Clustering of real estate ads for sale in Poland

Sugarbayar Enkhbayar

1/13/2023

Contents

Introduction

The clustering model is one of the important models in machine learning. This method is widely used in all fields such as medicine, economics, biology, and chemistry. By clustering users and making the most suitable incentives and activations for each cluster, it is possible to increase profit and revenue. This project will cluster real estate ads for sale in Poland on realting.com. This website is one of the biggest international affiliate sales systems with 20 years of experience. Data is scraped from realting.com with Rvest package. I also created an interactive dashboard with the Kmeans model using the results of the last model. You can see it at (https://sugarbayar.shinyapps.io/Cluster_KMeans/). Firstly, you have to choose a number of cluster and distance methods. After it, you can see the result of this model as a graphic and table.

Datasets

Data collection - web scrape

You need 30 minutes to run the below web scraping code. After doing the web scraping, I saved the results as a df_all.RDS file. We have a total of 137 pages of web data. So I wrote a loop up to 137. Other useful information is written as a comment.

```
library(rvest)
library(tidyverse)
library(stringr)
library(xml2)
```

```r
library(knitr)
# copy urls
df_urls=data.frame(urls=character()) # create emply df
for (i in 4:7) { # 137 pages. But in order to show you example, I write 4:7 instead of 1:137
  url=paste0('https://realting.com/property-for-sale/poland?page=',i,'&movemap-input=1&slug=property-fo
  url=read_html(url) # read i_th page link
  url_all=html_nodes(url,'a') %>% html_attr('href') # all links on i_th page
  url_tail=data.frame(x=url_all[seq(71,129,2)]) # create necessary tail of all links on i_th pag
  url_head=data.frame(y=rep('https://realting.com',nrow(url_tail))) # create same head of all links on
  url_full=cbind(url_head,url_tail) # column bind two dfs
  url_full$url_long=paste0(url_full$y,url_full$x) # combine head of link and tail of link
  add_url=data.frame(urls=url_full$url_long)
  df_urls=rbind(df_urls,add_url) # row bind each i_th df
  df_urls=unique(df_urls) # unique and check
}
head(df_urls,3)
```

```
##                                                                        urls
## 1 https://realting.com/property-for-sale/poland/etalon-estate-group/1547783
## 2 https://realting.com/property-for-sale/poland/etalon-estate-group/1547782
## 3 https://realting.com/property-for-sale/poland/etalon-estate-group/1547781
```

I have about 4000 adv links. Each link contains price data and text data. Text data includes other useful information such as count of bedrooms, count of bathrooms, and count of floors...

```r
# copy price and other data
df_pre=data.frame(y=character(),x=character()) # create empty df. x is text data. y is price data
for (i in 1:5) { # We have about 4000 urls. But in order to show you example, I write 5 instead of nrow
  url_read=read_html(df_urls[i,])
  url_price=html_nodes(url_read,'.price') %>% html_text() # price data of i_th adv
  #url_price=data.frame(y=url_price[1])
  url_text=html_nodes(url_read,'.newb-params') %>% html_text2() # text data of i_th adv. use class name
  df_add=data.frame(x=url_text,y=url_price[1]) # combine price and text data
  df_pre=rbind(df_pre,df_add) # row bind
  df_pre<-unique(df_pre) # unique
}
head(df_pre,1)
```

```
##
## 1 Posted at: 15.01.2023\nUpdated at: 16.01.2023\nLocation\nStrana: Poland\nOblast shtat: Masovian Vo
##            y
## 1 \210 134,728
```

There are some important variables of property data for sale in Poland.

- Address - address of property
- Bathrooms - count of bathroom
- Bedrooms - count of bedroom
- Floor - floor number of building
- Gorod -
- Number - floor number of apartment
- Oblast

- Posted - posted data of adv
- price - price dollar $
- Rooms - count of rooms
- Updated - updated date of adv
- Strana -
- Total - total square meter

As you see, all the information we need is in very long text. So we need to write a function to distinguish between them. I used gsub function. After it, I wrote a text mining code suitable for each case.

```
# create df from web scrape
df_all<-data.frame(c1=c("Address:","Bathrooms","Bedrooms","Floor:","Gorod:","Number","Oblast","Posted",
                   check=c('','','','','','','','','','','','','')) # create empty df for last datafram
for (i in 1:nrow(df_pre)) {
  try({ # loop will continue. don't care error
    Split <- function(string) {
      s1 <- gsub("\n", "\\1\n", string)
      #s2 <- gsub("(.{3})", "\\1\n", s1)
      spl <- strsplit(s1, "\n")
      lapply(spl, function(s) replace(s, s == "   ", ""))
    }
    df_adv=Split(df_pre[i,]$x)
    df_text=df_adv[[1]][1:16]
    df_text=data.frame(c=df_text)
    df_text$c1=word(df_text$c,1)
    df_text=df_text[complete.cases(df_text),]
    df_text$value[df_text$c1=='Posted']<-sub("Posted at:*", "", df_text$c[df_text$c1=='Posted'])
    df_text$value[df_text$c1=='Updated']<-sub("Updated at:*", "", df_text$c[df_text$c1=='Updated'])
    df_text$value[df_text$c1=='Strana:']<-sub("Strana:*", "", df_text$c[df_text$c1=='Strana:'])
    df_text$value[df_text$c1=='Oblast']<-sub("Oblast shtat:*", "", df_text$c[df_text$c1=='Oblast'])
    df_text$value[df_text$c1=='Gorod:']<-sub("Gorod:*", "", df_text$c[df_text$c1=='Gorod:'])
    df_text$value[df_text$c1=='Address:']<-sub("Address:*", "", df_text$c[df_text$c1=='Address:'])
    df_text$value[df_text$c1=='Number']<-sub("Number of floors:*", "", df_text$c[df_text$c1=='Number'])
    df_text$value[df_text$c1=='Floor:']<-sub("Floor:*", "", df_text$c[df_text$c1=='Floor:'])
    df_text$value[df_text$c1=='Rooms:']<-sub("Rooms:*", "", df_text$c[df_text$c1=='Rooms:'])
    df_text$value[df_text$c1=='Bedrooms']<-sub("Bedrooms*", "", df_text$c[df_text$c1=='Bedrooms'])
    df_text$value[df_text$c1=='Bathrooms']<-sub("Bathrooms*", "", df_text$c[df_text$c1=='Bathrooms'])
    df_text$value[df_text$c1=='Total']<-sub("Total area:*", "", df_text$c[df_text$c1=='Total'])
    df_text=df_text %>% select(c1,value)
    df_price=data.frame(c1='price',value=df_pre[i,]$y)
    df_adv_add=rbind(df_text,df_price)
    colnames(df_adv_add)[2]<-i
    df_adv_add=unique(df_adv_add)
    df_all<-merge(df_all,df_adv_add,by=c('c1'),all.x = T,all.y = F)
  })
}
df_all[,1:4]
```

```
##          c1 check                  1                 2
## 1   Address:                Chmielna      Pabla Nerudy
## 2 Bathrooms                     <NA>              <NA>
## 3  Bedrooms                        2                 3
## 4    Floor:                     <NA>              <NA>
```

```
## 5     Gorod:                         Warsaw                Warsaw
## 6     Number                           <NA>                  <NA>
## 7     Oblast        Masovian Voivodeship  Masovian Voivodeship
## 8     Posted                     15.01.2023            15.01.2023
## 9      price                   \210 134,728           \210 117,271
## 10    Rooms:                           <NA>                  <NA>
## 11   Strana:                         Poland                Poland
## 12     Total                    35 m<U+00B2>          44 m<U+00B2>
## 13   Updated                     16.01.2023            16.01.2023
```

