMUM_PAYMENTS",
 "MIN_PAYMENTS_RATIO",
 "PRC_FULL_PAYMENT",
 "TENURE")
```

```
Outliers<-t(data.frame(apply(credit[Num_Vars], 2, mystats)))
View(Outliers)
```

Output:



	n	nmiss	outlier_flag	mean	stdev	min	p1.1%	p5.5%	p10.10%	q1.25%
BALANCE	8950	0	1	1564.4748277	2081.5318795	0.000000000	6.510059e-02	8.814518e+00	2.357553e+01	1.282819e+02
BALANCE_FREQUENCY	8950	0	1	0.8772707	0.2369040	0.000000000	9.090900e-02	2.727270e-01	4.545450e-01	8.888890e-01
PURCHASES	8950	0	1	1003.2048335	2136.6347819	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	3.963500e+01
Monthly_Avg_PURCHASES	6907	2043	1	184.8991609	307.5659547	0.000000000	9.612484e+00	1.827000e+01	2.525706e+01	4.681958e+01
ONEOFF_PURCHASES	8950	0	1	592.4373709	1659.8879174	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
INSTALLMENTS_PURCHASES	8950	0	1	411.0676447	904.3381152	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
CASH_ADVANCE	8950	0	1	978.8711125	2097.1638766	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
Monthly_CASH_ADVANCE	4322	4628	1	717.7235629	920.4073201	14.222230222	1.933440e+01	4.474049e+01	7.596135e+01	1.882666e+02
PURCHASES_FREQUENCY	8950	0	0	0.4903505	0.4013707	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	8.333300e-02
ONEOFF_PURCHASES_FREQUENCY	8950	0	1	0.2024577	0.2983361	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
PURCHASES_INSTALLMENTS_FREQUENCY	8950	0	0	0.3644373	0.3974478	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
CASH_ADVANCE_FREQUENCY	8950	0	1	0.1351442	0.2001214	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
CASH_ADVANCE_TRX	8950	0	1	3.2488268	6.8246467	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
PURCHASES_TRX	8950	0	1	14.7098324	24.8576491	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
CREDIT_LIMIT	8949	1	1	4494.4494504	3638.8157255	50.000000000	5.000000e+02	1.000000e+03	1.200000e+03	1.600000e+03
LIMIT_USAGE	8949	1	1	0.3889264	0.3897223	0.000000000	2.840431e-05	2.942695e-03	7.793031e-03	4.152667e-02
PAYMENTS	8950	0	1	1733.1438520	2895.0637569	0.000000000	0.000000e+00	8.998892e+01	1.796171e+02	3.832762e+02

```
write.csv(Outliers,"Outliers.csv")
```

```
# Outlier Treatment
credit$BALANCE[credit$BALANCE>5727.53]<-5727.53
```

```
credit$BALANCE_FREQUENCY[credit$BALANCE_FREQUENCY>1.3510787]<-1.3510787

credit$PURCHASES[credit$PURCHASES>5276.46]<-5276.46

credit$Monthly_Avg_PURCHASES[credit$Monthly_Avg_PURCHASES>800.03] <- 800.03

credit$ONEOFF_PURCHASES[credit$ONEOFF_PURCHASES>3912.2173709]<-3912.2173709

credit$INSTALLMENTS_PURCHASES[credit$INSTALLMENTS_PURCHASES>2219.7438751]<-2219.7438751

credit$CASH_ADVANCE[credit$CASH_ADVANCE>5173.1911125]<-5173.1911125

credit$Monthly_CASH_ADVANCE[credit$Monthly_CASH_ADVANCE>2558.53] <-2558.53

credit$PURCHASES_FREQUENCY[credit$PURCHASES_FREQUENCY>1.2930919]<-1.2930919

credit$ONEOFF_PURCHASES_FREQUENCY[credit$ONEOFF_PURCHASES_FREQUENCY>0.7991299]<-0.7991299

credit$PURCHASES_INSTALLMENTS_FREQUENCY[credit$PURCHASES_INSTALLMENTS_FREQUENCY>1.1593329]<-1.1593329

credit$CASH_ADVANCE_FREQUENCY[credit$CASH_ADVANCE_FREQUENCY>0.535387]<-0.535387
```

