Business Data Analytics. Practice Session

Customer segmentation

Today practice session is about customer segmentation, which has become an essential part of marketing. Now we will deal with the RFM model, while later on we will demonstrate the automatic segmentation discovery via kmeans and hierarchical clustering.

RFM

Let's first load a dataset, where there is an information about clients and their orders along with the order details:

```
orders <- read.table("C://Users/Margarit Shmavonyan/Desktop/orders_rfm.csv", header=TRUE, sep=',')
```

The next logical step is to investigate what kind of data you have. In order to do that we need to keep the goal in mind. In our case we need to perform **RFM analysis**, which means to calculate three key measures:

1. **R** - recency score
2. **F** - frequency score
3. **M** - monetary score

In our case recency will be expressed as number of days since the last order, frequency will be defined as the number of purchased items, and monetary score is the total amount spent during the defined period. There are a lot of variations of these definitions. For example, when necessary, you may want to aggregate recency component on a yearly basis rather than using days; frequency and monetary scores can be expressed as the percentage of one period to another, etc. Moreover, it is important to define the **period** under investigation:

```
#reporting_date <- as.Date('2017-04-09', format='%Y-%m-%d') # any date of interest
```

```
orders$order_date <- as.Date(orders$order_date, format='%Y-%m-
%d') # transform to the date format
reporting_date <- max(orders$order_date) # we will use all the
data
# by defining reporting date as of the last available purchase.
reporting_date
```

```
## [1] "2017-04-11"
```

As we discussed previously, the descriptive part helps to get sense of the data:

```
# base analysis - how many clients, how many unique products, etc
length(unique(orders$client_id))
```

```
## [1] 457
```

```
table(orders$product)
```

```
##
##    a    b    c
## 2635 1147  596
```

Note. We will load packages on the go in order to follow where the functions originate from. Usually it is more convenient to collect the list of the loaded libraries at the beginning of the script.

We will calculate the frequency, recency and monetary values in the following way:

```
library(dplyr)
frm_tbl_initial <- orders %>%
  group_by(client_id) %>%
  summarise(order_frequency = n(), order_recency =
min(reporting_date - order_date),
            order_monetary = sum(money_spent))
head(frm_tbl_initial)
```

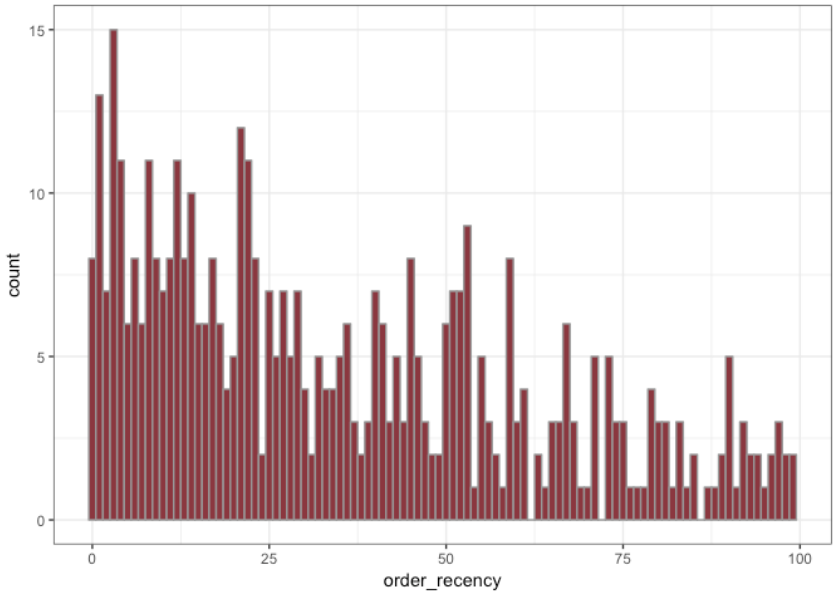
```
## # A tibble: 6 x 4
##   client_id order_frequency order_recency order_monetary
##       <int>           <int>         <time>          <dbl>
## 1         1               6       37 days         570.40
## 2         2               7       34 days         537.62
## 3         3              12       27 days        1123.90
## 4         4              11       17 days        1755.30
## 5         5               8        8 days         492.74
## 6         6              15       12 days        1196.70
```