Data cleaning

I have 16 variables and, 4014 observations. I don't use address, gorod, oblast, and strana variables. So I deleted them from df_all. After that, I changed the type of variables. And I created two variables. **Posted day** variable is the count of days since posted day. But **Updated day** variable is the count of days since updated day.

```
library(tidyverse)
df_all=readRDS("df_all.RDS")
df_all=df_all[-2]
df_all<-data.frame(t(df_all))
colnames(df_all)=df_all[1,]
df_all<-df_all[-1,]
df_all <- df_all[, !duplicated(colnames(df_all))]
df_all<-unique(df_all)
df_all<-df_all %>% select(-`Address:`,-`Gorod:`,-Oblast,-`Strana:`)
for (i in 1:length(df_all)) {
  df_all[,i]=gsub("[^[:digit:]., ]", "", df_all[,i])
  df_all[,i]=gsub("[][!#$%()*,:;<=>@^_`|~{}]", "", df_all[,i])
  df_all[,i]=gsub(" ", "", df_all[,i])
}
df_all$Bathrooms<-as.numeric(df_all$Bathrooms)
df_all$Bedrooms<-as.numeric(df_all$Bedrooms)
df_all$`Floor:`<-as.numeric(df_all$`Floor:`)
df_all$Number<-as.numeric(df_all$Number)
df_all$`Rooms:`<-as.numeric(df_all$`Rooms:`)
df_all$Total<-as.numeric(df_all$Total)
df_all$price<-as.numeric(df_all$price)
df_all$Posted<-as.Date(df_all$Posted,'%d.%m.%Y')
df_all$Updated<-as.Date(df_all$Updated,'%d.%m.%Y')
df_all$Posted_day=Sys.Date()-df_all$Posted
df_all$Updated_day=Sys.Date()-df_all$Updated
head(df_all)
```

```
##   Bathrooms Bedrooms Floor: Number     Posted   price Rooms: Total    Updated
## 1         1        1     NA      1 2023-01-13   88893      2    50 2023-01-05
## 2         2        5     NA      1 2023-01-12  848951      6   324 2023-01-13
## 3         2        5     NA      1 2023-01-12  765762      6   480 2023-01-13
## 4         5        5     NA      3 2023-01-12 4500000      8   758 2023-01-13
## 5         5        4     NA      2 2023-01-12 1162508     10  1076 2023-01-13
## 6         3        3     NA      2 2023-01-12 2879607      6  1077 2023-01-13
##   Posted_day Updated_day
## 1     5 days     13 days
```

```
## 2     6 days      5 days
## 3     6 days      5 days
## 4     6 days      5 days
## 5     6 days      5 days
## 6     6 days      5 days
```

```
count(df_all)
```

```
##      n
## 1 3985
```

Exploratory Data Analysis

Before using machine learning model, we need to exploratory data analysis. ### Variable identification We will use these variables for cluster analysis.

```
colnames(df_all)
```

```
##  [1] "Bathrooms"   "Bedrooms"    "Floor:"      "Number"      "Posted"
##  [6] "price"       "Rooms:"      "Total"       "Updated"     "Posted_day"
## [11] "Updated_day"
```

There are two *date variables* and two *difftime variables*. Also, there are seven *numeric variables*.

```
sapply(df_all,class)
```

```
##   Bathrooms    Bedrooms      Floor:      Number      Posted       price
##   "numeric"   "numeric"   "numeric"   "numeric"      "Date"   "numeric"
##      Rooms:       Total     Updated  Posted_day Updated_day
##   "numeric"   "numeric"      "Date"  "difftime"  "difftime"
```

Missing values treatment

In our dataset, there are many NA rows. We can generate some NA values using price data. But this is not an optimal way. Bathrooms, floor, bedrooms, and room column has the most NA values. So I decided to delete rows with NA value from the dataset.

```
sapply(df_all, function(x) sum(is.na(x)))
```

```
##   Bathrooms    Bedrooms      Floor:      Number      Posted       price
##        3387        1775        2594        3352        1193           0
##      Rooms:       Total     Updated  Posted_day Updated_day
##        1747          27           0        1193           0
```

```
df_all=df_all[complete.cases(df_all),]
```

After removing NA rows, there are 297 observations. We will use it from now on. It is our final dataset.

```r
sapply(df_all, function(x) sum(is.na(x)))
```

```
##   Bathrooms    Bedrooms      Floor:      Number      Posted       price 
##           0           0           0           0           0           0 
##      Rooms:       Total     Updated  Posted_day Updated_day 
##           0           0           0           0           0
```

```r
count(df_all)
```

```
##     n
## 1 297
```

```r
df_all=df_all %>% select(-Posted,-Updated) # we don't need posted, updated variables anymore. Because w
df_all$Posted_day<-as.numeric(df_all$Posted_day)
df_all$Updated_day<-as.numeric(df_all$Updated_day)
```

Univariate analysis

Central Tendency The maximum number of bathrooms and bedrooms is 8. The highest floor is 27. The maximum number of rooms is 8. The biggest apartment is 554-meter square. The maximum price of the apartment is 1628k, the minimum price of the apartment is 66k, and the average price of the apartment is 200k.

```r
summary(df_all)
```

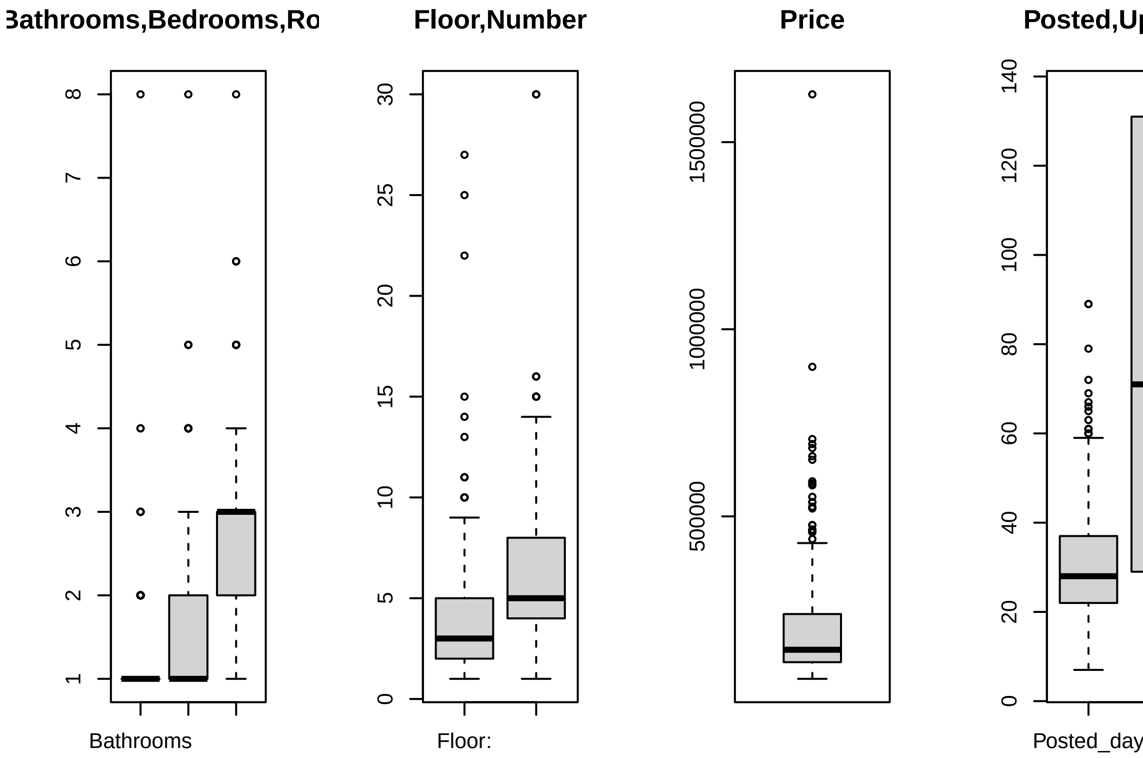
```
##    Bathrooms        Bedrooms         Floor:           Number      
##  Min.   :1.000   Min.   :1.000   Min.   : 1.000   Min.   : 1.000  
##  1st Qu.:1.000   1st Qu.:1.000   1st Qu.: 2.000   1st Qu.: 4.000  
##  Median :1.000   Median :1.000   Median : 3.000   Median : 5.000  
##  Mean   :1.249   Mean   :1.744   Mean   : 3.667   Mean   : 6.051  
##  3rd Qu.:1.000   3rd Qu.:2.000   3rd Qu.: 5.000   3rd Qu.: 8.000  
##  Max.   :8.000   Max.   :8.000   Max.   :27.000   Max.   :30.000  
##      price             Rooms:          Total          Posted_day   
##  Min.   :  65954   Min.   :1.000   Min.   : 25.00   Min.   : 7.00  
##  1st Qu.: 110465   1st Qu.:2.000   1st Qu.: 45.00   1st Qu.:22.00  
##  Median : 143649   Median :3.000   Median : 56.00   Median :28.00  
##  Mean   : 201213   Mean   :2.768   Mean   : 69.82   Mean   :30.77  
##  3rd Qu.: 239078   3rd Qu.:3.000   3rd Qu.: 74.00   3rd Qu.:37.00  
##  Max.   :1627761   Max.   :8.000   Max.   :554.00   Max.   :89.00  
##   Updated_day    
##  Min.   :  5.00  
##  1st Qu.: 29.00  
##  Median : 71.00  
##  Mean   : 73.04  
##  3rd Qu.:131.00  
##  Max.   :136.00
```

```r
library(e1071)
library(moments)
tibble(
    Column   = names(df_all),
    Variance = purrr::map_dbl(df_all, var),
    SD       = purrr::map_dbl(df_all, sd),
    IQR      = purrr::map_dbl(df_all,IQR),
    SKW      = purrr::map_dbl(df_all,skewness),
    KRT      = purrr::map_dbl(df_all,kurtosis))
```