```
credit$CASH_ADVANCE_TRX[credit$CASH_ADVANCE_TRX>16.8981202]<-
16.8981202

credit$PURCHASES_TRX[credit$PURCHASES_TRX>64.4251306]<-64.4251306

credit$CREDIT_LIMIT[credit$CREDIT_LIMIT>11772.09]<-11772.09

credit$LIMIT_USAGE[credit$LIMIT_USAGE>1.1683] <- 1.1683

credit$PAYMENTS[credit$PAYMENTS>7523.26]<-7523.26

credit$MINIMUM_PAYMENTS[credit$MINIMUM_PAYMENTS>5609.1065423]<-
5609.1065423

credit$MIN_PAYMENTS_RATIO[credit$MIN_PAYMENTS_RATIO>249.9239] <-
249.9239

credit$PRC_FULL_PAYMENT[credit$PRC_FULL_PAYMENT>0.738713]<-0.738713

credit$TENURE[credit$TENURE>14.19398]<-14.19398

# Missing Value Imputation with mean

credit$MINIMUM_PAYMENTS[which(is.na(credit$MINIMUM_PAYMENTS))] <-
721.9256368

credit$CREDIT_LIMIT[which(is.na(credit$CREDIT_LIMIT))] <- 4343.62
```

credit$Monthly_Avg_PURCHASES[which(is.na(credit$Monthly_Avg_PURCHASES))] <-184.8991609

credit$Monthly_CASH_ADVANCE[which(is.na(credit$Monthly_CASH_ADVANCE))] <-717.7235629

credit$LIMIT_USAGE[which(is.na(credit$LIMIT_USAGE))] <-0.3889264

credit$MIN_PAYMENTS_RATIO[which(is.na(credit$MIN_PAYMENTS_RATIO))] <-9.3500701

Checking Missing Value

check_Missing_Values<-t(data.frame(apply(credit[Num_Vars], 2, mystats)))

View(check_Missing_Values)

Output:

a × | credit × | Outliers × | check_Missing_Values × | corrm × | credit_new × | tt2 × | credit_card × | Credit_Card_Segmentation_FINAL.R*

Filter

	n	nmiss	outlier_flag	mean	stdev	min	p1.1%	p5.5%	p10.10%	q1.25%	q2
BALANCE	8950	0	1	1442.5552088	1664.1918379	0.000000000	6.510059e-02	8.814518e+00	2.357553e+01	1.282819e+02	
BALANCE_FREQUENCY	8950	0	1	0.8772707	0.2369040	0.000000000	9.090900e-02	2.727270e-01	4.545450e-01	8.888890e-01	
PURCHASES	8950	0	1	872.7142291	1254.6831652	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	3.963500e+01	
Monthly_Avg_PURCHASES	8950	0	1	170.0517713	159.2959982	0.000000000	1.041000e+01	2.015800e+01	2.990826e+01	5.925768e+01	
ONEOFF_PURCHASES	8950	0	1	493.1332258	897.4122344	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
INSTALLMENTS_PURCHASES	8950	0	1	354.9417955	555.0698308	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
CASH_ADVANCE	8950	0	1	841.0403976	1419.4364448	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
Monthly_CASH_ADVANCE	8950	0	1	690.2976420	446.2754218	14.222230222	2.871419e+01	7.798953e+01	1.526895e+02	4.667683e+02	
PURCHASES_FREQUENCY	8950	0	0	0.4903505	0.4013707	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	8.333300e-02	
ONEOFF_PURCHASES_FREQUENCY	8950	0	1	0.1890612	0.2653414	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
PURCHASES_INSTALLMENTS_FREQUENCY	8950	0	0	0.3644373	0.3974478	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
CASH_ADVANCE_FREQUENCY	8950	0	1	0.1248669	0.1697467	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
CASH_ADVANCE_TRX	8950	0	1	2.8153640	4.5417562	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
PURCHASES_TRX	8950	0	1	13.0205305	16.6943873	0.000000000	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	
