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 :

1) 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 costumers 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 costumers 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.

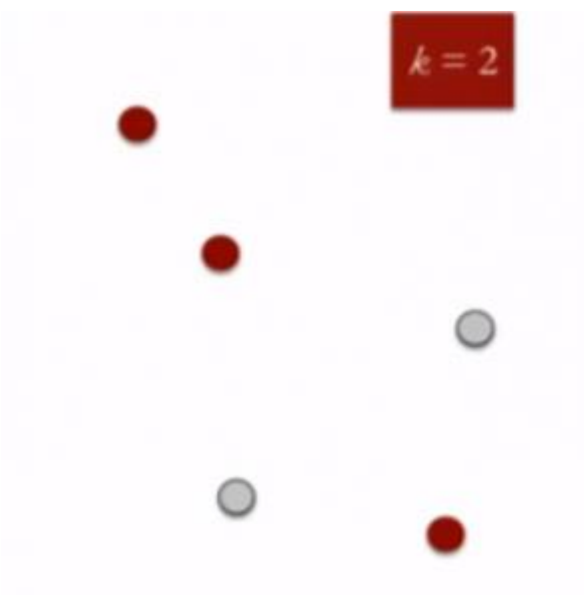
- **Density Models:** These models search the data space for areas of varied density of data points in the data space. It isolates various different density regions and assign the data points within these regions in the same cluster. Popular examples of density models are DBSCAN and OPTICS.

Now I will be taking you through two of the most popular clustering algorithms in detail – K Means clustering and Hierarchical clustering. Let's begin.

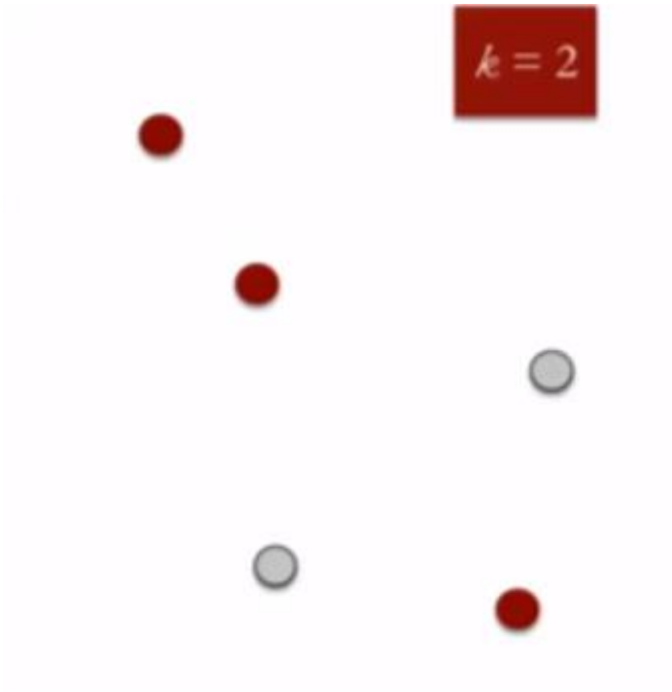
K Means Clustering

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

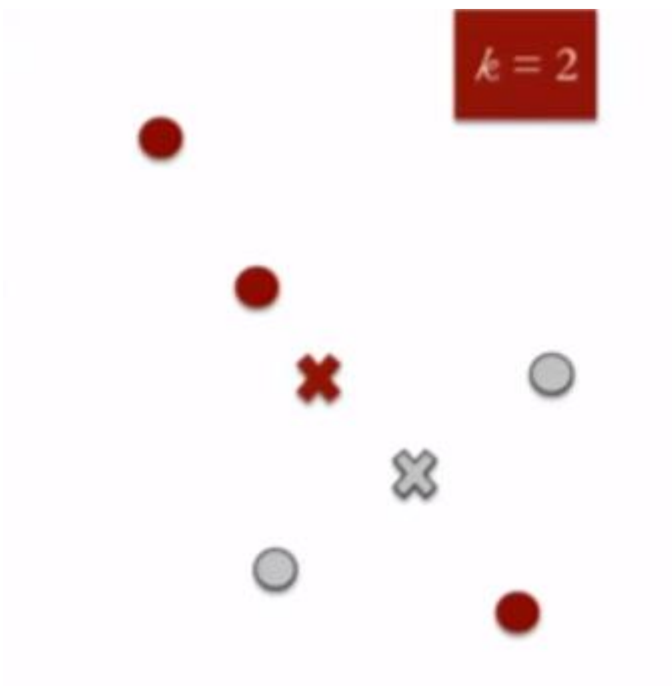
1. Specify the desired number of clusters K : Let us choose k=2 for these 5 data points in 2-D space.



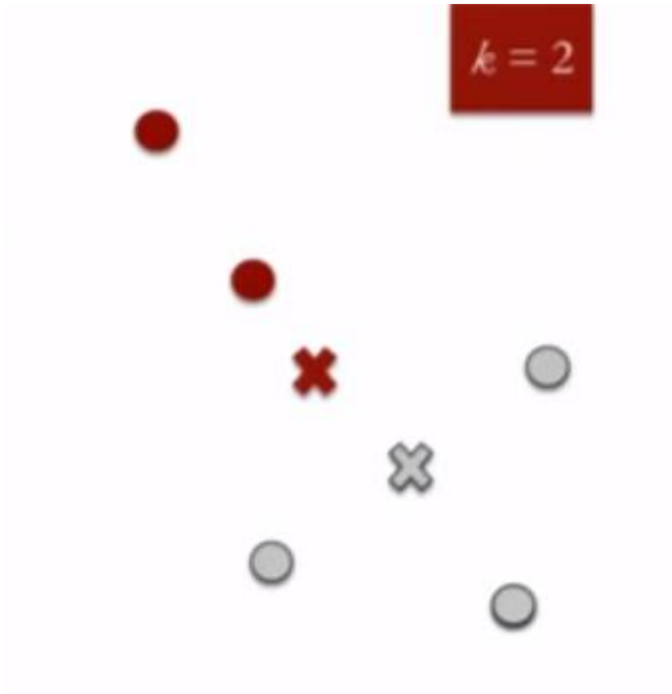
2. Randomly assign each data point to a cluster : Let's assign three points in cluster 1 shown using red color and two points in cluster 2 shown using grey color.



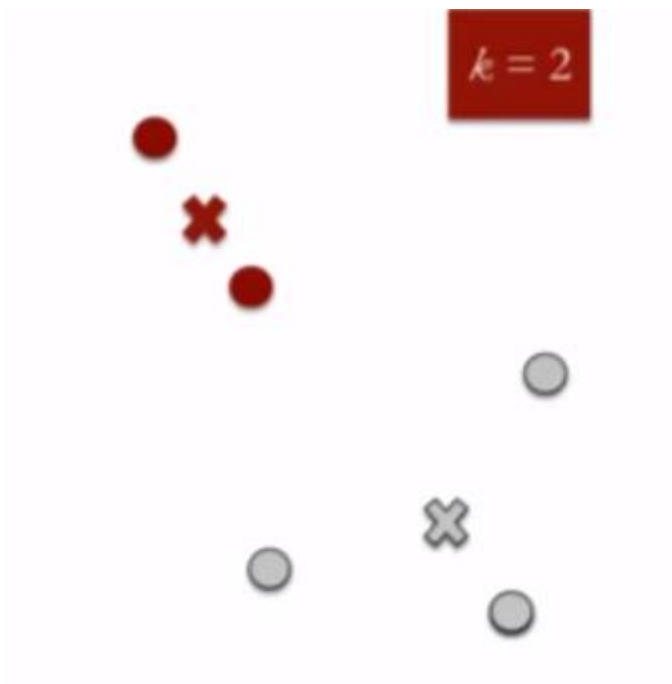
3. Compute cluster centroids : The centroid of data points in the red cluster is shown using red cross and those in grey cluster using grey cross.



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:

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.

Buisness Problem: Credit Card Segmentation

In [1]:
```
# importing all the necesary library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import pandas_profiling # For Overview of data-summary statistics with plots
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tools\_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the pu blic
API at pandas.testing instead.
                import pandas.util.testing as tm
```

These are most common library used use which ever necessary and explore it more

In [2]:
```
# ! pip install --user pandas_profiling
```

In [3]:
```
# !pip install --user joblib==0.14.1
```

In [4]:
```
df = pd.read_csv('credit-card-data.csv')
```

In [136]:
```
df.head()
```

Out[136]:

	cust_id	balance	balance_frequency	purchases	oneoff_purchases	installments_purchase
0	C10001	40.900749	0.818182	95.40	0.00	95.
1	C10002	3202.467416	0.909091	0.00	0.00	0.
2	C10003	2495.148862	1.000000	773.17	773.17	0.
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.
4	C10005	817.714335	1.000000	16.00	16.00	0.

5 rows × 23 columns

In [6]:
```
df.tail()
```
Out[6]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTA
8945	C19186	28.493517	1.000000	291.12	0.00	
8946	C19187	19.183215	1.000000	300.00	0.00	
8947	C19188	23.398673	0.833333	144.40	0.00	
8948	C19189	13.457564	0.833333	0.00	0.00	
8949	C19190	372.708075	0.666667	1093.25	1093.25	

In [7]:
```
df.shape
```
Out[7]:
```
(8950, 18)
```

In [8]:
```
df.dtypes
```
Out[8]:
```
CUST_ID                              object
BALANCE                             float64
BALANCE_FREQUENCY                   float64
PURCHASES                           float64
ONEOFF_PURCHASES                    float64
INSTALLMENTS_PURCHASES              float64
CASH_ADVANCE                        float64
PURCHASES_FREQUENCY                 float64
ONEOFF_PURCHASES_FREQUENCY          float64
PURCHASES_INSTALLMENTS_FREQUENCY    float64
CASH_ADVANCE_FREQUENCY              float64
CASH_ADVANCE_TRX                      int64
PURCHASES_TRX                         int64
CREDIT_LIMIT                        float64
PAYMENTS                            float64
MINIMUM_PAYMENTS                    float64
PRC_FULL_PAYMENT                    float64
TENURE                                int64
dtype: object
```

In [9]:
```
df.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
```

In [10]:
```
df.describe()
```

Out[10]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMEN
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	
std	2081.531879	0.236904	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	
max	19043.138560	1.000000	49039.570000	40761.250000	

In [11]:
```
##make the column name in lower case so that it is easy to type
df.columns=df.columns.str.lower()
```

In [12]:
```
df.describe().T
```
Out[12]:

	count	mean	std	min	25%	
balance	8950.0	1564.474828	2081.531879	0.000000	128.281915	87
balance_frequency	8950.0	0.877271	0.236904	0.000000	0.888889	
purchases	8950.0	1003.204834	2136.634782	0.000000	39.635000	36
oneoff_purchases	8950.0	592.437371	1659.887917	0.000000	0.000000	3
installments_purchases	8950.0	411.067645	904.338115	0.000000	0.000000	8
cash_advance	8950.0	978.871112	2097.163877	0.000000	0.000000	
purchases_frequency	8950.0	0.490351	0.401371	0.000000	0.083333	
oneoff_purchases_frequency	8950.0	0.202458	0.298336	0.000000	0.000000	
purchases_installments_frequency	8950.0	0.364437	0.397448	0.000000	0.000000	
cash_advance_frequency	8950.0	0.135144	0.200121	0.000000	0.000000	
cash_advance_trx	8950.0	3.248827	6.824647	0.000000	0.000000	
purchases_trx	8950.0	14.709832	24.857649	0.000000	1.000000	
credit_limit	8949.0	4494.449450	3638.815725	50.000000	1600.000000	300
payments	8950.0	1733.143852	2895.063757	0.000000	383.276166	85
minimum_payments	8637.0	864.206542	2372.446607	0.019163	169.123707	31
prc_full_payment	8950.0	0.153715	0.292499	0.000000	0.000000	
tenure	8950.0	11.517318	1.338331	6.000000	12.000000	1

before starting and preprocess to have a glance over your data set generally profiling wil be used

In [13]:
```python
import pandas_profiling
pandas_profiling.ProfileReport(df)
```

Out[13]:

Duplicate rows (%)	0.0%
Total size in memory	1.2 MiB
Average record size in memory	144.0 B

Variable types

NUM	17
CAT	1

Reproduction

Analysis started	2020-07-03 08:46:51.917883
Analysis finished	2020-07-03 08:49:33.748122
Duration	2 minutes and 41.83 seconds
Version	pandas-profiling v2.8.0 (https://github.com/pandas-profiling/pandas-profiling
Command line	`pandas_profiling --config_file config.yaml [YOUR_FILE.csv]`
Download configuration	config.yaml (data:text/plain;charset=utf-8,title%3A%20Pandas%20Profiling %200.95%0A%20%20%20%20%20%20%20%20%20skewness_threshold%3A

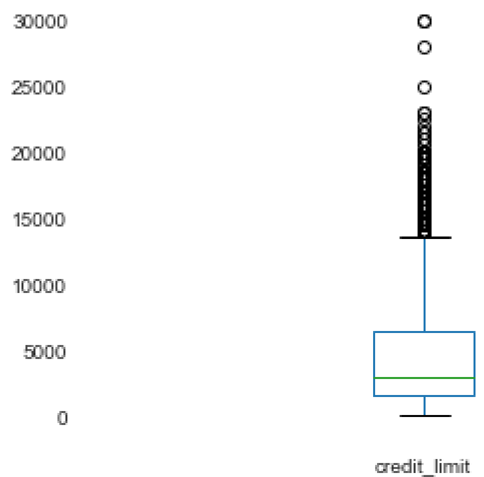
Warnings

`oneoff_purchases` is highly correlated with `purchases`	High correlation
`purchases` is highly correlated with `oneoff_purchases`	High correlation
`minimum_payments` has 313 (3.5%) missing values	Missing
`cust_id` has unique values	Unique
`purchases` has 2044 (22.8%) zeros	Zeros
`oneoff_purchases` has 4302 (48.1%) zeros	Zeros
`installments_purchases` has 3916 (43.8%) zeros	Zeros

In [14]:
```
##Check for the outlier in column ehich have null values& make the function to check the outlier
def boxplot(value):
    return value.plot.box()
```

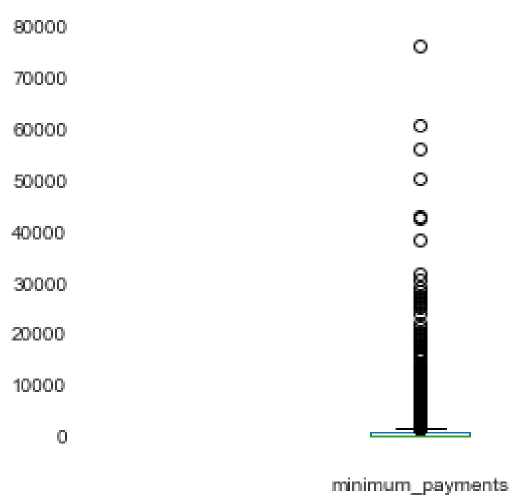
In [15]:
```
boxplot(df['credit_limit'])
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf8d9d7b8>



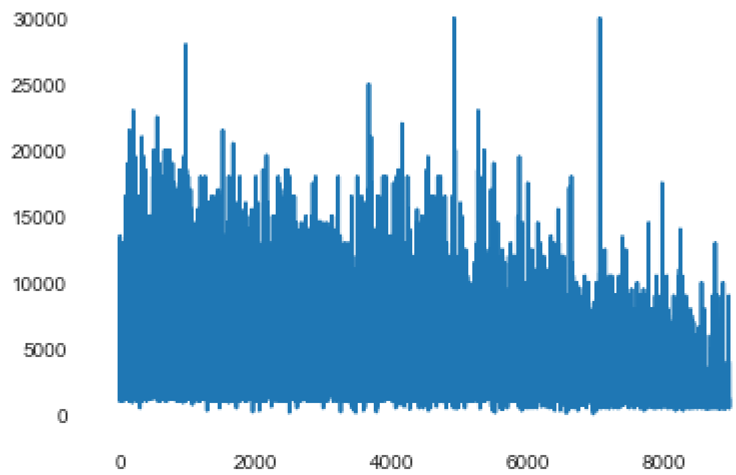
In [16]:
```
boxplot(df['minimum_payments'])
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf8d9db00>



