edWisor
Get Skilled.Get Hired

CREDIT CARD SEGMENTATION PROJECT

Using R and Python

Submitted by: Sanjeev Kumar Pandit
sanjeev110795@gmail.com

Contents

1. Pre requisite And Steps to Execute Codes

Pre requisite:

A. To run the R project in the system should have R &R-studio; Below are the versions of both which are used:
   i) R version 3.6.1 (64 bit).
   ii) R-studio IDE (Version 1.2.5001).
B. To run the Python (ipynb) file in the system should have Anaconda Navigator to be installed Below are the versions of software used:
   i) Jupyter Notebook (Version – 6.0.0)
   ii) Python Version – 3.7.5
C. Download the file and save it in your local. (In my case):
   i) C:\Users\Sanjeev\Desktop\Edwisor\credit_card_project\ Credit_Card_Segmentation_Python_Code.ipynb
   ii) C:\Users\Sanjeev\Desktop\Edwisor\credit_card_project\ Credit_Card_Segmentation_R_Code.R
   iii) C:\Users\Sanjeev\Desktop\Edwisor\credit_card_project\ credit-card-data.csv
   iv) C:\Users\Sanjeev\Desktop\Edwisor\credit_card_project\ Credit_Card_Segmentation_report.docx

Steps to execute the Python code:

i) Python please open Anaconda prompt type Jupyter notebook, load .ipynb file and run the Code.
ii) If your jupyter notebook is not able to load FancyImpute Library for KNN Imputation then in the code please comment KNN imputation and just un-comment mean Imputation.
iii) .py file also attached please download the same run in cmd suggested version of python is 3.6 because kNN fancy impute supported only in python 3.6.

Steps to execute the R-code:

i) In R-studio: For R code open R file and run the R-script to get the output in console and plots in Plot window.
ii) Please make sure that all that library I have used in the code is installed in your system

For Complete project report Open word/ pdf document “Credit_Card_Segmentation_report. Pdf or . docx”

2. Introduction

1.1. Problem Statement

This objective of this case study to develop a customer segmentation to define marketing strategy. The sample dataset summarizes the usage behaviour of about 8950 active credit card holders during the last 6 months. The file is at a customer level with 18 behavioural variables.

1.2. Data

Understanding of data is the very first and important step in the process of finding solution of any business problem. Let's have a quick preview of data. Will discuss in detail about the data understanding in the CRISP-DM process section.

Let's understand shape of dataset:

```
credit.shape
```

```
(8950, 18)
```

Preview of data:

```
credit.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333

ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT
0.000000	0.083333	0.000000	0	2	1000.0
0.000000	0.000000	0.250000	4	0	7000.0
1.000000	0.000000	0.000000	0	12	7500.0
0.083333	0.000000	0.083333	1	1	7500.0
0.083333	0.000000	0.000000	0	1	1200.0

PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
201.802084	139.509787	0.000000	12
4103.032597	1072.340217	0.222222	12
622.066742	627.284787	0.000000	12
0.000000	NaN	0.000000	12
678.334763	244.791237	0.000000	12

2. Process and Requirement Specification

2.1. Process:

1. Understanding the Data
2. Exploratory Data Analysis
3. Feature Engineering
4. Missing Value and Outlier Treatment
5. Standardizing the data
6. Reducing Dimensions using PCA
7. Applying K-Means
8. Selecting optimum number of clusters using Silhouette Coefficient

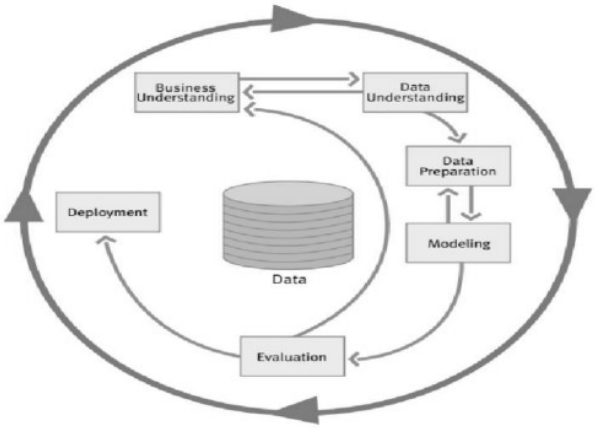
2.2. Software Requirements:

1. Python
2. Jupyter Notebook
3. R
4. R Studio

3. Methodology

CRISP-DM Process: CRISP-DM stands for cross-industry process for data mining. The CRISP-DM methodology provides a structured approach to planning a data mining works. The project follows CRISP DM process to develop the model for the given problem. It involves the following major steps:

1. Business understanding.
2. Data understanding.
3. Data preparation.
4. Modelling.
5. Evaluation.
6. Deployment



3.1. Data Understanding

DATA DICTIONARY:

- CUST_ID: Credit card holder ID
- BALANCE: Monthly average balance (based on daily balance averages)
- BALANCE_FREQUENCY: Ratio of last 12 months with balance
- PURCHASES: Total purchase amount spent during last 12 months
- ONEOFF_PURCHASES: Total amount of one-off purchases
- INSTALLMENTS_PURCHASES: Total amount of installment purchases

- CASH_ADVANCE: Total cash-advance amount
- PURCHASES_ FREQUENCY: Frequency of purchases (Percent of months with at least one
- purchase)
- ONEOFF_PURCHASES_FREQUENCY: Frequency of one-off-purchases
- PURCHASES_INSTALLMENTS_FREQUENCY: Frequency of installment purchases
- CASH_ADVANCE_ FREQUENCY: Cash-Advance frequency
- CASH_ADVANCE_TRX: Number of cash-advance transaction
- PURCHASES_TRX: Number of 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_PAYMENTS: Percentage of months with full payment of the due statement balance
- TENURE: Number of months as a customer

3.2. Data Pre-Processing:

Data pre-processing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviour trends, and is likely to contain many errors. Data pre-processing is a proven method of resolving such issues. Data pre-processing prepares raw data for further processing.

Data pre-processing is defining the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we pre-process our data before feeding it into our model. As we already know the quality of our inputs decide the quality of our output.

So, once we have got our business hypothesis ready, it makes sense to spend lot of time and efforts here. Approximately, data exploration, cleaning and preparation can take up to 60-70% of our total project time. This process is often called as Exploratory Data Analysis Listed below are data pre-processing techniques used for this Project

1) Data Exploration and Cleaning

2) Missing Value Analysis

3) Outlier analysis

4) Feature Engineering

5) Feature Selection

6) Feature Scaling

Let’s discuss /explain this technique in detail.

3.2.1. Data Exploration and Cleaning

Data Pre-processing is the very first step which comes with any data science project is data exploration and cleaning,

Removing the Cust ID Column as It Is not useful for our analysis (Non-Numeric)

Check the range of values in frequency columns:

Python Code:

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

ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT
8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8949.000000
0.202458	0.364437	0.135144	3.248827	14.709832	4494.449450
0.298336	0.397448	0.200121	6.824647	24.857649	3638.815725
0.000000	0.000000	0.000000	0.000000	0.000000	50.000000
0.000000	0.000000	0.000000	0.000000	1.000000	1600.000000
0.083333	0.166667	0.000000	0.000000	7.000000	3000.000000
0.300000	0.750000	0.222222	4.000000	17.000000	6500.000000
1.000000	1.000000	1.500000	123.000000	358.000000	30000.000000

PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
8950.000000	8637.000000	8950.000000	8950.000000
1733.143852	864.206542	0.153715	11.517318
2895.063757	2372.446607	0.292499	1.338331
0.000000	0.019163	0.000000	6.000000
383.276166	169.123707	0.000000	12.000000
856.901546	312.343947	0.000000	12.000000
1901.134317	825.485459	0.142857	12.000000
50721.483360	76406.207520	1.000000	12.000000

R code:

```
> ##########Check frequency range in cloumns#############
>
> range(credit$BALANCE_FREQUENCY)#0-1
[1] 0 1
> range(credit$PURCHASES_FREQUENCY)#0-1
[1] 0 1
> range(credit$ONEOFF_PURCHASES_FREQUENCY)#0-1
[1] 0 1
> range(credit$PURCHASES_INSTALLMENTS_FREQUENCY)#0-1
[1] 0 1
> range(credit$CASH_ADVANCE_FREQUENCY)
[1] 0.0 1.5
```

From the output we can see that frequency range for "CASH_ADVANCE_FREQUENCY " is 0 to 1.5.

As the frequency can't be greater than 1. So, the observation for which frequency is greater than 1 is wrong.

We can correct the data by making the frequency equals to 1 for the observation for which it is greater than 1.

Python Code:

Replace the frequency > 1 with 1

```
credit['CASH_ADVANCE_FREQUENCY'].values[credit['CASH_ADVANCE_FREQUENCY'].values > 1] = 1
```

R code:

```
##########Replace the frequency > 1 with 1 ##########

credit$CASH_ADVANCE_FREQUENCY[credit$CASH_ADVANCE_FREQUENCY > 1] <- 1
```

3.2.2. Missing Value Analysis

In real world missing value is the common occurrence of incomplete observations in an asset of data. Missing values can arise due to many reasons, error in uploading data, unable to measure an observation etc. Due to presence of missing values in the form of 0, NA or NAN. These will affect the accuracy of model which we are building. Hence it is necessary to check for any missing values in the given data. Let's check of the dataset for missing values and identify what type are they? Listed below are some of the missing values in different forms:

- Values containing '0'- These will first need to be changed to NA and Nan in R and python respectively.
- Blank spaces that are taken as NA and NaN in R and Python respectively.

Depending on the percentage of missing values we can decide if we need to keep a variable or drop it based on the following conditions:

- Missing value percentage < 30% - We can include the variable
- Missing value percentage > 30 % - We need to remove those variable/s because even if we impute values, they are not the actual values, the variable will not contain actual values and hence will not contribute effectively to our model.

For the given train dataset, we can see that we have only two variables MINIMUM_PAYMENTS and CREDIT_LIMIT with missing values and the percentage of missing value is less as per the general practice mentioned above, so we are going to include those variables then in later part of the analysis will impute missing values. Below picture shows no. of missing Values and Percentage in train & test dataset

Python Code:

```
# Check missing value after imputing with KNN imputation
# Find missing value in each feature
missing_val = credit.isnull().sum().sort_values(ascending=False)
#Reset index
missing_val = missing_val.reset_index()

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

#Calculate percentage
missing_val['Missing_percentage'] = (missing_val['Missing_Value_count']/len(credit))*100

#descending order
missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
missing_val
```

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

R Code:

```
########Missing Values Analysis#####################

missing_val = data.frame(apply(credit,2,function(x){sum(is.na(x))}))

missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(credit)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]ss
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
write.csv(missing_val, "Miising_perc.csv", row.names = F)
```

```
                           Columns Missing_percentage
1                  MINIMUM_PAYMENTS         3.49720670
2                      CREDIT_LIMIT         0.01117318
3                           BALANCE         0.00000000
4                 BALANCE_FREQUENCY         0.00000000
5                         PURCHASES         0.00000000
6                  ONEOFF_PURCHASES         0.00000000
7           INSTALLMENTS_PURCHASES          0.00000000
8                      CASH_ADVANCE         0.00000000
9              PURCHASES_FREQUENCY          0.00000000
10       ONEOFF_PURCHASES_FREQUENCY         0.00000000
11 PURCHASES_INSTALLMENTS_FREQUENCY         0.00000000
12          CASH_ADVANCE_FREQUENCY          0.00000000
13                CASH_ADVANCE_TRX          0.00000000
14                   PURCHASES_TRX          0.00000000
15                        PAYMENTS          0.00000000
16                PRC_FULL_PAYMENT          0.00000000
17                          TENURE          0.00000000
```

Now the question is which method we should use to impute missing values?

We have four method to impute missing value. MEAN, MEDIAN, MODE and KNN imputation.

MODE method is used to impute for Categorial variable.

As MINIMUM_PAYMENTS and CREDIT_LIMIT variable values are continuous.

So, we left with three method.

To select the best method we are going to perform one experiment.

In MINIMUM_PAYMENT variable for 500 th observation I have note down the actual value and made the value as NA. And with the help of all three method I have imputed the value and note down the imputed value for that particular cell.

```
# credit['MINIMUM_PAYMENTS'].loc[500]
# Actual = 457.255
# Mean = 1199.94
# Median = 873.09
# KNN Imputation (K=4) = 469.99
```

From the above table we can see that value imputed with KNN Imputation is very close to actual value as compared to other imputation method.

Value of K is taken as 4 as the variable is continuous

- k= odd (for categorical)
- K= even (for continuous)

So, we will impute the value with KNN imputation method.

Python Code:

```
#Impute with mean
#credit = credit.fillna(credit.mean())

#Impute with median
#credit = credit.fillna(credit.median())

#Apply KNN imputation algorithm
credit = pd.DataFrame(KNN(k = 4).fit_transform(credit), columns = credit.columns)
```

Check missing value after imputation.

	Variables	Missing_Value_count	Missing_percentage
0	TENURE	0	0.0
1	PRC_FULL_PAYMENT	0	0.0
2	MINIMUM_PAYMENTS	0	0.0
3	PAYMENTS	0	0.0
4	CREDIT_LIMIT	0	0.0
5	PURCHASES_TRX	0	0.0
6	CASH_ADVANCE_TRX	0	0.0
7	CASH_ADVANCE_FREQUENCY	0	0.0
8	PURCHASES_INSTALLMENTS_FREQUENCY	0	0.0
9	ONEOFF_PURCHASES_FREQUENCY	0	0.0
10	PURCHASES_FREQUENCY	0	0.0
11	CASH_ADVANCE	0	0.0
12	INSTALLMENTS_PURCHASES	0	0.0
13	ONEOFF_PURCHASES	0	0.0
14	PURCHASES	0	0.0
15	BALANCE_FREQUENCY	0	0.0
16	BALANCE	0	0.0

R Code:

```
########## Missing value Impuatation ############

#Mean Method
#credit$MINIMUM_PAYMENTS[is.na(credit$MINIMUM_PAYMENTS)] = mean(credit$MINIMUM_PAYMENTS, na.rm = T)
#credit$CREDIT_LIMIT[is.na(credit$CREDIT_LIMIT)] = mean(credit$CREDIT_LIMIT, na.rm = T)

#Median Method
#credit$MINIMUM_PAYMENTS[is.na(credit$MINIMUM_PAYMENTS)] = median(credit$MINIMUM_PAYMENTS, na.rm = T)
#credit$CREDIT_LIMIT[is.na(credit$CREDIT_LIMIT)] = median(credit$CREDIT_LIMIT, na.rm = T)


# kNN Imputation
credit = knnImputation(credit, k = 4)
```

3.2.3. Outlier Analysis and Exploratory data analysis

An Outlier is any inconsistent or abnormal observation in a variable of dataset that deviates from the rest of the observations.

These inconsistent observations can be due to manual error, poor quality/ malfunctioning equipment's, Experimental error or correct but exceptional data based on business use case.

It can cause an error in predicting the target variable/s. Hence, we need to check for the outliers and either remove the observations containing them or replace them with NA, then impute or set upper limits/lower limits or mean/median values imputation.

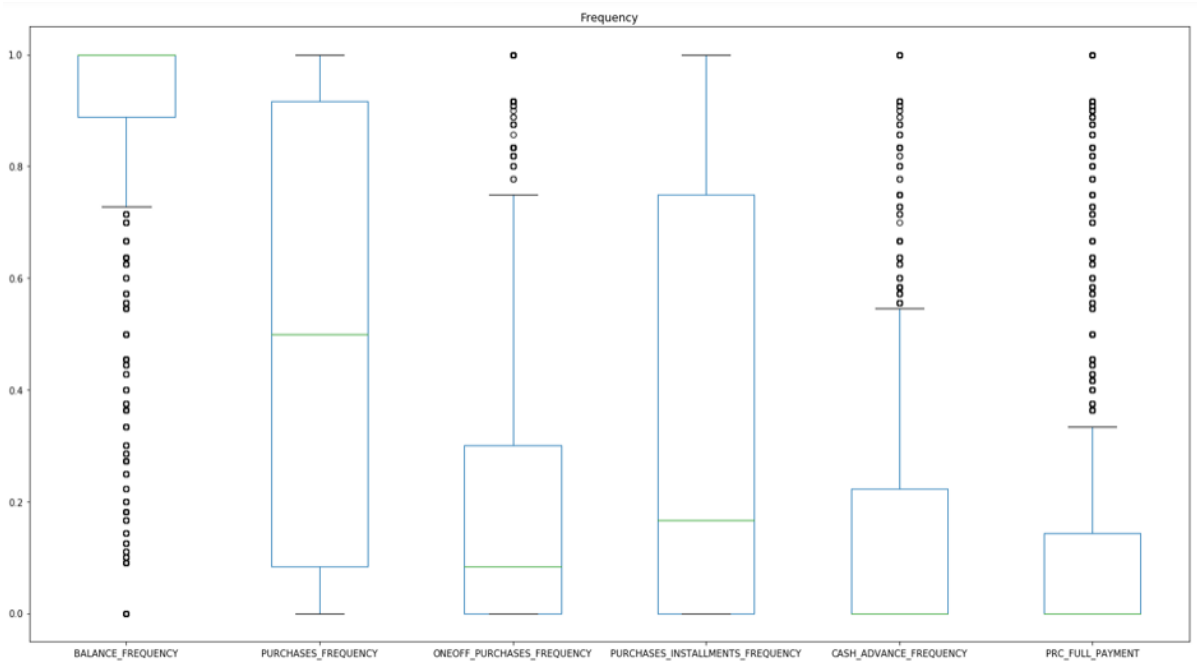
For Outlier analysis, Boxplot to visualize and summary descriptive statistics to check range of each numeric variable and sorted the variables to detect some strange values like zeros and negative values.

Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modelling or hypothesis testing task

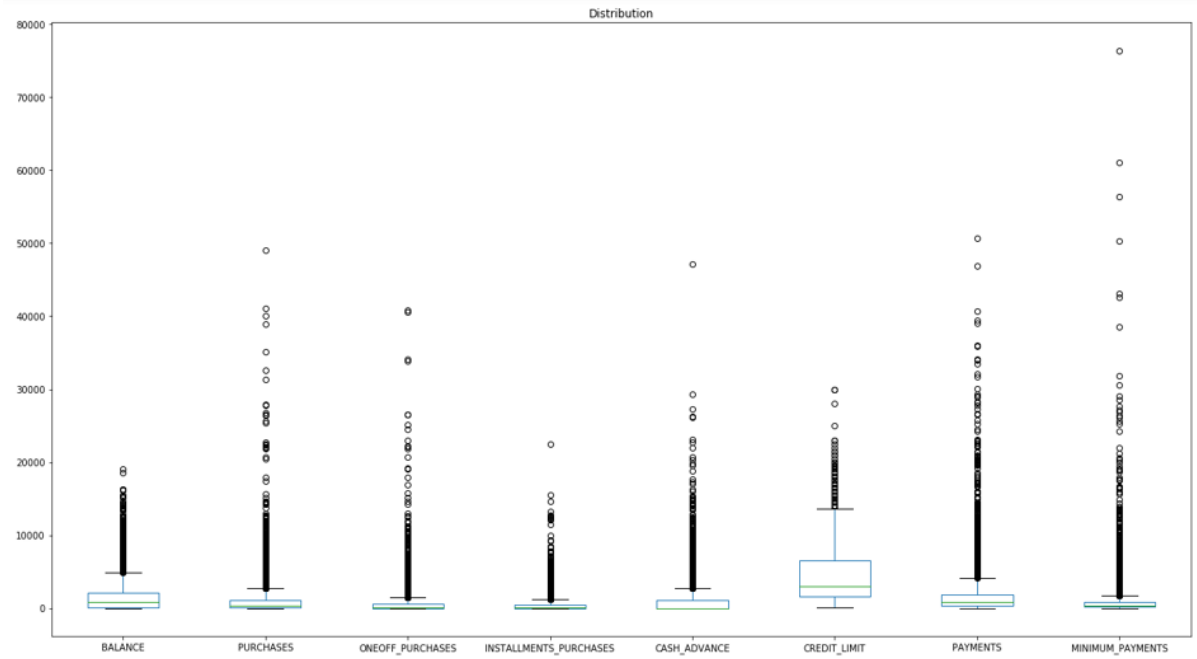
Python Code:

Box plot to check the outlier in dataset

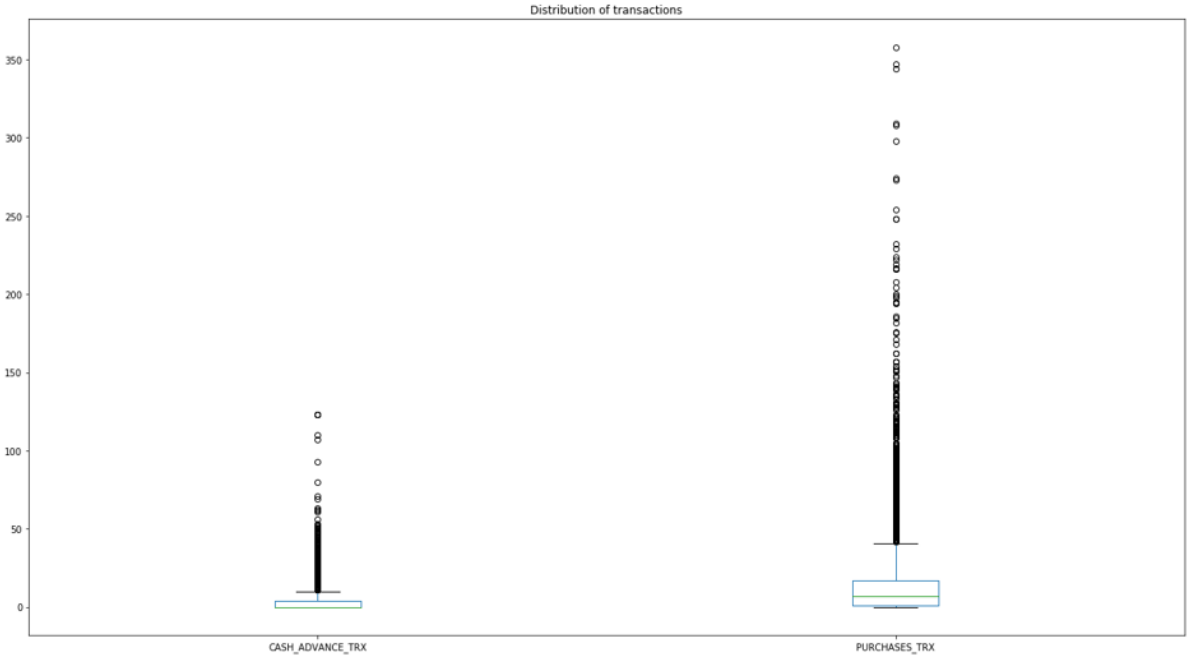
```
#let´s see how are distributed the frequency variables

