Project Name – Credit Card Segmentation

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.

DATA:

Number of attributes:

● CUST_ID Credit card holder ID
● BALANCE Monthly average balance (based on daily balance averages)
● BALANCE_FREQUENCY Ratio of last 12 months with balance
● PURCHASES Total purchase amount spent during last 12 months
● ONEOFF_PURCHASES Total amount of one-off purchases
● INSTALLMENTS_PURCHASES Total amount of installment purchases
● CASH_ADVANCE Total cash-advance amount
● PURCHASES_ FREQUENCY-Frequency of purchases (percentage of months with at least on purchase)
● ONEOFF_PURCHASES_FREQUENCY Frequency of one-off-purchases
● PURCHASES_INSTALLMENTS_FREQUENCY Frequency of installment purchases
● CASH_ADVANCE_ FREQUENCY Cash-Advance frequency
● AVERAGE_PURCHASE_TRX Average amount per purchase transaction
● CASH_ADVANCE_TRX Average amount per cash-advance transaction
● PURCHASES_TRX Average amount per purchase transaction
● CREDIT_LIMIT Credit limit
● PAYMENTS-Total payments (due amount paid by the customer to decrease their statement balance) in the period
● MINIMUM_PAYMENTS Total minimum payments due in the period.
● PRC_FULL_PAYMENT- Percentage of months with full payment of the due statement balance
● TENURE Number of months as a customer

In [ ]:

```
1
```

Credit Card Customers Segmenting with Machine Learning

I am going to identifying marketable segments with unsupervised machine learning.

Segmentation in marketing is a technique used to divide customers or other entities into groups based on attributes such as behaviour or demographics.

It is useful to identify segments of customers who may respond in a similar way to specific marketing techniques such as email subject lines or display advertisements.

As it gives businesses the ability to tailor marketing messages and timing to generate better response rates and provide improved consumer experiences.

In the following project, I will be using a dataset containing a number of behavioural attributes for credit card customers (dataset given by the edwisor).

I will be using the scikit-learn python machine learning library to apply an unsupervised machine learning technique known as clustering to identify segments that may not immediately be apparent to human cognition.

The dataset consists of 18 features about the behaviour of credit card customers. These include variables such as the balance currently on the card, the number of purchases that have been made on the account, the credit limit, and many others.

There are two broad set of methodologies for segmentation: **Objective (supervised)** and **Non-Objective (unsupervised) segmentation methodologies**.
As the name indicates, a supervised methodology requires the objective to be stated as the basis for segmentation.<br.

But here after reading and understading the dataset, the segments are different with respect to the “generic profile” of observations belonging to each segment, but not with regards to any specific outcome of interest (i.e. no target label is available). That's why I choose unsupervised machine learning.

By analysing the dataset with Unsurvised Machine Learning Algorithm, I have to determine based on the dataset available what are the Marketing Strategy is there.

As I am going to identifying marketable segments with unsupervised machine learning. So here I am not going to use the supervised machine learning algorithm like Multiple Linear Regression, Logistic Regression, Random Forest etc.

The techniques for building non-objective segmentation I am going to use are cluster analysis, K nearest neighbor techniques, Hierarchichal Clustering(Agglomerative), Principal

component Analysis(PCA).

Clustering

Clustering is one of the most common exploratory data analysis technique used to get an intuition about the structure of the data.
It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different.
In other words, we try to find homogeneous subgroups within the data such that data points in each cluster are as similar as possible according to a similarity measure such as euclidean-based distance or correlation-based distance.
The decision of which similarity measure to use is application-specific.

Clustering analysis can be done on the basis of features where we try to find subgroups of samples based on features or on the basis of samples where we try to find subgroups of features based on samples.
We'll cover here clustering based on features.
Clustering is used in market segmentation; where we try to fined customers that are similar to each other whether in terms of behaviors or attributes, image segmentation/compression; where we try to group similar regions together, document clustering based on topics, etc.

Unlike supervised learning, clustering is considered an unsupervised learning method since we don't have the ground truth to compare the output of the clustering algorithm to the true labels to evaluate its performance.
We only want to try to investigate the structure of the data by grouping the data points into distinct subgroups.

In [ ]:

```
1
```

K-means Clustering in Machine Learning

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group.

Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.

The objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset."

A cluster refers to a collection of data points aggregated together because of certain similarities.

I’ll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster.

Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares.

In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

The ‘means’ in the K-means refers to averaging of the data; that is, finding the centroid.

The way kmeans algorithm works is as follows:

The goal of this algorithm is to find K groups in the data. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the K-means clustering algorithm are:

- The centroids of the K clusters, which can be used to label new data
- Labels for the training data (each data point is assigned to a single cluster)

K-means works by defining spherical clusters that are separable in a way so that the mean value converges towards the cluster center. Because of this, K-Means may underperform sometimes.

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.

1. It halts creating and optimizing clusters when either:
2. The centroids have stabilized — there is no change in their values because the clustering has been successful.
3. The defined number of iterations has been achieved.

In [ ]:

```
1
```

Hierarchical clustering

Hierarchical clustering is one of the popular and easy to understand clustering technique.
In data mining and statistics, hierarchical clustering analysis is a method of cluster analysis which seeks to build a hierarchy of clusters i.e. tree type structure based on the hierarchy.
This clustering technique is divided into two types:

1. Agglomerative

2. Divisive

1. Agglomerative Hierarchical clustering Technique:

Also known as bottom-up approach or hierarchical agglomerative clustering (HAC).
A structure that is more informative than the unstructured set of clusters returned by flat clustering.
This clustering algorithm does not require us to prespecify the number of clusters.
Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerates pairs of clusters until all clusters have been merged into a single cluster that contains all data.
In this technique, initially each data point is considered as an individual cluster.
At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed.

Algorithm :

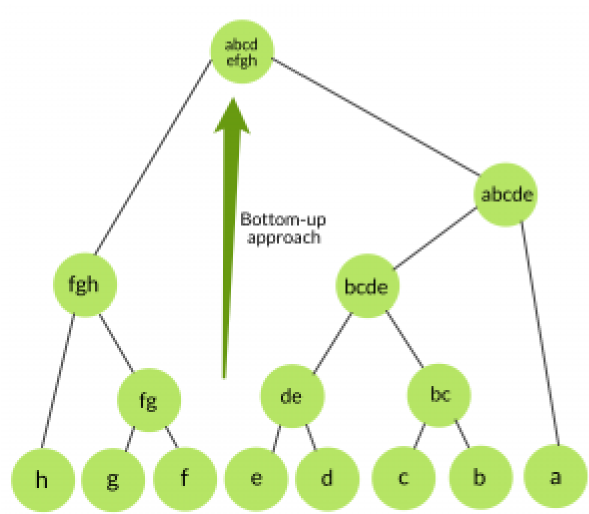
given a dataset (d1, d2, d3, ...dN) of size N

#compute the distance matrix
for i=1 to N:

#as the distance matrix is symmetric about

#the primary diagonal so we compute only lower

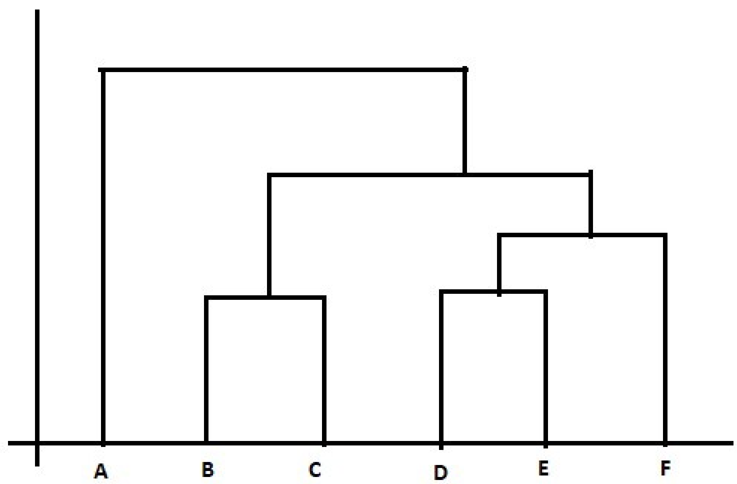
#part of the primary diagonal
--- for j=1 to i:
--- --- dis_mat[i][j] = distance[di, dj]
each data point is a singleton cluster
repeat
--- merge the two cluster having minimum distance
--- update the distance matrix
untill only a single cluster remains



Hierarchical agglomerative clustering

The Hierarchical clustering Technique can be visualized using a Dendrogram.

A Dendrogram is a tree-like diagram that records the sequences of merges or splits which represent the Dendogram.



2. Divisive Hierarchical clustering Technique:

Since the Divisive Hierarchical clustering Technique is not much used in the real world, I'll give a brief of the Divisive Hierarchical clustering Technique.
In simple words, we can say that the Divisive Hierarchical clustering is exactly the opposite of the **Agglomerative Hierarchical clustering.**
Also known as top-down approach.
This algorithm also does not require to prespecify the number of clusters.
Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been splitted into singleton cluster.

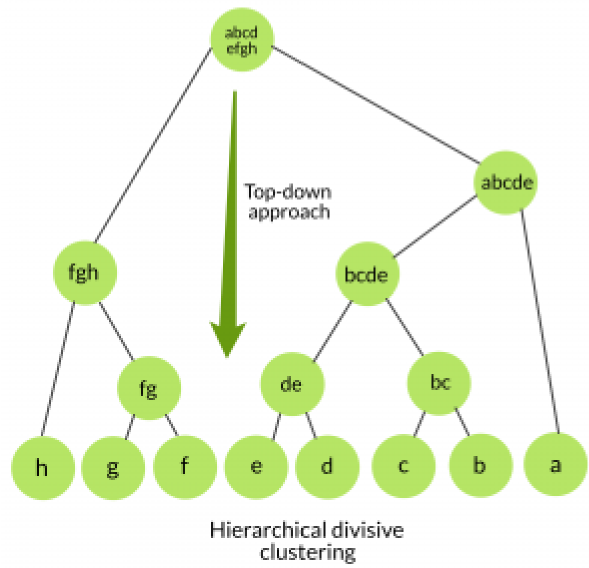
In Divisive Hierarchical clustering, we consider all the data points as a single cluster and in each iteration, we separate the data points from the cluster which are not similar.
Each data point which is separated is considered as an individual cluster. In the end, we'll be left with n clusters.

As we're dividing the single clusters into n clusters, it is named as **Divisive Hierarchical clustering.**

Algorithm :

given a dataset (d1, d2, d3, ...dN) of size N
at the top we have all data in one cluster
the cluster is split using a flat clustering method eg. K-Means etc

repeat
choose the best cluster among all the clusters to split
split that cluster by the flat clustering algorithm
untill each data is in its own singleton cluster



In [ ]:
```
1
```

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a **dimension-reduction tool** that can be used **to reduce a large set of variables to a small set** that still contains most of the information in the large set.

Objectives of principal component analysis:

- Principal component analysis (PCA) is a mathematical procedure that transforms a number of (possibly) **correlated variables into a (smaller) number of uncorrelated variables** called principal components.
- The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.
- PCA reduces attribute space from a larger number of variables to a smaller number of factors and as such is a "non-dependent" procedure (that is, it does not assume a dependent variable is specified).
- PCA is a dimensionality reduction or data compression method. The goal is dimension reduction and there is no guarantee that the dimensions are interpretable (a fact often not appreciated by (amateur) statisticians).
- To select a subset of variables from a larger set, based on which original variables have the highest correlations with the principal component.

I am going to use Principal component analysis (PCA) because of the three reasons:

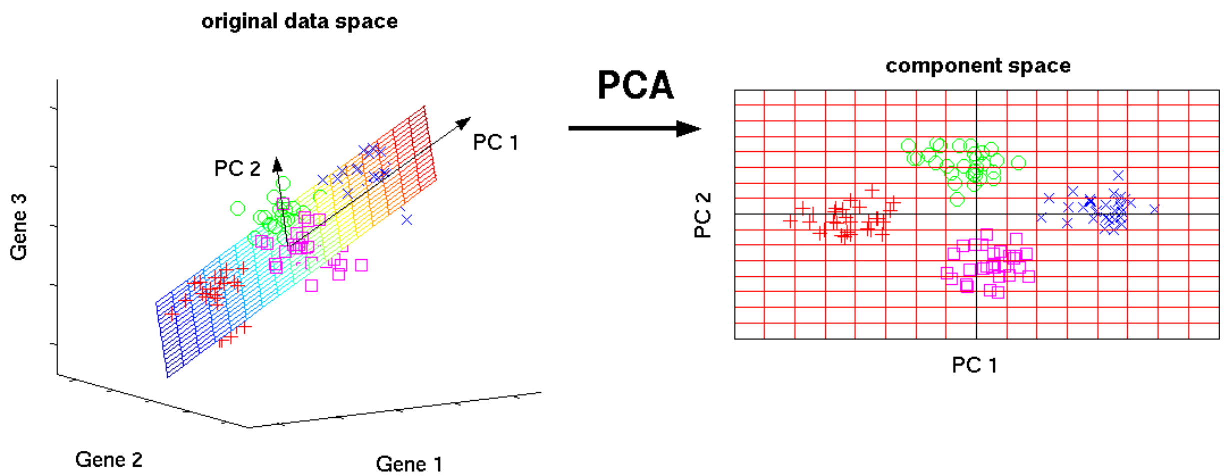
- I want to reduce the number of variables, but aren't able to identify variables to completely remove from consideration.
- I want to ensure variables are independent of one another.
- I comfortable making independent variables less interpretable.

Let's understand it using an example:

Let's say I have a data set of dimension 300 (n) × 50 (p). n represents the number of observations and p represents number of predictors. Since I have a large p = 50, there can be p(p-1)/2 scatter plots i.e more than 1000 plots possible to analyze the variable relationship.

In this case, it would be a lucid approach to select a subset of p (p << 50) predictor which captures as much information. Followed by plotting the observation in the resultant low dimensional space.

The image below shows the transformation of a high dimensional data (3 dimension) to low dimensional data (2 dimension) using PCA. Not to forget, each resultant dimension is a linear combination of p features.



What are principal components ?

A principal component is a normalized linear combination of the original predictors in a data set. In image above, PC1 and PC2 are the principal components. Let's say we have a set of predictors as X^1, X^2...,Xp

The principal component can be written as:

$$Z^1 = \Phi^{11}X^1 + \Phi^{21}X^2 + \Phi^{31}X^3 + ... + \Phi p^1 Xp$$

where,

- Z^1 is first principal component

- Φp^1 is the loading vector comprising of loadings (Φ^1, Φ^2..) of first principal component. The loadings are constrained to a sum of square equals to 1. This is because large magnitude of loadings may lead to large variance. It also defines the direction of the principal component (Z^1) along which data varies the most. It results in a line in p dimensional space which is closest to the n observations. Closeness is measured using average squared euclidean distance.
- X^1..Xp are normalized predictors. Normalized predictors have mean equals to zero and standard deviation equals to one.

Therefore,

First principal component is a linear combination of original predictor variables which captures the maximum variance in the data set. It determines the direction of highest variability in the data. Larger the variability captured in first component, larger the information captured by component. No other component can have variability higher than first principal component.

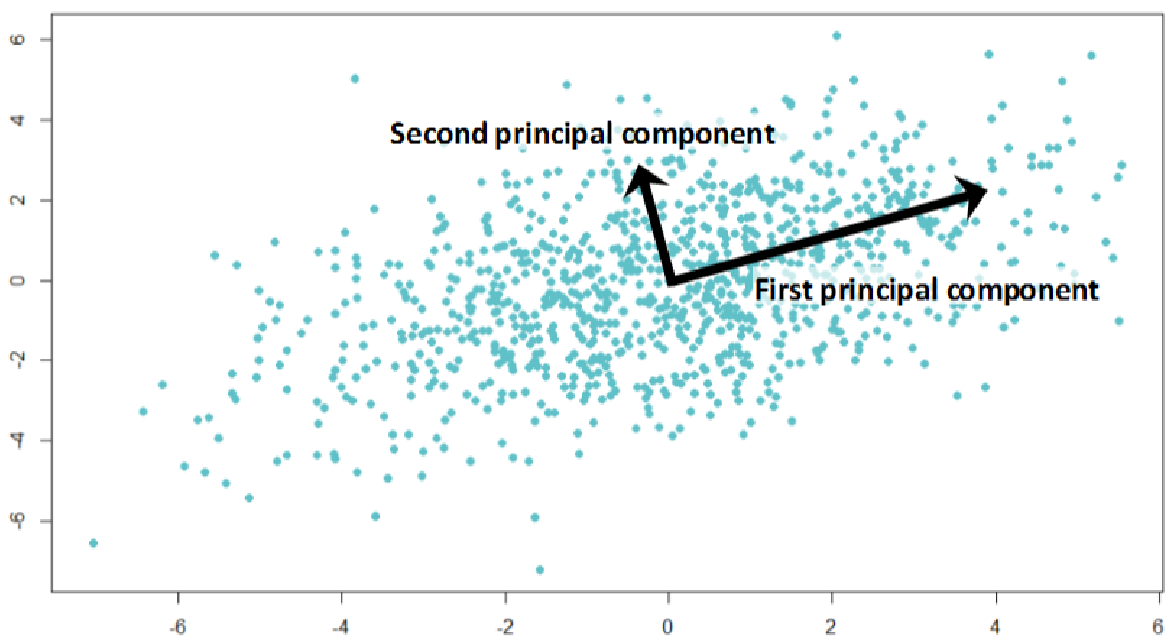
The first principal component results in a line which is closest to the data i.e. it minimizes the sum of squared distance between a data point and the line.

Similarly, we can compute the second principal component also.

Second principal component (Z^2) is also a linear combination of original predictors which captures the remaining variance in the data set and is uncorrelated with Z^1. In other words, the correlation between first and second component should is zero. It can be represented as:

$Z^2 = \Phi^{12}X^1 + \Phi^{22}X^2 + \Phi^{32}X^3 + ... + \Phi p2Xp$

If the two components are uncorrelated, their directions should be orthogonal (image below). This image is based on a simulated data with 2 predictors. Notice the direction of the components, as expected they are orthogonal. This suggests the correlation b/w these components in zero.



All succeeding principal component follows a similar concept i.e. they capture the remaining variation without being correlated with the previous component. In general, for n × p dimensional data, min(n-1, p) principal component can be constructed.

The directions of these components are identified in an unsupervised way i.e. the response variable(Y) is not used to determine the component direction. Therefore, it is an unsupervised approach.

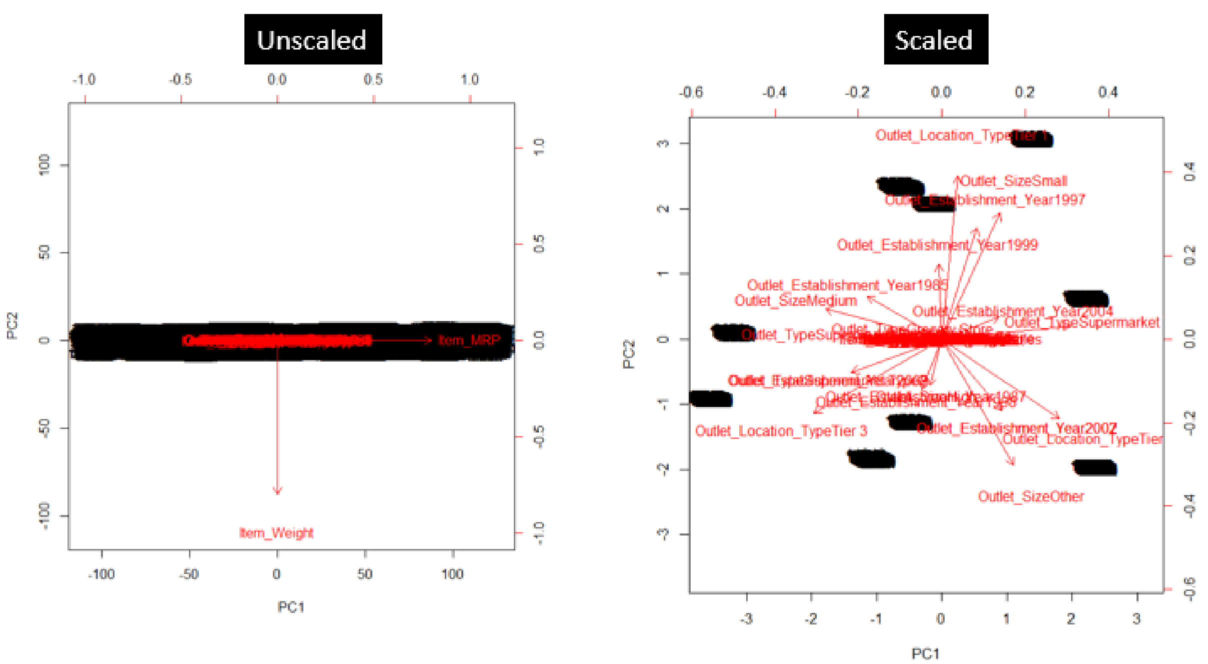
Note: Partial least square (PLS) is a supervised alternative to PCA. PLS assigns higher weight to variables which are strongly related to response variable to determine principal components.

Why is normalization of variables necessary ?

The principal components are supplied with normalized version of original predictors. This is because, the original predictors may have different scales. For example: Imagine a data set with variables' measuring units as gallons, kilometers, light years etc. It is definite that the scale of variances in these variables will be large.

Performing PCA on un-normalized variables will lead to insanely large loadings for variables with high variance. In turn, this will lead to dependence of a principal component on the variable with high variance. This is undesirable.

As shown in image below, PCA was run on a data set twice (with unscaled and scaled predictors). This data set has ~40 variables. You can see, first principal component is dominated by a variable Item_MRP. And, second principal component is dominated by a variable Item_Weight. This domination prevails due to high value of variance associated with a variable. When the variables are scaled, we get a much better representation of variables in 2D space.



In [ ]: 1

Silhouette analysis (S.A.)

S.A. is a way to measure how close each point in a cluster is to the points in its neighboring clusters.
Its a neat way to find out the optimum value for k during k-means clustering. Silhouette values lies in the range of [-1, 1].
A value of +1 indicates that the sample is far away from its neighboring cluster and very close to the cluster its assigned.
Similarly, value of -1 indicates that the point is close to its neighboring cluster than to the cluster its assigned.
And, a value of 0 means its at the boundary of the distance between the two cluster. Value of +1 is idea and -1 is least preferred. Hence, higher the value better is the cluster configuration.

Mathematically: Lets define Silhouette for each of the sample in the data set.

For an example (i) in the data, lets define a(i) to be the mean distance of point (i) w.r.t to all the other points in the cluster its assigned (A). We can interpret a(i) as how well the point is assigned to the cluster. Smaller the value better the assignment.

Similarly, lets define b(i) to be the mean distance of point(i) w.r.t. to other points to its closet neighboring cluster (B). The cluster (B) is the cluster to which point (i) is not assigned to but its distance is closest amongst all other cluster.

Thus, the silhouette s(i) can be calculated as

s(i) = (b(i) - a(i))/max(b(i), a(i))

We can easily say that s(i) lies in the range of [-1,1].

For s(i) to be close to 1, a(i) has be be very small as compared to b(i), i.e. a(i) <<< b(i). This happens when a(i) is very close to its assigned cluster. A large value of b(i) implies its extremely far from its next closest cluster. Hence, s(i) == 1 indicates that the data set (i) is well matched in the cluster assignment.

The above definition talks about Silhouette score for each data item.

Mean Silhouette score: Mean score can be simply calculated by taking the mean of silhouette score of all the examples in the data set. This gives us one value representing the Silhouette score of the entire cluster.

Advantages of using S.A: The best advantage of using S.A. score for finding the best number of cluster is that you use it for un-labelled data set. This is usually the case when running k-means. Hence, I prefer this k-means scores.

In [ ]:

```
1
```

Choosing K

If the true label is not known in advance, then K-Means clustering can be evaluated using Elbow Criterion , Silhouette Coefficient , cross-validation, information criteria, the information theoretic jump method, and the G-means algorithm.

Elbow Criterion Method:

The idea behind elbow method is to run k-means clustering on a given dataset for a range of values of k (e.g k=1 to 10), for each value of k, calculate sum of squared errors (SSE).

Calculate the mean distance between data points and their cluster centroid. Increasing the number of clusters(K) will always reduce the distance to data points, thus decrease this metric, to the extreme of reaching zero when K is as same as the number of data points. So the goal is to choose a small value of k that still has a low SSE.

We run the algorithm for different values of K(say K = 10 to 1) and plot the K values against SSE(Sum of Squared Errors). And select the value of K for the elbow point.

Silhouette Coefficient Method:

A higher Silhouette Coefficient score relates to a model with better-defined clusters. The Silhouette Coefficient is defined for each sample and is composed of two scores:

- The mean distance between a sample and all other points in the same class.
- The mean distance between a sample and all other points in the next nearest cluster.

