edWisor Project

Credit Card Segmentation

Submitted By

Naveen Thomas

INDEX

1.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.

Dataset:-

1) credit-card-data.csv (8950 observations, 18 variables)

Number of attributes:

1. **CUST_ID** Credit card holder ID.
2. **BALANCE** Monthly average balance (based on daily balance averages).
3. **BALANCE_FREQUENCY** Ratio of last 12 months with balance.
4. **PURCHASES** Total purchase amount spent during last 12 months.
5. **ONEOFF_PURCHASES** Total amount of one-off purchases.
6. **INSTALLMENTS_PURCHASES** Total amount of instalment purchases.
7. **CASH_ADVANCE** Total cash-advance amount.
8. **PURCHASES_ FREQUENCY** Frequency of purchases (percentage of months with at least on purchase).
9. **ONEOFF_PURCHASES_FREQUENCY** Frequency of one-off-purchases.
10. **PURCHASES_INSTALLMENTS_FREQUENCY** Frequency of instalment purchases.
11. **CASH_ADVANCE_ FREQUENCY** Cash-Advance frequency.
12. **CASH_ADVANCE_TRX** Average amount per cash-advance transaction.
13. **PURCHASES_TRX** Average amount per purchase transaction.
14. **CREDIT_LIMIT** Credit limit.
15. **PAYMENTS** Total payments (due amount paid by the customer to decrease their statement balance) in the period.
16. **MINIMUM_PAYMENTS** Total minimum payments due in the period.
17. **PRC_FULL_PAYMENT** Percentage of months with full payment of the due statement balance.
18. **TENURE** Number of months as a customer.

2.Data Cleaning

Contents of credit-card-data.csv is stored in 'credit' dataframe.

Data type of credit-card-data.csv:

```
<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
```

Here, we will check for any nonsensible values in all columns. If found any, we will make that observation value as NaN and finally impute them along with other missing values.

2.1 CASH_ADVANCE_FREQUENCY

```
count    8950.000000
mean        0.135144
std         0.200121
min         0.000000
25%         0.000000
50%         0.000000
75%         0.222222
max         1.500000
Name: CASH_ADVANCE_FREQUENCY, dtype: float64
```

Here maximum value of CASH_ADVANCE_FREQUENCY is 1.5 which is impossible as maximum value should be 1. So, we will be setting those values greater than 1 to NaN.

CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY
7240.433194	0.250000	0.250000	0.0	1.250000
3719.650168	0.333333	0.333333	0.0	1.166667
1651.286918	0.125000	0.125000	0.0	1.125000
4109.465221	0.100000	0.100000	0.0	1.100000
1932.460679	0.000000	0.000000	0.0	1.500000
2794.326341	0.000000	0.000000	0.0	1.166667
6084.858872	0.363636	0.363636	0.0	1.090909
2127.213754	0.000000	0.000000	0.0	1.142857

No. of observations put to NaN= 8

3.Missing Value Analysis

	Count	Percentage
MINIMUM_PAYMENTS	313	3.497207
CASH_ADVANCE_FREQUENCY	8	0.089385
CREDIT_LIMIT	1	0.011173
CUST_ID	0	0.000000
BALANCE	0	0.000000
PRC_FULL_PAYMENT	0	0.000000
PAYMENTS	0	0.000000
PURCHASES_TRX	0	0.000000
CASH_ADVANCE_TRX	0	0.000000
PURCHASES_INSTALLMENTS_FREQUENCY	0	0.000000
ONEOFF_PURCHASES_FREQUENCY	0	0.000000
PURCHASES_FREQUENCY	0	0.000000
CASH_ADVANCE	0	0.000000
INSTALLMENTS_PURCHASES	0	0.000000
ONEOFF_PURCHASES	0	0.000000
PURCHASES	0	0.000000
BALANCE_FREQUENCY	0	0.000000
TENURE	0	0.000000

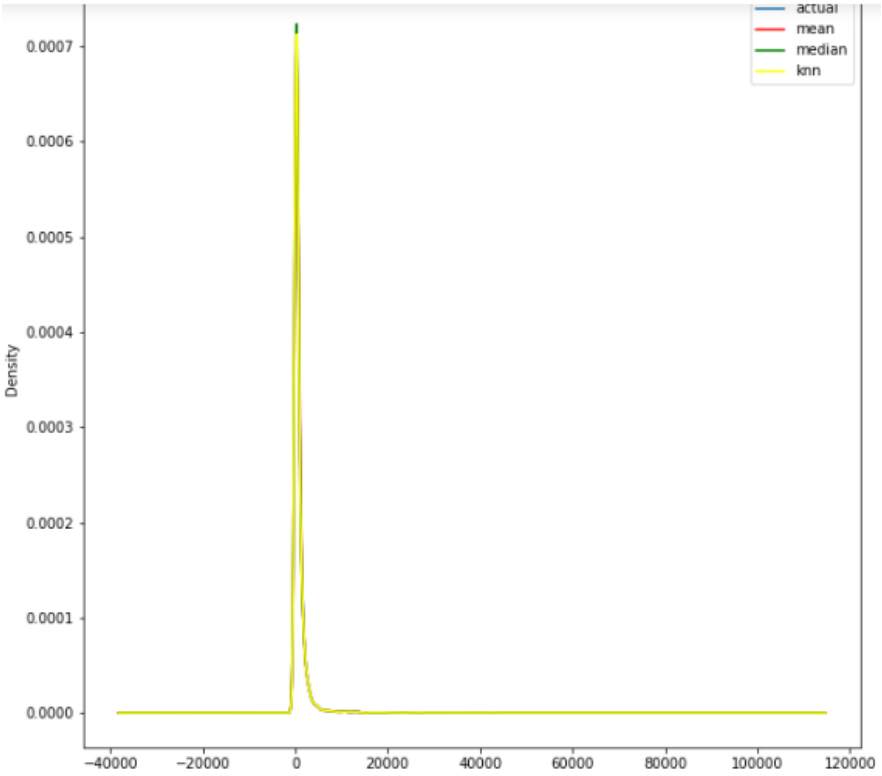
We will be setting a known value of a missing value variable to NaN and perform mean, median and knn imputation and check which imputation is better by comparing it with the actual value.

3.1 MINIMUM_PAYMENTS

We set the 100th observation of MINIMUM_PAYMENTS (which is known to us) to NaN. Then performed mean, median and knn imputation.

```
Actual value:  60.913577000000004
Mean Imputation:  864.2995590912445
Median Imputation:  312.4522915
KNN Imputation at k=7:  65.45922057142857
```

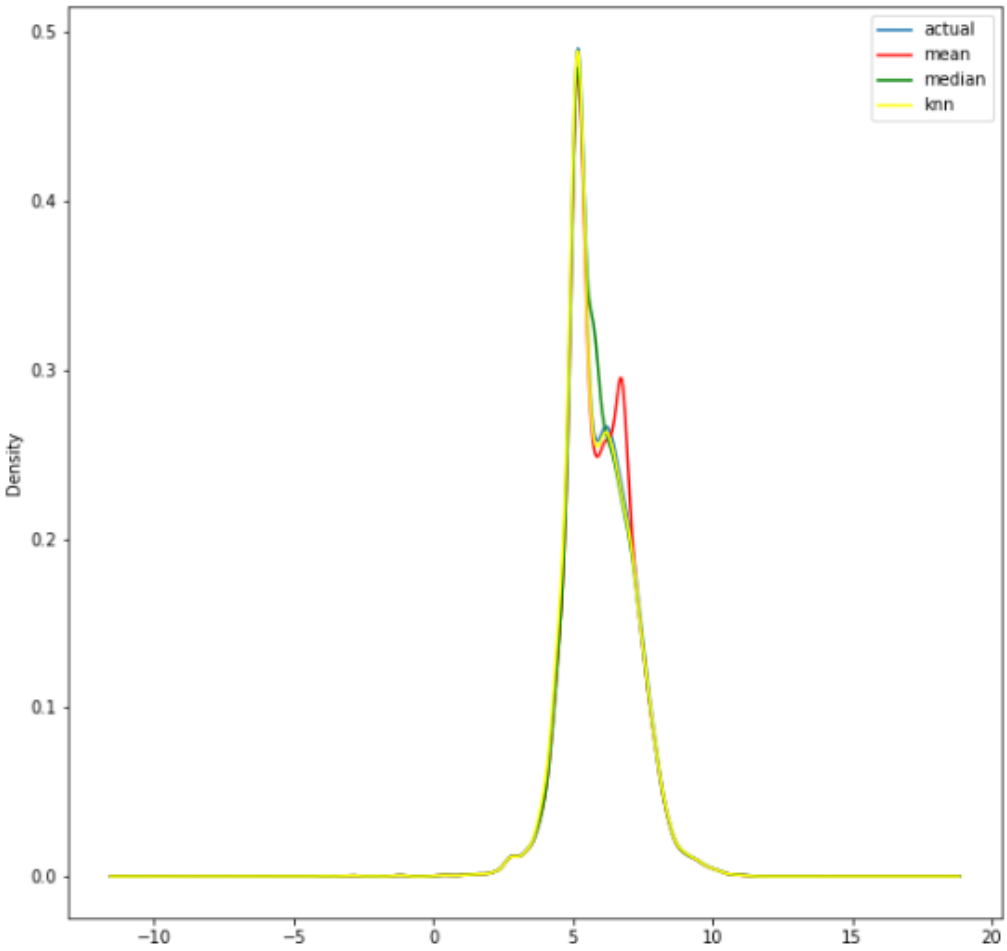
Based on these values KNN performs the best.