CREDIT_LIMIT	8950	0	1	4343.6151108	3192.2490957	50.000000000	5.000000e+02	1.000000e+03	1.200000e+03	1.600000e+03	
LIMIT_USAGE	8950	0	1	0.3866509	0.3516907	0.000000000	2.856613e-05	2.942983e-03	7.793267e-03	4.153257e-02	
PAYMENTS	8950	0	1	1536.5469689	1786.0866070	0.000000000	0.000000e+00	8.998892e+01	1.796171e+02	3.832762e+02	

write.csv(credit,"Missing_value_treatment.csv")

Variable Reduction

Step_nums <- credit[Num_Vars]

corrm<- cor(Step_nums)

View(corrm)

Output:

credit | Outliers | check_Missing_Values | corrm | credit_new | tt2 | credit_card | Credit_Card_Segmentation_FINAL.R*

Filter

	BALANCE	BALANCE_FREQUENCY	PURCHASES	Monthly_Avg_PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHA
BALANCE	1.00000000	0.36569111	0.13912924	0.205284064	0.14731598	0.05228897
BALANCE_FREQUENCY	0.36569111	1.00000000	0.18477765	0.036526630	0.14740257	0.16366958
PURCHASES	0.13912924	0.18477765	1.00000000	0.663379014	0.87776651	0.70809096
Monthly_Avg_PURCHASES	0.20528406	0.03652663	0.66337901	1.000000000	0.69619496	0.28422444
ONEOFF_PURCHASES	0.14731598	0.14740257	0.87776651	0.696194956	1.00000000	0.32551776
INSTALLMENTS_PURCHASES	0.05228897	0.16366958	0.70809096	0.284224442	0.32551776	1.00000000
CASH_ADVANCE	0.56409729	0.11657134	-0.12422051	0.059953117	-0.08396514	-0.13573644
Monthly_CASH_ADVANCE	0.17891275	-0.09256103	0.02754061	0.050095258	0.01066144	0.03317611
PURCHASES_FREQUENCY	-0.10381191	0.22971547	0.55880204	-0.064861934	0.37801255	0.61124709
ONEOFF_PURCHASES_FREQUENCY	0.07824756	0.20404551	0.65408938	0.315206449	0.74120285	0.25538413
PURCHASES_INSTALLMENTS_FREQUENCY	-0.09065772	0.17607940	0.44127958	-0.105561259	0.16719968	0.69984378
CASH_ADVANCE_FREQUENCY	0.49288159	0.20034424	-0.18079400	0.029188528	-0.11945465	-0.19585472
CASH_ADVANCE_TRX	0.48408256	0.17511117	-0.13447073	0.032041725	-0.08661369	-0.14933544
PURCHASES_TRX	0.09582788	0.23097507	0.79701169	0.309849858	0.63520613	0.70635922
CREDIT_LIMIT	0.50858084	0.10552850	0.35455596	0.283293448	0.33904492	0.21786957
LIMIT_USAGE	0.61704763	0.44650013	-0.10008734	0.027830702	-0.07434666	-0.10719933
PAYMENTS	0.34560027	0.11836762	0.52371498	0.460421451	0.47219063	0.34717098

write.csv(corrm, "Correlation_matrix.csv")

eigen(corrm)$values

Output:

```
> eigen(corrm)$values
 [1] 5.43628692 4.34186609 2.10299897 1.65342188 1.24504759 1.05006876 0.9
8231813 0.73860862 0.72744967 0.63620389
[11] 0.57783813 0.35278271 0.28990257 0.26560402 0.16088455 0.11783311 0.1
1503442 0.09749872 0.05163199 0.04126943
[21] 0.01544985
```

```
require(dplyr)

eigen_values <- mutate(data.frame(eigen(corrm)$values)
                       ,cum_sum_eigen=cumsum(eigen.corrm..values)
                       , pct_var=eigen.corrm..values/sum(eigen.corrm..values)
                       , cum_pct_var=cum_sum_eigen/sum(eigen.corrm..values))

write.csv(eigen_values, "EigenValues2.csv")

require(psych)
FA<-fa(r=corrm, 7, rotate="varimax", fm="ml")

