R Notebook

Code

Assignment 06

Baran Gulmez
2534089

Load Libraries

Hide

```
library(mlr)
library(tidyverse)
library(DataExplorer)
library(factoextra)
library(dendextend)
library(reshape2)
library(ggforce)
library(cluster)
library(corrplot)
library(ggplot2)
library(ggpubr) # this might be conflicting (select function)
library(MASS) # this might be conflicting (select function)
              # MASS library is used isoMDS()
library(conflicted)
library(factoextra) # for fviz
library(clusterCrit) # cluster validation (intCriteria function)
library(clValid) # (clValid function)
```

Read Dataset

The sample Dataset summarizes the usage behavior of about 9000 active credit card holders during the last 6 months.

CUSTID : Identification of Credit Card holder (Categorical)
BALANCE : Balance amount left in their account to make purchases
BALANCEFREQUENCY : How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated)
PURCHASES : Amount of purchases made from account
ONEOFFPURCHASES : Maximum purchase amount done in one-go
INSTALLMENTSPURCHASES : Amount of purchase done in installment
CASHADVANCE : Cash in advance given by the user
PURCHASESFREQUENCY : How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
ONEOFFPURCHASESFREQUENCY : How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)
PURCHASESINSTALLMENTSFREQUENCY : How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)
CASHADVANCEFREQUENCY : How frequently the cash in advance being paid
CASHADVANCETRX : Number of Transactions made with “Cash in Advanced”
PURCHASESTRX : Numbe of purchase transactions made
CREDITLIMIT : Limit of Credit Card for user
PAYMENTS : Amount of Payment done by user

MINIMUM_PAYMENTS : Minimum amount of payments made by user
PRCFULLPAYMENT : Percent of full payment paid by user
TENURE : Tenure of credit card service for user

Hide

```
dat = read.csv("input/CC GENERAL.csv",
               stringsAsFactors = F,
               na.strings = c(" "))
```

Basic Analysis

This dataset is quite useful since because of two reasons. The first is that the dataset does not need preprocessing since all features are numeric. The second is that there are more than enough data interms of both number of samples and number of features. ## Glimpse

Hide

```
glimpse(dat)
```

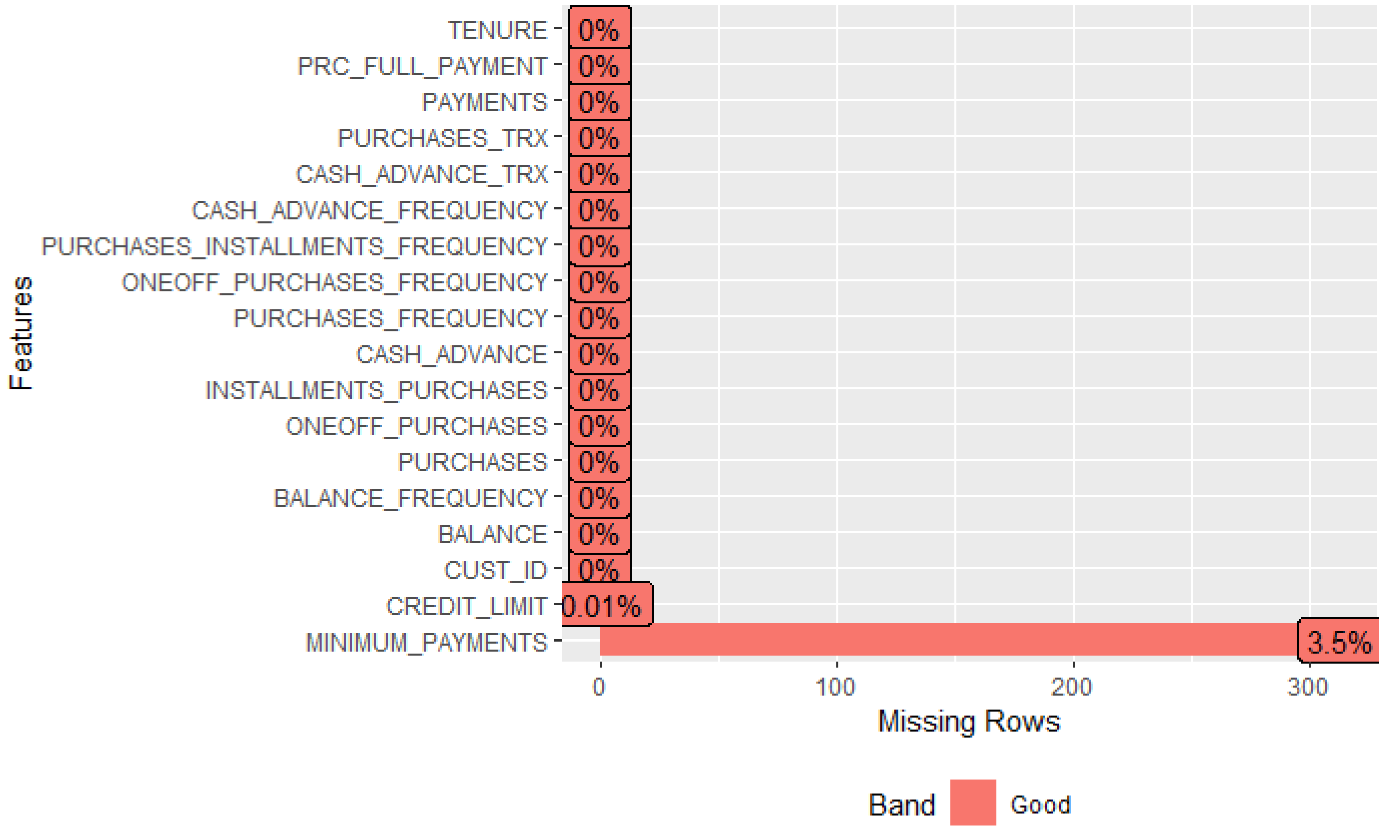
```
Rows: 8,950
Columns: 18
$ CUST_ID                          <chr> "C10001", "C10002", "C10003", "C10004", "C10005", "C10006", "C
10007", "C10008", "C10009", "C10010~
$ BALANCE                          <dbl> 40.90075, 3202.46742, 2495.14886, 1666.67054, 817.71434, 1809.
82875, 627.26081, 1823.65274, 1014.~
$ BALANCE_FREQUENCY                <dbl> 0.818182, 0.909091, 1.000000, 0.636364, 1.000000, 1.000000, 1.
000000, 1.000000, 1.000000, 0.54545~
$ PURCHASES                        <dbl> 95.40, 0.00, 773.17, 1499.00, 16.00, 1333.28, 7091.01, 436.20,
861.49, 1281.60, 920.12, 1492.18, ~
$ ONEOFF_PURCHASES                 <dbl> 0.00, 0.00, 773.17, 1499.00, 16.00, 0.00, 6402.63, 0.00, 661.4
9, 1281.60, 0.00, 1492.18, 2500.23,~
$ INSTALLMENTS_PURCHASES           <dbl> 95.40, 0.00, 0.00, 0.00, 0.00, 1333.28, 688.38, 436.20, 200.0
0, 0.00, 920.12, 0.00, 717.76, 1717.~
$ CASH_ADVANCE                     <dbl> 0.00000, 6442.94548, 0.00000, 205.78802, 0.00000, 0.00000, 0.0
0000, 0.00000, 0.00000, 0.00000, 0.~
$ PURCHASES_FREQUENCY              <dbl> 0.166667, 0.000000, 1.000000, 0.083333, 0.083333, 0.666667, 1.
000000, 1.000000, 0.333333, 0.16666~
$ ONEOFF_PURCHASES_FREQUENCY       <dbl> 0.000000, 0.000000, 1.000000, 0.083333, 0.083333, 0.000000, 1.
000000, 0.000000, 0.083333, 0.16666~
$ PURCHASES_INSTALLMENTS_FREQUENCY <dbl> 0.083333, 0.000000, 0.000000, 0.000000, 0.000000, 0.583333, 1.
000000, 1.000000, 0.250000, 0.00000~
$ CASH_ADVANCE_FREQUENCY           <dbl> 0.000000, 0.250000, 0.000000, 0.083333, 0.000000, 0.000000, 0.
000000, 0.000000, 0.000000, 0.00000~
$ CASH_ADVANCE_TRX                 <int> 0, 4, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 3, 0, 0, 0, 0,
6, 0, 13, 4, 0, 5, 0, 16, 0, 10, 2~
$ PURCHASES_TRX                    <int> 2, 0, 12, 1, 1, 8, 64, 12, 5, 3, 12, 6, 26, 26, 0, 11, 0, 8,
9, 12, 8, 92, 17, 13, 0, 12, 2, 12, ~
$ CREDIT_LIMIT                     <dbl> 1000, 7000, 7500, 7500, 1200, 1800, 13500, 2300, 7000, 11000,
1200, 2000, 3000, 7500, 3000, 8000,~
$ PAYMENTS                         <dbl> 201.8021, 4103.0326, 622.0667, 0.0000, 678.3348, 1400.0578, 63
54.3143, 679.0651, 688.2786, 1164.7~
$ MINIMUM_PAYMENTS                 <dbl> 139.50979, 1072.34022, 627.28479, NA, 244.79124, 2407.24604, 1
98.06589, 532.03399, 311.96341, 100~
$ PRC_FULL_PAYMENT                 <dbl> 0.000000, 0.222222, 0.000000, 0.000000, 0.000000, 0.000000, 1.
000000, 0.000000, 0.000000, 0.00000~
$ TENURE                           <int> 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 1
2, 8, 12, 12, 12, 12, 12, 12, 12, 11~
```

Missing Data

An insignificant portion of the data is missing.

Hide

```
plot_missing(dat)
```



Summaryy

Hide

```
summary(dat)
```

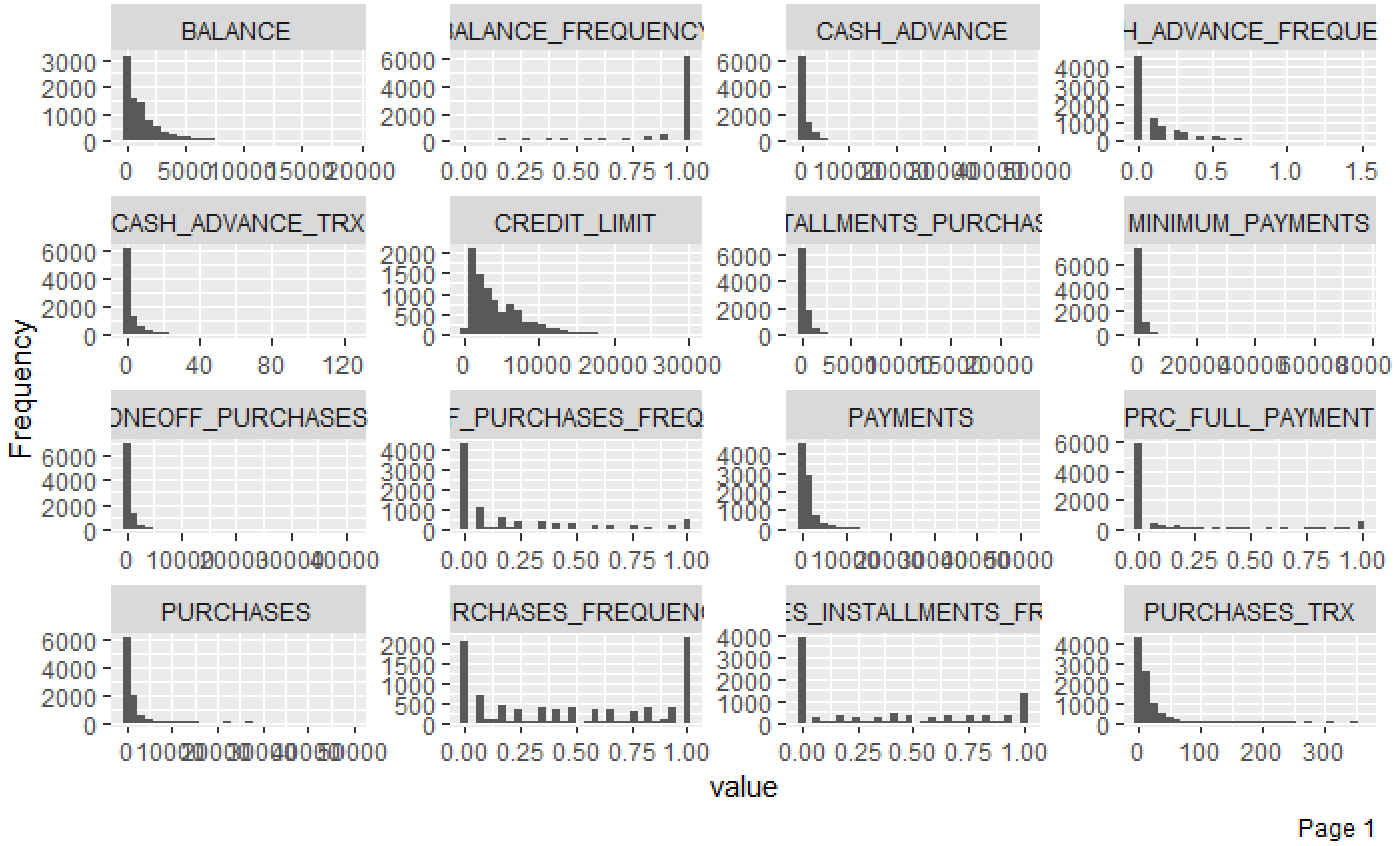
```
   CUST_ID             BALANCE        BALANCE_FREQUENCY    PURCHASES        ONEOFF_PURCHASES   INSTALLMEN
TS_PURCHASES  CASH_ADVANCE    
 Length:8950        Min.   :    0.0   Min.   :0.0000    Min.   :    0.00   Min.   :    0.0   Min.   :
0.0        Min.   :    0.0  
 Class :character   1st Qu.:  128.3   1st Qu.:0.8889    1st Qu.:   39.63   1st Qu.:    0.0   1st Qu.:
0.0        1st Qu.:    0.0  
 Mode  :character   Median :  873.4   Median :1.0000    Median :  361.28   Median :   38.0   Median :
89.0        Median :    0.0  
                    Mean   : 1564.5   Mean   :0.8773    Mean   : 1003.20   Mean   :  592.4   Mean   :
411.1        Mean   :  978.9  
                    3rd Qu.: 2054.1   3rd Qu.:1.0000    3rd Qu.: 1110.13   3rd Qu.:  577.4   3rd Qu.:
468.6        3rd Qu.: 1113.8  
                    Max.   :19043.1   Max.   :1.0000    Max.   :49039.57   Max.   :40761.2   Max.   :22
500.0        Max.   :47137.2  
                                                                                                      
                                
 PURCHASES_FREQUENCY ONEOFF_PURCHASES_FREQUENCY PURCHASES_INSTALLMENTS_FREQUENCY CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX  PURCHASES_TRX   
 Min.   :0.00000     Min.   :0.00000           Min.   :0.0000                   Min.   :0.0000        
Min.   :  0.000   Min.   :  0.00  
 1st Qu.:0.08333     1st Qu.:0.00000           1st Qu.:0.0000                   1st Qu.:0.0000        
1st Qu.:  0.000   1st Qu.:  1.00  
 Median :0.50000     Median :0.08333           Median :0.1667                   Median :0.0000        
Median :  0.000   Median :  7.00  
 Mean   :0.49035     Mean   :0.20246           Mean   :0.3644                   Mean   :0.1351        
Mean   :  3.249   Mean   : 14.71  
 3rd Qu.:0.91667     3rd Qu.:0.30000           3rd Qu.:0.7500                   3rd Qu.:0.2222        
3rd Qu.:  4.000   3rd Qu.: 17.00  
 Max.   :1.00000     Max.   :1.00000           Max.   :1.0000                   Max.   :1.5000        
Max.   :123.000   Max.   :358.00  
                                                                                                      
                                  
  CREDIT_LIMIT      PAYMENTS       MINIMUM_PAYMENTS   PRC_FULL_PAYMENT     TENURE     
 Min.   :   50   Min.   :    0.0   Min.   :    0.02   Min.   :0.0000   Min.   : 6.00  
 1st Qu.: 1600   1st Qu.:  383.3   1st Qu.:  169.12   1st Qu.:0.0000   1st Qu.:12.00  
 Median : 3000   Median :  856.9   Median :  312.34   Median :0.0000   Median :12.00  
 Mean   : 4494   Mean   : 1733.1   Mean   :  864.21   Mean   :0.1537   Mean   :11.52  
 3rd Qu.: 6500   3rd Qu.: 1901.1   3rd Qu.:  825.49   3rd Qu.:0.1429   3rd Qu.:12.00  
 Max.   :30000   Max.   :50721.5   Max.   :76406.21   Max.   :1.0000   Max.   :12.00  
 NA's   :1                         NA's   :313                                        
```