Probability density function of actual value, mean, median and knn imputation of MINIMUM_PAYMENTS.

Since MINIMUM_PAYMENTS has very large outliers, we are not able to distinguish PDF of different imputations.

So we are inputting the imputed MINIMUM_PAYMENTS to log function to reduce the effect of outliers so that we will be able to distinguish PDF of different imputations.



Probability density function of actual value, mean, median and knn imputation of log transformation of MINIMUM_PAYMENTS.

From this given PDF, we can see knn Imputation has very little variation compared to actual value. So we are using knn imputation to impute the missing values of MINIMUM_PAYMENTS.

3.2 CASH_ADVANCE_FREQUENCY

We set the 453th observation of CASH_ADVANCE_FREQUENCY (which is known to us) to NaN. Then performed mean, median and knn imputation.

```
Actual value:  1.0
Mean Imputation:  0.13410116239794367
Median Imputation:  0.0
KNN Imputation at k=4:  0.47916675
```

Based on these values KNN performs the best. So we are using knn imputation to impute the missing values of CASH_ADVANCE_FREQUENCY.

3.3 CREDIT_LIMIT

We set the 100th observation of CREDIT_LIMIT (which is known to us) to NaN. Then performed mean, median and knn imputation.

```
Actual value:  1500.0
Mean Imputation:  4494.784100504358
Median Imputation:  3000.0
KNN Imputation at k=2:  2400.0
```

Based on these values KNN performs the best. So we are using knn imputation to impute the missing values of CREDIT_LIMIT.

3.4 Imputation of missing values

All the missing values are imputed by KNN as it performed best.

4.Feature Extraction (Deriving New KPIs)

4.1 Monthly Average Purchases

From PURCHASES and TENURE variables, we can derive a new variable called MONTHLY_AVG_PURCHASES.

MONTHLY_AVG_PURCHASES = PURCHASES / TENURE

```
count    8950.000000
mean       86.175173
std       180.508787
min         0.000000
25%         3.399375
50%        31.936667
75%        97.228333
max      4086.630833
Name: MONTHLY_AVG_PURCHASES, dtype: float64
```

4.2 Monthly Average Cash Advance Amount

From CASH_ADVANCE and TENURE variables, we can derive a new variable called MONTHLY_AVG_CASH_ADVANCE.

MONTHLY_AVG_CASH_ADVANCE = CASH_ADVANCE / TENURE

```
count    8950.000000
mean       88.977984
std       193.136115
min         0.000000
25%         0.000000
50%         0.000000
75%        99.085196
max      3928.100980
Name: MONTHLY_AVG_CASH_ADVANCE, dtype: float64
```

4.3 Purchases by Type

From ONEOFF_PURCHASES and INSTALLMENTS_PURCHASES variables, we can derive a new variable called PURCHASE_TYPE.

- (ONEOFF_PURCHASES == 0) & (INSTALLMENTS_PURCHASES == 0) --> PURCHASE_TYPE= “NONE”
- (ONEOFF_PURCHASES > 0) & (INSTALLMENTS_PURCHASES > 0) --> PURCHASE_TYPE= “BOTH ONEOFF & INSTALMENT”
- (ONEOFF_PURCHASES > 0) & (INSTALLMENTS_PURCHASES == 0) --> PURCHASE_TYPE= “ONEOFF”
- (ONEOFF_PURCHASES == 0) & (INSTALLMENTS_PURCHASES > 0) --> PURCHASE_TYPE= “INSTALMENT”

```
BOTH ONEOFF & INSTALMENT    2774
INSTALMENT                  2260
NONE                        2042
ONEOFF                      1874
Name: PURCHASE_TYPE, dtype: int64
```

4.4 Limit Usage (Balance to credit limit ratio)

From BALANCE and CREDIT_LIMIT variables, we can derive a new variable called LIMIT_USAGE.

LIMIT_USAGE = BALANCE / CREDIT_LIMIT

```
count    8950.000000
mean        0.388884
std         0.389722
min         0.000000
25%         0.041494
50%         0.302720
75%         0.717571
max        15.909951
Name: LIMIT_USAGE, dtype: float64
```

4.5 Payments to Minimum Payments Ratio

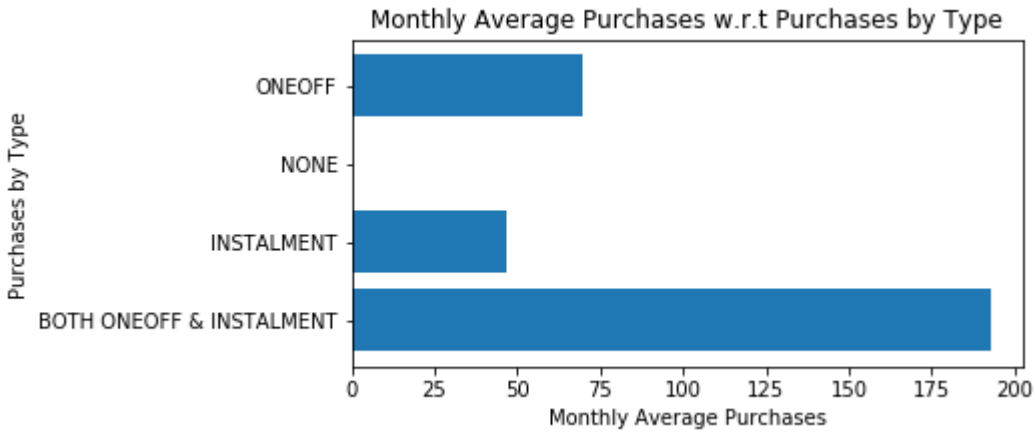
From PAYMENTS and MINIMUM_PAYMENTS variables, we can derive a new variable called PAYMENTS_MIN_PAYMENTS_RATIO.

PAYMENTS_MIN_PAYMENTS_RATIO = PAYMENTS / MINIMUM_PAYMENTS

```
count    8950.000000
mean        9.061145
std       118.176094
min         0.000000
25%         0.918910
50%         2.048261
75%         6.079428
max      6840.528861
Name: PAYMENTS_MIN_PAYMENTS_RATIO, dtype: float64
```