#SORTING THE LOADINGS
FA_SORT<-fa.sort(FA)
FA_SORT$loadings
```

Output:

```
> FA_SORT$loadings

Loadings:
                                  ML3    ML1    ML2    ML6    ML4    ML7    ML5   
ONEOFF_PURCHASES                   0.950  0.170                             -0.175
PURCHASES                          0.856  0.450                       0.135  0.175
Monthly_Avg_PURCHASES              0.797 -0.163                              0.245
ONEOFF_PURCHASES_FREQUENCY         0.642  0.320                       0.119 -0.342
PAYMENTS                           0.506  0.135  0.285         0.273  0.227  0.149
PURCHASES_FREQUENCY                0.175  0.943 -0.170                      -0.205
PURCHASES_INSTALLMENTS_FREQUENCY          0.905 -0.132                            
INSTALLMENTS_PURCHASES             0.308  0.701                       0.138  0.592
PURCHASES_TRX                      0.541  0.668                       0.114       
CASH_ADVANCE_FREQUENCY                   -0.169  0.934  0.267 -0.116              
CASH_ADVANCE_TRX                         -0.115  0.879  0.236                     
CASH_ADVANCE                             -0.141  0.768  0.206  0.559  0.105       
LIMIT_USAGE                              -0.122  0.169  0.899        -0.172       
BALANCE                            0.164         0.338  0.712  0.190  0.440       
MINIMUM_PAYMENTS                                 0.118  0.625  0.145  0.219  0.109
PRC_FULL_PAYMENT                          0.270 -0.106 -0.480                     
BALANCE_FREQUENCY                  0.112  0.261  0.119  0.478                     
MIN_PAYMENTS_RATIO                 0.164               -0.310                     
Monthly_CASH_ADVANCE                                           0.886              
CREDIT_LIMIT                       0.267         0.172         0.188  0.861       
TENURE                                          -0.149                0.188       

                  ML3   ML1   ML2   ML6   ML4   ML7   ML5
SS loadings     3.511 3.253 2.626 2.450 1.297 1.203 0.695
Proportion Var  0.167 0.155 0.125 0.117 0.062 0.057 0.033
Cumulative Var  0.167 0.322 0.447 0.564 0.626 0.683 0.716
```

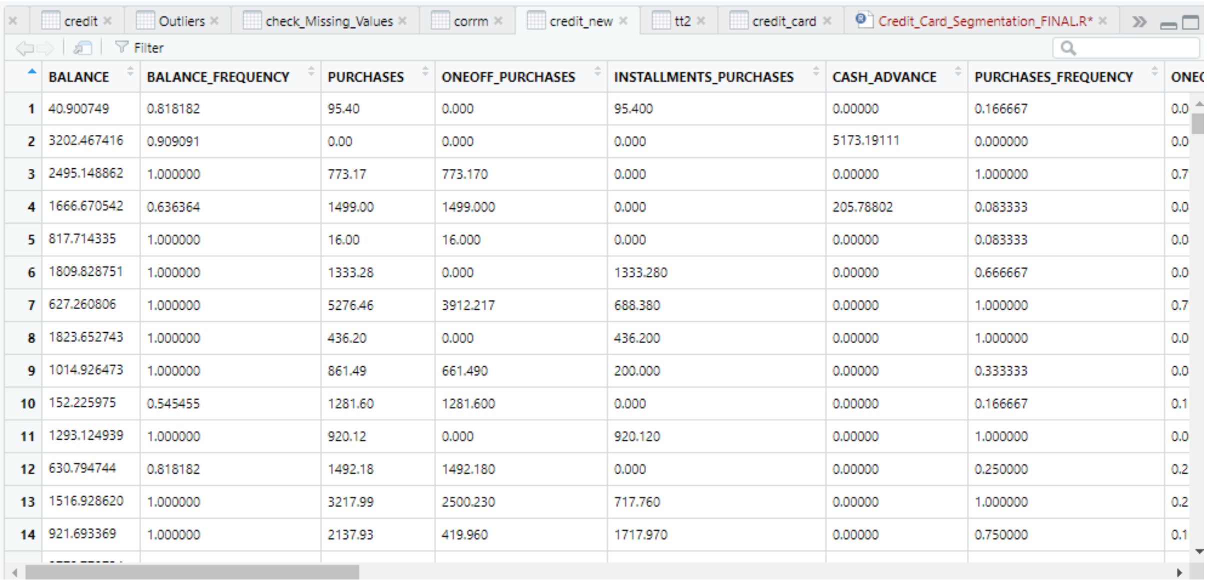
```
Loadings<-data.frame(FA_SORT$loadings[1:ncol(Step_nums),])

write.csv(Loadings, "loadings2.csv")

