CREDIT CARD SEGMENTATION

Submitted by: C.Adimurthy

Saiadithya689@gmail.com

Contents

1. Pre-requisite and Steps to Execute Code

A. To run the R project in the system , you should have R &R-studio; Below are the versions of both which are used:

1. R version 3.6.1(64 bit)
2. R-studio IDE (Version 1.2.5019).

B. To run the Python(ipynb) file in the system , you should have Anaconda Navigator to be installed .Below are the versions of software used:

1. Anaconda Navigator(2019.10.0.0)
2. Jupyter Notebook(6.0.2)
3. Python version(3.7.4)

C. Download the file and save it in your system. Following are the files:-

- Credit-card-data.csv
- CreditCardProject.ipynb
- CreditCardProject.R
- Credit Card Segmentation in .docx format
- Folder with r-plots

Steps to run the R code:

1. Run the application by following the below steps using Command Prompt:
    - Go to the path where you have installed R, select the path of the file at the top of the window then type “cmd” in that search bar and press enter.
    - You will be directed to command prompt with R default location from where you executed the command.
    - Next, type the R-Script file name path complete .

- 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

Steps to run the Python code:

1. For Python, please open Anaconda prompt type Jupyter Notebook and load ipynb file and run the code.
2. For complete project, open .docx document named “Credit Card Segmentation”.

2. Introduction

2.1 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.

2.2 Data

Understanding of data is the very first and important step in the process of finding solution of any business problem. Here in our case our company has provided a data set with following features, we need to go through each and every variable of it to understand and for better functioning.

Size of Dataset Provided: - 8950 rows, 18 Columns

3. Process and Requirement Specification

3.1 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

3.2 Software Requirements

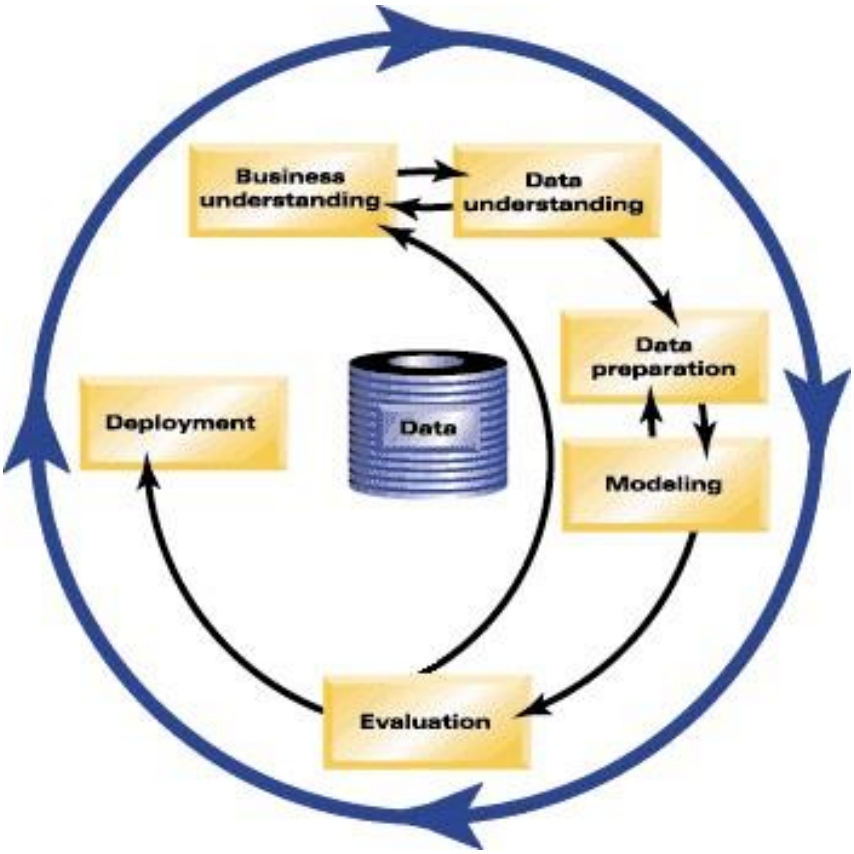
1. Python
2. Jupyter Notebook
3. R
4. R Studio

4. 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



4.1 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 (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

4.2. 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.

4.2.1 Data Exploration and Cleaning

In Data Pre-processing , 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

PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX
8950.000000	8950.000000	8950.000000	8950.000000	8950.00000(
0.490351	0.202458	0.364437	0.135144	3.24882
0.401371	0.298336	0.397448	0.200121	6.82464
0.000000	0.000000	0.000000	0.000000	0.00000(
0.083333	0.000000	0.000000	0.000000	0.00000(
0.500000	0.083333	0.166667	0.000000	0.00000(
0.916667	0.300000	0.750000	0.222222	4.00000(
1.000000	1.000000	1.000000	1.500000	123.00000(

CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
8950.000000	8950.000000	8950.000000	8949.000000	8950.000000	8637.000000	8950.000000	8950.000000
0.135144	3.248827	14.709832	4494.449450	1733.143852	864.206542	0.153715	11.517318
0.200121	6.824647	24.857649	3638.815725	2895.063757	2372.446607	0.292499	1.338331
0.000000	0.000000	0.000000	50.000000	0.000000	0.019163	0.000000	6.000000
0.000000	0.000000	1.000000	1600.000000	383.276166	169.123707	0.000000	12.000000
0.000000	0.000000	7.000000	3000.000000	856.901546	312.343947	0.000000	12.000000
0.222222	4.000000	17.000000	6500.000000	1901.134317	825.485459	0.142857	12.000000
1.500000	123.000000	358.000000	30000.000000	50721.483360	76406.207520	1.000000	12.000000

R code:

```
> ############ Find the range of frequency variable in dataset #############
> 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 freq >1 with 1 ############
credit$CASH_ADVANCE_FREQUENCY[credit$CASH_ADVANCE_FREQUENCY > 1] = 1
```

4.2.2 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:

```
# 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

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),]
> row.names(missing_val) = NULL
> missing_val = missing_val[,c(2,1)]
> View(missing_val)
>
```

	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

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 500th 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 = 755.94
# Median = 873.09
```

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

I didn’t choose KNN imputation as it is only supported by Python 3.6.

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

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

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:

```
#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)
```

4.2.3 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.

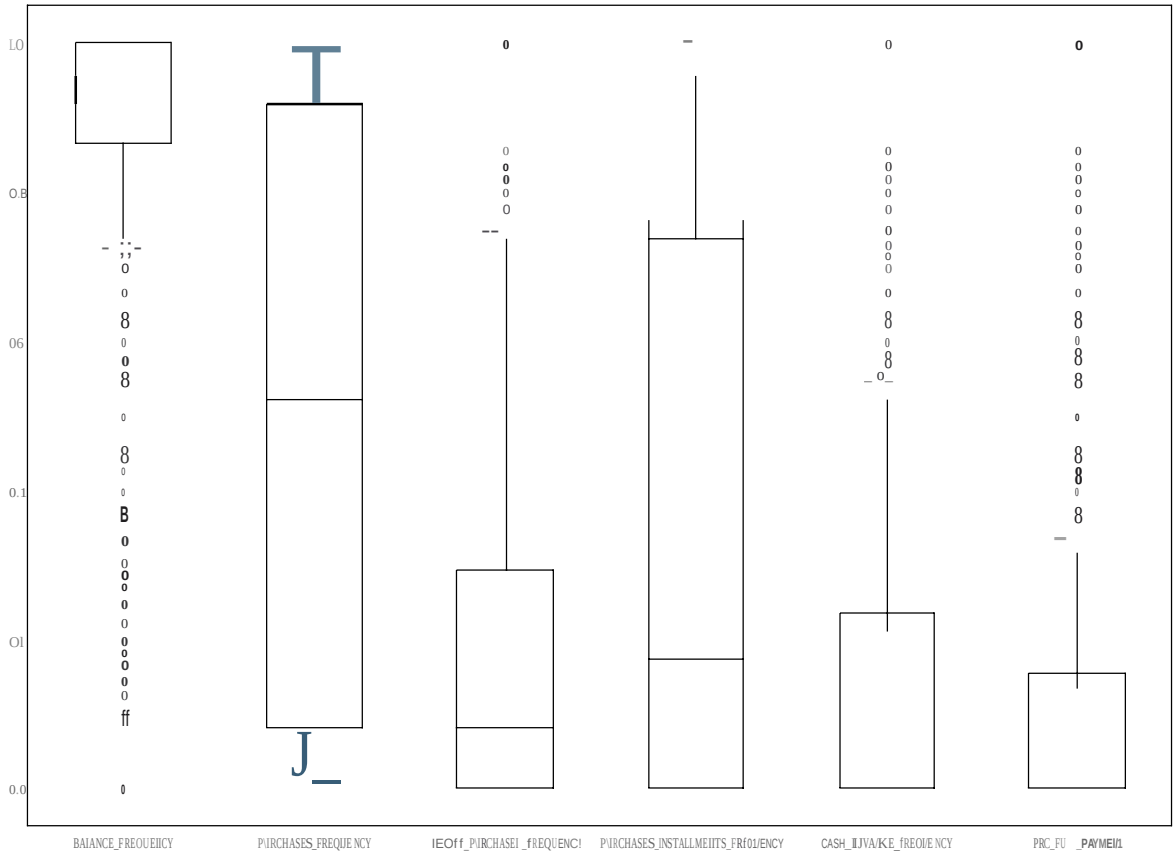
For Outlier analysis, Boxplot to visualize and summary descriptive statistics to check range of each numeric variable and sorted the variables to detect some strange values like zeros and negative values.

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

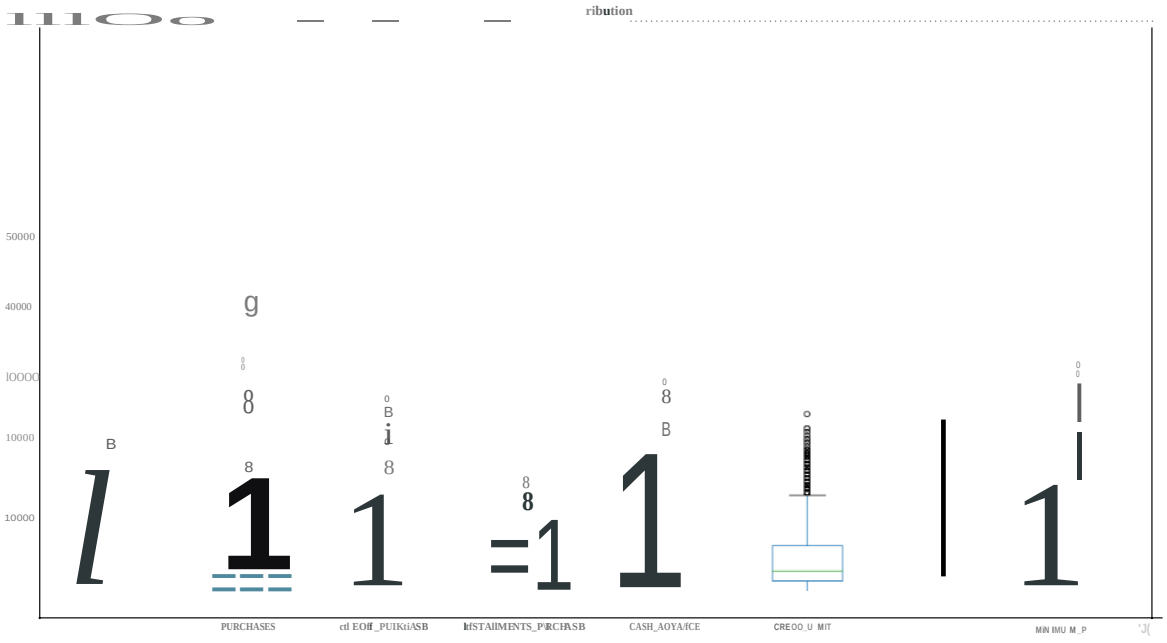
```
4]: #Let´s see how are distributed the frequency variables

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()
```



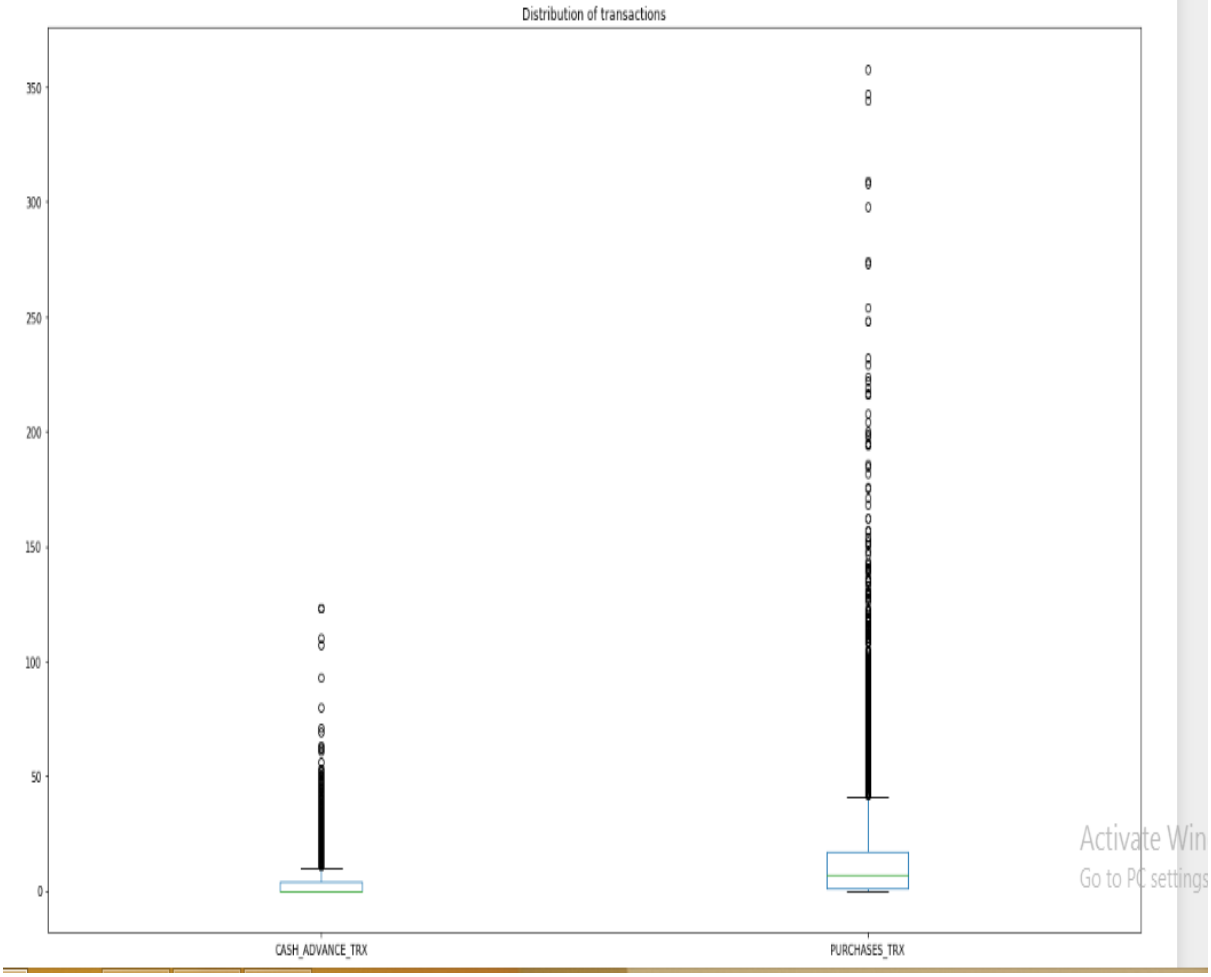
#Let·s see ho1 are distributed the numeri.c variables

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

There are also many outliers, but 1He 1i1.U keep them for noi

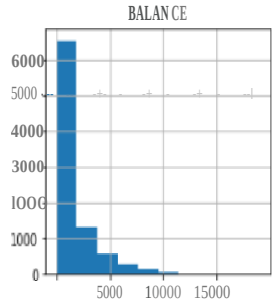

ribution
50000
40000
10000
10000
PURCHASES
CASH_AOYA/ICE
CREOO_U_MIT
MIN IMU M_P

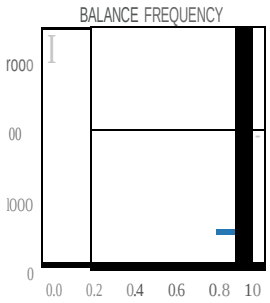
```
#Let´s see how are distributed the numeric variables