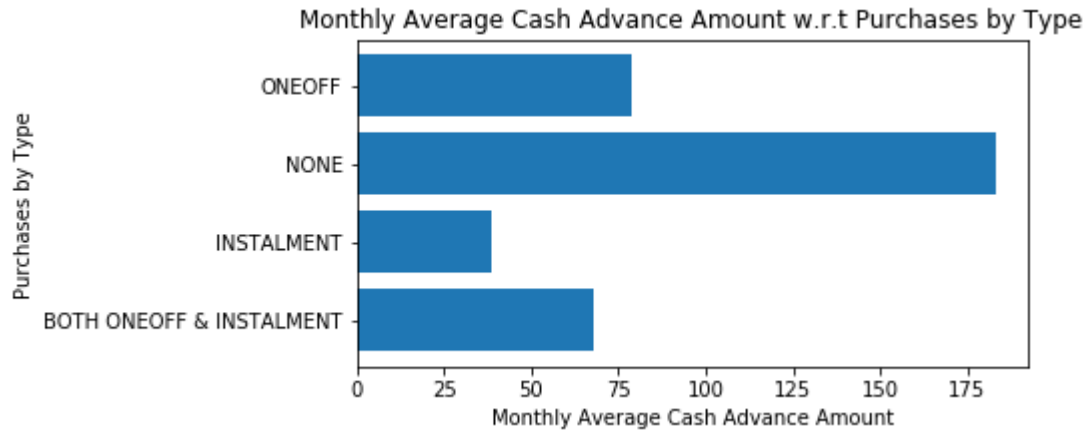
4.6 Insights from KPIs

4.6.1 Monthly Average Purchases w.r.t Purchases by Type



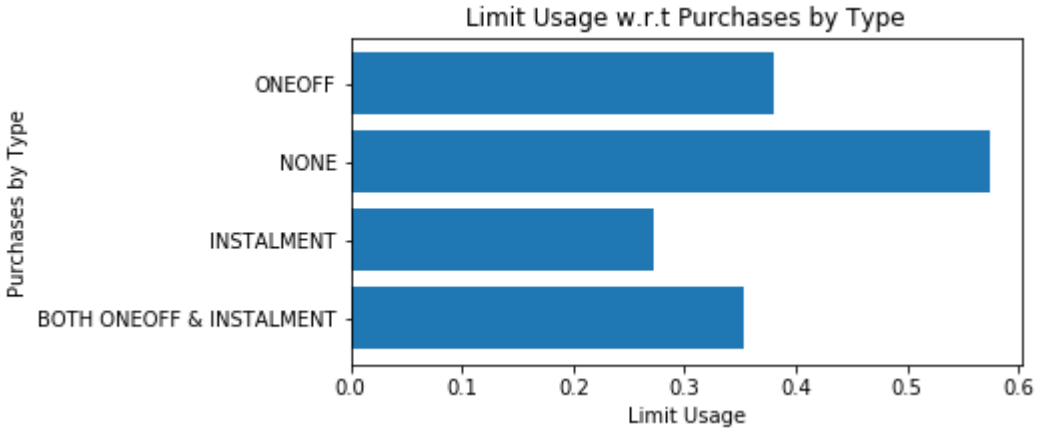
Based on this bar graph we can say that customers who purchase by both one-off and instalment spend money more for purchase monthly.

4.6.2 Monthly Average Cash Advance Amount w.r.t Purchases by Type



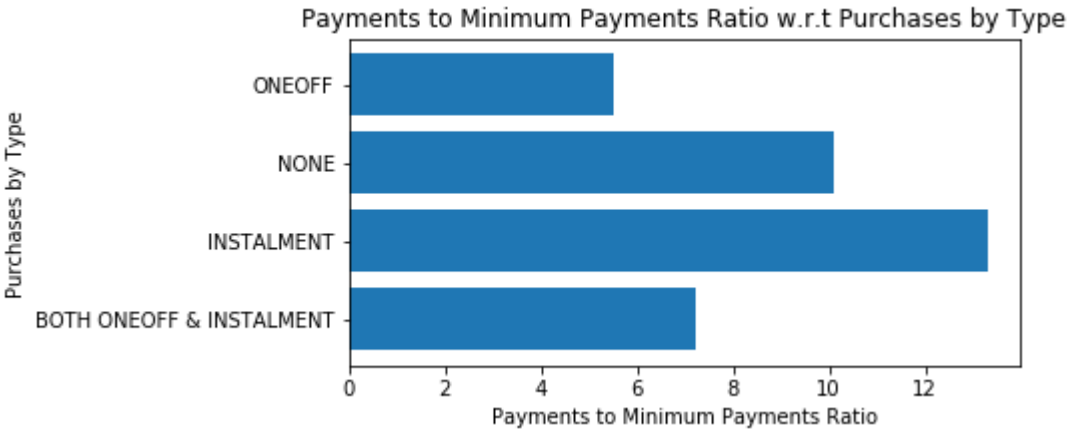
Based on this bar graph we can say that customers who neither purchase by both one-off nor by instalment withdraw money as cash more in a month whereas customers who purchase by only instalment withdraw the least.

4.6.3 Limit Usage w.r.t Purchases by Type



Based on this bar graph we can say that customers who neither purchase by both one-off nor by instalment utilize the credit card more in a month whereas customers who purchase by only instalment utilize the least.

4.6.4 Payments to Minimum Payments Ratio w.r.t Purchases by Type



Based on this bar graph we can say that customers who purchase by only instalment have the highest Payments to Minimum Payments ratio while customers who purchase by only one-off have the least ratio.

One-hot encoding for nominal category variable

Since PURCHASE_TYPE is a nominal categorical variable, we need to do one-hot encoding to convert to numerical variable.

PURCHASE_TYPE			BOTH ONEOFF & INSTALMENT	INSTALMENT	NONE	ONEOFF
INSTALMENT		**0**	0	1	0	0
NONE		**1**	0	0	1	0
ONEOFF		**2**	0	0	0	1
ONEOFF		**3**	0	0	0	1
ONEOFF	---->	**4**	0	0	0	1

This one-hot encoded variable is stored in a new dataframe ‘cr_encode’.

Storage of original data

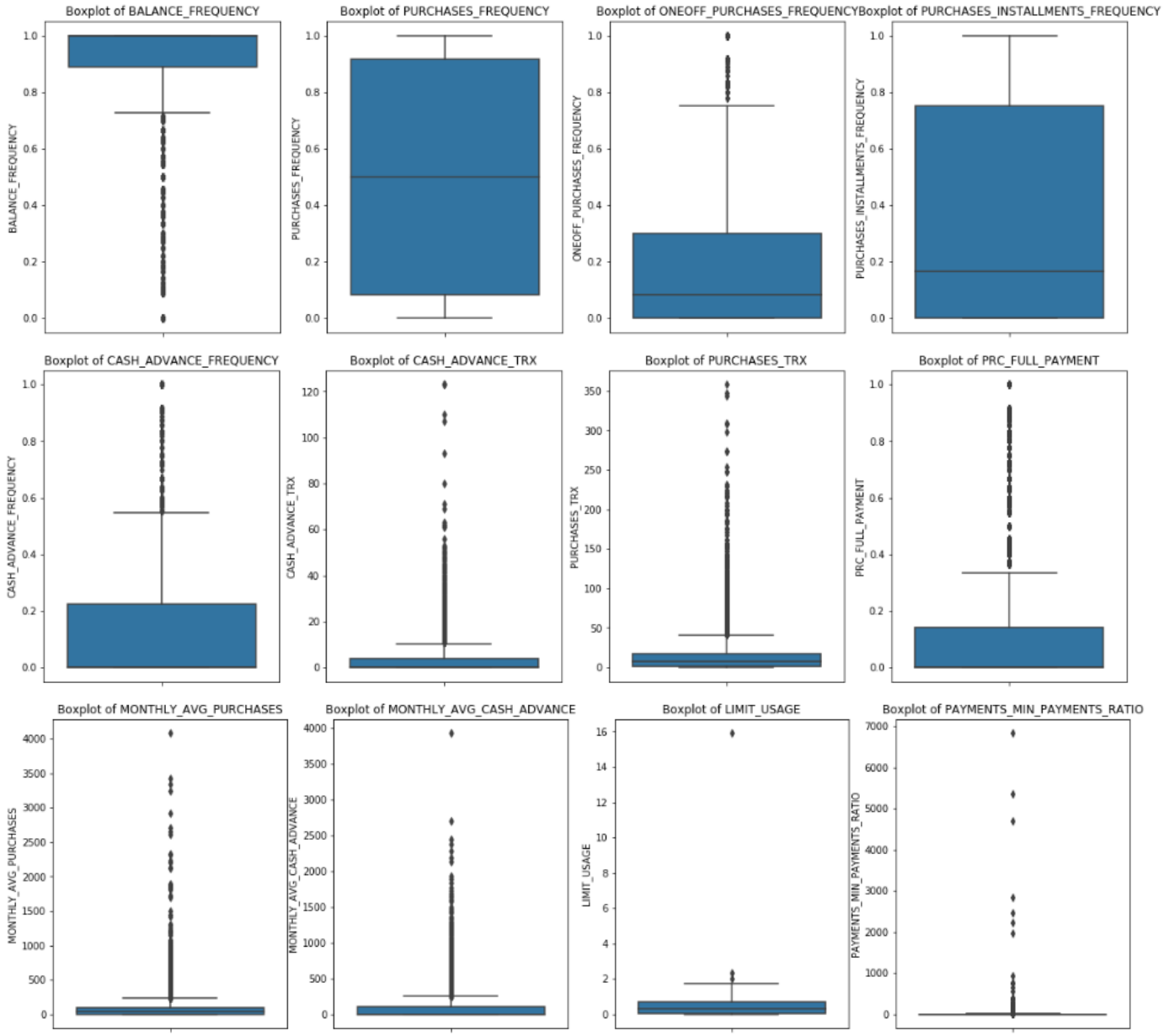
Now before moving to outlier analysis, we need to store these data in a new dataframe ‘credit_original’ which we will need for identifying customer behaviour in each cluster (after model building).