Order recency is a time object (days), which can cause errors later. We need to transform it into numeric value:

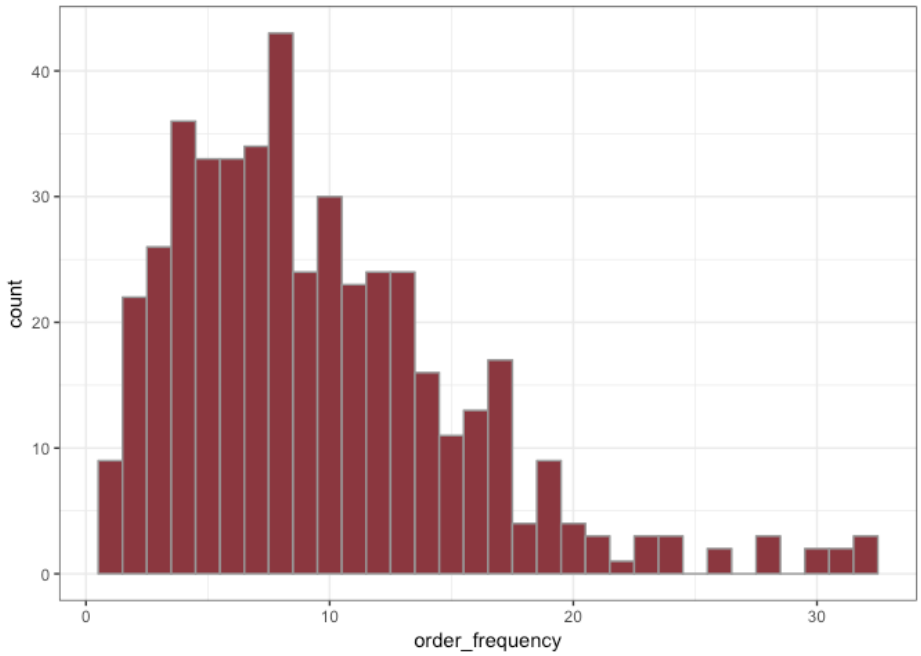
```
class(frm_tbl_initial$order_recency)
## [1] "difftime"
frm_tbl_initial$order_recency <-
as.numeric(frm_tbl_initial$order_recency)
```

We will investigate the distribution of the values in our RFM calculations (I hope that this code is already painfully familiar to you):

```
library(ggplot2)
ggplot(frm_tbl_initial, aes(x=order_recency)) +
geom_histogram(fill='#8b3840', color='grey60', binwidth = 1) +
theme_bw()
```



```
ggplot(frm_tbl_initial, aes(x=order_frequency)) +
geom_histogram(fill='#8b3840', color='grey60', binwidth=1) +
theme_bw()
```



The RFM approach is widely used and has a lot of use cases on practice. We can find our best and worst customers. It is clear and reasonably easy to interpret (the only problem is how to visualize the three dimensions simultaneously). However, it requires your attention and highly depends on your choices. It is unlikely you can discover something unexpected. Let's see how it can be done.

K-means clustering

Firstly, if we want to use k-means approach we need to scale features by subtracting mean and dividing by standard deviation. This can be done using `scale` function. We firstly will cluster the previous dataset:

```
data_clustering <- frm_tbl_initial %>%
  mutate(order_frequency=scale(order_frequency),
         order_recency=scale(order_recency),
         order_monetary=scale(order_monetary))
head(data_clustering)
```

```
## # A tibble: 6 x 4
##   client_id order_frequency order_recency order_monetary
##       <int>           <dbl>         <dbl>          <dbl>
## 1         1      -0.5996230    0.07218577     -0.5799359
## 2         2      -0.4321244   -0.03876343     -0.6308460
## 3         3       0.4053686   -0.29764490      0.2796954
## 4         4       0.2378700   -0.66747556      1.2603119
## 5         5      -0.2646258   -1.00032316     -0.7005483
## 6         6       0.9078644   -0.85239090      0.3927598
```

