CREDIT CARD SEGMENTATION

By

Jerin Joseph

Project Name – Credit Card Segmentation

Deadline - 15 Days

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.

Expectations from the student:

1. Advanced data preparation. Build an ‘enriched’ customer profile by deriving ‘intelligent’ KPI’s 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.
2. Advanced reporting. Use the derived KPI’s to gain insight on the customer profiles.
3. 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

Data Set:

credit-card-data.csv

Number of attributes:

- 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 (percentage of months with at least on purchase)
- ONEOFF_PURCHASES_FREQUENCY: Frequency of one-off-purchases
- PURCHASES_INSTALLMENTS_FREQUENCY: Frequency of installment 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_PAYMENT: Percentage of months with full payment of the due statement balance
- TENURE: Number of months as a customer

Overview

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

Let’s understand this with an example. Suppose, we are the head of a rental store and wish to understand preferences of our customers to scale up our business. Is it possible for us to look at details of each costumer and devise a unique business strategy for each one of them? Definitely not. But what we can do is to cluster all of our customers into say 10 groups based on their purchasing habits and use a separate strategy for costumers in each of these 10 groups. And this is what we call clustering.

Types of clustering algorithms:

Since the task of clustering is subjective, the means that can be used for achieving this goal are plenty. Every methodology follows a different set of rules for defining the 'similarity' among data points. In fact, there are more than 100 clustering algorithms known. But few of the algorithms are used popularly, let's look at them in detail:

Connectivity models: As the name suggests, these models are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away. These models can follow two approaches. In the first approach, they start with classifying all data points into separate clusters & then aggregating them as the distance decreases. In the second approach, all data points are classified as a single cluster and then partitioned as the distance increases. Also, the choice of distance function is subjective. These models are very easy to interpret but lacks scalability for handling big datasets. Examples of these models are hierarchical clustering algorithm and its variants.

Centroid models: These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. K-Means clustering algorithm is a popular algorithm that falls into this category. In these models, the no. of clusters required at the end have to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run iteratively to find the local optima.

Distribution models: These clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution (For example: Normal, Gaussian). These models often suffer from overfitting. A popular example of these models is Expectation-maximization algorithm which uses multivariate normal distributions.

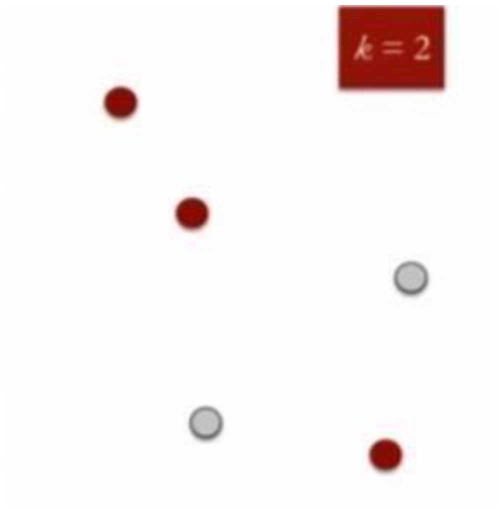
Density Models: These models search the data space for areas of varied density of data points in the data space. It isolates various different density regions and assign the data points within these regions in the same cluster. Popular examples of density models are DBSCAN and OPTICS.

Now I will be taking you through the most popular clustering algorithms in detail – K Means clustering

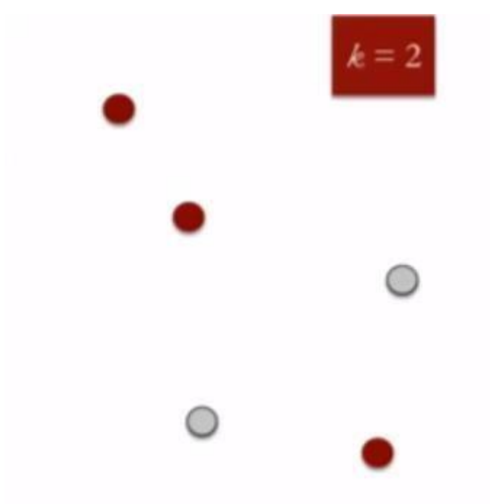
K Means Clustering

K means is an iterative clustering algorithm that aims to find local maxima in each iteration. This algorithm works in these 5 steps:

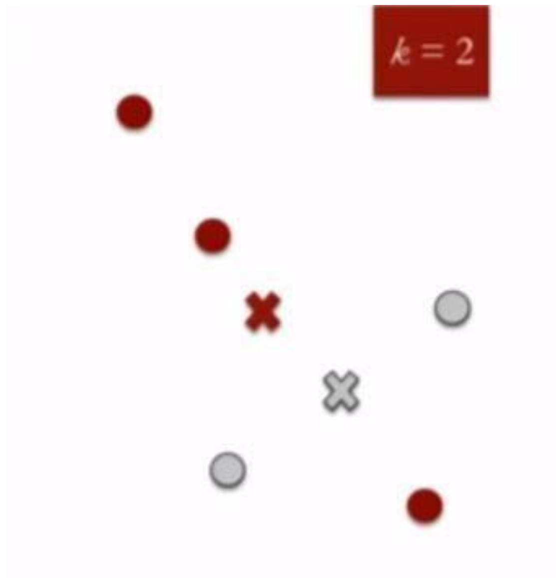
1. Specify the desired number of clusters K: Let us choose k=2 for these 5 data points in 2-D space.



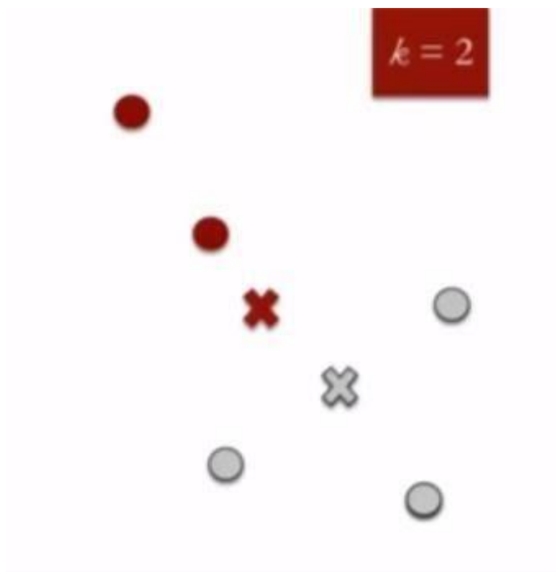
2. Randomly assign each data point to a cluster: Let's assign three points in cluster 1 shown using red colour and two points in cluster 2 shown using grey colour.



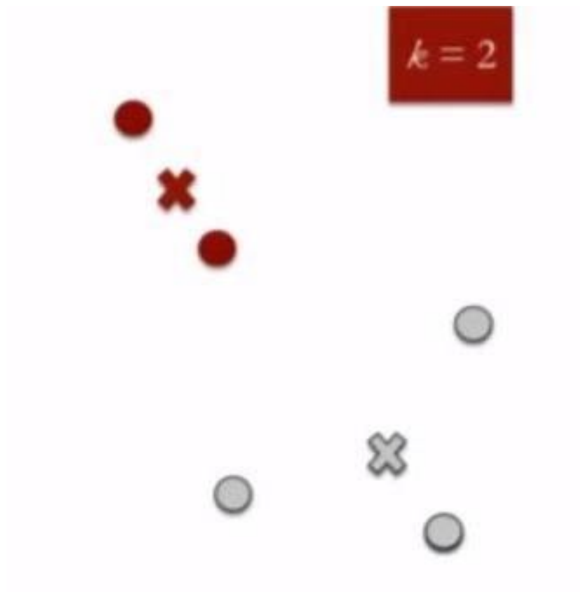
3. Compute cluster centroids: The centroid of data points in the red cluster is shown using red cross and those in grey cluster using grey cross.



4. Re-assign each point to the closest cluster centroid: Note that only the data point at the bottom is assigned to the red cluster even though its closer to the centroid of grey cluster. Thus, we assign that data point into grey cluster



5. Re-compute cluster centroids: Now, re-computing the centroids for both the clusters.



6. Repeat steps 4 and 5 until no improvements are possible: Similarly, we’ll repeat the 4th and 5th steps until we’ll reach global optima. When there will be no further switching of data points between two clusters for two successive repeats. It will mark the termination of the algorithm if not explicitly mentioned.

PREVIEW OF OUR PROJECT:

From the problem statement and the input attributes we can clearly understand that this belongs to unsupervised machine learning model in which there will be no target label, every attribute has to be considered as input feature, we have to find the hidden patterns among these features and establish the hidden patterns.

We intend to segment the customer who are using credit cards, by using K Mean model as it a clustering project and comes under unsupervised learning. We will analyse the customer insights and derive the KPI’s which would enable the organization to focus on the key areas. To start with, we will be using Python and later on R.

Business Problem: Credit Card Segmentation

LOAD THE DATA

```
import os
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

```
#set working directory

path = "C:/Users/jerin/Desktop/PYTHON WORK/PYTHON PROJECT/EDWISOR PROJECTS/CREDIT CARD SEGMENTATION"
os.chdir(path)
os.getcwd()

credit = pd.read_csv("CC GENERAL.csv")
```

```
In [5]: credit.head()
```

Out[5]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUEI
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

```
credit.info()
```

```
In [6]: credit.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
```

```
# Initial descriptive analysis of data.
credit.describe().T
```

```
In [7]: # Intital descriptive analysis of data.
        credit.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
BALANCE	8950.0	1564.474828	2081.531879	0.000000	128.281915	873.385231	2054.140036	19043.13856
BALANCE_FREQUENCY	8950.0	0.877271	0.236904	0.000000	0.888889	1.000000	1.000000	1.00000
PURCHASES	8950.0	1003.204834	2136.634782	0.000000	39.635000	361.280000	1110.130000	49039.57000
ONEOFF_PURCHASES	8950.0	592.437371	1659.887917	0.000000	0.000000	38.000000	577.405000	40761.25000
INSTALLMENTS_PURCHASES	8950.0	411.067645	904.338115	0.000000	0.000000	89.000000	468.637500	22500.00000
CASH_ADVANCE	8950.0	978.871112	2097.163877	0.000000	0.000000	0.000000	1113.821139	47137.21176
PURCHASES_FREQUENCY	8950.0	0.490351	0.401371	0.000000	0.083333	0.500000	0.916667	1.00000
ONEOFF_PURCHASES_FREQUENCY	8950.0	0.202458	0.298336	0.000000	0.000000	0.083333	0.300000	1.00000
PURCHASES_INSTALLMENTS_FREQUENCY	8950.0	0.364437	0.397448	0.000000	0.000000	0.166667	0.750000	1.00000
CASH_ADVANCE_FREQUENCY	8950.0	0.135144	0.200121	0.000000	0.000000	0.000000	0.222222	1.50000
CASH_ADVANCE_TRX	8950.0	3.248827	6.824647	0.000000	0.000000	0.000000	4.000000	123.00000
PURCHASES_TRX	8950.0	14.709832	24.857649	0.000000	1.000000	7.000000	17.000000	358.00000
CREDIT_LIMIT	8949.0	4494.449450	3638.815725	50.000000	1600.000000	3000.000000	6500.000000	30000.00000
PAYMENTS	8950.0	1733.143852	2895.063757	0.000000	383.276166	856.901546	1901.134317	50721.48336
MINIMUM_PAYMENTS	8637.0	864.206542	2372.446607	0.019163	169.123707	312.343947	825.485459	76406.20752
PRC_FULL_PAYMENT	8950.0	0.153715	0.292499	0.000000	0.000000	0.000000	0.142857	1.00000
TENURE	8950.0	11.517318	1.338331	6.000000	12.000000	12.000000	12.000000	12.00000