‘credit_original’ consist of credit and cr_encode.

Dropping off some columns from credit dataframe

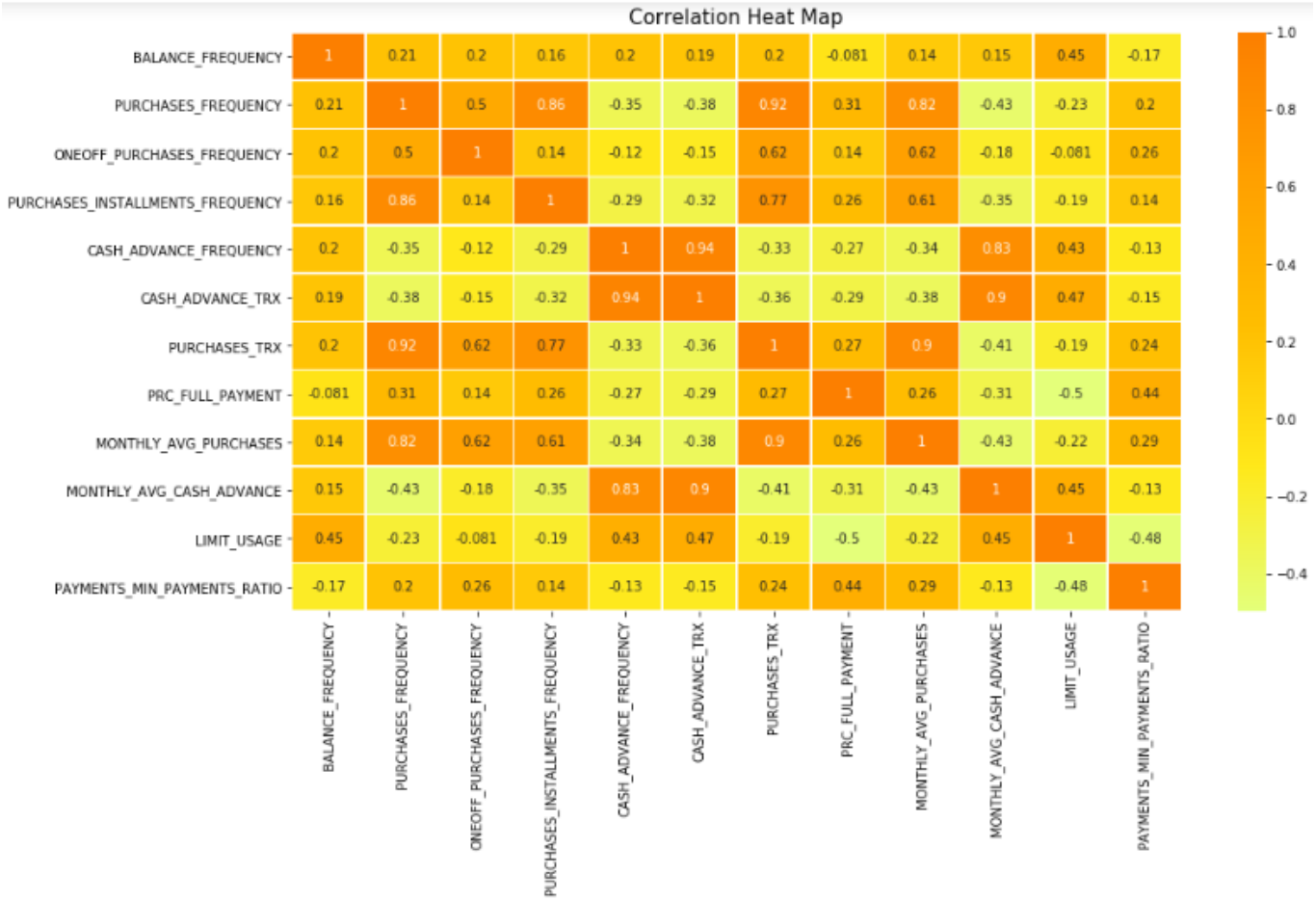
Now we are dropping those variables from which new KPIs are derived. These are PURCHASE_TYPE, BALANCE, PURCHASES, PAYMENTS, ONEOFF_PURCHASES, INSTALLMENTS_PURCHASES, CASH_ADVANCE, MINIMUM_PAYMENTS, CREDIT_LIMIT, TENURE. We are also dropping CUST_ID which doesn't require for model building.

5.Outlier Analysis



Since outliers are present in this dataset and may give valuable information to our model, we are not going to remove or set values to NaN. Instead we will be doing log transformation to reduce outlier effect. Resultant values are stored in 'credit_log'.

6.Feature Selection



Here we can find that there are many features which are having multicollinearity.

We will be using PCA to remove multicollinearity and to reduce the dimensions. Before applying PCA we will standardize data to avoid effect of scale on our result.

Merging of both numerical and categorical variables

Now numerical variables stored in 'credit_log' and categorical variables stored in 'cr_encode' is merged together before feature scaling and the resultant output is stored in 'credit_encode'.

7.Feature Scaling (Standardization)

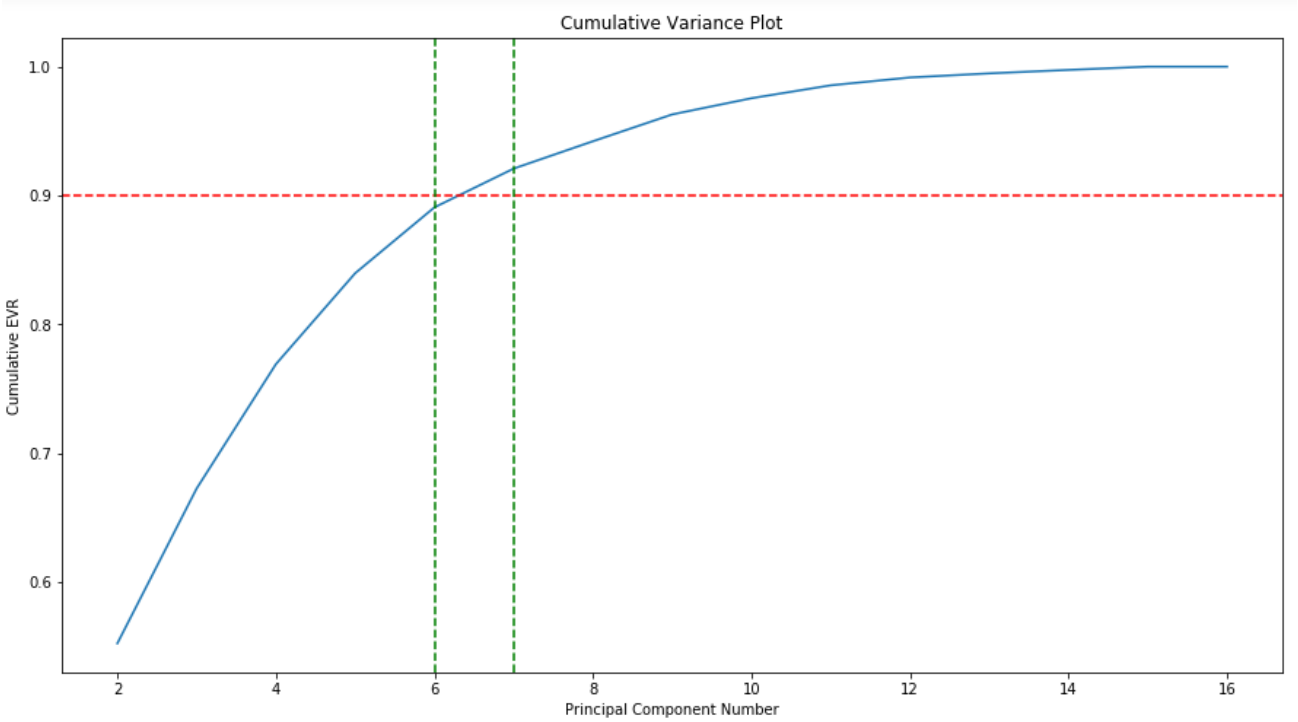
Now standardization is performed on all variables before using PCA.
z = (x-u)/sigma

8.Dimensionality Reduction (PCA)

8.1 Calculation of cumulative Explained Variance Ratio (EVR)

```
{2: 0.5521638293657727,
 3: 0.6724015390801605,
 4: 0.7689682782229399,
 5: 0.8395681031207789,
 6: 0.8907289523294157,
 7: 0.920840587070076,
 8: 0.9420637115918276,
 9: 0.9628895083252493,
 10: 0.9754671820993732,
 11: 0.9854992410254795,
 12: 0.9915956162010514,
 13: 0.9947662931023447,
 14: 0.9974558314996169,
 15: 1.0,
 16: 1.0}
```

8.2 Plotting of cumulative EVR (Cumulative Variance Plot)



Since 7 components are explaining more than 90% variance so we will select 7 components.