The Silhouette Coefficient is for a single sample is then given as:

s = (b-a)/max(a,b)

To find the optimal value of k for KMeans, loop through 1..n for n_clusters in KMeans and calculate Silhouette Coefficient for each sample.

A higher Silhouette Coefficient indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.

In [ ]:
```
1
```

In [ ]:
```
1
```

Importing Libraries

In [1]:

```
#Basic python library which need to import
import pandas as pd
import numpy as np

#Date stuff
from datetime import datetime
from datetime import timedelta

#Library for Nice graphing
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as sn
%matplotlib inline

#Library for statistics operation
import scipy.stats as stats

# Date Time library
from datetime import datetime

#Machine learning Library
import statsmodels.api as sm
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error, mean_squared_error

#Import preprocessing libraries
from sklearn.preprocessing import MinMaxScaler , StandardScaler, Imputer


# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Settings
pd.set_option('display.max_columns', None)
sns.set(style="darkgrid")
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
```

Data Pre-Processing

In [2]:

```
# reading data into dataframe
credit= pd.read_csv("credit-card-data.csv")
```

Information about data set

In [3]:
```
# shows the top 5 rows of data in the DataFrame.
credit.head()
```

Out[3]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTA
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	

In [4]:
```
# info() is used to get a concise summary of the dataframe.
credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
CUST_ID                             8950 non-null object
BALANCE                             8950 non-null float64
BALANCE_FREQUENCY                   8950 non-null float64
PURCHASES                           8950 non-null float64
ONEOFF_PURCHASES                    8950 non-null float64
INSTALLMENTS_PURCHASES              8950 non-null float64
CASH_ADVANCE                        8950 non-null float64
PURCHASES_FREQUENCY                 8950 non-null float64
ONEOFF_PURCHASES_FREQUENCY          8950 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY    8950 non-null float64
CASH_ADVANCE_FREQUENCY              8950 non-null float64
CASH_ADVANCE_TRX                    8950 non-null int64
PURCHASES_TRX                       8950 non-null int64
CREDIT_LIMIT                        8949 non-null float64
PAYMENTS                            8950 non-null float64
MINIMUM_PAYMENTS                    8637 non-null float64
PRC_FULL_PAYMENT                    8950 non-null float64
TENURE                              8950 non-null int64
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

In [5]:
```
# Return a tuple representing the dimensionality of the DataFrame.
credit.shape
```

Out[5]: (8950, 18)

Firstly I had run the following code to inspect the data types to find out if there are any categorical variables that may need transforming. I can see from the result that all features are numeric except for **CUST_ID**. But since I don't need this feature to train the model I don't have to do any transforming here.

Data cleaning

To begin with, we will need to inspect the data to find out what cleaning and transformation may be needed. The scikit-learn library requires that all data have no null values and that all values must be numeric.

In [6]:
```
# Find the total number of missing values in the dataframe
print ("\nMissing values :  ", credit.isnull().sum().values.sum())

# printing total numbers of Unique value in the dataframe.
print ("\nUnique values :  \n",credit.nunique())
```

```
Missing values :   314

Unique values :
 CUST_ID                             8950
BALANCE                             8871
BALANCE_FREQUENCY                     43
PURCHASES                           6203
ONEOFF_PURCHASES                    4014
INSTALLMENTS_PURCHASES              4452
CASH_ADVANCE                        4323
PURCHASES_FREQUENCY                   47
ONEOFF_PURCHASES_FREQUENCY            47
PURCHASES_INSTALLMENTS_FREQUENCY      47
CASH_ADVANCE_FREQUENCY                54
CASH_ADVANCE_TRX                      65
PURCHASES_TRX                        173
CREDIT_LIMIT                         205
PAYMENTS                            8711
MINIMUM_PAYMENTS                    8636
PRC_FULL_PAYMENT                      47
TENURE                                 7
dtype: int64
```

Running the following code tells me that only two features have null values **‘CREDIT_LIMIT’ and ‘MINIMUM_PAYMENTS’**. Additionally, less than 5% of each column has nulls. This means that we should be ok to fill these with a sensible replacement value and should still be able to use the feature.

In [7]:
```
# Intital descriptive analysis of data. describe() is used to view some
# like percentile, mean, std etc. of a data frame or a series of numeric
credit.describe()
```

Out[7]:

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

In [8]:
```
# to see the  how many null value are true
credit['CREDIT_LIMIT'].isnull().value_counts()
```

Out[8]:
```
False    8949
True        1
Name: CREDIT_LIMIT, dtype: int64
```

In [9]:
```
# descriptive analysis of data (i.e. CREDIT_LIMIT).
credit['CREDIT_LIMIT'].describe()
```

Out[9]:
```
count     8949.000000
mean      4494.449450
std       3638.815725
min         50.000000
25%       1600.000000
50%       3000.000000
75%       6500.000000
max      30000.000000
Name: CREDIT_LIMIT, dtype: float64
```

In [10]:
```
credit[credit['CREDIT_LIMIT'].isnull()]
```

Out[10]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INST
5203	C15349	18.400472	0.166667	0.0	0.0	

Missing Value Treatment

The following code fills the missing value with the most commonly occurring value in the column. I could equally use mean or median, or indeed another approach.

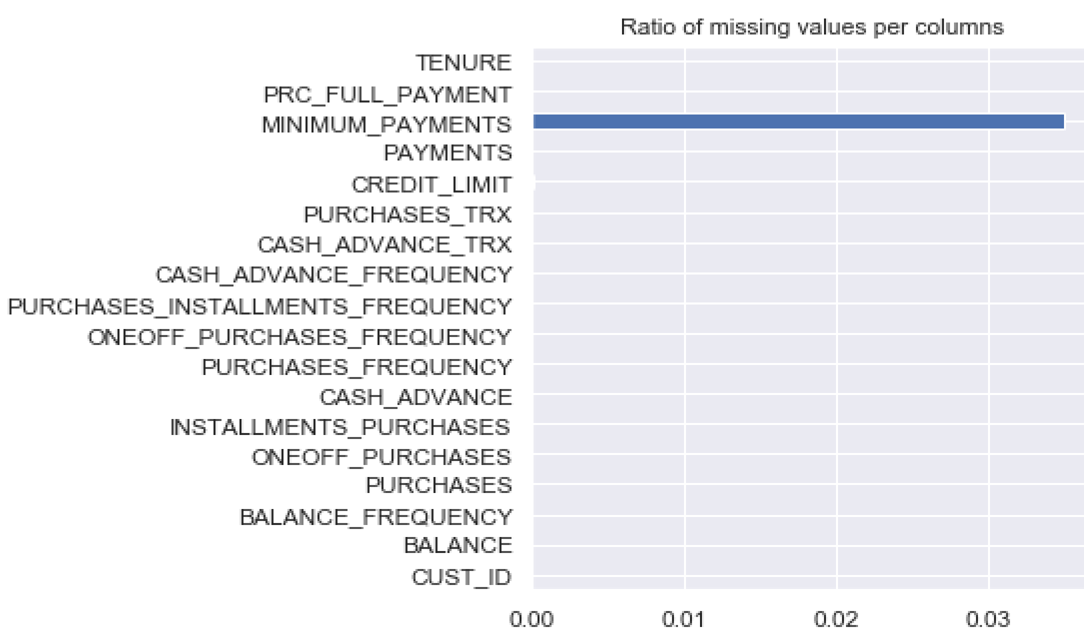
-Since there are missing values in the data so we are imputing them with **median**.

In [11]:
```
# total sum of null values present in dataframe
credit.isnull().sum()
```

Out[11]:
```
CUST_ID                              0
BALANCE                              0
BALANCE_FREQUENCY                    0
PURCHASES                            0
ONEOFF_PURCHASES                     0
INSTALLMENTS_PURCHASES               0
CASH_ADVANCE                         0
PURCHASES_FREQUENCY                  0
ONEOFF_PURCHASES_FREQUENCY           0
PURCHASES_INSTALLMENTS_FREQUENCY     0
CASH_ADVANCE_FREQUENCY               0
CASH_ADVANCE_TRX                     0
PURCHASES_TRX                        0
CREDIT_LIMIT                         1
PAYMENTS                             0
MINIMUM_PAYMENTS                   313
PRC_FULL_PAYMENT                     0
TENURE                               0
dtype: int64
```

In [12]:
```
#  visualize the Null values in the graph
plt.figure(figsize=(5, 5))
credit.isnull().mean(axis=0).plot.barh()
plt.title("Ratio of missing values per columns")
```

Out[12]:
```
Text(0.5, 1.0, 'Ratio of missing values per columns')
```



In [13]:
```
#  missing values in the data so I am imputing them with median.
credit['CREDIT_LIMIT'].fillna(credit['CREDIT_LIMIT'].median(),inplace=Tr
credit['MINIMUM_PAYMENTS'].fillna(credit['MINIMUM_PAYMENTS'].median(),in
credit.isnull().sum()
```

Out[13]:
```
CUST_ID                             0
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
CREDIT_LIMIT                        0
PAYMENTS                            0
MINIMUM_PAYMENTS                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

I am going to drop the CUST_ID column as I won’t need this for training.

In [14]:
```
cols_to_drop = 'CUST_ID'
credit = credit.drop([cols_to_drop], axis=1)
```

EXPLORATORY DATA ANALYSIS

In [15]:
```
# for checking purpose what is the total lenght of BALANCE is less than
# It means that total count of BALANCE(Monthly average balance) is less
len(credit[credit["BALANCE"]<2000])
```

Out[15]:
```
6660
```

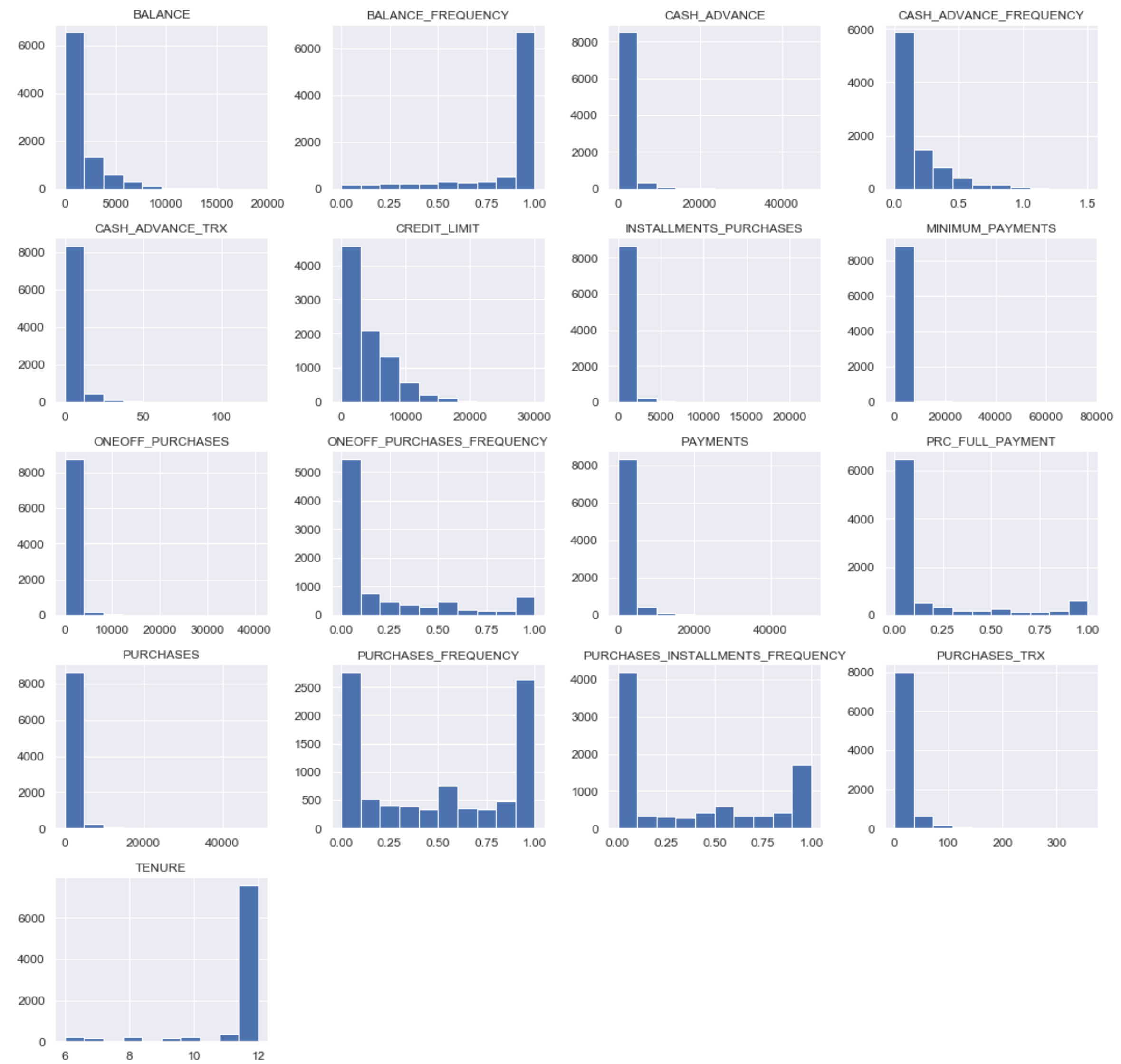
In [16]:
```
# for checking purpose what is the total lenght of CASH_ADVANCE is less
# It means that total count of CASH_ADVANCE(Total cash-advance amount) i
len(credit[credit["CASH_ADVANCE"]<5000])
```

Out[16]:
```
8559
```

In [17]:
```
# I have taken above two lenght just to cross check with my histogram gr
# the data with respect to the total length or total count of the each d
```

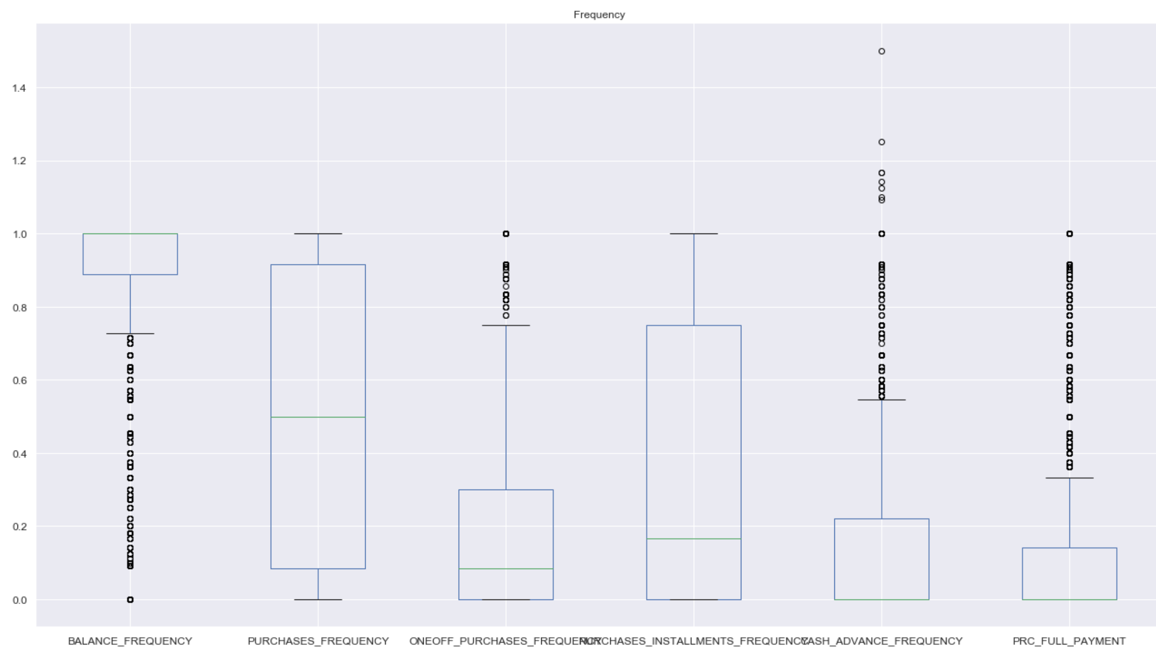
In [18]:

```
credit.hist(figsize=(18,18));
```



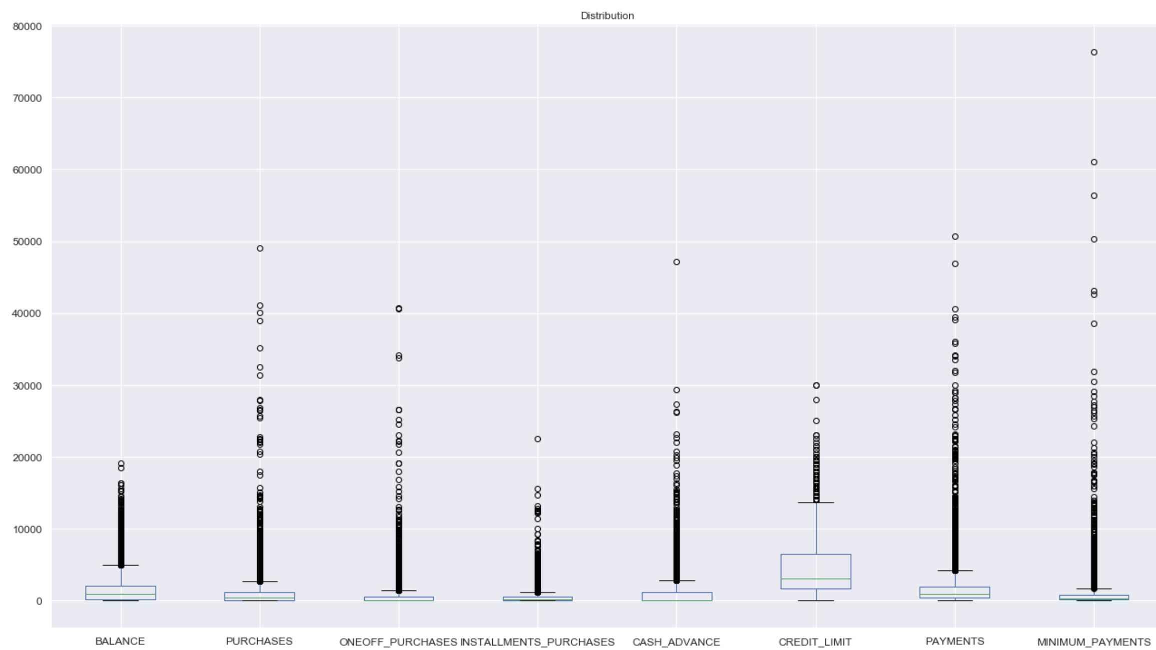
In [19]:

```
#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=T
plt.tight_layout()
# We have data on Cash_advance_frequency that is wrong. I will clean the
# There are also many outliers(the black dots), but I will keep then for
```

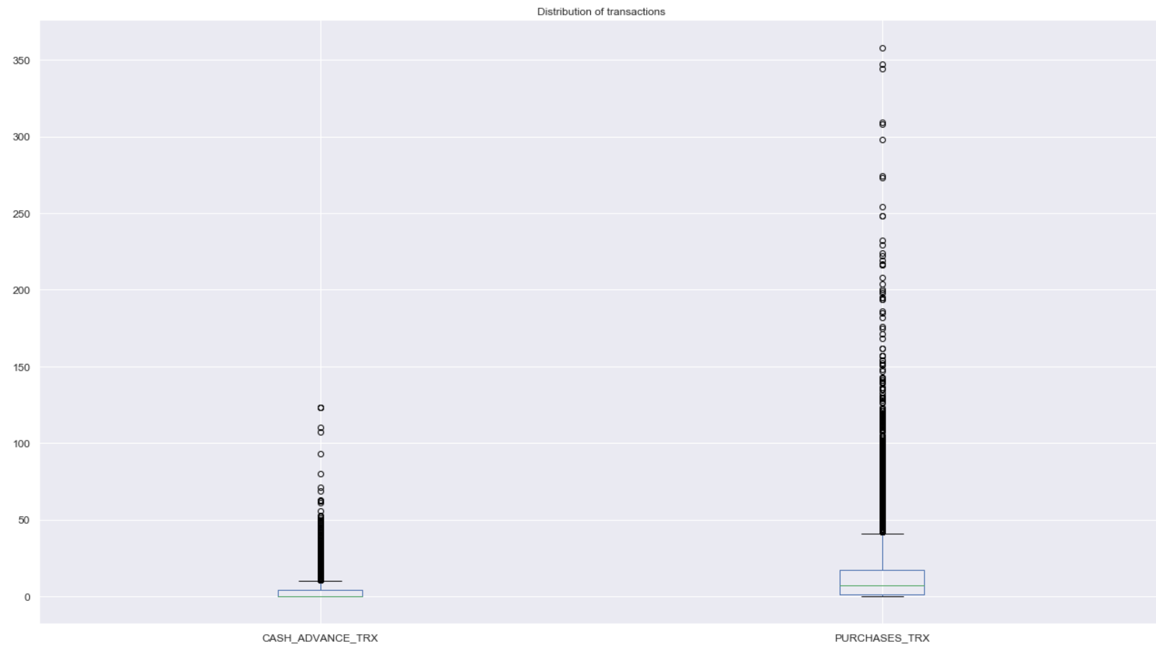


In [20]:

```
#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()

# There are also many outliers(the black dots), but I will keep them for
```



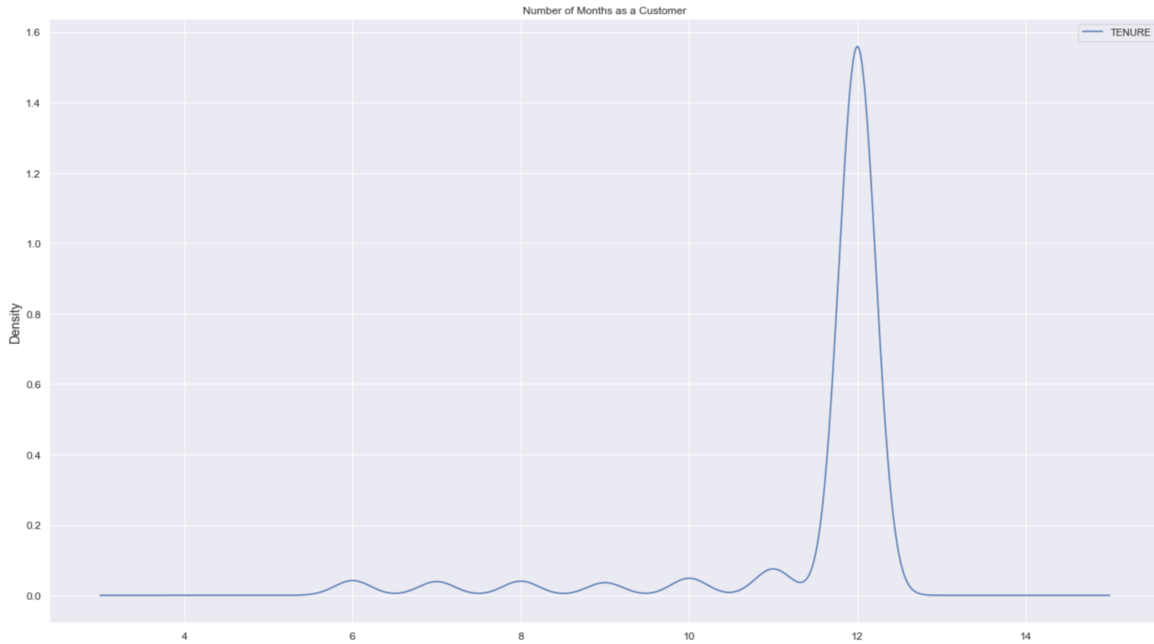
In [21]:
```
#let´s see how are distributed the numeric variables of transactions

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

# There are also many outliers(the black dots), but I will keep them for
```



As I can see, There are many outliers. But, I can't simply drop the outliers as they may contain useful information. So, I'll treat them as extreme values

In [22]:

```
#let´s see how is distributed the tenure

credit[['TENURE']].plot.kde(figsize=(18,10),title='Number of Months as a
plt.tight_layout()

# it shows that most of the distribution of TENURE is 12 months as a cus
```



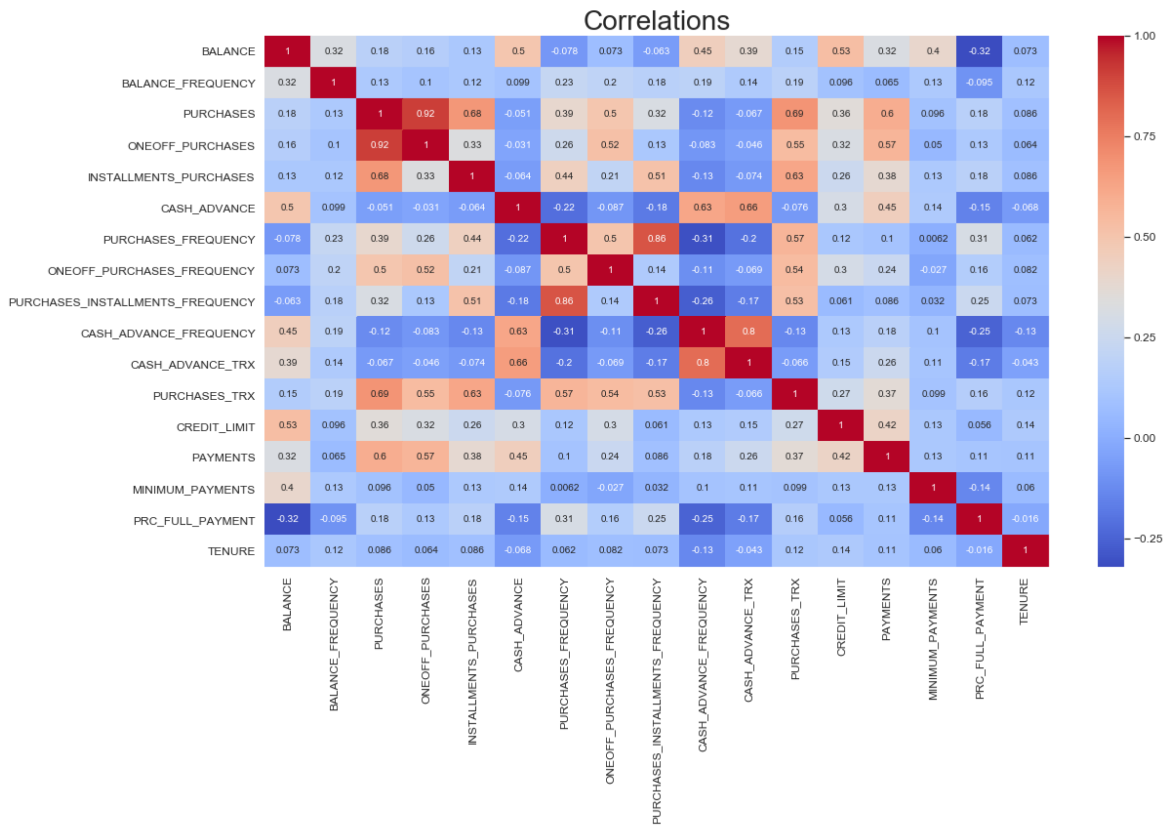
Correlations

- **The dataframe.corr() method will actually get rid of the columns that are not suited for correlation. If we wanted less categories in the heat map, we should select only those categories.**
- **The dataframe.corr() is used to find the pairwise correlation of all columns in the dataframe. Any NA values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored.**
- **By default the method I am going to use in credit.corr() is "Pearson".**

In [23]:

```python
#Lets take a look at how the  variables are correlated
plt.figure(figsize=(18,10))
sns.heatmap(credit.corr(),cmap='coolwarm',annot=True)
plt.title('Correlations', size = 28)

# annot = True will give the correlation matrix, cmap is the color the m
# attomatically adds color bar.
```

Out[23]: Text(0.5, 1.0, 'Correlations')



- **Great! Red means positive, Blue means negative. The stronger the color, the larger the correlation magnitude.**
- **The Blues are the negative correlation, the darker the blue the stronger the correlation. Similar to Red that are positive correlated, the darker the Red the stronger the correlation.**
- **Then we have the middle section where the colors are real light white or almost gray where there's much not correlation at all.**
- **The Reds are the positive correlation, the darker the red the stronger the correlation.**
- **So for Example if we want to know the correlation between PURCHASES and ONEOFFF_PURCHASES I can see that there is very strong positive(dark Red) correlation of 0.92. That means when we increases/decreases the PURCHASES,**

ONEOFF_PURCHASES is also increases/decreases that is directly proportional to each other.