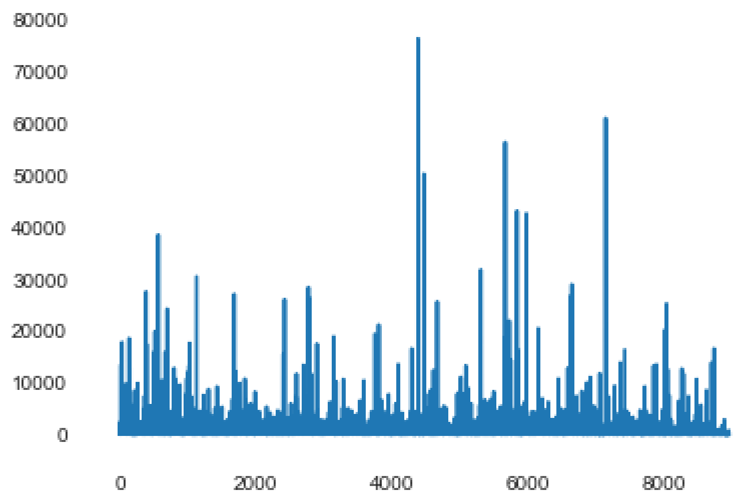
In [17]:
```
plt.plot(df['credit_limit'])
```

Out[17]: [<matplotlib.lines.Line2D at 0x23b80003198>]



In [18]:
```
plt.plot(df['minimum_payments'])
```

Out[18]: [<matplotlib.lines.Line2D at 0x23b80304c18>]



In [19]:
```
df.head()
```

Out[19]:

	cust_id	balance	balance_frequency	purchases	oneoff_purchases	installments_purchase
0	C10001	40.900749	0.818182	95.40	0.00	95.
1	C10002	3202.467416	0.909091	0.00	0.00	0.
2	C10003	2495.148862	1.000000	773.17	773.17	0.
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.
4	C10005	817.714335	1.000000	16.00	16.00	0.

In [20]
```
##imputation of na value
```

In [21]:
```
df['credit_limit'].median()
```

Out[21]:
```
3000.0
```

In [22]:
```
df['credit_limit'].fillna(df['credit_limit'].median(),inplace=True)
```

In [23]:
```
df.isna().sum()
```

Out[23]:
```
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                    313
prc_full_payment                      0
tenure                                0
dtype: int64
```

In [24]:
```
df['minimum_payments'].median()
```

Out[24]:
```
312.343947
```

In [25]:
```
df['minimum_payments'].fillna(df['minimum_payments'].median(),inplace =True)
```

In [26]:
```
df.isna().sum()
```

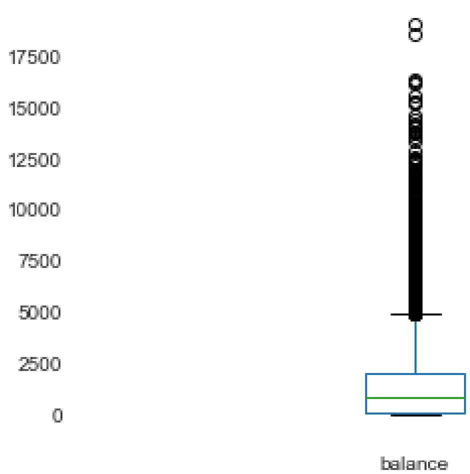
Out[26]:
```
cust_id                             0
balance                             0
balance_frequency                   0
purchases                           0
oneoff_purchases                    0
installments_purchases              0
cash_advance                        0
purchases_frequency                 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 [27]:
```
##Visulaisation
```

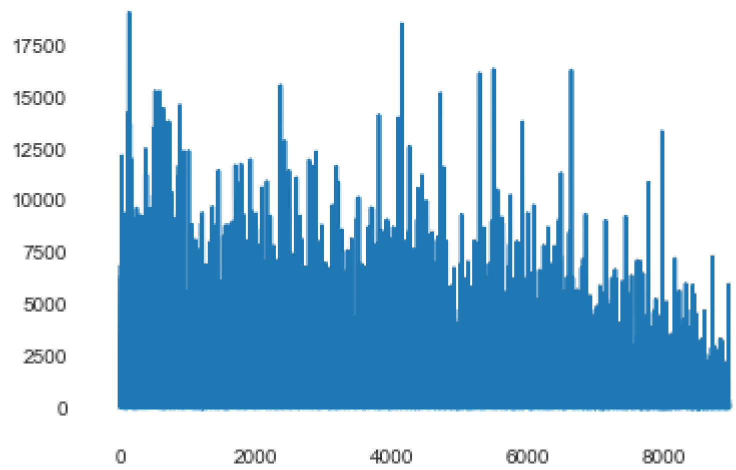
In [28]:
```
def boxplot(value):
    return value.plot.box()
```

In [29]:
```
boxplot(df['balance']) ##We can see there are many outlier
```

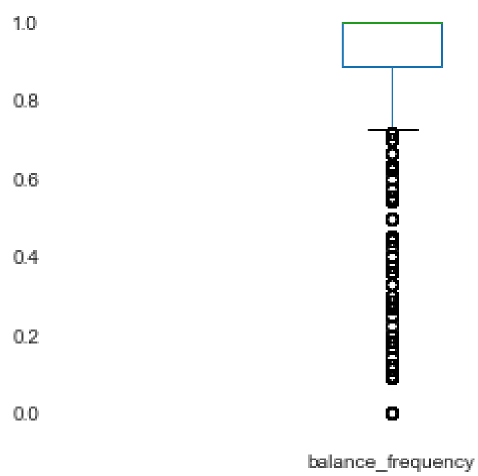
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x23b8031e358>



In [30]:
```
plt.plot(df['balance'])
```
Out[30]: [<matplotlib.lines.Line2D at 0x23b800e72e8>]

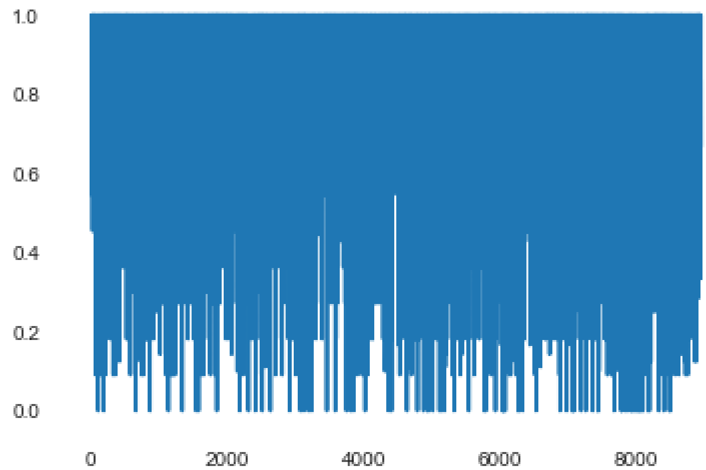


In [31]:
```
boxplot(df['balance_frequency'])
```
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x23b80317978>



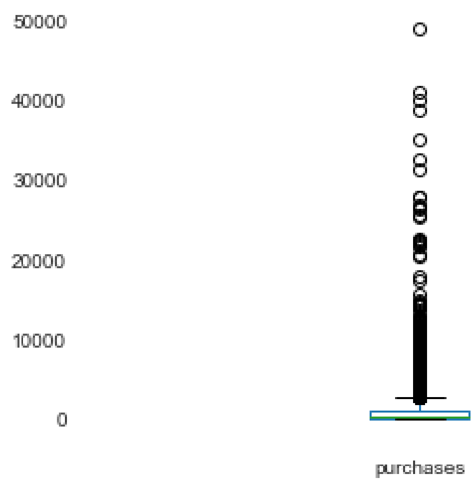
In [32]:
```
plt.plot(df['balance_frequency'])
```

Out[32]: [<matplotlib.lines.Line2D at 0x23b80192940>]



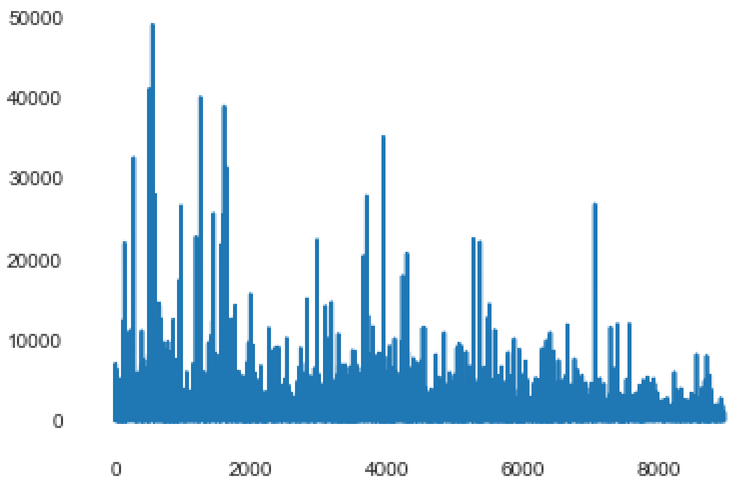
In [33]:
```
boxplot(df['purchases'])
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x23b801c8438>



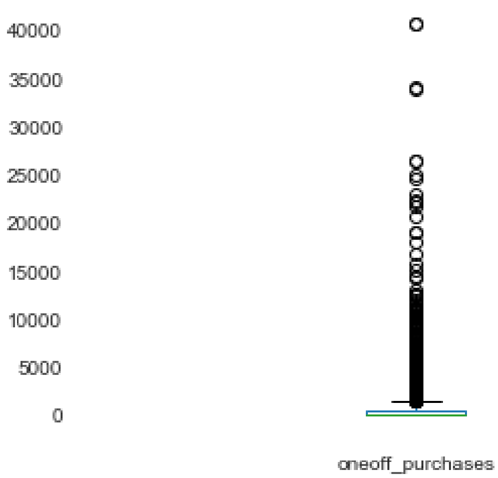
In [34]:
```
plt.plot(df['purchases'])
```

Out[34]: [<matplotlib.lines.Line2D at 0x23b80242860>]

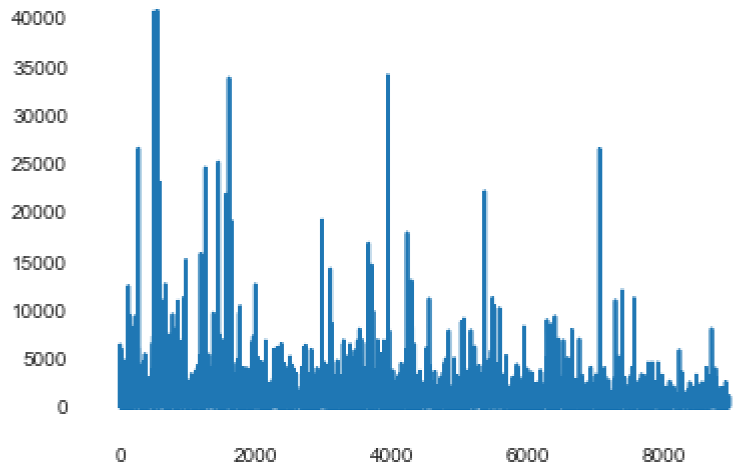


In [35]:
```
boxplot(df['oneoff_purchases'])
```

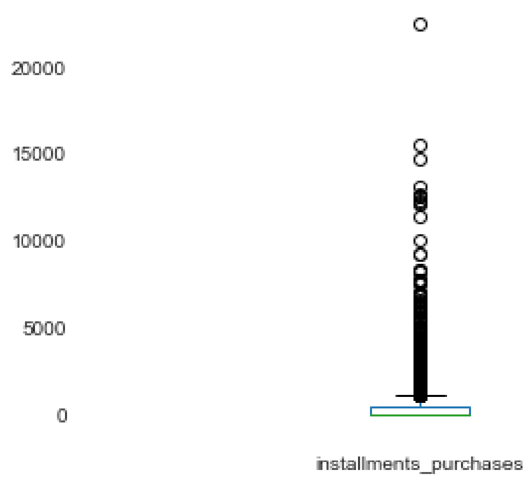
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x23b8026eb70>



In [36]:
```
plt.plot(df['oneoff_purchases'])
```
Out[36]: [<matplotlib.lines.Line2D at 0x23b815ed390>]

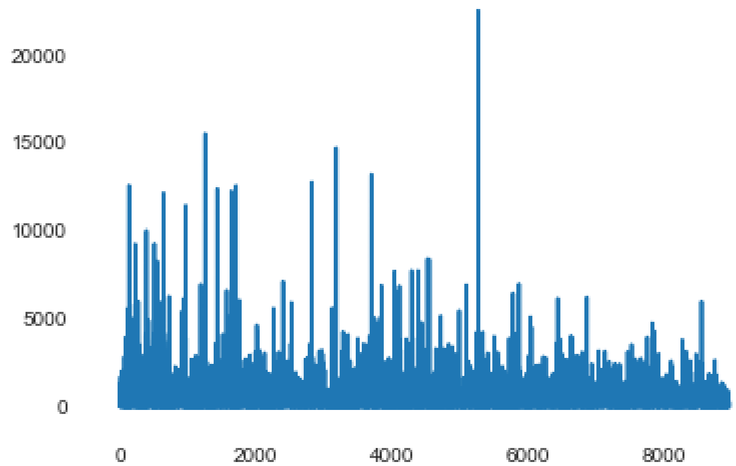


In [37]:
```
boxplot(df['installments_purchases'])
```
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x23b8160d828>



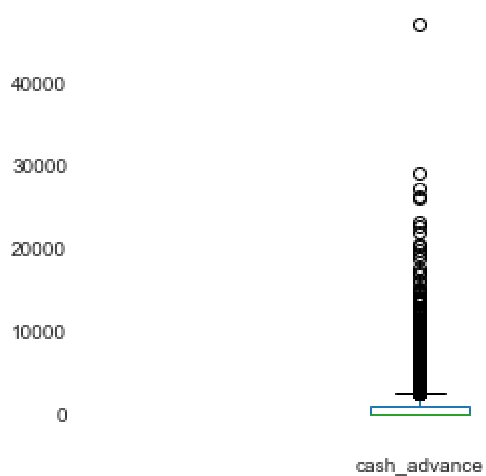
In [38]:
```
plt.plot(df['installments_purchases'])
```