	PC_1	PC_2	PC_3	PC_4	PC_5	PC_6	PC_7
0	0.065509	-2.351500	-0.808282	-0.852112	-0.024910	0.220955	0.235479
1	3.580346	-0.125000	-0.183906	1.429836	-0.430823	-0.743903	-0.081130
2	-1.401315	0.397136	2.493168	-2.132849	0.301884	-0.633239	-0.234572
3	1.078138	-0.858779	1.743438	-1.650217	0.900553	0.865316	-0.821598
4	1.147298	-1.170643	1.655936	-1.946654	-0.242003	-0.845406	0.665289

This will be our final data for model building, stored in 'credit_model' dataframe.

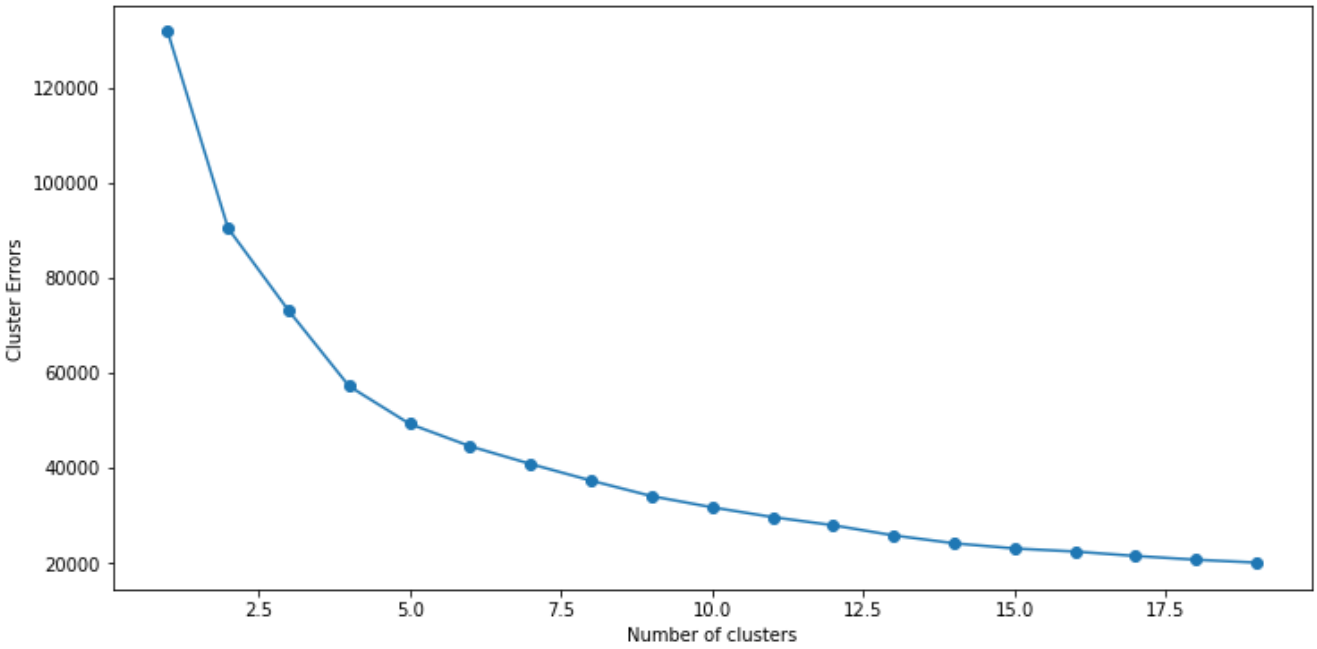
9.Model Building

We will use KMeans Clustering algorithm for building our model.

9.1 KMeans Clustering

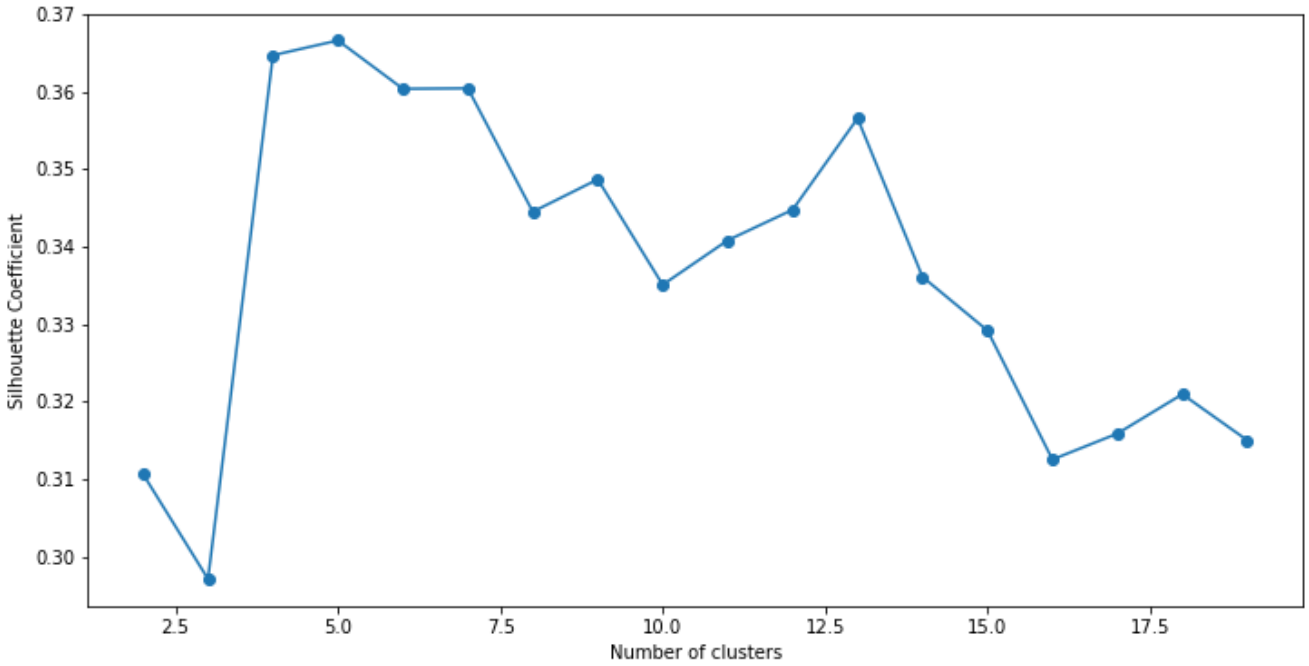
Since we don't know the K value (no. of clusters) for performing KMeans Clustering, we will find the optimal K value from Elbow method and Silhouette Coefficient score.