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

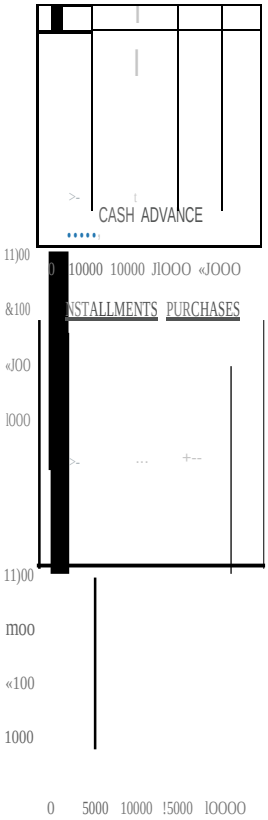


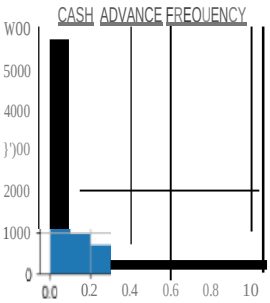
From above box plot we can see that we have so many outliers in the dataset.

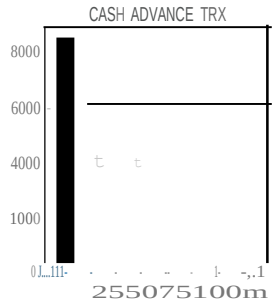
```
# Exploratory Data Analysis
credit.hist(figsize=(18,18));
```

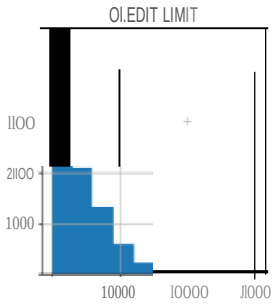


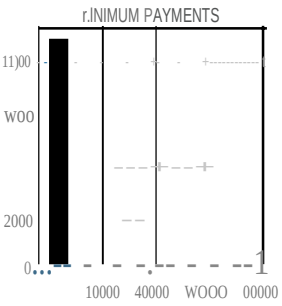


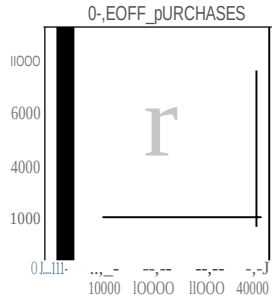


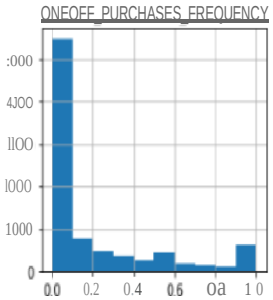


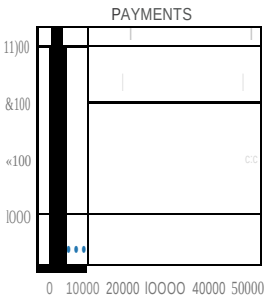


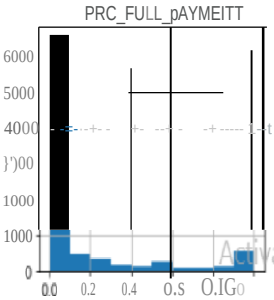


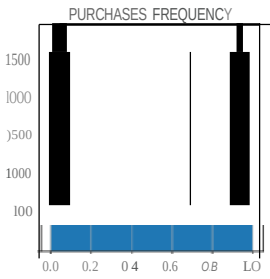


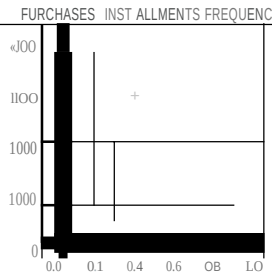


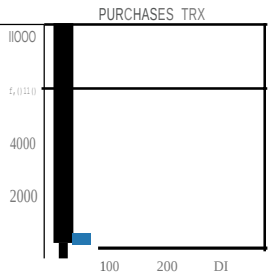


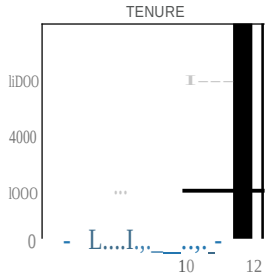












By looking above histogram plot we can see that the data in most of the features is skewed.

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 many 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.