Out[38]: [<matplotlib.lines.Line2D at 0x23b8169f9b0>]



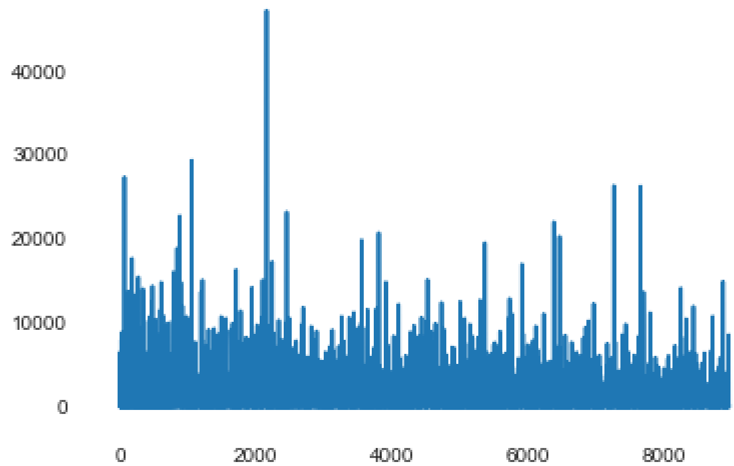
In [39]:
```
boxplot(df['cash_advance'])
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x23b816d9ef0>



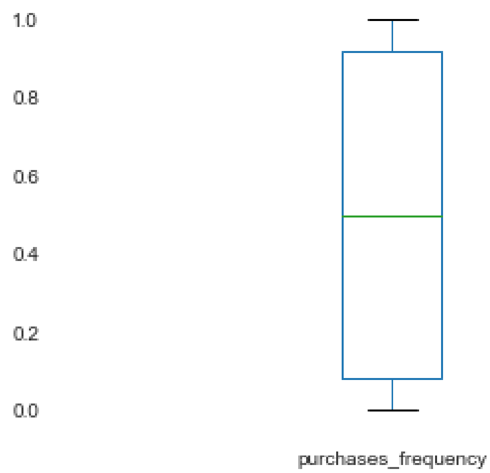
In [40]:
```
plt.plot(df['cash_advance'])
```

Out[40]: [<matplotlib.lines.Line2D at 0x23b81746fd0>]

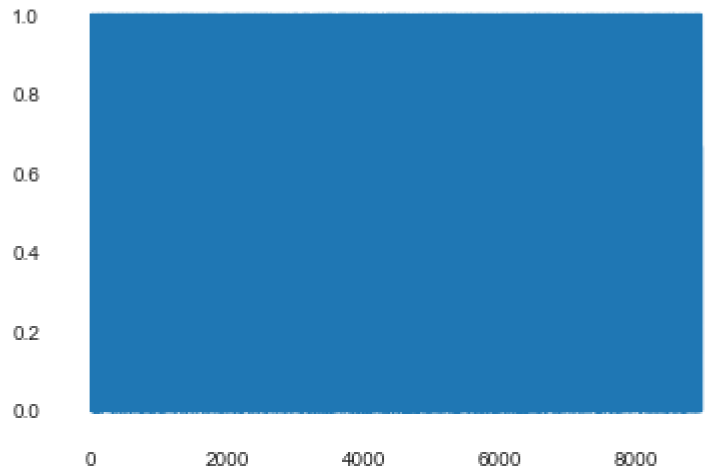


In [41]:
```
boxplot(df['purchases_frequency']) ## In this variable there is no outlier
```

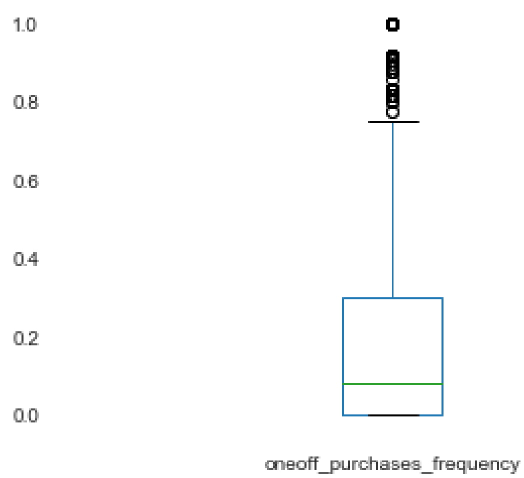
Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x23b817820b8>



In [42]:
```
plt.plot(df['purchases_frequency'])
```
Out[42]: [<matplotlib.lines.Line2D at 0x23bf8db35c0>]

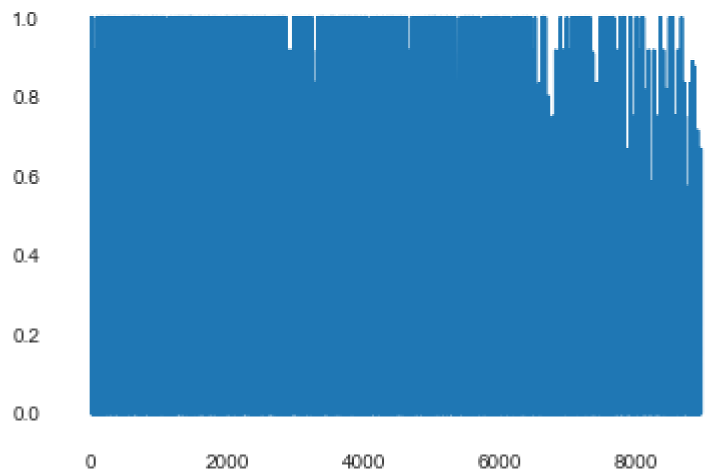


In [43]:
```
boxplot(df['oneoff_purchases_frequency'])
```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf8dc87f0>



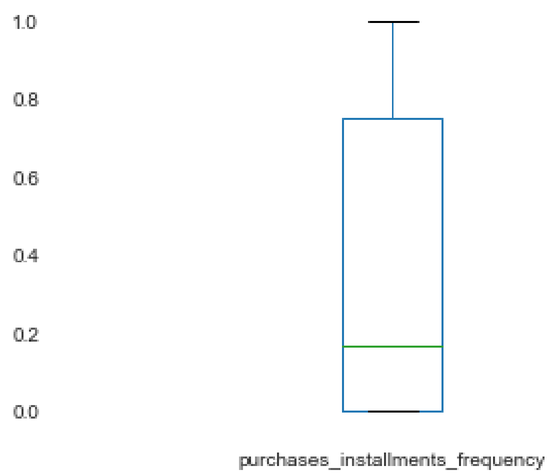
In [44]:
```
plt.plot(df['oneoff_purchases_frequency'])
```

Out[44]: [<matplotlib.lines.Line2D at 0x23bf7b81d30>]



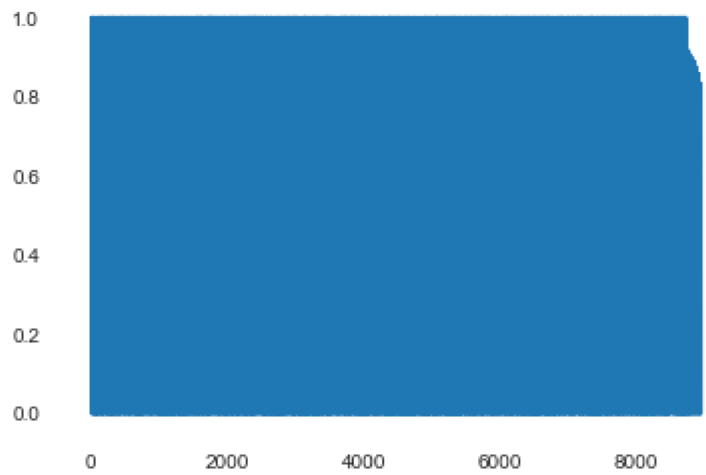
In [45]:
```
boxplot(df['purchases_installments_frequency'])
```

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf799fa90>



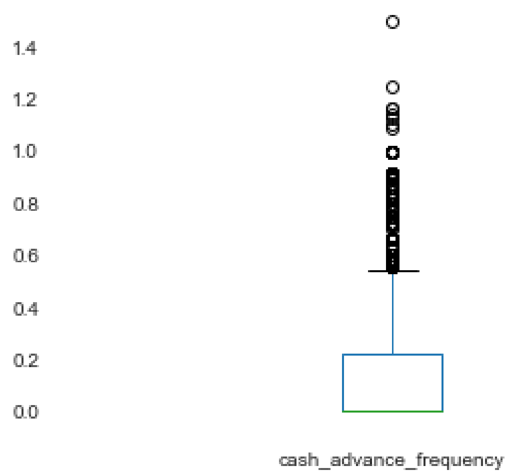
In [46]:
```
plt.plot(df['purchases_installments_frequency'])
```

Out[46]: [<matplotlib.lines.Line2D at 0x23bf76c7e80>]



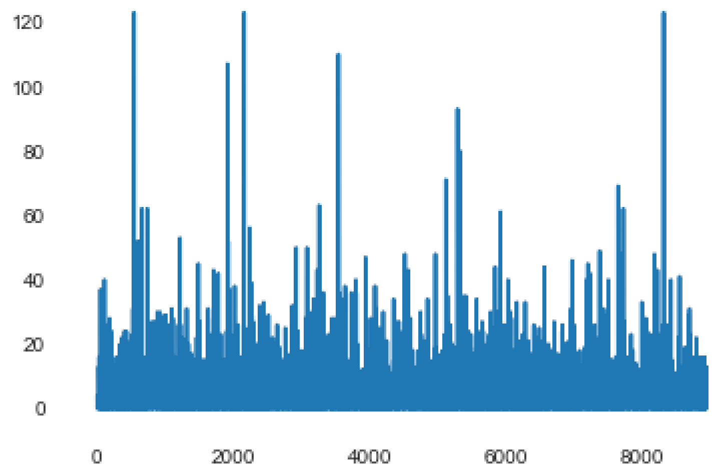
In [47]:
```
boxplot(df['cash_advance_frequency'])
```

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf7811518>



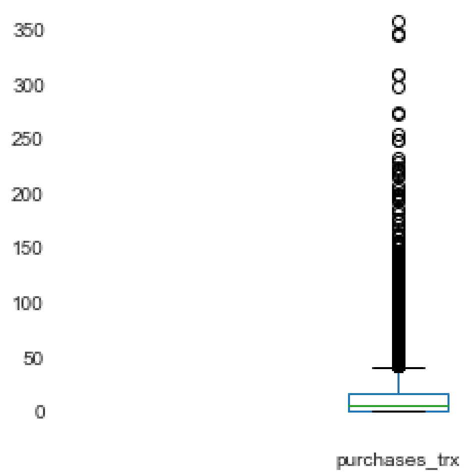
In [48]:
```
plt.plot(df['cash_advance_trx'])
```

Out[48]: [<matplotlib.lines.Line2D at 0x23bf73e4a90>]

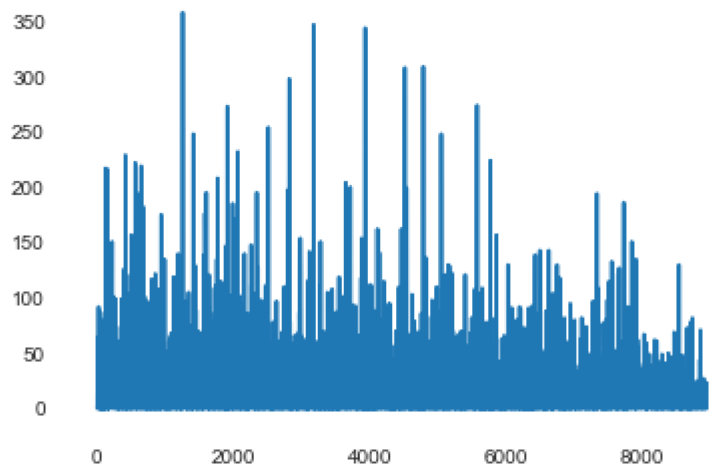


In [49]:
```
boxplot(df['purchases_trx'])
```

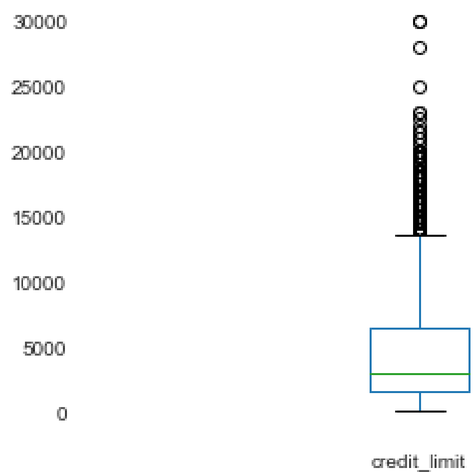
Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf72f3b70>



In [50]:
```
plt.plot(df['purchases_trx'])
```
Out[50]: [<matplotlib.lines.Line2D at 0x23bf5feb0b8>]

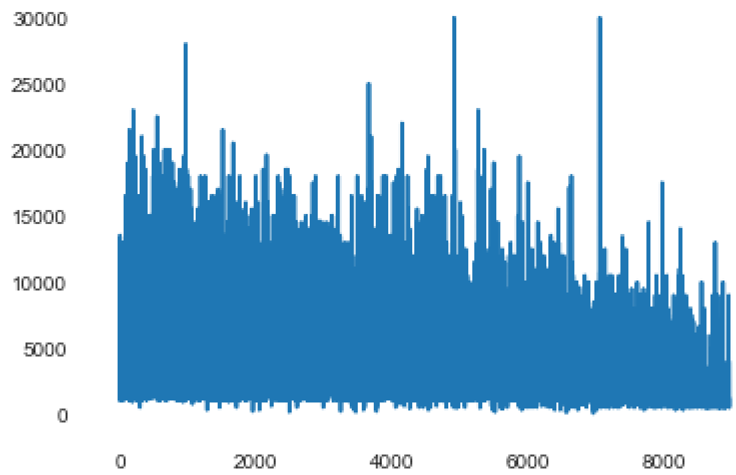


In [51]:
```
boxplot(df['credit_limit'])
```
Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf5f45b38>



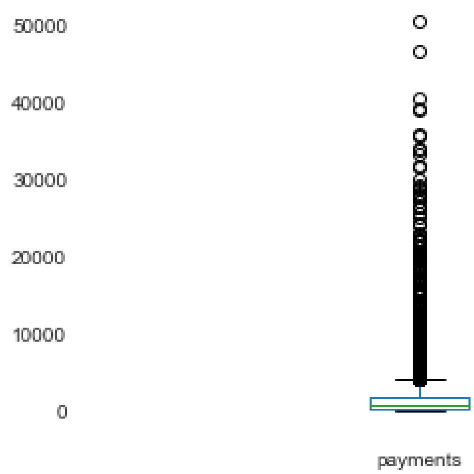
In [52]:
```
plt.plot(df['credit_limit'])
```

Out[52]: [<matplotlib.lines.Line2D at 0x23bf535f518>]

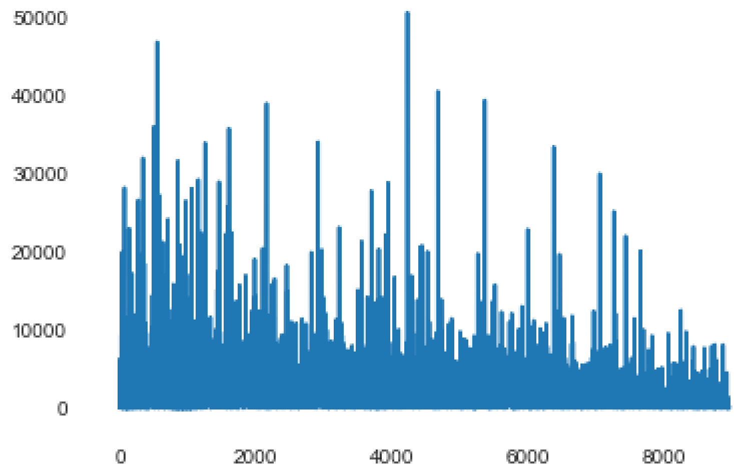


In [53]:
```
boxplot(df['payments'])
```

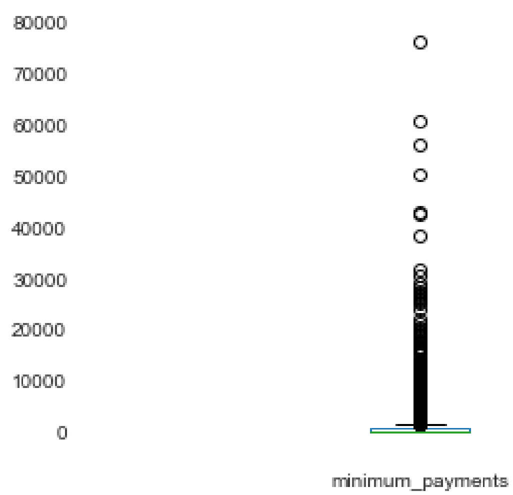
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf6022a90>



In [54]:
```
plt.plot(df['payments'])
```
Out[54]: [<matplotlib.lines.Line2D at 0x23bf506e240>]

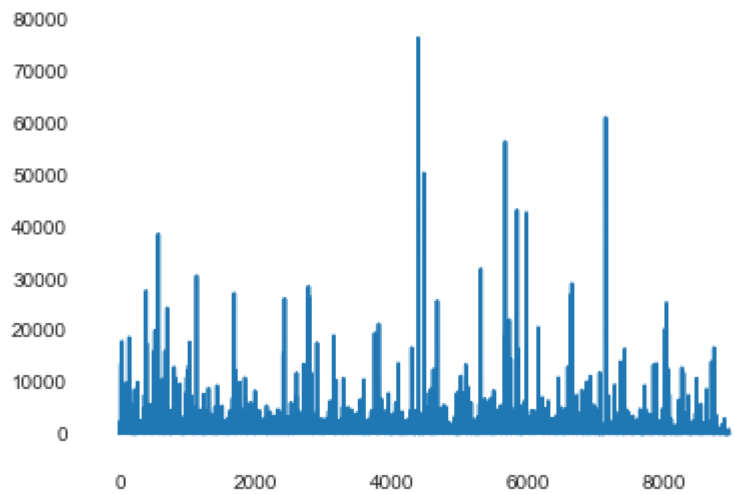


In [55]:
```
boxplot(df['minimum_payments'])
```
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf5450128>



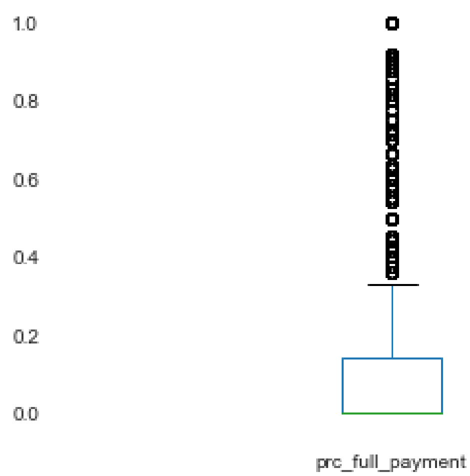
In [56]:
```
plt.plot(df['minimum_payments'])
```

Out[56]: [<matplotlib.lines.Line2D at 0x23bf4f61240>]

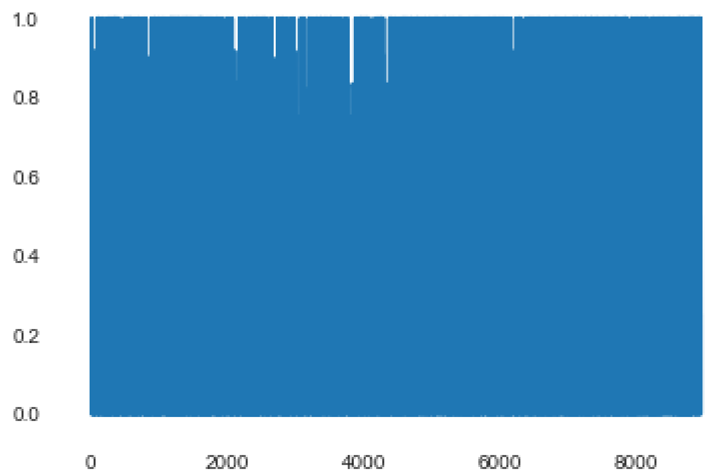


In [57]:
```
boxplot(df['prc_full_payment'])
```

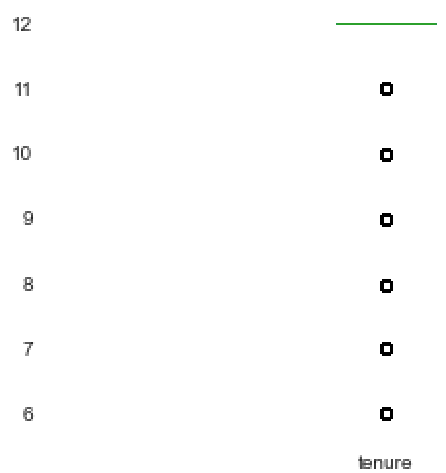
Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf4f434a8>



In [58]:
```
plt.plot(df['prc_full_payment'])
```
Out[58]: [<matplotlib.lines.Line2D at 0x23bf758f518>]

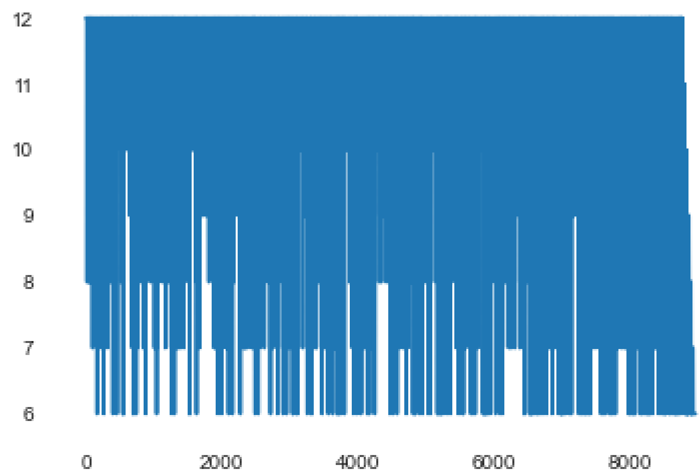


In [59]:
```
boxplot(df['tenure'])
```
Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf5054b00>



In [60]:
```
plt.plot(df['tenure'])
```

Out[60]: [<matplotlib.lines.Line2D at 0x23bf7abd7f0>]



Building KPI(Key Performance Indicator)

In [61]:
```
df['monthly_avg_purchase']=df['purchases']/df['tenure']
df['monthly_avg_purchase'].head()
```

Out[61]:
```
0      7.950000
1      0.000000
2     64.430833
3    124.916667
4      1.333333
Name: monthly_avg_purchase, dtype: float64
```

In [62]:
```
df['monthly_cash_advance']=df['cash_advance']/df['tenure']
df['monthly_cash_advance'].head()
```

Out[62]:
```
0      0.000000
1    536.912124
2      0.000000
3     17.149001
4      0.000000
Name: monthly_cash_advance, dtype: float64
```

In [63]:
```
df['oneoff_purchases'].value_counts() ##record of oneoff_purchase equal to zer
o most frequent
```

Out[63]:
```
0.00       4302
45.65        46
50.00        17
200.00       15
60.00        13
           ...
2814.23       1
2281.56       1
458.00        1
155.00        1
153.11        1
Name: oneoff_purchases, Length: 4014, dtype: int64
```

Puchase Type

In [64]:
```
#for customers who do only oneoff_purchases
df[(df['oneoff_purchases']>0) & (df['installments_purchases']==0)].shape
```

Out[64]: (1874, 20)

In [65]:
```
#For customers who do only installment purchases
df[(df['oneoff_purchases']==0) & (df['installments_purchases']>0)].shape
```

Out[65]: (2260, 20)

In [66]:
```
#For the customers who do both one-off purchases and installment purchases
df[(df['oneoff_purchases']>0) & (df['installments_purchases']>0)].shape
```

Out[66]: (2774, 20)

In [67]:
```
#For the customers neither do one-off purchases nor installment purchases
df[(df['oneoff_purchases']==0) & (df['installments_purchases']==0)].shape
```

Out[67]: (2042, 20)

Insights and observation There are four type of customer in the entire dataset on the basis of transaction 1.Customer who do only oneoff_purchase transactions 2.Customer who do only installments purchase Transactions 3.Customer who do both oneoff and installments transactions 4.Customer who neither do oneff and nor installments transactions

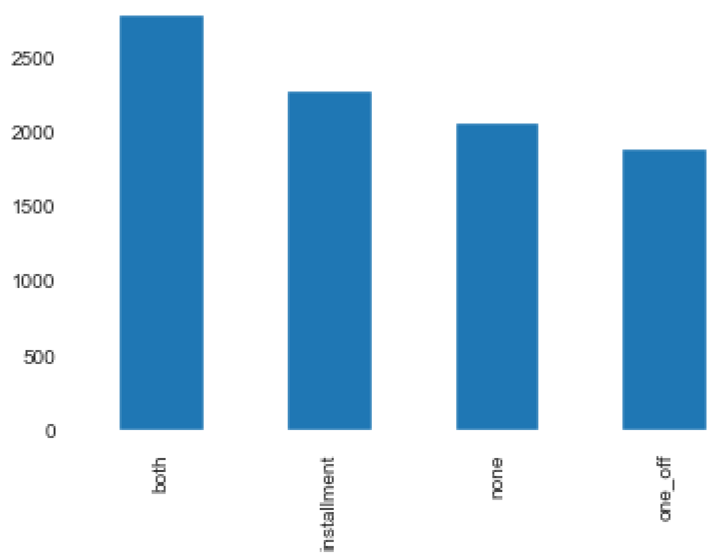
In [68]:
```
#Writing a function for all the transaction types
def transaction(df):
    if (df['oneoff_purchases']>0) & (df['installments_purchases']==0):
        return 'one_off'
    if (df['oneoff_purchases']==0) & (df['installments_purchases']>0):
        return 'installment'
    if (df['oneoff_purchases']>0) & (df['installments_purchases']>0):
         return 'both'
    if (df['oneoff_purchases']==0) & (df['installments_purchases']==0):
        return 'none'
```

In [69]:
```
#Creating label type for the transaction types
df['transaction_type']=df.apply(transaction,axis=1)
df['transaction_type'].value_counts()
```

Out[69]:
```
both           2774
installment    2260
none           2042
one_off        1874
Name: transaction_type, dtype: int64
```

In [70]:
```
df['transaction_type'].value_counts().plot.bar()
```

Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x23b802bf208>



In [71]:
```
#Finding the limit usage for each customer
df['limit_usage']=df.apply(lambda x: x['balance']/x['credit_limit'], axis=1)
df['limit_usage'].head()
```

Out[71]:
```
0    0.040901
1    0.457495
2    0.332687
3    0.222223
4    0.681429
Name: limit_usage, dtype: float64
```

In [72]:
```
#finding Payment to minimum payments Ratio
df['payment_minpay']=df.apply(lambda x:x['payments']/x['minimum_payments'],axi
s=1)
df['payment_minpay'].describe().T
```

Out[72]:
```
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 [73]:
```
df.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  transaction_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
```