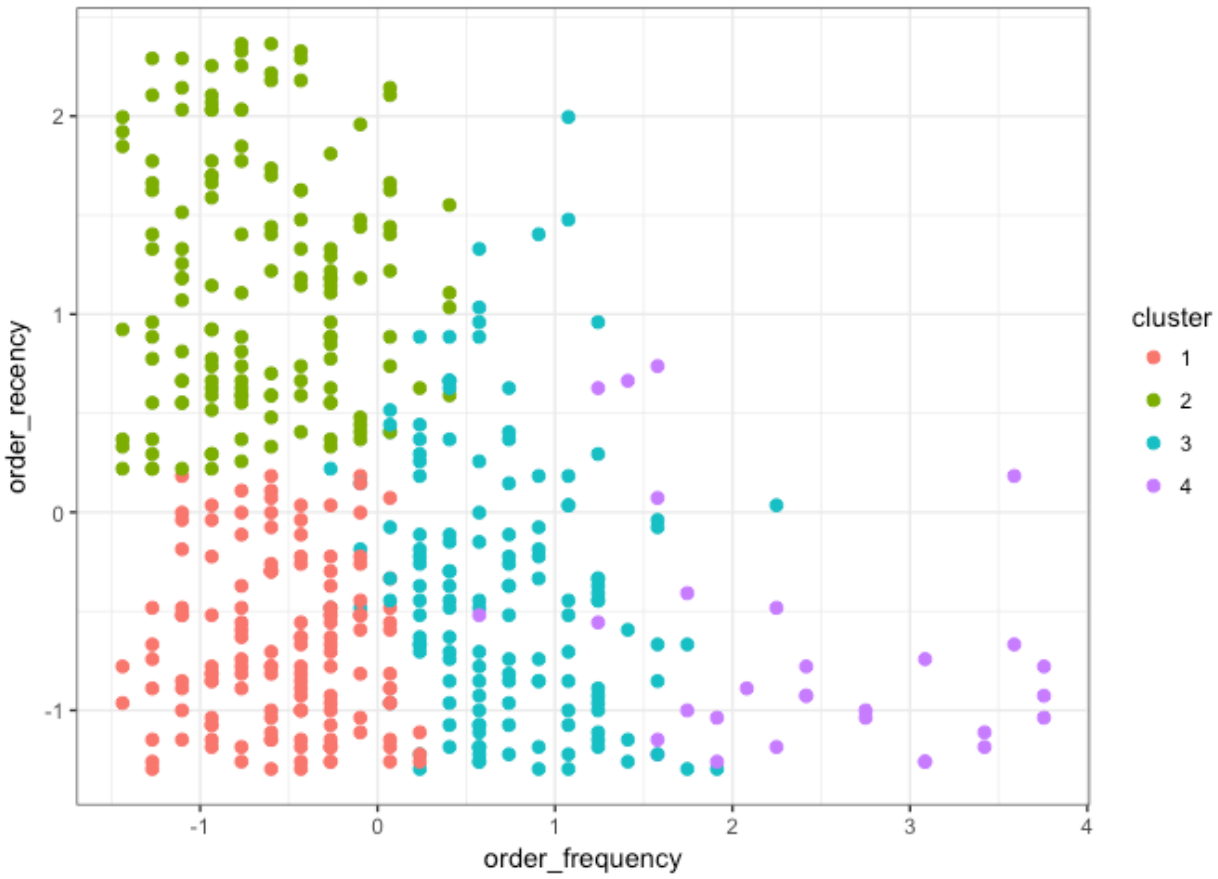
Then, it is wise to explore what parameters kmeans method requires:

```
?kmeans
clusters <- kmeans(data_clustering[,-1], centers = 4, nstart=20)
clusters
## K-means clustering with 4 clusters of sizes 141, 145, 141, 30
##
## Cluster means:
##   order_frequency order_recency order_monetary
## 1      -0.5283470    -0.6758689     -0.5509910
## 2      -0.6700879     1.1673946     -0.6003760
## 3       0.7071037    -0.3752831      0.6542823
## 4       2.3986019    -0.7019931      2.4163480
##
## Clustering vector:
##   [1] 1 1 3 3 1 3 2 1 3 3 1 1 2 3 2 2 4 2 2 4 1 3 2 3 1 2 2 3
3 2 3 3 2 1 1