4.2.4 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:

1. Monthly average purchase and cash advance amount

Mo11U1y_avg_1Purchase

[27]: credit ['Mont hl y_avg_pur chase ']=credit ['PLJRCHASES']/ credi t['TEMLJR E']

Mo11 hly_cash_adva nce Amoun t

[28]: credi t ['Mont hl y_cash_advan ce']=credit ['CASH_ADVANCE '] / credi t[' THILJR E ']

```
def purchas e(credi t ):
    if (credit [ 'ONEOFF_PURCHASES' ]==0) & (credit [ ' HISTALL ME NTS_PURCHASES' ]==0):
        return 'non e'
    if (credit [ 'ONEOFF_PURCHASES' ]>0) & (credi t [ ' INSTALLMENTS_PURCHASES' ]>0):
        return 'bot h_ oneoff_ instal lm elt '
    if (credit [ 'ONEOFF_PURCHASES' ]>0) & (credi t [ ' INSTALLMENTS_PURCHASES' ]==0):
        return 'one off'
    if (credit [ 'ONEOFF_PURCHASES' ]==0) & (credit [ ' HISTALL ME NTS_PURCHASES' ]>0):
        return 'i ns t alln ent '

cre di t[ ' pur c hase_t yp e ']=c r ed i t . appl y( p u rc has e, axi s=1 )

credi t [ ' pur chase_t ype ' ) . val ue_count s( )

bot h_ oneoff _ i nst alln ent      2 7 4
irlst allm ent                    2260
none                             2042
one off                           187 4
Name: purchase_t ype, dt ype: i nt 64
```

Limit_usage(balance to credit limit ratio) c edit card utilization

. Lower value implies cutomers are maintaing thie'r balance properly. Lower value means good credit score

```
credit ['lin it _us age']=cr edi t .appl y(lan bda x: x['BALA NCE']/x['CREDIT_W IIT'L axis=1 )
```

Payments to minimum payme nts rati o

```
#PAYfJIENT NI NPAYMENT
#The 1here clause is being used to avoi d div by zero error
ere di t [ ' payment _min payment ' ) = np.1,here(credit [ 'MINIMUM_PAYME NTS' )== 0, credi t [ ' PAYMEIHS' ],
                                                credit [ 'PAYMENTS' ]/ credi t [ 'MINIMUM_PAYME NTS' ) )
```

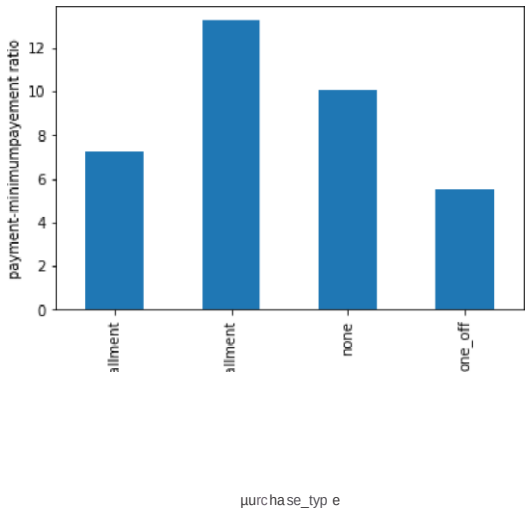
R code:

```
#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$MINIMUM_PAYMENTS
cc2$TENURE = as.numeric(cc2$TENURE)
```

4.2.5 Insights of Derived KPI’s

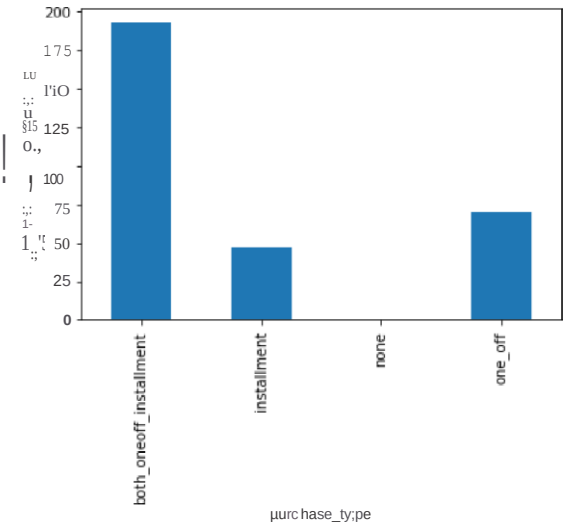
Average payment_minpayment ratio for each purchse type

```
credit.groupby('purchase_type').apply(lambda x: npmean(x['paymentminpayment'])).plot.bar()
plt.ylabel('payment-minimum payement ratio')
pit.show()
```



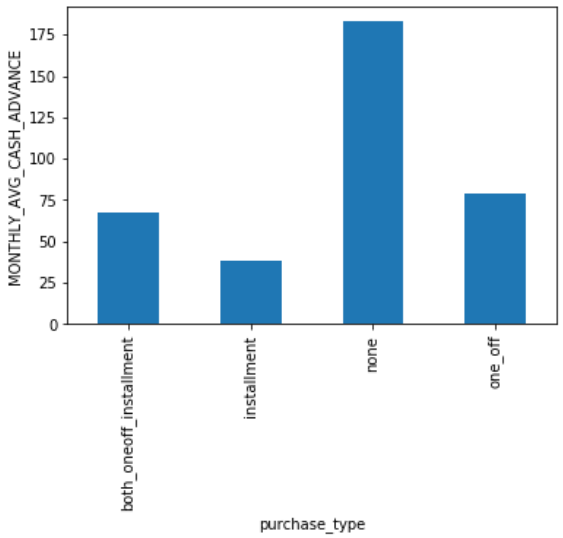
Insights : Customers with installments have highest paymen t-minimum payement ratio

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



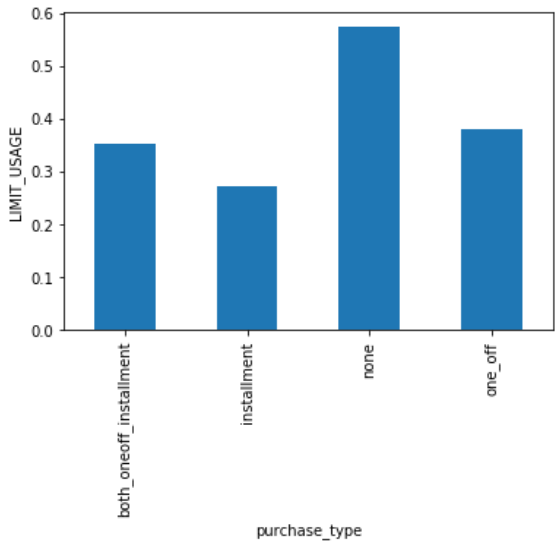
Insights : Customers with one off and installments do most monthly purchase

```
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

```
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:

```
####  Extreme value Treatment
####  Since there are variables having extreme values so I am doing
####  log-transformation on the dataset to remove outlier effect

cc2_log = (log(cc2 + 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	PAYMENT
8950.000000	8950.000000	8950.000000	...	8950.000000	8950.000000	8950.00000
0.158699	0.270072	0.113432	...	1.894731	8.094870	6.62454
0.216672	0.281852	0.156385	...	1.373856	0.819635	1.59176
0.000000	0.000000	0.000000	...	0.000000	3.931826	0.00000
0.000000	0.000000	0.000000	...	0.693147	7.378384	5.95136
0.080042	0.154151	0.000000	...	2.079442	8.006701	6.75448
0.262364	0.559616	0.200671	...	2.890372	8.779711	7.55073
0.693147	0.693147	0.693147	...	5.883322	10.308986	10.83412

LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE	Monthly_avg_purchase	Monthly_cash_advance	limit_usage	payment_minpayment
)0000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000	8950.000000
)4870	6.624540	5.951566	0.117730	2.519680	3.050877	2.163970	0.296081	1.353998
19635	1.591763	1.179646	0.211617	0.130367	2.002823	2.429741	0.250303	0.940919
)1826	0.000000	0.018982	0.000000	1.945910	0.000000	0.000000	0.000000	0.000000
'8384	5.951361	5.146667	0.000000	2.564949	1.481458	0.000000	0.040656	0.645689
)6701	6.754489	5.818892	0.000000	2.564949	3.494587	0.000000	0.264455	1.104339
'9711	7.550732	6.763023	0.133531	2.564949	4.587295	4.606022	0.540911	1.952918
)8986	10.834125	11.243832	0.693147	2.564949	8.315721	8.276166	2.827902	8.830767

4.2.6 Feature Selection

- Selection of 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

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

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 ]]
```