credit[['BALANCE_FREQUENCY',
 'PURCHASES_FREQUENCY',
 'ONEOFF_PURCHASES_FREQUENCY',
 'PURCHASES_INSTALLMENTS_FREQUENCY',
 'CASH_ADVANCE_FREQUENCY',
'PRC_FULL_PAYMENT']].plot.box(figsize=(18,10),title='Frequency',legend=True);
plt.tight_layout()
```




```
#Let´s see how are distributed the numeric variables

credit[['BALANCE',
 'PURCHASES',
 'ONEOFF_PURCHASES',
 'INSTALLMENTS_PURCHASES',
 'CASH_ADVANCE',
 'CREDIT_LIMIT',
 'PAYMENTS',
 'MINIMUM_PAYMENTS'
]].plot.box(figsize=(18,10),title='Distribution',legend=True);
plt.tight_layout()
```

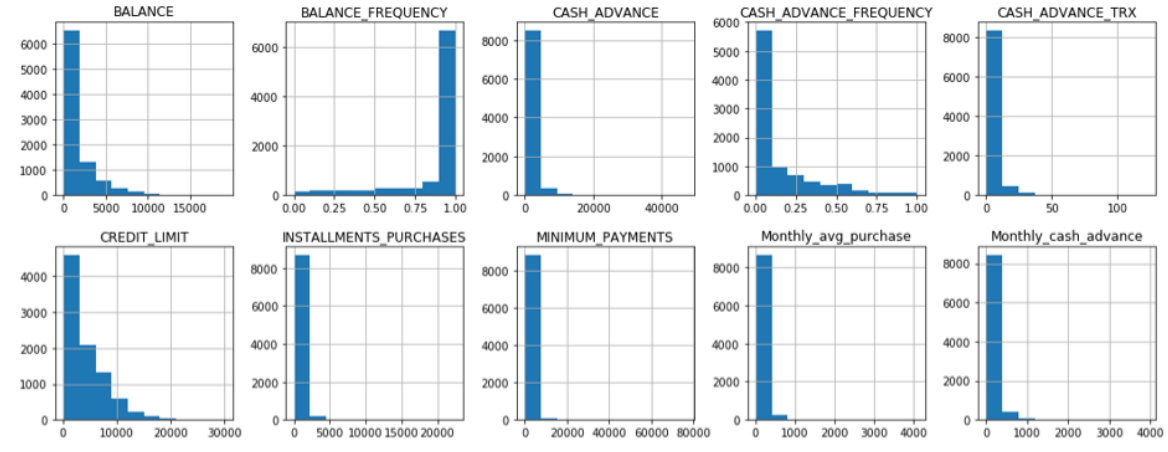


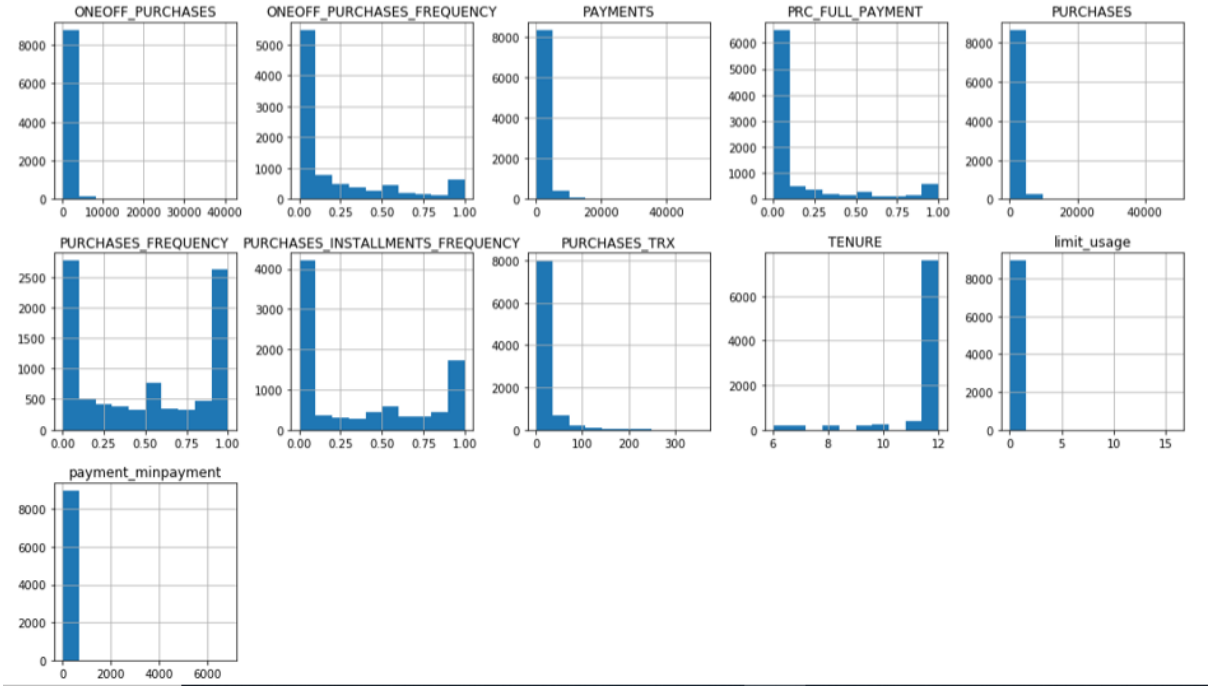
```
#let´s see how are distributed the numeric variables

credit[[ 'CASH_ADVANCE_TRX',
 'PURCHASES_TRX'
]].plot.box(figsize=(18,10),title='Distribution of transactions',legend=True);
plt.tight_layout()
```



From above box plot we can see that we have so many outliers in the dataset.