##  [36] 2 3 3 3 1 3 3 1 2 2 2 1 3 2 3 2 1 1 1 4 1 1 3 2 2 3 2 1
2 4 3 1 2 1 2
##  [71] 2 4 1 1 1 3 2 1 2 3 3 2 3 2 2 1 3 1 3 3 1 2 1 2 2 3 2 3
2 1 1 1 3 1 2
## [106] 3 2 3 3 1 2 2 4 4 1 2 3 1 3 2 3 4 1 1 3 2 1 2 1 3 3 3 2
3 3 2 3 3 3 1
## [141] 1 3 2 1 3 1 3 1 3 3 1 3 3 2 3 3 4 1 1 3 3 2 3 3 4 3 2 2
1 4 4 2 4 1 2
## [176] 2 2 3 2 3 2 3 2 2 1 1 2 3 2 2 1 2 1 1 1 2 2 3 3 1 1 2 3
1 1 3 3 3 4 2
## [211] 2 3 1 1 1 1 1 2 2 2 1 3 2 3 2 4 2 1 2 3 2 2 3 2 3 3 2 3
2 2 2 4 1 3 1
## [246] 2 1 3 3 1 2 1 1 2 3 1 2 4 2 1 1 2 1 3 3 2 3 1 3 1 4 1 3
2 3 1 1 1 3 3
```

```
## [281] 1 2 2 3 1 1 3 2 3 1 2 3 3 3 1 2 3 3 2 2 3 1 1 3 1 2 2 1
1 3 3 2 2 4 2