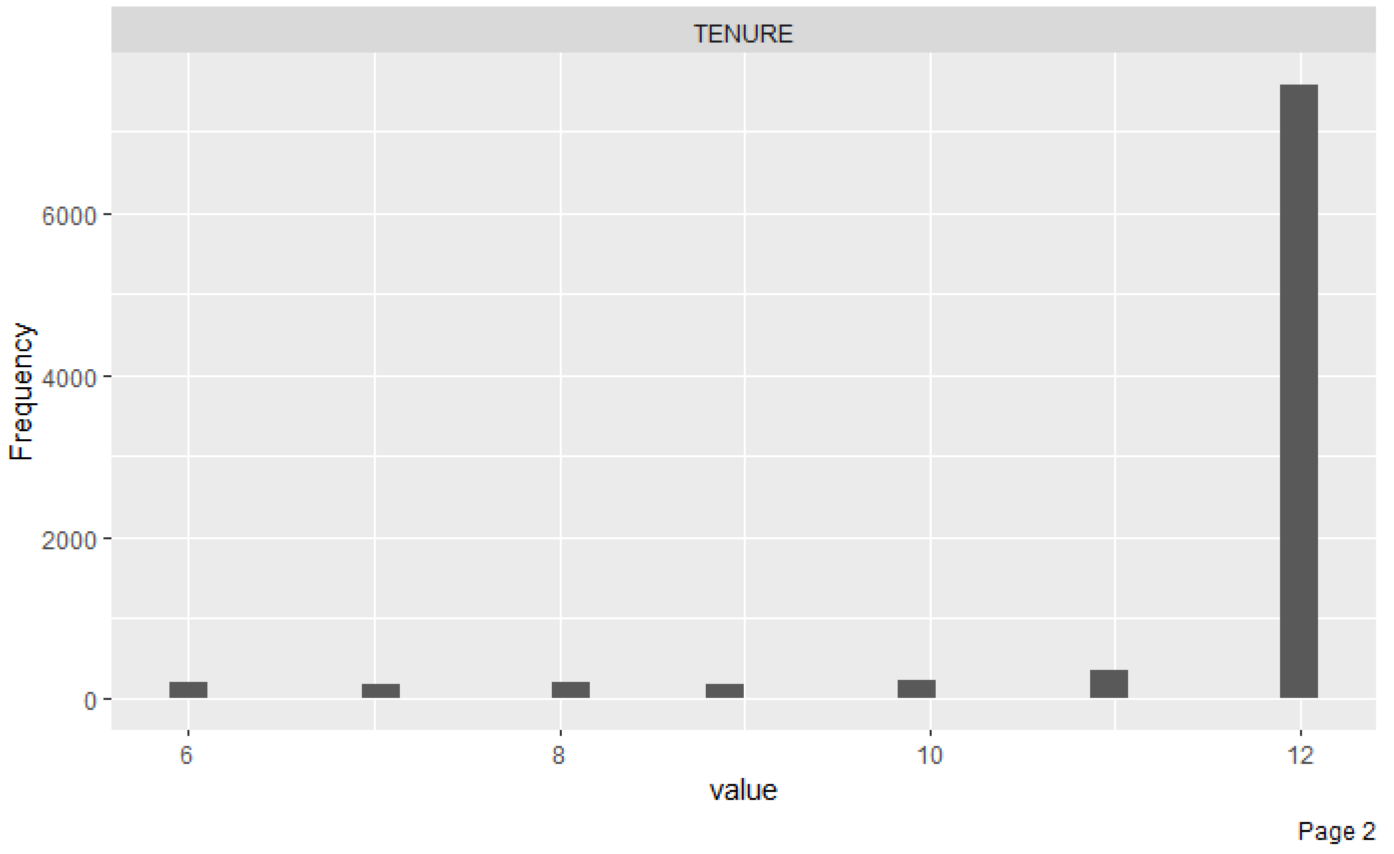
Histograms

Here it is seen that almost all features are skewed.

Hide

```
plot_histogram(dat)
```





Data Reorganization

Missing data deleted and some useless CUST_ID deleted.

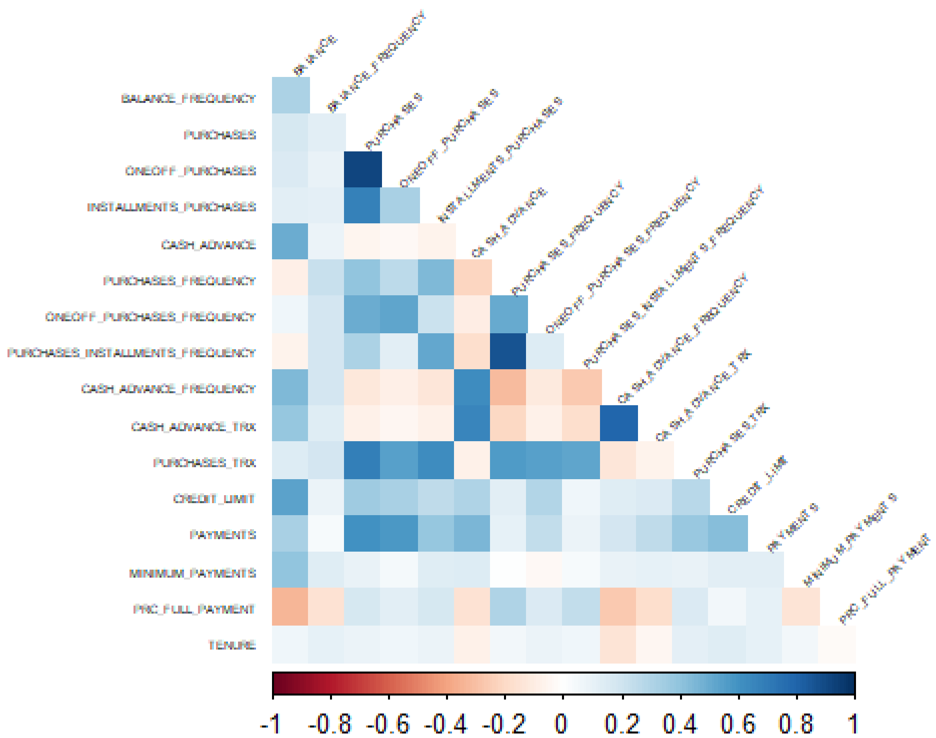
Hide

```
dat_reorg = dat %>%
  dplyr::select(-CUST_ID) %>% # calling select from dplyr to prevent conflict
  drop_na()
#
# shuffle data
if(FALSE){
set.seed(357) # fix seed
dat_reorg_backup <- dat_reorg[sample(nrow(dat_reorg)), ] # generate random index using sample and
dat_reorg = dat_reorg_backup[1:100,1:ncol(dat_reorg_backup)] # choose first 100
}
```

Correlation

Hide

```
corrplot(cor(dat_reorg), diag = FALSE, type = "lower", tl.srt = 45, tl.col = "black", method = 'color',
tl.cex = 0.4)
```



PCA

prcomp() expects the samples to be rows to be columns

Hide

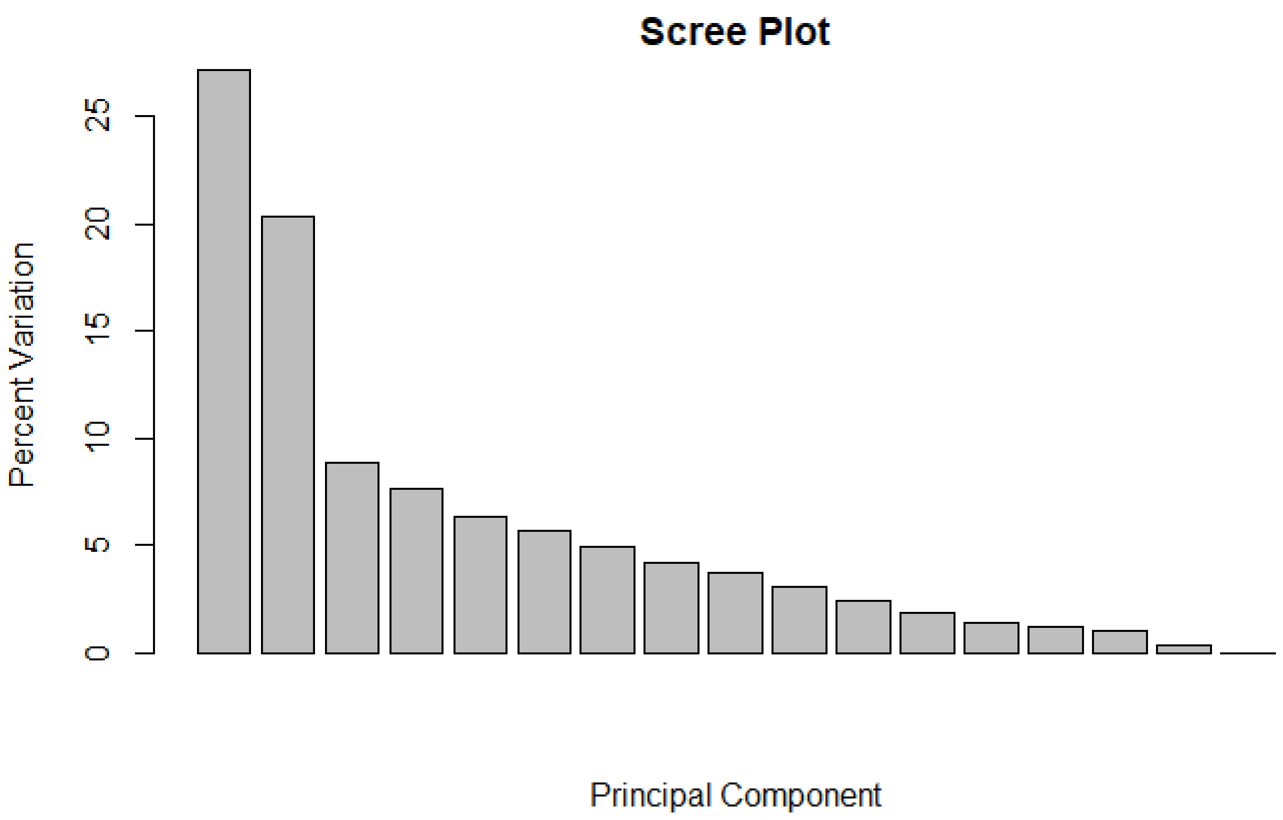
```
dat_reorg_scaled = scale(dat_reorg)
pca <-prcomp(dat_reorg_scaled)
```

PCA Analysis

Scree plot shows how much the principal components are responsible the PCA component is responsible of the variation of the data.