- **I can see that diagonally that there is complete correlation which really doesn't tell much for any of the elements that are the same so for BALANCE and BALANCE we will get 1.**
- **if we want to know the correlation between CASH_ADVANCE_FREQUENCY and BALANCE I can see that there is slightly positive(light Red) correlation of 0.45.**
- **if we want to know the correlation between BALANCE and PRC_FULL_PAYMENT I can see that there is very strong negatively(dark Blue) correlation of -0.32. That means when we increases/decreases the BALANCE, PRC_FULL_PAYMENT is also decreases/increases that is inversly proportional to each other.**

Data Cleaning

In [24]:
```
# here I am cleaning the CASH_ADVANCE_FREQUENCY because some of the data
# which is not valid in frequency.
# Lets clean the data (inputing values and eliminating wrong data) befor
credit.loc[(credit['CASH_ADVANCE_FREQUENCY']>1)]

# we have 8 records for which the frequency is higher that 1. I will eli
```

Out[24]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMEN
681	5656.069801	1.000000	362.36	362.36	
1626	2876.009336	1.000000	152.61	152.61	
2555	5906.184924	1.000000	141.80	141.80	
2608	7801.511533	1.000000	231.40	231.40	
3038	3846.742530	1.000000	0.00	0.00	
3253	5709.486507	0.833333	0.00	0.00	
8055	1917.895730	1.000000	285.07	285.07	
8365	3857.562230	1.000000	0.00	0.00	

In [25]:
```
# dropping the records with frequency higher that 1
credit = credit[(credit[['CASH_ADVANCE_FREQUENCY']] <= 1).all(axis=1)]
```

In [26]:
```
credit.shape
```

Out[26]: (8942, 17)

Scaling the data

We scale the data because it helps to normalise the data within a particular range and every feature transforms to a common scale.

In [27]:
```
from scipy.stats import zscore
```

- **Z-score of the input data, relative to the sample mean and standard deviation.**
- **It allows us to calculate the probability of a score occurring within our normal distribution and enables us to compare two scores that are from different normal distributions.**
- **A Z-score is the number of standard deviations from the mean a data point is.**
- **A Z-score is also known as a standard score and it can be placed on a normal distribution curve.**
- **The Z-score is a test of statistical significance that helps you decide whether or not to reject the null hypothesis. The p-value is the probability that you have falsely rejected the null hypothesis.**
- **Z-scores are measures of standard deviation.**

In [28]:
```
data_scaled=credit.apply(zscore)
data_scaled.head()
```

Out[28]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PI
0	-0.731298	-0.248965	-0.425100	-0.357027	
1	0.789014	0.134664	-0.469735	-0.357027	
2	0.448884	0.518292	-0.107987	0.108603	
3	0.050491	-1.016222	0.231613	0.545724	
4	-0.357750	0.518292	-0.462249	-0.347391	

- **Firstly I have accepted the zscore feature scaling the data to start from simple. But after visualysing the graph after implementation PCA for clustering which i have discuss later. The cluster I have got from the same region which is very mixed up, It is very difficult to visualize and analyse the cluster what each cluster means to say that.**
- **Thats why I have dropped the zscore method and I have accepted the standarization method in further section.**

According to the questions that is expected from me.

1.Deriving New KPI

1A. Monthly_avg_purchase and Cash Advance Amount

In [29]:
```
# Monthly_avg_purchas
credit['Monthly_avg_purchase']=credit['PURCHASES']/credit['TENURE']

# Monthly_cash_advance Amount
credit['Monthly_cash_advance']=credit['CASH_ADVANCE']/credit['TENURE']
```

In [30]:
```
credit['Monthly_avg_purchase'].head()
```

Out[30]:
```
0      7.950000
1      0.000000
2     64.430833
3    124.916667
4      1.333333
Name: Monthly_avg_purchase, dtype: float64
```

In [31]:
```
credit['TENURE'].head()
```

Out[31]:
```
0    12
1    12
2    12
3    12
4    12
Name: TENURE, dtype: int64
```

In [32]:
```
credit['PURCHASES'].head()
```

Out[32]:
```
0      95.40
1       0.00
2     773.17
3    1499.00
4      16.00
Name: PURCHASES, dtype: float64
```

In [33]:
```
credit['Monthly_cash_advance'].head()
```

Out[33]:
```
0      0.000000
1    536.912124
2      0.000000
3     17.149001
4      0.000000
Name: Monthly_cash_advance, dtype: float64
```

In [34]:
```
credit[credit['ONEOFF_PURCHASES']==0]['ONEOFF_PURCHASES'].count()
```

Out[34]:
```
4299
```

1B. Purchase_type

To find what type of purchases customers are making on credit card,lets explore the data.

In [35]:
```
credit.loc[:,['ONEOFF_PURCHASES','INSTALLMENTS_PURCHASES']].head(20)
```

Out[35]:

	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES
0	0.00	95.40
1	0.00	0.00
2	773.17	0.00
3	1499.00	0.00
4	16.00	0.00
5	0.00	1333.28
6	6402.63	688.38
7	0.00	436.20
8	661.49	200.00
9	1281.60	0.00
10	0.00	920.12
11	1492.18	0.00
12	2500.23	717.76
13	419.96	1717.97
14	0.00	0.00
15	0.00	1611.70
16	0.00	0.00
17	0.00	519.00
18	166.00	338.35
19	0.00	398.64

In [36]:
```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES
```

Out[36]: (2039, 19)

In [37]:
```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES'
```

Out[37]: (2774, 19)

In [38]:
```
credit[(credit['ONEOFF_PURCHASES']>0) & (credit['INSTALLMENTS_PURCHASES'
```

Out[38]: (1869, 19)

In [39]:
```
credit[(credit['ONEOFF_PURCHASES']==0) & (credit['INSTALLMENTS_PURCHASES
```

Out[39]: (2260, 19)

In [40]:

```
credit.columns
```

Out[40]:

```
Index(['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES',
       'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FREQUENCY',
       'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY',
       'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX',
       'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT',
       'TENURE', 'Monthly_avg_purchase', 'Monthly_cash_advance'],
      dtype='object')
```

As per above detail I found out that there are 4 types of purchase behaviour in the data set. So I need to derive a categorical variable based on their behaviour¶

In [41]:

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

In [42]:

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

In [43]:

```
credit['purchase_type'].value_counts()
```

Out[43]:

```
both_oneoff_installment    2774
installment                2260
none                       2039
one_off                    1869
Name: purchase_type, dtype: int64
```

I found out that there are 4 types of purchase behaviour in the data set.

```
1.People who only do One-Off Purchases.
2.People who only do Installments Purchases.
3.People who do both.
4.People who do none.
```

1C.Limit_Usage (balance to credit limit ratio)

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

In [44]:
```
credit['limit_usage']=credit.apply(lambda x: x['BALANCE']/x['CREDIT_LIMI
```

In [45]:
```
credit['limit_usage'].head()
```

Out[45]:
```
0    0.040901
1    0.457495
2    0.332687
3    0.222223
4    0.681429
Name: limit_usage, dtype: float64
```

1D.Payment to minimum payments Ratio

In [46]:
```
credit['payment_minpay']=credit.apply(lambda x:x['PAYMENTS']/x['MINIMUM_
credit['payment_minpay'].describe()
```

Out[46]:
```
count    8942.000000
mean        9.065530
std       118.233156
min         0.000000
25%         0.913716
50%         2.036371
75%         6.056885
max      6840.528861
Name: payment_minpay, dtype: float64
```

In [47]:

```
credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8942 entries, 0 to 8949
Data columns (total 22 columns):
BALANCE                             8942 non-null float64
BALANCE_FREQUENCY                   8942 non-null float64
PURCHASES                           8942 non-null float64
ONEOFF_PURCHASES                    8942 non-null float64
INSTALLMENTS_PURCHASES              8942 non-null float64
CASH_ADVANCE                        8942 non-null float64
PURCHASES_FREQUENCY                 8942 non-null float64
ONEOFF_PURCHASES_FREQUENCY          8942 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY    8942 non-null float64
CASH_ADVANCE_FREQUENCY              8942 non-null float64
CASH_ADVANCE_TRX                    8942 non-null int64
PURCHASES_TRX                       8942 non-null int64
CREDIT_LIMIT                        8942 non-null float64
PAYMENTS                            8942 non-null float64
MINIMUM_PAYMENTS                    8942 non-null float64
PRC_FULL_PAYMENT                    8942 non-null float64
TENURE                              8942 non-null int64
Monthly_avg_purchase                8942 non-null float64
Monthly_cash_advance                8942 non-null float64
purchase_type                       8942 non-null object
limit_usage                         8942 non-null float64
payment_minpay                      8942 non-null float64
dtypes: float64(18), int64(3), object(1)
memory usage: 1.6+ MB
```

Extreme value Treatment

Since there are variables having extreme values, I am doing log-transformation on the dataset to remove outlier effect.

I am also going to drop the purchase_type column as I won’t need this for training.

In [48]:

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

In [49]:
```
cr_log.describe()
```

Out[49]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMEI
count	8942.000000	8942.000000	8942.000000	8942.000000	
mean	6.159660	0.619884	4.901012	3.204122	
std	2.013077	0.148642	2.916760	3.246861	
min	0.000000	0.000000	0.000000	0.000000	
25%	4.860106	0.635989	3.708866	0.000000	
50%	6.771280	0.693147	5.895243	3.663562	
75%	7.624446	0.693147	7.013866	6.362183	
max	9.854515	0.693147	10.800403	10.615512	

In [50]:
```
col=['BALANCE','PURCHASES','CASH_ADVANCE','TENURE','PAYMENTS','MINIMUM_P
cr_pre=cr_log[[x for x in cr_log.columns if x not in col ]]
```

In [51]:
```
cr_pre.columns
```

Out[51]:
```
Index(['BALANCE_FREQUENCY', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'Monthly_avg_purchase',
       'Monthly_cash_advance', 'limit_usage', 'payment_minpay'],
      dtype='object')
```

In [52]:
```
cr_log.columns
```

Out[52]:
```
Index(['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES',
       'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FREQUENCY',
       'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY',
       'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX',
       'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT',
       'TENURE', 'Monthly_avg_purchase', 'Monthly_cash_advance', 'limit_usa
ge',
       'payment_minpay'],
      dtype='object')
```

2.Insights from new KPI's

Average payment_minpayment ratio for each purchse type.

In [53]:

```
x=credit.groupby('purchase_type').apply(lambda x: np.mean(x['payment_min
print(type(x))

# this will give the value of mean payment_minpay for each purchase_type
x.values
```

```
<class 'pandas.core.series.Series'>
```

Out[53]:

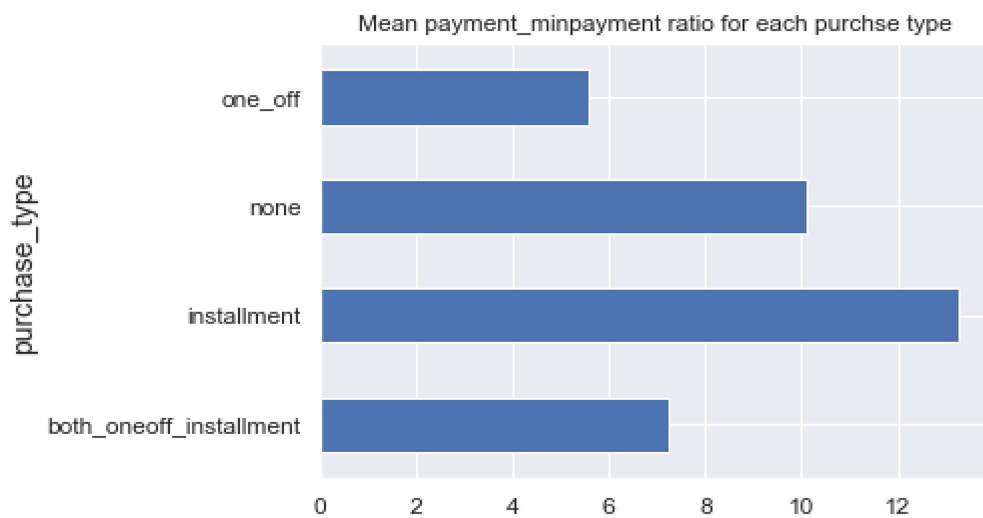
```
array([ 7.23698216, 13.2590037 , 10.10107046,  5.57900144])
```

In [54]:

```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['payment_minpa
plt.title('Mean payment_minpayment ratio for each purchse type')
```

Out[54]:

```
Text(0.5, 1.0, 'Mean payment_minpayment ratio for each purchse type')
```



- **The graph will show with each purchase type among 4 types as shown in the graph what is the mean of payment_minpayment done by customer.**
- **From the graph we can visualize that maximum min payment done by the customer is installment and minimum is done by the one_off customer.**

In [55]:

```
credit.describe()
```

Out[55]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLME
count	8942.000000	8942.000000	8942.000000	8942.000000	
mean	1561.672808	0.877180	1003.971150	592.836192	
std	2079.666731	0.236985	2137.433159	1660.572134	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.037855	0.888889	39.807500	0.000000	
50%	871.427704	1.000000	362.305000	38.000000	
75%	2046.646301	1.000000	1110.945000	578.510000	
max	19043.138560	1.000000	49039.570000	40761.250000	

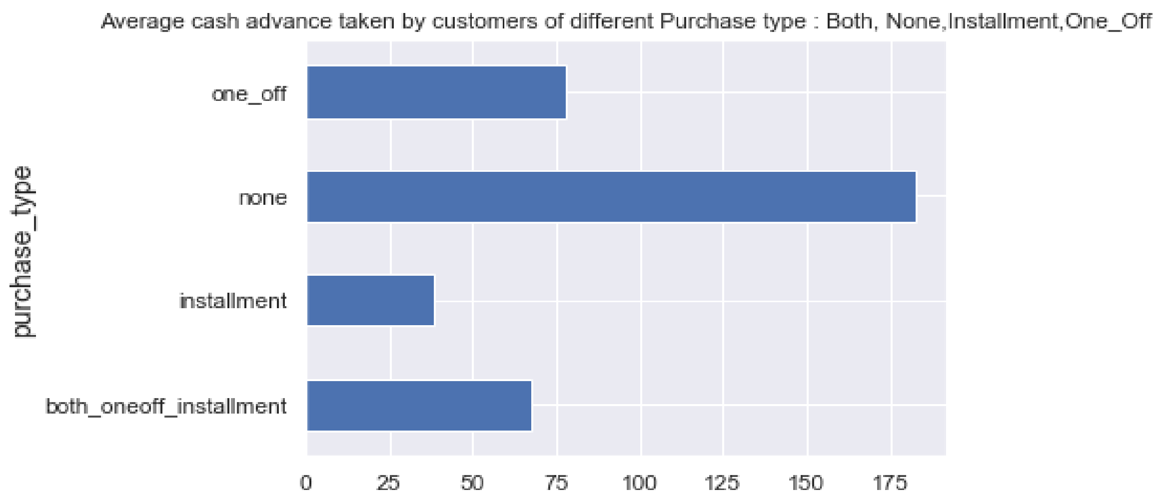
Insight 1: Customers With Installment Purchases are Paying Dues

In [56]:
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['Monthly_cash_
```

Out[56]:
```
array([ 67.82198527,  38.3982058 , 182.59180804,  77.76507082])
```

In [57]:
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['Monthly_cash_

plt.title('Average cash advance taken by customers of different Purchase
```

Out[57]:
```
Text(0.5, 1.0, 'Average cash advance taken by customers of different Purcha
se type : Both, None,Installment,One_Off')
```



- **The graph will show with each purchase type among 4 types as shown in the graph with the Average cash advance taken by customers of different Purchase type : Both, None,Installment,One_Off.**
- **From the graph we can visualize that maximum Average cash advance taken by customers is neither installment nor one_off and minimum is done by the installment customer.**

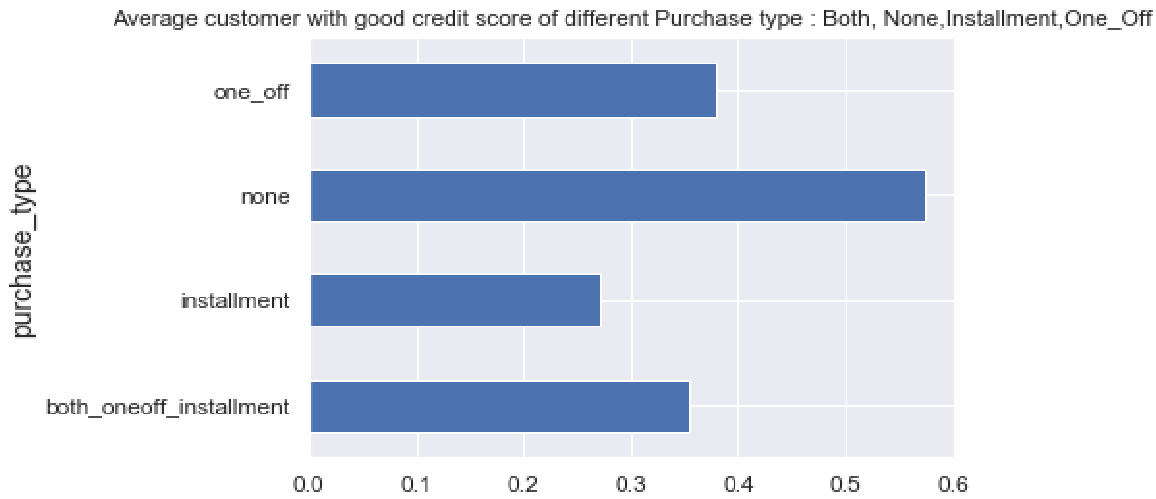
Insight 2: Customers with installment purchases have good credit score.

In [58]:
```
# Lower value implies cutomers are maintaing thier balance properly. Low
credit.groupby('purchase_type').apply(lambda x: np.mean(x['limit_usage']
```

Out[58]:
```
array([0.3535485 , 0.2716782 , 0.57343304, 0.38036302])
```

In [59]:
```
credit.groupby('purchase_type').apply(lambda x: np.mean(x['limit_usage']
plt.title('Average customer with good credit score of different Purchase
```

Out[59]:
```
Text(0.5, 1.0, 'Average customer with good credit score of different Purcha
se type : Both, None,Installment,One_Off')
```



- **The graph will show with each purchase type among 4 types as shown in the graph with the Average customer with good credit score of different Purchase type : Both, None,Installment,One_Off.**
- **From the graph we can visualize that Customers with installment purchases have good credit score. Because Lower value implies cutomers are maintaing thier balance properly. Lower value means good credit score**

Original dataset with categorical column converted to number type.