Measure of dispersion

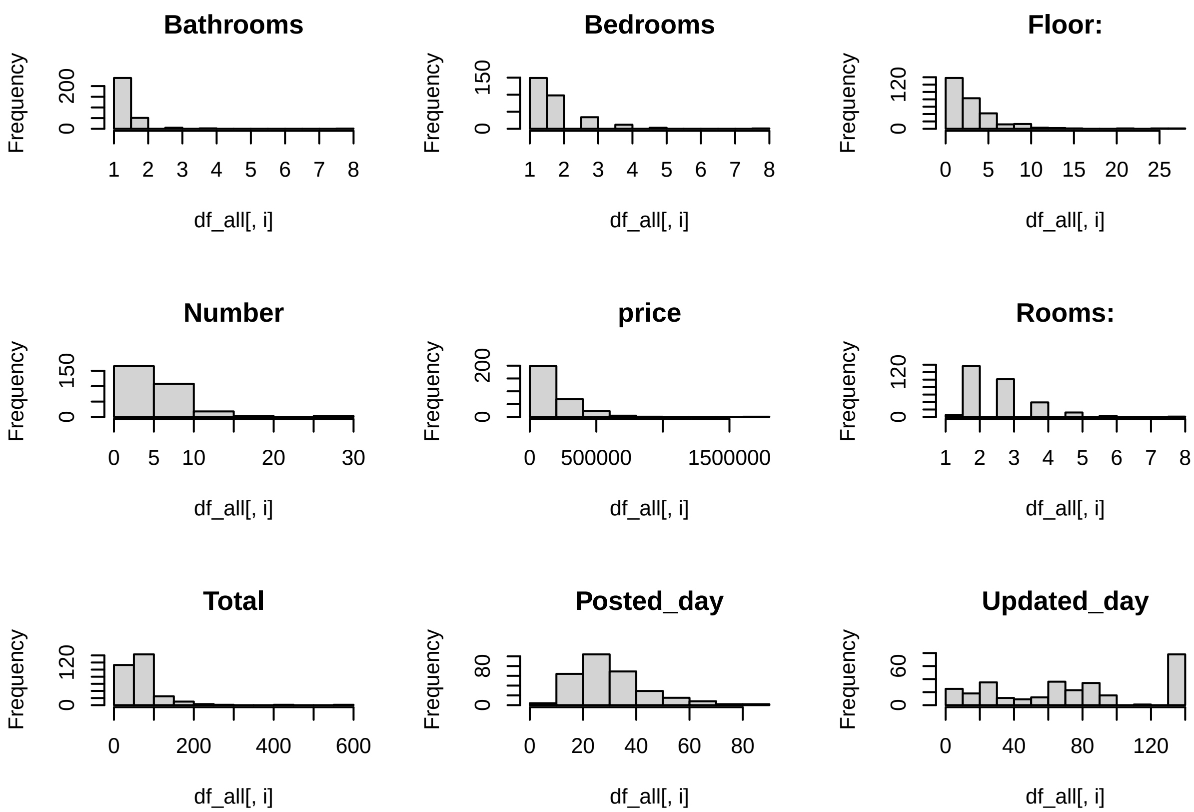
```
## # A tibble: 9 x 6
##   Column      Variance         SD    IQR    SKW   KRT
##   <chr>          <dbl>      <dbl>  <dbl>  <dbl> <dbl>
## 1 Bathrooms   4.04e- 1      0.636      0 5.19   46.9
## 2 Bedrooms    9.28e- 1      0.963      1 1.87    9.10
## 3 Floor:      1.11e+ 1      3.34       3 3.20   18.7
## 4 Number      1.60e+ 1      4.00       4 2.62   14.5
## 5 price       2.50e+10 158244.    128613 3.65   26.2
## 6 Rooms:      9.42e- 1      0.971      1 1.34    5.96
## 7 Total       2.52e+ 3     50.2       29 4.87   38.8
## 8 Posted_day  1.87e+ 2     13.7       15 1.28    5.24
## 9 Updated_day 1.91e+ 3     43.7      102 0.0739  1.74
```

```r
par(mfrow = c(1,4)) # two rows, one column
boxplot(df_all[c(1,2,6)],main='Bathrooms,Bedrooms,Rooms')
boxplot(df_all[c(3,4)],main='Floor,Number')
boxplot(df_all[c(5)],main="Price")
boxplot(df_all[c(8,9)],main="Posted,Updated")
```



Visualization of EDA

```
par(mfrow = c(3,3)) # two rows, one column
for(i in 1:9){
  hist(df_all[,i],main = colnames(df_all)[i])
}
```

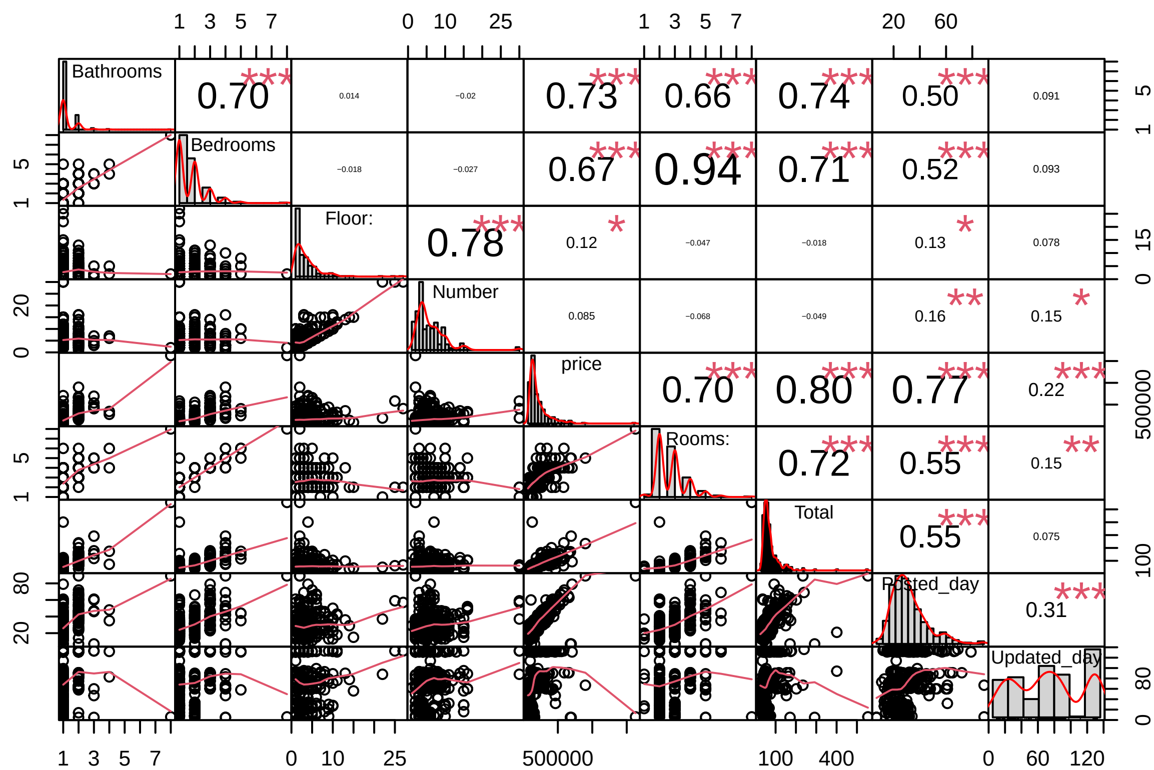



```r
par(mfrow = c(1,1))
```

Bi-variate analysis

As you can see in the below graph, there are many variables that are strongly positively correlated. For example, as the number of bathrooms and bedrooms increases, the price increases.

```r
library(PerformanceAnalytics)
chart.Correlation(df_all)
```



Outlier treatment

Outlier data were identified using IQR values. There were a total of 62 outlier data, which were removed from the original data. Also, the Bathroom variable is right-skewed, so it should be removed.

```
for (i in 1:9) {
  q1=quantile(df_all[,i], .25)
  q3=quantile(df_all[,i], .75)
  IQR=IQR(df_all[,i])
  count_out<-subset(df_all, df_all[,i] > (q1 - 1.5*IQR) & df_all[,i] < (q3 + 1.5*IQR))
  print(paste0(colnames(df_all)[i]," variable count of outlier - ",count(df_all)-count(count_out)))
}
```

```
## [1] "Bathrooms variable count of outlier - 297"
## [1] "Bedrooms variable count of outlier - 16"
## [1] "Floor: variable count of outlier - 16"
## [1] "Number variable count of outlier - 15"
## [1] "price variable count of outlier - 22"
## [1] "Rooms: variable count of outlier - 16"
## [1] "Total variable count of outlier - 28"
## [1] "Posted_day variable count of outlier - 16"
## [1] "Updated_day variable count of outlier - 0"
```

```
outliers=df_all[0,]
for (i in 2:9) {
  q1=quantile(df_all[,i], .25)
  q3=quantile(df_all[,i], .75)
  IQR=IQR(df_all[,i])
  count_out<-subset(df_all, df_all[,i] > (q1 - 1.5*IQR) & df_all[,i] < (q3 + 1.5*IQR))
  add_out<-setdiff(df_all,count_out)
  outliers<-rbind(outliers,add_out)
  outliers<-unique(outliers)
}
df_all<-setdiff(df_all,outliers)
count(df_all)
```

```
##     n
## 1 235
```