Hide

```
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
barplot(pca.var.per, main="Scree Plot", xlab="Principal Component", ylab="Percent Variation")
```



format the data

Hide

```
pca.data <- data.frame(Sample=rownames(dat_reorg), X=pca$x[,1], Y=pca$x[,2])
pca.data
```

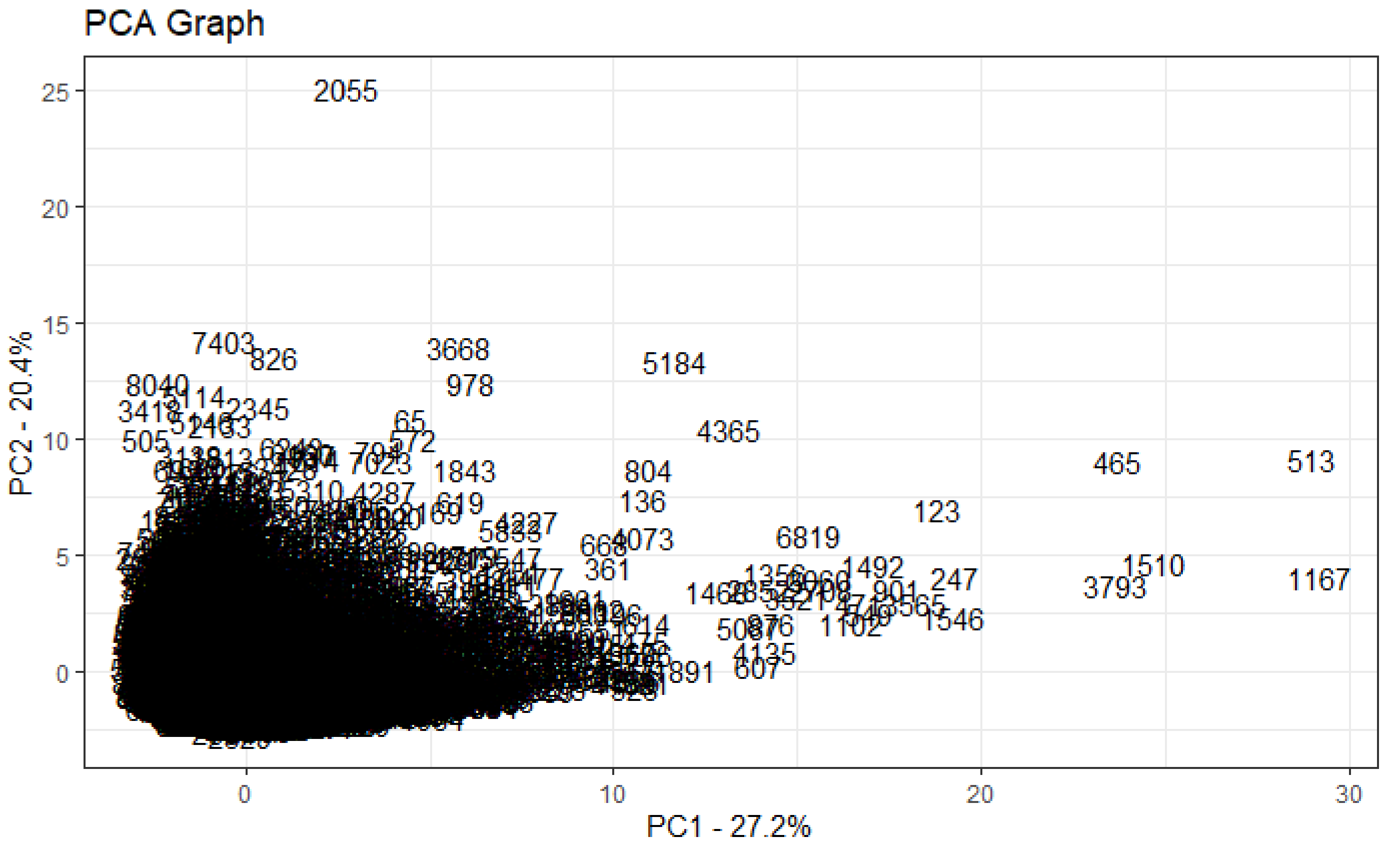
Sample <chr>	X <dbl>	Y <dbl>
1	-1.696297065	-1.122518989
2	-1.215610445	2.435496753
3	0.935799104	-0.385179263
4	-1.614544792	-0.724544205
5	0.223687663	-0.783564452
6	6.265235048	-0.609413897
7	0.261651730	-1.295560288
8	-0.465311641	-0.477670250
9	-0.599646437	-0.408567662
10	0.522740791	-1.312087280

1-10 of 8,636 rows Previous 1 2 3 4 5 6 ... 100 Next

plot the data but with indexes

Hide

```
ggplot(data=pca.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  xlab(paste("PC1 - ", pca.var.per[1], "%", sep="")) +
  ylab(paste("PC2 - ", pca.var.per[2], "%", sep="")) +
  theme_bw() +
  ggtitle("PCA Graph")
```



Hide

```
# just another way to draw first two principal components
# plot(pca$x[,1], pca$x[,2], main="First two PCs", xlab = paste("PC1 - ", pca.var.per[1], "%", sep="")
 , ylab=paste("PC2 - ", pca.var.per[2], "%", sep=""))
```

Negative loading scores push left as positive ones push right.

Hide

```
loading_scores <- pca$rotation[,1]
loading_scores_abs <- abs(loading_scores)
loading_scores
```

```
                    BALANCE                 BALANCE_FREQUENCY                         PURCHASES
ONEOFF_PURCHASES
                 0.09198590                        0.10981218                        0.41215123
0.34677536
     INSTALLMENTS_PURCHASES                      CASH_ADVANCE               PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
                 0.33705564                       -0.03058765                        0.32366488
0.29476135
PURCHASES_INSTALLMENTS_FREQUENCY           CASH_ADVANCE_FREQUENCY                  CASH_ADVANCE_TRX
PURCHASES_TRX
                 0.27722626                       -0.09914541                       -0.05696036
0.39106653
               CREDIT_LIMIT                          PAYMENTS                  MINIMUM_PAYMENTS
PRC_FULL_PAYMENT
                 0.21005184                        0.26372547                        0.05932632
0.13056503
                     TENURE
                 0.07791867
```

Multi-Dimensional Scaling (MDS)

Euclidean and manhattan distances are most widely used distance metrics. Therefore they are chosen as distance metrics. These two distances also usually work the best to my experience.

Hide

```
distEuc.matrix <- stats::dist(dat_reorg_scaled, method="euclidean")
distMnh.matrix <- stats::dist(dat_reorg_scaled, method="manhattan")
```

Classical Multi-Dimensional Scaling

Euclidean Distance

Hide

```
mdsCmdEuc.stuff <- cmdscale(distEuc.matrix, eig=TRUE, x.ret=TRUE)
mdsCmdEuc.var.per <- round(mdsCmdEuc.stuff$eig/sum(mdsCmdEuc.stuff$eig)*100, 1)
# graph
mdsCmdEuc.values <- mdsCmdEuc.stuff$points
mdsCmdEuc.data <- base::data.frame(Sample=rownames(dat_reorg), X=mdsCmdEuc.values[,1], Y=mdsCmdEuc.valu
es[,2])
mdsCmdEuc.data
```

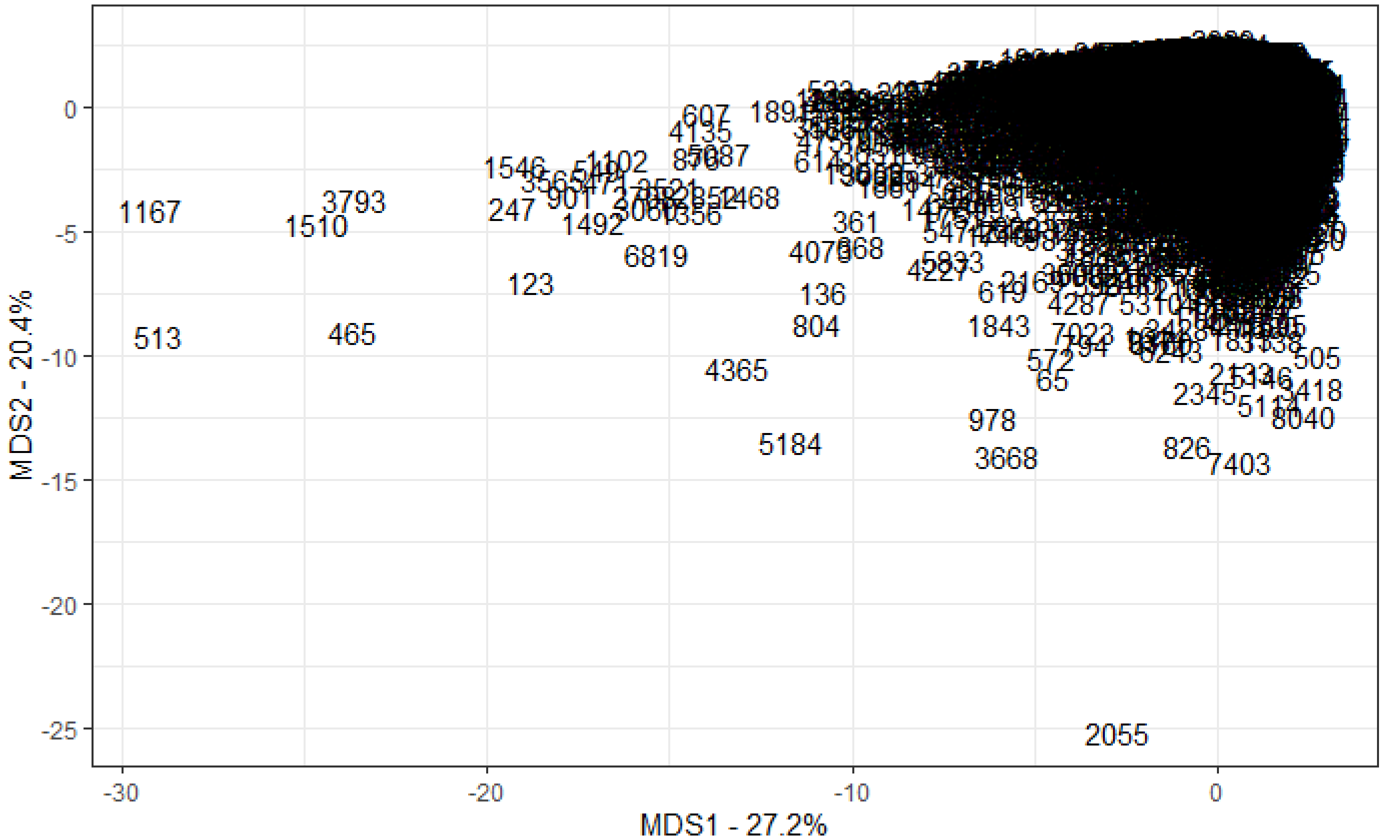
Sample <chr>	X <dbl>	Y <dbl>
1	1.696297065	1.122518989
2	1.215610445	-2.435496753
3	-0.935799104	0.385179263
4	1.614544792	0.724544205
5	-0.223687663	0.783564452
6	-6.265235048	0.609413897
7	-0.261651730	1.295560288

Sample	X	Y
<chr>	<dbl>	<dbl>
8	0.465311641	0.477670250
9	0.599646437	0.408567662
10	-0.522740791	1.312087280

1-10 of 8,636 rows Previous 1 2 3 4 5 6 ... 100 Next

Hide

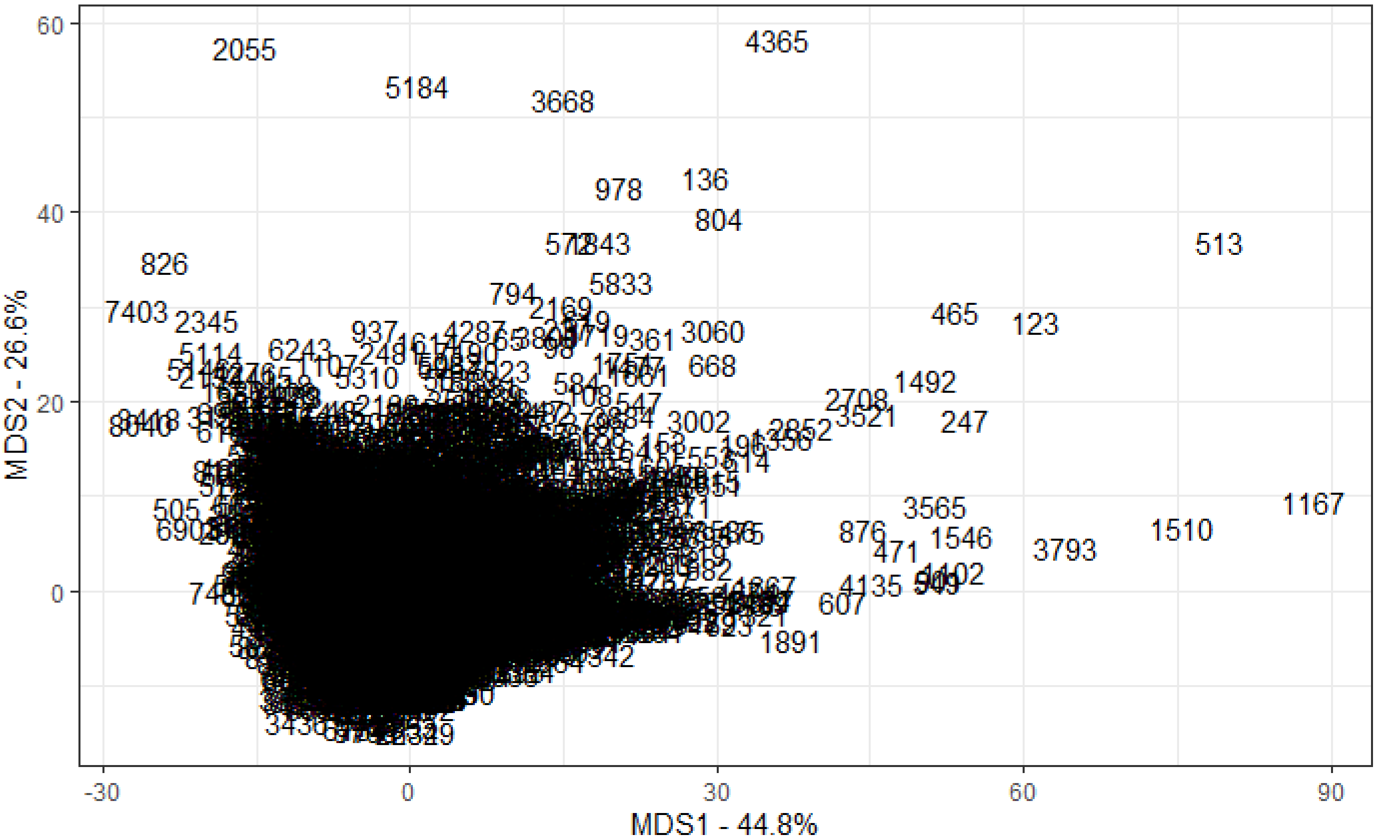
```
ggplot(data=mdsCmdEuc.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  theme_bw() +
  xlab(paste("MDS1 - ", mdsCmdEuc.var.per[1], "%", sep="")) +
  ylab(paste("MDS2 - ", mdsCmdEuc.var.per[2], "%", sep=""))
```