MISSING VALUE ANALYSIS

```
# finding missing values

credit.isnull().sum()
```

```
Out[9]: 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                         1
        PAYMENTS                             0
        MINIMUM_PAYMENTS                   313
        PRC_FULL_PAYMENT                     0
        TENURE                               0
        dtype: int64
```

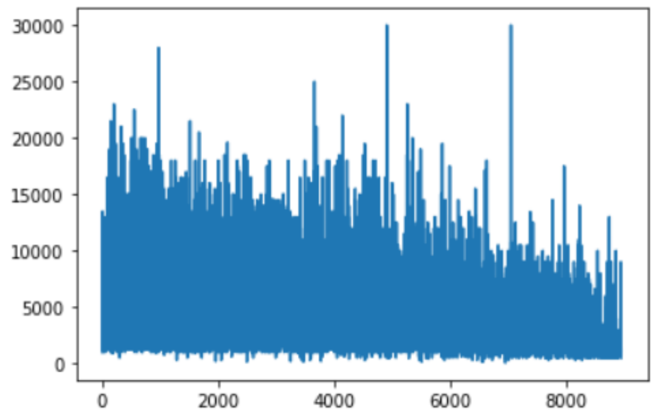
Observation

- there are missing values in the data so we will have to treat them accordingly

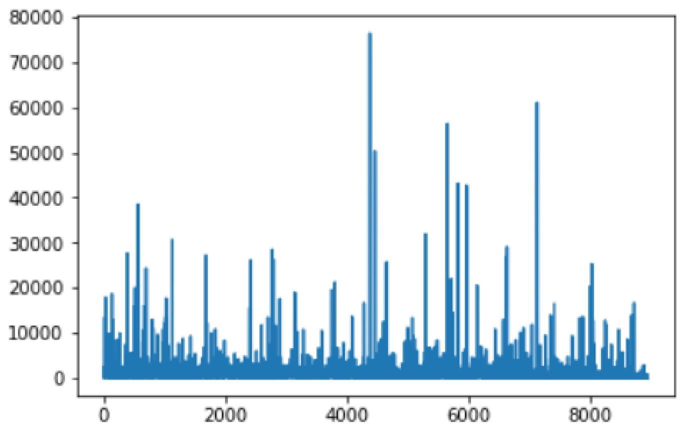
```
credit['CREDIT_LIMIT'].describe()
```

```
Out[10]: 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 [11]:
```
plt.plot(credit['CREDIT_LIMIT'])
```
Out[11]: [<matplotlib.lines.Line2D at 0x187514f4e48>]



In [12]:
```
plt.plot(credit['MINIMUM_PAYMENTS'])
```
Out[12]: [<matplotlib.lines.Line2D at 0x18751d3b7c8>]



Observation

- From the graph we can see that there are some outlier data in the distribution of columns "CREDIT_LIMIT" and "MINIMUM_PAYMENTS" and also, we don't want any data to be lost in this dataset and hence we will fill the null values with median imputation rather than mean imputation.

- This is because mean can't give the measure of central tendency if there is any outlier data available in the data distribution.

```
# imputing missing values with median

credit['CREDIT_LIMIT'].fillna(credit['CREDIT_LIMIT'].median(),inplace=True)
credit['MINIMUM_PAYMENTS'].fillna(credit['MINIMUM_PAYMENTS'].median(),inplace=True)
credit.isnull().sum()
```

```
Out[13]: 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
```

Deriving Key Performance Indicators (KPI)

1. Monthly average purchase and cash advance amount

```
credit['Monthly_avg_purchase']=credit['PURCHASES']/credit['TENURE']
credit['Monthly_cash_advance']=credit['CASH_ADVANCE']/credit['TENURE']
```

```
In [15]: credit['Monthly_avg_purchase'].head()

Out[15]: 0      7.950000
         1      0.000000
         2     64.430833
         3    124.916667
         4      1.333333
         Name: Monthly_avg_purchase, dtype: float64
```

```
In [17]: credit['Monthly_cash_advance'].head()

Out[17]: 0      0.000000
         1    536.912124
         2      0.000000
         3     17.149001
         4      0.000000
         Name: Monthly_cash_advance, dtype: float64
```

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

Out: 4302
```

2. Purchase_type

- To find what type of purchases customers are making on credit card, let's explore the data.

```
credit.loc[:,['ONEOFF_PURCHASES','INSTALLMENTS_PURCHASES']].head(20)
```

Out[18]:

	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 [23]: credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
Out[23]: (2042, 20)
```

```
In [24]: credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
Out[24]: (2774, 20)

In [25]: credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']==0)].shape
Out[25]: (1874, 20)

In [26]: credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']>0)].shape
Out[26]: (2260, 20)
```

Observations:

We can see that there are four types of customers in the entire dataset, they are

- Customers who do only one-off purchase transactions
- Customers who do only installment purchase transaction
- Customers who do both one-off purchase and installment purchase transactions
- Customers who neither do one-off purchase transactions nor installment purchase transactions.

So, deriving a categorical variable based on the behavior.

```
def purchase(credit):
    if (credit['ONEOFF_PURCHASES']==0) &
(credit['INSTALLMENTS_PURCHASES']==0):
        return 'none'
    if (credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0):
         return 'both one_off & installment'
    if (credit['ONEOFF_PURCHASES']>0) &
(credit['INSTALLMENTS_PURCHASES']==0):
        return 'one_off'
    if (credit['ONEOFF_PURCHASES']==0) &
(credit['INSTALLMENTS_PURCHASES']>0):
        return 'installment'
```

```
In [28]: credit['purchase_type']=credit.apply(purchase,axis=1)
```

```
In [29]: credit['purchase_type'].value_counts()

Out[29]: both one_off & installment    2774
         installment                  2260
         none                         2042
         one_off                      1874
         Name: purchase_type, dtype: int64
```

3. Limit_Usage (balance to credit limit ratio)

- Lower value implies customers are maintaining their balance properly. Lower value means good credit score

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

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

```
In [27]: credit['limit_usage'].head()

Out[27]: 0    0.040901
         1    0.457495
         2    0.332687
         3    0.222223
         4    0.681429
         Name: limit_usage, dtype: float64
```

3. Payment to minimum payments Ratio

```
In [28]: credit['payment_minpay']=credit.apply(lambda x:x['PAYMENTS']/x['MINIMUM_PAYMENTS'],axis=1)
         credit['payment_minpay'].head()

Out[28]: 0    1.446508
         1    3.826241
         2    0.991682
         3    0.000000
         4    2.771075
         Name: payment_minpay, dtype: float64
```

```
credit.info()
```

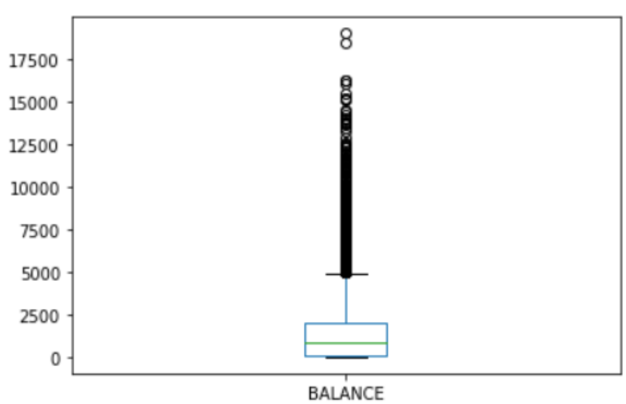
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 23 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   CUST_ID                           8950 non-null   object 
 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                      8950 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8950 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64  
 18  Monthly_avg_purchase              8950 non-null   float64
 19  Monthly_cash_advance              8950 non-null   float64
 20  purchase_type                     8950 non-null   object 
 21  limit_usage                       8950 non-null   float64
 22  payment_minpay                    8950 non-null   float64
dtypes: float64(18), int64(3), object(2)
memory usage: 1.6+ MB
```

OUTLIER ANALYSIS

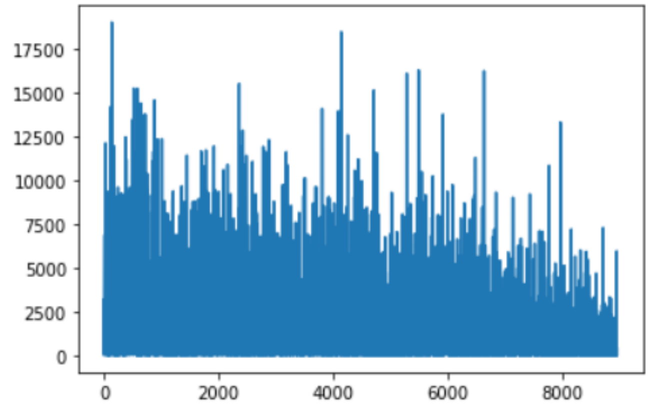
```
##make the function to check the outlier

def boxplot(value):
    return value.plot.box()
```

```
In [36]: boxplot(credit['BALANCE']) ##We can see there are many outlier
```

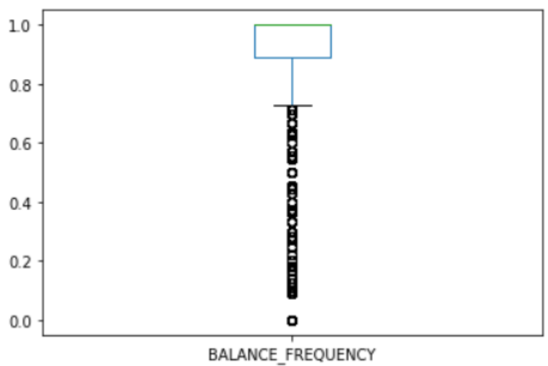
```
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x11938313348>
```



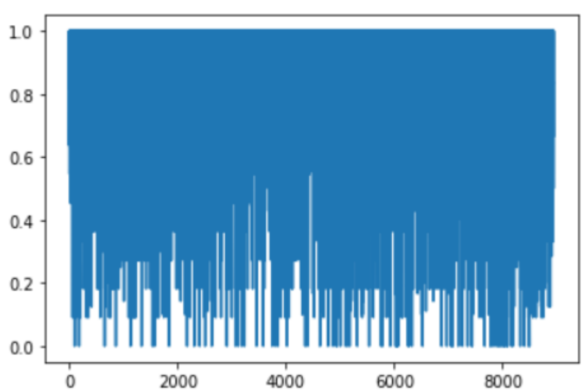
```
In [37]: plt.plot(credit['BALANCE'])
Out[37]: [<matplotlib.lines.Line2D at 0x119383b2088>]
```



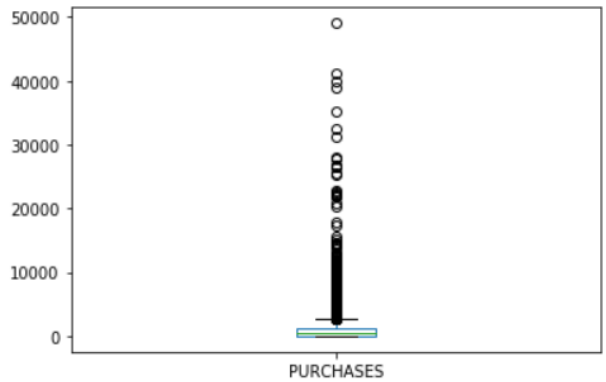

```
In [38]: boxplot(credit['BALANCE_FREQUENCY'])
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x119383e2a08>
```