In [74]:
```
##log transformation
transform=df.drop(['cust_id','transaction_type'],axis=1).applymap(lambda x: np
.log(x+1))
```

In [75]:
```
transform.describe().T
```

Out[75]:

	count	mean	std	min	25%	50%	
balance	8950.0	6.161637	2.013303	0.000000	4.861995	6.773521	7.62
balance_frequency	8950.0	0.619940	0.148590	0.000000	0.635989	0.693147	0.69
purchases	8950.0	4.899647	2.916872	0.000000	3.704627	5.892417	7.01
oneoff_purchases	8950.0	3.204274	3.246365	0.000000	0.000000	3.663562	6.36
installments_purchases	8950.0	3.352403	3.082973	0.000000	0.000000	4.499810	6.15
cash_advance	8950.0	3.319086	3.566298	0.000000	0.000000	0.000000	7.01
purchases_frequency	8950.0	0.361268	0.277317	0.000000	0.080042	0.405465	0.65
oneoff_purchases_frequency	8950.0	0.158699	0.216672	0.000000	0.000000	0.080042	0.26
purchases_installments_frequency	8950.0	0.270072	0.281852	0.000000	0.000000	0.154151	0.55
cash_advance_frequency	8950.0	0.113512	0.156716	0.000000	0.000000	0.000000	0.20
cash_advance_trx	8950.0	0.817570	1.009316	0.000000	0.000000	0.000000	1.60
purchases_trx	8950.0	1.894731	1.373856	0.000000	0.693147	2.079442	2.89
credit_limit	8950.0	8.094825	0.819629	3.931826	7.378384	8.006701	8.7
payments	8950.0	6.624540	1.591763	0.000000	5.951361	6.754489	7.55
minimum_payments	8950.0	5.916079	1.169929	0.018982	5.146667	5.747301	6.67
prc_full_payment	8950.0	0.117730	0.211617	0.000000	0.000000	0.000000	0.13
tenure	8950.0	2.519680	0.130367	1.945910	2.564949	2.564949	2.56
monthly_avg_purchase	8950.0	3.050877	2.002823	0.000000	1.481458	3.494587	4.58
monthly_cash_advance	8950.0	2.163970	2.429741	0.000000	0.000000	0.000000	4.60
limit_usage	8950.0	0.296081	0.250303	0.000000	0.040656	0.264455	0.5
payment_minpay	8950.0	1.357600	0.940149	0.000000	0.648817	1.109459	1.95

In [76]:
```
col=['balance','purchases','cash_advance','tenure','payments','minimum_payment
s','prc_full_payment','credit_limit']
transform_pre=transform[[x for x in transform.columns if x not in col ]]
```