Manhattan Distance

Hide

```
mdsCmdMnh.stuff <- cmdscale(distMnh.matrix, eig=TRUE, x.ret=TRUE)
mdsCmdMnh.var.per <- round(mdsCmdMnh.stuff$eig/sum(mdsCmdMnh.stuff$eig)*100, 1)
# graph
mdsCmdMnh.values <- mdsCmdMnh.stuff$points
mdsCmdMnh.data <- data.frame(Sample=rownames(dat_reorg),
  X=mdsCmdMnh.values[,1],
  Y=mdsCmdMnh.values[,2])
ggplot(data=mdsCmdMnh.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  theme_bw() +
  xlab(paste("MDS1 - ", mdsCmdMnh.var.per[1], "%", sep="")) +
  ylab(paste("MDS2 - ", mdsCmdMnh.var.per[2], "%", sep=""))
```



Metric Multi-Dimesional Scaling

Euclidean Distance

Hide

```
mdsSamEuc.stuff <- sammon(distEuc.matrix)
```

```
Initial stress        : 0.18811
stress after   0 iters: 0.18811
```

Hide

```
mdsSamEuc.var.per <- round(mdsSamEuc.stuff$eig/sum(mdsSamEuc.stuff$eig)*100, 1)
# graph
mdsSamEuc.values <- mdsSamEuc.stuff$points
mdsSamEuc.data <- base::data.frame(Sample=rownames(dat_reorg), X=mdsSamEuc.values[,1], Y=mdsSamEuc.valu
es[,2])
mdsSamEuc.data
```

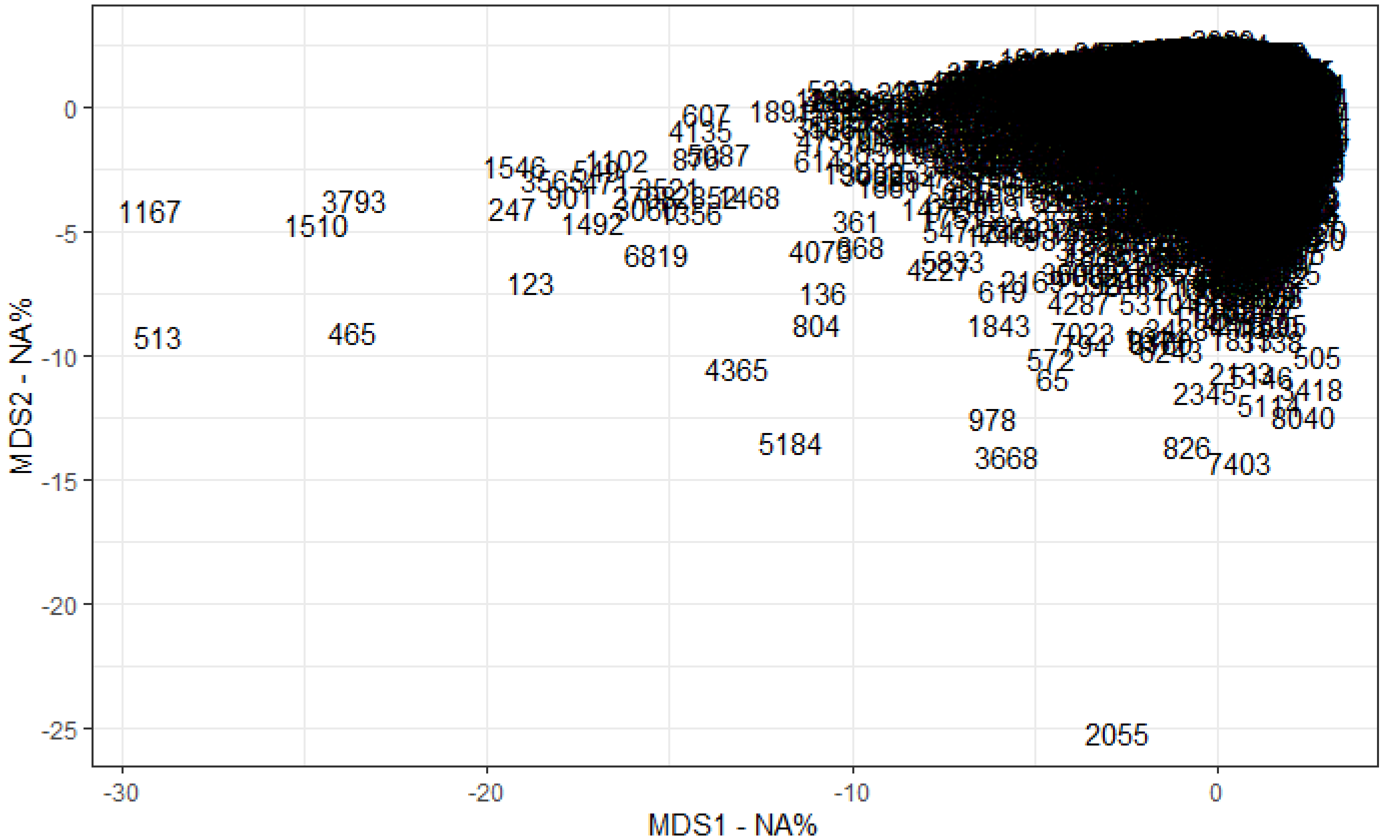
Sample <chr>	X <dbl>	Y <dbl>
1	1.696297065	1.122518989
2	1.215610445	-2.435496753
3	-0.935799104	0.385179263
4	1.614544792	0.724544205
5	-0.223687663	0.783564452
6	-6.265235048	0.609413897
7	-0.261651730	1.295560288

Sample <chr>	X <dbl>	Y <dbl>
8	0.465311641	0.477670250
9	0.599646437	0.408567662
10	-0.522740791	1.312087280

1-10 of 8,636 rows Previous **1** 2 3 4 5 6 ... 100 Next

Hide

```
ggplot(data=mdsSamEuc.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  theme_bw() +
  xlab(paste("MDS1 - ", mdsSamEuc.var.per[1], "%", sep="")) +
  ylab(paste("MDS2 - ", mdsSamEuc.var.per[2], "%", sep=""))
```



Manhattan Distance

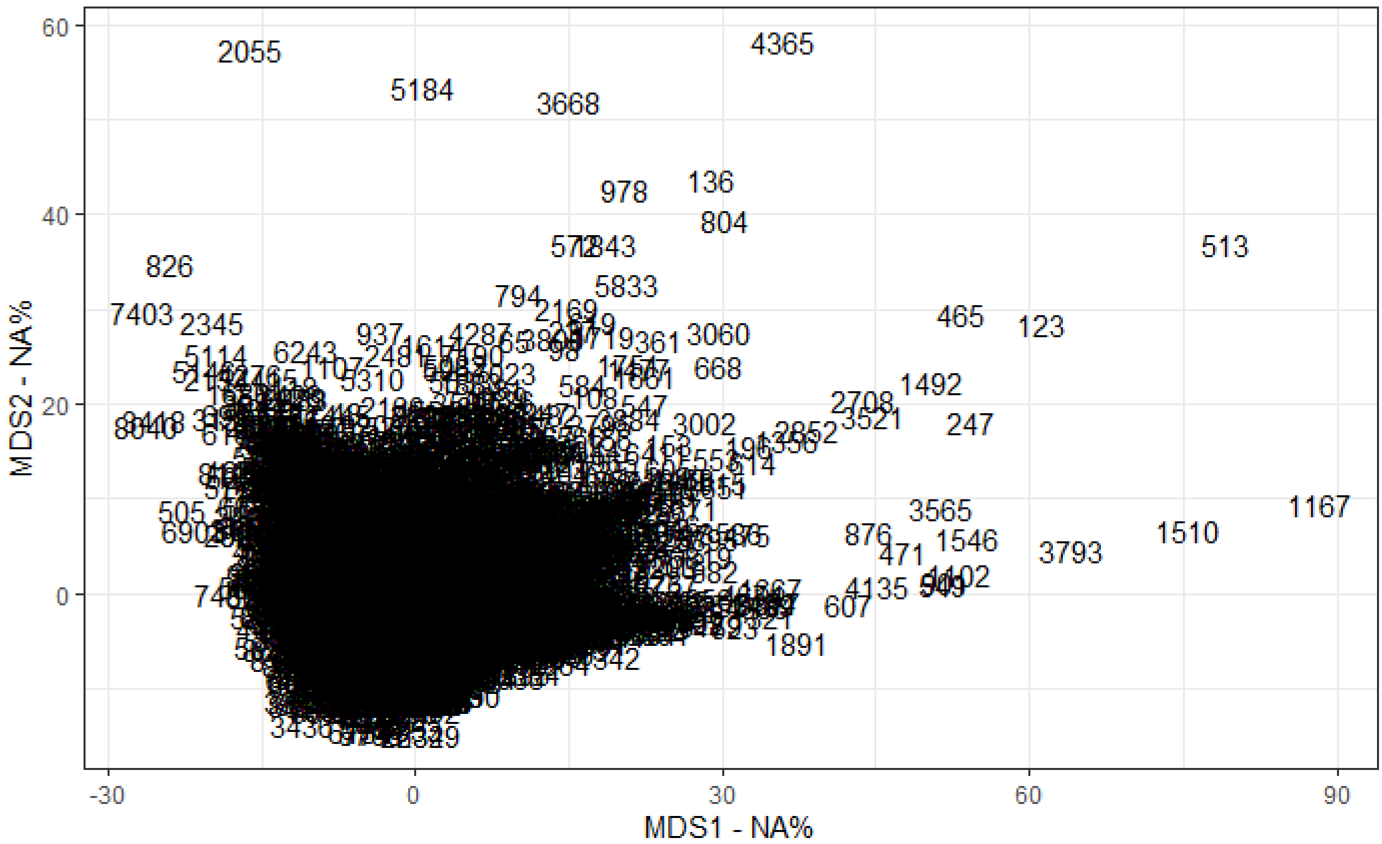
Hide

```
mdsSamMnh.stuff <- sammon(distMnh.matrix)
```

```
Initial stress        : 0.08243
stress after   0 iters: 0.08243
```

Hide

```
mdsSamMnh.var.per <- round(mdsSamMnh.stuff$eig/sum(mdsSamMnh.stuff$eig)*100, 1)
# graph
mdsSamMnh.values <- mdsSamMnh.stuff$points
mdsSamMnh.data <- data.frame(Sample=rownames(dat_reorg),
  X=mdsSamMnh.values[,1],
  Y=mdsSamMnh.values[,2])
ggplot(data=mdsSamMnh.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  theme_bw() +
  xlab(paste("MDS1 - ", mdsSamMnh.var.per[1], "%", sep="")) +
  ylab(paste("MDS2 - ", mdsSamMnh.var.per[2], "%", sep=""))
```



Non-Metric Multi-Dimensional Scaling

Euclidean Distance

Hide

```
mdsIsoEuc.stuff <- isoMDS(distEuc.matrix)
```

```
initial  value 28.346012
iter   5 value 17.743101
iter  10 value 14.923771
final  value 14.886499
converged
```