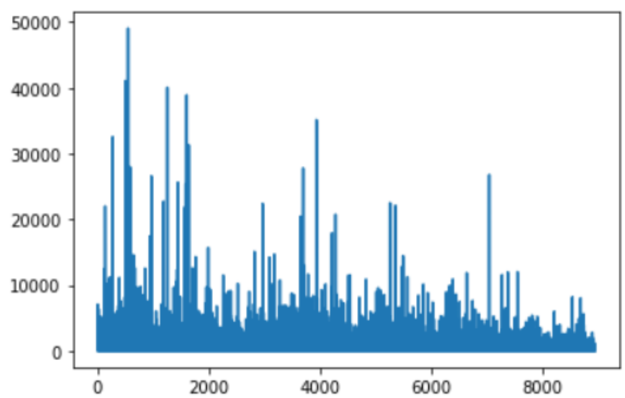

```
In [39]: plt.plot(credit['BALANCE_FREQUENCY'])
Out[39]: [<matplotlib.lines.Line2D at 0x11938471a48>]
```



```
In [40]: boxplot(credit['PURCHASES'])
Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x119384afd48>
```




```
In [41]: plt.plot(credit['PURCHASES'])
Out[41]: [<matplotlib.lines.Line2D at 0x11938bf98c8>]
```



Observations:

- From the above description of some variables, we can see that there is high variance among the values and this leads to the skewness in the data.
- Hence to avoid this we will be applying log transformation on all the variables present in the dataset, this solves the problem of skewness.

```
# log transformation

cr_log=credit.drop(['CUST_ID','purchase_type'],axis=1).applymap(lambda x: np.log(x+1))
```

```
In [43]: cr_log.describe().T
```

Out[43]:

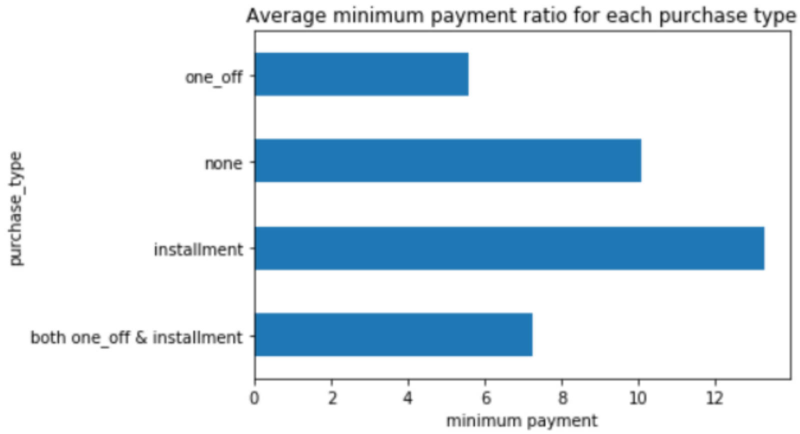
	count	mean	std	min	25%	50%	75%	max
BALANCE	8950.0	6.161637	2.013303	0.000000	4.861995	6.773521	7.628099	9.854515
BALANCE_FREQUENCY	8950.0	0.619940	0.148590	0.000000	0.635989	0.693147	0.693147	0.693147
PURCHASES	8950.0	4.899647	2.916872	0.000000	3.704627	5.892417	7.013133	10.800403
ONEOFF_PURCHASES	8950.0	3.204274	3.246365	0.000000	0.000000	3.663562	6.360274	10.615512
INSTALLMENTS_PURCHASES	8950.0	3.352403	3.082973	0.000000	0.000000	4.499810	6.151961	10.021315
CASH_ADVANCE	8950.0	3.319086	3.566298	0.000000	0.000000	0.000000	7.016449	10.760839
PURCHASES_FREQUENCY	8950.0	0.361268	0.277317	0.000000	0.080042	0.405465	0.650588	0.693147
ONEOFF_PURCHASES_FREQUENCY	8950.0	0.158699	0.216672	0.000000	0.000000	0.080042	0.262364	0.693147
PURCHASES_INSTALLMENTS_FREQUENCY	8950.0	0.270072	0.281852	0.000000	0.000000	0.154151	0.559616	0.693147
CASH_ADVANCE_FREQUENCY	8950.0	0.113512	0.156716	0.000000	0.000000	0.000000	0.200671	0.916291
CASH_ADVANCE_TRX	8950.0	0.817570	1.009316	0.000000	0.000000	0.000000	1.609438	4.820282
PURCHASES_TRX	8950.0	1.894731	1.373856	0.000000	0.693147	2.079442	2.890372	5.883322
CREDIT_LIMIT	8950.0	8.094825	0.819629	3.931826	7.378384	8.006701	8.779711	10.308986
PAYMENTS	8950.0	6.624540	1.591763	0.000000	5.951361	6.754489	7.550732	10.834125
MINIMUM_PAYMENTS	8950.0	5.916079	1.169929	0.018982	5.146667	5.747301	6.671670	11.243832
PRC_FULL_PAYMENT	8950.0	0.117730	0.211617	0.000000	0.000000	0.000000	0.133531	0.693147
TENURE	8950.0	2.519680	0.130367	1.945910	2.564949	2.564949	2.564949	2.564949
Monthly_avg_purchase	8950.0	3.050877	2.002823	0.000000	1.481458	3.494587	4.587295	8.315721
Monthly_cash_advance	8950.0	2.163970	2.429741	0.000000	0.000000	0.000000	4.606022	8.276166
limit_usage	8950.0	0.296081	0.250303	0.000000	0.040656	0.264455	0.540911	2.827902
payment_minpay	8950.0	1.357600	0.940149	0.000000	0.648817	1.109459	1.953415	8.830767

```
col=['BALANCE','PURCHASES','CASH_ADVANCE','TENURE','PAYMENTS','MINIMUM_PAYMENTS','PRC_FULL_PAYMENT','CREDIT_LIMIT']
cr_pre=cr_log[[x for x in cr_log.columns if x not in col ]]
```

Finding insights from the data

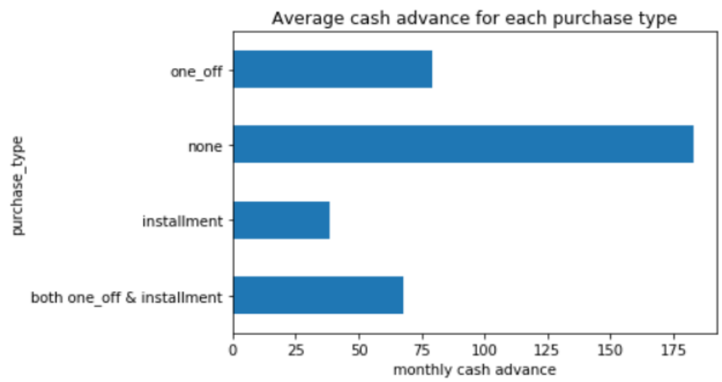
```
# Average payment_minpayment ratio for each purchase type.

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

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

```
In [46]: credit.groupby('purchase_type').apply(lambda x: np.mean(x['payment_minpay'])).plot.barh()
plt.title('Average minimum payment ratio for each purchase type')
plt.xlabel('minimum payment');
```



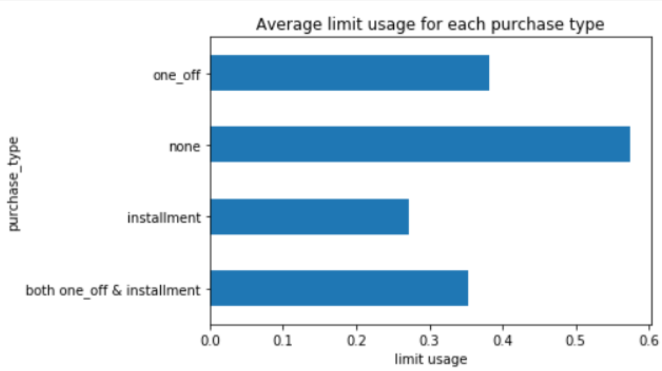
Insight 1: Customers who make transactions in installments are paying the amount regularly

```
In [72]: credit.groupby('purchase_type').apply(lambda x: np.mean(x['Monthly_cash_advance'])).plot.barh()
plt.title('Average cash advance for each purchase type')
plt.xlabel('monthly cash advance');
```



Insight 2: Customers who neither make a transaction in one-off payments nor installments are having high monthly cash advances

```
In [48]: credit.groupby('purchase_type').apply(lambda x: np.mean(x['limit_usage'])).plot.barh()
plt.title('Average limit usage for each purchase type')
plt.xlabel('limit usage');
```



Insight 3: Less limit usage gives high credit score and the good score is with the customers who make transactions in installments

Dataset Preparations for model selection

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

cre_original.describe().T
```

Out[44]:

	count	mean	std	min	25%	50%	75%	max
BALANCE	8950.0	1564.474828	2081.531879	0.000000	128.281915	873.385231	2054.140036	19043.138560
BALANCE_FREQUENCY	8950.0	0.877271	0.236904	0.000000	0.888889	1.000000	1.000000	1.000000
PURCHASES	8950.0	1003.204834	2136.634782	0.000000	39.635000	361.280000	1110.130000	49039.570000
ONEOFF_PURCHASES	8950.0	592.437371	1659.887917	0.000000	0.000000	38.000000	577.405000	40761.250000
INSTALLMENTS_PURCHASES	8950.0	411.067645	904.338115	0.000000	0.000000	89.000000	468.637500	22500.000000
CASH_ADVANCE	8950.0	978.871112	2097.163877	0.000000	0.000000	0.000000	1113.821139	47137.211760
PURCHASES_FREQUENCY	8950.0	0.490351	0.401371	0.000000	0.083333	0.500000	0.916667	1.000000
ONEOFF_PURCHASES_FREQUENCY	8950.0	0.202458	0.298336	0.000000	0.000000	0.083333	0.300000	1.000000
PURCHASES_INSTALLMENTS_FREQUENCY	8950.0	0.364437	0.397448	0.000000	0.000000	0.166667	0.750000	1.000000
CASH_ADVANCE_FREQUENCY	8950.0	0.135144	0.200121	0.000000	0.000000	0.000000	0.222222	1.500000
CASH_ADVANCE_TRX	8950.0	3.248827	6.824647	0.000000	0.000000	0.000000	4.000000	123.000000
PURCHASES_TRX	8950.0	14.709832	24.857649	0.000000	1.000000	7.000000	17.000000	358.000000
CREDIT_LIMIT	8950.0	4494.282473	3638.646702	50.000000	1600.000000	3000.000000	6500.000000	30000.000000
PAYMENTS	8950.0	1733.143852	2895.063757	0.000000	383.276166	856.901546	1901.134317	50721.483360
MINIMUM_PAYMENTS	8950.0	844.906767	2332.792322	0.019163	170.857654	312.343947	788.713501	76406.207520
PRC_FULL_PAYMENT	8950.0	0.153715	0.292499	0.000000	0.000000	0.000000	0.142857	1.000000
TENURE	8950.0	11.517318	1.338331	6.000000	12.000000	12.000000	12.000000	12.000000
Monthly_avg_purchase	8950.0	86.175173	180.508787	0.000000	3.399375	31.936667	97.228333	4086.630833
Monthly_cash_advance	8950.0	88.977984	193.136115	0.000000	0.000000	0.000000	99.085196	3928.100980
limit_usage	8950.0	0.388884	0.389722	0.000000	0.041494	0.302720	0.717571	15.909951
payment_minpay	8950.0	9.059164	118.180526	0.000000	0.913275	2.032717	6.052729	6840.528861
both one_off & installment	8950.0	0.309944	0.462496	0.000000	0.000000	0.000000	1.000000	1.000000
installment	8950.0	0.252514	0.434479	0.000000	0.000000	0.000000	1.000000	1.000000
none	8950.0	0.228156	0.419667	0.000000	0.000000	0.000000	0.000000	1.000000
one_off	8950.0	0.209385	0.406893	0.000000	0.000000	0.000000	0.000000	1.000000

```
cr_pre['purchase_type']=credit.loc[:,'purchase_type']

cr_pre.head()
```

In [46]: cr_pre.head()