```
# Exploratory Data Analysis
credit.hist(figsize=(18,18));
```





By looking above histogram plot we can see that the data in most of the features is skewed.

As the data set is related to credit card expenditure. People from middle class to upper class use credit card and based on their income and requirement their expenditure is different. It may vary from very low to very high. So, we can't delete the observation as there are so may outliers in data. If we drop the observation or clap or assigning a new value. we may end up with some wrong insights or we may lose some potential customers.

So, the best way to use log transformation which will reduce the outlier effect and make dataset normally distributed.

To deal with outlier and skewed data later after deriving new KPI's, we are going to perform log transformation to reduce the outliers and make the data normally distributed.

3.2.4. Feature Engineering

Based on the domain knowledge, we came up with new features that might affect the segmentation. We will create five new features:

Python Code:

Monthly_avg_purchase

```
In [14]: credit['Monthly_avg_purchase']=credit['PURCHASES']/credit['TENURE']
```

Monthly_cash_advance Amount

```
In [15]: credit['Monthly_cash_advance']=credit['CASH_ADVANCE']/credit['TENURE']
```

Purchases by type (one-off, installments)

- To find what type of purchases customers are making on credit card

```
In [21]: def purchase(credit):
             if (credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']==0):
                 return 'none'
             if (credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']>0):
                  return 'both_oneoff_installment'
             if (credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES']==0):
                 return 'one_off'
             if (credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES']>0):
                 return 'istallment'
```

```
In [22]: credit['purchase_type']=credit.apply(purchase,axis=1)
```

```
In [23]: credit['purchase_type'].value_counts()
```

```
Out[23]: both_oneoff_installment    2774
         istallment                 2260
         none                       2042
         one_off                    1874
         Name: purchase_type, dtype: int64
```

Limit_usage (balance to credit limit ratio) credit card utilization

- Lower value implies cutomers are maintaing thier balance properly. Lower value means good credit score

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

Payments to minimum payments ratio

```
[26]: #PAYMENT_MINPAYMENT
      #The where clause is being used to avoid div by zero error
      credit['payment_minpayment'] = np.where(credit['MINIMUM_PAYMENTS']== 0, credit['PAYMENTS'],
                                              credit['PAYMENTS']/credit['MINIMUM_PAYMENTS'])
```

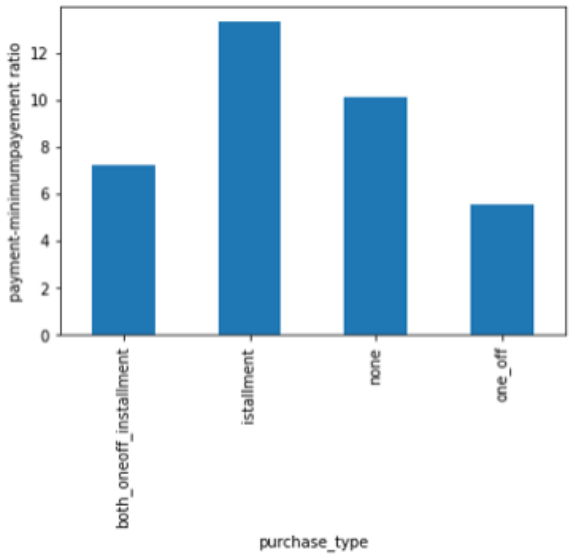
R Code:

```
#derived KPI's variables

cc2$monthly_avg_purchase <- cc2$PURCHASES/cc2$TENURE
cc2$monthly_cash_advance <- cc2$CASH_ADVANCE/cc2$TENURE
cc2$limit_usage <- cc2$BALANCE/cc2$CREDIT_LIMIT

cc2$purchase_type <- ifelse(cc2$ONEOFF_PURCHASES==0 & cc2$INSTALLMENTS_PURCHASES==0,'NONE',
                            ifelse(cc2$ONEOFF_PURCHASES>0 & cc2$INSTALLMENTS_PURCHASES==0,'one_off',
                                   ifelse(cc2$ONEOFF_PURCHASES==0 & cc2$INSTALLMENTS_PURCHASES>0,'installment',
                                          ifelse(cc2$ONEOFF_PURCHASES>0 & cc2$INSTALLMENTS_PURCHASES>0,'both','NA'))))
cc2$purchase_type_none <- ifelse(cc2$purchase_type=='NONE',1,0)
cc2$purchase_type_one_off <- ifelse(cc2$purchase_type=='one_off',1,0)
cc2$purchase_type_installment <- ifelse(cc2$purchase_type=='installment',1,0)
cc2$purchase_type_both <- ifelse(cc2$purchase_type=='both',1,0)

cc2$payment_minpayment <- cc2$PAYMENTS/cc2$MINIMUM_PAYMENTS
```

3.2.5. Insights of Derived KPI's

Average payment_minpayment ratio for each purchse type. ¶

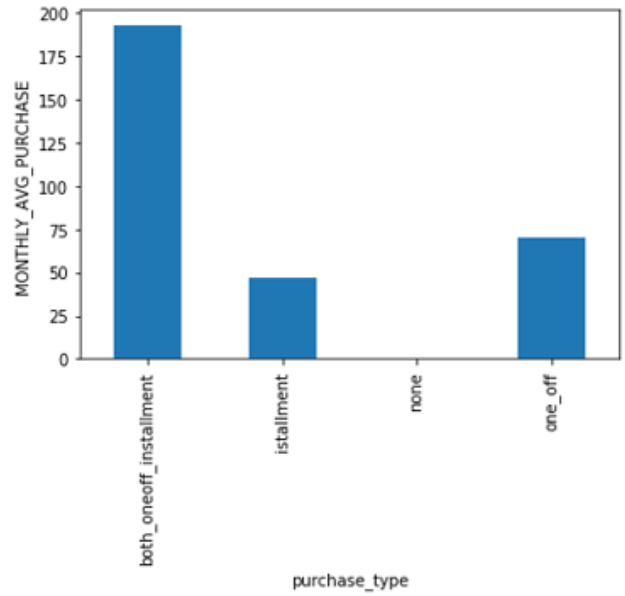
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['payment_minpayment'])).plot.bar()
plt.ylabel('payment-minimumpayement ratio')
plt.show()
```



Insights : Customers with installments have highest payment-minimum payement ratio

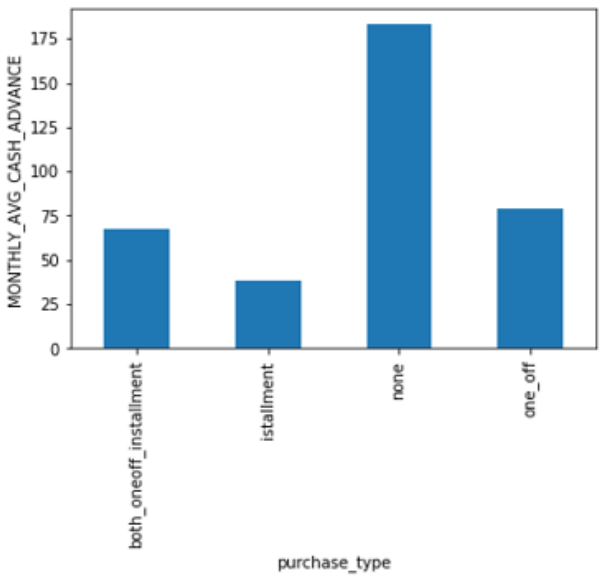
Customers with installments only purchases are paying their dues fast.

```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['Monthly_avg_purchase'])).plot.bar()
plt.ylabel('MONTHLY_AVG_PURCHASE')
plt.show()
```



Insights : Customers with one off and installments do most monthly purchase

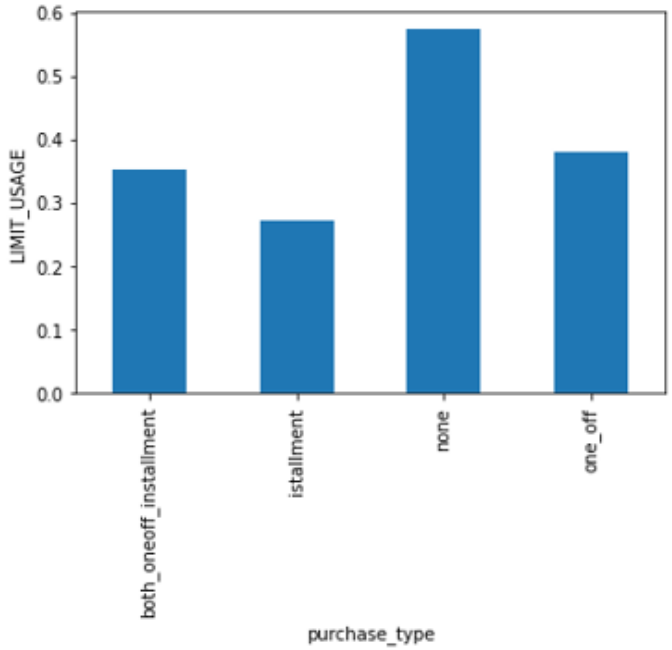
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['Monthly_cash_advance'])).plot.bar()
plt.ylabel('MONTHLY_AVG_CASH_ADVANCE')
plt.show()
```



Insights : Customers with no one off and installments take more monthly cash advance

- Customers who didn't do either of oneoff or installment purchases take highest monthly cash advance

```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['limit_usage'])).plot.bar()
plt.ylabel('LIMIT_USAGE')
plt.show()
```



Insights : Customers with no one off and installments have highest limit usage

Now, I am going to perform log transformation to reduce outlier effect and make the dataset normally distributed.

Python Code:

Extreme value Treatment

- Since there are variables having extreme values so I am doing log-transformation on the dataset to remove outlier effect and make the data normally distributed

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

R Code:

```
cc2_log = (log(cc2_subset + 1 ))
################################################
```

Summary of data set after log transformation.

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

ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	...	PURCHASES_TRX	CREDIT_LIMIT
8950.000000	8950.000000	8950.000000	...	8950.000000	8950.000000
0.158699	0.270072	0.113512	...	1.894731	8.094806
0.216672	0.281852	0.156716	...	1.373856	0.819633
0.000000	0.000000	0.000000	...	0.000000	3.931826
0.000000	0.000000	0.000000	...	0.693147	7.378384
0.080042	0.154151	0.000000	...	2.079442	8.006701
0.262364	0.559616	0.200671	...	2.890372	8.779711
0.693147	0.693147	0.916291	...	5.883322	10.308986

PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE	Monthly_avg_purchase	Monthly_cash_advance	limit_usage	payment_minpayment
8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000
6.624540	5.896738	0.117730	2.519680	3.050877	2.163970	0.296081	1.361780
1.591763	1.200316	0.211617	0.130367	2.002823	2.429741	0.250303	0.938991
0.000000	0.018982	0.000000	1.945910	0.000000	0.000000	0.000000	0.000000
5.951361	5.119191	0.000000	2.564949	1.481458	0.000000	0.040656	0.651757
6.754489	5.710413	0.000000	2.564949	3.494587	0.000000	0.264455	1.115398
7.550732	6.691357	0.133531	2.564949	4.587295	4.606022	0.540911	1.958091
10.834125	11.243832	0.693147	2.564949	8.315721	8.276166	2.827902	8.830767

3.2.6. Feature selection

• What is Feature Selection??

– Selecting a subset of relevant features (variables, predictors) for use in model construction

– subset of a learning algorithm's input variables upon which it should focus attention, while ignoring the rest

Correlation Analysis

• Correlation tells you the association between two continuous variables

• Ranges from -1 to 1

• Measures the direction and strength of the linear relationship between two quantitative variables

• Represented by "r"

• Correlation can be calculated as

$$r = \frac{Covariance\ (x,y)}{S.D.(x)S.D.(y)}$$

- Covariance

$$Cov(X,Y) = \frac{\sum(X_i - \bar{X}) * (Y_i - \bar{Y})}{n}$$

Deleting the features from dataset which are used in deriving KPI's

Python Code:

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

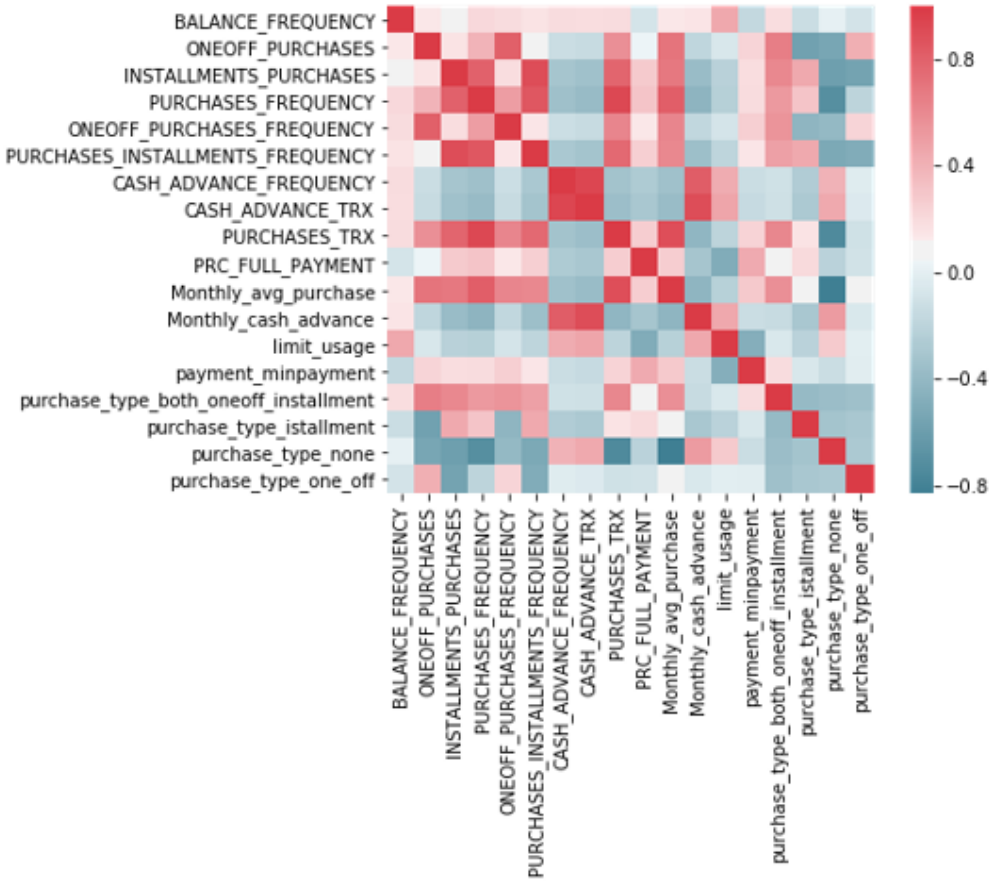
R Code:

```
############ Deleting the features used in deriving KPI's ###################
cc2_subset = subset(cc2, select = -c(BALANCE,PURCHASES,CASH_ADVANCE,TENURE,
                                     PAYMENTS,MINIMUM_PAYMENTS,CREDIT_LIMIT))
############################################################
```

3.2.6.1. Correlation Plot

Python Code:

```
##Correlation analysis
#Correlation plot
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(7, 5))

#Generate correlation matrix
credit_new_corr=cr_dummy.corr()

#Plot using seaborn library
sns.heatmap(credit_new_corr, mask=np.zeros_like(credit_new_corr, dtype=np.bool), cmap=sns.diverging_palette(220, 10, as_cmap=True
            square=True, ax=ax)
```



R Code:

```
corrm <- cor(cc2_log)

corrgram( cc2_log, order = F,
          upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")
```

From the above correlation plot we can see that

- One off purchase frequency is highly positively corelated to One off purchase.

- Installments purchase frequency is highly positively corelated to Installments purchases.
- Cash advance frequency is highly positively corelated to Monthly Cash advance and Cash advance TRX.
- Purchase frequency is highly positively corelated to Purchase TRX and monthly average purchase.

So, neglecting 'ONEOFF_PURCHASES_FREQUENCY','PURCHASES_INSTALLMENTS_FREQUENCY','CASH_ADVANCE_FREQUENCY','PURCHASES_FREQUENCY','PURCHASES_TRX','CASH_ADVANCE_TRX' features in further analysis:

Python Code:

```
col=['BALANCE_FREQUENCY','ONEOFF_PURCHASES_FREQUENCY','PURCHASES_INSTALLMENTS_FREQUENCY','CASH_ADVANCE_FREQUENCY','PURCHASES_FREC
cr_dummy=cr_dummy[[x for x in cr_dummy.columns if x not in col ]]
```

R code:

```
#########################
cr_dummy=subset(cc2_log, select = -c(ONEOFF_PURCHASES_FREQUENCY,PURCHASES_INSTALLMENTS_FREQUENCY,
                CASH_ADVANCE_FREQUENCY,PURCHASES_FREQUENCY,PURCHASES_TRX,CASH_ADVANCE_TRX))
"""""""""
```

Shape of data

Python code:

```
cr_dummy=pd.DataFrame(cr_dummy)
cr_dummy.shape
```

```
(8950, 12)
```

R Code:

```
> dim(cr_dummy)
[1] 8950   12
```

3.2.6.2. Standardize the Data

Our data may be at different levels or may contain different units. It will not be suitable to move ahead and use this data without solving this problem. This can be done by standardizing the data.

- Calculate the mean absolute deviation:
- Calculate the standardized measurement (z-score)
- Using mean absolute deviation is more robust than using standard deviation. Since the deviations are not squared the effect of outliers is somewhat reduced but their z-scores do not become too small; therefore, the outliers remain detectable.

The idea behind StandardScaler is that it will transform the data such that its distribution will have a mean value 0 and standard deviation of 1.

Python Code:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
cr_scaled=sc.fit_transform(cr_dummy)
cnames=cr_dummy.columns
```

R Code:

```
# cr_dummy contains all the variables that will be used for clustering
Scaled_data<- data.frame(scale(cr_dummy)) # standardizing data
```

Principal Component Analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components.

If there are n observations with p variables, then the number of distinct principal components is. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.

The resulting vectors (each being a linear combination of the variables and containing n observations) are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It is often used to visualize genetic distance and relatedness between populations. PCA can be done by eigenvalue decomposition of a data covariance (or correlation) matrix or singular value decomposition of a data matrix, usually after a normalization step of the initial data.

The normalization of each attribute consists of mean centering – subtracting each data value from its variable's measured mean so that its empirical mean (average) is zero – and, possibly, normalizing each variable's variance to make it equal to 1; see Z-scores. The results of a PCA are usually discussed in terms of component scores, sometimes called factor scores (the transformed variable values corresponding to a particular data point), and loadings (the weight by which each standardized original variable should be multiplied to get the component score).

If component scores are standardized to unit variance, loadings must contain the data variance in them (and that is the magnitude of eigenvalues). If component scores are not standardized (therefore they contain the data variance) then loadings must be unit-scaled, ("normalized") and these weights are called eigenvectors; they are the cosines of orthogonal rotation of variables into principal components or back.