In [60]:
```
cre_original=pd.concat([credit,pd.get_dummies(credit['purchase_type'])],
```

In [61]:
```
cre_original.head()
```

Out[61]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	

3.Preparing for Machine learning

We do have some categorical data which need to convert with the help of dummy creation

In [62]:
```
# creating Dummies for categorical variable
cr_pre['purchase_type']=credit.loc[:,'purchase_type']
pd.get_dummies(cr_pre['purchase_type']).head()
```

Out[62]:

	both_oneoff_installment	installment	none	one_off
0	0	1	0	0
1	0	0	1	0
2	0	0	0	1
3	0	0	0	1
4	0	0	0	1

Now merge the created dummy with the original data frame cr_dummy

In [63]:
```
cr_dummy=pd.concat([cr_pre,pd.get_dummies(cr_pre['purchase_type'])],axis
```

In [64]:
```
cr_dummy.head()
```

Out[64]:

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

In [65]:

```
cr_dummy.shape
```

Out[65]:

```
(8942, 18)
```

In [66]:

```
l=['purchase_type']
l
```

Out[66]:

```
['purchase_type']
```

Drop the categorical purchase_type

In [67]:

```
cr_dummy=cr_dummy.drop(l,axis=1)
cr_dummy.isnull().sum()
```

Out[67]:

```
BALANCE_FREQUENCY                   0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
Monthly_avg_purchase                0
Monthly_cash_advance                0
limit_usage                         0
payment_minpay                      0
both_oneoff_installment             0
installment                         0
none                                0
one_off                             0
dtype: int64
```

In [68]:

```
cr_dummy.describe()
```

Out[68]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHAS
count	8942.000000	8942.000000	8942.000000	
mean	0.619884	3.204122	3.355403	
std	0.148642	3.246861	3.082720	
min	0.000000	0.000000	0.000000	
25%	0.635989	0.000000	0.000000	
50%	0.693147	3.663562	4.505515	
75%	0.693147	6.362183	6.152956	
max	0.693147	10.615512	10.021315	

In [69]:

```
cr_dummy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8942 entries, 0 to 8949
Data columns (total 17 columns):
BALANCE_FREQUENCY                     8942 non-null float64
ONEOFF_PURCHASES                      8942 non-null float64
INSTALLMENTS_PURCHASES                8942 non-null float64
PURCHASES_FREQUENCY                   8942 non-null float64
ONEOFF_PURCHASES_FREQUENCY            8942 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY      8942 non-null float64
CASH_ADVANCE_FREQUENCY                8942 non-null float64
CASH_ADVANCE_TRX                      8942 non-null float64
PURCHASES_TRX                         8942 non-null float64
Monthly_avg_purchase                  8942 non-null float64
Monthly_cash_advance                  8942 non-null float64
limit_usage                           8942 non-null float64
payment_minpay                        8942 non-null float64
both_oneoff_installment               8942 non-null uint8
installment                           8942 non-null uint8
none                                  8942 non-null uint8
one_off                               8942 non-null uint8
dtypes: float64(13), uint8(4)
memory usage: 1013.0 KB
```

In [70]:

```
cr_dummy.head()
```

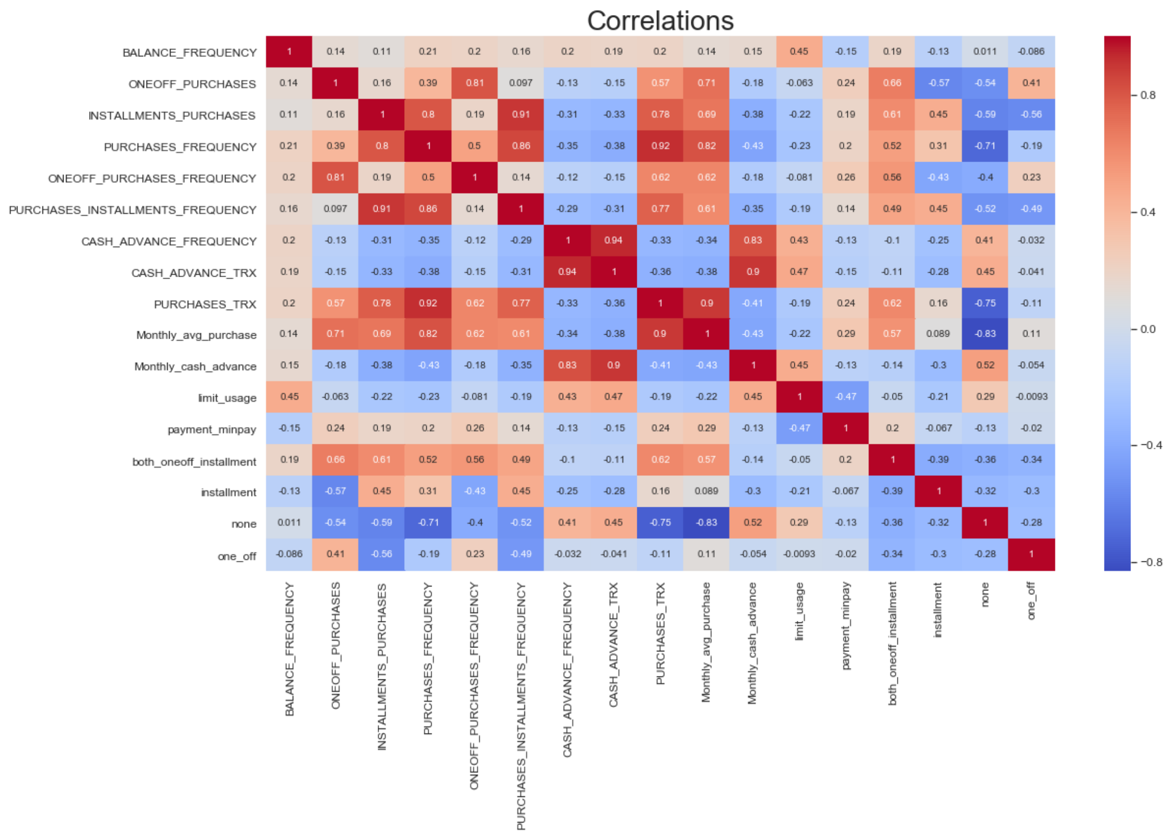
Out[70]:

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

In [71]:

```
#Lets take a look at how the  variables are correlated for the dataframe
plt.figure(figsize=(18,10))
sns.heatmap(cr_dummy.corr(),cmap='coolwarm',annot=True)
plt.title('Correlations', size = 28)

# annot = True will give the correlation matrix, cmap is the color the m
# attomatically adds color bar.
```

Out[71]: `Text(0.5, 1.0, 'Correlations')`



- **Great! Red means positive, Blue means negative. The stronger the color, the larger the correlation magnitude.**

- **The Blues are the negative correlation, the darker the blue the stronger the correlation. Similar to Red that are positive correlated, the darker the Red the stronger the correlation.**
- **Then we have the middle section where the colors are real light white or almost gray where there's much not correlation at all.**
- **The Reds are the positive correlation, the darker the red the stronger the correlation.**
- **So for Example if we want to know the correlation between PURCHASES_FREQUENCY and PURCHASES_TRX I can see that there is very strong positive(dark Red) correlation of 0.92. That means when we increases/decreases the PURCHASES_FREQUENCY, PURCHASES_TRX is also increases/decreases that is directly proportional to each other.**
- **I can see that diagonally that there is complete correlation which really doesn't tell much for any of the elements that are the same so for BALANCE_FREQUENCY and BALANCE_FREQUENCY we will get 1.**
- **if we want to know the correlation between BALANCE_FREQUENCY and LIMIT_USAGE I can see that there is slightly positive(light Red) correlation of 0.45.**
- **if we want to know the correlation between MONTHLY_AVG_PURCHASES and NONE I can see that there is very strong negatively(dark Blue) correlation of -0.83. That means when we increases/decreases the MONTHLY_AVG_PURCHASES, NONE is also decreases/increases that is inversly proportional to each other.**

In [72]:
```
cr_dummy.shape
```
Out[72]:
```
(8942, 17)
```

Standardrizing data

Before applying PCA we will standardize data to avoid effect of scale on our result. Centering and Scaling will make all features with equal weight.

To put data on the same scale

In [73]:
```
from sklearn.preprocessing import  StandardScaler
sc=StandardScaler()
```

In [74]:
```
credit.columns
```
Out[74]:
```
Index(['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES',
       'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FREQUENCY',
       'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENCY',
       'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX',
       'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT',
       'TENURE', 'Monthly_avg_purchase', 'Monthly_cash_advance',
       'purchase_type', 'limit_usage', 'payment_minpay'],
      dtype='object')
```

In [75]:
```
credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8942 entries, 0 to 8949
Data columns (total 22 columns):
BALANCE                             8942 non-null float64
BALANCE_FREQUENCY                   8942 non-null float64
PURCHASES                           8942 non-null float64
ONEOFF_PURCHASES                    8942 non-null float64
INSTALLMENTS_PURCHASES              8942 non-null float64
CASH_ADVANCE                        8942 non-null float64
PURCHASES_FREQUENCY                 8942 non-null float64
ONEOFF_PURCHASES_FREQUENCY          8942 non-null float64
PURCHASES_INSTALLMENTS_FREQUENCY    8942 non-null float64
CASH_ADVANCE_FREQUENCY              8942 non-null float64
CASH_ADVANCE_TRX                    8942 non-null int64
PURCHASES_TRX                       8942 non-null int64
CREDIT_LIMIT                        8942 non-null float64
PAYMENTS                            8942 non-null float64
MINIMUM_PAYMENTS                    8942 non-null float64
PRC_FULL_PAYMENT                    8942 non-null float64
TENURE                              8942 non-null int64
Monthly_avg_purchase                8942 non-null float64
Monthly_cash_advance                8942 non-null float64
purchase_type                       8942 non-null object
limit_usage                         8942 non-null float64
payment_minpay                      8942 non-null float64
dtypes: float64(18), int64(3), object(1)
memory usage: 1.6+ MB
```

- **Before using K-Means, as in K-means we optimize the sum of squared distances between the observations and their centroids.**
- **standardization is the process of putting different variables on the same scale. This process allows to compare scores between different types of variables.**
- **To standardize variables, we can calculate the mean and standard deviation for a variable. Then, for each observed value of the variable, we can subtract the mean and divide by the standard deviation.**
- **And as some varibles are expresed in different variables i.e frequencies, currency amount and number of transactions, we need to standardize.**
- **I would like to explore the standardizing the data will give us better results.**
- **Then, I would follow the analysis with cr_scaled.**

In [76]:
```
cr_dummy.shape
```

Out[76]: (8942, 17)

In [77]:
```
# Fit on the data and transform
cr_scaled=sc.fit_transform(cr_dummy)
```

In [78]:
```
cr_scaled.shape
```
Out[78]: (8942, 17)

Applying PCA

With the help of principal component analysis we will reduce features.

PCA transforms a large set of variables into a smaller one that still contains most of the information in the large set.Reducing the number of variables of a data.

The main idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent.

The dataset on which PCA technique is to be used must be scaled (I have used as cr_scaled from standardization). The results are also sensitive to the relative scaling. As a layman, it is a method of summarizing data.

In [79]:
```
# importing PCA from sklearn decompition library class
from sklearn.decomposition import PCA
```

In [80]:
```
cr_dummy.shape
```
Out[80]: (8942, 17)

In [81]:
```
#We have 17 features so our n_component will be 17.
pc=PCA(n_components=17)
cr_pca=pc.fit(cr_scaled)
```

In [82]:
```
#Lets check if we will take 17 component then how much varience it expla
sum(cr_pca.explained_variance_ratio_)
```
Out[82]: 0.9999999999999999

In [83]:
```
# getting the variance ratio to find out how many components for pca is
# all features while reducing the fetures.
var_ratio={}
for n in range(2,18):
    pc=PCA(n_components=n)
    cr_pca=pc.fit(cr_scaled)
    var_ratio[n]=sum(cr_pca.explained_variance_ratio_)
```

In [84]:
```
var_ratio
```

Out[84]:
```
{2: 0.5825012321824052,
 3: 0.7300732591583463,
 4: 0.8117296085433605,
 5: 0.877135158935572,
 6: 0.9187375718123484,
 7: 0.9411907786782436,
 8: 0.9617042714878689,
 9: 0.974079691142425,
 10: 0.9836323165922601,
 11: 0.9897668518214651,
 12: 0.992785175833388,
 13: 0.9954223365608661,
 14: 0.9979602892281134,
 15: 0.9996358230884447,
 16: 0.9999999999999999,
 17: 0.9999999999999999}
```

Since 6 components are explaining about 90% variance so we select 5 components

In [85]:
```
# selecting pca n_componets = 5 because after reducing to 5 features it
pc=PCA(n_components=5)
```

In [86]:
```
# Fit on the data
p=pc.fit(cr_scaled)
```

In [87]:
```
cr_scaled.shape
```

Out[87]:
```
(8942, 17)
```

In [88]:
```
p.explained_variance_
```

Out[88]:
```
array([6.83427679, 3.0693517 , 2.50900505, 1.3883132 , 1.11201872])
```

In [89]:
```
np.sum(p.explained_variance_)
```

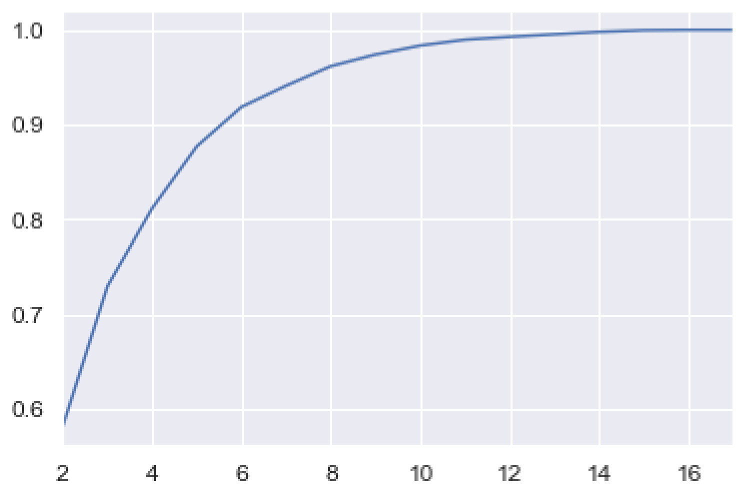
Out[89]:
```
14.912965445747917
```

In [90]:
```
var_ratio
```

Out[90]:
```
{2: 0.5825012321824052,
 3: 0.7300732591583463,
 4: 0.8117296085433605,
 5: 0.8771351589355728,
 6: 0.9187375718123484,
 7: 0.9411907786782436,
 8: 0.9617042714878689,
 9: 0.974079691142425 5,
 10: 0.9836323165922601,
 11: 0.9897668518214651,
 12: 0.992785175833388,
 13: 0.9954223365608661,
 14: 0.9979602892281134,
 15: 0.9996358230884447,
 16: 0.9999999999999999,
 17: 0.9999999999999999}
```

In [91]:
```
# while visualization the var_ratio to find out how many componets are b
# 90% variance.
# Since 6 components are explaining about 90% variance so we select 5 co
pd.Series(var_ratio).plot()
```

Out[91]: <matplotlib.axes._subplots.AxesSubplot at 0x1425e1cd6d8>



Since 5 components are explaining about 87% variance so we select 5 components

In [92]:
```
pc_final=PCA(n_components=5).fit(cr_scaled)

# Fit the data and the transform
reduced_cr=pc_final.fit_transform(cr_scaled)
```

In [93]:
```
dd=pd.DataFrame(reduced_cr)
```

In [94]:
```
dd.shape
```
Out[94]: (8942, 5)

So initially we had 17 variables now its 5 so our variable go reduced

In [95]:
```
dd.head()
```
Out[95]:

	0	1	2	3	4
0	-0.245367	-2.760833	0.333813	-0.413266	-0.007819
1	-3.981189	0.155246	-0.545383	1.018122	-0.431104
2	1.287526	1.495470	2.719668	-1.891480	0.029416
3	-1.048933	0.663479	2.505172	-1.299680	0.775330
4	-1.451981	-0.185896	2.290707	-1.627842	-0.547878

In [96]:
```
# to get the columns name list
col_list=cr_dummy.columns
```
In [97]:
```
col_list
```
Out[97]:
```
Index(['BALANCE_FREQUENCY', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'Monthly_avg_purchase',
       'Monthly_cash_advance', 'limit_usage', 'payment_minpay',
       'both_oneoff_installment', 'installment', 'none', 'one_off'],
      dtype='object')
```

In [98]:

```
# here I got the result of the selected 5 component for pca for each col
pd.DataFrame(pc_final.components_.T, columns=['PC_' +str(i) for i in ran
```

Out[98]:

	PC_0	PC_1	PC_2	PC_3	PC_
BALANCE_FREQUENCY	0.029912	0.240653	-0.261270	-0.355869	-0.22631
ONEOFF_PURCHASES	0.214339	0.405090	0.240616	0.001329	-0.02269
INSTALLMENTS_PURCHASES	0.311937	-0.097229	-0.316182	0.087830	-0.00338
PURCHASES_FREQUENCY	0.345823	0.016117	-0.162857	-0.073990	0.11644
ONEOFF_PURCHASES_FREQUENCY	0.214834	0.361625	0.164541	0.035693	-0.05074
PURCHASES_INSTALLMENTS_FREQUENCY	0.295342	-0.110902	-0.330497	0.023468	0.02511
CASH_ADVANCE_FREQUENCY	-0.214419	0.287109	-0.278658	0.098845	0.35833
CASH_ADVANCE_TRX	-0.229267	0.292582	-0.284250	0.104557	0.33232
PURCHASES_TRX	0.355534	0.106792	-0.102446	-0.053821	0.10555
Monthly_avg_purchase	0.346135	0.141081	0.024335	-0.077999	0.19480
Monthly_cash_advance	-0.243782	0.265330	-0.256535	0.135885	0.26754
limit_usage	-0.146047	0.236370	-0.249422	-0.434088	-0.17860
payment_minpay	0.119413	0.021425	0.135540	0.592875	0.21181
both_oneoff_installment	0.241313	0.274742	-0.130851	0.251973	-0.34280
installment	0.082001	-0.443136	-0.210543	-0.188013	0.35461
none	-0.310503	-0.003924	-0.096654	0.242296	-0.34378
one_off	-0.041784	0.165110	0.473630	-0.335728	0.36572

So above data gave us eigen vector for each component we had all eigen vector value very small we can remove those variable bur in our case its not.

In [99]:

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

Out[99]:

```
PC_0    0.401971
PC_1    0.180530
PC_2    0.147572
PC_3    0.081656
PC_4    0.065406
dtype: float64
```

In [100]:

```
type(cr_pca)
```

Out[100]:

```
sklearn.decomposition.pca.PCA
```

Clustering

To find out how many clusters are going to b used.. I need to tell the K-Means algorithm the number of clusters it should use. There are a number of techniques that can be used to find the optimal number.

For this example, I am going to use the elbow method so named because the chart that it produces is similar in shape to the curve of an elbow. This method computes the sum of squared distances for clusters k.

As more clusters are used the variance will reduce until you reach a point at which increasing clusters no longer results in a better model. T

Based on the intuition on type of purchases made by customers and their distinctive behavior exhibited based on the purchase_type (as visualized above in Insights from KPI) , I am starting with 4 clusters.

In [101]:
```
# import KMeans cluster
from sklearn.cluster import KMeans
km_4=KMeans(n_clusters=4,random_state=123)
```

In [102]:
```
# fit the data
km_4.fit(reduced_cr)
```

Out[102]:
```
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)
```

In [103]:
```
# geting the cluster labels for each data in reduced_cr
labels = km_4.labels_
labels
```

Out[103]:
```
array([1, 0, 3, ..., 1, 0, 3])
```

In [104]:
```
credit.shape
```

Out[104]:
```
(8942, 22)
```

In [105]:
```
# dropping the categorial purchase_type in new dataframe credit_2
credit_2=credit.drop(l,axis=1)
credit_2.shape
```

Out[105]:
```
(8942, 21)
```

In [106]:

```
# concatenate the new dataframe cluster into new data that is clusters
clusters=pd.concat([credit_2, pd.DataFrame({'cluster':labels})], axis=1)
clusters.head()
```

Out[106]:

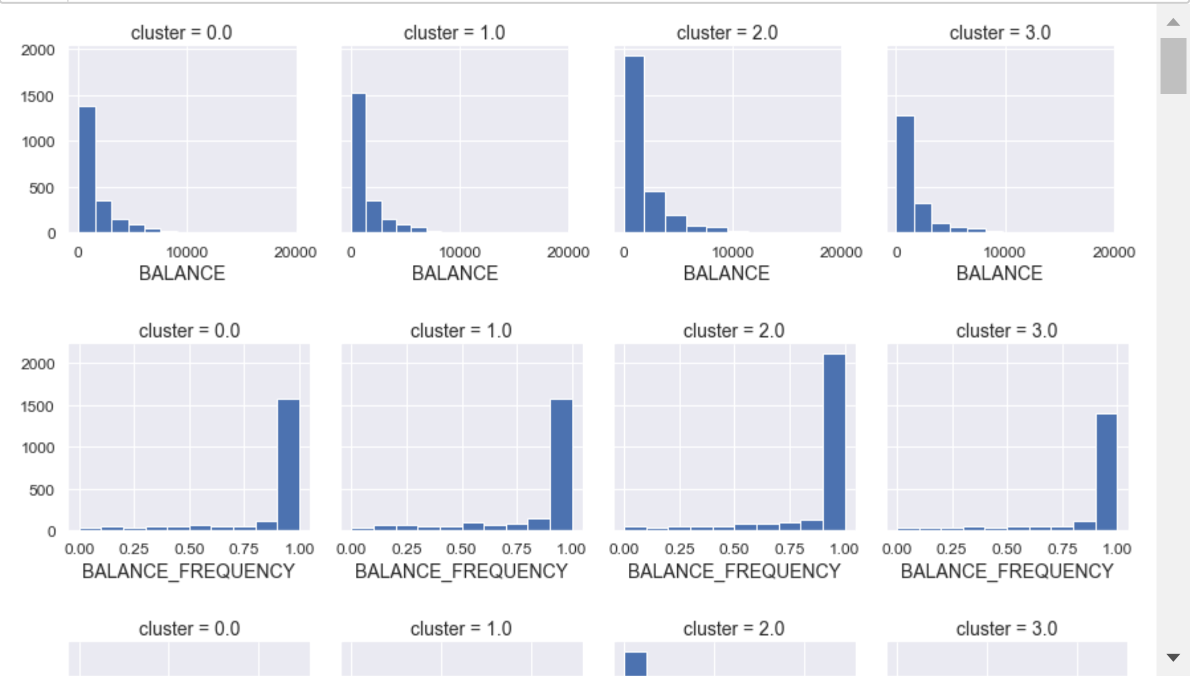
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	

Interpretation of clusters

Cluster analysis is an exploratory analysis that tries to identify structures within the data. Cluster analysis is also called segmentation analysis or taxonomy analysis. More specifically, it tries to identify homogenous groups of cases if the grouping is not previously known.

In [107]:

```
for c in clusters:
    grid= sns.FacetGrid(clusters, col='cluster')
    grid.map(plt.hist, c)
```



It is same like EDA that I have explain in starting in Exploratory Data Analysis. The only difference is that here is I have to analyse with all other 4 clusters.

Analysing the clusters

I am going to use the pandas groupby function to analyse a number of features for the clusters in order to understand if the model has successfully identified unique segments.

In [108]:

```
cluster_pca = clusters.groupby('cluster').mean()
cluster_pca
```

Out[108]:

cluster	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMI
0.0	1558.009898	0.880551	810.042010	456.277564	
1.0	1313.909562	0.858399	828.377163	454.877410	
2.0	1744.587965	0.888570	1299.736257	783.119467	
3.0	1597.091118	0.879153	995.670814	630.068469	

Just looking at 'PURCHASES_FREQUENCY' we can see that the model has identified some high-frequency purchase segments, clusters 1 and 2.
Let's understand the differences between these two segments to further determine why they are in separate clusters.
We can see that cluster 2 has a higher number of total purchases, a higher credit limit, they make frequent one-off purchases and are more likely to pay in full.
We can draw the conclusion that these are high-value customers and therefore there will almost certainly be a difference between how you may market to these customers.

As a first iteration of the model, this appears to be identifying some useful segments. There are many ways in which we could tune the model including alternative data cleaning methods, feature engineering, dropping features with high correlation, which I have already implemented in this project.

Applying PCA

We apply PCA to transform data to 2 dimensions for visualization. We won't be able to visualize the data in 17 dimensions so reducing the dimensions with PCA.

PCA transforms a large set of variables into a smaller one that still contains most of the information in the large set.Reducing the number of variables of a data.

In [109]:

```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(cr_scaled)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component
principalDf.head()
```

Out[109]:

	principal component 1	principal component 2
0	-0.245367	-2.760833
1	-3.981189	0.155246
2	1.287526	1.495470
3	-1.048933	0.663479
4	-1.451981	-0.185896

In [110]:

```
finalDf = pd.concat([principalDf, pd.DataFrame({'cluster':labels})], axi
finalDf.head()
```

Out[110]:

	principal component 1	principal component 2	cluster
0	-0.245367	-2.760833	1
1	-3.981189	0.155246	0
2	1.287526	1.495470	3
3	-1.048933	0.663479	3
4	-1.451981	-0.185896	3

In [111]:

```
plt.figure(figsize=(15,10))
ax = sns.scatterplot(x="principal component 1", y="principal component 2
plt.show()
```

Visualizing the 4 cluster after applying the concept of pca to reduce the feature in two-dimensional to find out clusters behaviour.

In [112]:

```
# Number of clients by cluster
pd.Series(km_4.labels_).value_counts()
```

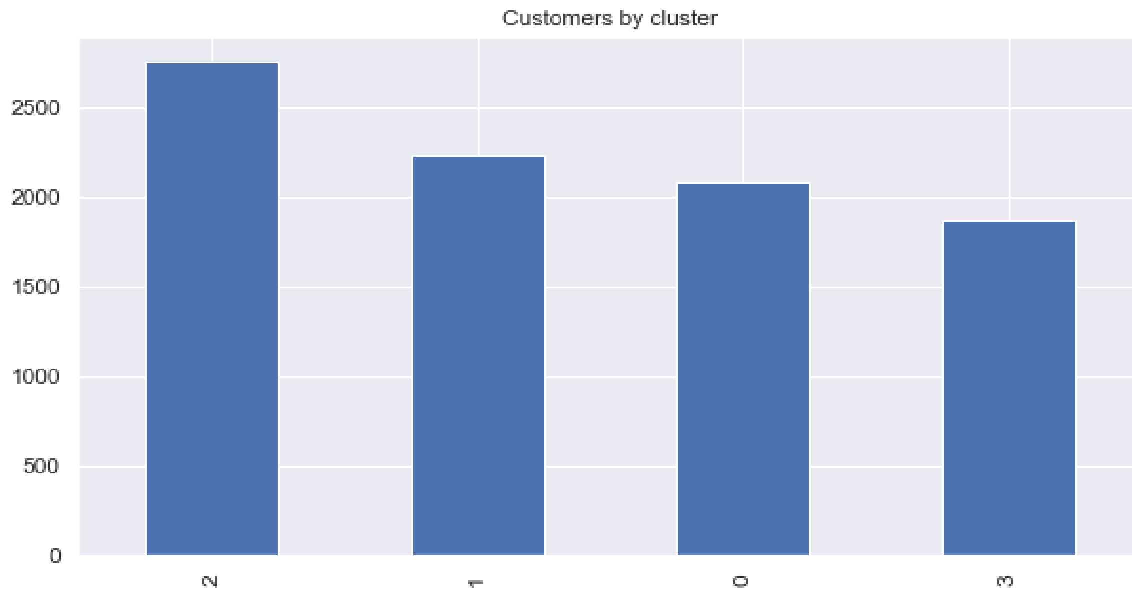
Out[112]:

```
2    2758
1    2228
0    2087
3    1869
dtype: int64
```

In [113]:

```
pd.Series(km_4.labels_).value_counts().plot.bar(figsize=(10,5), title='C

# In this bar graph we will count the number of customer for each cluste
```

Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x1426745b630>



Here we donot have known k value so we will find the K. To do that we need to take a cluster range between 1 and 21.