In [77]:
```
transform_pre.describe().T
```

Out[77]:

	count	mean	std	min	25%	50%	75%
balance_frequency	8950.0	0.619940	0.148590	0.0	0.635989	0.693147	0.693147
oneoff_purchases	8950.0	3.204274	3.246365	0.0	0.000000	3.663562	6.360274
installments_purchases	8950.0	3.352403	3.082973	0.0	0.000000	4.499810	6.151961
purchases_frequency	8950.0	0.361268	0.277317	0.0	0.080042	0.405465	0.650588
oneoff_purchases_frequency	8950.0	0.158699	0.216672	0.0	0.000000	0.080042	0.262364
purchases_installments_frequency	8950.0	0.270072	0.281852	0.0	0.000000	0.154151	0.559616
cash_advance_frequency	8950.0	0.113512	0.156716	0.0	0.000000	0.000000	0.200671
cash_advance_trx	8950.0	0.817570	1.009316	0.0	0.000000	0.000000	1.609438
purchases_trx	8950.0	1.894731	1.373856	0.0	0.693147	2.079442	2.890372
monthly_avg_purchase	8950.0	3.050877	2.002823	0.0	1.481458	3.494587	4.587295
monthly_cash_advance	8950.0	2.163970	2.429741	0.0	0.000000	0.000000	4.606022
limit_usage	8950.0	0.296081	0.250303	0.0	0.040656	0.264455	0.540911
payment_minpay	8950.0	1.357600	0.940149	0.0	0.648817	1.109459	1.953415

Finding the insights frm the data

In [78]:
```
df[df['transaction_type']=='n']
```

Out[78]:

cust_id	balance	balance_frequency	purchases	oneoff_purchases	installments_purchases	ca

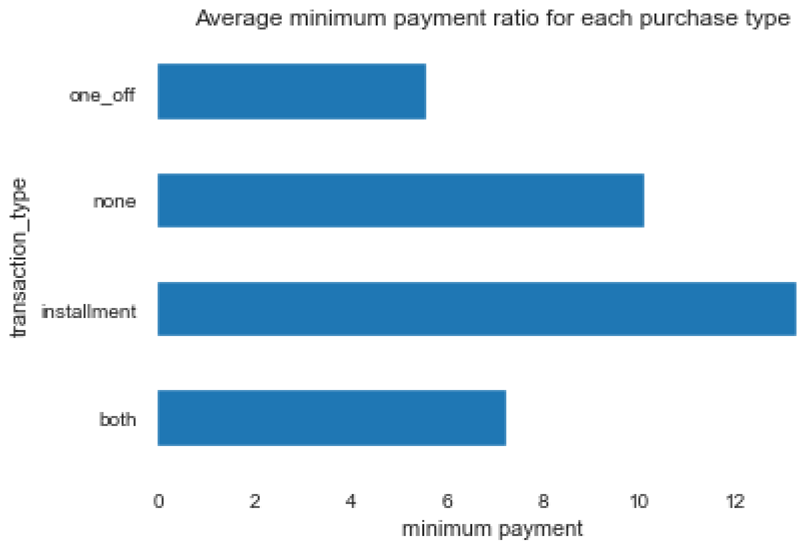
0 rows × 23 columns

In [79]:
```
# Average payment_minpayment ratio for each purchse type.
x=df.groupby('transaction_type').apply(lambda x: np.mean(x['payment_minpay']))
type(x)
x.values
```

Out[79]:
```
array([ 7.23698216, 13.2590037 , 10.08745106,  5.57108156])
```

In [80]:
```
df.groupby('transaction_type').apply(lambda x: np.mean(x['payment_minpay'])).p
lot.barh()
plt.title('Average minimum payment ratio for each purchase type')
plt.xlabel('minimum payment')
```

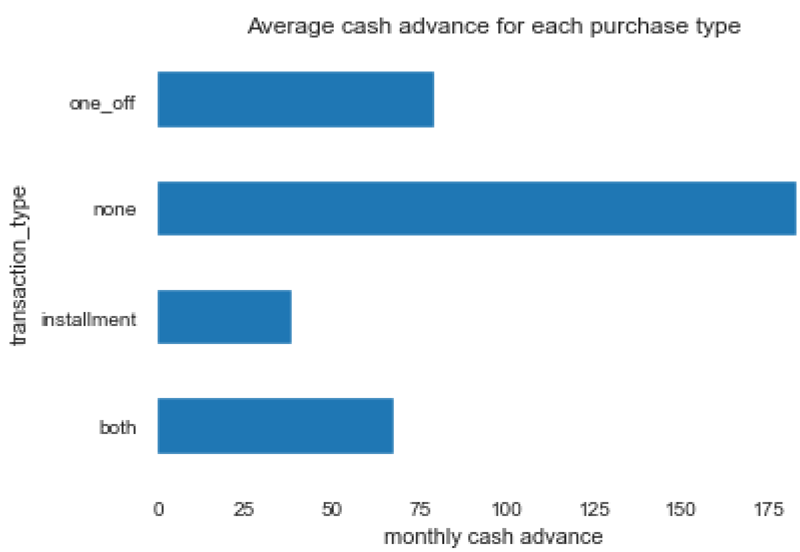
Out[80]: Text(0.5, 0, 'minimum payment')



Customers who make transactions in installments are paying the amount regularly

In [81]:
```
df.groupby('transaction_type').apply(lambda x: np.mean(x['monthly_cash_advanc
e'])).plot.barh()
plt.title('Average cash advance for each purchase type')
plt.xlabel('monthly cash advance')
```

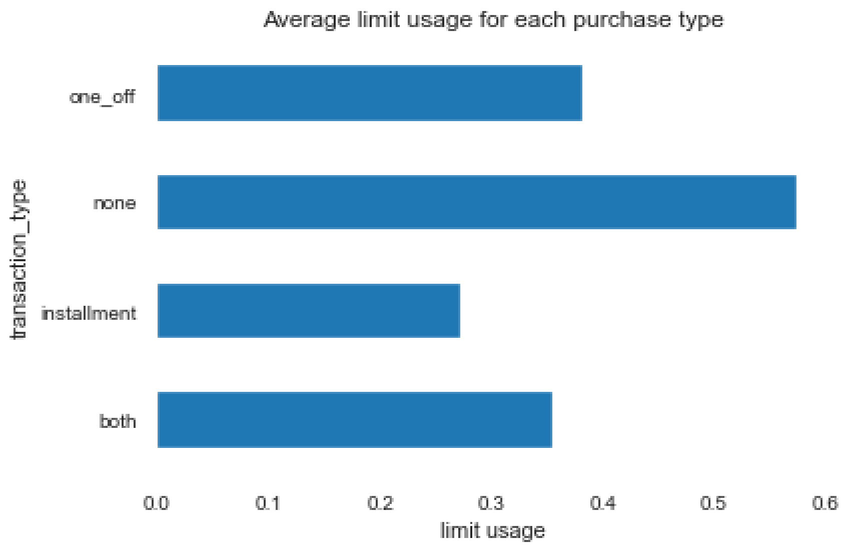
Out[81]: Text(0.5, 0, 'monthly cash advance')



2. Customers neither make a transaction in one_off payments nor installments are having high monthly cash advances

In [82]:
```
df.groupby('transaction_type').apply(lambda x: np.mean(x['limit_usage'])).plot
.barh()
plt.title('Average limit usage for each purchase type')
plt.xlabel('limit usage')
```

Out[82]: Text(0.5, 0, 'limit usage')



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

Dataset preparation set for model

In [83]:
```
df_original=pd.concat([df,pd.get_dummies(df['transaction_type'])],axis=1)
```

In [84]:
```
df_original.describe().T
```

Out[84]:

	count	mean	std	min	25%	
balance	8950.0	1564.474828	2081.531879	0.000000	128.281915	87
balance_frequency	8950.0	0.877271	0.236904	0.000000	0.888889	
purchases	8950.0	1003.204834	2136.634782	0.000000	39.635000	36
oneoff_purchases	8950.0	592.437371	1659.887917	0.000000	0.000000	3
installments_purchases	8950.0	411.067645	904.338115	0.000000	0.000000	8
cash_advance	8950.0	978.871112	2097.163877	0.000000	0.000000	
purchases_frequency	8950.0	0.490351	0.401371	0.000000	0.083333	
oneoff_purchases_frequency	8950.0	0.202458	0.298336	0.000000	0.000000	
purchases_installments_frequency	8950.0	0.364437	0.397448	0.000000	0.000000	
cash_advance_frequency	8950.0	0.135144	0.200121	0.000000	0.000000	
cash_advance_trx	8950.0	3.248827	6.824647	0.000000	0.000000	
purchases_trx	8950.0	14.709832	24.857649	0.000000	1.000000	
credit_limit	8950.0	4494.282473	3638.646702	50.000000	1600.000000	300
payments	8950.0	1733.143852	2895.063757	0.000000	383.276166	85
minimum_payments	8950.0	844.906767	2332.792322	0.019163	170.857654	31
prc_full_payment	8950.0	0.153715	0.292499	0.000000	0.000000	
tenure	8950.0	11.517318	1.338331	6.000000	12.000000	1
monthly_avg_purchase	8950.0	86.175173	180.508787	0.000000	3.399375	3
monthly_cash_advance	8950.0	88.977984	193.136115	0.000000	0.000000	
limit_usage	8950.0	0.388884	0.389722	0.000000	0.041494	
payment_minpay	8950.0	9.059164	118.180526	0.000000	0.913275	
both	8950.0	0.309944	0.462496	0.000000	0.000000	
installment	8950.0	0.252514	0.434479	0.000000	0.000000	
none	8950.0	0.228156	0.419667	0.000000	0.000000	
one_off	8950.0	0.209385	0.406893	0.000000	0.000000	

In [85]:
```
df.head()
```

Out[85]:

	cust_id	balance	balance_frequency	purchases	oneoff_purchases	installments_purchase
0	C10001	40.900749	0.818182	95.40	0.00	95.
1	C10002	3202.467416	0.909091	0.00	0.00	0.
2	C10003	2495.148862	1.000000	773.17	773.17	0.
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.
4	C10005	817.714335	1.000000	16.00	16.00	0.

5 rows × 23 columns

In [86]:
```
transform_pre['transaction_type']=df.loc[:,'transaction_type']
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [87]:
```
transform_pre.head()
```

Out[87]:

	balance_frequency	oneoff_purchases	installments_purchases	purchases_frequency	oneoff_pu
0	0.597837	0.000000	4.568506	0.154151	
1	0.646627	0.000000	0.000000	0.000000	
2	0.693147	6.651791	0.000000	0.693147	
3	0.492477	7.313220	0.000000	0.080042	
4	0.693147	2.833213	0.000000	0.080042	

In [88]:
```
df_dummy=pd.concat([transform_pre,pd.get_dummies(transform_pre['transaction_ty
pe'])],axis=1)
df_dummy.head()
```

Out[88]:

	balance_frequency	oneoff_purchases	installments_purchases	purchases_frequency	oneoff_pu
0	0.597837	0.000000	4.568506	0.154151	
1	0.646627	0.000000	0.000000	0.000000	
2	0.693147	6.651791	0.000000	0.693147	
3	0.492477	7.313220	0.000000	0.080042	
4	0.693147	2.833213	0.000000	0.080042	

In [89]:
```
df_dummy=df_dummy.drop(['transaction_type'],axis=1)
```

In [90]:
```
df_dummy.head()
```

Out[90]:

	balance_frequency	oneoff_purchases	installments_purchases	purchases_frequency	oneoff_pu
0	0.597837	0.000000	4.568506	0.154151	
1	0.646627	0.000000	0.000000	0.000000	
2	0.693147	6.651791	0.000000	0.693147	
3	0.492477	7.313220	0.000000	0.080042	
4	0.693147	2.833213	0.000000	0.080042	

In [91]:
```
df_dummy.describe().T
```

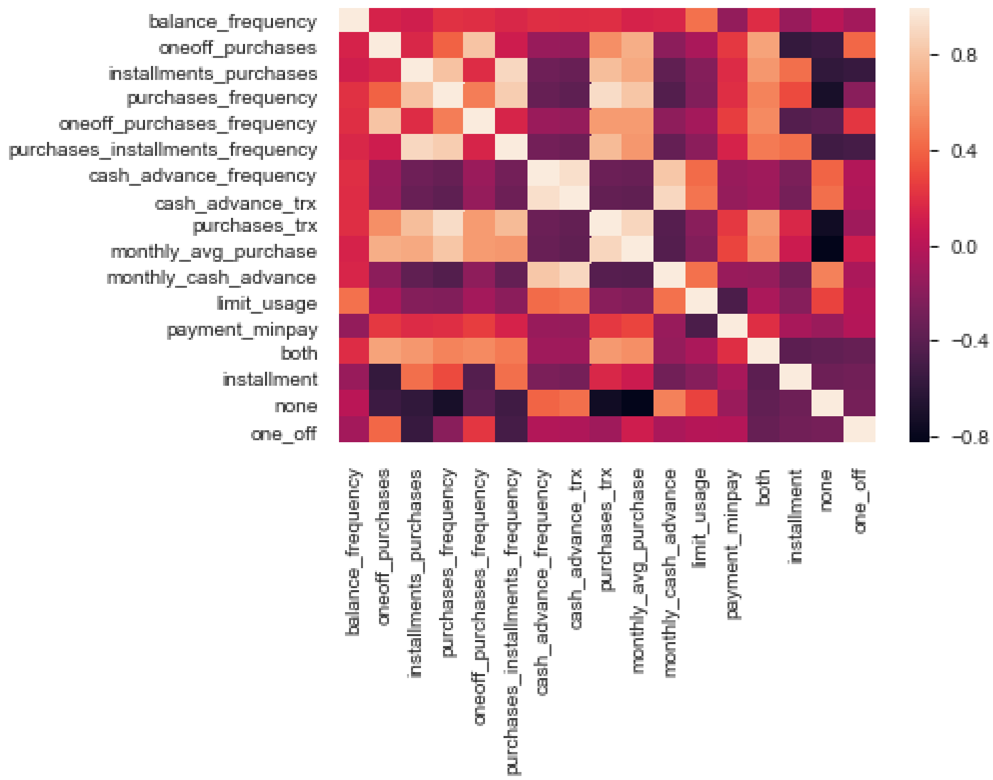
Out[91]:

	count	mean	std	min	25%	50%	75%
balance_frequency	8950.0	0.619940	0.148590	0.0	0.635989	0.693147	0.693147
oneoff_purchases	8950.0	3.204274	3.246365	0.0	0.000000	3.663562	6.360274
installments_purchases	8950.0	3.352403	3.082973	0.0	0.000000	4.499810	6.151961
purchases_frequency	8950.0	0.361268	0.277317	0.0	0.080042	0.405465	0.650588
oneoff_purchases_frequency	8950.0	0.158699	0.216672	0.0	0.000000	0.080042	0.262364
purchases_installments_frequency	8950.0	0.270072	0.281852	0.0	0.000000	0.154151	0.559616
cash_advance_frequency	8950.0	0.113512	0.156716	0.0	0.000000	0.000000	0.200671
cash_advance_trx	8950.0	0.817570	1.009316	0.0	0.000000	0.000000	1.609438
purchases_trx	8950.0	1.894731	1.373856	0.0	0.693147	2.079442	2.890372
monthly_avg_purchase	8950.0	3.050877	2.002823	0.0	1.481458	3.494587	4.587295
monthly_cash_advance	8950.0	2.163970	2.429741	0.0	0.000000	0.000000	4.606022
limit_usage	8950.0	0.296081	0.250303	0.0	0.040656	0.264455	0.540911
payment_minpay	8950.0	1.357600	0.940149	0.0	0.648817	1.109459	1.953415
both	8950.0	0.309944	0.462496	0.0	0.000000	0.000000	1.000000
installment	8950.0	0.252514	0.434479	0.0	0.000000	0.000000	1.000000
none	8950.0	0.228156	0.419667	0.0	0.000000	0.000000	0.000000
one_off	8950.0	0.209385	0.406893	0.0	0.000000	0.000000	0.000000

Finding the correlation among the variables in dataset

In [92]:
```
sns.heatmap(df_dummy.corr())
```

Out[92]: <matplotlib.axes._subplots.AxesSubplot at 0x23bf5f25668>



Observations:

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 use Principal Component Analysis technique, but before we use this we must make sure that the data available in the dataset have no weightage issues. So we use standard scaler technique if there are any weightage issues among the variables of the dataset.

choosing pca model

In [93]:
```
from sklearn.preprocessing import  StandardScaler
sc=StandardScaler()
df_scaled=sc.fit_transform(df_dummy)
```

In [94]:
```
from sklearn.decomposition import PCA
var_ratio={}
for n in range(4,15):
    pc=PCA(n_components=n)
    df_pca=pc.fit(df_scaled)
    var_ratio[n]=sum(df_pca.explained_variance_ratio_)
```

In [95]:
```
type(df_pca)
```

Out[95]: sklearn.decomposition.pca.PCA

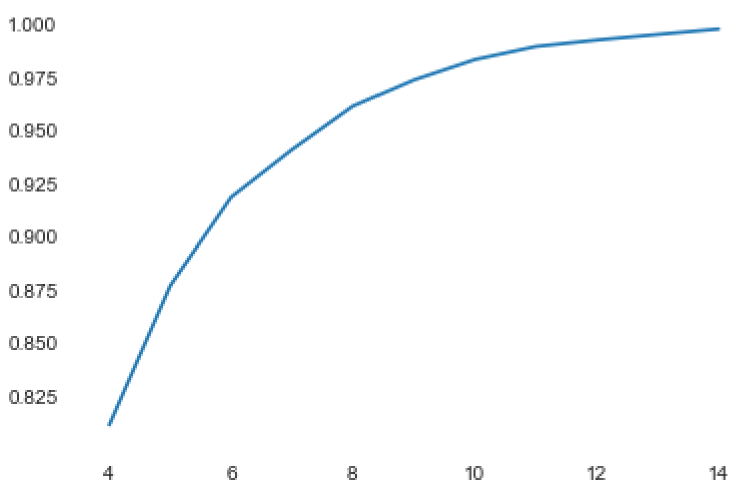
In [96]:
```
var_ratio
```

Out[96]:
```
{4: 0.8115442762351261,
 5: 0.8770555795291429,
 6: 0.9186492443512606,
 7: 0.9410925256030125,
 8: 0.961611405368306,
 9: 0.9739787081990646,
 10: 0.9835896584630703,
 11: 0.9897248107341954,
 12: 0.9927550009135223,
 13: 0.995390756238542,
 14: 0.9979616898169593}
```

Observation: From the above variance ratio we can see that the maximum variance is available when the number of components are 5. Hence we choose n_components as 5 to reduce the dimensionality in the datset

In [97]:
```
pd.Series(var_ratio).plot()
```

Out[97]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x23bf7361a58>
```