# standardizing the data

segment_prepared <-credit[Num_Vars]

segment_prepared = scale(segment_prepared)

write.csv(segment_prepared, "standardized data.csv")

#building clusters using k-means clustering

cluster_three <- kmeans(segment_prepared,3)

cluster_four <- kmeans(segment_prepared,4)
```

cluster_five <- kmeans(segment_prepared,5)

cluster_six <- kmeans(segment_prepared,6)

credit_new<-cbind(credit,km_clust_3=cluster_three$cluster,km_clust_4=cluster_four$cluster,km_clust_5=cluster_five$cluster ,km_clust_6=cluster_six$cluster)

View(credit_new)



	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONE(
1	40.900749	0.818182	95.40	0.000	95.400	0.00000	0.166667	0.0
2	3202.467416	0.909091	0.00	0.000	0.000	5173.19111	0.000000	0.0
3	2495.148862	1.000000	773.17	773.170	0.000	0.00000	1.000000	0.7
4	1666.670542	0.636364	1499.00	1499.000	0.000	205.78802	0.083333	0.0
5	817.714335	1.000000	16.00	16.000	0.000	0.00000	0.083333	0.0
6	1809.828751	1.000000	1333.28	0.000	1333.280	0.00000	0.666667	0.0
7	627.260806	1.000000	5276.46	3912.217	688.380	0.00000	1.000000	0.7
8	1823.652743	1.000000	436.20	0.000	436.200	0.00000	1.000000	0.0
9	1014.926473	1.000000	861.49	661.490	200.000	0.00000	0.333333	0.0
10	152.225975	0.545455	1281.60	1281.600	0.000	0.00000	0.166667	0.1
11	1293.124939	1.000000	920.12	0.000	920.120	0.00000	1.000000	0.0
12	630.794744	0.818182	1492.18	1492.180	0.000	0.00000	0.250000	0.2
13	1516.928620	1.000000	3217.99	2500.230	717.760	0.00000	1.000000	0.2
14	921.693369	1.000000	2137.93	419.960	1717.970	0.00000	0.750000	0.1

Profiling

Num_Vars2 <- c(

"Monthly_Avg_PURCHASES",

"Monthly_CASH_ADVANCE",

"CASH_ADVANCE",

"CASH_ADVANCE_TRX",

```
  "CASH_ADVANCE_FREQUENCY",
  "ONEOFF_PURCHASES",
  "ONEOFF_PURCHASES_FREQUENCY",
  "PAYMENTS",
  "CREDIT_LIMIT",
  "LIMIT_USAGE",
  "PURCHASES_INSTALLMENTS_FREQUENCY",
  "PURCHASES_FREQUENCY",
  "INSTALLMENTS_PURCHASES",
  "PURCHASES_TRX",
  "MINIMUM_PAYMENTS",
  "MIN_PAYMENTS_RATIO",
  "BALANCE",
  "TENURE"
)

require(tables)

tt <-cbind(tabular(1+factor(km_clust_3)+factor(km_clust_4)+factor(km_clust_5)+
            factor(km_clust_6)~Heading()*length*All(credit[1]),

data=credit_new),tabular(1+factor(km_clust_3)+factor(km_clust_4)+factor(km_clust_5)+
                        factor(km_clust_6)~Heading()*mean*All(credit[Num_Vars2]),
                       data=credit_new))