Non-hierarchical method

There are many non-hierarchical methods such as kmeans, pam, calra, and fanny. Also, we tried euclidean, manhattan, minkowski, and canberra as hc_metrics. But results of these parameters are the same. So I decided to use only euclidean. Before any clustering model, we need to normalize our dataset. Because it is more efficient to divide normalized data into clusters.

```
df=df_all[c('Bathrooms','Bedrooms','Floor:','Number','price','Rooms:','Total','Posted_day','Updated_day
### Data normalized
df$Bathrooms<-scale(df$Bathrooms)
df$Bedrooms<-scale(df$Bedrooms)
df$`Floor:`<-scale(df$`Floor:`)
df$Number<-scale(df$Number)
df$price<-scale(df$price)
df$`Rooms:`<-scale(df$`Rooms:`)
df$Total<-scale(df$Total)
df$Posted_day<-scale(df$Posted_day)
df$Updated_day<-scale(df$Updated_day)
```

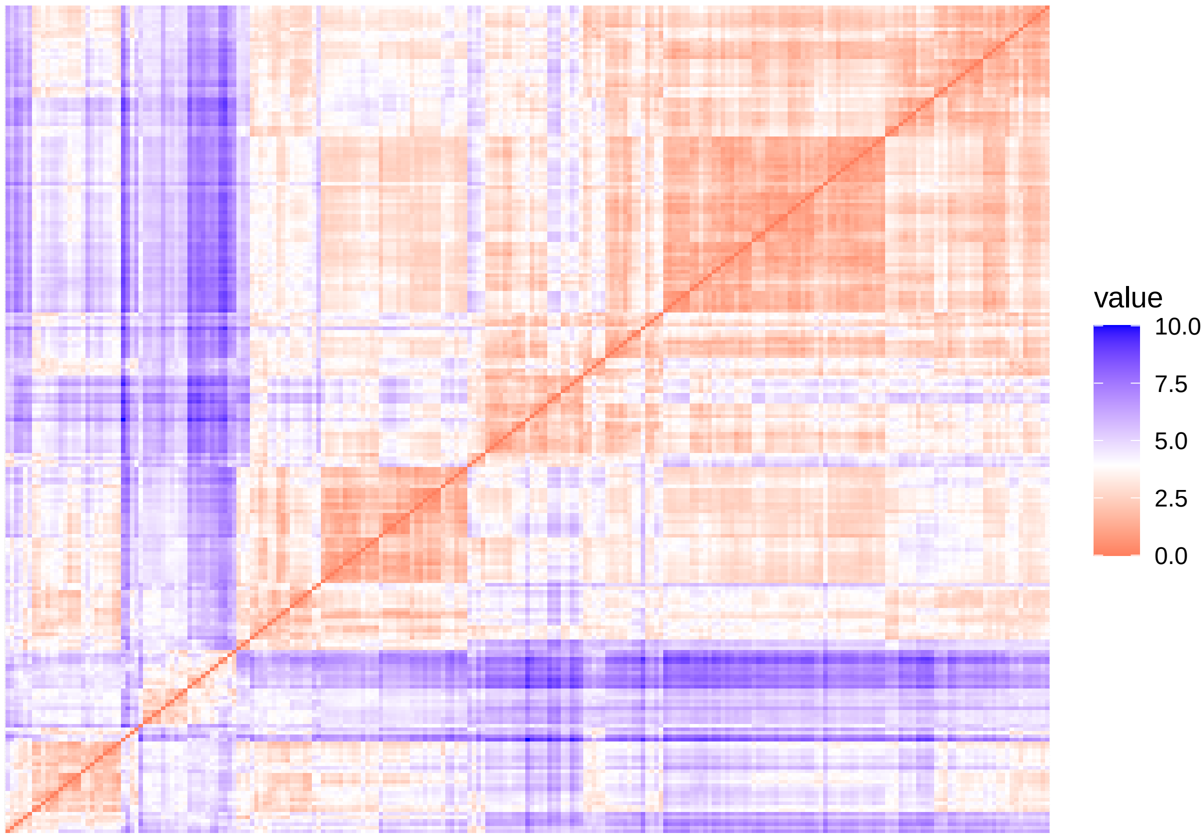
I used Hopkins statistics to measure of cluster tendency of the data set. Hopkin=1 - data is highly clustered Hopkin=0.5 - random data Hopkin=0 - uniformly distributed data. Hopkins statistic of our data set is 0.987, which is almost 1. So it means our data set is highly clustered good data set.

```
library(factoextra)
hopkins::hopkins(df)
```

```
## [1] 0.9981807
```

```
get_clust_tendency(df, 2, graph=TRUE, gradient=list(low="red", mid="white", high="blue"))
```

```
## $hopkins_stat
## [1] 0.6980075
##
## $plot
```



silhouette

I used all possible non-hierarchical models and a number of clusters. We can see silhouette values for each case. The higher silhouette value is better. As you see, the 2 cluster kmeans model's silhouette value is the highest one. It is 0,3446. So it is the best model.

```
library(factoextra)
library(cluster)
library(formattable)
res_k=data.frame(Kcount=as.numeric(),sil=as.numeric())
cluster_model <- function(model_in,model_out) {
  res=data.frame(Kcount=as.numeric(),sil=as.numeric())
  for(i in 2:10){
    km1=eclust(df,FUNcluster = c(model_in),k=i,graph = F,hc_metric = 'euclidean')
    add_res<-data.frame(kcount=i,sil=km1$silinfo$avg.width)
    res<-rbind(res,add_res)
    res<-unique(res)
  }
  return(res)
}
for (z in c("kmeans", "pam", "clara", "fanny", "hclust", "agnes", "diana")) {
  assign(paste0(z,'_out'),cluster_model(z,result_kmeans))
  #kmeans_out=cluster_model(z,result_kmeans)
}
colnames(kmeans_out)[2]<-'kmeans.sil'
```

```
colnames(pam_out)[2]<-'pam.sil'
colnames(clara_out)[2]<-'clara.sil'
colnames(fanny_out)[2]<-'fanny.sil'
colnames(hclust_out)[2]<-'hclust.sil'
colnames(agnes_out)[2]<-'agnes.sil'
colnames(diana_out)[2]<-'diana.sil'
df_list <- list(kmeans_out,pam_out,clara_out,fanny_out,hclust_out,agnes_out,diana_out)
Result_Models=Reduce(function(x, y) merge(x, y, all=TRUE), df_list)
formattable(Result_Models,list(kmeans.sil=color_tile('transparent','lightgreen'),
                               pam.sil=color_tile('transparent','lightgreen'),
                               clara.sil=color_tile('transparent','lightgreen'),
                               fanny.sil=color_tile('transparent','lightgreen'),
                               hclus.silt=color_tile('transparent','lightgreen'),
                               agnes.sil=color_tile('transparent','lightgreen'),
                               diana.sil=color_tile('transparent','lightgreen')))
```

kcount

kmeans.sil

pam.sil

clara.sil

fanny.sil

hclust.sil

agnes.sil

diana.sil

2

0.3446526

0.2919032

0.2872919

0.26597518

0.3492066

0.3492066

0.3579309

3

0.2363866

0.2546550

0.2104728

0.26597518

0.2147055

0.2147055

0.3231118

4

0.2447696

0.1755392

0.1652333

0.26597518

0.2307936

0.2307936

0.2953633

5

0.2686015

0.2153519

0.1745290

0.26597518

0.2569593

0.2569593

0.2268705

6

0.2597109

0.2575768

0.2485170

0.01761893

0.2350901

0.2350901

0.1727056

7

0.2561664

0.2519145

0.2398448

0.26597518

0.2469550

0.2469550

0.1708146

8

0.2618610

0.1921360

0.2405070

0.26597518

0.2472768

0.2472768

0.1695754

9

0.2516598

0.2013606

0.2397367

0.26597518

0.2558529

0.2558529

0.1617934

10

0.2432278

0.2090590

0.2055449

0.26597518

0.2557379

0.2557379

0.2018517

Also, we can use gap statistic. Lower gap value is better. As you see, 2 cluster kmeans model's gap value is the lowest one. It is 0,4776. ### gap

```
res_kmean_gap=clusGap(df,FUN=kmeans,K.max = 10,B=2)
res_kmean_gap=data.frame(res_kmean_gap$Tab)
res_kmean_gap=data.frame(kcount=1:10,res_kmean_gap$gap)
res_pam_gap=clusGap(df,FUN=pam,K.max = 10,B=2)
res_pam_gap=data.frame(res_pam_gap$Tab)
res_pam_gap=data.frame(kcount=1:10,res_pam_gap$gap)
res_clara_gap=clusGap(df,FUN=clara,K.max = 10,B=2)
res_clara_gap=data.frame(res_clara_gap$Tab)
res_clara_gap=data.frame(kcount=1:10,res_clara_gap$gap)
res_fanny_gap=clusGap(df,FUN=fanny,K.max = 10,B=2)
res_fanny_gap=data.frame(res_fanny_gap$Tab)
res_fanny_gap=data.frame(kcount=1:10,res_fanny_gap$gap)
df_list <- list(res_kmean_gap,res_pam_gap,res_clara_gap,res_fanny_gap)
Result_gap=Reduce(function(x, y) merge(x, y, all=TRUE), df_list)
formattable(Result_gap[-1,],list(res_kmean_gap.gap=color_tile('transparent','lightgreen'),
                                 res_pam_gap.gap    =color_tile('transparent','lightgreen'),
                                 res_clara_gap.gap  =color_tile('transparent','lightgreen'),
                                 res_fanny_gap.gap=color_tile('transparent','lightgreen')))
```