Out[46]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_
0	0.597837	0.000000	4.568506	0.154151	0.000000	
1	0.646627	0.000000	0.000000	0.000000	0.000000	
2	0.693147	6.651791	0.000000	0.693147	0.693147	
3	0.492477	7.313220	0.000000	0.080042	0.080042	
4	0.693147	2.833213	0.000000	0.080042	0.080042	

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

df_dummy=df_dummy.drop(['purchase_type'],axis=1)

df_dummy.head()
```

```
In [49]: df_dummy.head()
```

Out[49]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_I
0	0.597837	0.000000	4.568506	0.154151	0.000000	
1	0.646627	0.000000	0.000000	0.000000	0.000000	
2	0.693147	6.651791	0.000000	0.693147	0.693147	
3	0.492477	7.313220	0.000000	0.080042	0.080042	
4	0.693147	2.833213	0.000000	0.080042	0.080042	

```
In [43]: df_dummy.describe().T
```

Out[43]:

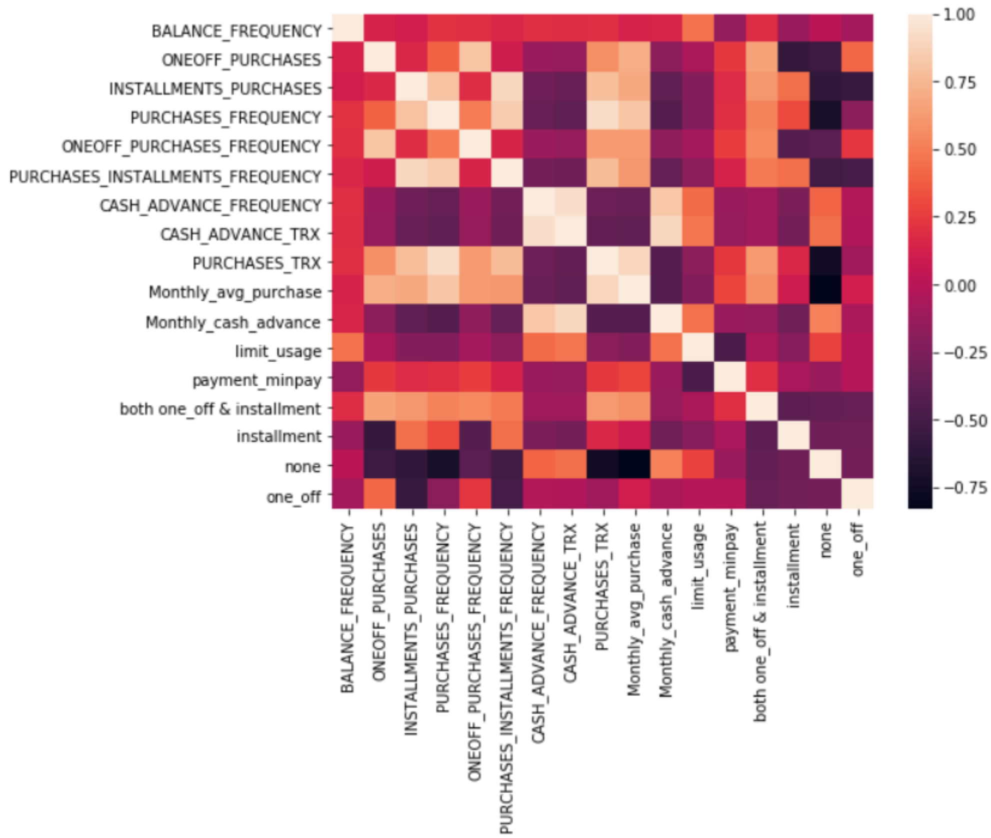
	count	mean	std	min	25%	50%	75%	max
BALANCE_FREQUENCY	8950.0	0.619940	0.148590	0.0	0.635989	0.693147	0.693147	0.693147
ONEOFF_PURCHASES	8950.0	3.204274	3.246365	0.0	0.000000	3.663562	6.360274	10.615512
INSTALLMENTS_PURCHASES	8950.0	3.352403	3.082973	0.0	0.000000	4.499810	6.151961	10.021315
PURCHASES_FREQUENCY	8950.0	0.361268	0.277317	0.0	0.080042	0.405465	0.650588	0.693147
ONEOFF_PURCHASES_FREQUENCY	8950.0	0.158699	0.216672	0.0	0.000000	0.080042	0.262364	0.693147
PURCHASES_INSTALLMENTS_FREQUENCY	8950.0	0.270072	0.281852	0.0	0.000000	0.154151	0.559616	0.693147
CASH_ADVANCE_FREQUENCY	8950.0	0.113512	0.156716	0.0	0.000000	0.000000	0.200671	0.916291
CASH_ADVANCE_TRX	8950.0	0.817570	1.009316	0.0	0.000000	0.000000	1.609438	4.820282
PURCHASES_TRX	8950.0	1.894731	1.373856	0.0	0.693147	2.079442	2.890372	5.883322
Monthly_avg_purchase	8950.0	3.050877	2.002823	0.0	1.481458	3.494587	4.587295	8.315721
Monthly_cash_advance	8950.0	2.163970	2.429741	0.0	0.000000	0.000000	4.606022	8.276166
limit_usage	8950.0	0.296081	0.250303	0.0	0.040656	0.264455	0.540911	2.827902
payment_minpay	8950.0	1.357600	0.940149	0.0	0.648817	1.109459	1.953415	8.830767
both one_off & installment	8950.0	0.309944	0.462496	0.0	0.000000	0.000000	1.000000	1.000000
installment	8950.0	0.252514	0.434479	0.0	0.000000	0.000000	1.000000	1.000000
none	8950.0	0.228156	0.419667	0.0	0.000000	0.000000	0.000000	1.000000
one_off	8950.0	0.209385	0.406893	0.0	0.000000	0.000000	0.000000	1.000000

- Finding the correlation among the variables in dataset

```
plt.subplots(figsize=(8, 6))
sns.heatmap(df_dummy.corr())
```

```
In [51]: plt.subplots(figsize=(8, 6))
         sns.heatmap(df_dummy.corr())
```

```
Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x2931cdc0548>
```



Observation

- The variables available for the model selection are very high in this dataset and this leads to dimensionality curse. In order to reduce the high dimensionality, curse we will use Principal Component Analysis technique.
- Before applying PCA we will standardize data to avoid effect of scale on our result. Centering and Scaling will make all features with equal weight. So we use standard scaler technique if there are any weightage issues among the variables of the dataset.

Standardizing data

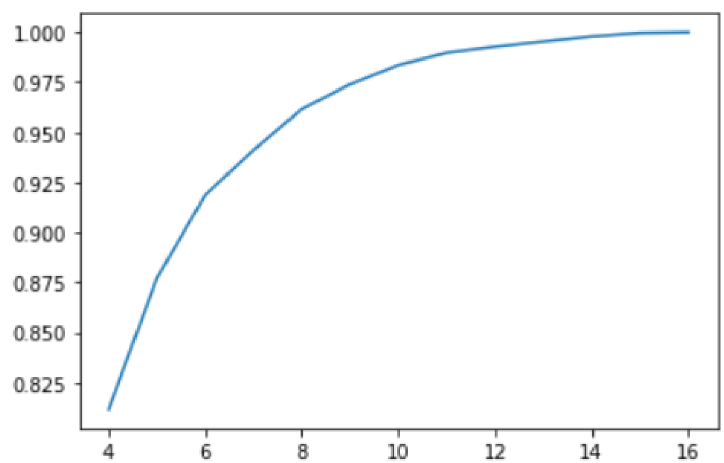
- To put data on the same scale

```
from sklearn.preprocessing import  StandardScaler
sc=StandardScaler()
df_scaled=sc.fit_transform(df_dummy)
```

```
from sklearn.decomposition import PCA
var_ratio={}
for n in range(4,17):
    pc=PCA(n_components=n,svd_solver='full')
    df_pca=pc.fit(df_scaled)
    var_ratio[n]=sum(df_pca.explained_variance_ratio_)

var_ratio
```

```
In [56]: var_ratio

Out[56]: {4: 0.8115442762351257,
          5: 0.8770555795291428,
          6: 0.9186492443512615,
          7: 0.941092525603013,
          8: 0.9616114053683061,
          9: 0.9739787081990645,
          10: 0.9835896584630706,
          11: 0.9897248107341953,
          12: 0.9927550009135226,
          13: 0.9953907562385423,
          14: 0.9979616898169592,
          15: 0.9996360473172953,
          16: 0.9999999999999998}
```

```
In [57]: pd.Series(var_ratio).plot();
```



Observation

- From the above variance ratio, we can see that the maximum variance of about 87% is explained when the number of components is 5. Hence, we choose n_components as 5 to reduce the dimensionality in the dataset.

```
df_scaled.shape
Out : (8950,17)
```

```
pc_final=PCA(n_components=5,svd_solver='full').fit(df_scaled)
reduced_df=pc_final.fit_transform(df_scaled)

df1=pd.DataFrame(reduced_df)
df1.head()
```

In [61]:
```
df1=pd.DataFrame(reduced_df)
df1.head()
```

Out[61]:

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

```
df1.shape

Out: (8950, 5)

col_list=df_dummy.columns
col_list

Out: 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 one_off & installment', 'installment', 'none', 'one_off'],
      dtype='object')

pd.DataFrame(pc_final.components_.T, columns=['PC_' +str(i) for i in range(5)],index=col_list)
```

	PC_0	PC_1	PC_2	PC_3	PC_4
BALANCE_FREQUENCY	0.029707	0.240072	-0.263140	-0.353549	-0.228681
ONEOFF_PURCHASES	0.214107	0.406078	0.239165	0.001520	-0.023197
INSTALLMENTS_PURCHASES	0.312051	-0.098404	-0.315625	0.087983	-0.002181
PURCHASES_FREQUENCY	0.345823	0.015813	-0.162843	-0.074617	0.115948
ONEOFF_PURCHASES_FREQUENCY	0.214702	0.362208	0.163222	0.036303	-0.051279
PURCHASES_INSTALLMENTS_FREQUENCY	0.295451	-0.112002	-0.330029	0.023502	0.025871
CASH_ADVANCE_FREQUENCY	-0.214336	0.286074	-0.278586	0.096353	0.360132
CASH_ADVANCE_TRX	-0.229393	0.291556	-0.285089	0.103484	0.332753
PURCHASES_TRX	0.355503	0.106625	-0.102743	-0.054296	0.104971
Monthly_avg_purchase	0.345992	0.141635	0.023986	-0.079373	0.194147
Monthly_cash_advance	-0.243861	0.264318	-0.257427	0.135292	0.268026
limit_usage	-0.146302	0.235710	-0.251278	-0.431682	-0.181885
payment_minpay	0.119632	0.021328	0.136357	0.591561	0.215446
both one_off & installment	0.241392	0.273676	-0.131935	0.254710	-0.340849
installment	0.082209	-0.443375	-0.208683	-0.190829	0.353821
none	-0.310283	-0.005214	-0.096911	0.245104	-0.342222
one_off	-0.042138	0.167737	0.472749	-0.338549	0.362585

```
# Factor Analysis: variance explained by each component-
pd.Series(pc_final.explained_variance_ratio_,index=['PC_'+ str(i) for i in range(5)])

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

Model Selection

Clustering

Based on our 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.**

```
from sklearn.cluster import KMeans

km_4=KMeans(n_clusters=4,random_state=42)
km_4.fit(reduced_df)
km_4.labels_
```

```
Out: array([0, 1, 3, ..., 0, 1, 3])

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

Out:
2    2758
0    2228
1    2090
3    1874
dtype: int64
```