Identify cluster Error.

In [114]:

```
# This will give an idea to chose the best K value ie number of cluster
cluster_range = range(1,21)
cluster_errors = []

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

In [115]:

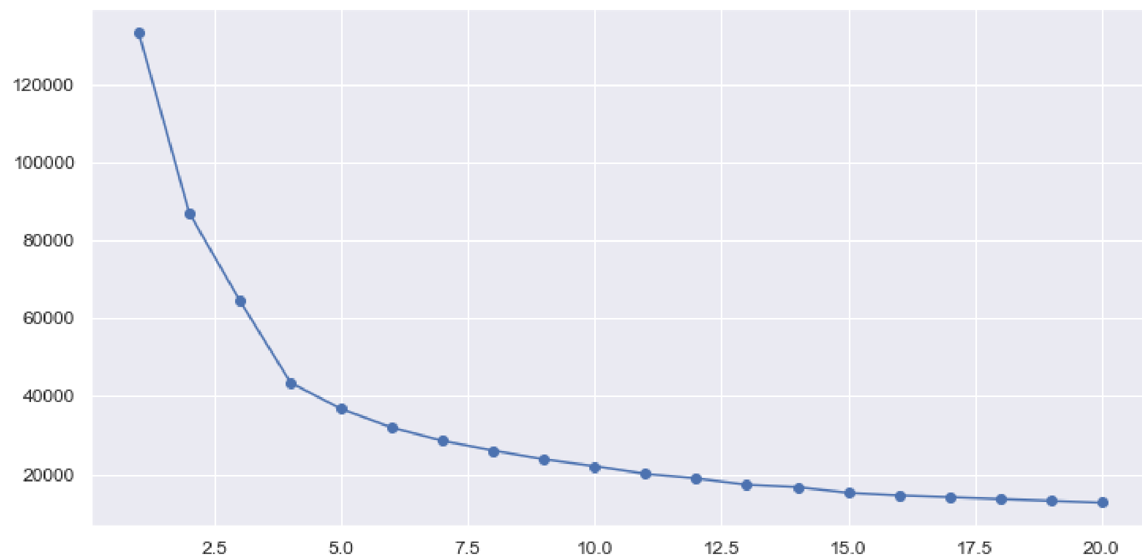
```
clusters_df = pd.DataFrame( { "num_clusters":cluster_range, "cluster_err

clusters_df[0:21]
```

Out[115]:

	num_clusters	cluster_errors
0	1	133336.824050
1	2	86989.482640
2	3	64471.974649
3	4	43473.290249
4	5	36743.185209
5	6	31975.766263
6	7	28593.876282
7	8	26081.683385
8	9	23851.363100
9	10	22064.583914
10	11	20121.290434
11	12	18952.313063
12	13	17338.428696
13	14	16718.898230
14	15	15268.773001
15	16	14587.035443
16	17	14121.563450
17	18	13654.008983
18	19	13197.404720
19	20	12731.346227

In [116]:
```
# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
plt.plot( clusters_df.num_clusters, clusters_df.cluster_errors, marker =
```

Out[116]: [<matplotlib.lines.Line2D at 0x142687a7278>]



From above graph I will find elbow range. here it is 4,5,6. I can see that after almost 4,5,6 clusters adding more gives minimal benefit to the model. I am therefore going to use 5 clusters to train my model.

Silhouette Coefficient

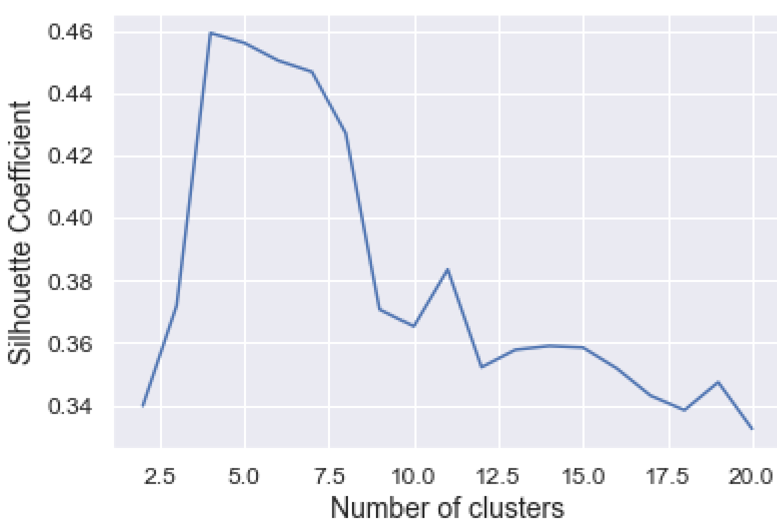
In [117]:
```
from sklearn import metrics
```

In [118]:
```
# calculate SC for K=3 through K=12
k_range = range(2, 21)
scores = []
for k in k_range:
    km = KMeans(n_clusters=k, random_state=1)
    km.fit(reduced_cr)
    scores.append(metrics.silhouette_score(reduced_cr, km.labels_))
```

In [119]:
```
scores
```

Out[119]:
```
[0.33978485655399177,
 0.3719860030766553,
 0.45930430077424117,
 0.4561306957634831,
 0.4504716109032892,
 0.44684165952880067,
 0.42711569565481,
 0.3707144525210207,
 0.3652990995632888,
 0.38357883565022505,
 0.3521787311783152,
 0.3577749909775027,
 0.35902188576772137,
 0.35850738496535867,
 0.35180428609942493,
 0.343053454348012,
 0.3383905933608716,
 0.3473690526740195,
 0.3323960832110805]
```

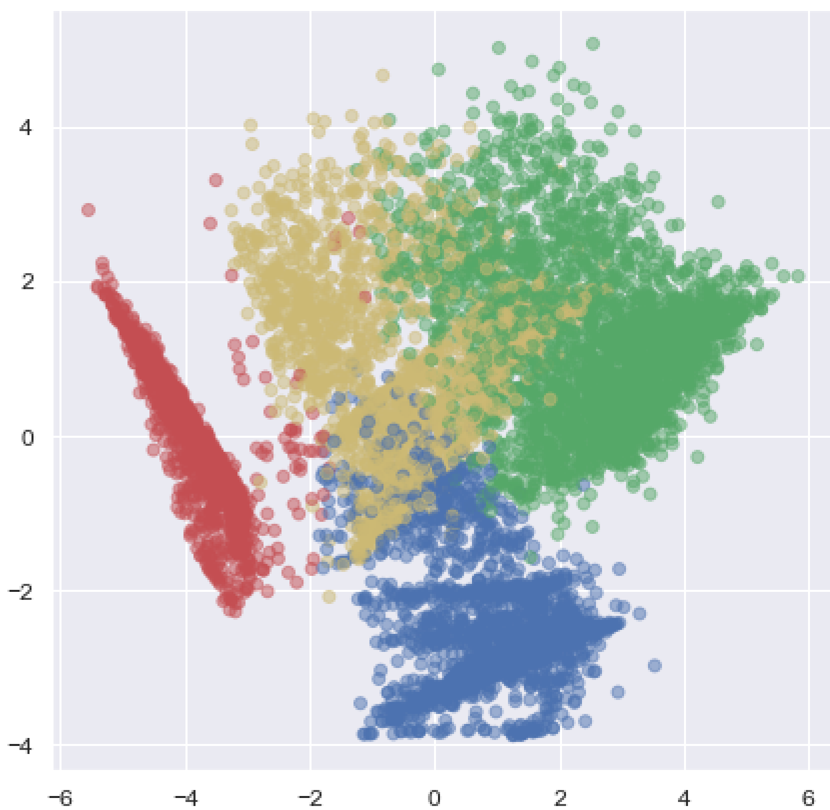
In [120]:
```
# Plot the Result
plt.plot(k_range, scores)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.grid(True)
```



From metrics.silhouette_score method above graph it is suggesting to choose the K value is 4 i.e. number of cluster = 4

In [121]:
```
color_map={0:'r',1:'b',2:'g',3:'y'}
label_color=[color_map[l] for l in km_4.labels_]
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=label_color,cmap='Spectral
```

Out[121]: <matplotlib.collections.PathCollection at 0x14200067f28>



It is very difficult to draw iddividual plot for cluster, so we will use pair plot which will provide us all graph in one shot. To do that we need to take following steps

In [122]:
```
df_pair_plot=pd.DataFrame(reduced_cr,columns=['PC_' +str(i) for i in ran
```

In [123]:
```
df_pair_plot['Cluster']=km_4.labels_ #Add cluster column in the data fra
```

In [124]:
```
df_pair_plot.head()
```

Out[124]:

	PC_0	PC_1	PC_2	PC_3	PC_4	Cluster
0	-0.245367	-2.760833	0.333813	-0.413266	-0.007819	1
1	-3.981189	0.155246	-0.545383	1.018122	-0.431104	0
2	1.287526	1.495470	2.719668	-1.891480	0.029416	3
3	-1.048933	0.663479	2.505172	-1.299680	0.775330	3
4	-1.451981	-0.185896	2.290707	-1.627842	-0.547878	3

In [ ]:
```

```

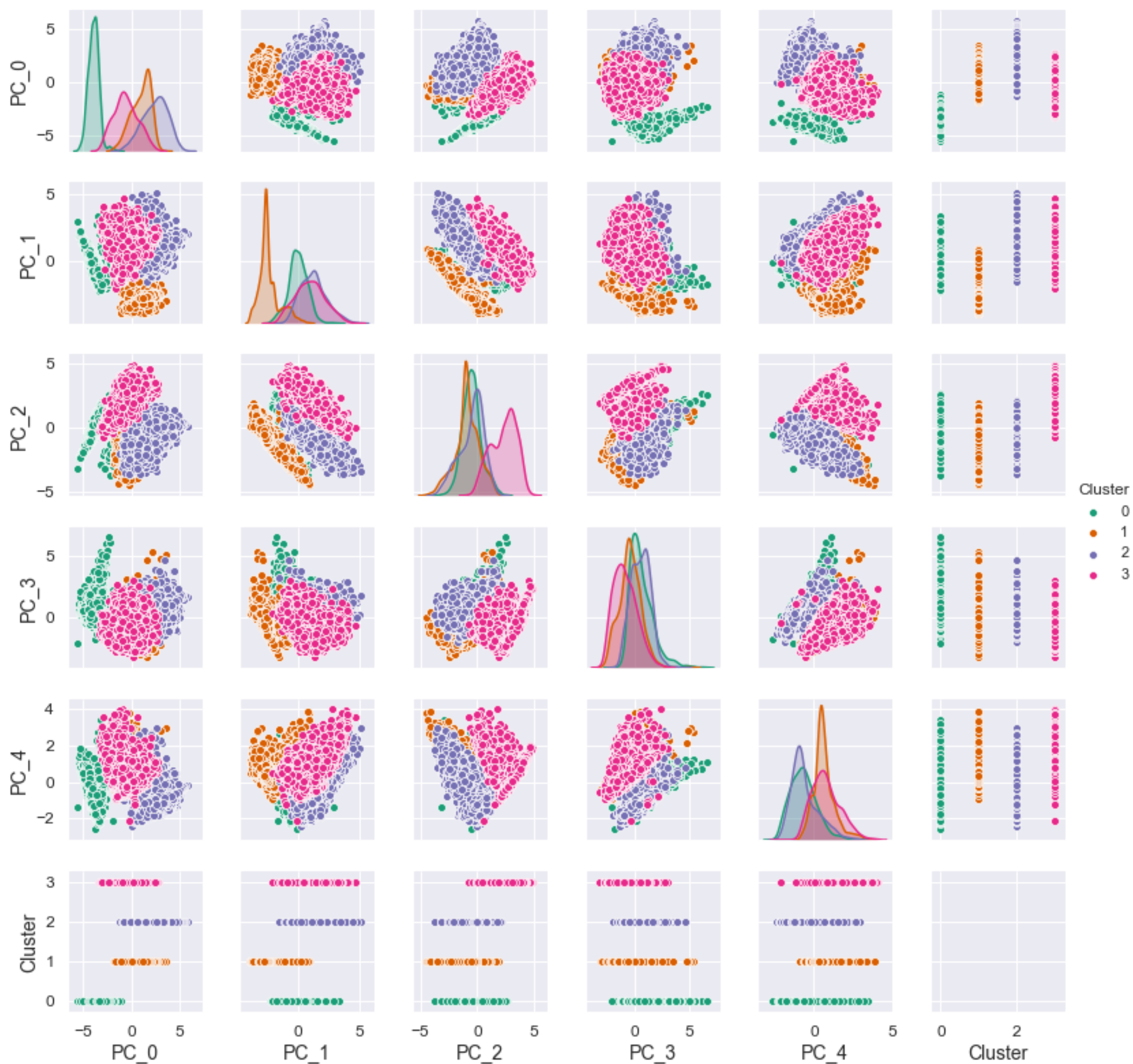
In [125]:
```
cr_dummy.dtypes
```

Out[125]:
```
BALANCE_FREQUENCY                    float64
ONEOFF_PURCHASES                     float64
INSTALLMENTS_PURCHASES               float64
PURCHASES_FREQUENCY                  float64
ONEOFF_PURCHASES_FREQUENCY           float64
PURCHASES_INSTALLMENTS_FREQUENCY     float64
CASH_ADVANCE_FREQUENCY               float64
CASH_ADVANCE_TRX                     float64
PURCHASES_TRX                        float64
Monthly_avg_purchase                 float64
Monthly_cash_advance                 float64
limit_usage                          float64
payment_minpay                       float64
both_oneoff_installment                uint8
installment                            uint8
none                                   uint8
one_off                                uint8
dtype: object
```

Pairwise relationship of components on the data

We can't visulaize the five-dimensional dataset there has to some way to for us to visualize our dataset which has 5 features PC_0, PC_1, PC_2, PC_3 and PC_4.
It's a five-dimensional data and we can't do five-dimensional scatters plot, so there some smart way of visulaizing all of this data at once. so once such intresting way of doing is pairplot.
So to determine how many such pair exits is since I have five variables I want to create pairs of two then i can do permutation and combination just like 5C2 = 10 unique pair above the diagonal and when I visualize the 10 plot I can sense the what the data is in five-dimension.
So instead of visualizing the five-dimension scatter plot which we cannot do so we will be visualize it 10 two dimensional plot to understand what the data is and such a plot is called a pair plot. The other 10 pair below the diagonal is almost the same as the pair plot above the diagonal.

In [126]:
```
1 sns.pairplot(df_pair_plot,hue='Cluster', palette= 'Dark2', diag_kind='kd
```
Out[126]: <seaborn.axisgrid.PairGrid at 0x1420007afd0>



Observation from the above pair plot:

The goal was to segment the customers in order to define a marketing strategy.
It shows that first two components are able to indentify clusters.

- **PC_0 and PC_1 are the most useful features to identify various cluster types.**
- **compare to the other components except PC_0 and PC_1 mostly the green cluster and red cluster are overlap with pink and blue cluster but with PC_1 and PC_2 we are able to see all 4 cluster with difference just slightly overlap.**
- **Here we can find circles and with some condition to build a simple model to classify the cluster types.**

Now we have done here with priciple component now we need to come bring our original data frame and we will merge the cluster with

them.

To interprate result we need to use our data frame

Key performace variable selection . here i am taking varibales which we will use in derving new KPI. We can take all 17 variables but it will be difficult to interprate.So are are selecting less no of variables.

In [127]:

```
# Key performace variable selection . here i am dropping varibales which

col_kpi=['PURCHASES_TRX','Monthly_avg_purchase','Monthly_cash_advance','
         'payment_minpay','both_oneoff_installment','installment','one_o
```

In [128]:

```
cr_pre.describe()
```

Out[128]:

	BALANCE_FREQUENCY	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHAS
count	8942.000000	8942.000000	8942.000000	
mean	0.619884	3.204122	3.355403	
std	0.148642	3.246861	3.082720	
min	0.000000	0.000000	0.000000	
25%	0.635989	0.000000	0.000000	
50%	0.693147	3.663562	4.505515	
75%	0.693147	6.362183	6.152956	
max	0.693147	10.615512	10.021315	

In [129]:

```
# Conactenating labels found through Kmeans with data
cluster_df_4=pd.concat([cre_original[col_kpi],pd.Series(km_4.labels_,nam
```

In [130]:

```
cluster_df_4.head()
```

Out[130]:

	PURCHASES_TRX	Monthly_avg_purchase	Monthly_cash_advance	limit_usage	CASH_ADVA
0	2.0	7.950000	0.000000	0.040901	
1	0.0	0.000000	536.912124	0.457495	
2	12.0	64.430833	0.000000	0.332687	
3	1.0	124.916667	17.149001	0.222223	
4	1.0	1.333333	0.000000	0.681429	

In [131]:

```
# 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
```

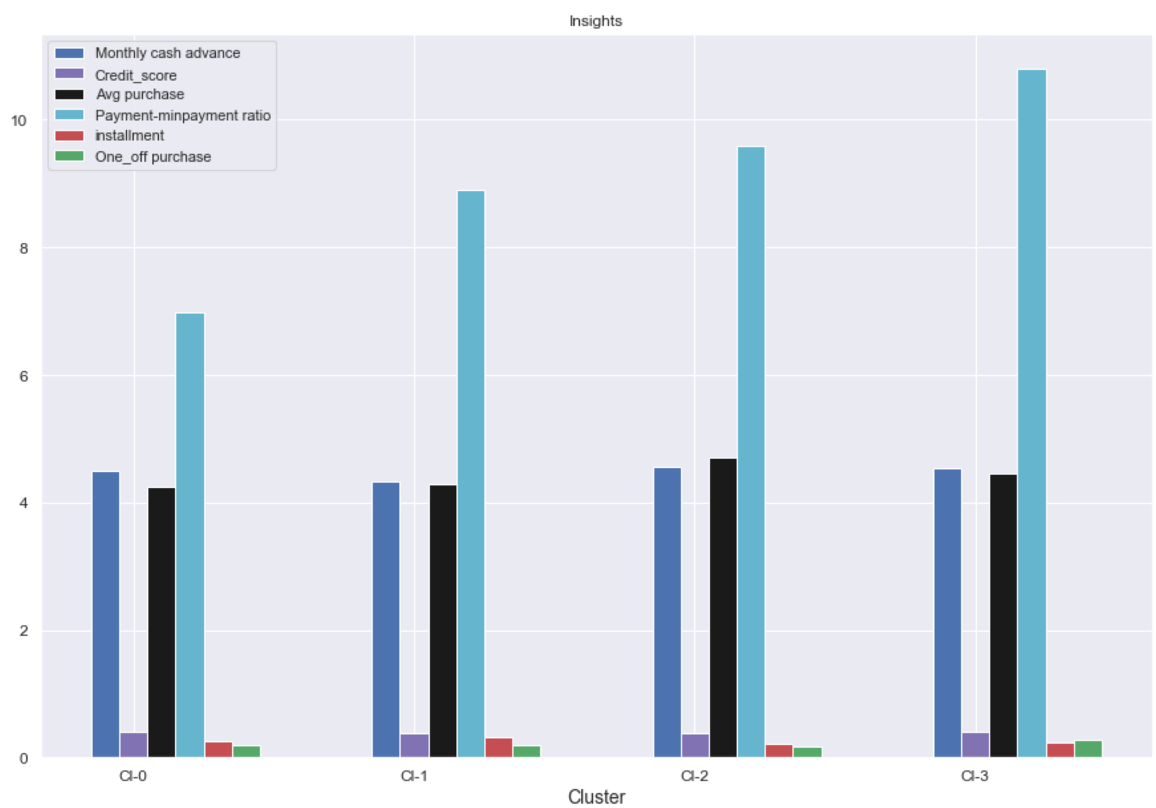
Out[131]:

Cluster_4	0.0	1.0	2.0	3.0
PURCHASES_TRX	12.894484	13.209982	17.923105	13.870985
Monthly_avg_purchase	69.217579	72.279009	110.822384	85.669090
Monthly_cash_advance	89.374306	76.249676	95.250877	93.151555
limit_usage	0.392937	0.377663	0.391016	0.394608
CASH_ADVANCE_TRX	3.241247	2.907374	3.336598	3.479657
payment_minpay	6.989812	8.907880	9.600647	10.812352
both_oneoff_installment	0.285372	0.262140	0.389191	0.279979
installment	0.252278	0.323291	0.212913	0.226981
one_off	0.194724	0.193795	0.179180	0.286403
none	0.267626	0.220773	0.218716	0.206638
CREDIT_LIMIT	4393.237410	3877.762926	5027.603291	4557.484535

In [132]:

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

cash_advance=np.log(cluster_4.loc['Monthly_cash_advance',:].values)
credit_score=(cluster_4.loc['limit_usage',:].values)
purchase= np.log(cluster_4.loc['Monthly_avg_purchase',:].values)
payment=cluster_4.loc['payment_minpay',:].values
installment=cluster_4.loc['installment',:].values
one_off=cluster_4.loc['one_off',:].values


bar_width=.10
b1=plt.bar(index,cash_advance,color='b',label='Monthly cash advance',wid
b2=plt.bar(index+bar_width,credit_score,color='m',label='Credit_score',w
b3=plt.bar(index+2*bar_width,purchase,color='k',label='Avg purchase',wid
b4=plt.bar(index+3*bar_width,payment,color='c',label='Payment-minpayment
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',w
b6=plt.bar(index+5*bar_width,one_off,color='g',label='One_off purchase',

plt.xlabel("Cluster")
plt.title("Insights")
plt.xticks(index + bar_width, ('Cl-0', 'Cl-1', 'Cl-2', 'Cl-3'))
plt.legend()
```

Out[132]: <matplotlib.legend.Legend at 0x14201e0c6d8>



Insights

Clusters are clearly distinguishing behavior within customers

Findings through clustering is validating Insights dervied from KPI. (as shown above in Insights from KPI

In [133]:
```
# Percentage of each cluster in the total customer base
s=cluster_df_4.groupby('Cluster_4').apply(lambda x: x['Cluster_4'].value
print (s,'\n')

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

```
Cluster_4
0.0        0.0    2087
1.0        1.0    2228
2.0        2.0    2758
3.0        3.0    1869
Name: Cluster_4, dtype: int64

Cluster -4

   Size  Percentage
0  2087   23.318436
1  2228   24.893855
2  2758   30.815642
3  1869   20.882682
```

In [134]:
```
cash_advance=np.log(cluster_4.loc['Monthly_cash_advance',:].values)
credit_score=list(cluster_4.loc['limit_usage',:].values)
purchase= np.log(cluster_4.loc['Monthly_avg_purchase',:].values)
payment=list(cluster_4.loc['payment_minpay',:].values)
installment=list(cluster_4.loc['installment',:].values)
one_off=list(cluster_4.loc['one_off',:].values)
```

In [135]:
```
cash_advance
```

Out[135]:
```
array([4.49283324, 4.33401316, 4.55651422, 4.53422779])
```

In [136]:
```
credit_score
```

Out[136]:
```
[0.3929369101259434, 0.3776631100107033, 0.3910163176458304, 0.394607595816
666]
```

In [137]:
```
purchase
```

Out[137]:
```
array([4.23725486, 4.28053375, 4.70792878, 4.45049208])
```

In [138]:
```
payment
```
Out[138]:
```
[6.989812363401613, 8.907880419653846, 9.600647126501185, 10.81235158206002
6]
```

In [139]:
```
installment
```
Out[139]:
```
[0.25227817745803355,
 0.3232913669064748,
 0.21291258614435982,
 0.22698072805139186]
```

In [140]:
```
one_off
```
Out[140]:
```
[0.1947242206235012,
 0.19379496402877697,
 0.17918026840768952,
 0.28640256959314775]
```

Finding behaviour with 5 Clusters:

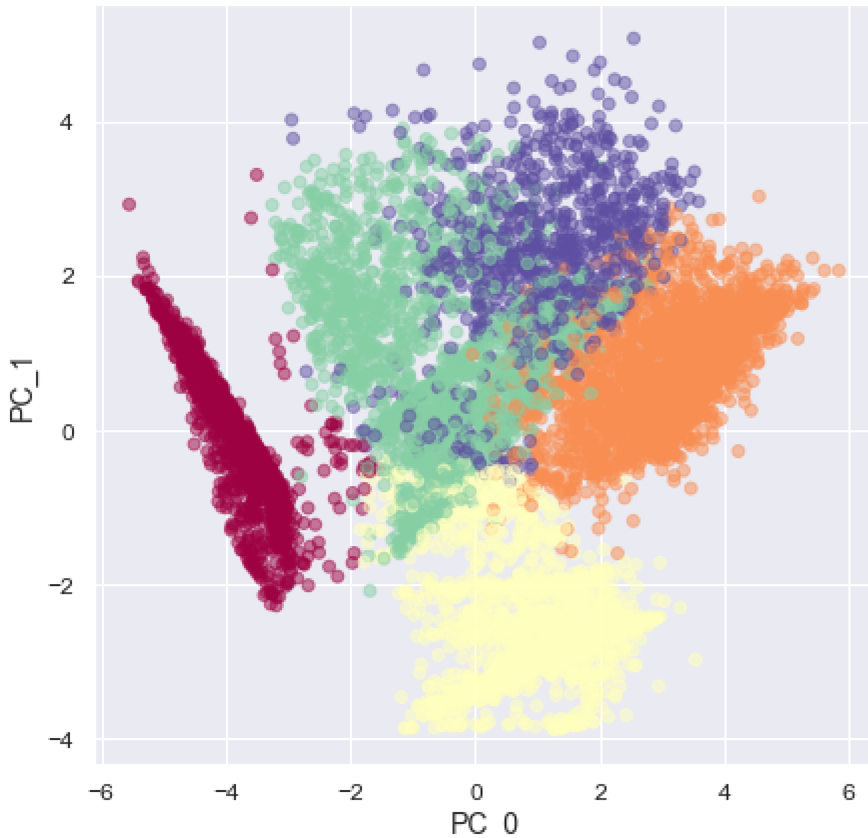
In [141]:
```
km_5=KMeans(n_clusters=5,random_state=123)
km_5=km_5.fit(reduced_cr)
km_5.labels_
```
Out[141]:
```
array([2, 0, 3, ..., 2, 0, 3])
```

In [142]:
```
pd.Series(km_5.labels_).value_counts()
```
Out[142]:
```
2    2130
0    2081
1    1984
3    1857
4     890
dtype: int64
```

In [143]:

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

Out[143]: Text(0, 0.5, 'PC_1')



In [144]:

```
cluster_df_5=pd.concat([cre_original[col_kpi],pd.Series(km_5.labels_,nam
```

In [145]:
```
# Finding Mean of features for each cluster
cluster_df_5.groupby('Cluster_5')\
.apply(lambda x: x[col_kpi].mean()).T
```

Out[145]:

Cluster_5	0.0	1.0	2.0	3.0	4.0
PURCHASES_TRX	12.928812	18.330141	13.066322	13.962823	16.478065
Monthly_avg_purchase	69.420265	113.107271	72.013400	85.988683	100.347921
Monthly_cash_advance	88.137382	82.887873	73.980875	92.875813	129.573087
limit_usage	0.392890	0.378925	0.374729	0.394603	0.423726
CASH_ADVANCE_TRX	3.214526	3.019153	2.848542	3.468750	4.224972
payment_minpay	6.979259	10.688515	9.099627	10.828463	6.627688
both_oneoff_installment	0.284752	0.396169	0.260113	0.280172	0.363330
installment	0.252044	0.209173	0.324553	0.226293	0.232846
one_off	0.196248	0.173387	0.192380	0.286099	0.195726
none	0.266955	0.221270	0.222954	0.207435	0.208099
CREDIT_LIMIT	4394.588745	5076.714855	3832.829138	4552.629909	4895.050619

With 5 clusters :

1. we have a group of customers (cluster 2) having highest avergae purchases but there is Cluster 4 also having highest cash advance & secong highest purchase behaviour but their type of purchases are same.
2. Cluster 0 and Cluster 4 are behaving similar in terms of Credit_limit and have cash transactions is on higher side

So we don't have quite distinguishable characteristics with 5 clusters,

In [146]:
```
s1=cluster_df_5.groupby('Cluster_5').apply(lambda x: x['Cluster_5'].valu
s1
```

Out[146]:
```
Cluster_5
0.0        0.0    2081
1.0        1.0    1984
2.0        2.0    2130
3.0        3.0    1857
4.0        4.0     890
Name: Cluster_5, dtype: int64
```

In [147]:

```
# percentage of each cluster

print ("Cluster-5",'\n')
per_5=pd.Series((s1.values.astype('float')/ cluster_df_5.shape[0])*100,n
pd.concat([pd.Series(s1.values,name='Size'),per_5],axis=1)
```

```
Cluster-5
```

Out[147]:

	Size	Percentage
0	2081	23.251397
1	1984	22.167598
2	2130	23.798883
3	1857	20.748603
4	890	9.944134

In [148]:

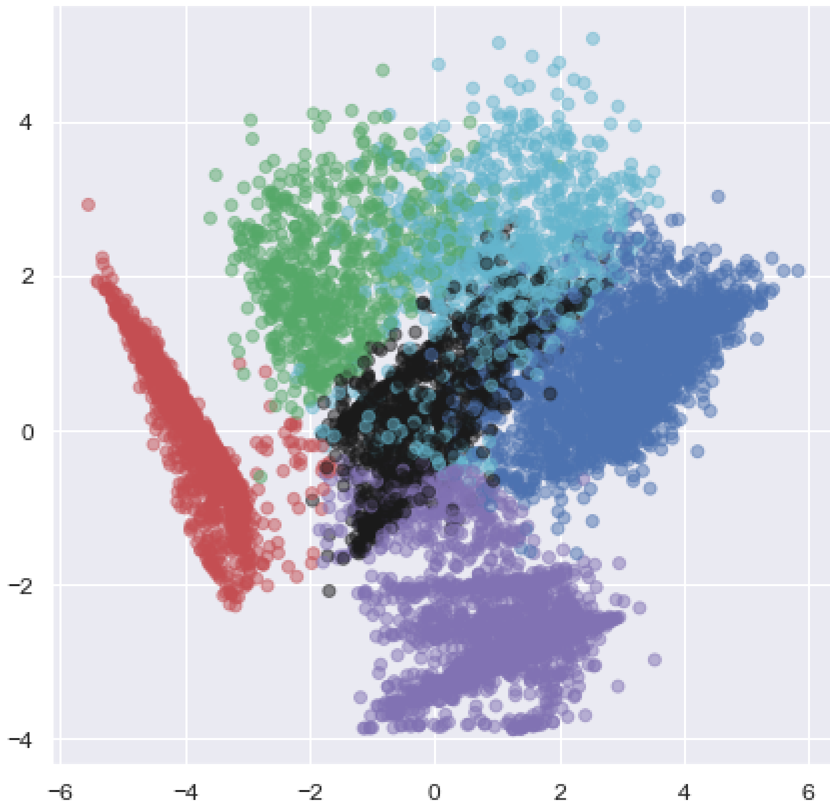
```
km_6=KMeans(n_clusters=6).fit(reduced_cr)
km_6.labels_
```

Out[148]:

```
array([4, 0, 5, ..., 4, 0, 2])
```

In [149]:
```
color_map={0:'r',1:'b',2:'g',3:'c',4:'m',5:'k'}
label_color=[color_map[l] for l in km_6.labels_]
plt.figure(figsize=(7,7))
plt.scatter(reduced_cr[:,0],reduced_cr[:,1],c=label_color,cmap='Spectral
```

Out[149]: <matplotlib.collections.PathCollection at 0x14265dd75c0>



In [150]:
```
cluster_df_6=pd.concat([cre_original[col_kpi],pd.Series(km_6.labels_,nam
```

In [151]:

```
six_cluster=cluster_df_6.groupby('Cluster_6').apply(lambda x: x[col_kpi]
six_cluster
```

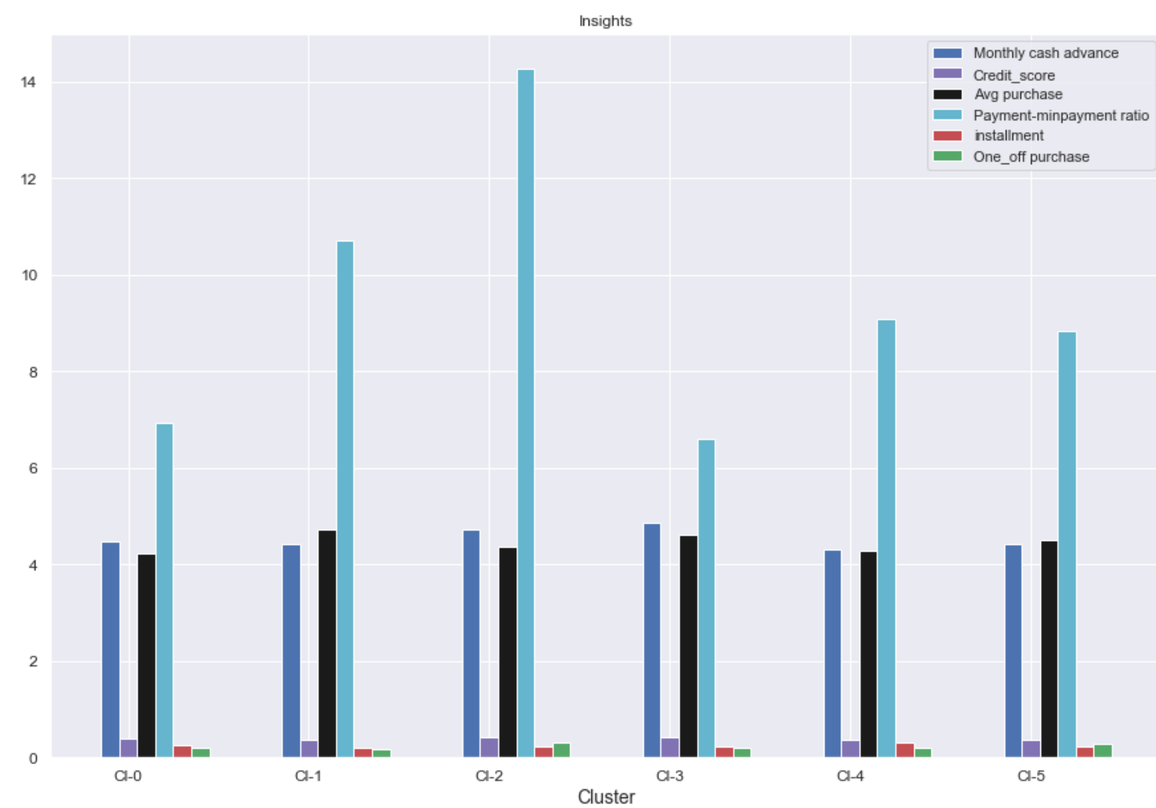
Out[151]:

Cluster_6	0.0	1.0	2.0	3.0	4.0	
PURCHASES_TRX	12.958997	18.298682	13.340517	16.628959	13.066322	1
Monthly_avg_purchase	69.593683	113.267586	78.193280	100.645526	72.013400	9
Monthly_cash_advance	87.754066	82.933591	112.106857	128.747045	73.980875	8
limit_usage	0.391908	0.378995	0.429352	0.424740	0.374729	
CASH_ADVANCE_TRX	3.199228	3.026876	3.998563	4.186652	2.848542	
payment_minpay	6.926606	10.701269	14.284919	6.596767	9.099627	
both_oneoff_installment	0.286059	0.396552	0.287356	0.363122	0.260113	
installment	0.250844	0.208418	0.216954	0.231900	0.324553	
one_off	0.196334	0.172921	0.303161	0.195701	0.192380	
none	0.266763	0.222110	0.192529	0.209276	0.222954	
CREDIT_LIMIT	4400.940666	5080.604601	4561.637931	4860.011312	3832.829138	456

In [152]:

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

cash_advance=np.log(six_cluster.loc['Monthly_cash_advance',:].values)
credit_score=(six_cluster.loc['limit_usage',:].values)
purchase= np.log(six_cluster.loc['Monthly_avg_purchase',:].values)
payment=six_cluster.loc['payment_minpay',:].values
installment=six_cluster.loc['installment',:].values
one_off=six_cluster.loc['one_off',:].values

bar_width=.10
b1=plt.bar(index,cash_advance,color='b',label='Monthly cash advance',wid
b2=plt.bar(index+bar_width,credit_score,color='m',label='Credit_score',w
b3=plt.bar(index+2*bar_width,purchase,color='k',label='Avg purchase',wid
b4=plt.bar(index+3*bar_width,payment,color='c',label='Payment-minpayment
b5=plt.bar(index+4*bar_width,installment,color='r',label='installment',w
b6=plt.bar(index+5*bar_width,one_off,color='g',label='One_off purchase',

plt.xlabel("Cluster")
plt.title("Insights")
plt.xticks(index + bar_width, ('Cl-0', 'Cl-1', 'Cl-2', 'Cl-3','Cl-4','Cl

plt.legend()
```

Out[152]: <matplotlib.legend.Legend at 0x142676066d8>



In [153]:
```python
cash_advance=np.log(six_cluster.loc['Monthly_cash_advance',:].values)
credit_score=list(six_cluster.loc['limit_usage',:].values)
cash_advance
```

Out[153]:
```
array([4.47453819, 4.41804017, 4.7194525 , 4.85784959, 4.30380661,
       4.41647666])
```

Insights with 6 clusters

1. Here also groups are overlapping .
2. Cl-0 and Cl-2 behaving same

Checking performance metrics for Kmeans

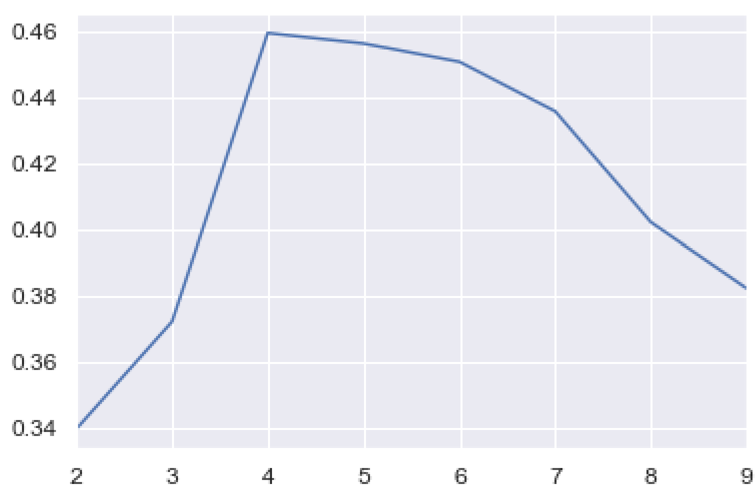
I am validating performance with 2 metrics Calinski harabaz and Silhouette score

In [154]:
```python
from sklearn.metrics import calinski_harabaz_score,silhouette_score
```

In [155]:
```python
score={}
score_c={}
for n in range(2,10):
    km_score=KMeans(n_clusters=n)
    km_score.fit(reduced_cr)
    score_c[n]=calinski_harabaz_score(reduced_cr,km_score.labels_)
    score[n]=silhouette_score(reduced_cr,km_score.labels_)
```

In [156]:
```python
pd.Series(score).plot()
```

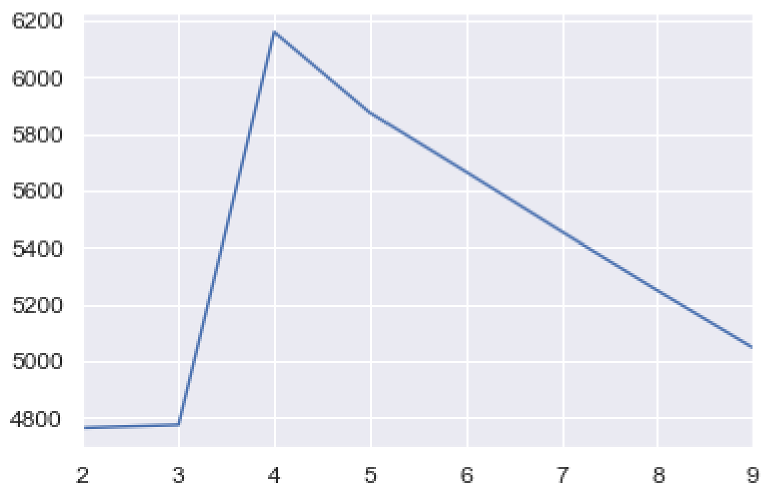
Out[156]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x14265450e48>
```



From calinski_harabaz_score method above graph it is also suggesting to choose the K value is 4 i.e. number of cluster = 4

In [157]:
```
pd.Series(score_c).plot()
```

Out[157]: <matplotlib.axes._subplots.AxesSubplot at 0x14265cccf60>



Even from silhouette_score method above graph it is also suggesting to choose the K value is 4 i.e. number of cluster = 4

Performance metrics also suggest that K-means with 4 cluster is able to show distinguished characteristics of each cluster.

I am going to neglect with the K-means with 5 and 6 clusters and going to accept the K-means with 4 cluster. As I have already mentionaed the reason above with 5 and 6 clusters behaving the similar kind with other cluster too. Thats why I choose the 4 cluster

Insights with 4 Clusters

1. Cluster 2 is the group of customers who have highest Monthly_avg purchases and doing both installment as well as one_off purchases, have comparatively good credit score. **This group is about 31% of the total customer base**

2. cluster 1 is taking maximum advance_cash and is paying comparatively less minimum payment and poor credit_score & doing no purchase transaction. **This group is about 23% of the total customer base**

3. Cluster 0 customers are doing maximum One_Off transactions and least payment ratio and credit_score on lower side. **This group is about 21% of the total customer base**

4. Cluster 3 customers have maximum credit score and are paying dues and are doing maximum installment purchases. **This group is about 25% of the total customer base**

Marketing Strategy Suggested:

a. Group 2

They are potential target customers who are paying dues and doing purchases and maintaining comparatively good credit score) -- we can increase credit limit or can lower down interest rate -- Can be given premium card /loyality cards to increase transactions

b. Group 1

They have poor credit score and taking only cash on advance. We can target them by providing less interest rate on purchase transaction

c. Group 0

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

d. Group 3

This group is performing best among all as cutomers are maintaining good credit score and paying dues on time. -- Giving rewards point will make them perform more purchases.

Agglomerative / Hierarchichal Clustering

In [158]:
```
data_scaled.head()
```

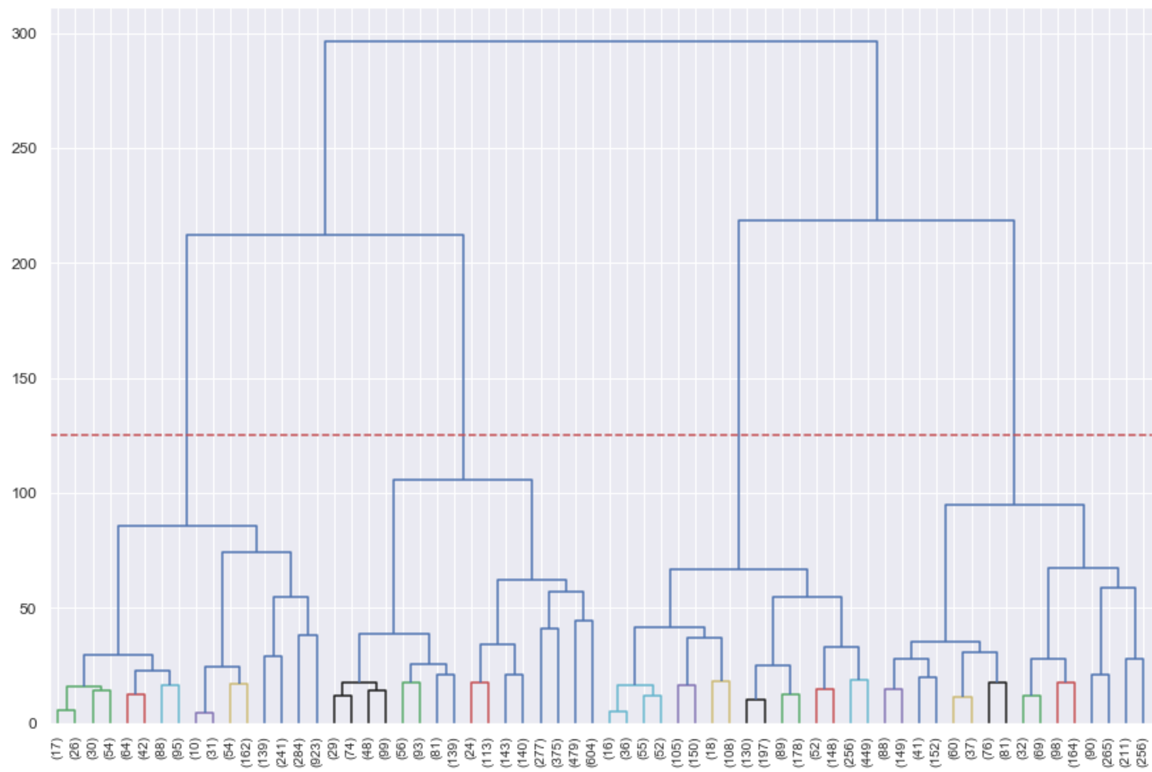
Out[158]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PU
0	-0.731298	-0.248965	-0.425100	-0.357027	
1	0.789014	0.134664	-0.469735	-0.357027	
2	0.448884	0.518292	-0.107987	0.108603	
3	0.050491	-1.016222	0.231613	0.545724	
4	-0.357750	0.518292	-0.462249	-0.347391	

In [159]:
```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram,linkage
```

In [160]:
```
Z=linkage(cr_scaled,method="ward")
```

In [161]:
```
plt.figure(figsize=(15,10))
dendrogram(Z,leaf_rotation=90,p=5,color_threshold=20,leaf_font_size=10,t
plt.axhline(y=125, color='r', linestyle='--')
plt.show()
```



If we draw a horizontal line that passes through longest distance without a horizontal line, It intersects 4 vertical lines

So, Optimal cluster = 4

In [162]:
```
model=AgglomerativeClustering(n_clusters=4,affinity='euclidean',linkage=
```

In [163]:
```
model.fit(cr_scaled)
```

Out[163]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='ward', memory=None, n_clusters=4,
                        pooling_func='deprecated')
```

In [164]:
```
model.labels_
```

Out[164]:
```
array([3, 2, 1, ..., 3, 2, 1], dtype=int64)
```

In [165]:
```
clusters_agg=pd.concat([credit_2, pd.DataFrame({'cluster':model.labels_}
clusters_agg.head()
```

Out[165]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	1666.670542	0.636364	1499.00	1499.00	
4	817.714335	1.000000	16.00	16.00	

In [166]:
```
clusters_agg.groupby('cluster').mean()
```

Out[166]:

cluster	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLME
0.0	1739.353406	0.887807	1295.766895	780.857115	
1.0	1600.949929	0.879679	995.693389	629.884454	
2.0	1554.729164	0.880859	821.661119	463.576156	
3.0	1322.249983	0.858881	819.013205	448.922332	

In [167]:
```
# PCA
cluster_pca
```

Out[167]:

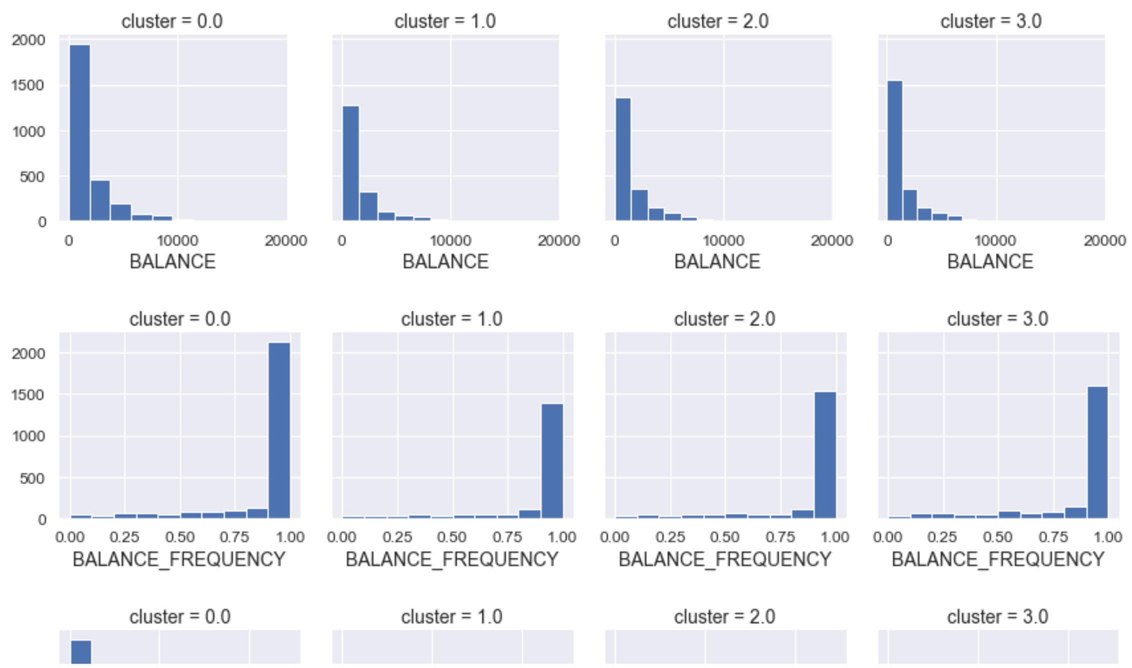
cluster	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLME
0.0	1558.009898	0.880551	810.042010	456.277564	
1.0	1313.909562	0.858399	828.377163	454.877410	
2.0	1744.587965	0.888570	1299.736257	783.119467	
3.0	1597.091118	0.879153	995.670814	630.068469	

Almost similar result as kmeans clustering

But only difference is there Cluster 0 and 2 are interchanged and, cluster 1 and 3 are interchanged.