tt2 <- as.data.frame.matrix(tt)
```

View(tt2)

Output:

credit | Outliers | check_Missing_Values | corrm | credit_new | tt2 | credit_card | Credit_Card_Segmentation_FINAL.R*

Filter

	term	term	term	term	term	term	term	term	term	term
1	8950	170.051771304778	690.297641971682	841.040397647654	2.81536404844693	0.124866906145251	493.133225761475	0.189061207597765	1536.54696892212	4343.6
2	1383	353.901229788287	695.534594415182	490.230849467462	1.65424651352133	0.0705564728850325	2010.61329758872	0.561581073608098	3397.55586193565	6848.7
3	5152	121.659613390821	629.198631121988	174.635346140431	0.692138610287267	0.0430642909549689	237.572843555901	0.137514588858696	813.256836640528	3252.4
4	2415	168.003157014928	817.643148666615	2463.60289474845	8.00984976612837	0.330481151552795	169.312169772257	0.0856960751138716	2013.82045234948	5236.7
5	3440	169.564574802188	552.949795510582	477.843592319767	2.09897982587209	0.117657979069767	212.074148255814	0.0989066452906977	854.191998103779	3039.3
6	1498	167.721718192267	1028.96146719628	3346.88660426135	9.84945363538051	0.371517876502003	219.191713865754	0.107792131909212	2642.93666038652	6599.5
7	2954	87.7399933163074	674.146581621281	105.307325729519	0.46716316858497	0.0265209390656737	285.675934326337	0.184652642924848	943.743618277251	3671.0
8	1058	404.754387287188	702.460604083986	528.181122852552	1.74152749224953	0.073666095463138	2374.07505027817	0.609567511720227	3843.79691183932	7267.7
9	2668	164.906712226793	471.196416804296	619.798051806597	3.18202322586207	0.168567776611694	200.150307346327	0.102105820989505	846.286040157047	2869.8
10	960	422.834823266937	703.623979945003	547.235676257813	1.82253954854167	0.075792178125	2506.25895124406	0.624953947604167	3996.80443089167	7396.9
11	2627	95.6560643788253	669.97461731201	121.271222968786	0.521468641111534	0.0293390677578987	344.149836315188	0.214531648915112	1036.56955823373	3817.6
12	1441	148.654548345053	742.386591631763	299.1757572127	0.683411686606523	0.0356514469118668	195.660055517002	0.080673827480916	927.001493323387	3641.4
13	1254	167.91956087083	1128.97164006816	3667.18687291587	10.0506682135566	0.372098903508772	229.270332831659	0.11156145199362	2869.5331592496	7050.2
14	533	188.28880024693	1936.64897780362	3972.84930415854	6.01677974859287	0.23452309380863	165.62800632439	0.0614642786116323	3227.29842991182	6845.1

rownames(tt2)<-c(

"ALL",

"KM3_1",

"KM3_2",

"KM3_3",

"KM4_1",

"KM4_2",

"KM4_3",

"KM4_4",

"KM5_1",

"KM5_2",

"KM5_3",

"KM5_4",

"KM5_5",

"KM6_1",

```
  "KM6_2",
  "KM6_3",
  "KM6_4",
  "KM6_5",
  "KM6_6")