kcount

res_kmean_gap.gap

res__pam__gap.gap

res__clara__gap.gap

res__fanny__gap.gap

2

2

0.4776269

0.5102171

0.4925293

0.4673821

3

3

0.5177701

0.5202156

0.5263177

0.4673821

4

4

0.5444630

0.5281423

0.5468790

0.4673821

5

5

0.5814235

0.5935601

0.5837608

0.4780993

6

6

0.5665218

0.6548386

0.6705496

0.5166958

7

7

0.6307605

0.6552541

0.6620954

0.5367905

8

8

0.6295707

0.6416564

0.6714479

0.5262227

9

9

0.6343295

0.6462164

0.6682238

0.4673821

10

10

0.6489854

0.6525939

0.6639127

0.5330495

calinski

Also, you can use calinski value to choose the best model.

```
library(factoextra)
library(cluster)
library(formattable)
library(fpc)
res_k=data.frame(Kcount=as.numeric(),sil=as.numeric())
cluster_model <- function(model_in,model_out) {
  res=data.frame(Kcount=as.numeric(),sil=as.numeric())
  for(i in 2:10){
    km1=eclust(df,FUNcluster = c(model_in),k=i,graph = F,hc_metric = 'euclidean')
    add_res<-data.frame(kcount=i,calinski=round(calinhara(df,km1$cluster),digits=2))
    res<-rbind(res,add_res)
    res<-unique(res)
  }
  return(res)
}
for (z in c("kmeans", "pam", "clara", "fanny", "hclust", "agnes", "diana")) {
  assign(paste0(z,'_out'),cluster_model(z,result_kmeans))
```

```
}
colnames(kmeans_out)[2]<-'kmeans.calins'
colnames(pam_out)[2]<-'pam.calins'
colnames(clara_out)[2]<-'clara.calins'
colnames(fanny_out)[2]<-'fanny.calins'
colnames(hclust_out)[2]<-'hclust.calins'
colnames(agnes_out)[2]<-'agnes.calins'
colnames(diana_out)[2]<-'diana.calins'
df_list <- list(kmeans_out,pam_out,clara_out,fanny_out,hclust_out,agnes_out,diana_out)
Result_Models=Reduce(function(x, y) merge(x, y, all=TRUE), df_list)
formattable(Result_Models,list(kmeans.calins=color_tile('transparent','lightgreen'),
                               pam.calins=color_tile('transparent','lightgreen'),
                               clara.calins=color_tile('transparent','lightgreen'),
                               fanny.calins=color_tile('transparent','lightgreen'),
                               hclust.calins=color_tile('transparent','lightgreen'),
                               agnes.calins=color_tile('transparent','lightgreen'),
                               diana.calins=color_tile('transparent','lightgreen')))
```

kcount

kmeans.calins

pam.calins

clara.calins

fanny.calins

hclust.calins

agnes.calins

diana.calins

2

108.62

101.65

99.92

95.09

97.95

97.95

105.94

3

83.51

79.40

71.64

95.09

79.10

79.10

72.88

4

77.48

62.77

61.50

95.09

74.28

74.28

51.83

5

76.60

65.69

62.78

95.09

74.30

74.30

57.43

6

69.91

73.54

69.97

48.22

70.55

70.55

49.44

7

70.02

66.44

63.74

95.09

66.80

66.80

42.38

8

63.75

58.57

58.47

95.09

62.89

62.89

38.84

9

59.59

55.47

54.98

95.09

59.95

59.95

36.17

10

57.24

54.30

51.33

95.09

58.05

58.05

42.32

dudahard

```
####################################################
library(factoextra)
library(cluster)
library(formattable)
res_k=data.frame(Kcount=as.numeric(),sil=as.numeric())
cluster_model <- function(model_in,model_out) {
  res=data.frame(Kcount=as.numeric(),sil=as.numeric())
  for(i in 2:10){
    km1=eclust(df,FUNcluster = c(model_in),k=i,graph = F,hc_metric = 'euclidean')
    add_res<-data.frame(kcount=i,dudahart=dudahart2(df,km1$cluster)[1])
    res<-rbind(res,add_res)
    res<-unique(res)
  }
  return(res)
}
for (z in c("kmeans", "pam", "clara", "fanny", "hclust", "agnes", "diana")) {
  assign(paste0(z,'_out'),cluster_model(z,result_kmeans))
  #kmeans_out=cluster_model(z,result_kmeans)
}
colnames(kmeans_out)[2]<-'kmeans.duda'
```

```
colnames(pam_out)[2]<-'pam.duda'
colnames(clara_out)[2]<-'clara.duda'
colnames(fanny_out)[2]<-'fanny.duda'
colnames(hclust_out)[2]<-'hclust.duda'
colnames(agnes_out)[2]<-'agnes.duda'
colnames(diana_out)[2]<-'diana.duda'
df_list <- list(kmeans_out,pam_out,clara_out,fanny_out,hclust_out,agnes_out,diana_out)
Result_Models=Reduce(function(x, y) merge(x, y, all=TRUE), df_list)
formattable(Result_Models,list(kmeans.duda=color_tile('transparent','lightgreen'),
                               pam.duda=color_tile('transparent','lightgreen'),
                               clara.duda=color_tile('transparent','lightgreen'),
                               fanny.duda=color_tile('transparent','lightgreen'),
                               hclust.duda=color_tile('transparent','lightgreen'),
                               agnes.duda=color_tile('transparent','lightgreen'),
                               diana.duda=color_tile('transparent','lightgreen')))
```