Python Code:

With the help of principal component analysis we will reduce features

```
from sklearn.decomposition import PCA
```

```
cr_scaled_df.shape
```

```
(8950, 12)
```

```
#We have 17 features so our n_component will be 132
pc=PCA(n_components=12)
cr_pca=pc.fit(cr_scaled)
```

```
#Lets check if we will take 12 component then how much varience it explain. Ideally it should be 1 i.e 100%
sum(cr_pca.explained_variance_ratio_)
```

```
1.0
```

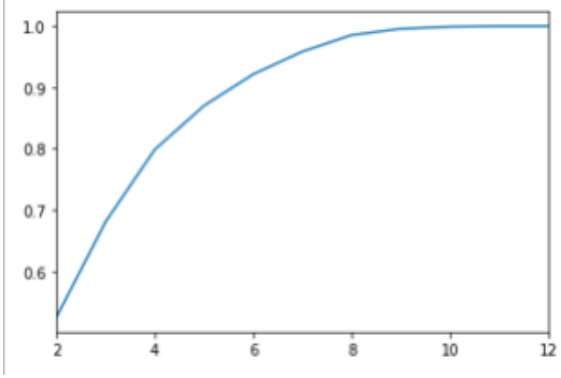
```
var_ratio={}
for n in range(2,13):
    pc=PCA(n_components=n)
    cr_pca=pc.fit(cr_scaled)
    var_ratio[n]=sum(cr_pca.explained_variance_ratio_)
```

```
var_ratio
```

```
{2: 0.5249204020778663,
 3: 0.6807419836429282,
 4: 0.7986318827690105,
 5: 0.8700582258439946,
 6: 0.9215094915232396,
 7: 0.958487223577202,
 8: 0.9854385343270677,
 9: 0.9956538883029205,
 10: 0.9991707003348477,
 11: 1.0,
 12: 1.0}
```

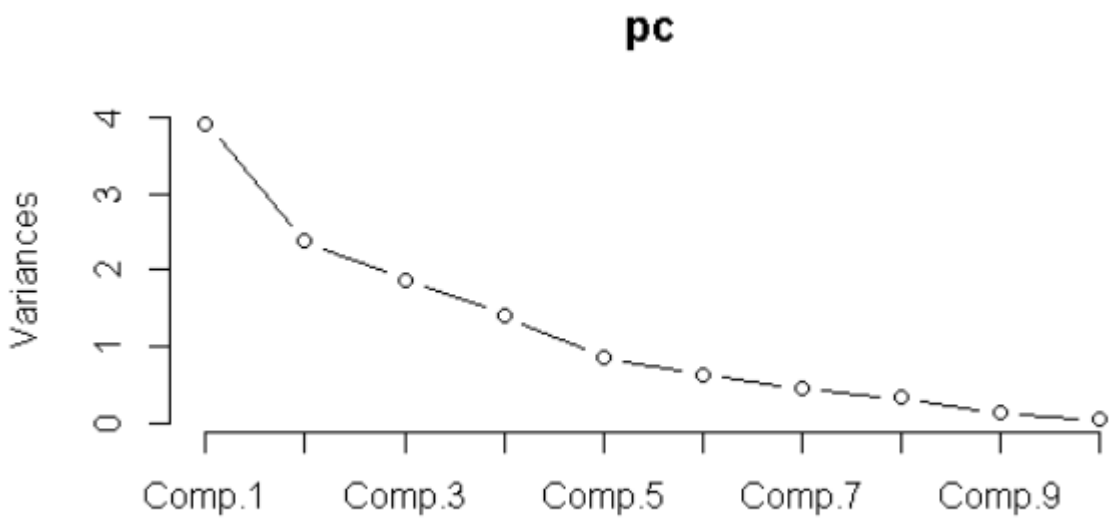
```
pd.Series(var_ratio).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x246d34e2308>
```



R Code:

```
> pc <- princomp(Scaled_data,cor="False")
> plot(pc)
> screeplot(pc, type='lines')
> Scaled_data_pca <- prcomp(Scaled_data)
```




```
> summary(Scaled_data_pca)
Importance of components:
                          PC1    PC2    PC3    PC4     PC5     PC6
Standard deviation     1.9806 1.5425 1.3688 1.1882 0.92049 0.78884
Proportion of Variance 0.3269 0.1983 0.1561 0.1177 0.07061 0.05186
Cumulative Proportion  0.3269 0.5252 0.6813 0.7990 0.86959 0.92145
                           PC7     PC8     PC9    PC10    PC11    PC12
Standard deviation     0.66511 0.57081 0.34975 0.20541 0.09976 1.8e-14
Proportion of Variance 0.03686 0.02715 0.01019 0.00352 0.00083 0.0e+00
Cumulative Proportion  0.95831 0.98546 0.99565 0.99917 1.00000 1.0e+00
> |
```

From the above graph and summary, we can see that 5 principal components can explain about 87 % of total variance of data.

So, we are going to use first five PC's to build the cluster.

3.2.6.3. Dimensionality Reduction

Such dimensionality reduction can be a very useful step for visualizing and processing high dimensional datasets, while still retaining as much of the variance in the dataset as possible. For example, selecting L = 2 and keeping only the first two principal components finds the two-dimensional plane through the high-dimensional dataset in which the data is most spread out, so if the data contains clusters these too may be most spread out, and therefore most visible to be plotted out in a two-dimensional diagram; whereas if two directions through the data (or two of the original variables) are chosen at random, the clusters may be much less spread apart from each other, and may in fact be much more likely to substantially overlay each other, making them indistinguishable.

Python Code:

```
pc_final=PCA(n_components=5).fit(cr_scaled)

reduced_cr=pc_final.fit_transform(cr_scaled)
```

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

```
(8950, 5)
```

R Code:

```
> Credit_PCs <- data.frame(Scaled_data_pca$x[,1:5])
> dim(Credit_PCs)
[1] 8950    5
```

Factor analysis

Principal component analysis creates variables that are linear combinations of the original variables. The new variables have the property that the variables are all orthogonal. The PCA transformation can be helpful as a pre-processing step before clustering. PCA is a variancefocused approach seeking to reproduce the total variable variance, in which components reflect both common and unique variance of the variable. PCA is generally preferred for purposes of data reduction (i.e., translating variable space into optimal factor space) but not when the goal is to detect the latent construct or factors.

Factor analysis is similar to principal component analysis, in that factor analysis also involves linear combinations of variables. Different from PCA, factor analysis is a correlation-focused approach seeking to reproduce the inter-correlations among variables, in which the factors "represent the common variance of variables, excluding unique variance". In terms of the correlation matrix, this corresponds with focusing on explaining the off-diagonal terms (i.e. shared co-variance), while PCA focuses on explaining the terms that sit on the diagonal. However, as a side result, when trying to reproduce the on-diagonal terms, PCA also tends to fit relatively well the off-diagonal correlations. Results given by PCA and factor analysis are very similar in most situations, but this is not always the case, and there are some problems where the results are significantly different. Factor analysis is generally used when the research purpose is detecting data structure (i.e., latent constructs or factors) or causal modeling.

Loadings

Definition: Component loadings are the ordinary product moment correlation between each original variable and each component score.

Interpretation: By looking at of component loadings one can ascertain which of the original variables tend to “load” on a given new variable. This may facilitate interpretations, creation of subscales, etc. Loadings=Eigenvectors * sqrt(Eigenvalues) Loadings are the covariance/correlations between the original variables and the unit-scaled components

Python Code:

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

	PC_0	PC_1	PC_2	PC_3	PC_4
BALANCE_FREQUENCY	-0.006282	-0.284032	-0.377648	-0.088859	-0.694155
ONEOFF_PURCHASES	0.309332	-0.476156	0.181170	0.026412	0.054112
INSTALLMENTS_PURCHASES	0.387217	0.143763	-0.417634	0.000190	0.090263
PRC_FULL_PAYMENT	0.236404	0.265579	0.144951	0.278382	-0.530399
Monthly_avg_purchase	0.456563	-0.150050	-0.038773	-0.160523	-0.073336
Monthly_cash_advance	-0.306719	-0.195282	-0.128531	0.238935	-0.113457
limit_usage	-0.247192	-0.323452	-0.341081	-0.211458	-0.135023
payment_minpayment	0.223295	0.092727	0.247059	0.493705	-0.250473
purchase_type_both_oneoff_installment	0.324530	-0.300247	-0.269057	0.308467	0.280363
purchase_type_istallment	0.077792	0.522579	-0.191329	-0.371258	-0.113905
purchase_type_none	-0.416012	0.033550	-0.055303	0.409581	-0.010045
purchase_type_one_off	-0.022871	-0.251335	0.567165	-0.376633	-0.186689

R Code:

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
BALANCE_FREQUENCY	0.008516501	0.28776628	0.37831172	0.098968163	0.69168864
ONEOFF_PURCHASES	-0.308405632	0.47551297	-0.18559271	-0.030393633	-0.04881261
INSTALLMENTS_PURCHASES	-0.387429710	-0.13897375	0.41862598	-0.004154726	-0.09420587
PRC_FULL_PAYMENT	-0.236308606	-0.26596672	-0.14011725	-0.266158002	0.55945766
monthly_avg_purchase	-0.456247348	0.15259472	0.03849811	0.159194456	0.06790802
monthly_cash_advance	0.307341703	0.19340505	0.12441350	-0.240925369	0.11001725
limit_usage	0.248103700	0.32544085	0.33743746	0.207437743	0.12052737
purchase_type_none	0.416156436	-0.03653071	0.05330203	-0.409054135	0.02284353
purchase_type_one_off	0.023146565	0.24829649	-0.56679949	0.381653519	0.17926577
purchase_type_installment	-0.078925329	-0.51889483	0.19824797	0.375911083	0.09582688
purchase_type_both	-0.323838591	0.30216354	0.26405182	-0.317733930	-0.26846358
payment_minpayment	-0.223318840	-0.10432305	-0.25219393	-0.486576997	0.22838752

4. Determining The Optimal Number Of Clusters

4.1. Elbow method

The basic idea behind partitioning methods, such as k-means clustering, is to define clusters such that the total intra-cluster variation [or total within-cluster sum of square (WSS)] is minimized. The total WSS measures the compactness of the clustering and we want it to be as small as possible.