9.1.1 Elbow Method



Plotting of cluster errors w.r.t number of clusters

9.1.2 Silhouette Coefficient

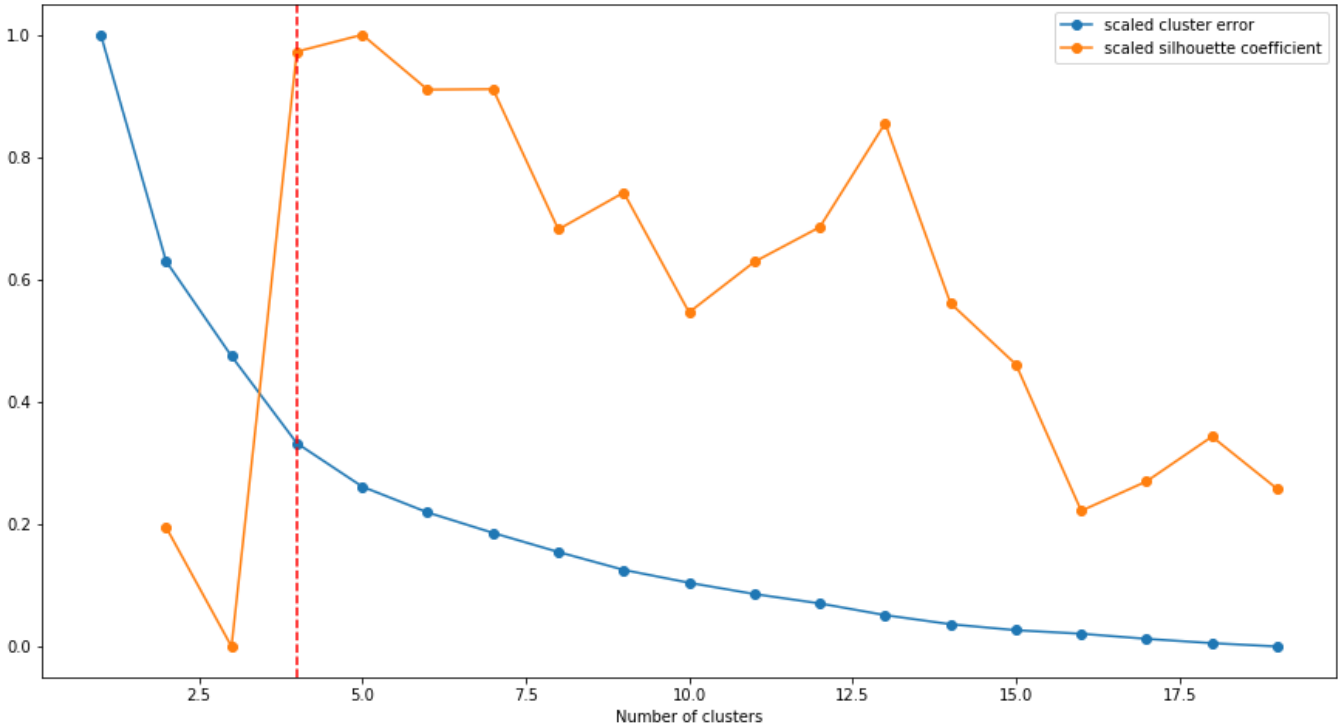


Plotting of silhouette coefficient w.r.t number of clusters

In order to select the optimal value of K with ease, it would be better to plot both cluster errors and silhouette coefficients together w.r.t number of clusters.

We will be using Minmax scaler to make cluster errors and silhouette coefficients of each cluster to same scale.

$$x_{scaled} = (x - x_{min}) / (x_{max} - x_{min})$$



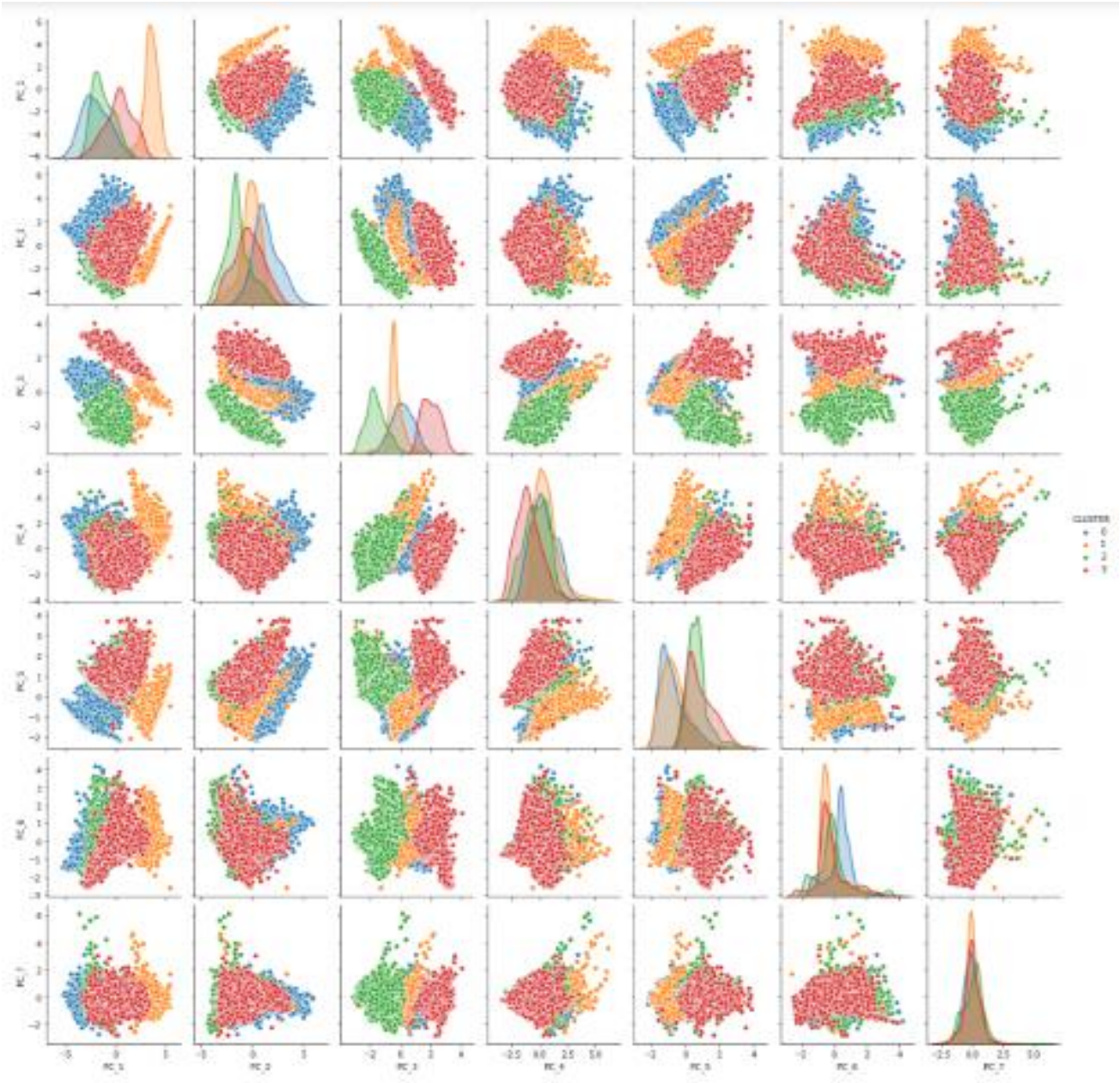
Plotting of both cluster errors and silhouette coefficient w.r.t number of clusters

Cluster error line is not decreasing steeply after K=4 in Elbow method and there is a sharp increase in Silhouette coefficient line at K=4.

Based on Elbow Method and Silhouette Coefficient score we choose optimum K value as 4.

	PC_1	PC_2	PC_3	PC_4	PC_5	PC_6	PC_7	CLUSTER
0	0.065509	-2.351500	-0.808282	-0.852112	-0.024910	0.220955	0.235479	2
1	3.580346	-0.125000	-0.183906	1.429836	-0.430823	-0.743903	-0.081130	1
2	-1.401315	0.397136	2.493168	-2.132849	0.301884	-0.633239	-0.234572	3
3	1.078138	-0.858779	1.743438	-1.650217	0.900553	0.865316	-0.821598	3
4	1.147298	-1.170643	1.655936	-1.946654	-0.242003	-0.845406	0.665289	3