In [98]:
```
df_scaled.shape
```

Out[98]:
```
(8950, 17)
```

In [99]:
```
pc_final=PCA(n_components=5).fit(df_scaled)
reduced_df=pc_final.fit_transform(df_scaled)
```

In [100]:
```
df1=pd.DataFrame(reduced_df)
df1.head()
```

Out[100]:

	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

In [101]:
```
df1.shape
```

Out[101]:
```
(8950, 5)
```

In [102]:
```
col_list=df_dummy.columns
col_list
```

Out[102]:
```
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',
       'installment', 'none', 'one_off'],
      dtype='object')
```

In [103]:
```
pd.DataFrame(pc_final.components_.T, columns=['PC_' +str(i) for i in range(5
)],index=col_list)
```

Out[103]:

	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	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

In [104]:
```
# Factor Analysis : variance explained by each component-
pd.Series(pc_final.explained_variance_ratio_,index=['PC_'+ str(i) for i in ran
ge(5)])
```

Out[104]:
```
PC_0    0.402058
PC_1    0.180586
PC_2    0.147294
PC_3    0.081606
PC_4    0.065511
dtype: float64
```

Model Selection

In [105]:
```
from sklearn.cluster import KMeans
km_4=KMeans(n_clusters=4,random_state=42)
km_4.fit(reduced_df)
km_4.labels_
```

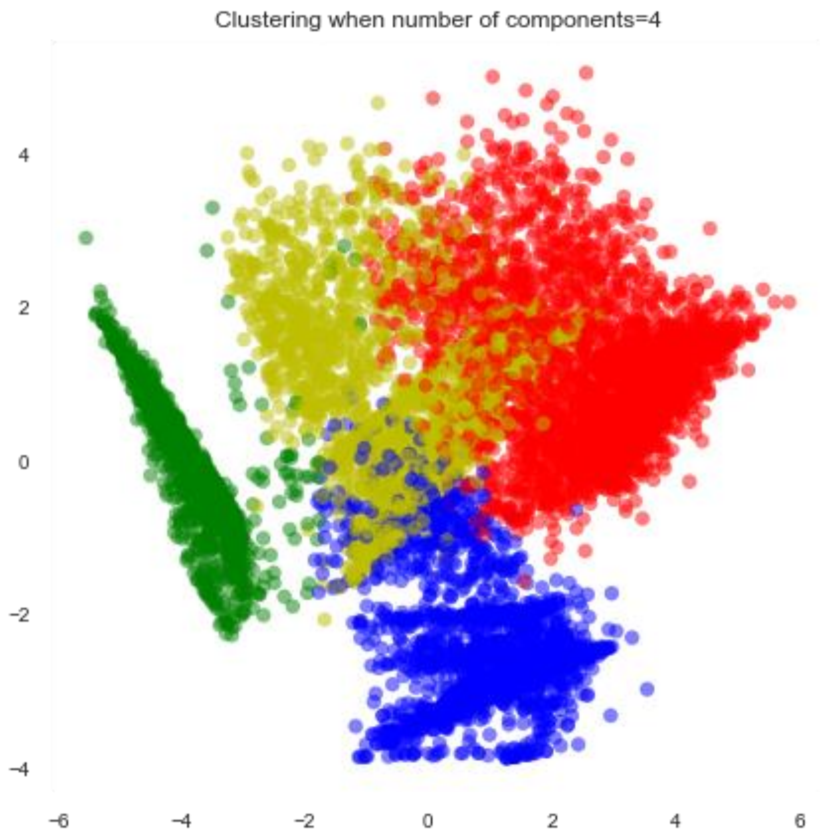
Out[105]:
```
array([1, 2, 3, ..., 1, 2, 3])
```

In [106]:
```
pd.Series(km_4.labels_).value_counts()
```

Out[106]:
```
0    2758
1    2228
2    2090
3    1874
dtype: int64
```

In [107]:
```
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',alph
a=0.5)
plt.title('Clustering when number of components=4')
```

Out[107]:
```
Text(0.5, 1.0, 'Clustering when number of components=4')
```

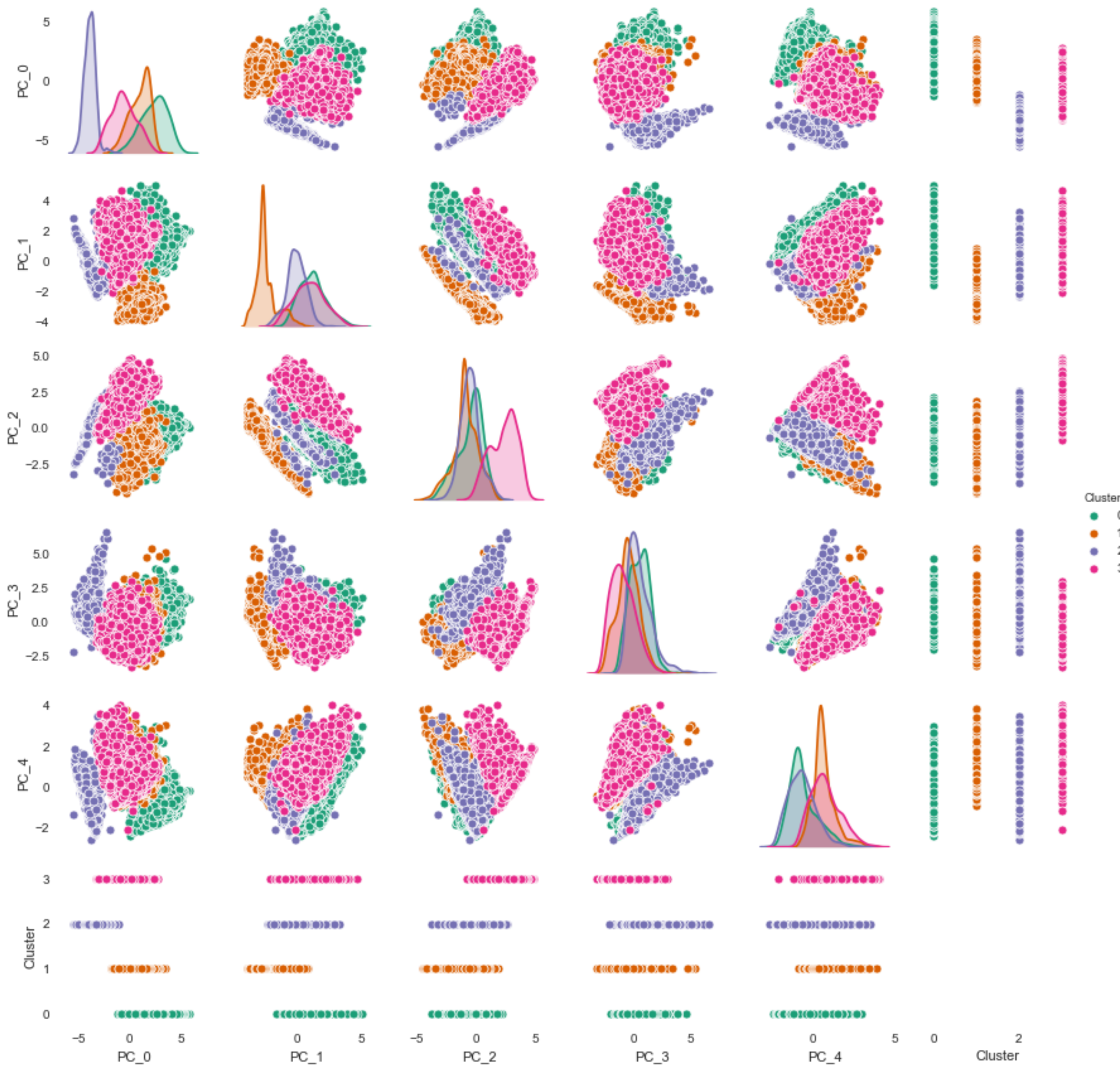


In [108]:

```
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',hei
ght=2)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:4 87:
RuntimeWarning: invalid value encountered in true_divide
binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetool s.py:34:
RuntimeWarning: invalid value encountered in double_scalars
FAC1 = 2*(np.pi*bw/RANGE)**2
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:4 87:
RuntimeWarning: invalid value encountered in true_divide
binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetool s.py:34:
RuntimeWarning: invalid value encountered in double_scalars
FAC1 = 2*(np.pi*bw/RANGE)**2
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:4 87:
RuntimeWarning: invalid value encountered in true_divide
binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetool s.py:34:
RuntimeWarning: invalid value encountered in double_scalars
FAC1 = 2*(np.pi*bw/RANGE)**2
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kde.py:4 87:
RuntimeWarning: invalid value encountered in true_divide
binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\nonparametric\kdetool s.py:34:
RuntimeWarning: invalid value encountered in double_scalars
FAC1 = 2*(np.pi*bw/RANGE)**2
```

```
Out[108]: <seaborn.axisgrid.PairGrid at 0x23bf74edd68>
```



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 increasding the number of clusters value to identify more number of insights about the customers present in the dataset.

In [109]:
```
# Key performace variable selection . here i am dropping varibales which are u
sed in derving new KPI
col_kpi=['purchases_trx','monthly_avg_purchase','monthly_cash_advance','limit_
usage','cash_advance_trx',
         'payment_minpay','both','installment','one_off','none','credit_limit'
]
```

In [110]:
```
transform_pre.describe().T
```
Out[110]:

	count	mean	std	min	25%	50%	75%
balance_frequency	8950.0	0.619940	0.148590	0.0	0.635989	0.693147	0.693147
oneoff_purchases	8950.0	3.204274	3.246365	0.0	0.000000	3.663562	6.360274
installments_purchases	8950.0	3.352403	3.082973	0.0	0.000000	4.499810	6.151961
purchases_frequency	8950.0	0.361268	0.277317	0.0	0.080042	0.405465	0.650588
oneoff_purchases_frequency	8950.0	0.158699	0.216672	0.0	0.000000	0.080042	0.262364
purchases_installments_frequency	8950.0	0.270072	0.281852	0.0	0.000000	0.154151	0.559616
cash_advance_frequency	8950.0	0.113512	0.156716	0.0	0.000000	0.000000	0.200671
cash_advance_trx	8950.0	0.817570	1.009316	0.0	0.000000	0.000000	1.609438
purchases_trx	8950.0	1.894731	1.373856	0.0	0.693147	2.079442	2.890372
monthly_avg_purchase	8950.0	3.050877	2.002823	0.0	1.481458	3.494587	4.587295
monthly_cash_advance	8950.0	2.163970	2.429741	0.0	0.000000	0.000000	4.606022
limit_usage	8950.0	0.296081	0.250303	0.0	0.040656	0.264455	0.540911
payment_minpay	8950.0	1.357600	0.940149	0.0	0.648817	1.109459	1.953415

In [111]:
```
df_original.columns
```
Out[111]:
```
Index(['cust_id', 'balance', 'balance_frequency', 'purchases',
       'oneoff_purchases', 'installments_purchases', 'cash_advance',
       'purchases_frequency', 'oneoff_purchases_frequency',
       'purchases_installments_frequency', 'cash_advance_frequency',
       'cash_advance_trx', 'purchases_trx', 'credit_limit', 'payments',
       'minimum_payments', 'prc_full_payment', 'tenure',
       'monthly_avg_purchase', 'monthly_cash_advance', 'transaction_type',
       'limit_usage', 'payment_minpay', 'both', 'installment', 'none',
       'one_off'],
      dtype='object')
```

In [112]:
```
# Conactenating labels found through Kmeans with data
cluster_df_4=pd.concat([df_original[col_kpi],pd.Series(km_4.labels_,name='Clus
ter_4')],axis=1)
```