In [168]:

```
for c in clusters_agg:
    grid= sns.FacetGrid(clusters_agg, col='cluster')
    grid.map(plt.hist, c)
```



In [169]:

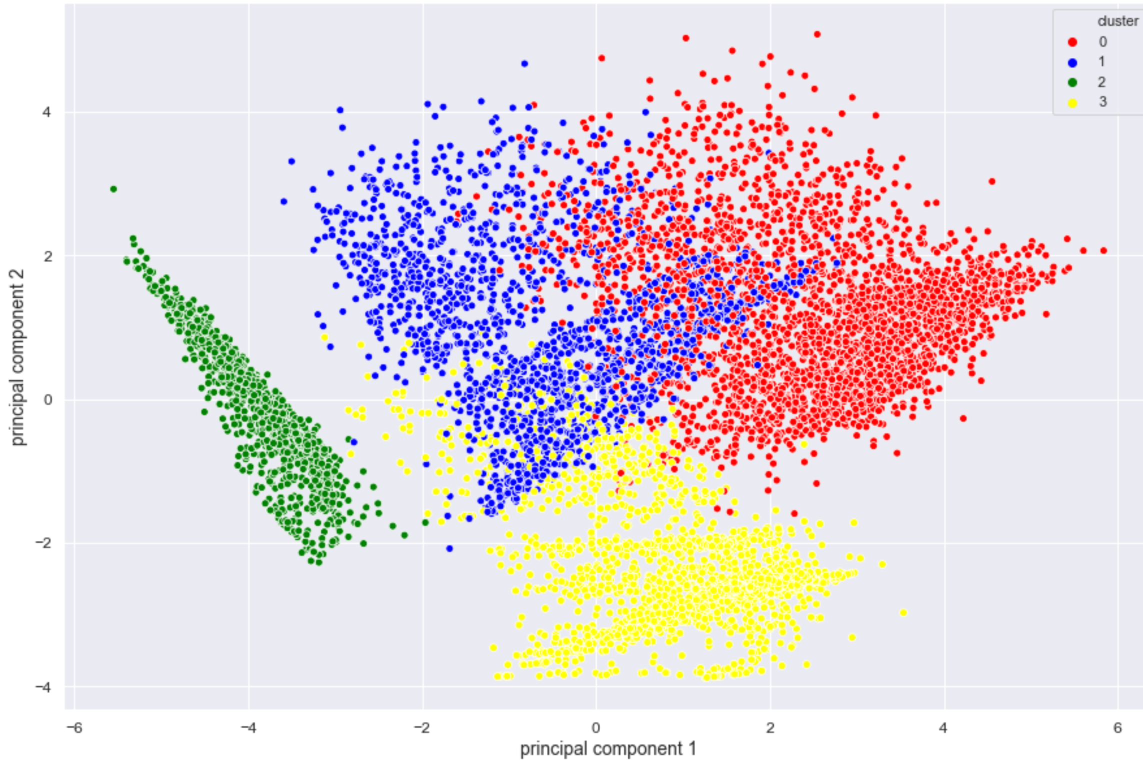
```
finalDf_agg = pd.concat([principalDf, pd.DataFrame({'cluster':model.labe
finalDf_agg.head()
```

Out[169]:

	principal component 1	principal component 2	cluster
0	-0.245367	-2.760833	3
1	-3.981189	0.155246	2
2	1.287526	1.495470	1
3	-1.048933	0.663479	1
4	-1.451981	-0.185896	1

In [170]:

```
# Agglomerative / Hierarchichal Clustering
plt.figure(figsize=(15,10))
ax = sns.scatterplot(x="principal component 1", y="principal component 2
plt.show()
```



In [171]:

```
# PCA
plt.figure(figsize=(15,10))
ax = sns.scatterplot(x="principal component 1", y="principal component 2
plt.show()
```

By analysing and visualiing the both concept with the Principal Component Analysing(PCA) i.e. KMeans and Agglomerative / Hierarchichal Clustering both algorithm had given 4 cluster.
Almost the behaviour of Agglomerative / Hierarchichal Clustering is same as KMeans with similar results. But only difference is there Cluster 0 and 2 are interchanged and, cluster 1 and 3 are interchanged.
So Final conclusion is that I have accepted KMeans algorithm over Agglomerative / Hierarchichal Clustering. Choosing the n_components for K with the both concept Elbow Criterion Method and Silhouette Coefficient Method which give the value 4.
I have tried to explain every concept which I have used in this project with acceptance as well as rejected.
Even I have tried to explain the graph and the code with the comments.
I have explain the market strategy and also the behaviour of all 4 cluster.

To Run Python file which is format of .ipynb file.
save the .ipynb file in with the dataset same folder location, open the command promt with same folder location.
Type jupyter notebook in command promt, It will open the browser. there you can run this .ipynb file.

R FILE

Clustering

Unlinked/uncorrelated variables can allow to cluster a datset, but determining the most likely number of clusters is another problem. In addition, it’s useful to identify variables that define clusters and that are likely to be actionable.

After cleaning the dataset, I transformed and scaled appropriate variables. I then visualized the distributions of each variable as well as relationships between pairs of variables. I identified pairs of variables that were highly correlated and removed one from each pair.

I used k-means clustering on the dataset of uncorrelated (less correlated than my cutoff) variables, forming numbers of clusters from K = 2 to 10. I used a consensus of 26 measurements that can help choose the best K. I then tried another method, the gap statistic, which is more computationally instensive than the other 26 methods, but tests each K against simulated null distributions from the dataset.

While there’s no clear winner for K to partition these data into clusters, using the consensus value for K, I created a table of summary statistics for customers in each cluster to characterize customer behavior for each cluster.

R Code

#First, I have cleared the environment using code in R by using
rm(list = ls())

#Then, I set mydirectory using code
setwd("D:/Online_corses/Edwisor/Edwisor/Projects/credit-card")

#View the directory
getwd()

library(tidyverse) ## manipulating and visualizing data (plyr, purrr, ggplot2, knitr...)
library(readr) ## read in csv files faster

library(kableExtra) ## make nice tables with wrapper for kable()
library(cluster) ## clustering algorithms and gap statistic
library(factoextra) ## visualization of clustering algorithm results
library(GGally) ## create matrix of variable plots
library(NbClust) ## clustering algorithms and identification of best K
library(caret) ## find correlated variables
library(DataExplorer) ## help with different tasks throughout data exploration process.
library(dplyr) ## provides a set of tools for efficiently manipulating datasets
library(kdensity)## Handles univariate non-parametric density
library(reshape2)## to transform data between wide and long formats.
library(purrr) ## fills the missing pieces in R's functional programming tools
library(mlr) ## Machine Learning in R Interface to a large number of classification and regression techniques, including machine-readable parameter descriptions.
library(dendextend) ## a set of functions for extending 'dendrogram' objects in R and to visualize and compare trees of 'hierarchical clusterings'.
library(ggforce) ## providing missing functionality to ggplot2 through the extension system

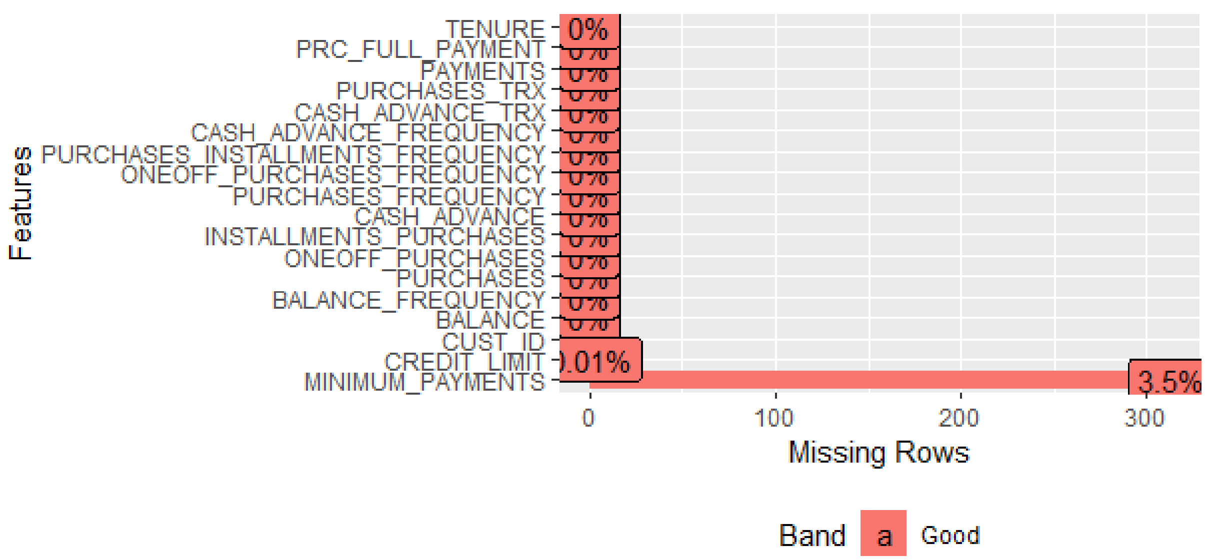
cc_data <- read.csv("credit-card-data.csv",header=TRUE) #load the dataset

glimpse(cc_data) ## show variable names, variable class, and examples of data in each column

summary(cc_data) ## get min, max, median, mean, # of NAs for each variable

#CHECKING How many NA VALUES are exits in dataset
sum(is.na(cc_data))

#CHECKING IF THERE ARE ANY NA VALUES with visualization.
plot_missing(cc_data)



##there is 313 NA in MINIMUM_PAYMENTS . replacing it with median of MINIMUM_PAYMENTS

```
cc_data$MINIMUM_PAYMENTS[which(is.na(cc_data$MINIMUM_PAYMENTS))]<-
median(cc_data$MINIMUM_PAYMENTS, na.rm=TRUE)
```

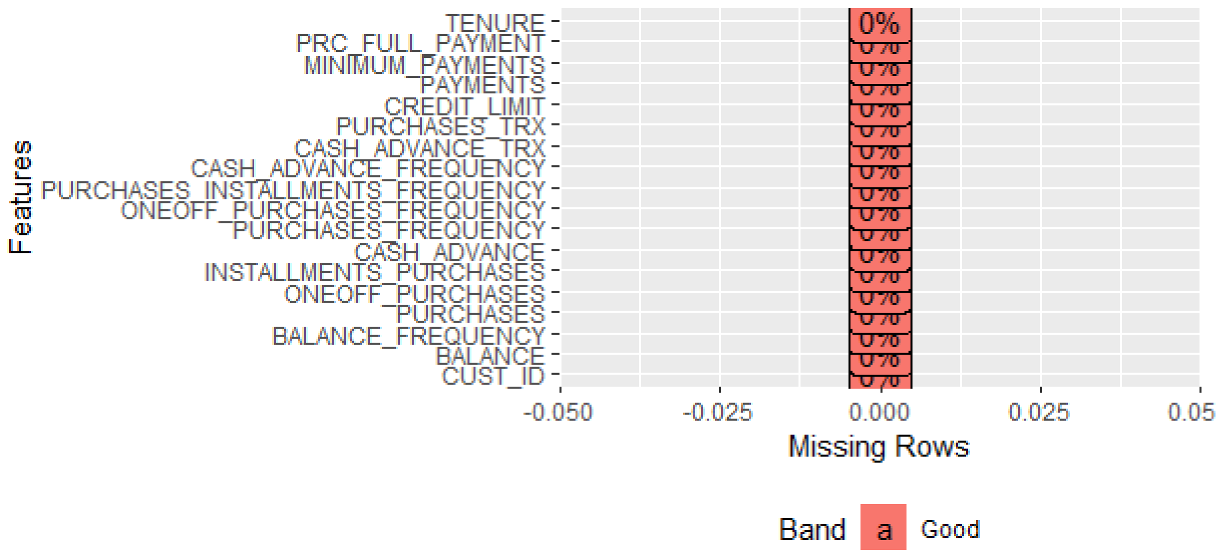
summary(cc_data)

#there is one NA in Credit_limit . replacing it with mean of credit_limit

```
cc_data$CREDIT_LIMIT[which(is.na(cc_data$CREDIT_LIMIT))] <-
median(cc_data$CREDIT_LIMIT, na.rm=TRUE)
```

summary(cc_data)

#CHECKING IF THERE ARE STILL ANY NA VALUES
plot_missing(cc_data)



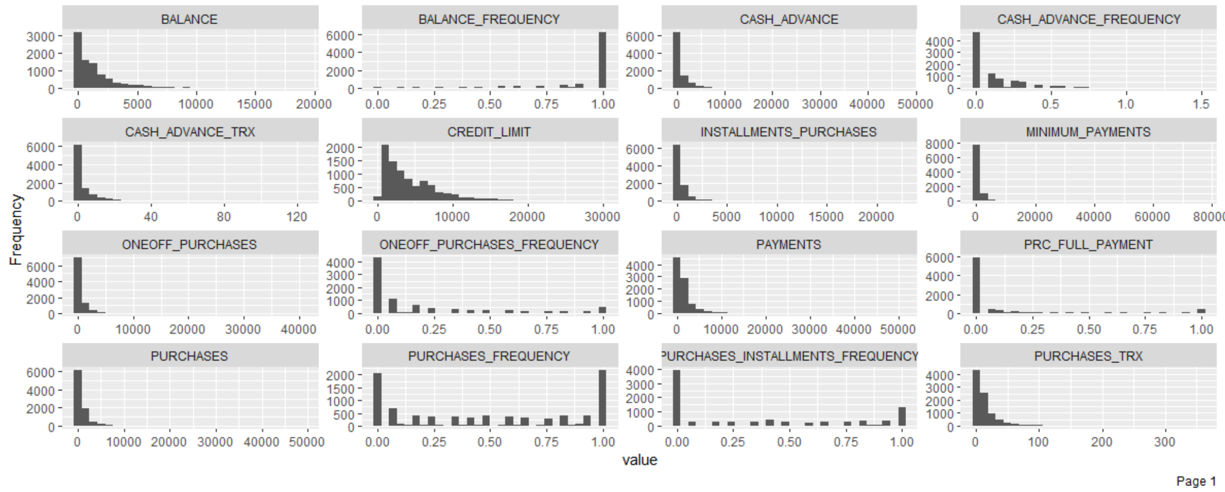
#NO MISSING VALUES NOW
str(cc_data)

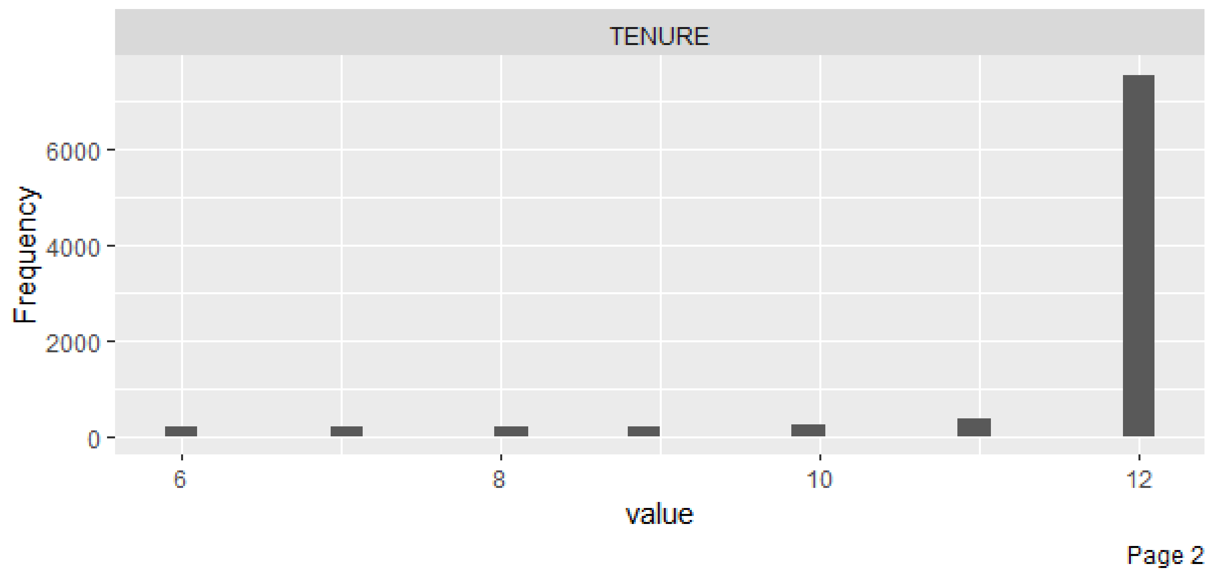
#removing first column i.e. drop CUST_ID since it´s useless.
cc_data <- cc_data[, -1]
length(cc_data)

str(cc_data)

EXPLORATORY DATA ANALYSIS

#Histograms
plot_histogram(cc_data)





It is same as I have already discuss in above in python code section.

#Nearly all variables are skewed and/or have some outliers.

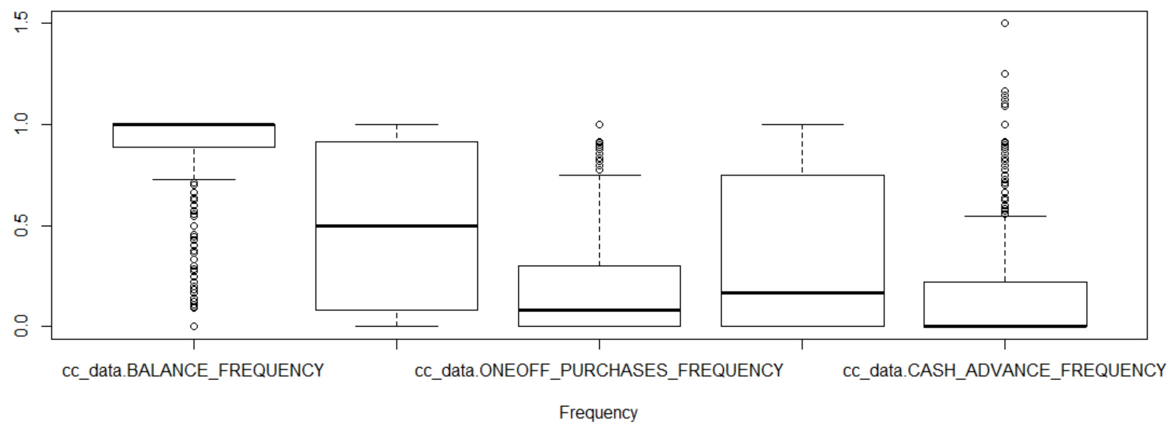
#Therefore, I will keep them for this analysis.

#let´s see how are distributed the frequency variables

```
frequency = data.frame(cc_data$BALANCE_FREQUENCY, cc_data$PURCHASES_FREQUENCY, cc_data$ONEOFF_PURCHASES_FREQUENCY,
                            cc_data$PURCHASES_INSTALLMENTS_FREQUENCY, cc_data$CASH_ADVANCE_FREQUENCY) <br>
```

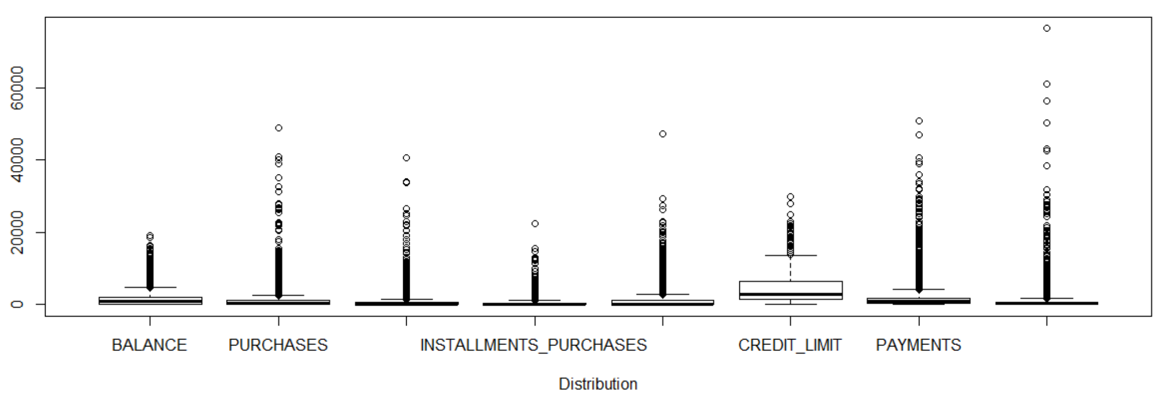
#We have data on Cash_advance_frequency that is wrong. I will clean the dataset later.

#There are also many outliers(the black dots), but I will keep then for now
boxplot(frequency, outline = TRUE, xlab='Frequency')



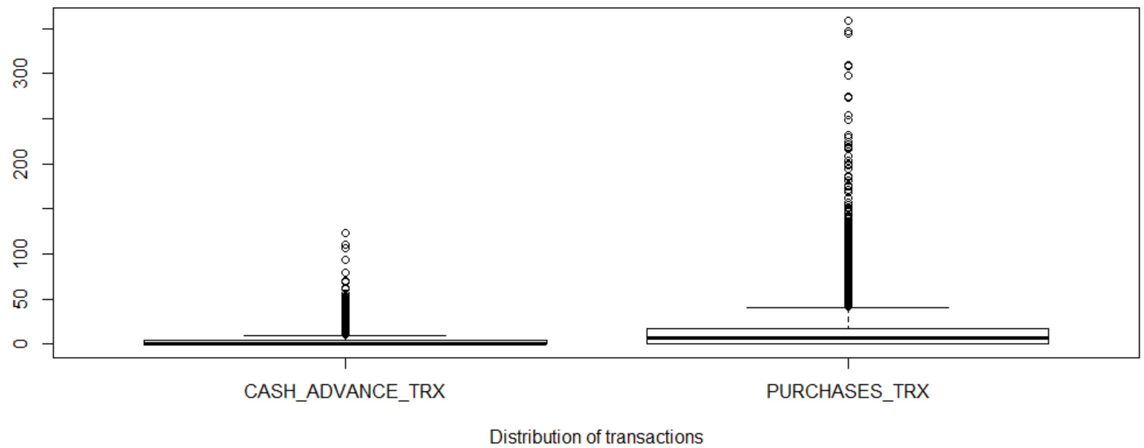
#let´s see how are distributed the numeric variables
numeric_variables = select(cc_data,'BALANCE', 'PURCHASES', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS')

#There are also many outliers(the black dots), but I will keep them for now
boxplot(numeric_variables, outline = TRUE, xlab='Distribution')



#let´s see how are distributed the numeric variables of transactions
transaction = select(cc_data, 'CASH_ADVANCE_TRX', 'PURCHASES_TRX')

#There are also many outliers(the black dots), but I will keep them for now
boxplot(transaction, outline = TRUE, xlab='Distribution of transactions')



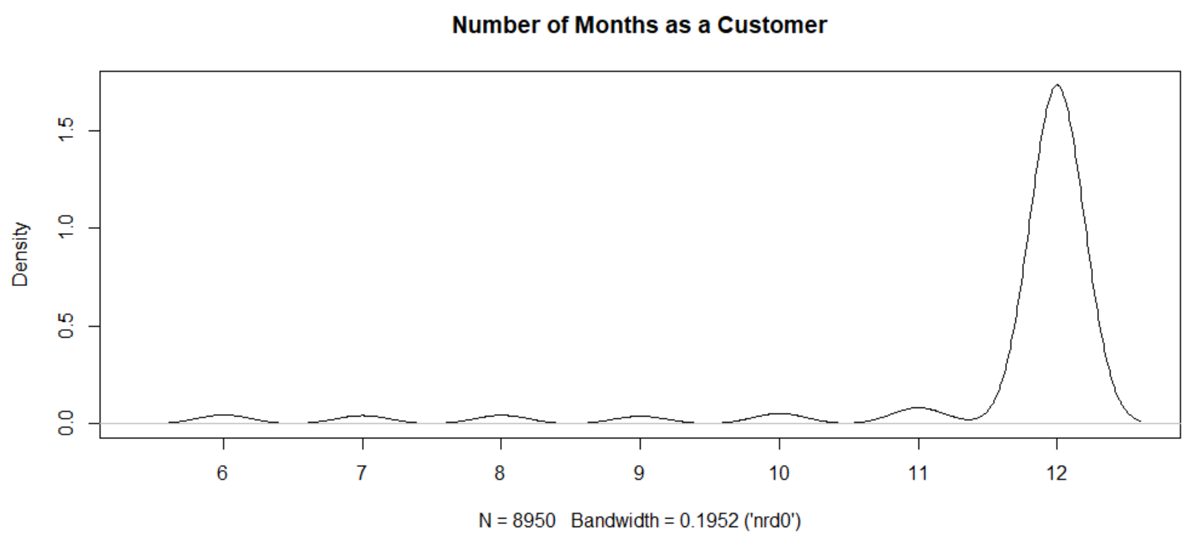
#As I can see, There are many outliers. But, I can't simply drop the outliers as they may contain useful information.