kcount

kmeans.duda

pam.duda

clara.duda

fanny.duda

hclust.duda

agnes.duda

diana.duda

2

0

9.992007e-16

2.664535e-15

4.041212e-14

8.104628e-15

8.104628e-15

1.110223e-16

3

0

0.000000e+00

0.000000e+00

4.041212e-14

0.000000e+00

0.000000e+00

0.000000e+00

4

0

0.000000e+00

0.000000e+00

4.041212e-14

0.000000e+00

0.000000e+00

0.000000e+00

5

0

0.000000e+00

0.000000e+00

4.041212e-14

0.000000e+00

0.000000e+00

0.000000e+00

6

0

0.000000e+00

0.000000e+00

2.442491e-15

0.000000e+00

0.000000e+00

0.000000e+00

7

0

0.000000e+00

0.000000e+00

4.041212e-14

0.000000e+00

0.000000e+00

0.000000e+00

8

0

0.000000e+00

0.000000e+00

4.041212e-14

0.000000e+00

0.000000e+00

0.000000e+00

9

0

0.000000e+00

0.000000e+00

4.041212e-14

0.000000e+00

0.000000e+00

0.000000e+00

10

0

0.000000e+00

0.000000e+00

4.041212e-14

0.000000e+00

0.000000e+00

0.000000e+00

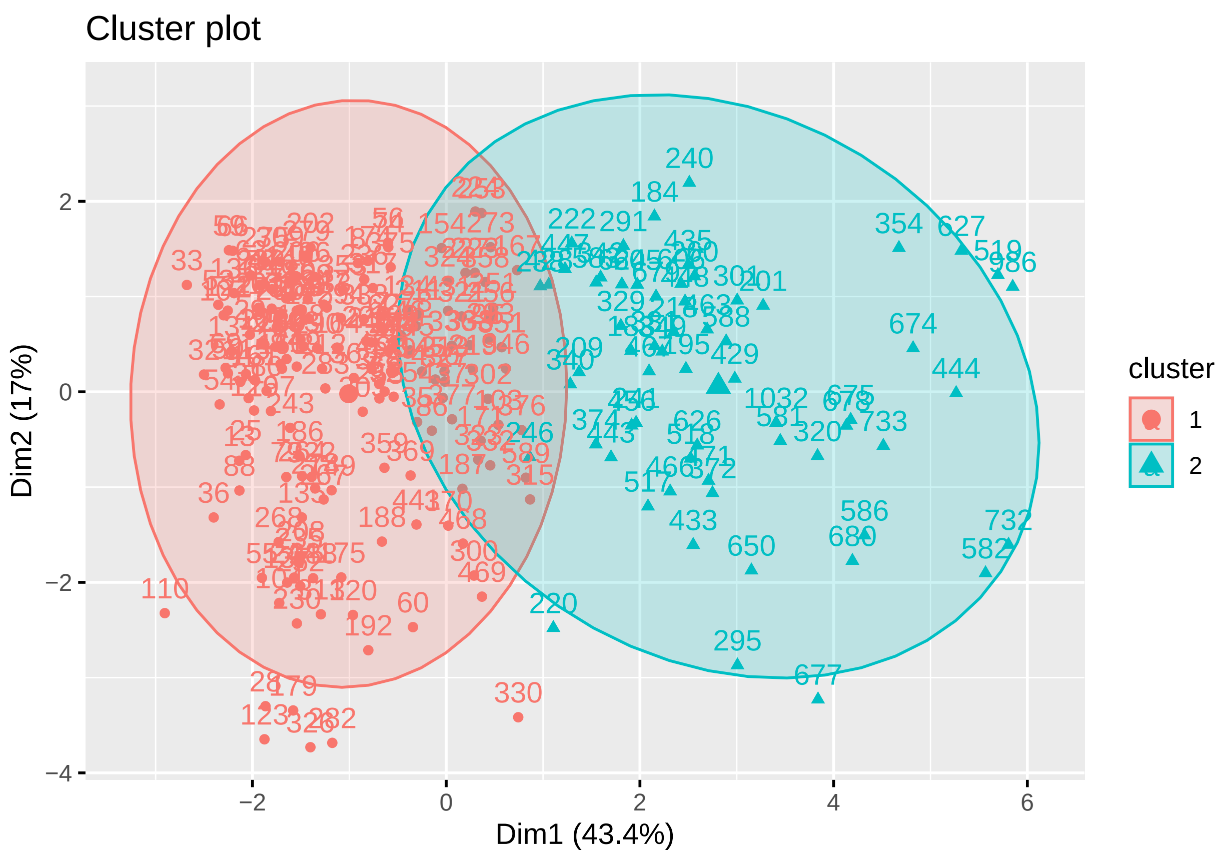
Result of non-hierarchical method

The best model is the 2 cluster kmeans model. Now we analyze the result of this model.

Clusters and silhouette

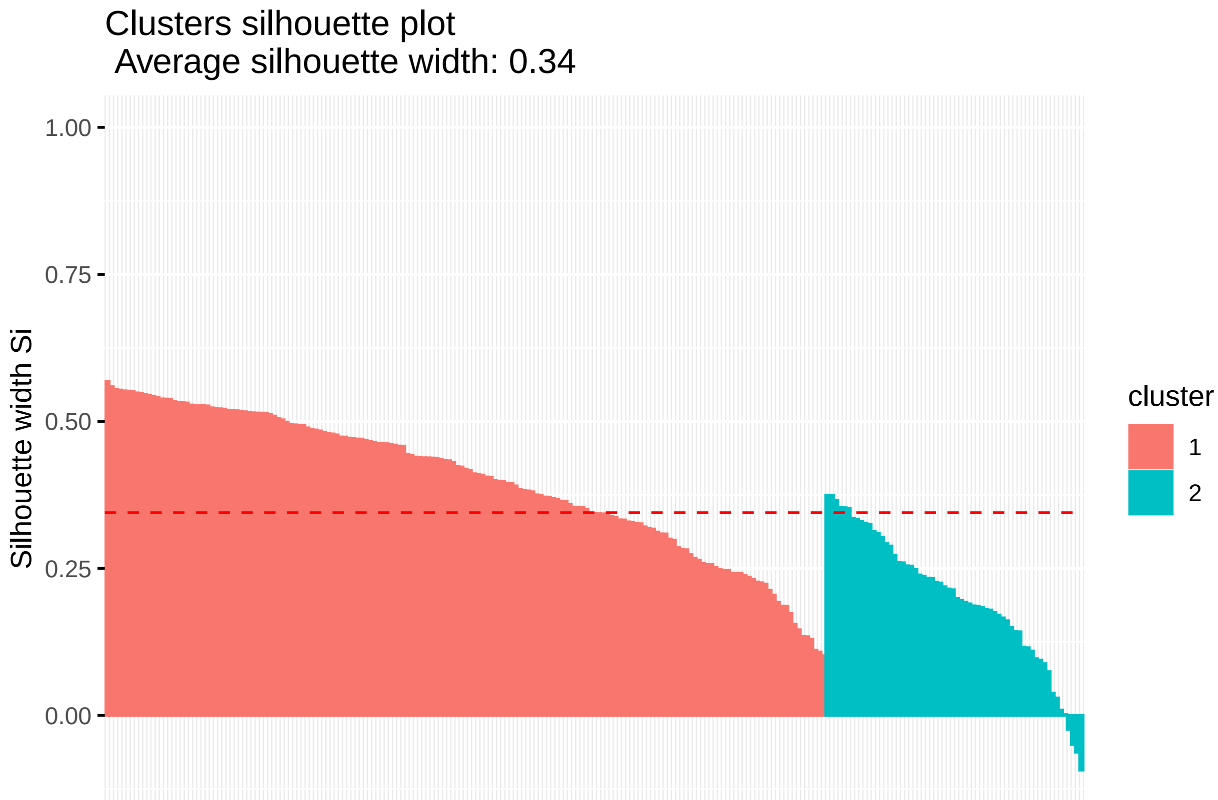
We have blue and red two clusters. And we have two dimensions. Variance of Dim1 is 43,4% of other variables. And variance of Dim2 is 17% of other variable. In other words, Dim1 can explain 43,4 percent of the other variables. Average silhouette value is 0,33.

```
km1=eclust(df,FUNcluster = c("kmeans"),k=2,graph = F,hc_metric = 'euclidean')
fviz_cluster(km1,ellipse.type = 'norm')
```




```
fviz_silhouette(km1)
```

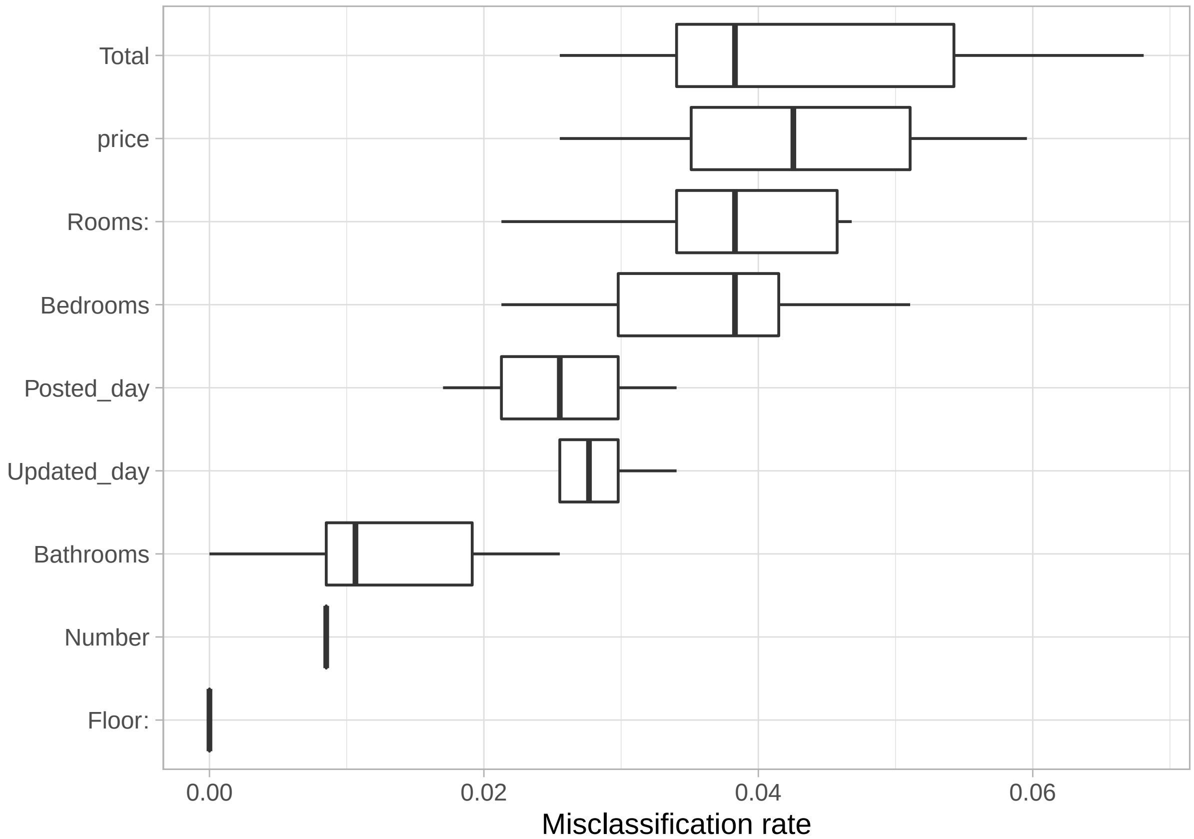
```
##   cluster size ave.sil.width
## 1       1  173           0.4
## 2       2   62           0.2
```