The Elbow method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't improve much better the total WSS.

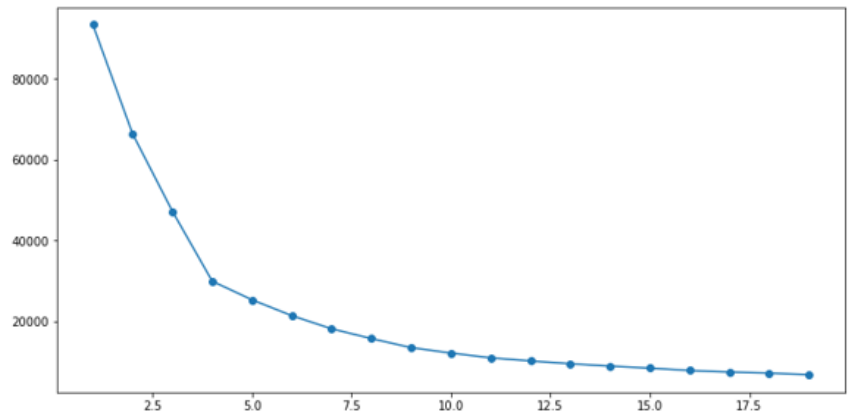
The optimal number of clusters can be defined as follow:

1. Compute clustering algorithm (e.g., k-means clustering) for different values of k. For instance, by varying k from 1 to 10 clusters.
2. For each k, calculate the total within-cluster sum of square (wss).
3. Plot the curve of wss according to the number of clusters k
4. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

Python Code:

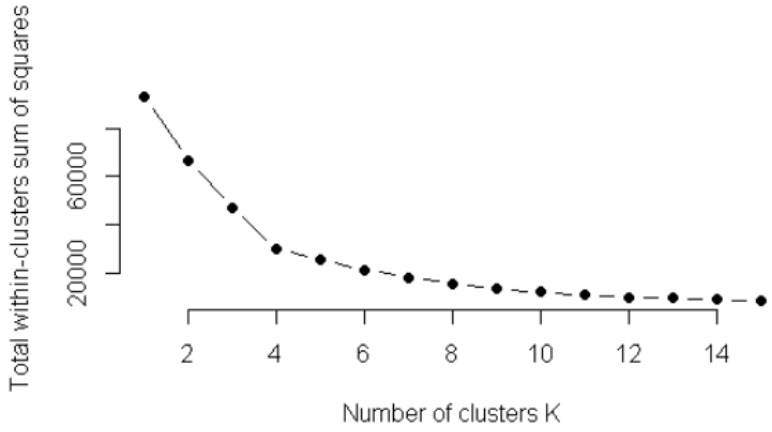
```
#Load required libraries
from sklearn.cluster import KMeans

#Estimate optimum number of clusters
cluster_range = range( 1, 20 )
cluster_errors = []

for num_clusters in cluster_range:
    clusters = KMeans(num_clusters).fit(reduced_cr)
    cluster_errors.append(clusters.inertia_)

#Create dataframe with cluster errors
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_errors": cluster_errors } )
```

```
#Plot line chart to visualise number of clusters
%matplotlib inline
plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker = "o" )
```



R Code:

```
#Elbow Method for finding the optimal number of clusters
set.seed(123)
# Compute and plot wss for k = 2 to k = 15.
k.max <- 15
data <- Credit_PCs
wss <- sapply(1:k.max,
              function(k){kmeans(data, k, nstart=50,iter.max = 100 )$tot.withinss})
wss
plot(1:k.max, wss,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")
```



The total within-cluster sum of square (wss) versus number ok cluster plot shows elbow for number of cluster equals to 4.

4.2. silhouette method

Another visualization that can help determine the optimal number of clusters is called silhouette method. Average silhouette method computes the average silhouette of observations for different values of k. The optimal number of clusters k is the one that maximize the average silhouette over a range of possible values for k. The optimal number of clusters k is the one that maximize the average silhouette over a range of possible values for k .

The algorithm is similar to the elbow method and can be computed as follow:

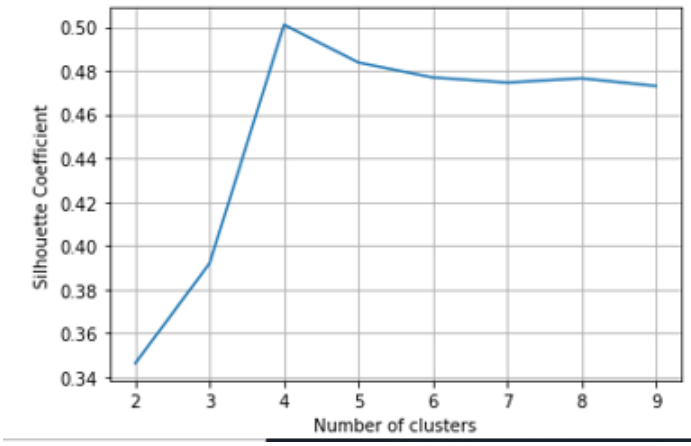
1. Compute clustering algorithm (e.g., k-means clustering) for different values of k. For instance, by varying k from 1 to 10 clusters.
2. For each k, calculate the average silhouette of observations
3. Plot the curve of *avg.sil* according to the number of clusters k.
4. The location of the maximum is considered as the appropriate number of clusters.

Python Code:

```
from sklearn import metrics
```

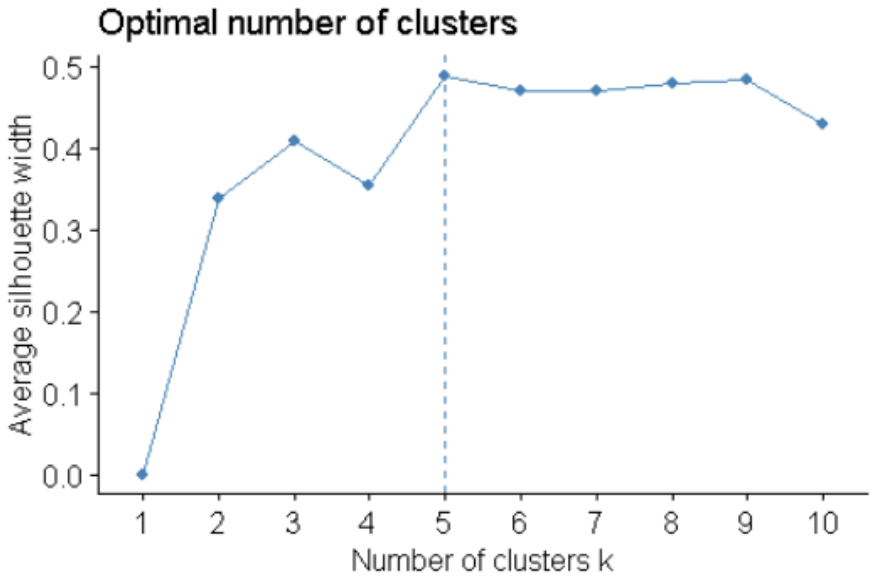
```
# calculate SC for K=2 to K=10
k_range = range(2, 10)
scores = []
for k in k_range:
    km = KMeans(n_clusters=k, random_state=100)
    km.fit(reduced_cr)
    scores.append(metrics.silhouette_score(reduced_cr, km.labels_))
```

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



R Code:

```
# Silhouette method
fviz_nbclust(Credit_PCs, kmeans, method = "silhouette", k.max = 10)
```



Average silhouette method shows highest average silhouette width for number of cluster equals to 4 in python code but 5 in R code for the same dataset.

But the total within-cluster sum of square (wss) versus number of cluster plot shows elbow for number of cluster equals to 4 in both python and R.

Based on elbow method and Average silhouette method optimal number of cluster for our dataset is 4.

So, we are going to build four cluster using K means algorithm in both R and Python.

5. Clustering

Clustering is the process of dividing the entire data into groups (also known as clusters) based on the patterns in the data.

Clustering of data is a method by which large sets of data are grouped into clusters of smaller sets of similar data.

Cluster: a collection of data objects

- Similar to one another within the same cluster
- Dissimilar to the objects in other clusters Clustering is unsupervised classification: no predefined classes

Definition: Given a set of data points, each having a set of attributes, and a similarity measure among them, find clusters such that:

- data points in one cluster are more similar to one another (high intra-class similarity)
- data points in separate clusters are less similar to one another (low inter-class similarity) Similarity measures:

e.g. Euclidean distance if attributes are continuous.