10.Result



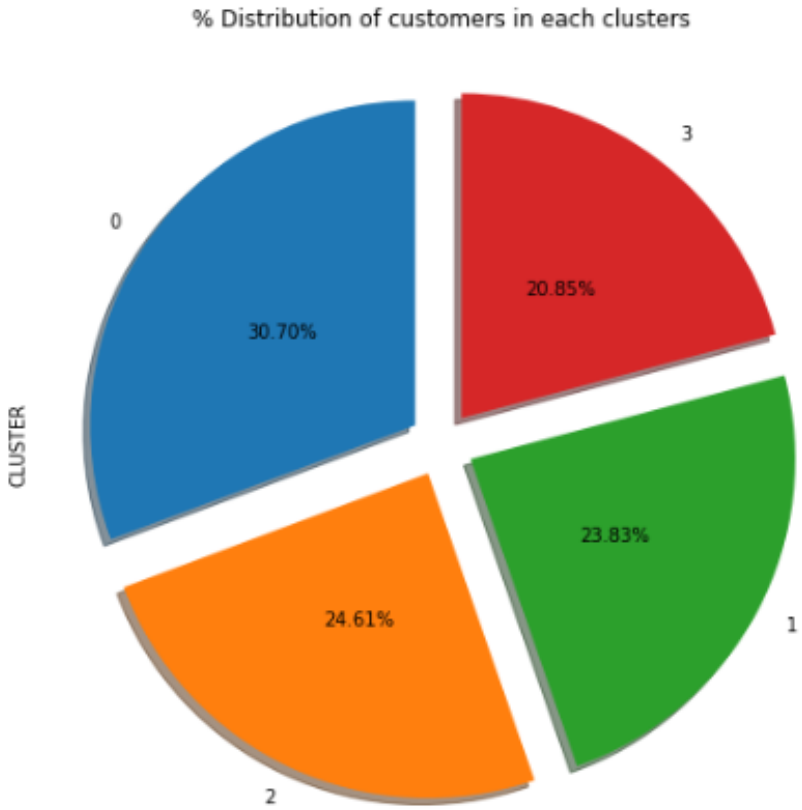
Pairplot of credit_model

CLUSTER	0	1	2	3
BALANCE	1797.415291	2218.201535	767.622799	1414.927967
BALANCE_FREQUENCY	0.947185	0.886537	0.818325	0.833311
PURCHASES_FREQUENCY	0.804533	0.009121	0.706919	0.322070
ONEOFF_PURCHASES_FREQUENCY	0.439519	0.001664	0.000038	0.321847
PURCHASES_INSTALLMENTS_FREQUENCY	0.644042	0.006572	0.670590	0.000300
CASH_ADVANCE_FREQUENCY	0.103917	0.281653	0.042577	0.119241
CASH_ADVANCE_TRX	2.733261	6.739803	0.908761	2.780279
PURCHASES_TRX	33.228894	0.132677	12.132547	7.143087
CREDIT_LIMIT	5739.644443	4097.124551	3314.676979	4506.824840
PAYMENTS	2549.783891	1695.565167	954.720176	1492.466395
MINIMUM_PAYMENTS	843.417914	1018.270772	817.355590	711.638990
PRC_FULL_PAYMENT	0.189192	0.043093	0.259681	0.102815
TENURE	11.738719	11.333802	11.463913	11.464094
MONTHLY_AVG_PURCHASES	194.248620	0.576532	47.936948	70.009513
MONTHLY_AVG_CASH_ADVANCE	65.986337	189.320011	30.857540	76.754424
LIMIT_USAGE	0.352668	0.578690	0.259884	0.377550
PAYMENTS_MIN_PAYMENTS_RATIO	7.283627	9.776577	13.564767	5.544060
BOTH ONEOFF & INSTALMENT	1.000000	0.009376	0.000454	0.002680
INSTALMENT	0.000000	0.027192	0.999546	0.000000
NONE	0.000000	0.957337	0.000000	0.000000
ONEOFF	0.000000	0.006095	0.000000	0.997320

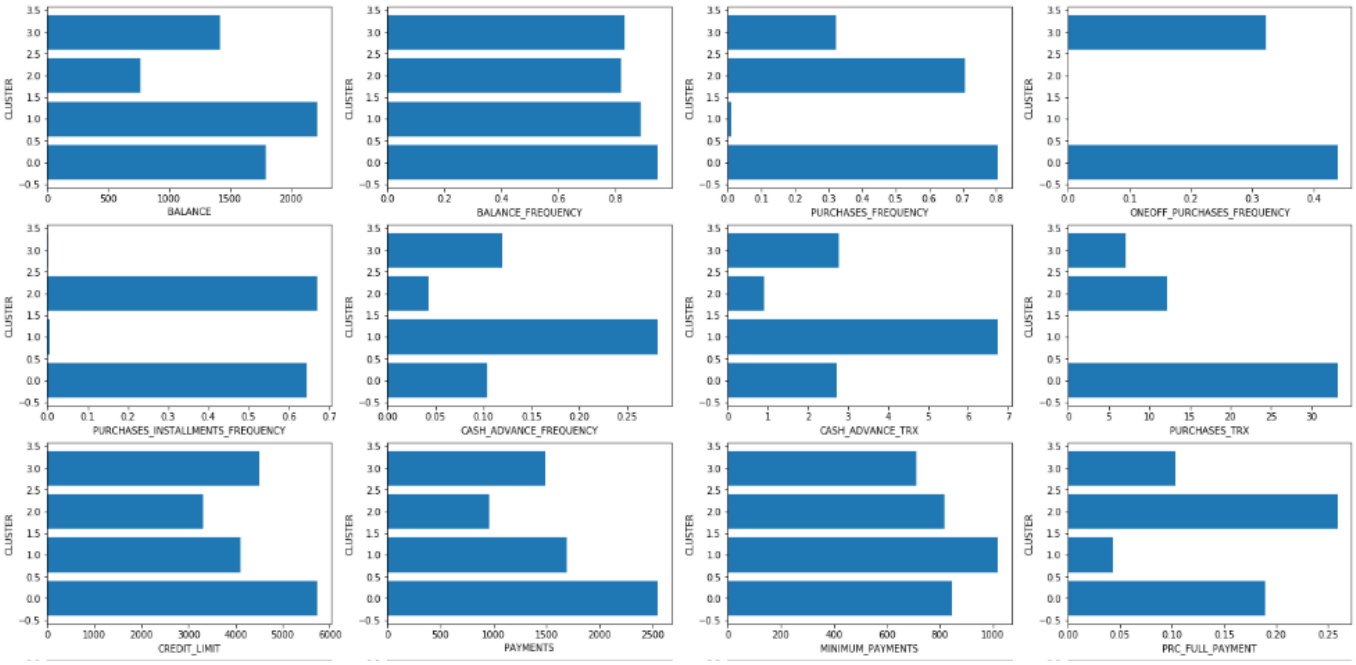
Summary of each clusters in python

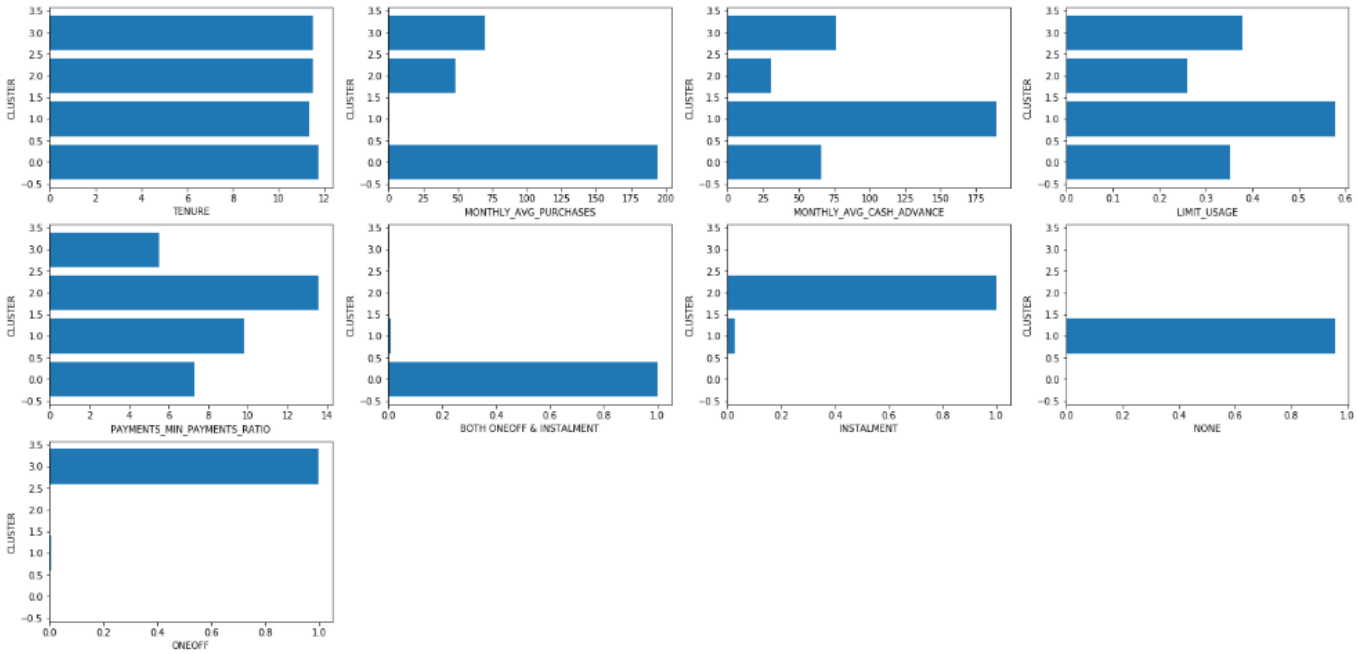
	CLUSTER 1	CLUSTER 2	CLUSTER 3	CLUSTER 4
BALANCE	1797.4152913	7.676228e+02	1.417825e+03	2.216042e+03
BALANCE_FREQUENCY	0.9471849	8.183246e-01	8.333998e-01	8.864833e-01
PURCHASES_FREQUENCY	0.8045327	7.069192e-01	3.219423e-01	9.086455e-03
ONEOFF_PURCHASES_FREQUENCY	0.4395187	3.782705e-05	3.217192e-01	1.625658e-03
PURCHASES_INSTALLMENTS_FREQUENCY	0.6440419	6.705902e-01	2.996909e-04	6.574622e-03
CASH_ADVANCE_FREQUENCY	0.1039174	4.257680e-02	1.201565e-01	2.819228e-01
CASH_ADVANCE_TRX	2.7332606	9.087608e-01	2.787359e+00	6.735460e+00
PURCHASES_TRX	33.2288937	1.213255e+01	7.139796e+00	1.322702e-01
CREDIT_LIMIT	5739.6444430	3.314677e+03	4.509767e+03	4.093522e+03
PAYMENTS	2549.7838913	9.547202e+02	1.492671e+03	1.695481e+03
MINIMUM_PAYMENTS	838.7537474	8.086389e+02	7.024614e+02	1.013262e+03
PRC_FULL_PAYMENT	0.1891922	2.596807e-01	1.027597e-01	4.311295e-02
TENURE	11.7387191	1.146391e+01	1.146438e+01	1.133349e+01
MONTHLY_AVG_PURCHASES	194.2486199	4.793695e+01	6.997814e+01	5.714339e-01
MONTHLY_AVG_CASH_ADVANCE	65.9863369	3.085754e+01	7.692823e+01	1.892206e+02
LIMIT_USAGE	0.3526680	2.598839e-01	3.777130e-01	5.786462e-01
PAYMENTS_MIN_PAYMENTS_RATIO	7.3242920	1.375490e+01	5.565306e+00	9.811822e+00
BOTH ONEOFF & INSTALMENT	1.0000000	4.539265e-04	2.678093e-03	9.380863e-03
INSTALMENT	0.0000000	9.995461e-01	0.000000e+00	2.720450e-02
NONE	0.0000000	0.000000e+00	0.000000e+00	9.577861e-01
ONEOFF	0.0000000	0.000000e+00	9.973219e-01	5.628518e-03