Hide

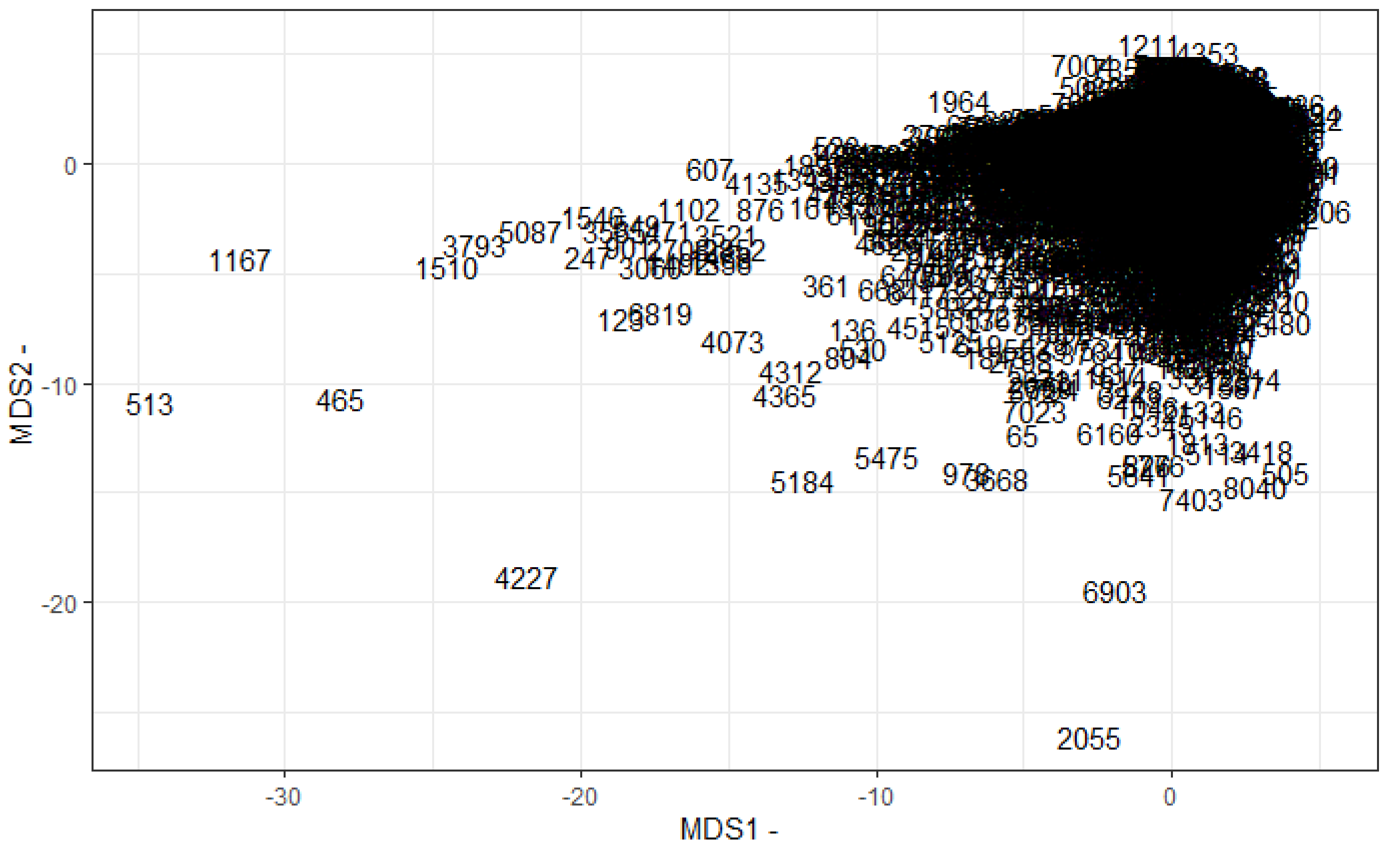
```
# graph
mdsIsoEuc.values <- mdsIsoEuc.stuff$points
mdsIsoEuc.data <- data.frame(Sample=rownames(dat_reorg),
  X=mdsIsoEuc.values[,1],
  Y=mdsIsoEuc.values[,2])
mdsIsoEuc.data
```

Sample <chr>	X <dbl>	Y <dbl>
1	1.280355162	0.706059168
2	1.053834590	-1.813733180
3	-1.370248828	-0.140027266
4	1.258167859	0.432953701
5	-0.067202264	0.521058485
6	-6.106103234	0.760970045
7	-0.261048357	1.090521517
8	0.451507023	0.240673661
9	0.841373045	0.257397005
10	-0.431128703	1.150794713

1-10 of 8,636 rows Previous **1** 2 3 4 5 6 ... 100 Next

Hide

```
ggplot(data=mdsIsoEuc.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  theme_bw() +
  xlab(paste("MDS1 - ")) +
  ylab(paste("MDS2 - "))
```



Manhattan Distance

Hide

```
mdsIsoMnh.stuff <- isoMDS(distMnh.matrix)
```

```
initial  value 22.024305
final  value 22.020486
converged
```

Hide

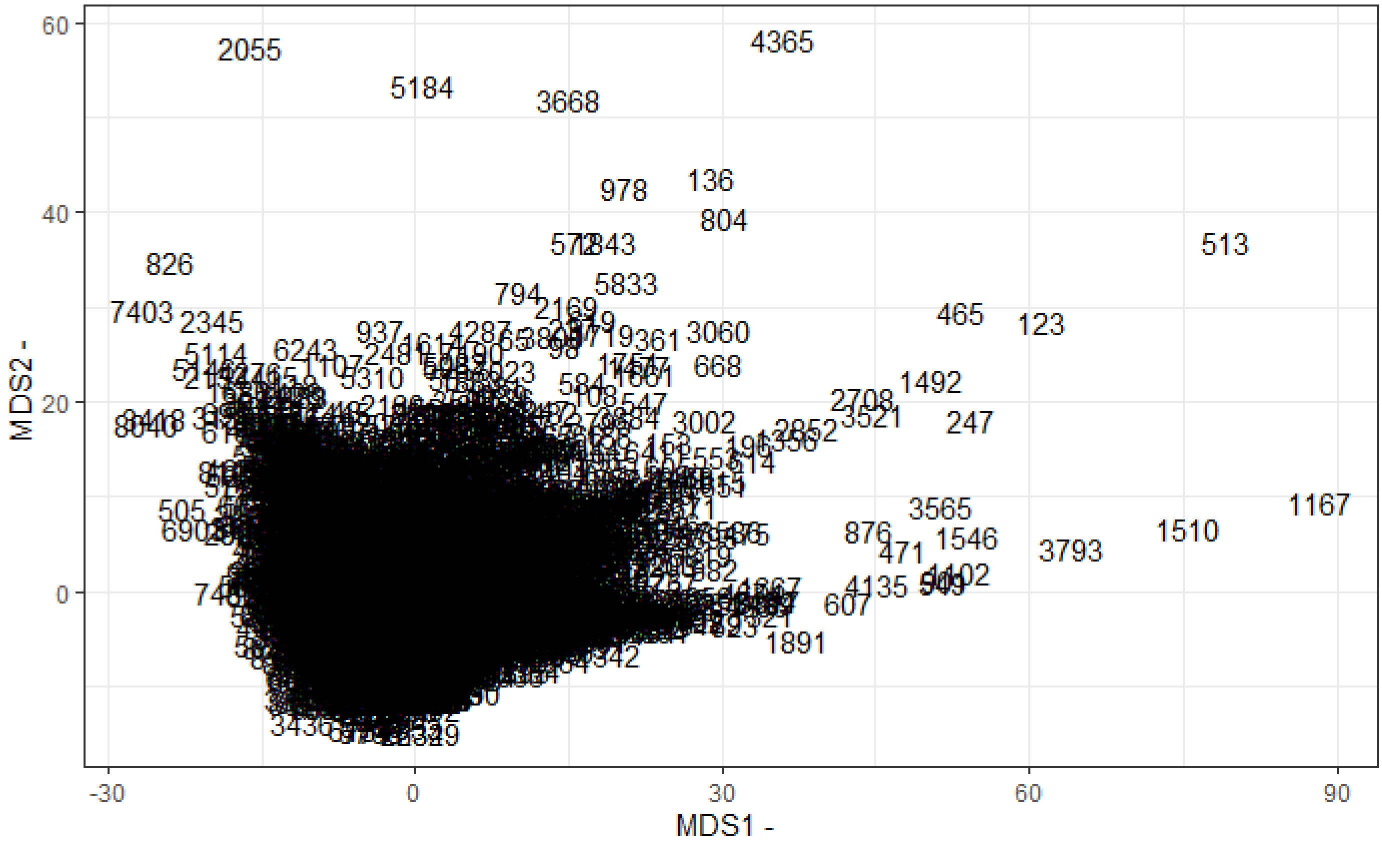
```
# graph
mdsIsoMnh.values <- mdsIsoMnh.stuff$points
mdsIsoMnh.data <- data.frame(Sample=rownames(dat_reorg),
  X=mdsIsoMnh.values[,1],
  Y=mdsIsoMnh.values[,2])
mdsIsoMnh.data
```

Sample <chr>	X <dbl>	Y <dbl>
1	-4.12217468	-4.636481458
2	-7.06373977	5.807594099
3	3.85313180	1.463960055
4	-3.71610128	-2.772121054
5	2.29220920	-0.495867094
6	22.81053019	-1.216820052
7	2.96938601	-1.460609252
8	-0.12917346	-0.424079563
9	-1.86588575	-2.271565354
10	3.80709155	-1.360532928

1-10 of 8,636 rows Previous **1** 2 3 4 5 6 ... 100 Next

Hide

```
ggplot(data=mdsIsoMnh.data, aes(x=X, y=Y, label=Sample)) +
  geom_text() +
  theme_bw() +
  xlab(paste("MDS1 - ")) +
  ylab(paste("MDS2 - "))
```



MDS Results

There is not much difference between euclidean and manhattan distances. Both Classical MDS and Non-Metric MDS support this. When Classical MDS and Non-Metric MDS are compared, classical MDS separates the samples into a wider spectrum which is better. This will help get more distinct groups during clustering.

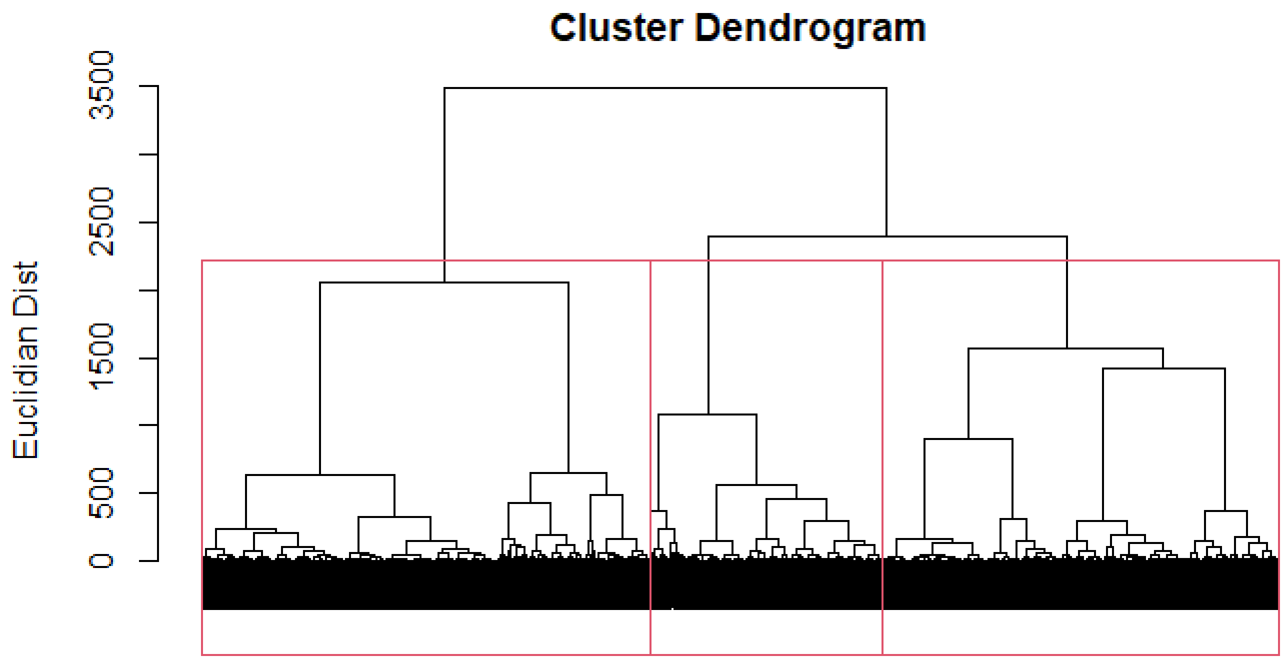
Clustering

The reason for me to seperate the data into 3 cluster is totally intuitively. I just assumed that people would be from low, mid and high income. But inspecting hierarchical clustering, there could be 4-6 clusters ideally.