Examples of Clustering Applications:

Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs

Market Segmentation: Grouping people (with willingness, purchasing power, and Authority to buy etc...) according to their similarity in several dimensions related to product under consideration

Sales Segmentation: Clustering can help you to identify what type of customers can buy what type of products

Credit Risk: Segmentation customers based on their credit history ν Operations: High Performer segmentation & promotion based on person's performance

Land use: Identification of areas of similar land use in an earth observation database

Insurance: Identifying groups of motor insurance policy holders with a high average claim cost

City-planning: Identifying groups of houses according to their house type, value, and geographical location

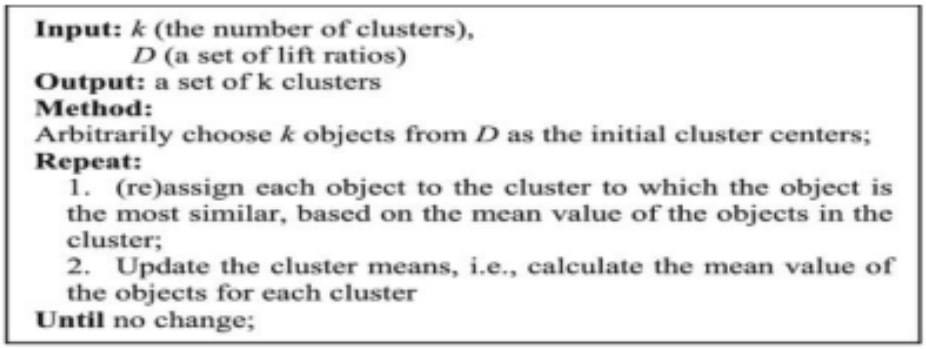
Earth-quake studies: Observed earth quake epicenters should be clustered along continent faults 3.10 K-Means Clust

Working

To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids

It halts creating and optimizing clusters when either:

- The centroids have stabilized — there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.



Python Code:

```
# It seems that the optimal number of clusters is 4.
# I am going to take 4 for the analysis
km_4=KMeans(n_clusters=4,random_state=123)
```

```
# applying kmeans
km_4.fit(reduced_cr)
```

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

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

```
3    2774
0    2259
1    2043
2    1874
dtype: int64
```

```
# Conactenating labels found through Kmeans with data
cluster_df_4=pd.concat([cre_original[col_kpi],pd.Series(km_4.labels_,name='Cluster_4')],axis=1)
```

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

R Code:

```
#building clusters using k-means clustering
CreditCluster <- kmeans(Credit_PCs, 4, nstart = 20)
```

```
# Conactenating labels found through Kmeans with dataset as column 'Cluster'
cc2_subset$cluster <- as.factor(CreditCluster$cluster)
```

```
as.data.frame(table(CreditCluster$cluster))
Var1 Freq
   1 2043
   2 2259
   3 2774
   4 1874
```

Extracting mean of all the feature corresponding to it cluster to make the interpretation simple.

Python Code:

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

Cluster_4	0	1	2	3
PURCHASES_TRX	11.904825	0.002937	7.109925	32.959625
ONEOFF_PURCHASES	0.000000	0.000000	786.827679	1379.884427
INSTALLMENTS_PURCHASES	538.114608	0.002173	0.000000	888.049776
Monthly_avg_purchase	46.994978	0.000181	69.688958	192.685172
Monthly_cash_advance	37.682472	183.578865	78.995966	67.821985
limit_usage	0.271618	0.573691	0.381074	0.353548
CASH_ADVANCE_TRX	1.261178	6.302007	2.932231	2.832733
payment_minpayment	13.300288	10.100468	5.542470	7.240871
both_oneoff_installment	0.000000	0.000000	0.000000	1.000000
istallment	1.000000	0.000489	0.000000	0.000000
one_off	0.000000	0.000000	1.000000	0.000000
none	0.000000	0.999511	0.000000	0.000000
CREDIT_LIMIT	3366.052848	4031.863283	4515.920572	5738.829463

R code:

```
# Mean value gives a good indication of the distribution of data.
# So we are finding mean value for each variable for each cluster

cluster_4<-as.data.frame(aggregate( .~ cluster, FUN=mean, data=cre_original))
# transpose
cluster_4 <- t(cluster_4)
```

	V1	V2	V3	V4
cluster	1	2	3	4
PURCHASES_TRX	11.904825144	32.959625090	0.002936858	7.109925293
ONEOFF_PURCHASES	0.0000	1379.8844	0.0000	786.8277
INSTALLMENTS_PURCHASES	5.381146e+02	8.880498e+02	2.173275e-03	0.000000e+00
monthly_avg_purchase	4.699498e+01	1.926852e+02	1.811062e-04	6.968896e+01
monthly_cash_advance	37.68247	67.82199	183.57887	78.99597
limit_usage	0.2716175	0.3535485	0.5737005	0.3810738
CASH_ADVANCE_TRX	1.261178	2.832733	6.302007	2.932231
payment_minpayment	13.494579	7.288366	10.122106	5.551561
purchase_type_both	0	1	0	0
purchase_type_installment	1.0000000000	0.0000000000	0.0004894763	0.0000000000
purchase_type_none	0.0000000	0.0000000	0.9995105	0.0000000
purchase_type_one_off	0	0	0	1
CREDIT_LIMIT	3366.053	5738.829	4030.954	4515.921

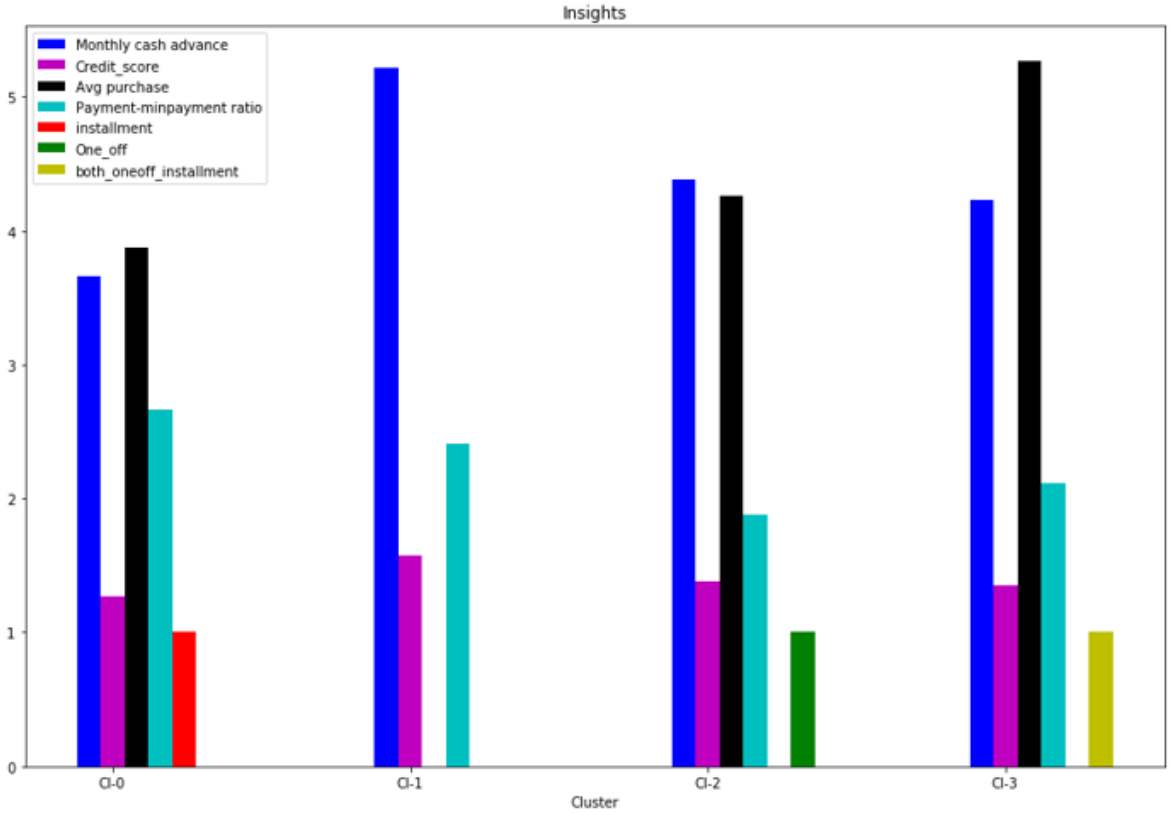
Here in output of both python and R code we can see that customers in each cluster of both the code is same and mean value in both the output is exactly same.

Only the cluster numbering is different, but the Insights is same.

Python	R
Cluster 0	Cluster 1
Cluster 1	Cluster 3
Cluster 2	Cluster 4
Cluster 3	Cluster 2

6. Insights

- We have 12 features in the dataset. It will be difficult to interpret the results if we consider all the feature. So, to get insights of the data we are selecting few most important variables,



Insights with 4 Clusters

- Cluster 0 customers are doing maximum Installments transactions, take least monthly cash advance, poor credit score and highest payment minpayment ratio. This group is about 25% of the total customer base.

- cluster 1 is taking maximum advance cash and is paying comparatively less minimum payment and good credit score & doing no purchase transaction. This group is about 23% of the total customer base.

- Cluster 2 is the group of customers who are doing maximum oneoff transactions, have comparatively good credit score and lowest payment minpayment ratio. This group is about 21% of the total customer base

- Cluster 3 customers who are doing both one off and Installment transactions have good credit score and do maximum monthly purchase. This group is about 31% of the total customer base.

7. Market Strategy Suggestion

Group 0

They are potential target customers who are paying dues and doing mostly installment purchases and have poor credit score)

- we can increase credit limit or can lower down interest rate
- Can be given premium card /loyality cards to increase transaction.

Group 1

They have good credit score and taking only cash on advance.

- We can target them by providing less interest rate on Installment purchase transaction
- cashback/discount on one off purchase transaction.

Group 2

This group is has minimum paying ratio and using card for just oneoff transactions (may be for utility bills only).

- We can target them by providing less interest rate on Installment purchase transaction.

Group 3

This group is performing best among all as customers are maintaining good credit score, doing both one off and instalment purchase, highest monthly average purchase and paying dues on time.

- Giving rewards point will make them perform more purchases.

8. References:

1. https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

2. https://www.datacamp.com/community/tutorials/principal-component-analysis-in-python

3. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

4. https://www.sciencedirect.com/topics/computer-science/log-transformation