In [113]:
```
# Mean value gives a good indication of the distribution of data. So we are fi
nding 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[113]:

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

In [114]:

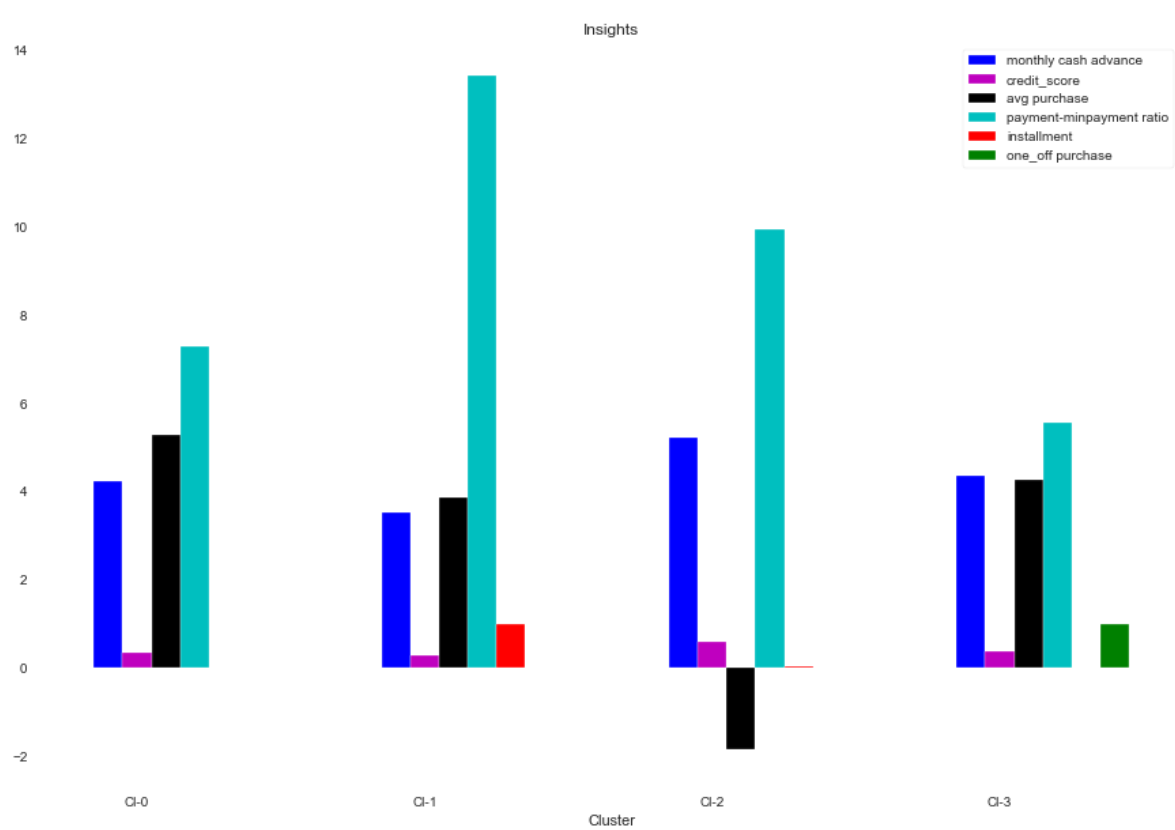
```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(cluster_4.columns))

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=b
ar_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 rati
o',width=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',width=b
ar_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[114]: <matplotlib.legend.Legend at 0x23bf3b0e4e0>



Observations: 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 [115]:
```
# Percentage of each cluster in the total customer base
s=cluster_df_4.groupby('Cluster_4').apply(lambda x: x['Cluster_4'].value_count
s())
print(s)

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

```
Cluster_4
0          0    2758
1          1    2228
2          2    2090
3          3    1874
Name: Cluster_4, dtype: int64
Cluster -4
   Size  Percentage
0  2758   30.815642
1  2228   24.893855
2  2090   23.351955
3  1874   20.938547
```

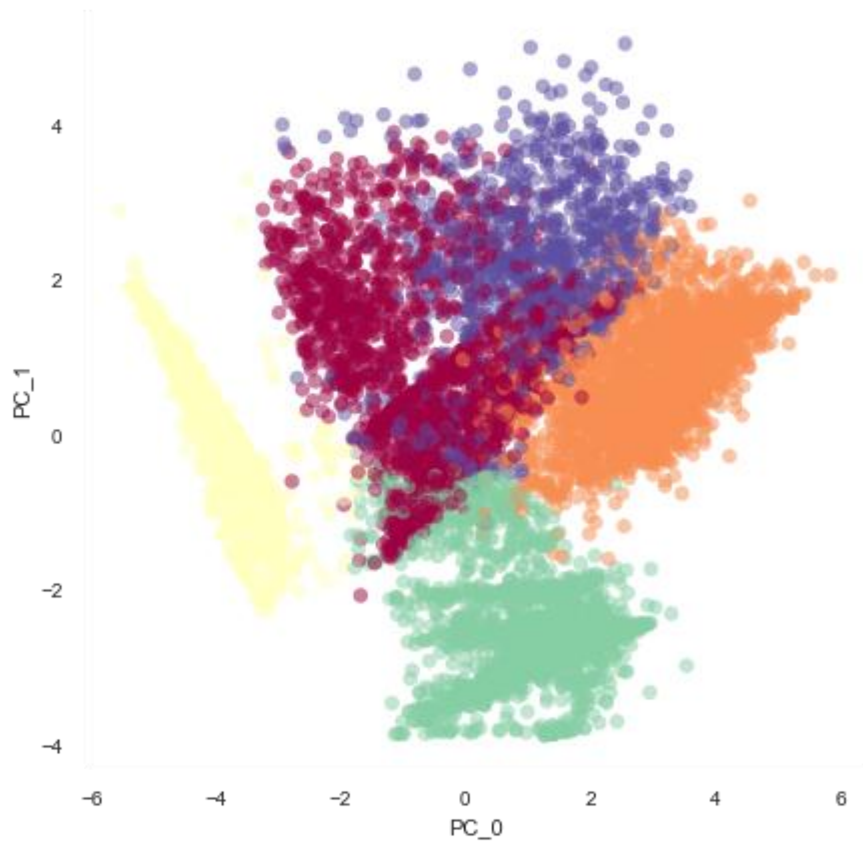
Exploring the insights if the number of cluster=5

In [116]:
```
#kmeans with 5 clusters
km_5=KMeans(n_clusters=5,random_state=42)
km_5=km_5.fit(reduced_df)
km_5.labels_
```

Out[116]:
```
array([3, 2, 0, ..., 3, 2, 0])
```

In [117]:
```
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')
```

Out[117]: Text(0, 0.5, 'PC_1')



In [118]:
```
cluster_df_5=pd.concat([df_original[col_kpi],pd.Series(km_5.labels_,name='Cluster_5')],axis=1)
```

In [119]:
```python
# 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
```

Out[119]:

Cluster_5	0	1	2	3	4
purchases_trx	7.067742	34.535786	0.035509	11.896762	27.566779
monthly_avg_purchase	68.685725	209.776126	0.096572	47.243825	141.906474
monthly_cash_advance	73.635703	3.975040	185.109488	19.155048	252.431524
limit_usage	0.377563	0.262654	0.576260	0.246733	0.595310
cash_advance_trx	2.648387	0.151714	6.454894	0.484280	10.511785
payment_minpay	5.540102	8.573019	9.950170	13.861937	3.917077
both	0.003226	1.000000	0.000000	0.000000	0.879910
installment	0.000000	0.000000	0.016795	1.000000	0.105499
one_off	0.996774	0.000000	0.003359	0.000000	0.014590
none	0.000000	0.000000	0.979846	0.000000	0.000000
credit_limit	4489.884490	5718.529702	4047.344850	3224.454896	5859.820426

In [120]:
```python
s1=cluster_df_5.groupby('Cluster_5').apply(lambda x: x['Cluster_5'].value_counts())
print(s1)
```

```
Cluster_5
0          0    1860
1          1    1984
2          2    2084
3          3    2131
4          4     891
Name: Cluster_5, dtype: int64
```

In [121]:

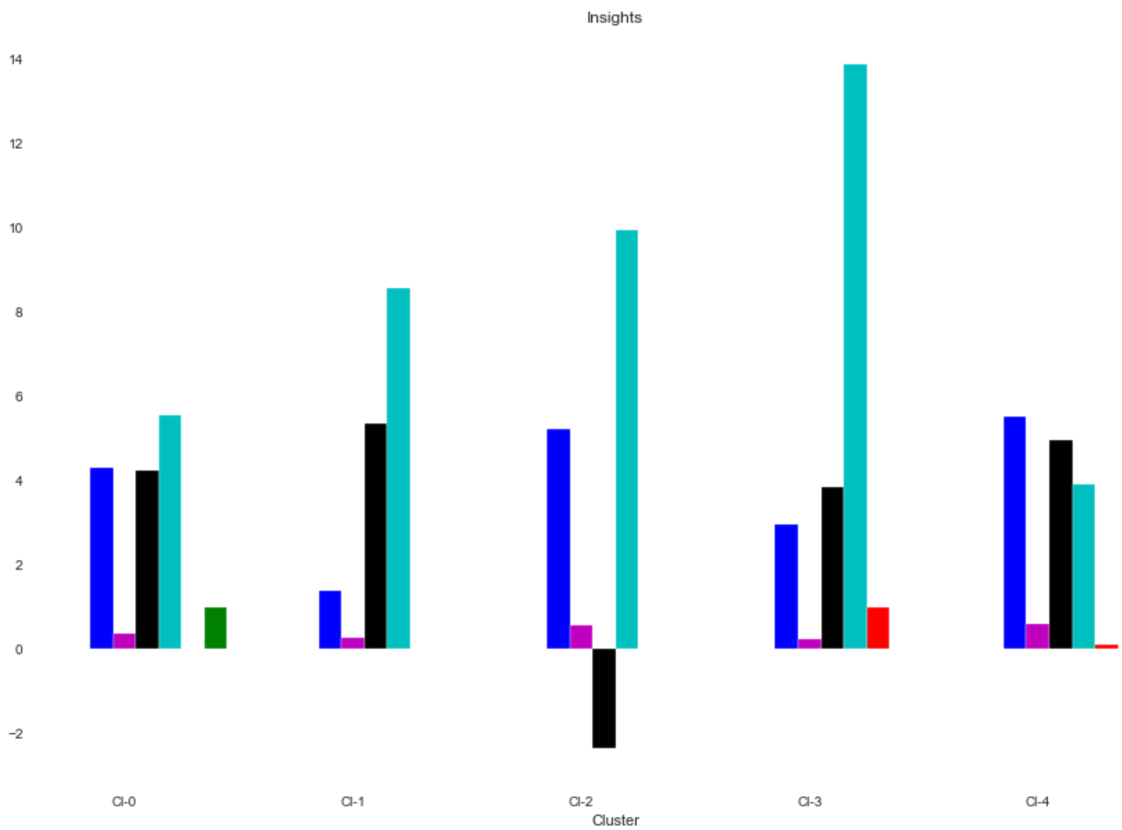
```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(five_cluster.columns))

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=b
ar_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 rati
o',width=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',width=b
ar_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'))
```

Out[121]:
```
([<matplotlib.axis.XTick at 0x23bf552df28>,
  <matplotlib.axis.XTick at 0x23bf552d9b0>,
  <matplotlib.axis.XTick at 0x23bf552d2e8>,
  <matplotlib.axis.XTick at 0x23bf719f668>,
  <matplotlib.axis.XTick at 0x23be02a08d0>],
 [Text(0, 0, 'Cl-0'),
  Text(0, 0, 'Cl-1'),
  Text(0, 0, 'Cl-2'),
  Text(0, 0, 'Cl-3'),
  Text(0, 0, 'Cl-4')])
```



Observations: From the above graph, we can't come to a particular conclusion regarding the behaviour 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.

In [122]:
```
# 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))
```

```
Cluster-5
   Size  Percentage
0  1860   20.782123
1  1984   22.167598
2  2084   23.284916
3  2131   23.810056
4   891    9.955307
```

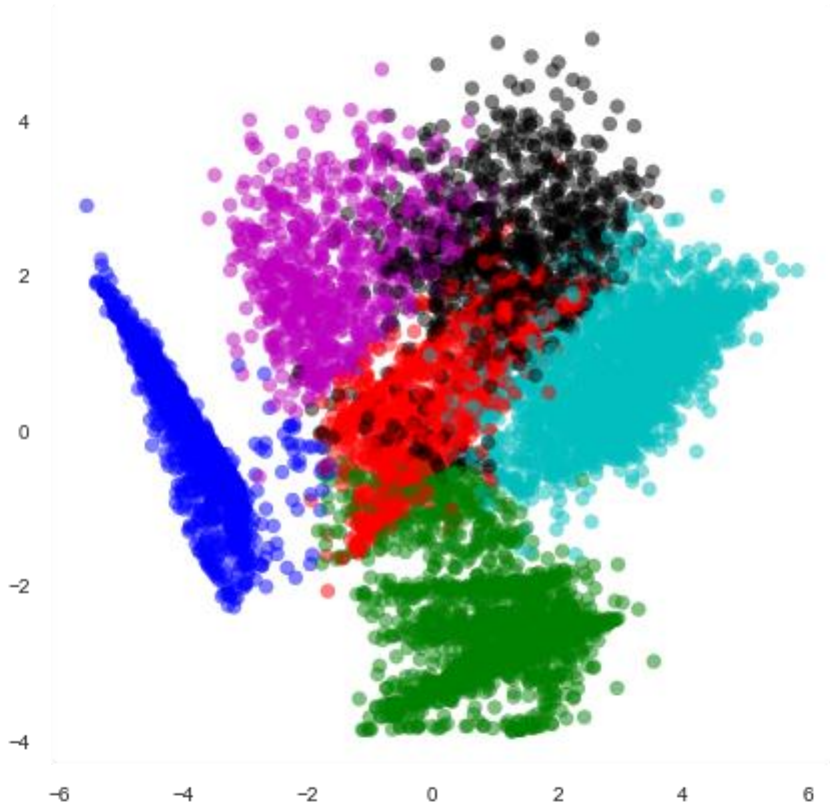
Exploring the insights when number of clusters=6

In [123]:
```
km_6=KMeans(n_clusters=6).fit(reduced_df)
km_6.labels_
```

Out[123]: array([2, 1, 0, ..., 2, 1, 4])

In [124]:
```
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',alph
a=0.5)
```

Out[124]: <matplotlib.collections.PathCollection at 0x23be01860b8>



In [125]:
```
cluster_df_6=pd.concat([df_original[col_kpi],pd.Series(km_6.labels_,name='Clus
ter_6')],axis=1)
```

In [126]:
```
six_cluster=cluster_df_6.groupby('Cluster_6').apply(lambda x: x[col_kpi].mean
()).T
six_cluster
```

Out[126]:

Cluster_6	0	1	2	3	4	
purchases_trx	7.748735	0.033205	11.896762	34.653320	5.971388	27.7
monthly_avg_purchase	78.450604	0.098395	47.243825	210.512330	54.099042	140.6
monthly_cash_advance	3.669832	184.912834	19.155048	3.942946	205.662979	243.9
limit_usage	0.245107	0.575884	0.246733	0.262170	0.606579	0.5
cash_advance_trx	0.131535	6.435034	0.484280	0.149012	7.642346	10.0
payment_minpay	6.893503	9.967837	13.861937	8.610468	3.255430	3.8
both	0.009275	0.000000	0.000000	1.000000	0.000000	0.8
installment	0.000000	0.017324	1.000000	0.000000	0.000000	0.1
one_off	0.990725	0.000000	0.000000	0.000000	1.000000	0.0
none	0.000000	0.982676	0.000000	0.000000	0.000000	0.0
credit_limit	4471.374879	4048.925249	3224.454896	5722.145428	4591.494343	5817.1

In [127]:

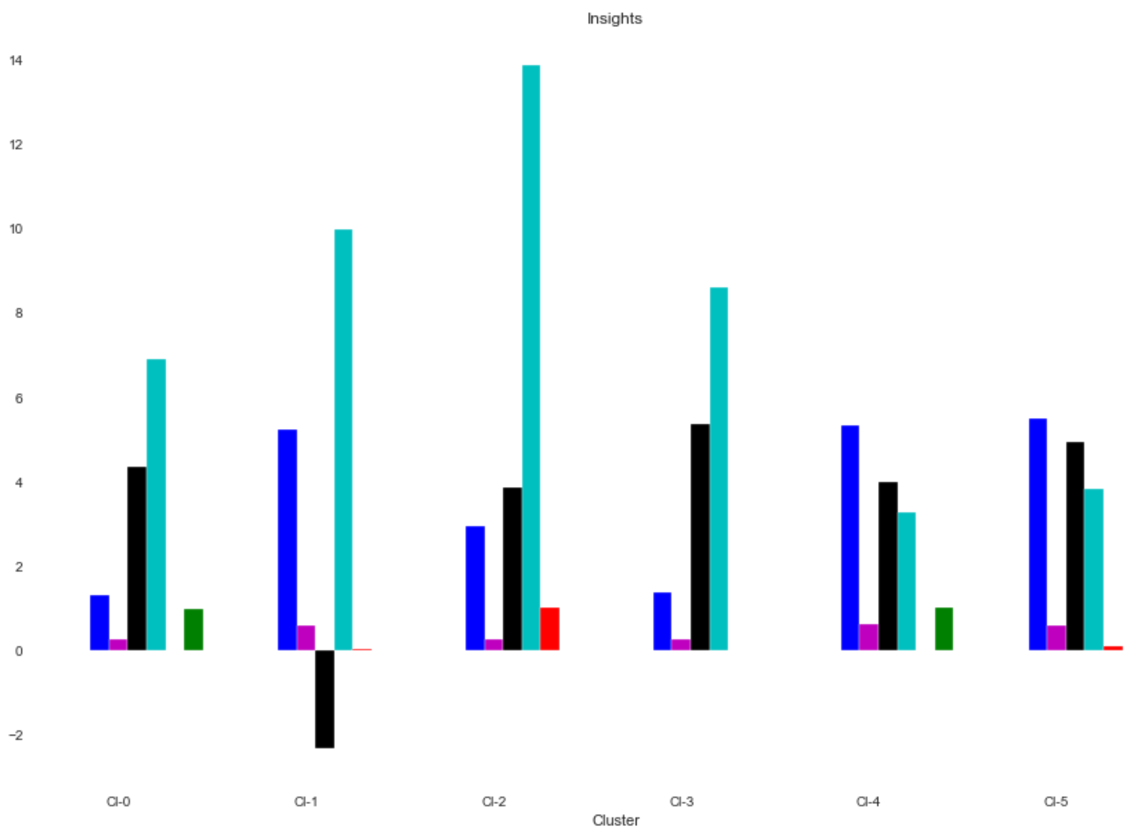
```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(six_cluster.columns))