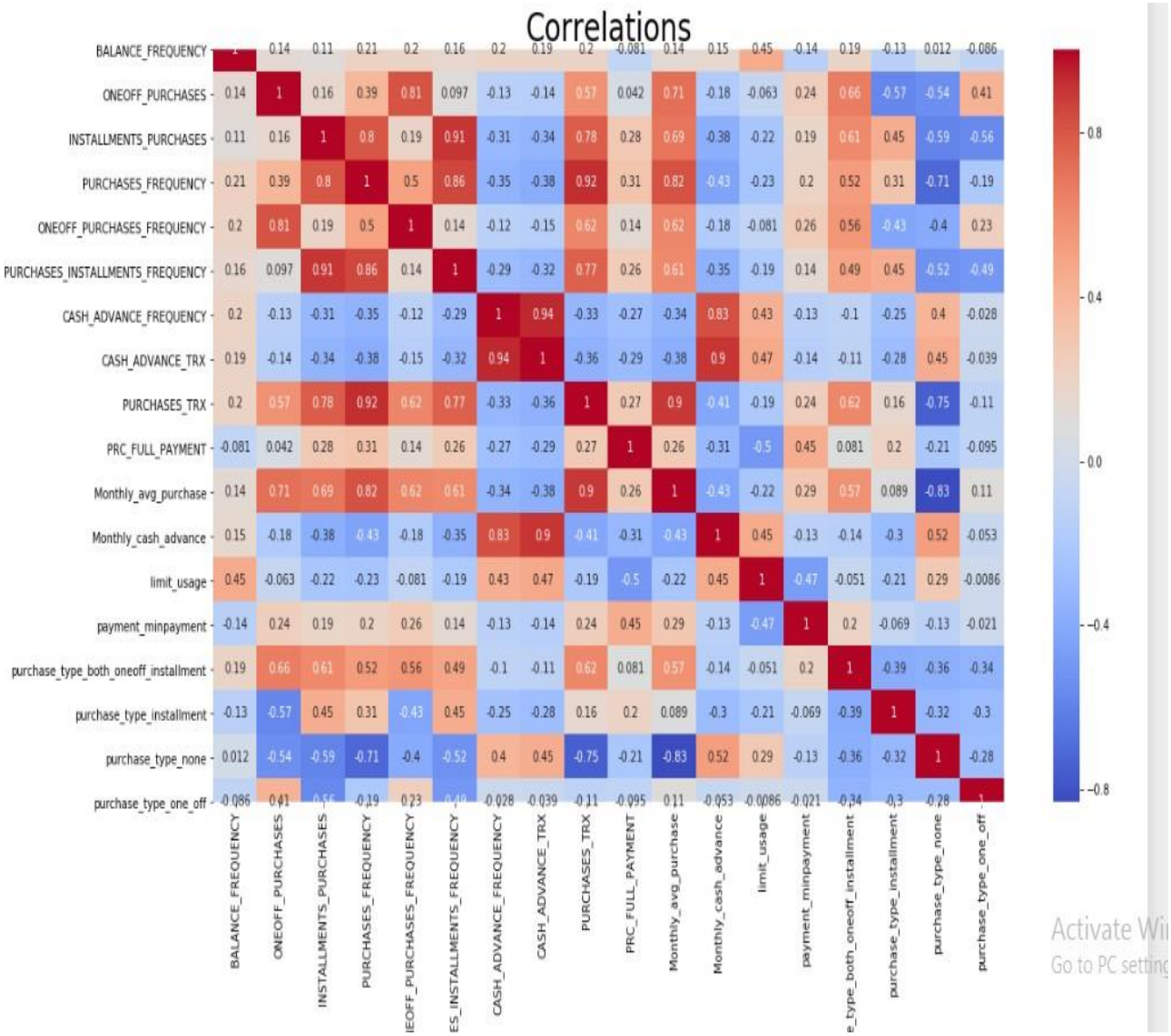
R code:

```
#---------- Deleting the features used in deriving KPI's

cc2_subset = subset(cc2_log, select = -c(BALANCE,PURCHASES,CASH_ADVANCE,TENURE,
                                         PAYMENTS,MINIMUM_PAYMENTS,CREDIT_LIMIT))
```

Co-relation Plot:

Python code:

```
##Correlation analysis
#Correlation plot
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(18, 10))

#Generate correlation matrix
credit_new_corr=cr_dummy.corr()

#Plot using seaborn library
sns.heatmap(credit_new_corr,cmap='coolwarm',annot=True);
plt.title('Correlations', size = 28);
```



Shape of data

Python code:

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

```
(8950, 12)
```

R code:

```
dim(cr_dummy)
.] 8950   12
```

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.

C. Standardizing the data

```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
```

```
cr_scaled = sc.fit_transform(cr_dummy)
```

```
cnames=cr_dummy.columns
```

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

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.

With the help of principal component analysis we will reducefeatures

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

```
er_scaled_df.head()
```

B ALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PRC_FULL_PAYMENT	Monthly_avg_purchase	Monthly_cash_advance	limit_us;
-0.148757	-0.987090	0.394480	-0.556368	-0.429030	-0.890667	-1.022
0.179616	-0.987090	-1.087454	0.391958	-1.523373	1.697282	0.322
0.492710	1.062022	-1.087454	-0 556368	0.564294	-0890667	-0 035
-0.857867	1.265778	-1.087454	-0.556368	0.891164	0.302372	-0.381
0.492710	-0.114307	-1.087454	-0.556368	-1.100298	-0.890667	0.893

```
#We have 17 features so our n_component i,il l be 12
pc=PCA(n_cornponents=12)
e_pca=pc.fit(er_scaled)
```

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

```
0.9999999999999999
```

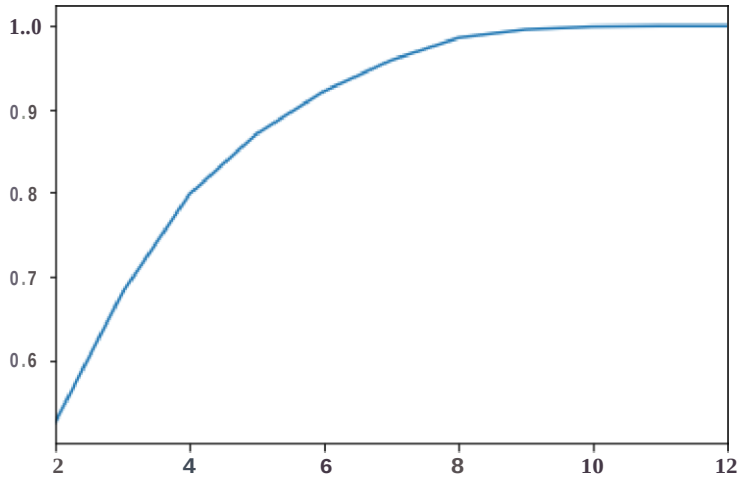
```
var_ratio={}
for n in range(2,13):
    pc=PCA(n_components=n)
    er_pca=pc.fit(er_scaled)
    var_ratio[n]=sum(er_pea.explained_variance_ratio_)
```

```
var ratio

{2: 0..524.5278273:il463  72,
 3"  0.6798 185763353778
 4"  0.797954 371936062:J
 5·  0.8703   142   7218 4 6 167  ,,
 6·  0..9 21.44078521746 5,
 7·  0.958.580  5083734691
 8"  0.98 5  00757191411J
 9 : 0.99  565 27471.024368 ,,
18 :  8 .999170648184879,
 11: 0.999999999999999,
 12: 0.9999999999999999}
```