Variable importance

The lower the value for the misclassification rate is the better. Total, price, rooms, and bedrooms variables are the most important variables.

```
library(flexclust)
library(FeatureImpCluster)
km=kcca(df, k=2)
FeatureImp_km<-FeatureImpCluster(km, as.data.table(df))
plot(FeatureImp_km)
```

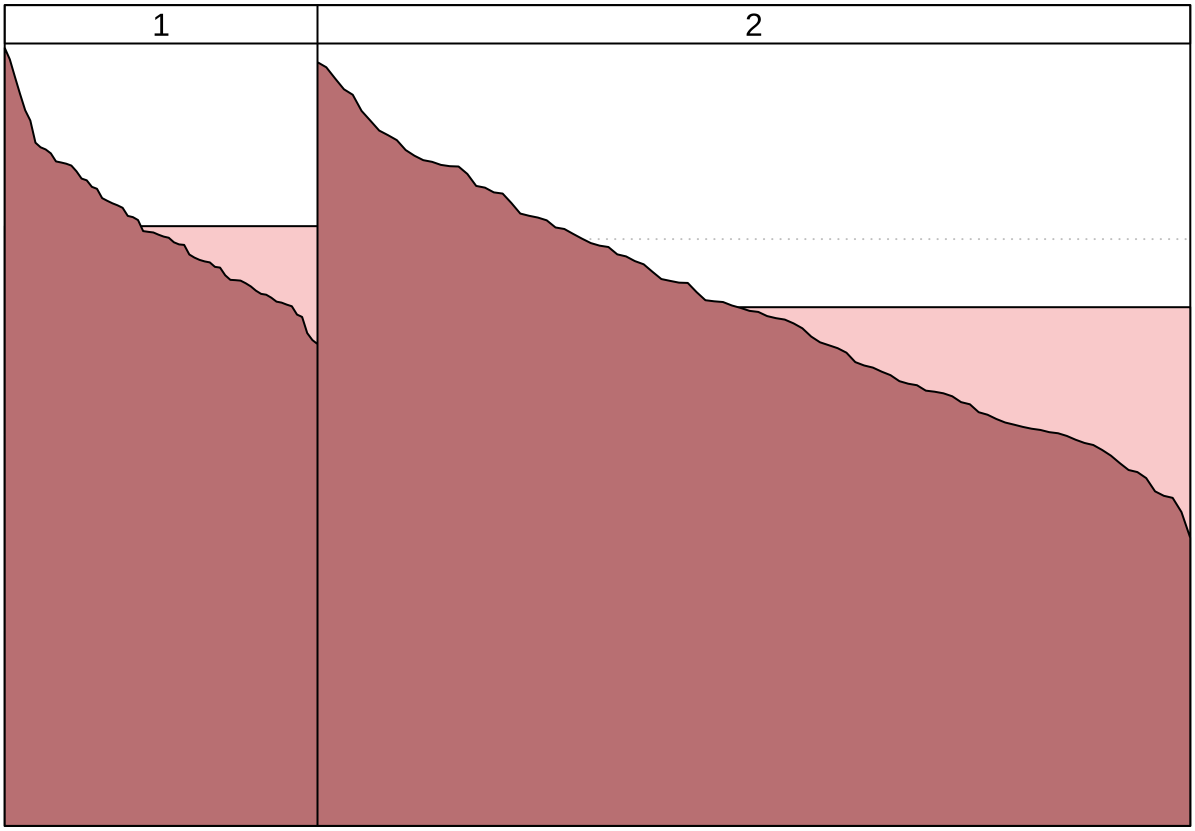


Shadow statistics

```r
library(flexclust)
library(FeatureImpCluster)
d1<-cclust(df,2,dist="euclidean")
shadow(d1)
```

```
##         1         2
## 0.7665264 0.6629363
```

```r
plot(shadow(d1))
```



Other statistics

```
km1=eclust(df,FUNcluster = c("kmeans"),k=2,graph = F,hc_metric = 'euclidean')
km1[2]
```

```
## $centers
##    Bathrooms   Bedrooms       Floor:      Number      price     Rooms:
## 1 -0.3008961 -0.3954708  0.009493993 -0.09453702 -0.4542309 -0.4106499
## 2  0.8395973  1.1034910 -0.026491302  0.26378879  1.2674507  1.1458458
##        Total Posted_day Updated_day
## 1 -0.4509896 -0.3608279  -0.2380238
## 2  1.2584064  1.0068262   0.6641633
```

```
km1[7]
```

```
## $size
## [1] 173  62
```

```
c1=data.frame(clusts=km1[1])
c1=cbind(c1,df)
c1 %>% group_by(cluster) %>% summarize(mean_bath=mean(Bathrooms),
                                      mean_bed=mean(Bedrooms),
```

```
                                       mean_floor=mean(`Floor:`),
                                       mean_num=mean(Number),
                                       mean_price=mean(price),
                                       mean_room=mean(`Rooms:`),
                                       mean_total=mean(Total),
                                       mean_posted=mean(Posted_day),
                                       mean_updated=mean(Updated_day))
```

```
## # A tibble: 2 x 10
##   cluster mean_bath mean_bed mean_floor mean_num mean_price mean_room mean_total
##     <int>     <dbl>    <dbl>      <dbl>    <dbl>      <dbl>     <dbl>      <dbl>
## 1       1    -0.301   -0.395    0.00949  -0.0945     -0.454    -0.411     -0.451
## 2       2     0.840    1.10    -0.0265    0.264       1.27      1.15       1.26
## # ... with 2 more variables: mean_posted <dbl>, mean_updated <dbl>
```

Hierarhical method

A hierarchical model with 2 clusters has the largest silhouette value. So the optimal number of clusters is 2. I will use this model from now on.

```
library(ClustGeo)
hier_res=data.frame(kcount=numeric(),Q=numeric(),sil=numeric())
for (i in 2:10) {
  dm<-dist(df)
  hc=hclust(dm,method = 'complete')
  clust<-cutree(hc,k=i)
  diss.mat<-dm
  Q_add=1-(withindiss(diss.mat,part=clust)/inertdiss(diss.mat))
  sil_add=data.frame(silhouette(clust,dm))
  sil_add=mean(sil_add$sil_width)
  add=data.frame(kcount=i,Q=Q_add,sil=sil_add)
  hier_res<-rbind(hier_res,add)
}
formattable(hier_res,list(Q=color_tile('transparent','lightgreen'),
                          sil=color_tile('transparent','lightgreen')))
```