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=b
ar_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 rati
o',width=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',width=b
ar_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'))
```

Out[127]:
```
([<matplotlib.axis.XTick at 0x23be010fbe0>,
  <matplotlib.axis.XTick at 0x23be010fba8>,
  <matplotlib.axis.XTick at 0x23be010f7f0>,
  <matplotlib.axis.XTick at 0x23be027ada0>,
  <matplotlib.axis.XTick at 0x23be02782b0>,
  <matplotlib.axis.XTick at 0x23be0278710>],
 [Text(0, 0, 'Cl-0'),
  Text(0, 0, 'Cl-1'),
  Text(0, 0, 'Cl-2'),
  Text(0, 0, 'Cl-3'),
  Text(0, 0, 'Cl-4'),
  Text(0, 0, 'Cl-5')])
```



Observations: 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

In [128]:
```
cash_advance=np.log(six_cluster.iloc[2,:].values)
credit_score=list(six_cluster.iloc[3,:].values)
print(cash_advance)
print(credit_score)
```

```
[1.30014588 5.21988454 2.95256629 1.37192804 5.32623881 5.49690086]
[0.24510688195597208, 0.5758841059122126, 0.24673287577047076, 0.262169628616
57617, 0.6065787726196032, 0.5957844119450174]
```

Metrics for the KMeans Model

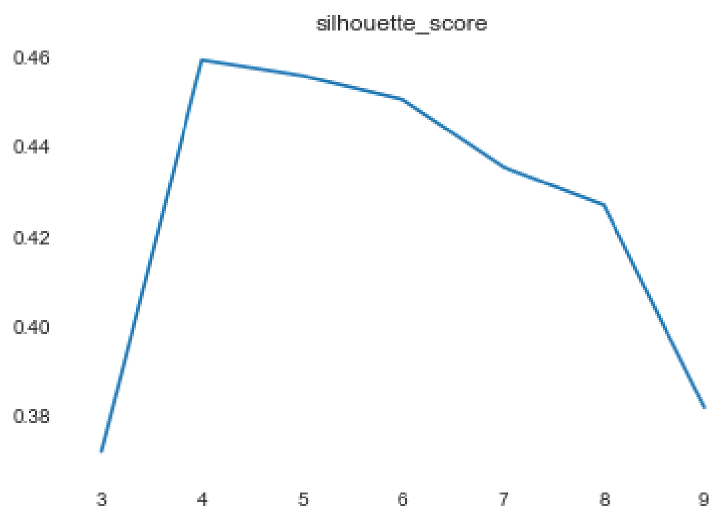
In [129]:
```
from sklearn.metrics import calinski_harabasz_score,silhouette_score
score={}
score_c={}
for n in range(3,10):
    km_score=KMeans(n_clusters=n)
    km_score.fit(reduced_df)
    score_c[n]=calinski_harabasz_score(reduced_df,km_score.labels_)
    score[n]=silhouette_score(reduced_df,km_score.labels_)
```

In [130]:
```
print(score)
```

```
{3: 0.37210314965312236, 4: 0.45925855175969704, 5: 0.4557517692878904, 6: 0.
450405001213826, 7: 0.4354151287227094, 8: 0.4269729550650407, 9: 0.381915485
49313575}
```

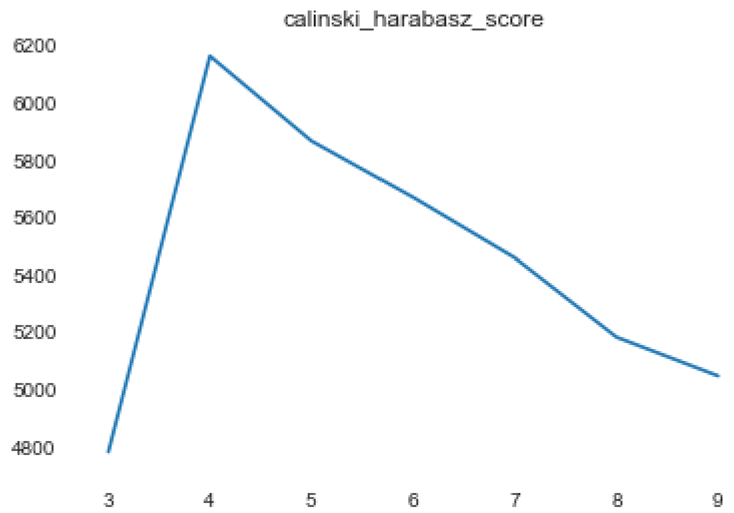
In [131]:
```
pd.Series(score).plot()
plt.title('silhouette_score')
```

Out[131]:
```
Text(0.5, 1.0, 'silhouette_score')
```

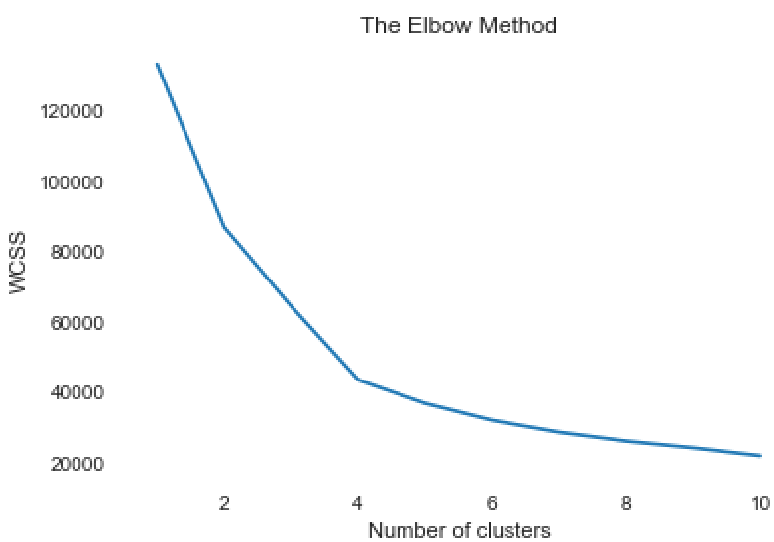


In [132]:
```
pd.Series(score_c).plot()
plt.title('calinski_harabasz_score')
```

Out[132]: Text(0.5, 1.0, 'calinski_harabasz_score')



In [133]:
```
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()
```



Observations:

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 cluster as four.

Final K-Means Model

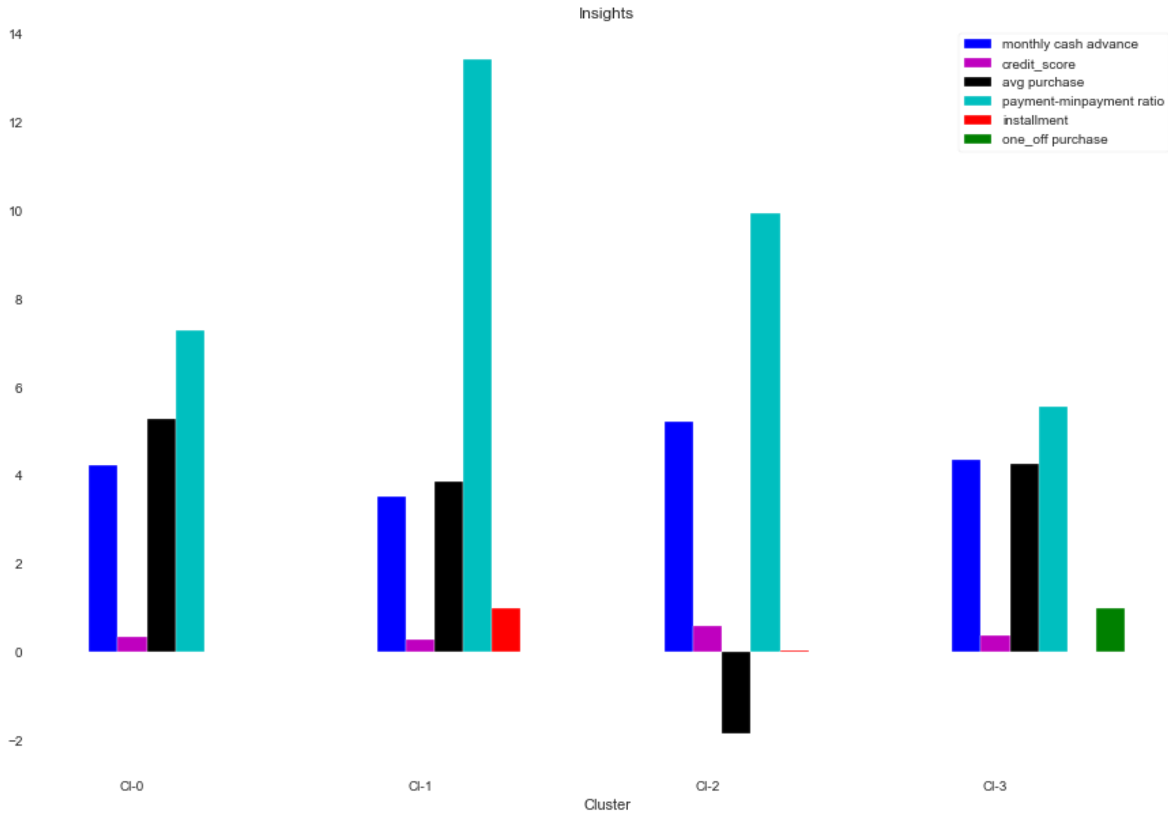
In [134]:
```
fig,ax=plt.subplots(figsize=(15,10))
index=np.arange(len(cluster_4.columns))

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=b
ar_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 rati
o',width=bar_width)
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',width=b
ar_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[134]: <matplotlib.legend.Legend at 0x23be0b39b00>


Insights
monthly cash advance
credit_score
avg purchase
payment-minpayment ratio
installment
one_off purchase
14
12
10
8
6
4
2
0
−2
Cl-0
Cl-1
Cl-2
Cl-3
Cluster

Marketing Strategies cluster:

CLUSTER 0:

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.

CLUSTER 1:

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 rewarded with reward points and thus make them do more transactions in future.

CLUSTER 2:

Customers belong to this category of cluster having the highest cash advance and poor avg purchase 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

CLUSTER 3:

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 cutomers. 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.

THE SAME THINGS WE DO IN R

```
seg <-read.csv(file.choose())
View(seg)
sum(is.na(seg$CUST_ID))##0
sum(is.na(seg$BALANCE))##0
sum(is.na(seg$BALANCE_FREQUENCY))##0
sum(is.na(seg$PURCHASES))##0
sum(is.na(seg$ONEOFF_PURCHASES))##0
sum(is.na(seg$INSTALLMENTS_PURCHASES))##0
sum(is.na(seg$CASH_ADVANCE))##0
sum(is.na(seg$PURCHASES_FREQUENCY))##0
sum(is.na(seg$ONEOFF_PURCHASES_FREQUENCY))##0
sum(is.na(seg$PURCHASES_INSTALLMENTS_FREQUENCY))##0
sum(is.na(seg$CASH_ADVANCE_FREQUENCY))##0
sum(is.na(seg$CASH_ADVANCE_TRX))##0
sum(is.na(seg$PURCHASES_TRX))##0
sum(is.na(seg$CREDIT_LIMIT))##1
sum(is.na(seg$PAYMENTS))##0
sum(is.na(seg$MINIMUM_PAYMENTS))##313
sum(is.na(seg$PRC_FULL_PAYMENT))##0
sum(is.na(seg$TENURE))##0

# 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
```

```
write.csv(seg,"New_variables_creation.csv")


#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

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


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")


eigen(corrm)$values
```

```
Output: eigen(corrm)$values
 [1] 5.43628692 4.34186609 2.10299897 1.65342188 1.24504759 1.05006876 0.98231813 0.738
60862 0.72744967 0.63620389 0.57783813 0.35278271
[13] 0.28990257 0.26560402 0.16088455 0.11783311 0.11503442 0.09749872 0.05163199 0.041
26943 0.01544985
```

```
install.packages(c('dplyr','psych','tables'))

library(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")



# 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_c
lust_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"
)



cluster_profiling2 = t(tt2)

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