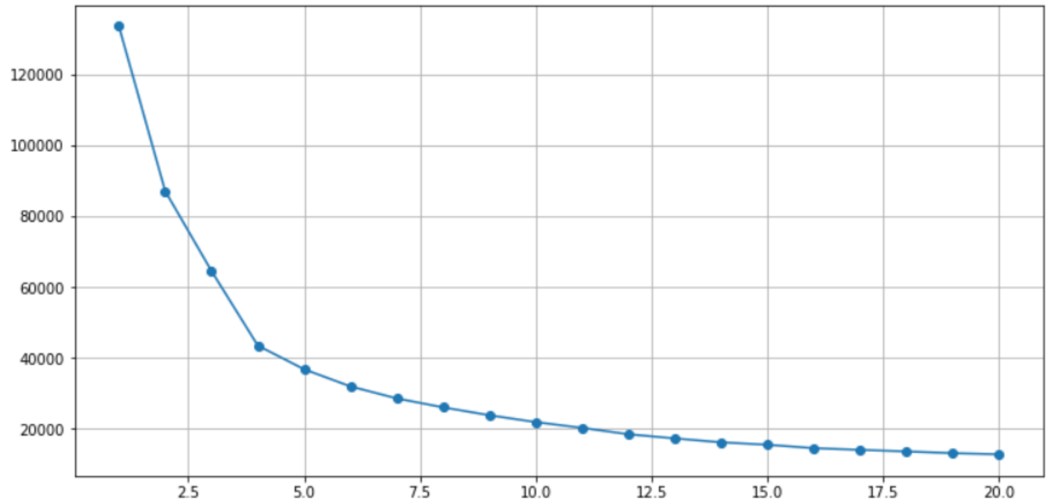
- Here we do not have known k value so we will find the K. To do that we need to take a cluster range between 1 and 21.

```
# Identify cluster errors

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

for num_clusters in cluster_range:
    clusters = KMeans( num_clusters )
    clusters.fit( reduced_df )
    cluster_errors.append( clusters.inertia_ ) # clusters.inertia_ is basically cluster error here

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

clusters_df[0:21]
```

	num_clusters	cluster_errors
0	1	133444.006425
1	2	87022.451581
2	3	64498.884321
3	4	43508.398870
4	5	36826.460438
5	6	32005.216927
6	7	28621.217763
7	8	26105.016911
8	9	23881.589387
9	10	21947.573027
10	11	20330.546458
11	12	18528.285614
12	13	17357.405897
13	14	16251.622741
14	15	15574.823924
15	16	14611.278311
16	17	14127.589818
17	18	13687.717089
18	19	13196.558450
19	20	12880.931020

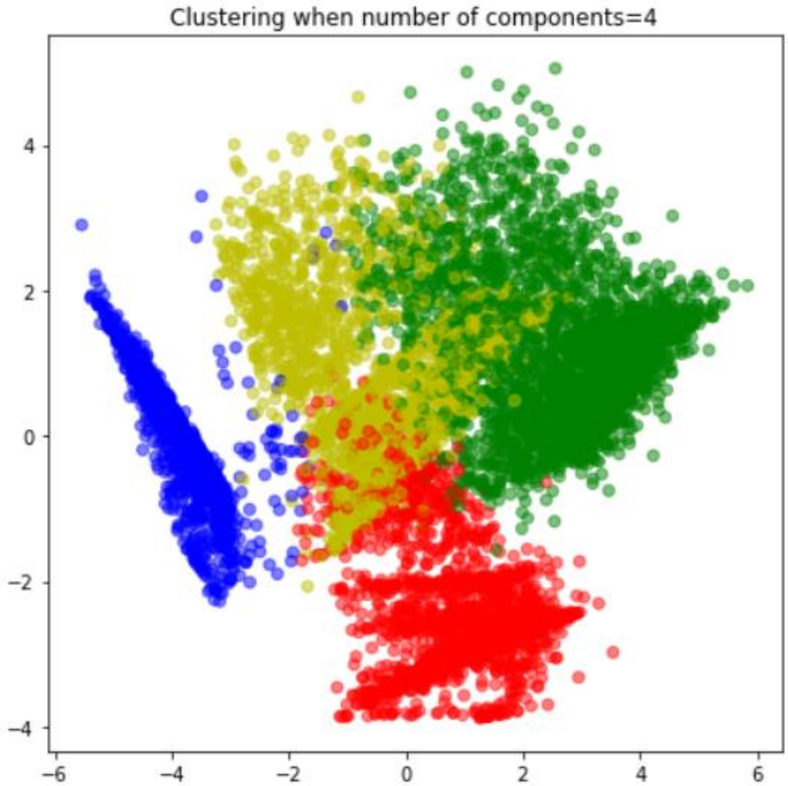
```
# checking k value from elbow plot

import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
plt.grid()
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
```

Out[71]: [<matplotlib.lines.Line2D at 0x293232d39c8>]

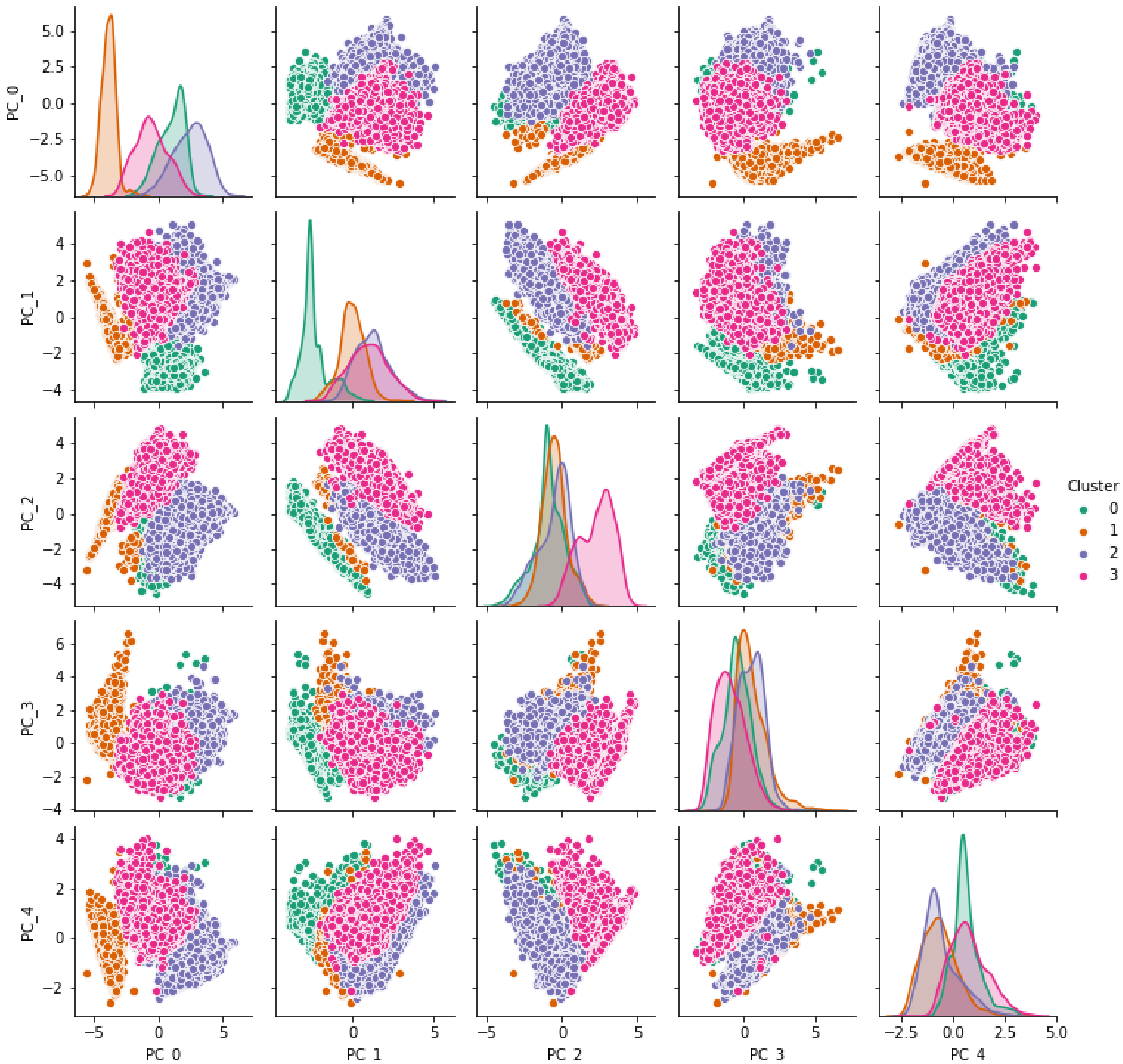


- From above graph, it is can take we can take k as 4,5 or 6

```
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_df[:,0],reduced_df[:,1],c=label_color,cmap='Spectral',alpha=0.5)

plt.title('Clustering when number of components=4');
```




```
df_pair_plot=pd.DataFrame(reduced_df,columns=['PC_' +str(i) for i in range(5)])

df_pair_plot['Cluster']=km_4.labels_

#pairwise relationship of components on the data

sns.pairplot(df_pair_plot,hue='Cluster', palette= 'Dark2', diag_kind='kde',height=2)

plt.savefig("pairplot")
```



Observations:

- From the above graphs we can conclude that the only PC_0 and PC_1 are identifiable clusters and hence we go with further analysis by increasing the number of clusters value to identify more number of insights about the customers present in the dataset.

```
# Key performance variable selection . here I am dropping variables which are used in deriving new KPI

col_kpi=['PURCHASES_TRX','Monthly_avg_purchase','Monthly_cash_advance','limit_usage','CASH_ADVANCE_TRX',
 'payment_minpay','both one_off & installment','installment','one_off','none','CREDIT_LIMIT']
```

```
cr_pre.describe().T
```

```
In [77]: cr_pre.describe().T
Out[77]:
```

	count	mean	std	min	25%	50%	75%	max
BALANCE_FREQUENCY	8950.0	0.619940	0.148590	0.0	0.635989	0.693147	0.693147	0.693147
ONEOFF_PURCHASES	8950.0	3.204274	3.246365	0.0	0.000000	3.663562	6.360274	10.615512
INSTALLMENTS_PURCHASES	8950.0	3.352403	3.082973	0.0	0.000000	4.499810	6.151961	10.021315
PURCHASES_FREQUENCY	8950.0	0.361268	0.277317	0.0	0.080042	0.405465	0.650588	0.693147
ONEOFF_PURCHASES_FREQUENCY	8950.0	0.158699	0.216672	0.0	0.000000	0.080042	0.262364	0.693147
PURCHASES_INSTALLMENTS_FREQUENCY	8950.0	0.270072	0.281852	0.0	0.000000	0.154151	0.559616	0.693147
CASH_ADVANCE_FREQUENCY	8950.0	0.113512	0.156716	0.0	0.000000	0.000000	0.200671	0.916291
CASH_ADVANCE_TRX	8950.0	0.817570	1.009316	0.0	0.000000	0.000000	1.609438	4.820282
PURCHASES_TRX	8950.0	1.894731	1.373856	0.0	0.693147	2.079442	2.890372	5.883322
Monthly_avg_purchase	8950.0	3.050877	2.002823	0.0	1.481458	3.494587	4.587295	8.315721
Monthly_cash_advance	8950.0	2.163970	2.429741	0.0	0.000000	0.000000	4.606022	8.276166
limit_usage	8950.0	0.296081	0.250303	0.0	0.040656	0.264455	0.540911	2.827902
payment_minpay	8950.0	1.357600	0.940149	0.0	0.648817	1.109459	1.953415	8.830767

```
# Concatenating 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 [79]: cluster_df_4.head()
Out[79]:
```