Summary of each clusters in R

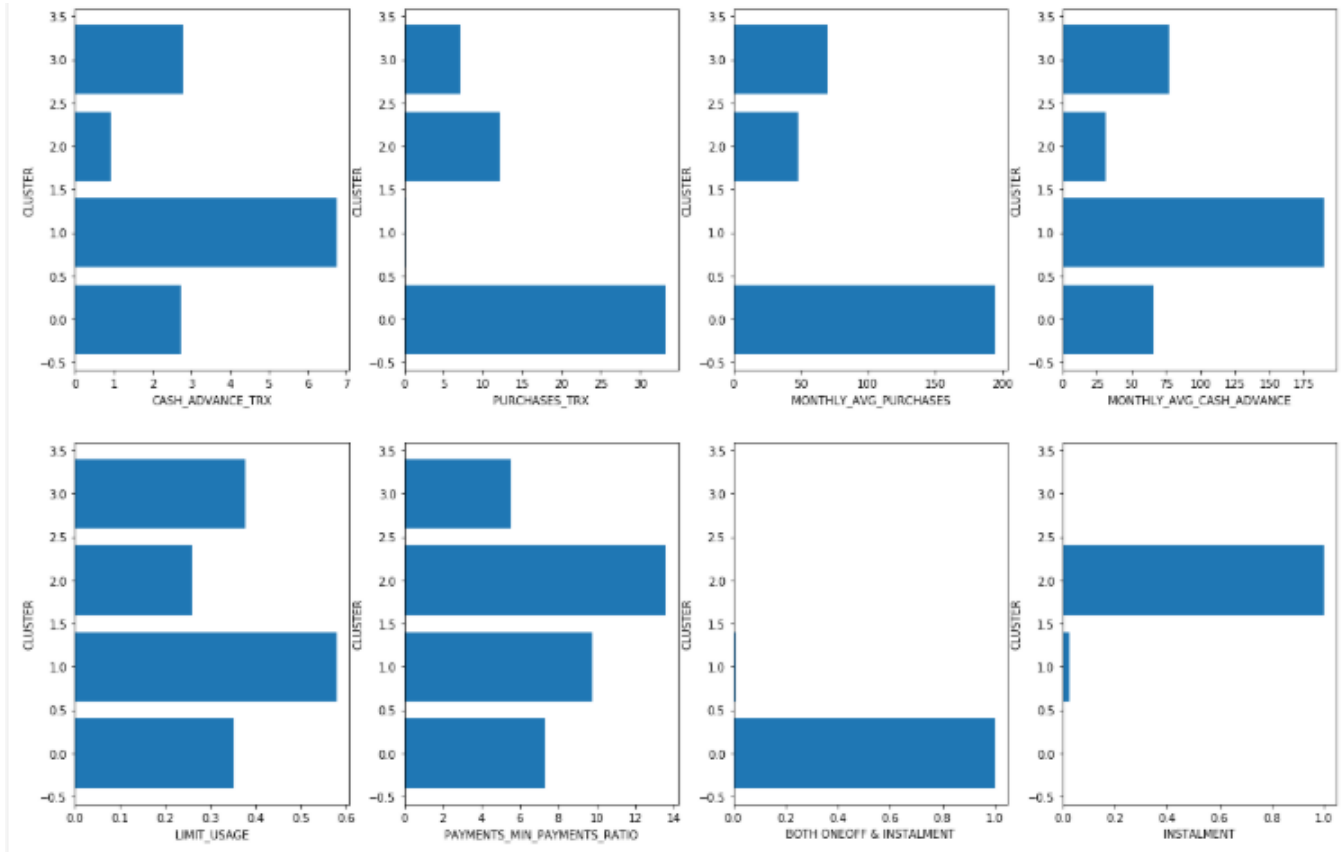

% Distribution of customers in each clusters
CLUSTER
0
30.70%
3
20.85%
1
23.83%
2
24.61%

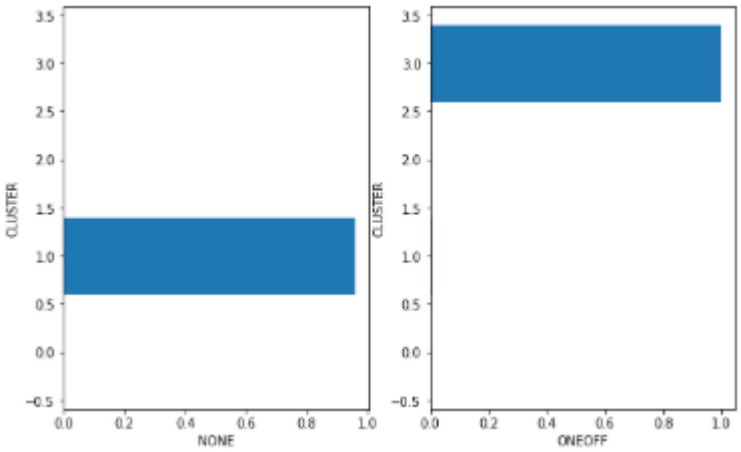
Pie chart


CLUSTER
BALANCE
BALANCE_FREQUENCY
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS
PRC_FULL_PAYMENT



Bar graph of all columns w.r.t CLUSTERS





Bar graph of all KPIs w.r.t CLUSTERS

11.Inference

Cluster 0 :

Group of customers who :-

- does both instalment as well as one_off purchases.
- have relatively high Balance statement.
- have the highest credit limit.
- have the highest Monthly Average Purchases.
- have relatively low Monthly Average Cash Advance.
- have relatively low Payments to Minimum Payments Ratio.
- This group is about 30.70% of the total customer base.

Cluster 1 :

Group of customers who :-

- does no purchase transaction most of the time.
- have the highest Balance statement.
- does the least number of full payments of statement balance.
- have highest Monthly Average Cash Advance.
- have the highest credit card utilisation.
- have relatively better Payments to Minimum Payments Ratio.
- This group is about 23.83% of the total customer base.

Cluster 2 :

Group of customers who :-

- does only instalment purchase transaction.
- have the least Balance statement.
- have the least credit limit.
- does the highest number of full payments of statement balance.
- have low Monthly Average Purchases.
- have least Monthly Average Cash Advance.
- have the least credit card utilisation.
- have the highest Payments to Minimum Payments Ratio.
- This group is about 24.61% of the total customer base.

Cluster 3 :

Group of customers who :-

- does only one-off purchase transaction.
- have moderate Balance statement.
- have relatively low Monthly Average Purchases.
- have relatively low Monthly Average Cash Advance.
- have the least Payments to Minimum Payments Ratio.
- This group is about 20.85% of the total customer base.

Note:-

- Cluster 0 in python is Cluster 1 in R.
- Cluster 1 in python is Cluster 4 in R.
- Cluster 2 in python is Cluster 2 in R.
- Cluster 3 in python is Cluster 3 in R.