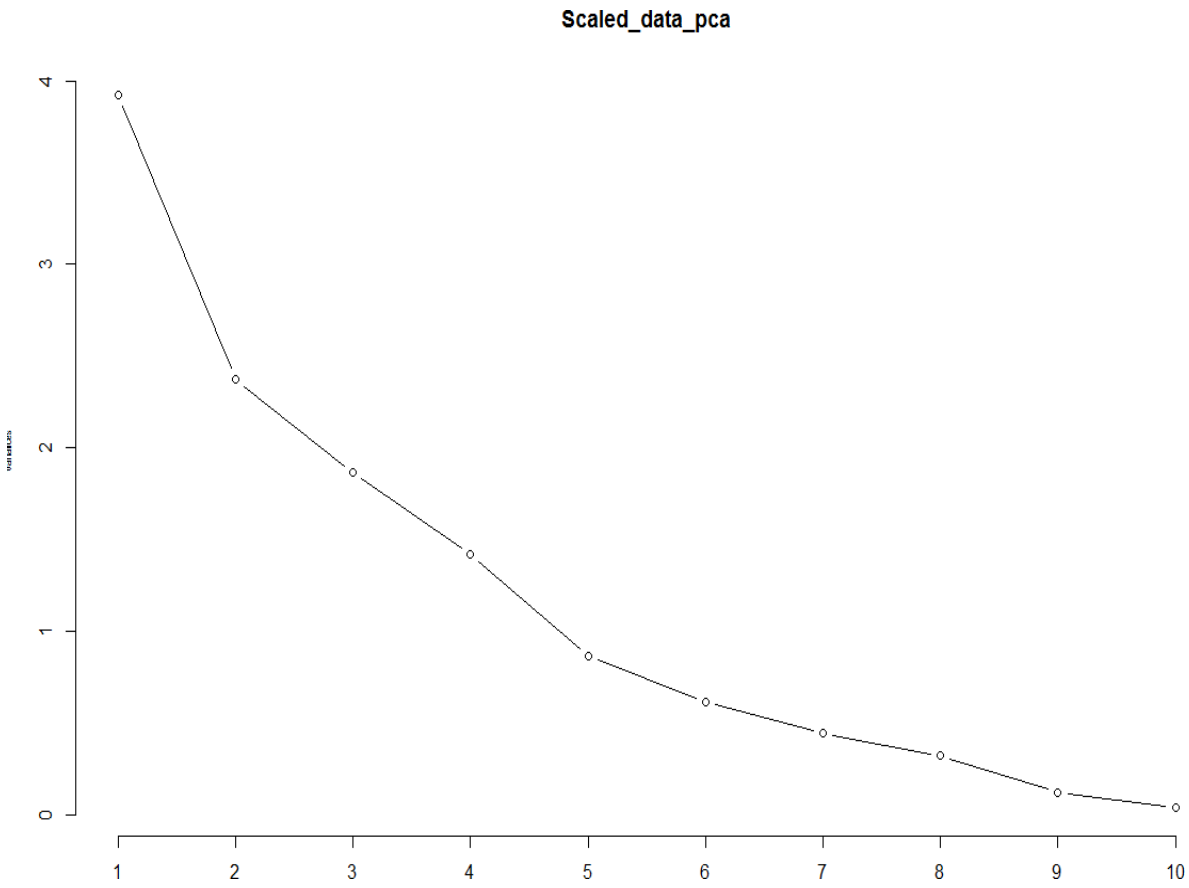
Pellformi 11g Fa cto Aniaiy sis

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

```
<matplotlib.axes._subplots.AxesSubplot at 0rxf7d0b6020t8>
```




```
> scaled_data = data.frame(scale(cr_dummy))
> scaled_data_pca = prcomp(scaled_data, center = TRUE)
> summary(scaled_data_pca)
Importance of components:
                          PC1   PC2   PC3   PC4    PCS    PC6    PC7    PC8    PC9   PC10   PC11     PC12
standard deviation     1.9803 1.5404 1.36511 .19060.931840.78327 0.66759 0.56 31 0.350 5 0.20546 0.09976 1.805e-14
Proportion of variance 0.3268 0.19 7 0.1553 0.1181 0.07236 0.05113 0.03714 0.02682 0.01025 0.00352 0.00083 0.000e.00
cumulative Proportion  0.3268 0.5245 0.6798 0.  980 0.8 031 0.92144 0.958580.98540 0.99565 0.99917 1.00000 0.000e+00
> plot(scaled_data_pca)
> screeplot(scaled_data_pca, type='li nes')
```



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.

```
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)
```

```
#        Get the first principal component for further
Credit_PCs = data.frame(Scaled_data_pca$x[,1:5])
#--------Get the dimension of Credit_PCs------------
dim(Credit_PCs)
] 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 variance focused 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.

```
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	-0.003657	-0.280037	-0.377715	-0.080637	-0.694366
ONEOFF_PURCHASES	0.310348	-0.476751	0.176584	0.022922	0.059421
INSTALLMENTS_PURCHASES	0.387344	0.149332	-0.415953	-0.005952	0.086688
PRC_FULL_PAYMENT	0.236269	0.264774	0.149254	0.292186	-0.500181
Monthly_avg_purchase	0.457091	-0.147058	-0.037818	-0.161685	-0.078480
Monthly_cash_advance	-0.306055	-0.197333	-0.134887	0.235794	-0.114960
limit_usage	-0.245785	-0.320928	-0.344660	-0.216925	-0.149191
payment_minpayment	0.222084	0.079774	0.236707	0.503980	-0.271904
purchase_type_both_oneoff_installment	0.325531	-0.297836	-0.275336	0.296625	0.292511
purchase_type_installment	0.076734	0.526673	-0.181909	-0.366295	-0.132341
purchase_type_none	-0.416051	0.030087	-0.060037	0.409001	0.003588
purchase_type_one_off	-0.022839	-0.254875	0.569127	-0.367873	-0.194871

5. Determining the Optimal number of clusters

5.1 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.