colnames(tt2)<-c(
  "SEGMENT_SIZE",
  "Monthly_Avg_PURCHASES",
  "Monthly_CASH_ADVANCE",
  "CASH_ADVANCE",
  "CASH_ADVANCE_TRX",
  "CASH_ADVANCE_FREQUENCY",
  "ONEOFF_PURCHASES",
  "ONEOFF_PURCHASES_FREQUENCY",
  "PAYMENTS",
  "CREDIT_LIMIT",
  "LIMIT_USAGE",
  "PURCHASES_INSTALLMENTS_FREQUENCY",
  "PURCHASES_FREQUENCY",
  "INSTALLMENTS_PURCHASES",
  "PURCHASES_TRX",
  "MINIMUM_PAYMENTS",
  "MIN_PAYMENTS_RATIO",
```

"BALANCE",

"TENURE"

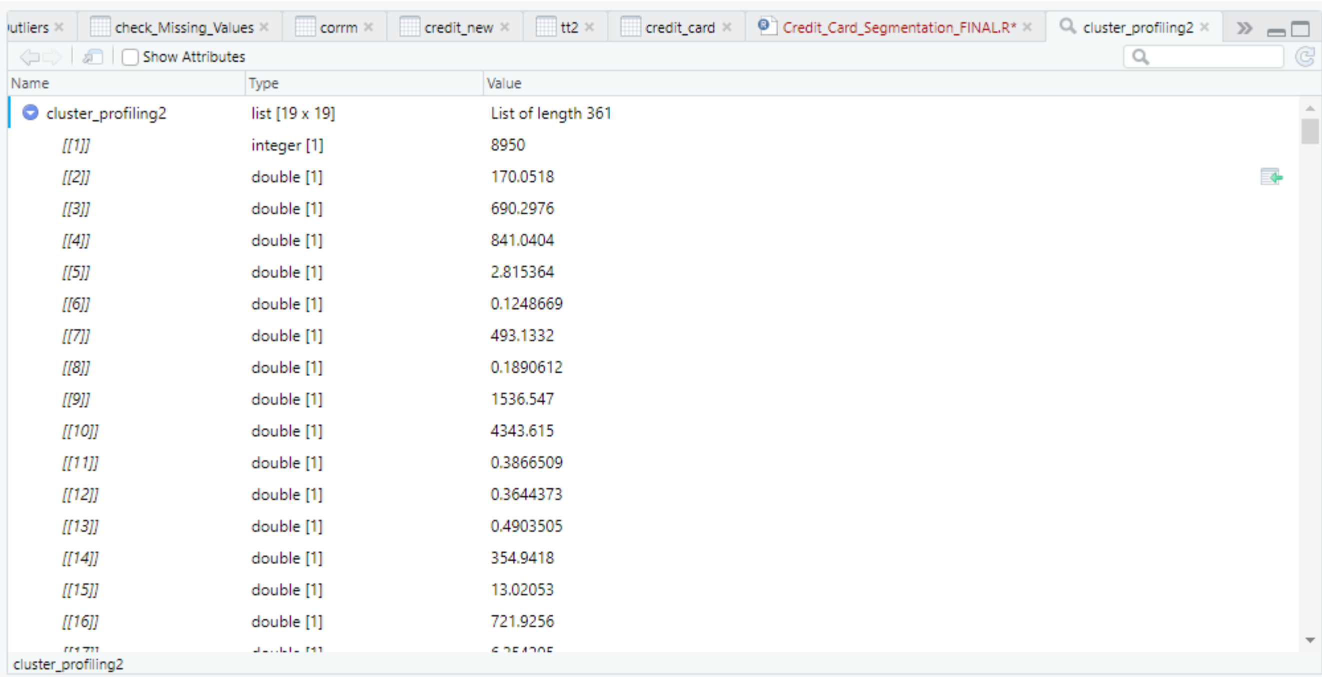
)

cluster_profiling2 <- t(tt2)

write.csv(cluster_profiling2,'cluster_profiling2.csv')

view(cluster_profiling2)

Output:



Conclusions with R code:

From this overall data analysis,

- I did data pre-processing as per my knowledge and that's gives us to clear picture of data and as helps in my future analysis.
- Data cleaning of given raw data to know what is there in our data. Overall, incorrect data is either removed, corrected, or imputed.
- Identifying Outlier effect and I have reduced that effect with creating new variables. Outliers are those data points that lie outside the overall pattern of distribution.
- Identifying missing values in our data and I have done missing value treatment with mean among from all other techniques.
- Then I have done variable reduction i.e. factor analysis
- Then I went through standardizing the data, to put all data in one scale using PCA.
- The given data is unsupervised data so I am going analyse through K means algorithm (clusters)

- Here from K means algorithm I take up to 6 clusters.

- After that finally I did Data profiling that is the process of examining the data available from an existing information source and collecting statistics or informative summaries about that data.

Importance of my project:

- From going through like this we can easily understand given raw data.

- This step of coding is simple coding. So we can easily make changes as per our need.

- This project is gone through simple to complex. So we can easily understand the code and we have to run easily.

- We can use the same codes in large another business data set.

- We can easily detect the errors in simple manner.

END PROJECT REPORT