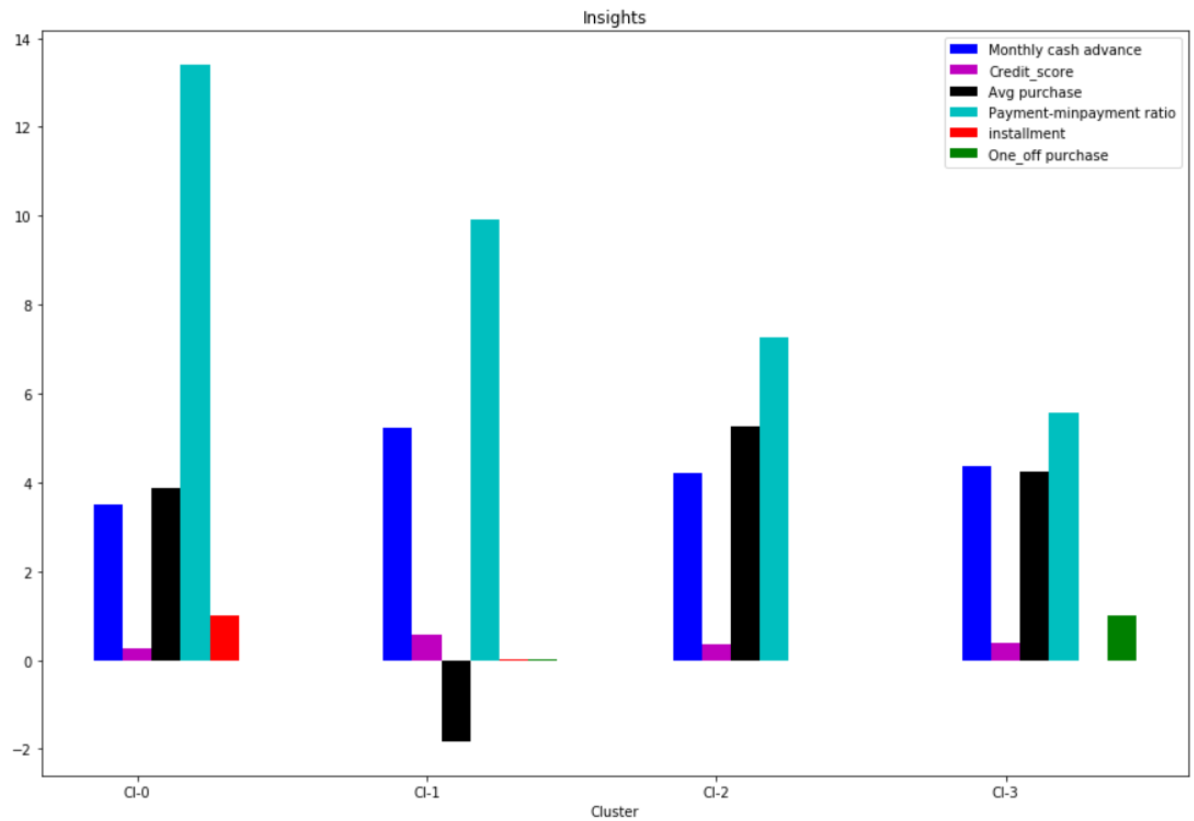
	PURCHASES_TRX	Monthly_avg_purchase	Monthly_cash_advance	limit_usage	CASH_ADVANCE_TRX	payment_minpay	both one_off & installment	installment	one_off	no
0	2	7.950000	0.000000	0.040901	0	1.446508	0	1	0	
1	0	0.000000	536.912124	0.457495	4	3.826241	0	0	0	
2	12	64.430833	0.000000	0.332687	0	0.991682	0	0	1	
3	1	124.916667	17.149001	0.222223	1	0.000000	0	0	1	
4	1	1.333333	0.000000	0.681429	0	2.771075	0	0	1	

```
# 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
```

Cluster_4	0	1	2	3
PURCHASES_TRX	12.053860	0.045933	33.125453	7.118997
Monthly_avg_purchase	47.573598	0.159337	193.696083	69.758276
Monthly_cash_advance	33.489846	186.298043	67.620006	77.843485
limit_usage	0.264275	0.576217	0.354487	0.378727
CASH_ADVANCE_TRX	1.019300	6.552632	2.807107	2.864995
payment_minpay	13.402660	9.927979	7.268605	5.561421
both one_off & installment	0.001795	0.002392	1.000000	0.003735
installment	0.998205	0.017225	0.000000	0.000000
one_off	0.000000	0.003349	0.000000	0.996265
none	0.000000	0.977033	0.000000	0.000000
CREDIT_LIMIT	3335.697210	4055.582137	5750.015565	4512.905630

```
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['installment',:].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()
```

```
<matplotlib.legend.Legend at 0x29326e87f48>
```



Observation

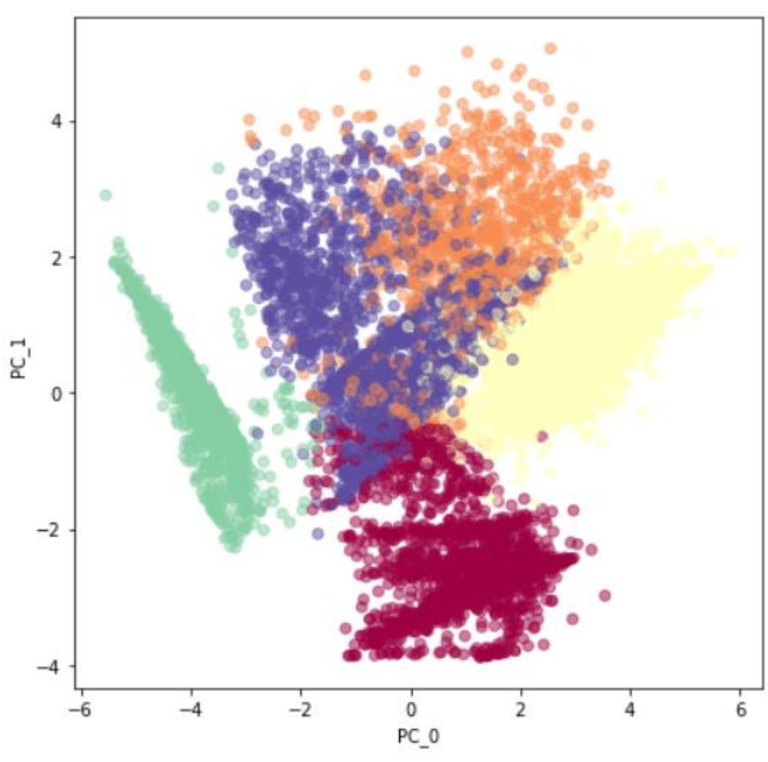
- From the above graph we can see that the four clusters have been categorised perfectly so that the difference in each cluster can be understood

In [82]:
```python
# 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='Percentage')
print( "Cluster -4 ",'\n')
print(pd.concat([pd.Series(s.values,name='Size'),per],axis=1),'\n')
```

```
Cluster_4
0          0    2228
1          1    2090
2          2    2758
3          3    1874
Name: Cluster_4, dtype: int64

Cluster -4

   Size  Percentage
0  2228   24.893855
1  2090   23.351955
2  2758   30.815642
3  1874   20.938547
```

Exploring the insights if the number of clusters=5

```
#kmeans with 5 clusters

km_5=KMeans(n_clusters=5,random_state=42)
km_5=km_5.fit(reduced_df)
km_5.labels_

Out:
array([0, 3, 4, ..., 0, 3, 4])

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

Out:
0    2130
3    2084
2    1985
4    1860
1     891
dtype: int64


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



```
cluster_df_5=pd.concat([cre_original[col_kpi],pd.Series(km_5.labels_,name='Cluster_5')],axis=1
)

# Finding Mean of features for each cluster

five_cluster=cluster_df_5.groupby('Cluster_5')\
.apply(lambda x: x[col_kpi].mean()).T

five_cluster
```

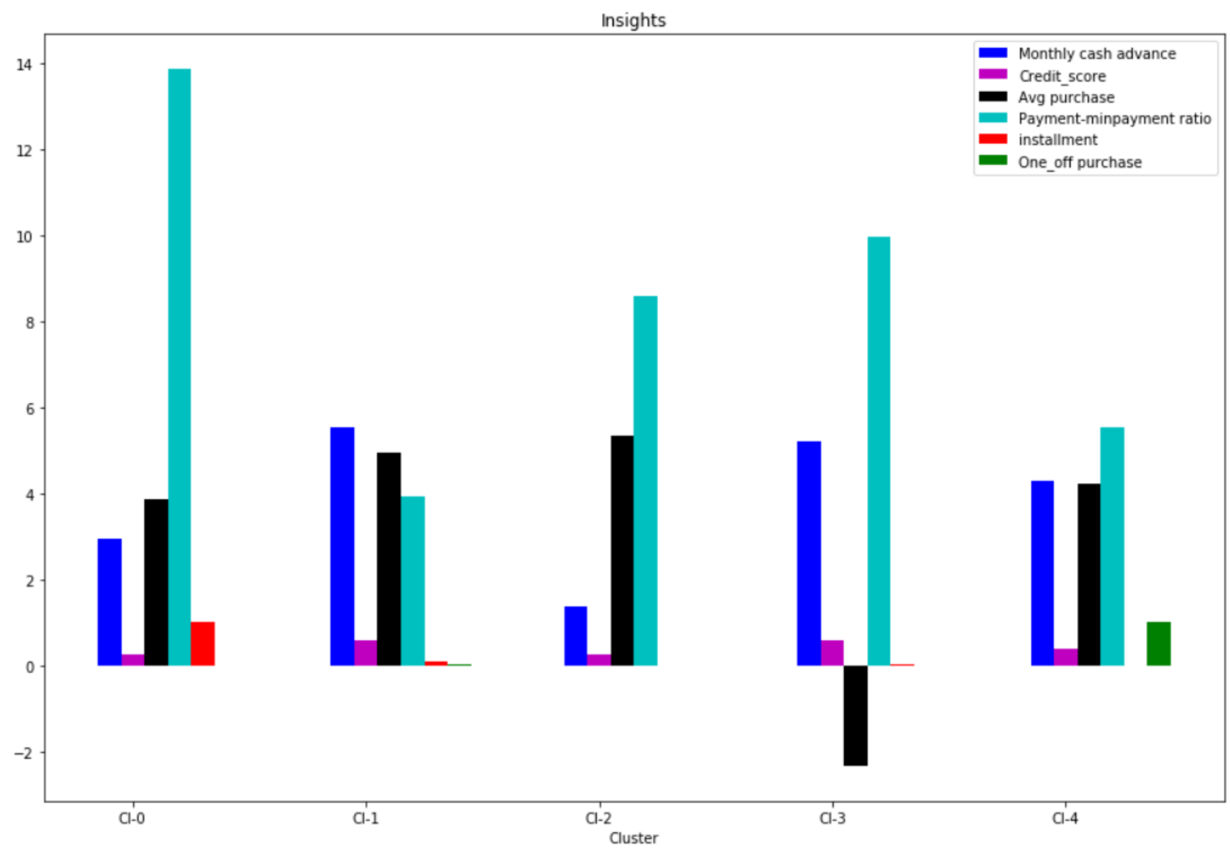
Cluster_5	0	1	2	3	4
PURCHASES_TRX	11.896714	27.536476	34.538035	0.035509	7.067742
Monthly_avg_purchase	47.239695	141.648931	209.814279	0.096572	68.685725
Monthly_cash_advance	19.154845	252.400192	3.996969	185.109488	73.635703
limit_usage	0.246825	0.594982	0.262694	0.576260	0.377563
CASH_ADVANCE_TRX	0.480282	10.519641	0.152645	6.454894	2.648387
payment_minpay	13.866212	3.920172	8.569707	9.950170	5.540102
both one_off & installment	0.000000	0.878788	1.000000	0.000000	0.003226
installment	1.000000	0.106622	0.000000	0.016795	0.000000
one_off	0.000000	0.014590	0.000000	0.003359	0.996774
none	0.000000	0.000000	0.000000	0.979846	0.000000
CREDIT_LIMIT	3223.856049	5845.791246	5724.213063	4047.344850	4489.884490

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

Out:
Cluster_5
0         0    2130
1         1     891
2         2    1985
3         3    2084
4         4    1860
Name: Cluster_5, dtype: int64
```