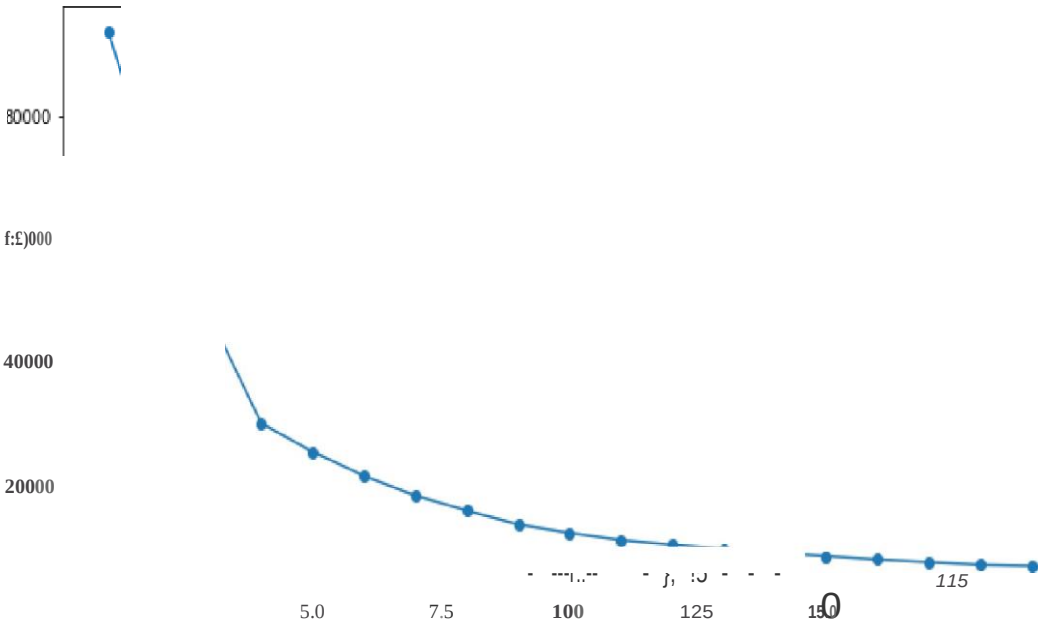
The Elbow method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn’t improve much better the total WSS.

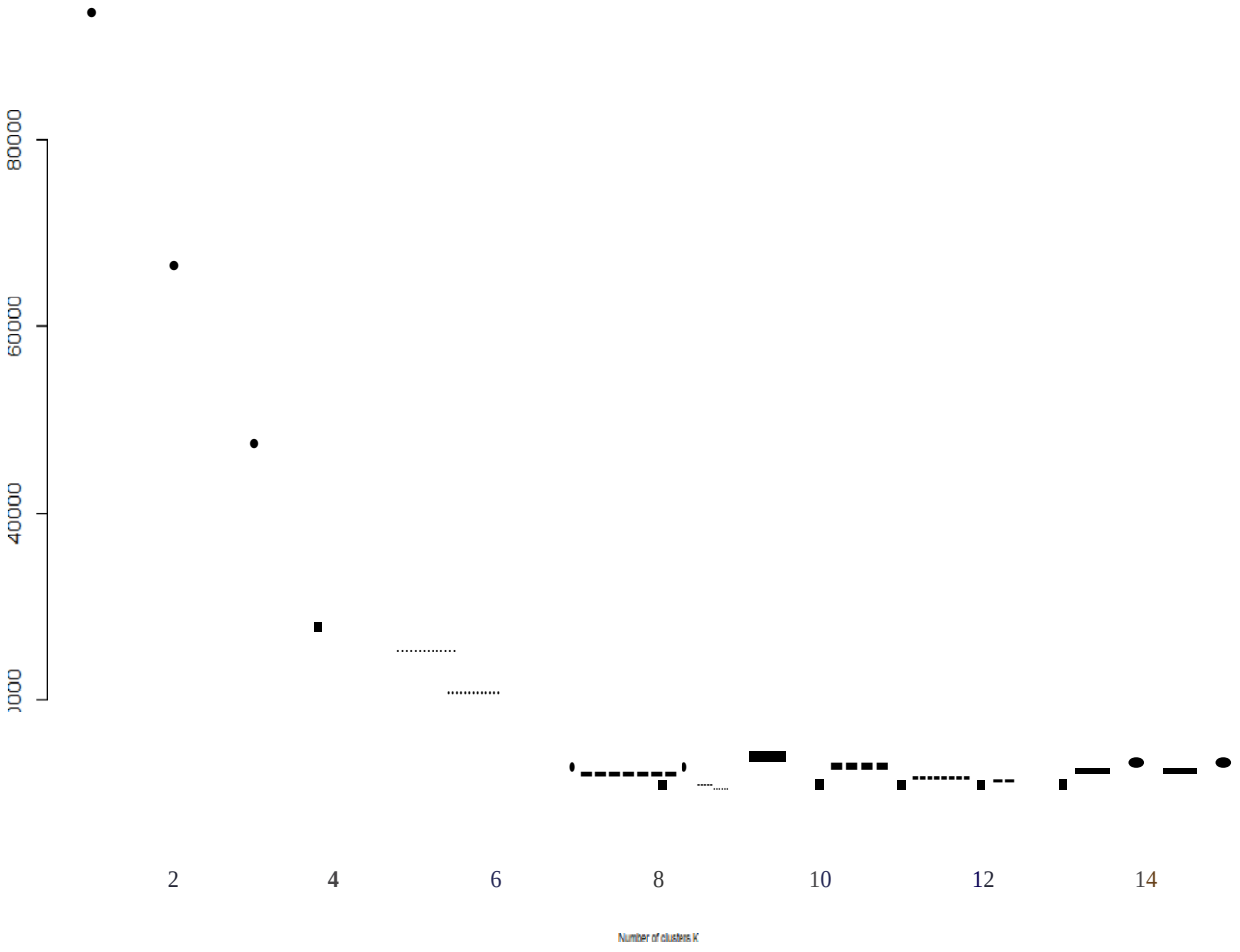
The optimal number of clusters can be defined as follow:

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

```
#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" )
```


80000
40000
20000
5.0
7.5
100
125
115


80000
60000
40000
2
4
6
8
10
12
14
Number of clusters K

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

5.2 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.

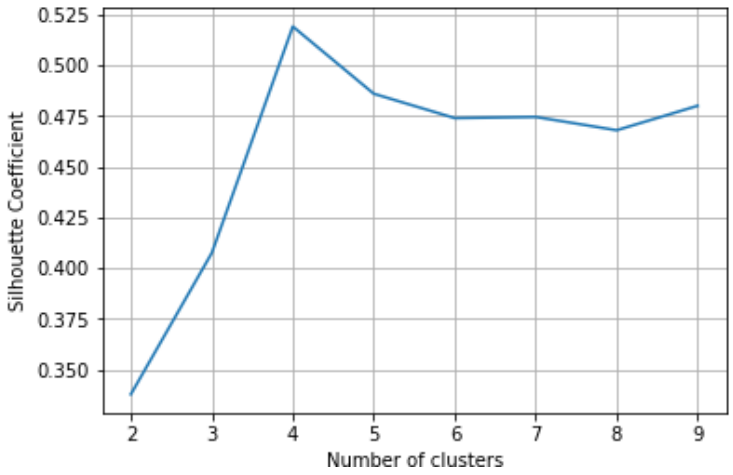
The algorithm is similar to the elbow method and can be computed as follow:

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

```
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)
```



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.