Hierarchical

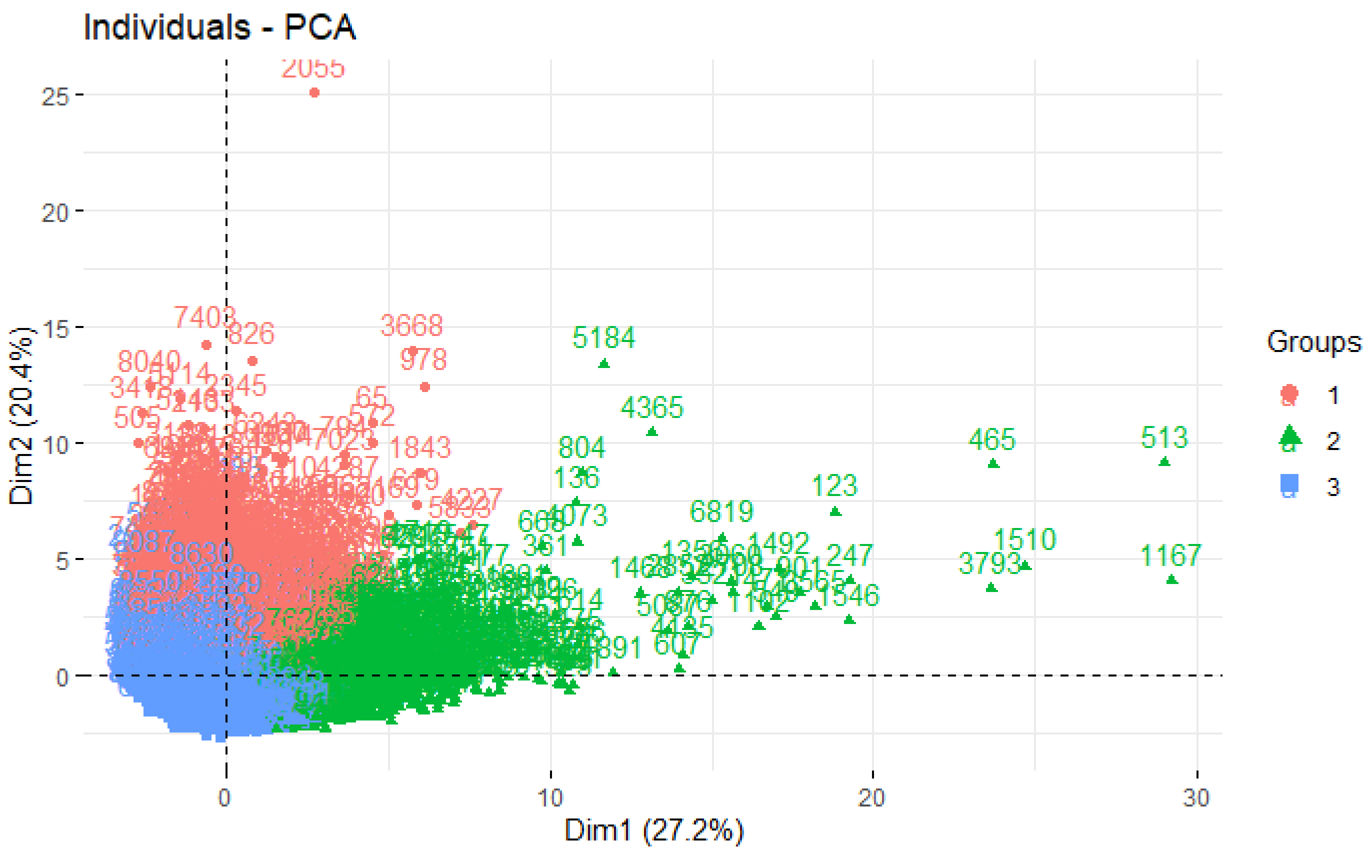
Hide

```
clustHrc = hclust(distEuc.matrix , method = "ward.D")
plot(clustHrc, labels = FALSE, sub = "", xlab = "", ylab = "Euclidian Dist")
rect.hclust(clustHrc, k = 3)
```



Hide

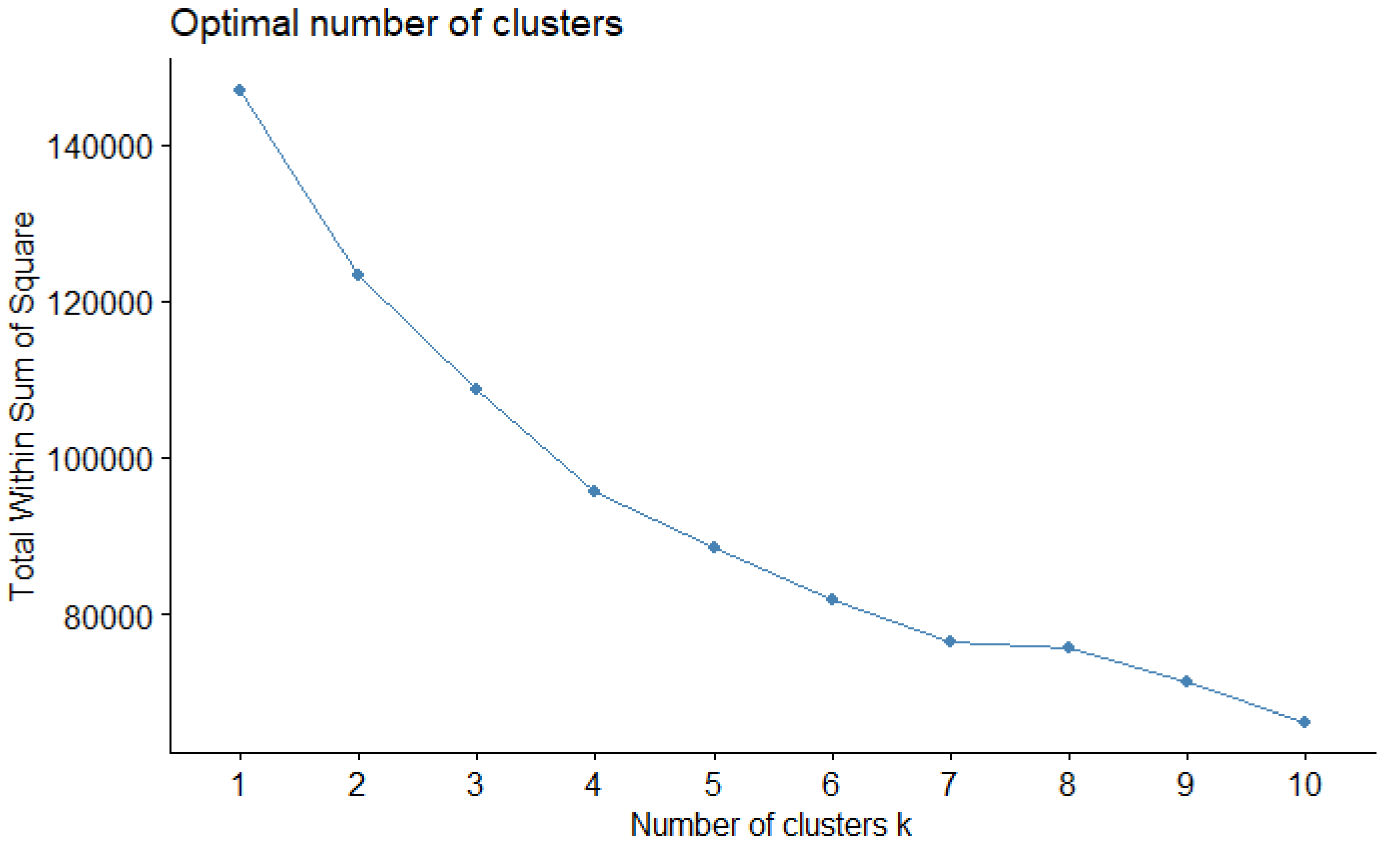
```
clustHrcClusts = cutree(clustHrc, k = 3)
fviz_pca_ind(pca, habillage = clustHrcClusts)
```



K-Means

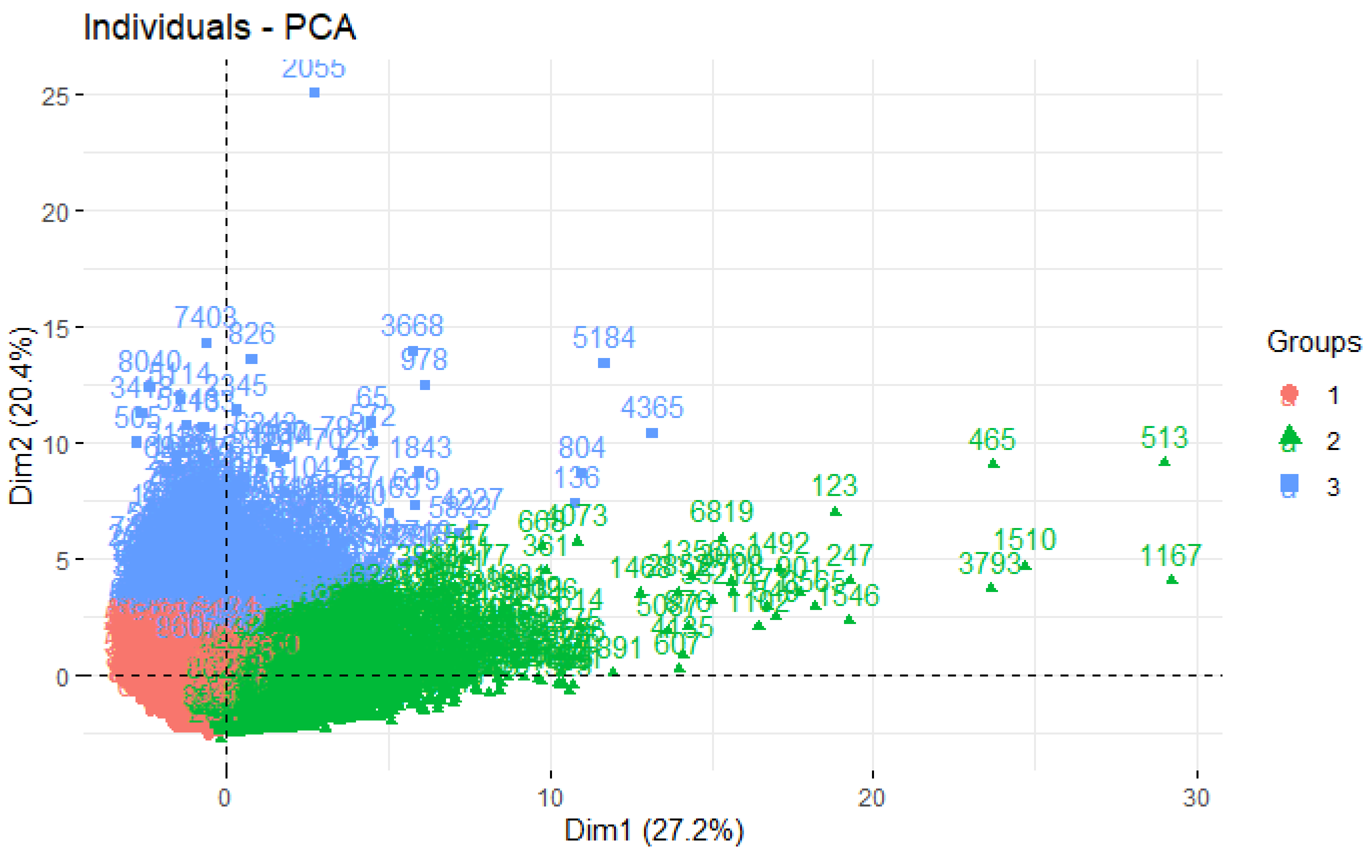
Hide

```
# draw optimal number of cluster
fviz_nbclust(dat_reorg_scaled, kmeans, method = "wss", k.max = 10)
```



Hide

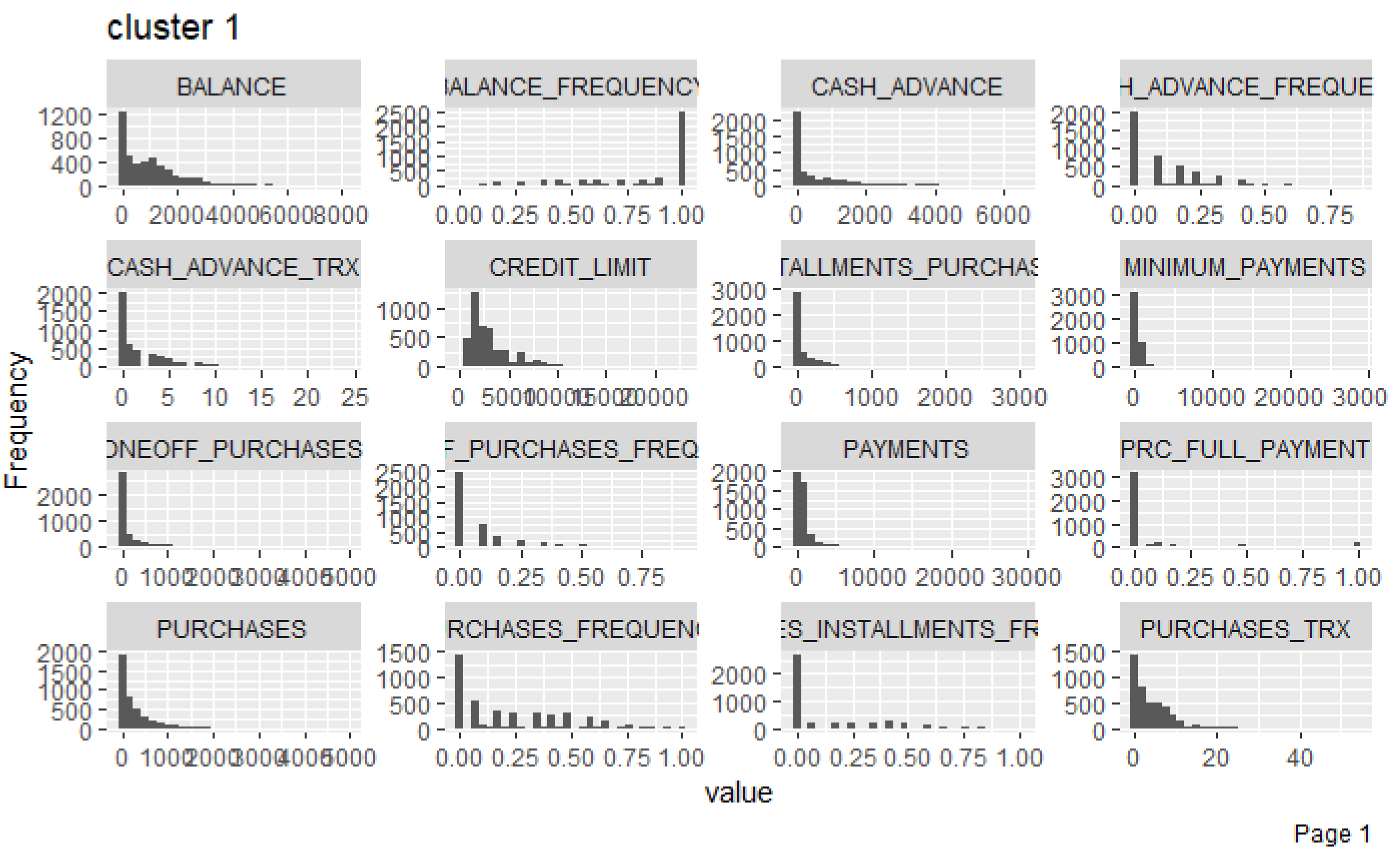
```
clustKm = kmeans(dat_reorg_scaled, centers = 3)
fviz_pca_ind(pca, habillage = clustKm$cluster)
```