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

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

```
purchase= np.log(five_cluster.loc['Monthly_avg_purchase',:].values)
payment=five_cluster.loc['payment_minpay',:].values
installment=five_cluster.loc['installment',:].values
one_off=five_cluster.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','Cl-4'))
plt.legend()
```

```
<matplotlib.legend.Legend at 0x2932862d108>
```



Observation

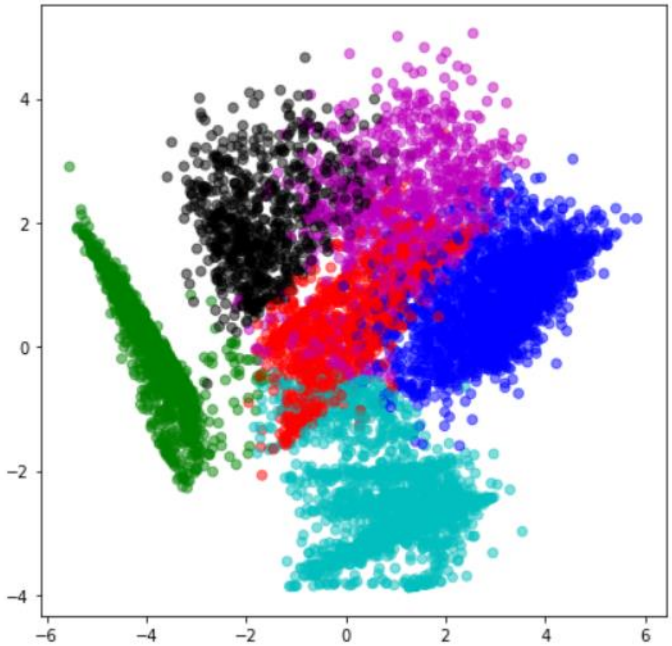
- From the above graph, we can't come to a particular conclusion regarding the behavior of customer groups, because cluster 2 is having highest average purchases in the transactions, but at the same time cluster1 has highest cash advance and second highest purchases.

```
# percentage of each cluster

print("Cluster-5")
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))
```

Exploring the insights if the number of cluster=6

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

Out:
array([3, 2, 0, ..., 3, 2, 5])

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_df[:,0],reduced_df[:,1],c=label_color,cmap='Spectral',alpha=0.5)
```

```
<matplotlib.collections.PathCollection at 0x2932422fb08>
```



```
cluster_df_6=pd.concat([cre_original[col_kpi],pd.Series(km_6.labels_,name='Cluster_6')],axis=1
)

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

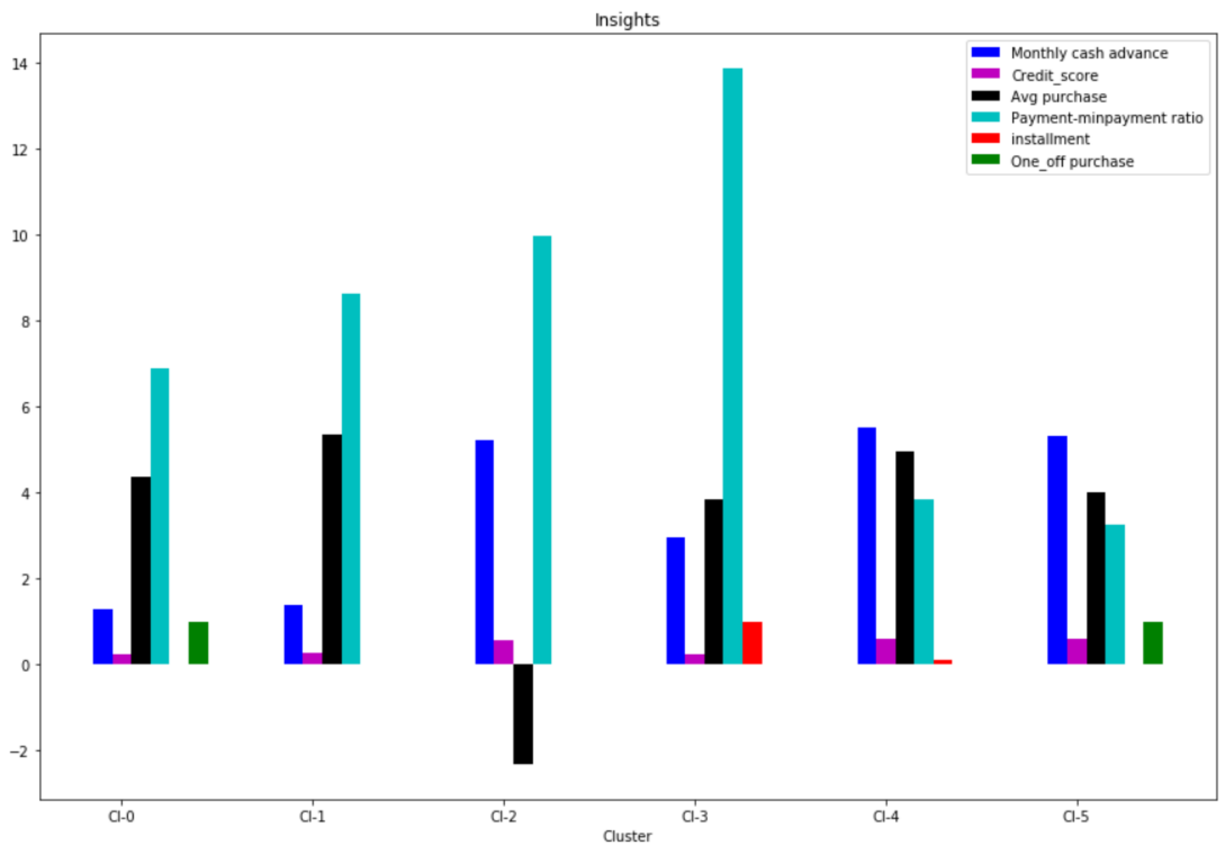
```
In [97]: six_cluster=cluster_df_6.groupby('Cluster_6').apply(lambda x: x[col_kpi].mean()).T
         six_cluster
```

Out[97]:

Cluster_6	0	1	2	3	4	5
PURCHASES_TRX	7.745148	34.653320	0.033205	11.896762	27.742922	5.980000
Monthly_avg_purchase	78.444637	210.512330	0.098395	47.243825	140.643565	54.143932
Monthly_cash_advance	3.654858	3.942946	184.912834	19.155048	243.934772	205.399766
limit_usage	0.244888	0.262170	0.575884	0.246733	0.595784	0.606433
CASH_ADVANCE_TRX	0.130802	0.149012	6.435034	0.484280	10.057758	7.632857
payment_minpay	6.898533	8.610468	9.967837	13.861937	3.835641	3.252112
both one_off & installment	0.009283	1.000000	0.000000	0.000000	0.894677	0.000000
installment	0.000000	0.000000	0.017324	1.000000	0.105323	0.000000
one_off	0.990717	0.000000	0.000000	0.000000	0.000000	1.000000
none	0.000000	0.000000	0.982676	0.000000	0.000000	0.000000
CREDIT_LIMIT	4465.865490	5722.145428	4048.925249	3224.454896	5817.157418	4600.649351

```
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['installment',:].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_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','Cl-4','Cl-5'))
plt.legend()
```

```
<matplotlib.legend.Legend at 0x29328581dc8>
```



Observation:

- From the above graph we can see that cluster 2 and cluster 4 have similar behavior regarding the parameters, hence distinguishing between the clusters is hard when we have the number of clusters as 6

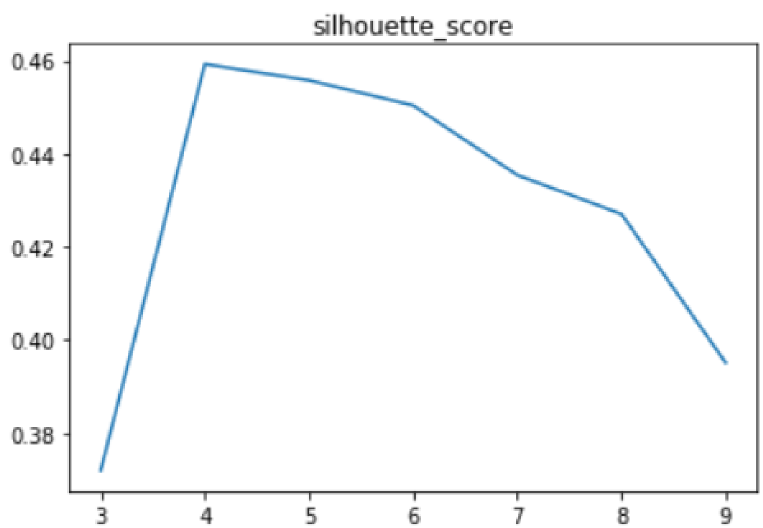
```
cash_advance=np.log(six_cluster.iloc[2,:].values)
credit_score=list(six_cluster.iloc[3,:].values)
print(cash_advance)
print(credit_score)

Out:
[1.29605733 1.37192804 5.21988454 2.95256629 5.49690086 5.32495816]
[0.24488793326165034, 0.26216962861657617, 0.5758841059122126,
0.24673287577047076, 0.5957844119450174, 0.6064330330654714]
```

Checking performance metrics for K means

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

```
from sklearn.metrics import calinski_harabasz_score,silhouette_score
score={}
score_c={}
for n in range(3,10):
    km_score=KMeans(n_clusters=n)
    km_score.fit(reduced_df)
    score_c[n]=calinski_harabasz_score(reduced_df,km_score.labels_)
    score[n]=silhouette_score(reduced_df,km_score.labels_)

print(score)

Out:
{3: 0.37199332646474775, 4: 0.45925855175999947, 5: 0.4557969467383015, 6:
0.45040500121395316, 7: 0.4354310442029417, 8: 0.42706833598976296, 9:
0.39512255230583815}

pd.Series(score).plot()
plt.title('silhouette_score')
```

```
Text(0.5, 1.0, 'silhouette_score')
```