6. 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

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.

```
# 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()
```

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

ADDING THE LABELS TO THE DATASET

```
# 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)
```

```
cluster_df_4.head()
```

	PURCHASES_TRX	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	Monthly_avg_purchase	Monthly_cash_advance	limit_usage	CASH_ADVANCE_TRX
0	2	0.00	95.4	7.950000	0.000000	0.040901	0
1	0	0.00	0.0	0.000000	536.912124	0.457495	4
2	12	773.17	0.0	64.430833	0.000000	0.332687	0
3	1	1499.00	0.0	124.916667	17.149001	0.222223	1
4	1	16.00	0.0	1.333333	0.000000	0.681429	0

```
# Mean vaLu:e giv e s a good ·ind -icat io n of the di s t r i but io n of dat a.
#So i\le are findfog me mi va l ue for each var ia bl.e for ea ch cl.us t er
cl us t e r_4=cl us t e r'_df _4.groupby ( 'Clust er'_4 ')\
.appl y(lam bda x: x[col _kpi ]m ean( ) ).T
cl us t e r 4
```

Cluste-r_4	0	1	2	3
PUR CHASES_TR.X	7.109925	11.904825	32.959625	0.002937
ONEOfF _PU RCH ASES;	7 8 6.827679	0.000000	137 9.884427	0.000000
INSTALl MENTS_PURCHASES;	0 000000	538_114 608	88'8.049775	0.002173
Mont hly-'avg_purdhase	69.688958	46.99 4978	192 .685172	0.000181
Monlh ly_cash-' ad\rarice	78.995966	37.6824 72	67.821985	183.578865
[imit_uiSage	0.381074	0.271618	0.353548	0.573 689
CA SH_ AD VAMCE_TR.X	2 .932231	1-261178	2.832733	6.302.007
payme-nt_minpayment	5.498134	13-243911	7.231360	10.079537
both_oneoff_iniS.tallment	0.000000	0.000000	1.000000	0.000000
installment	0 000000	1.000000	0 000000	0.000489
one_off	1 000000	0 000000	0 000000	0 000000
norie	0.000000	0.000000	0.000000	0.999511
CREDIT_LIMIT	45 15.920572	3366.052848	573 8.829463	4032.824824

```
> as.data.frma e(t abl e(ereditel uster· cl ust er )J
  var 1 Freq
1     1 2259
2     2 13.7
3     3 20 3
4     4 277
```

	V1	V2	V3	V4
cluster	1	2	3	4
PURCHASES_TRX	11.904825144	7.109925293	0.002936858	32.959625090
ONEOFF_PURCHASES	0.0000	786.8277	0.0000	1379.8844
INSTALLMENTS_PURCHASES	5.381146e+02	0.000000e+00	2.173275e-03	8.880498e+02
monthly_avg_purchase	4.699498e+01	6.968896e+01	1.811062e-04	1.926852e+02
monthly_cash_advance	37.68247	78.99597	183.57887	67.82199
limit_usage	0.2716175	0.3810738	0.5736890	0.3535485
CASH_ADVANCE_TRX	1.261178	2.932231	6.302007	2.832733
payment_minpayment	13.243911	5.498136	10.079538	7.231176
purchase_type_both	0	0	0	1
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	1	0	0
CREDIT_LIMIT	3366.053	4515.921	4032.825	5738.829

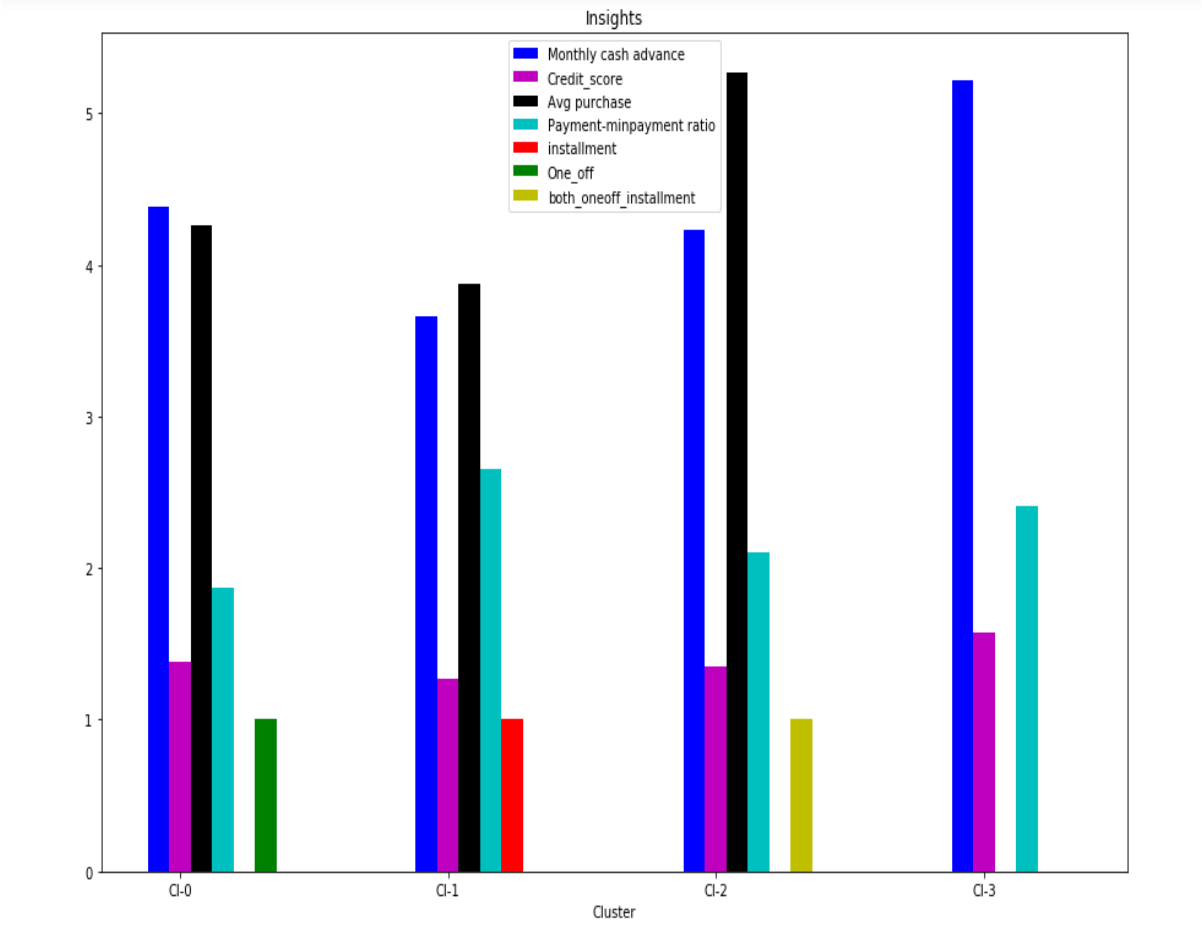
Here in output of both python and R code we can see that customers in each cluster of both the code is same and mean value in both the output is exactly same.

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

7. 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 have good credit score, doing maximum one off transactions, have least payment minpayment ratio, take high cash advance.
- Cluster 1 customers take least monthly cash advance, have lowest credit score, have highest payment minpayment ratio, doing maximum installment transactions.

- Cluster 2 customers have highest monthly purchase and do the highest one off and installment transaction and have good credit score.
- Cluster 3 customers take highest monthly cash advance, highest credit score and have high payment minpayment ratio.

8. Market Strategy Suggestion

a. Group 0

- 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.

b. Group 1

- 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 transactions

c. Group 2

- This group is performing best among all as cutomers 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.

d. Group 3

- They have good credit score and taking only cash on advance. We can target them by providing less interest rate on Installment purchase transaction or cashback/discount on one off purchase transaction.

9. References

1. CLUSTERING https://www.udemy.com/course/machinelearning/le
2. For PCA https://www.udemy.com/course/machinelearning/learn/lec