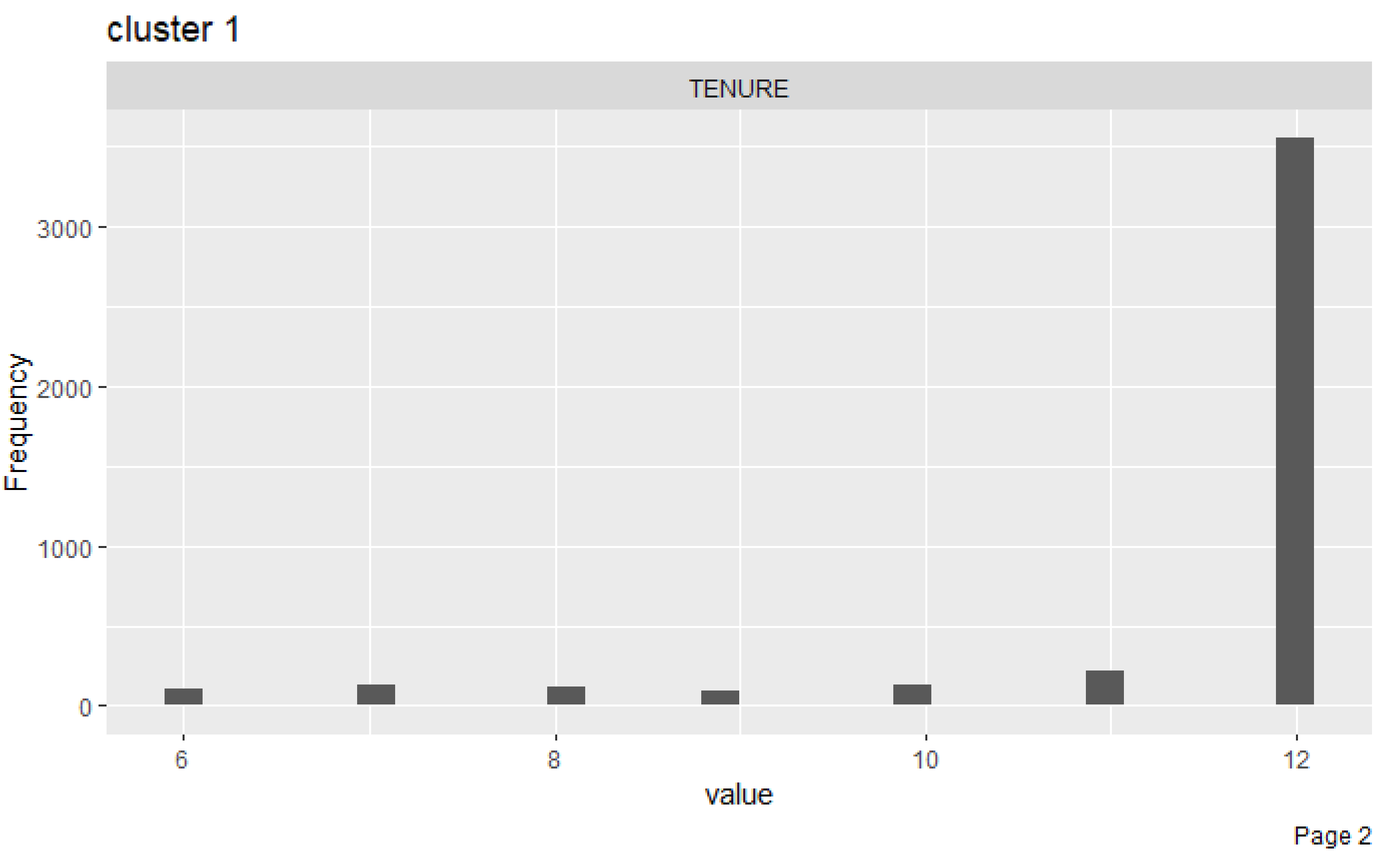


Analyze Groups

Hide

```
clustKm1.index = which(clustKm$cluster==1)
clustKm2.index = which(clustKm$cluster==2)
clustKm3.index = which(clustKm$cluster==3)
clustKm1.dat = dat_reorg[clustKm1.index,1:ncol(dat_reorg)]
clustKm2.dat = dat_reorg[clustKm2.index,1:ncol(dat_reorg)]
clustKm3.dat = dat_reorg[clustKm3.index,1:ncol(dat_reorg)]
plot_histogram(clustKm1.dat, title="cluster 1")
```

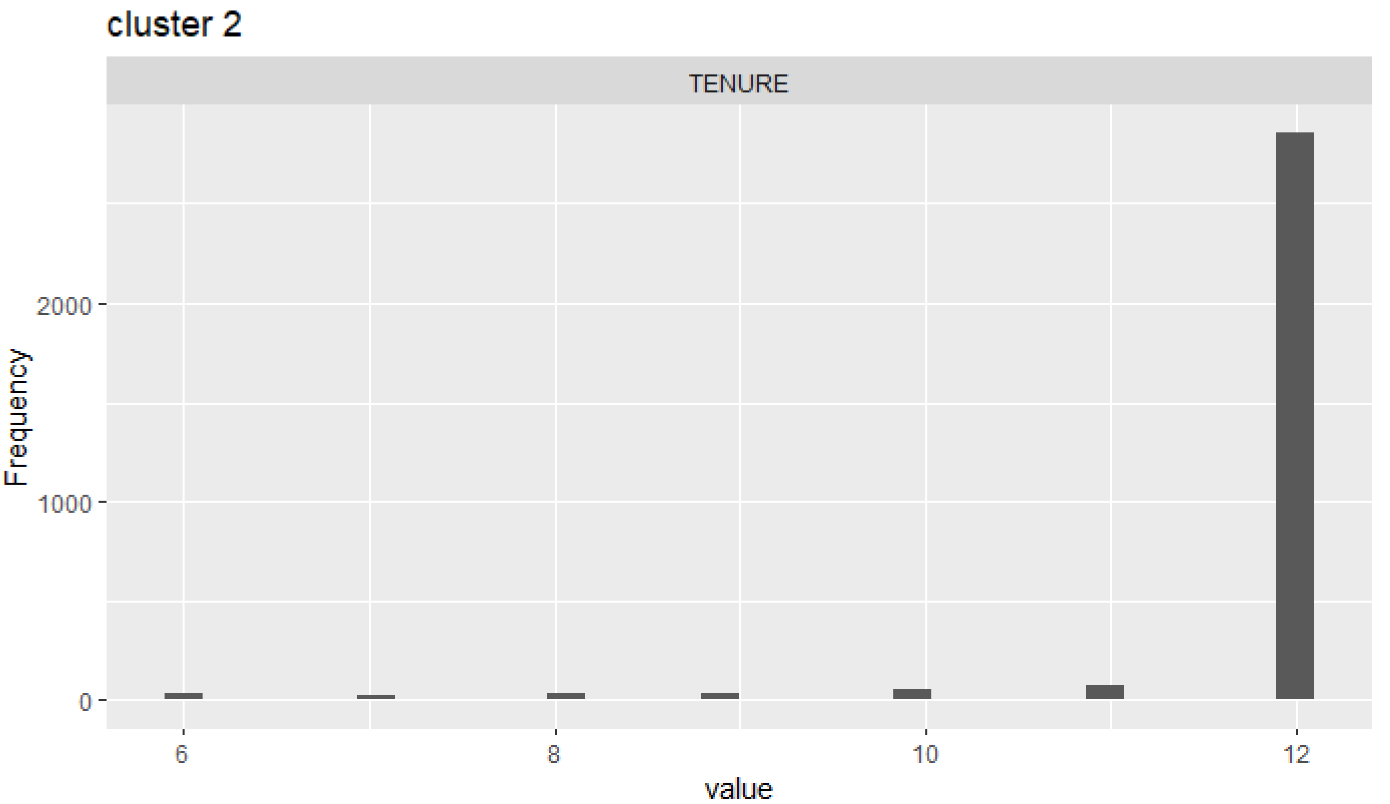




Hide

```
plot_histogram(clustKm2.dat, title="cluster 2")
```

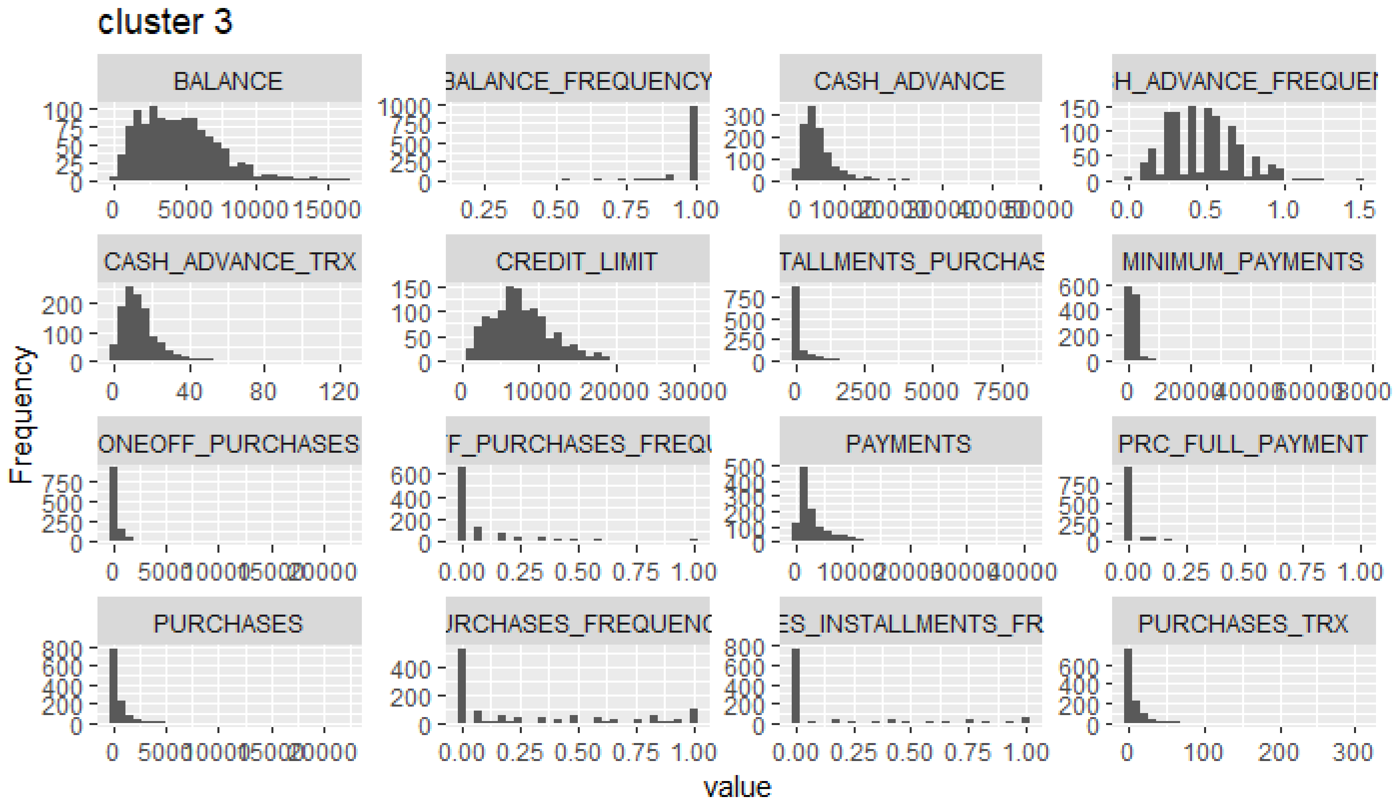
Page 1



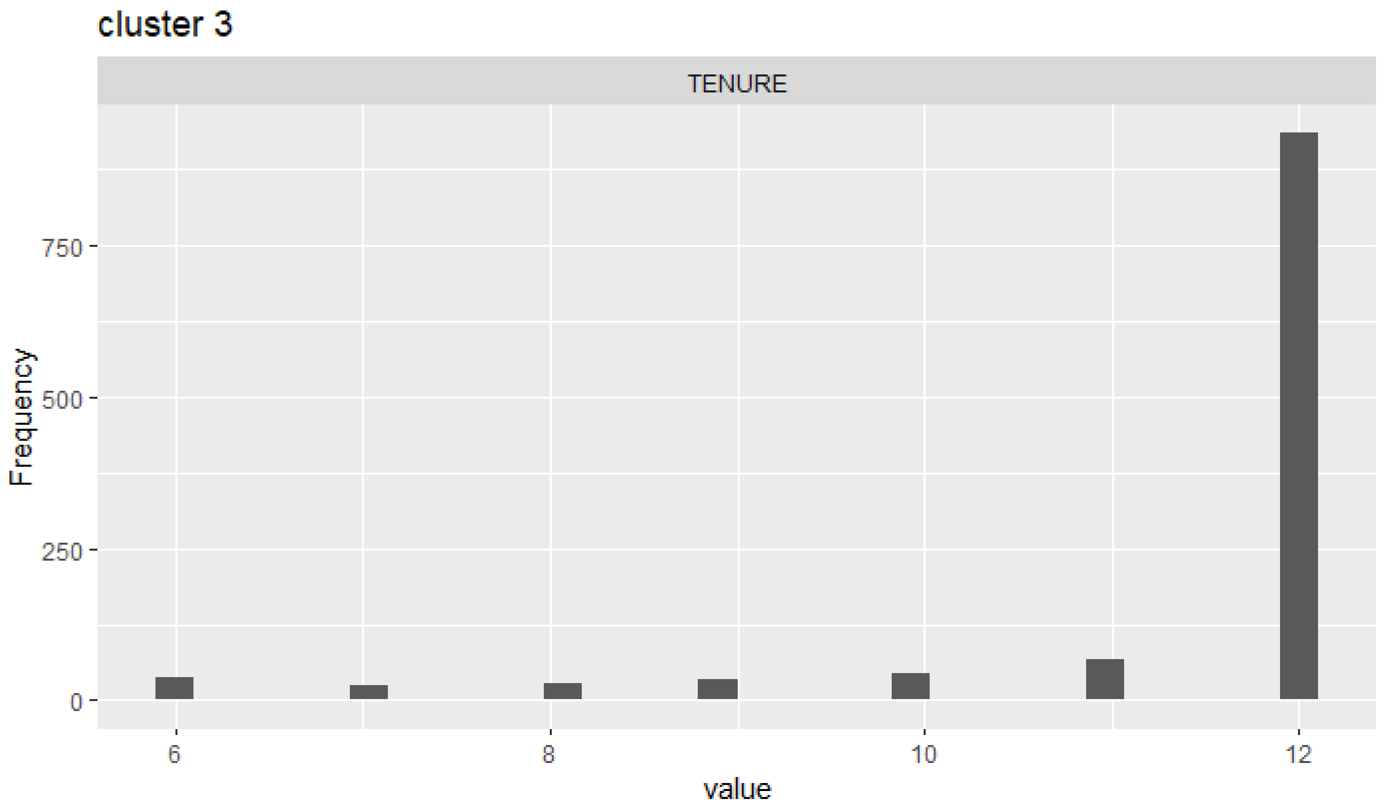
Page 2

Hide

```
plot_histogram(clustKm3.dat, title="cluster 3")
```



Page 1



Page 2

After clustering the samples I inspected each cluster seperately. According to my intuitive assumption about having 3 income groups, the groups are: “rich”, “middle class” and “poor”. When the group 1 is inspected they have higher credit limit, balance and have higher purchase frequency. This and other features clearly reveals that this group is the rich one. Accordingly group 2 is the “middle class” and group 3 is the “poor” group. # Validation Dunn Index(DI), Davies-Bouldin(DBI) Index and Silhouette Coefficient are inspected.