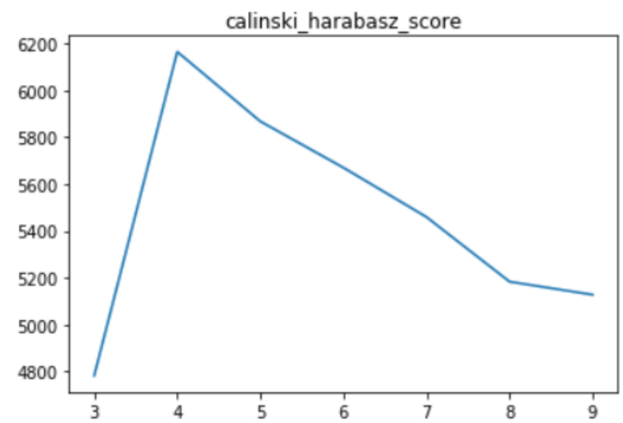



```
print(score_c)
```

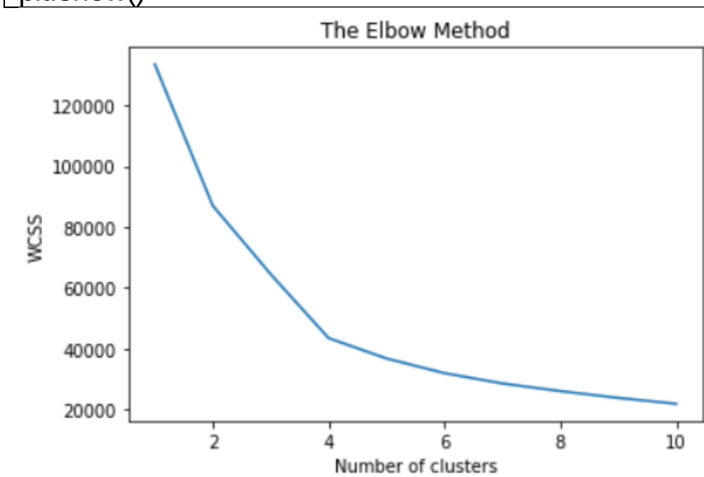
Out:
{3: 4781.934521021165, 4: 6164.054484808374, 5: 5867.003840603487, 6: 5669.5040396521545, 7: 5458.826824079906, 8: 5182.940231634099, 9: 5127.351583136153}

```
pd.Series(score_c).plot()
plt.title('calinski_harabasz_score')
```

```
Text(0.5, 1.0, 'calinski_harabasz_score')
```




```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(reduced_df)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



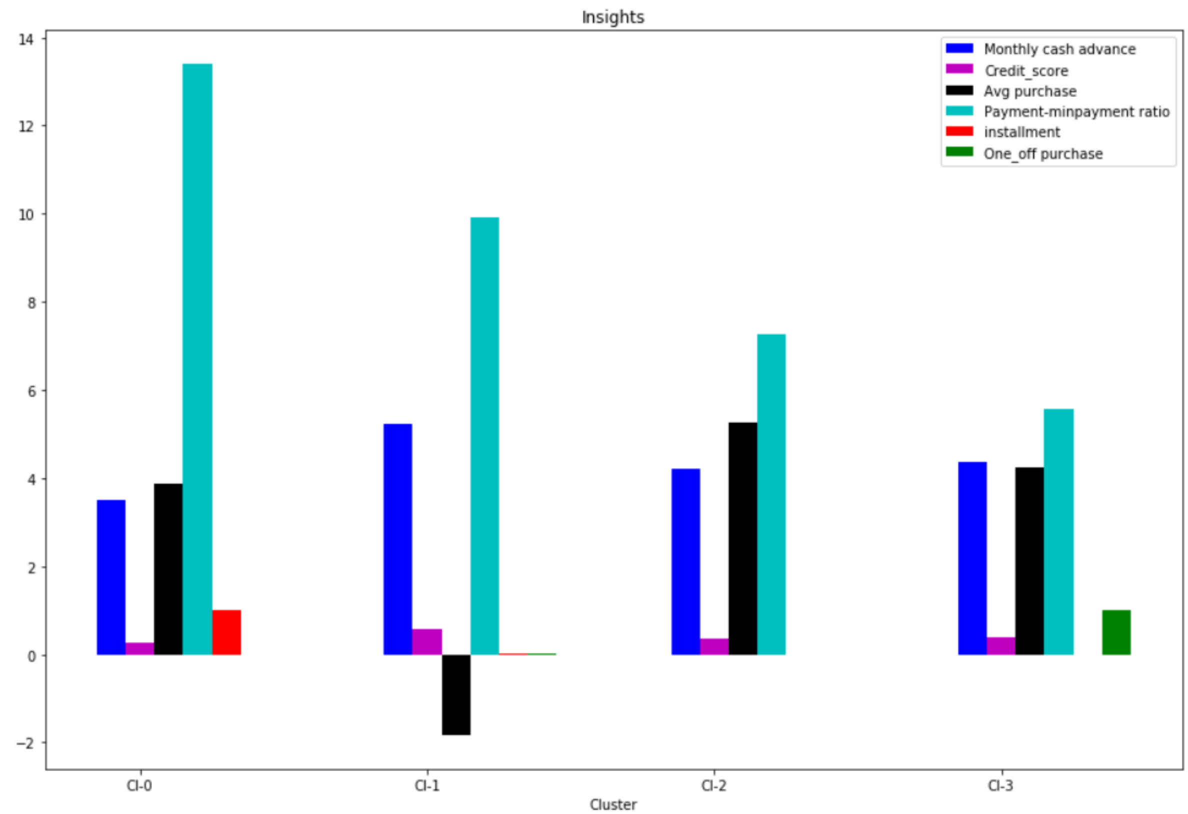
Observation:

- From all the above graphs we can conclude the performance of the KMeans Model regarding the explanation of data distribution and measure of spread is highest when we consider the number of clusters as four.

Final KMeans Model

```
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['installment',:].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()
```

<matplotlib.legend.Legend at 0x293287aad08>



Marketing Strategies

Cluster 0:

Customers who fall under this category of cluster are having the best credit card and also paying the dues on time without defaults. Hence these group of customers must be rewarded with reward points and thus make them do more transactions in future.

Cluster1:

Customers belong to this category of cluster having the highest cash advance and poor credit score yet these customers pay the due amounts of the installments on time. Hence these customers may be given with the loan amounts at less interest charges, thus help the banks providing continuous services to these group of customers in future

Cluster2:

Customers belong to this cluster must be the primary focus regarding the marketing strategy because the customers under this cluster are making frequent purchases and also paying the dues on time thus maintaining good credit score. Customers in this cluster must be given with good reward points and provided with increased credit limit or the premium credit cards with some exciting offers make them do more transactions in the future.

Cluster3:

Customers belong to this cluster has the least minimum payment ratio and always does the one off payment transactions, hence no bank offers can excite these kind of customers. The marketing to this group of customers is hard and when the usage is minimum, this group can be ignored from the marketing strategy. Further the customers falling under this category can be rejected from issuing the credit cards in future.

SAME THINGS WE DO IN R

R CODE

```
library(dplyr)


#load the data

setwd("C:/Users/jerin/Desktop/R work/EDWISOR PROJECT")

seg <-read.csv("CC GENERAL.csv")

View(seg)
sum(is.na(seg$CUST_ID))
sum(is.na(seg$BALANCE))
sum(is.na(seg$BALANCE_FREQUENCY))
sum(is.na(seg$PURCHASES))
sum(is.na(seg$ONEOFF_PURCHASES))
sum(is.na(seg$INSTALLMENTS_PURCHASES))
sum(is.na(seg$CASH_ADVANCE))
sum(is.na(seg$PURCHASES_FREQUENCY))
sum(is.na(seg$ONEOFF_PURCHASES_FREQUENCY))
sum(is.na(seg$PURCHASES_INSTALLMENTS_FREQUENCY))
sum(is.na(seg$CASH_ADVANCE_FREQUENCY))
sum(is.na(seg$CASH_ADVANCE_TRX))
sum(is.na(seg$PURCHASES_TRX))
sum(is.na(seg$CREDIT_LIMIT))##1
sum(is.na(seg$PAYMENTS))
sum(is.na(seg$MINIMUM_PAYMENTS))##313
sum(is.na(seg$PRC_FULL_PAYMENT))
sum(is.na(seg$TENURE))


# 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#

seg$Monthly_Avg_PURCHASES =
seg$PURCHASES/(seg$PURCHASES_FREQUENCY*seg$TENURE)
seg$Monthly_CASH_ADVANCE =
seg$CASH_ADVANCE/(seg$CASH_ADVANCE_FREQUENCY*seg$TENURE)
seg$LIMIT_USAGE = seg$BALANCE/seg$CREDIT_LIMIT
seg$MIN_PAYMENTS_RATIO = seg$PAYMENTS/seg$MINIMUM_PAYMENTS

-+
+
-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(seg[Num_Vars], 2, mystats)))
View(Outliers)

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

# Outlier Treatment

seg$BALANCE[seg$BALANCE>5727.53]=5727.53
seg$BALANCE_FREQUENCY[seg$BALANCE_FREQUENCY>1.3510787]=1.3510787
seg$PURCHASES[seg$PURCHASES>5276.46]=5276.46
seg$Monthly_Avg_PURCHASES[seg$Monthly_Avg_PURCHASES>800.03] = 800.03
seg$ONEOFF_PURCHASES[seg$ONEOFF_PURCHASES>3912.2173709]=3912.2173709
seg$INSTALLMENTS_PURCHASES[seg$INSTALLMENTS_PURCHASES>2219.7438751]=2219.7438751
seg$CASH_ADVANCE[seg$CASH_ADVANCE>5173.1911125]=5173.1911125
seg$Monthly_CASH_ADVANCE[seg$Monthly_CASH_ADVANCE>2558.53] = 2558.53
seg$PURCHASES_FREQUENCY[seg$PURCHASES_FREQUENCY>1.2930919]=1.2930919
seg$ONEOFF_PURCHASES_FREQUENCY[seg$ONEOFF_PURCHASES_FREQUENCY>0.7991299]=0.7991299
seg$PURCHASES_INSTALLMENTS_FREQUENCY[seg$PURCHASES_INSTALLMENTS_FREQUENCY>1.1593329]=1.1593329
seg$CASH_ADVANCE_FREQUENCY[seg$CASH_ADVANCE_FREQUENCY>0.535387]=0.535387
seg$CASH_ADVANCE_TRX[seg$CASH_ADVANCE_TRX>16.8981202]=16.8981202
seg$PURCHASES_TRX[seg$PURCHASES_TRX>64.4251306]=64.4251306
seg$CREDIT_LIMIT[seg$CREDIT_LIMIT>11772.09]=11772.09
seg$LIMIT_USAGE[seg$LIMIT_USAGE>1.1683] = 1.1683
seg$PAYMENTS[seg$PAYMENTS>7523.26]=7523.26
seg$MINIMUM_PAYMENTS[seg$MINIMUM_PAYMENTS>5609.1065423]=5609.1065423
seg$MIN_PAYMENTS_RATIO[seg$MIN_PAYMENTS_RATIO>249.9239] = 249.9239
seg$PRC_FULL_PAYMENT[seg$PRC_FULL_PAYMENT>0.738713]=0.738713
seg$TENURE[seg$TENURE>14.19398]=14.19398


# Missing Value Imputation with mean

seg$MINIMUM_PAYMENTS[which(is.na(seg$MINIMUM_PAYMENTS))] = 721.9256368
seg$CREDIT_LIMIT[which(is.na(seg$CREDIT_LIMIT))] = 4343.62
seg$Monthly_Avg_PURCHASES[which(is.na(seg$Monthly_Avg_PURCHASES))] =184.8991609
seg$Monthly_CASH_ADVANCE[which(is.na(seg$Monthly_CASH_ADVANCE))] = 717.7235629
seg$LIMIT_USAGE[which(is.na(seg$LIMIT_USAGE))] =0.3889264
seg$MIN_PAYMENTS_RATIO[which(is.na(seg$MIN_PAYMENTS_RATIO))]  = 9.3500701

# Checking Missing Value

check_Missing_Values=t(data.frame(apply(seg[Num_Vars], 2, mystats)))
```

```
View(check_Missing_Values)

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



# Variable Reduction (Factor Analysis)

Step_nums = seg[Num_Vars]
corrm= cor(Step_nums)
View(corrm)

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


scree(corrm,factors=T,pc=T,main="scree plot", hline=NULL, add=FALSE)### SCREE PLOT


eigen(corrm)$values



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")


# standardizing the data

segment_prepared =seg[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)

seg_new=cbind(seg,km_clust_3=cluster_three$cluster,km_clust_4=cluster_four$cluster,km_clu
st_5=cluster_five$cluster ,km_clust_6=cluster_six$cluster   )
View(seg_new)

# 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"
)




library(tables)

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

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

tt2 = as.data.frame.matrix(tt)
View(tt2)

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"
)

tt2
cluster_profiling2 = t(tt2)
```

References

https://www.kaggle.com

http://rprogramming.net/

https://medium.com/

https://stackoverflow.com/

https://www.rdocumentation.org

https://www.analyticsvidhya.com/blog