kcount

Q

sil

2

0.2962455

0.3045658

3

0.3184600

0.2572574

4

0.4169089

0.2317155

5

0.4299244

0.2280326

6

0.4774159

0.1973070

7

0.5041263

0.1885081

8

0.5882374

0.2132025

9

0.6096024

0.2180174

10

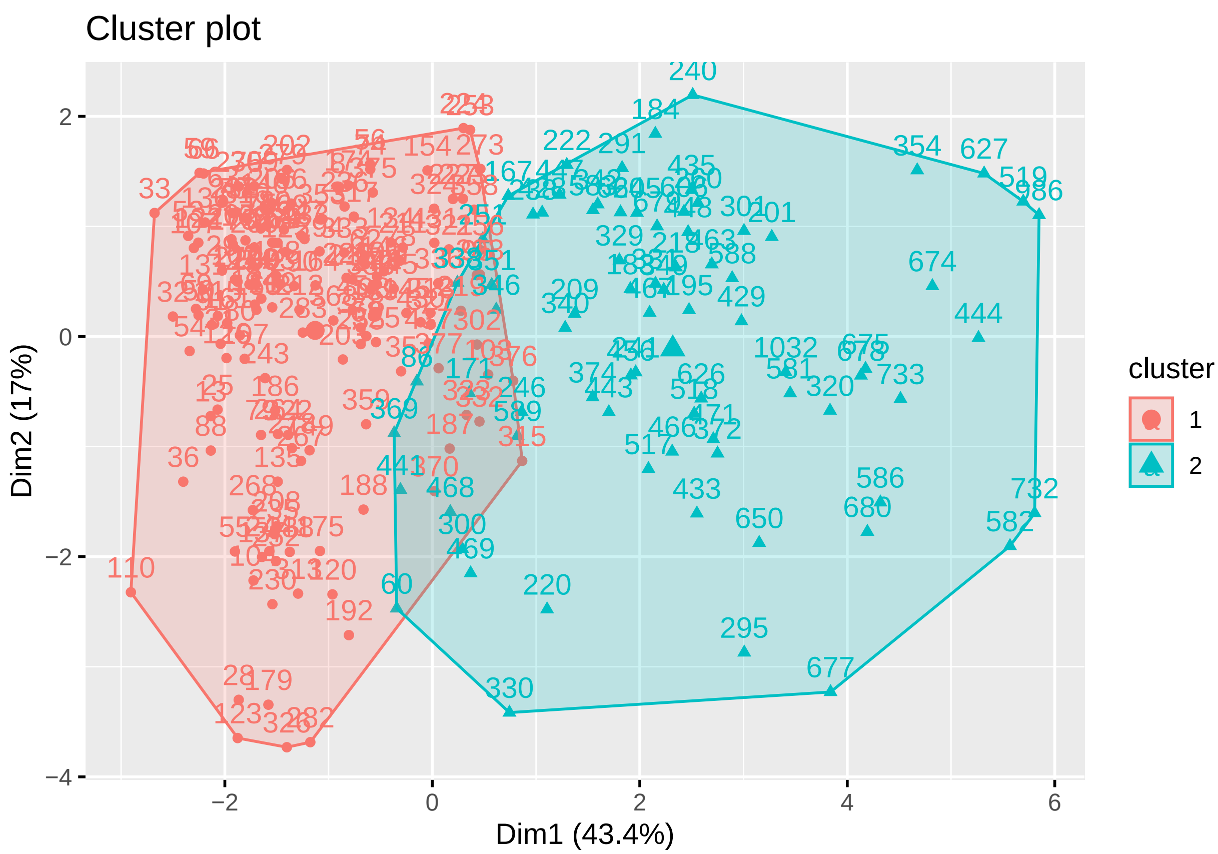
0.6251047

0.2142959

Result of hierarchical method

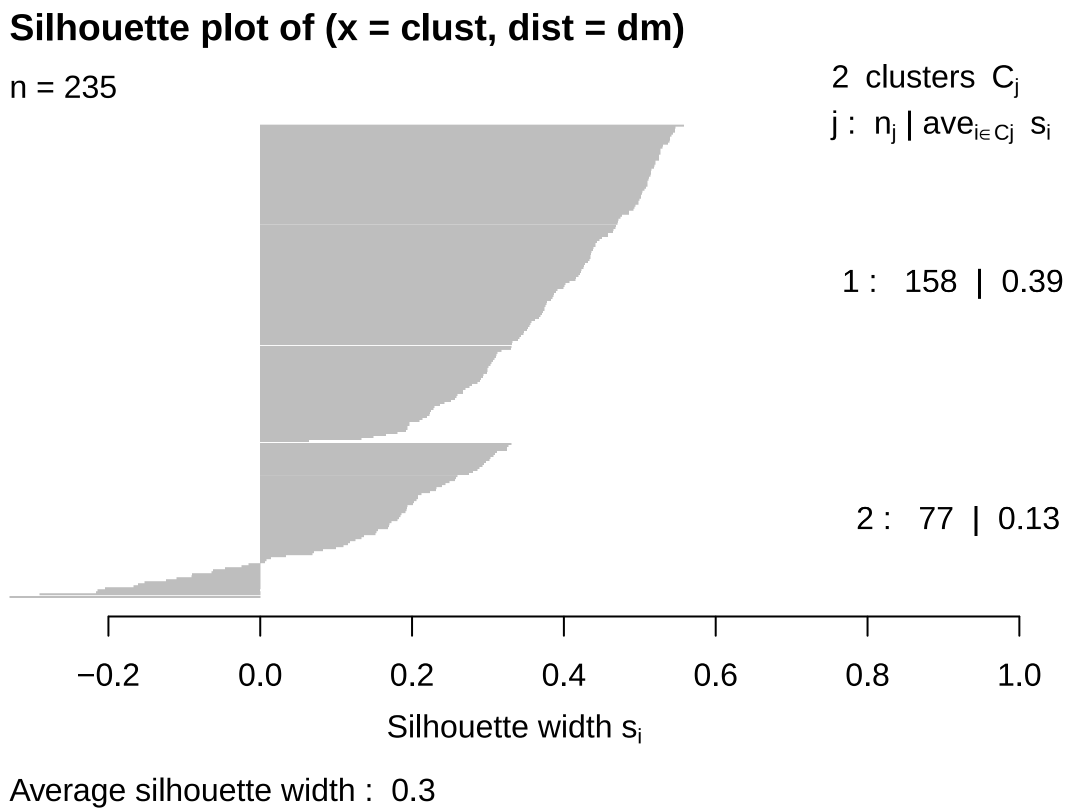
Cluster plot

```
dm<-dist(df)
hc=hclust(dm,method = 'complete')
clust<-cutree(hc,k=2)
diss.mat<-dm
fviz_cluster(list(data=df, cluster=clust))
```



Silhouette

```
dm<-dist(df)
hc=hclust(dm,method = 'complete')
clust<-cutree(hc,k=2)
diss.mat<-dm
plot(silhouette(clust,dm))
```



Statistic

```
dm<-dist(df)
hc=hclust(dm,method = 'complete')
clust<-cutree(hc,k=2)
diss.mat<-dm
y=data.frame(ks=clust)
y_sum=y %>% group_by(ks) %>% summarize(count=n())
formattable(y_sum)
```

ks

count

1

158

2

77

```
res_hiar_sta<-cbind(df,y)
res_hiar_sta %>% group_by(ks) %>% summarize(mean_bath=mean(Bathrooms),
                                      mean_bed=mean(Bedrooms),
                                      mean_floor=mean(`Floor:`),
```

```
                                          mean_num=mean(Number),
                                          mean_price=mean(price),
                                          mean_room=mean(`Rooms:`),
                                          mean_total=mean(Total),
                                          mean_posted=mean(Posted_day),
                                          mean_updated=mean(Updated_day))
```

```
## # A tibble: 2 x 10
##      ks mean_bath mean_bed mean_floor mean_num mean_price mean_room mean_total
##   <int>     <dbl>    <dbl>      <dbl>    <dbl>      <dbl>     <dbl>      <dbl>
## 1     1    -0.299   -0.447    -0.0388   -0.165     -0.514    -0.459     -0.483
## 2     2     0.614    0.918     0.0797    0.339      1.05      0.943      0.992
## # ... with 2 more variables: mean_posted <dbl>, mean_updated <dbl>
```

Difference between results of non-hierarchical and hierarchical method

Let's explain the difference between these two cluster models and the result of the project. * In Kmeans, 173 ads in cluster 1 and 62 ads in cluster 2 * In Hierarchical, 158 ads in cluster 1 and 77 ads in cluster 2 * Silhouette value of Kmeans is higher than the hierarchical model's * The 2nd cluster includes apartments that are more expensive, have more rooms, have a larger area, and are located at a higher elevation. * Cluster 1 - there are only ads for cheap property houses. * Cluster 2 - there are ads for Luxury houses with a high price.

```
formattable(table1)
```

model

sil

num_clus1

num_clus2

kmeans

0.3446526

173

62

hierarchical

0.3045658

158

77

```
formattable(table2)
```

model

cluster

avg_bathroom

avg_bedroom

avg_floor

avg_number

avg_price

avg_room

avg_total

avg_posted

avg_updated

kmeans

1

-0.3008961

-0.3954708

0.009493993

-0.09453702

-0.4542309

-0.4106499

-0.4509896

-0.3608279

-0.2380238

hierarchical

1

-0.2990163

-0.4472879

-0.038823905

-0.16517741

-0.5140443

-0.4594015

-0.4834434

-0.4245671

-0.2937548

```
formattable(table3)
```

model

cluster

avg_bathroom

avg_bedroom

avg_floor

avg_number

avg_price

avg_room

avg_total

avg_posted

avg_updated

2

kmeans

2

0.8395973

1.1034910

-0.02649130

0.2637888

1.267451

1.1458458

1.2584064

1.0068262

0.6641633

21

hierarchical

2

0.6135658

0.9178114

0.07966464

0.3389355

1.054792

0.9426681

0.9920008

0.8711896

0.6027695

Prediction

I will predict set.test using kmeans model. Test data is part of all data. To predict the 6 data below using the Kmeans model, all values belong to the 2nd cluster.

```r
set.train=df[-c(230:235),]
set.test=df[c(230:235),]
km_model=eclust(set.train,"kmeans",hc_metric = 'euclidean',k=2,graph = F)
km2.kcca<-as.kcca(km_model, set.train) # it is important
km2.pred<-predict(km2.kcca, set.test)
km2.pred
```

```
##  679  680  732  733  986 1032
##    2    2    2    2    2    2
```