Clustering gets better as Dunn Index increases. DI evaluates the clusters using the farthest points and in this dataset there are very far points which I think outliers. DI could be more meaningful if the dataset would not contain outliers. DI decreases as the number of clusters increase thus it indicates number of clusters being lower is better.

Clustering gets better as Davies-Bouldin Index decreases. DBI is a metric of seperation of clusters. When scores are inspected, hierarchical clustering gives much better DBI(Connectivity) scores than k-means clustering. So, hierachical clustering is much better at seperating clusters. This can also be visually seen when the colored clusters are inspected.

Clustering gets better as Silhouette Coefficient increases. Since its range is between 0 and 1 the optimal value is 1. SC is expected to be higher than 0.5 and again hierarchical clustering is much better.

Even though we do these validation measures after clustering, in practice they can be applied before clustering in order to understand which configuration should be used.

Hide

```
# https://rdrr.io/cran/clusterCrit/man/intCriteria.html
intCriteria(dat_reorg_scaled, clustKm$cluster, c("Dunn", "dav", "silhouette") )
```

```
$dunn
[1] 0.002012817

$davies_bouldin
[1] 1.663439

$silhouette
[1] 0.1524162
```

Internal Validation

Hide

```
# https://www.rdocumentation.org/packages/clValid/versions/0.7/topics/clValid
# https://cran.r-project.org/web/packages/clValid/vignettes/clValid.pdf
# https://rdrr.io/cran/clValid/man/clValid-class.html
valid.intern <- clValid(dat_reorg , 2:6, clMethods=c("hierarchical","kmeans"), validation="internal")
```

```
The number of items to be clustered is larger than 'maxitems'
The memory and time required may be excessive, do you wish to continue?
(y to continue, any other character to exit)
```

Hide

```
y
summary(valid.intern)
```

```
Clustering Methods:
 hierarchical kmeans

Cluster sizes:
 2 3 4 5 6

Validation Measures:
                                  2        3        4        5        6
                                                                       
hierarchical Connectivity    3.2956   6.2246  14.4980  19.7726  31.1563
             Dunn            0.1943   0.1943   0.1020   0.1020   0.1020
             Silhouette      0.9081   0.8842   0.8526   0.8433   0.7935
kmeans       Connectivity  555.7802 546.5504 650.4917 855.3817 965.4060
             Dunn            0.0019   0.0025   0.0021   0.0048   0.0023
             Silhouette      0.5113   0.5095   0.4628   0.3930   0.3743

Optimal Scores:
```

	Score <dbl>	Method <chr>	Clusters <chr>
Connectivity	3.2956	hierarchical	2
Dunn	0.1943	hierarchical	2
Silhouette	0.9081	hierarchical	2

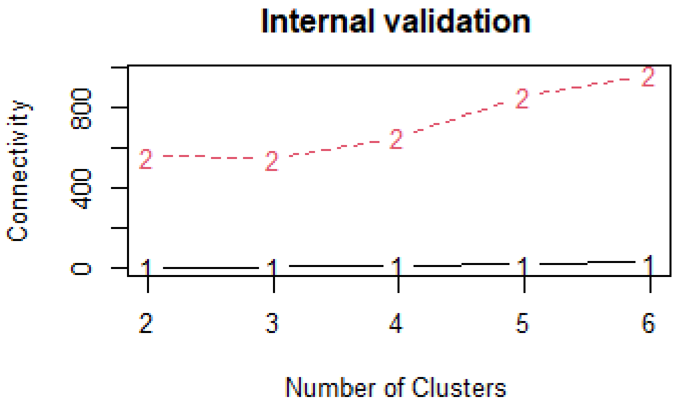
3 rows

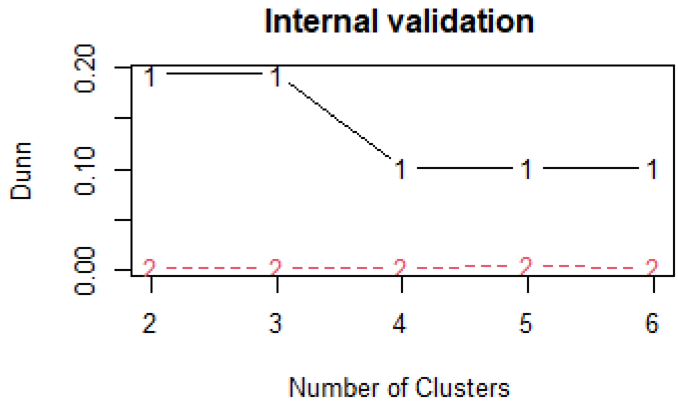
Hide

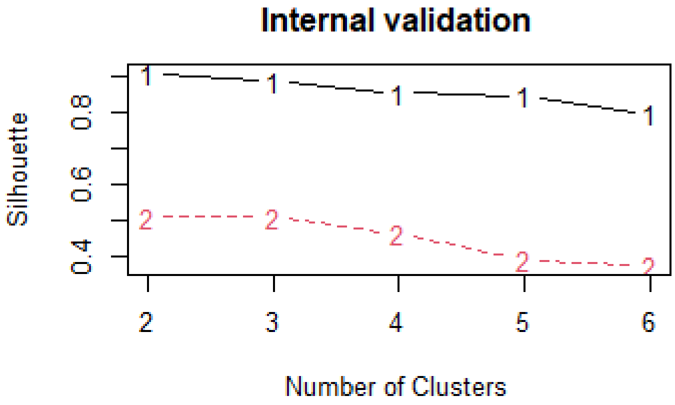
```
op <- par(no.readonly=TRUE)
par(mfrow=c(2,2),mar=c(4,4,3,1))
plot(valid.intern, legend=FALSE)
plot(nClusters(valid.intern), measures(valid.intern,"Dunn")[,,1],type="n",axes=F,xlab="",ylab="")
```

Hide

```
legend("center", clusterMethods(valid.intern), col=1:9, lty=1:9, pch=paste(1:9))
par(op)
```







Above results show that optimal scores are achieved by hierarchical clustering with k=2.

Stability Validation

Stabilty validation validates reproducibility of clustering solution another sample.The included measures are the average proportion of non-overlap (APN), the average distance (AD), the average distance between means (ADM), and the figure of merit (FOM) (Datta and Datta, 2003; Yeung et al., 2001). . In all cases the average is taken over all the deleted columns, and all measures should be minimized.

Hide

```
valid.stab <- clValid(dat_reorg , 2:6, clMethods=c("hierarchical","kmeans"), validation="stability")
```

```
The number of items to be clustered is larger than 'maxitems'
The memory and time required may be excessive, do you wish to continue?
(y to continue, any other character to exit)
```

Hide

```
y
summary(valid.stab)
```

```
Clustering Methods:
 hierarchical kmeans

Cluster sizes:
 2 3 4 5 6

Validation Measures:
                          2         3         4         5         6
                                                                   
hierarchical APN     0.0001    0.0009    0.0005    0.0013    0.0020
             AD   7005.3751 6992.6928 6832.9555 6825.3289 6801.3493
             ADM     7.4382   41.5973   31.1390   55.5458   77.1517
             FOM  1056.3065 1043.0755  990.3924  990.0132  988.3864
kmeans       APN     0.0221    0.0298    0.0344    0.0538    0.0670
             AD   5738.2163 5618.7797 5282.4759 4990.7224 4839.7741
             ADM   257.3348  362.1208  361.9208  561.9007  663.1631
             FOM   985.5752  991.3582  941.7552  936.3058  912.5554

Optimal Scores:
```

	Score <dbl>	Method <chr>	Clusters <chr>
APN	0.0001	hierarchical	2
AD	4839.7741	kmeans	6
ADM	7.4382	hierarchical	2
FOM	912.5554	kmeans	6

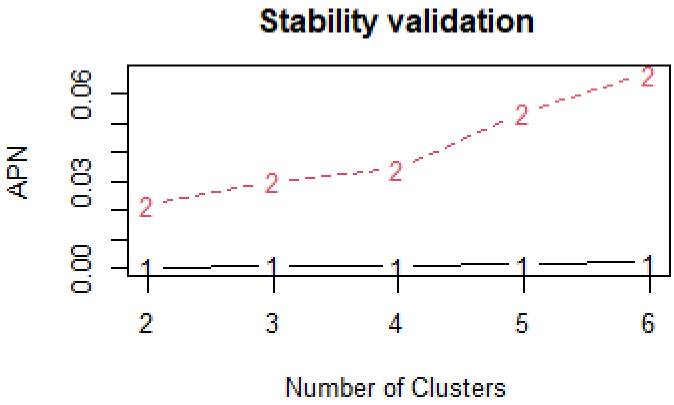
4 rows

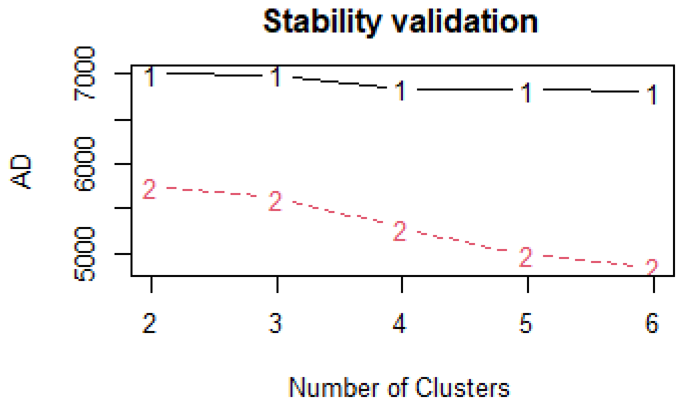
Hide

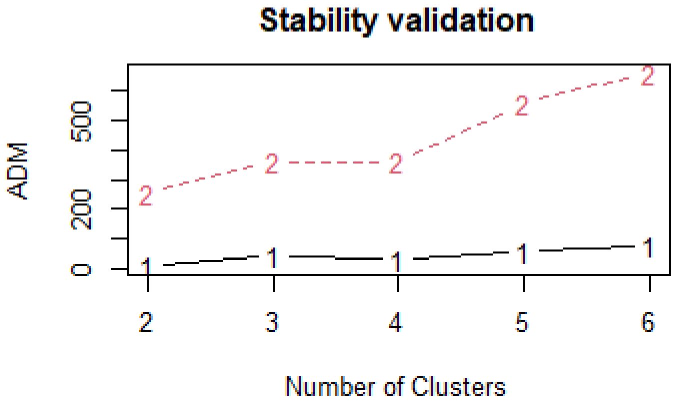
```
par(mfrow=c(2,2),mar=c(4,4,3,1))
plot(valid.stab, measure=c("APN","AD","ADM"),legend=FALSE)
plot(nClusters(valid.stab),measures(valid.stab,"APN")[,,1],type="n",axes=F,xlab="",ylab="")
```

Hide

```
legend("center", clusterMethods(valid.stab), col=1:9, lty=1:9, pch=paste(1:9))
par(op)
```







Here we see that hierarchical-2 and kmeans-6 performs the best in terms of stability. But inb internal validation hierarchical 2 was superior in all 3 metrics. Thus we can conclude that hierarchical-2 is the best.

Hide

```
valid.stabRankWeighs <-getRanksWeights(valid.stab)
print(valid.stabRankWeighs$ranks[,1:3], quote=FALSE)
```

```
    1              2              3
APN hierarchical-2 hierarchical-4 hierarchical-3
AD  kmeans-6       kmeans-5       kmeans-4
ADM hierarchical-2 hierarchical-4 hierarchical-3
FOM kmeans-6       kmeans-5       kmeans-4
```

Conclusion

In each section comments that are related to that section are made. So in the conclusion general comments are presented. The dataset is hard to work with since the data is not best suitable for clustering. Clusters almost overlap and first two components of PCA only cover the %47(27+20) of the data. For example, if first two components would add up to %80 percent of the data we could see much distinct and non overlapping clusters. Since my dataset also does not have labels, there is no way to verify if the clustering is correct. Thus, this dataset is rather open to comment.