## [316] 3 2 4 4 3 3 2 1 1 1 3 2 2 2 1 1 1 1 1 2 2 3 2 1 2 3 2 2
2 1 1 2 2 3 2
## [351] 2 1 4 2 2 1 1 3 1 1 2 2 1 3 2 1 2 1 1 2 3 2 2 3 2 3 2 3
3 2 1 2 1 1 3
## [386] 3 1 3 3 1 2 4 2 1 3 1 2 4 2 1 4 1 1 3 3 2 1 1 1 1 3 4 2
3 1 3 3 2 1 2
## [421] 2 4 4 1 3 3 3 2 2 1 1 3 3 2 1 1 1 1 1 3 3 3 3 2 2 1 4 2
1 4 3 3 1 3 1
## [456] 3 2
##
## Within cluster sum of squares by cluster:
## [1]  73.16152 135.21116 134.08466  50.72476
##  (between_SS / total_SS =  71.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

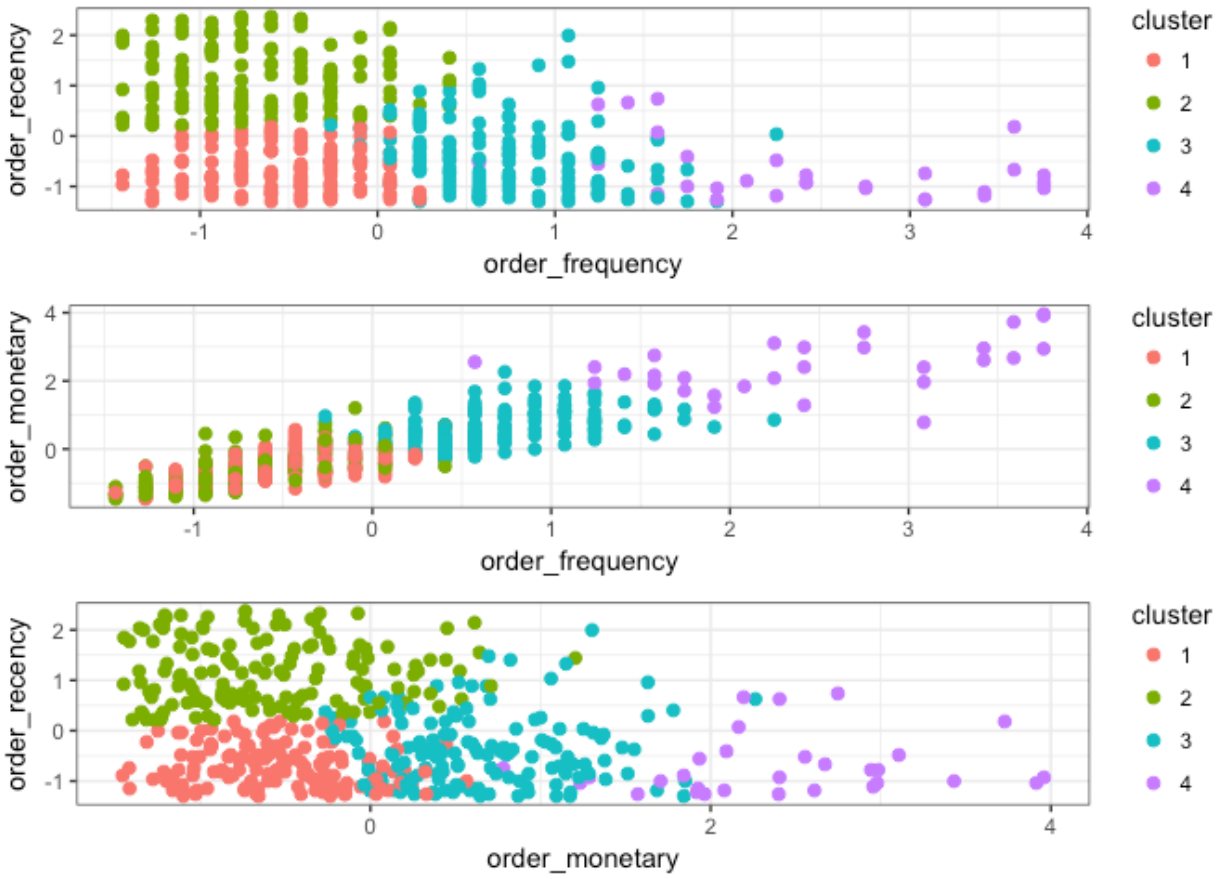
Let us assign clusters to the clients, by adding it to the data frame