#So, I'll treat them as extreme values

#let´s see how is distributed the tenure

kde = kdensity(cc_data$TENURE)

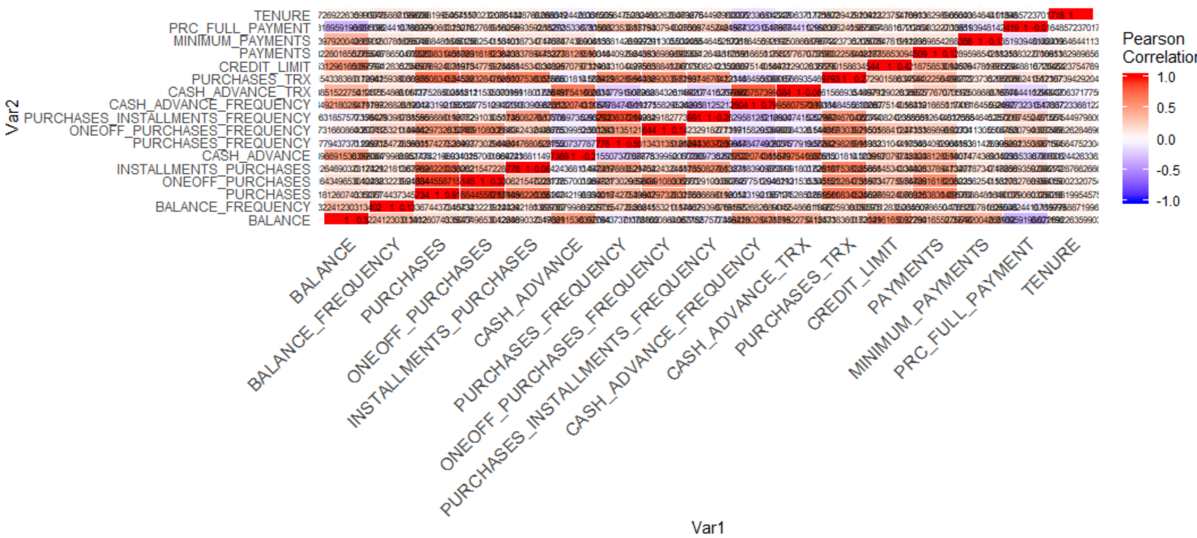
#it shows that most of the distribution of TENURE is 12 months as a customer
plot(kde, main = 'Number of Months as a Customer')



Correlations

#Lets take a look at how the variables are correlated

```
ggplot(data = melt(cor(cc_data)), aes(x=Var1, y=Var2, fill=value)) +
geom_tile(color = "white")+
scale_fill_gradient2(low = "blue", high = "red", mid = "white",
midpoint = 0, limit = c(-1,1), space = "Lab",
name="Pearson\nCorrelation")+
theme_minimal()+ # minimal theme
theme(axis.text.x = element_text(angle = 45, vjust = 1,
size = 12, hjust = 1))+
geom_text(aes(Var2, Var1, label = value), color = "black", size = 2)
```



It is same as I have already discuss in above in python code section.

Data Cleaning

#here I am cleaning the CASH_ADVANCE_FREQUENCY because some of the data given is wrong i.e. more than frequency 1

#wich is not valid in frequency.

#Lets clean the data (inputing values and eliminating wrong data) before the segmentation

```
cc_data = cc_data[!(cc_data$CASH_ADVANCE_FREQUENCY>1),]
```

#we have 8 records for which the frequency is higher that 1. I will eliminate these records

```
str(cc_data)
```

Clustering

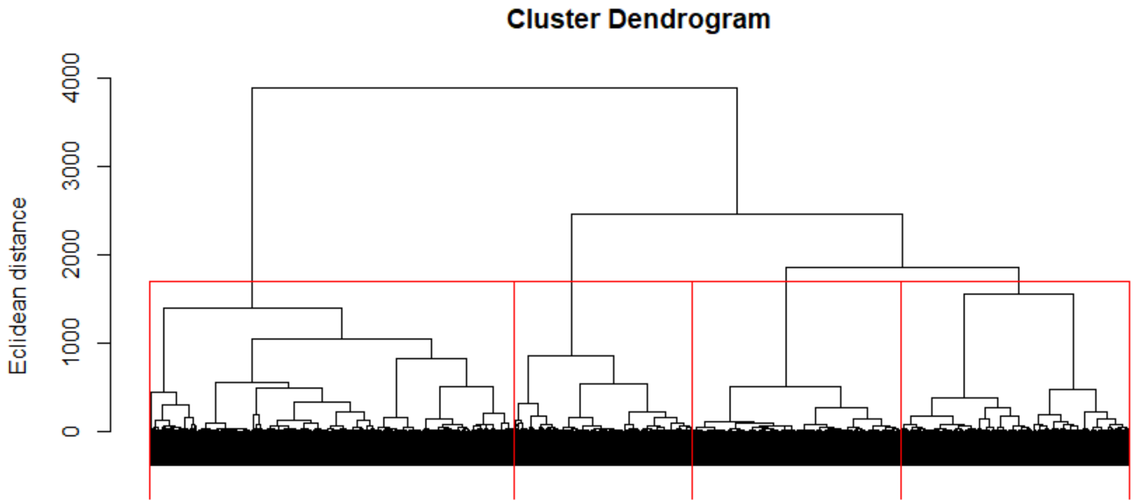
Hieracical clustering

#First I try hieracical clustering. Since all variables are categorical I use the eucldean distance.

```
fit_hc_clust = hclust(dist(scale(cc_data), method = "euclidean"), method = "ward")
```

#Regrading the dendrogramm 4 clusters seams to be a good size

plot(fit_hc_clust, labels = FALSE, sub = "", xlab = "", ylab = "Eclidean distance")



rect.hclust(fit_hc_clust, k = 4)

#So, I cut the dendrogram for 4 clusters.

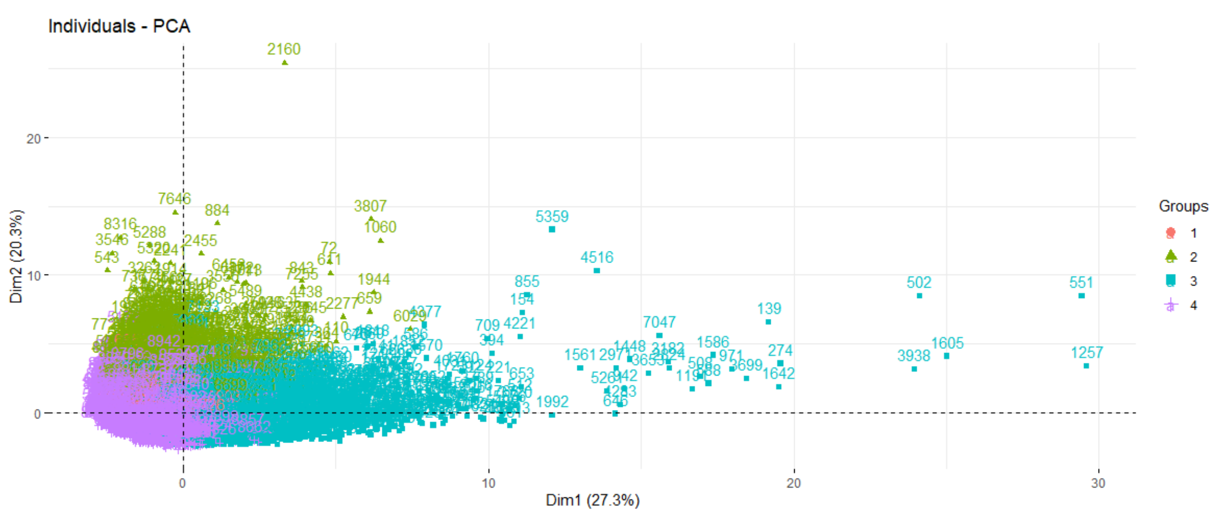
hc_cluster = cutree(fit_hc_clust, k = 4)

#The PCA plots the data in two-dimensional space. Overall, there are no clear clusters in the data.

#However, the generated clusters look quite noisy since they are overlapping.

hc_pc = prcomp(scale(cc_data))

fviz_pca_ind(hc_pc, habillage = hc_cluster)



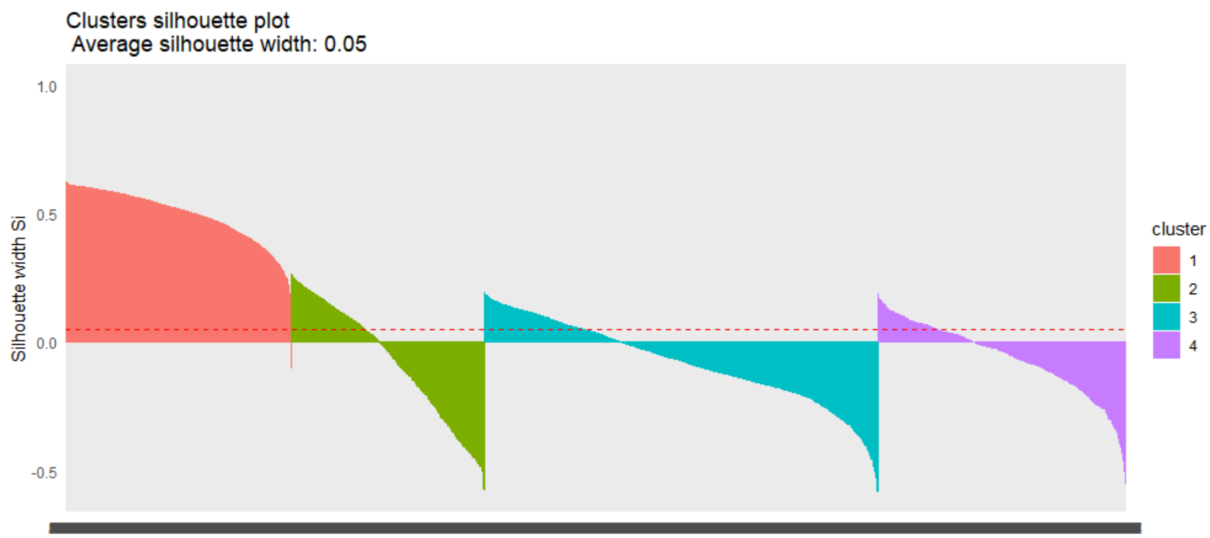
#Let´s take a look at the silhouette plot. It shows if an observation is associated with the right (1) or wrong (-1) cluster.

#The average silhouette width is quite low.

#Many observations probably in the wrong clusters.

hc_sil = silhouette(hc_cluster, dist(scale(cc_data), method = "euclidean"), lable = FALSE)

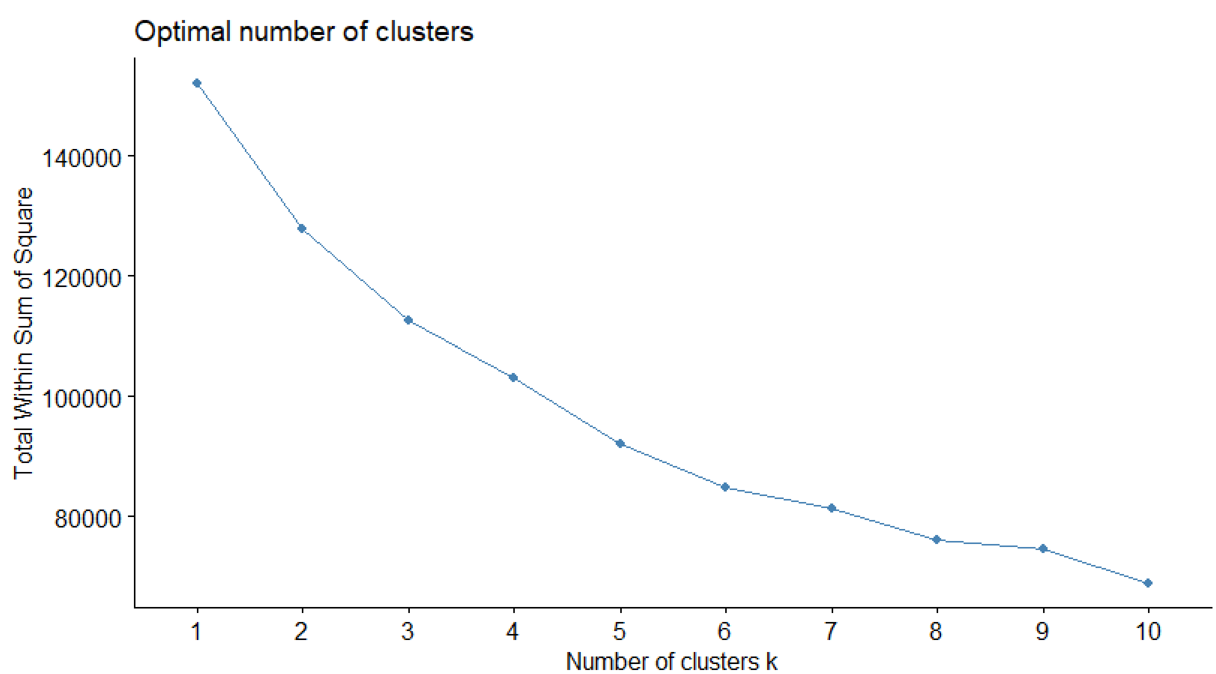
fviz_silhouette(hc_sil, print.summary = FALSE) + theme_minimal()



K-Means

#Second, I try K-Meams.

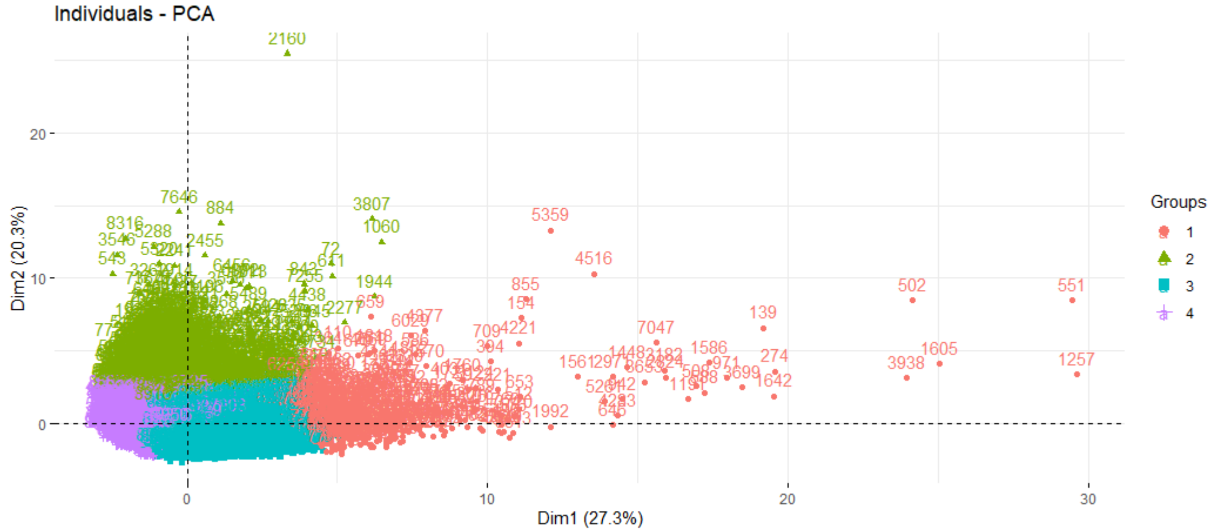
#Regarding the wss plot 4 clusters seem to be a proper number of clusters.
fviz_nbclust(scale(cc_data), kmeans, method = "wss", k.max = 10)



#Therefore, I fit K-Means with 4 clusters.
fit_km = kmeans(scale(cc_data), centers = 4)

#This PCA plot looks better then the plot before.

fviz_pca_ind(hc_pc, habillage = fit_km$cluster)



#Let´s take a look at this silhouette plot. Overall,

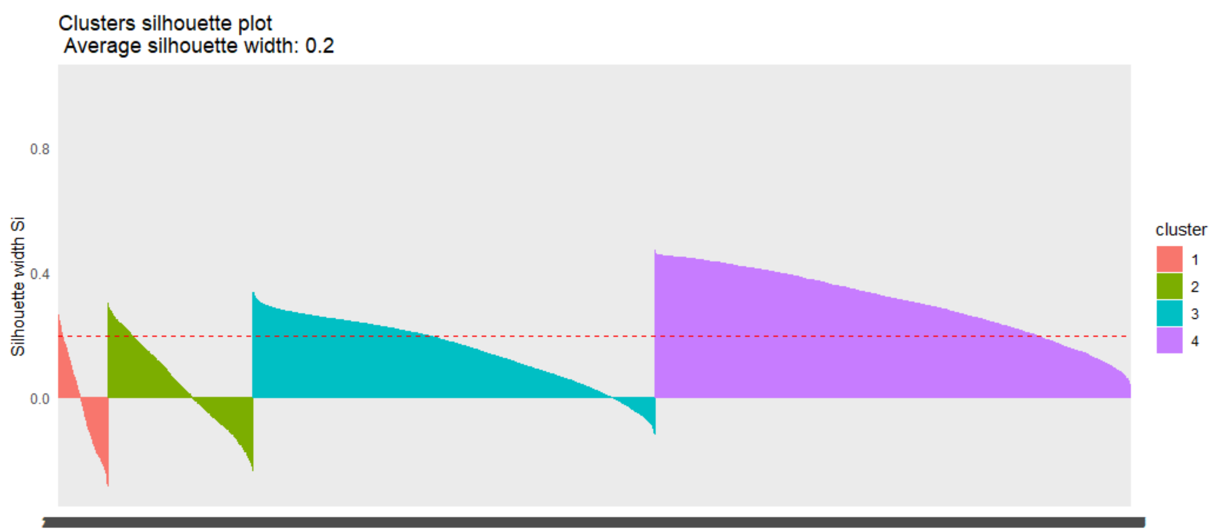
#the result is better than before. However, especially cluster 1 and 3

#have still some observations which are still in the wrong cluster.

#But it´s the best solution for now which I will use for interpretation.

hc_sil = silhouette(fit_km$cluster, dist(scale(cc_data), method = "euclidean"), lable = FALSE)

fviz_silhouette(hc_sil, print.summary = FALSE) + theme_minimal()



Interpretation

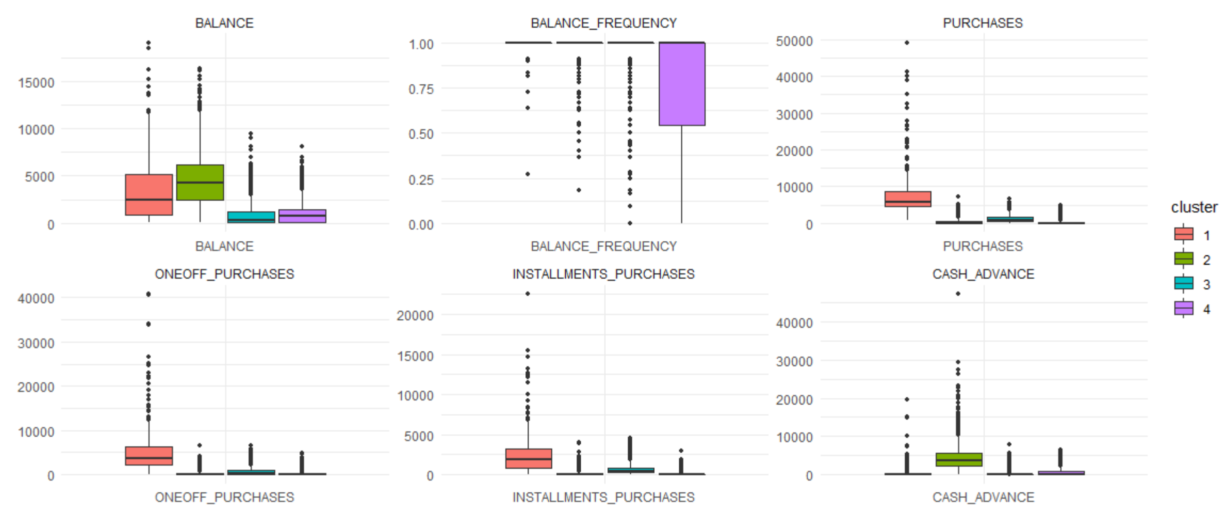
#In order to iterpretate the clusters grouped boxplots will be used for all 4 cluster for each data in a columns.

c = cc_data

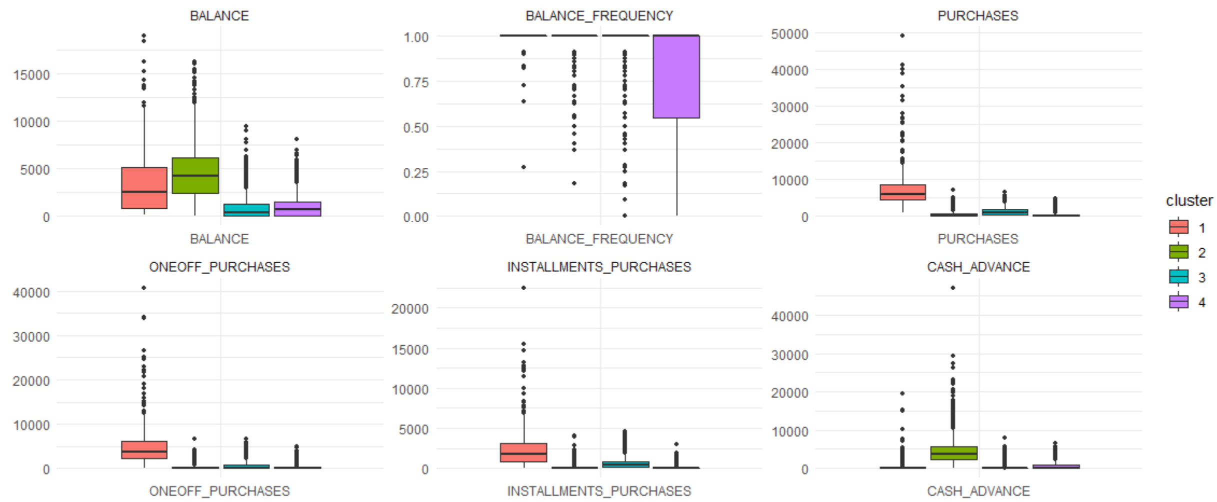
#The melt() function is used to convert a data frame with several measurement columns into a data frame in this canonical format, which has one row for every observed (measured) value.

```
c$cluster = fit_km$cluster

c_plots = melt(c, id.var = "cluster")

c_plots$cluster = as.factor(c$cluster)
```

c_plots %>%
ggplot(aes(x = variable, y = value)) +
geom_boxplot(aes(fill = cluster), outlier.size = 1) +
facet_wrap_paginate(~ variable, scales = "free", ncol = 3, nrow = 2, page = 1) +
labs(x = NULL, y = NULL) +
theme_minimal()



c_plots %>%
ggplot(aes(x = variable, y = value)) +
geom_boxplot(aes(fill = cluster), outlier.size = 1) +
facet_wrap_paginate(~ variable, scales = "free", ncol = 3, nrow = 2, page = 2) +
labs(x = NULL, y = NULL) +
theme_minimal()



c_plots %>%
ggplot(aes(x = variable, y = value)) +
geom_boxplot(aes(fill = cluster), outlier.size = 1) +
facet_wrap_paginate(~ variable, scales = "free", ncol = 3, nrow = 2, page = 3) +
labs(x = NULL, y = NULL) +
theme_minimal()

From the analysing the box plot for all 4 cluster, the clusters can be interpreted as follows (marketing wise) from my point of view:

Cluster 1: Frequent user, with (probably) lower income that spends his money mostly on consumer goods.

Cluster 2: Frequent user, with (probably) higher income that spends his money mostly on consumer goods.

Cluster 3: Mid to rare users, with (probably) mid to high income which spends his money more for higher priced products with longterm use.

Cluster 4: Rare user, with (probably) mid to low income which spends his money more on consumer goods

To Run R file..
open the Rstudio take R file which I have submitted save with some folder location with the dataset.
Run the whole code.

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. An Introduction to Statistical Learning. Vol. 6. Springer.

[https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/?# (https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/)](https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/)

[https://scikit-learn.org/stable/modules/clustering.html#clustering (https://scikit-learn.org/stable/modules/clustering.html#clustering)](https://scikit-learn.org/stable/modules/clustering.html#clustering)

Wickham, Hadley. 2009. Ggplot2: Elegant Graphics for Data Analysis. Springer Science & Business Media.

In [ ]:

```
1
```