```
data_clustering$cluster <- as.factor(clusters$cluster) # we need
it to be a factor for the plot
ggplot(data_clustering, aes(x=order_frequency, y=order_recency,
color=cluster)) + geom_point(size=2) + theme_bw()
```



We can plot all three plots together:

```
library(gridExtra)
p1 <- ggplot(data_clustering, aes(x=order_frequency,
y=order_recency, color=cluster)) + geom_point(size=2) +
theme_bw()
p2 <- ggplot(data_clustering, aes(x=order_frequency,
y=order_monetary, color=cluster)) + geom_point(size=2) +
theme_bw()
p3 <- ggplot(data_clustering, aes(x=order_monetary,
y=order_recency, color=cluster)) + geom_point(size=2) +
theme_bw()
grid.arrange(p1,p2,p3, nrow=3)
```

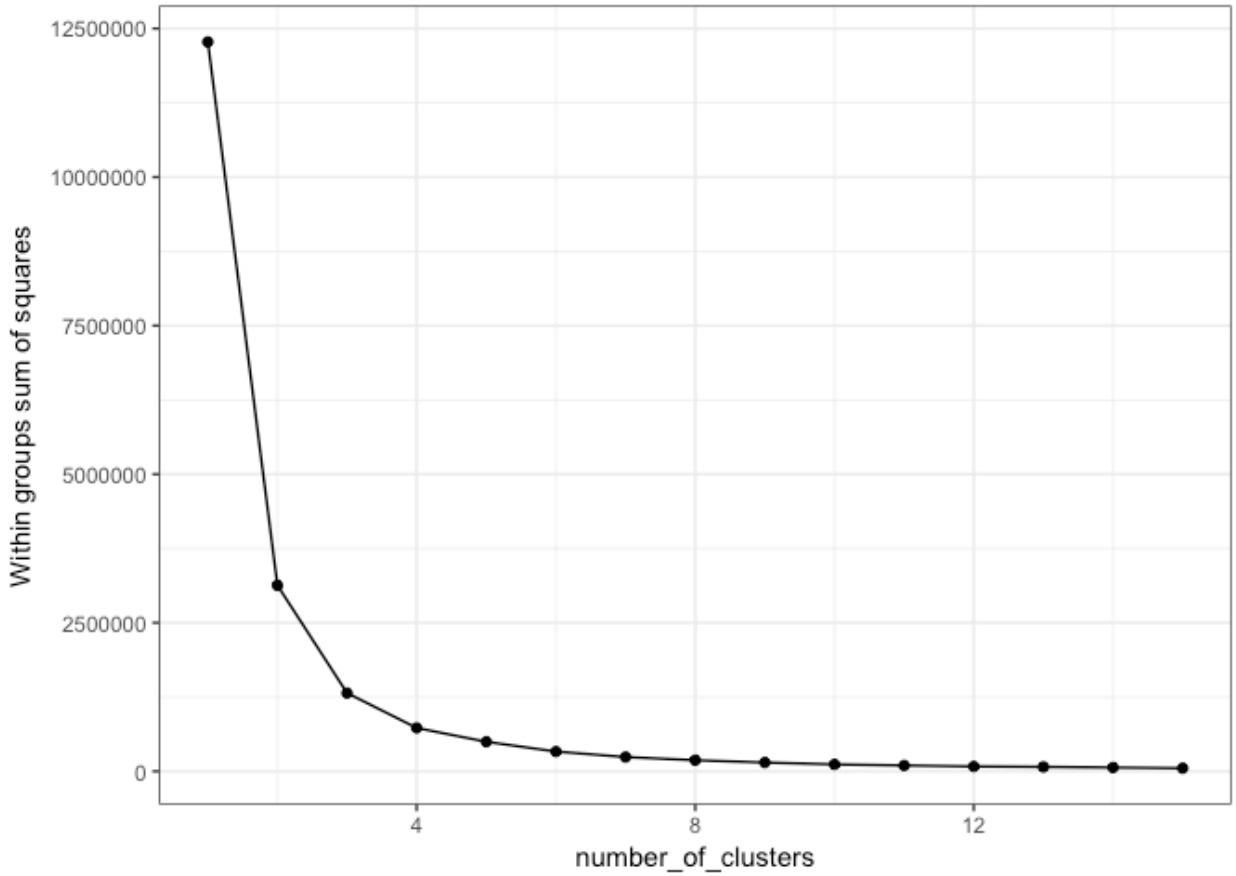


The reasonable question to ask is how to choose number of clusters? It is usually the iterative process where the clusters should be manually inspected. You should be asking the following questions. Are they different enough? Do they make sense? If I change number of clusters, does the structure change a lot or the pattern of clusters persists? There are also various techniques how to identify number of clusters. These methods provide you with some additional intuition (but they do not guarantee this is the best number of clusters for your problem). Also, be aware that the clusters can be found even when the underlying structure does not have them. To explore the possibilities take a look here: choosing number of clusters.

```
# finding optimal number of clusters
elbow_method <- function(data, max_k=15){
  require(ggplot2)
  wss <- (nrow(data)-1)*sum(apply(data,2,var))
  for (i in 1:max_k){
    wss[i] <- sum(kmeans(data_clustering, centers=i)$withinss)
```

```r
  }
  p <- data.frame(number_of_clusters=c(1:max_k), wss=wss) %>%
    ggplot(aes(x=number_of_clusters, y=wss, group=1)) + geom_point() +
    geom_line() + theme_bw() + ylab("Within groups sum of squares")
  return(print(p))
}

# apply the function
elbow_method(data_clustering[,-1], max_k=15)
```



Usually there is a tradeoff, we want as few clusters as possible, because it is easy to interpret them (nobody wants to interpret and act upon 79 segments of customers), but we want to minimize the `wss` statistics. The idea behind the elbow methods is that we want to find a point, where the wss is small, but the function does not become smooth (adding more clusters does not reduce much of the variance), so we are looking for an angle in the curve.