SistemaRecomendacaoInstacartMBA

Eduardo de Souza Dias

3/28/2020

```
# Project 10 - BusinessAnalytics - DSA (part of Formação cientista de dados)
# Forecasting products that users will buy on their next order

# At this project, we will analyse datasets provided by Instacart in their Kaggle competition.
# The idea is to understand users' behavior, how products are related and than forcast what
# product will be in the next user' order. For that, we have a dataset with the last
# users' orders, a dataset with the last order and information about products. Price and
# quantity are not avaliable.

# Define folder
setwd("C:/Cursos/FCD/05-BusinessAnalytics/Cap10-ProjetosFeedback/Projeto10")
getwd()
```

```
## [1] "C:/Cursos/FCD/05-BusinessAnalytics/Cap10-ProjetosFeedback/Projeto10"
```

```
library(data.table)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
library(ggplot2)
library(scales)
library(knitr)
library(arules)
```

```
## Warning: package 'arules' was built under R version 3.6.3
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
##
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
library(arulesViz)
```

```
## Warning: package 'arulesViz' was built under R version 3.6.3
```

```
## Loading required package: grid
```

```
## Registered S3 method overwritten by 'seriation':
##   method         from
##   reorder.hclust gclus
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 3.6.3
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(DMwR)
```

```
## Warning: package 'DMwR' was built under R version 3.6.3
```

```
## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(stringr)

# Datasets ------------------------------------------------------------------------
dfAisles <- fread("aisles.csv")
dfDepartments <- fread("departments.csv")
dfOrderProductsPrior <- fread("order_products__prior.csv")
dfOrderProductsTrain <- fread("order_products__train.csv")
dfOrders <- fread("orders.csv")
dfProducts <- fread("products.csv")

# Jutando os datasets
df <- left_join(dfOrderProductsPrior,dfProducts,by="product_id")
df <- left_join(df,dfAisles,by="aisle_id")
df <- left_join(df,dfDepartments,by="department_id")
df <- left_join(df,dfOrders,by="order_id")
df <- df[, c(1,2,10,5,8,9,3,4,12,13,14,15)]

dfOrderProductsTrain <- left_join(dfOrderProductsTrain,dfOrders, by='order_id')

dfProducts <- dfProducts %>% left_join(dfAisles) %>% left_join(dfDepartments) %>%
  select(-aisle_id, -department_id)
```

```
## Joining, by = "aisle_id"
```

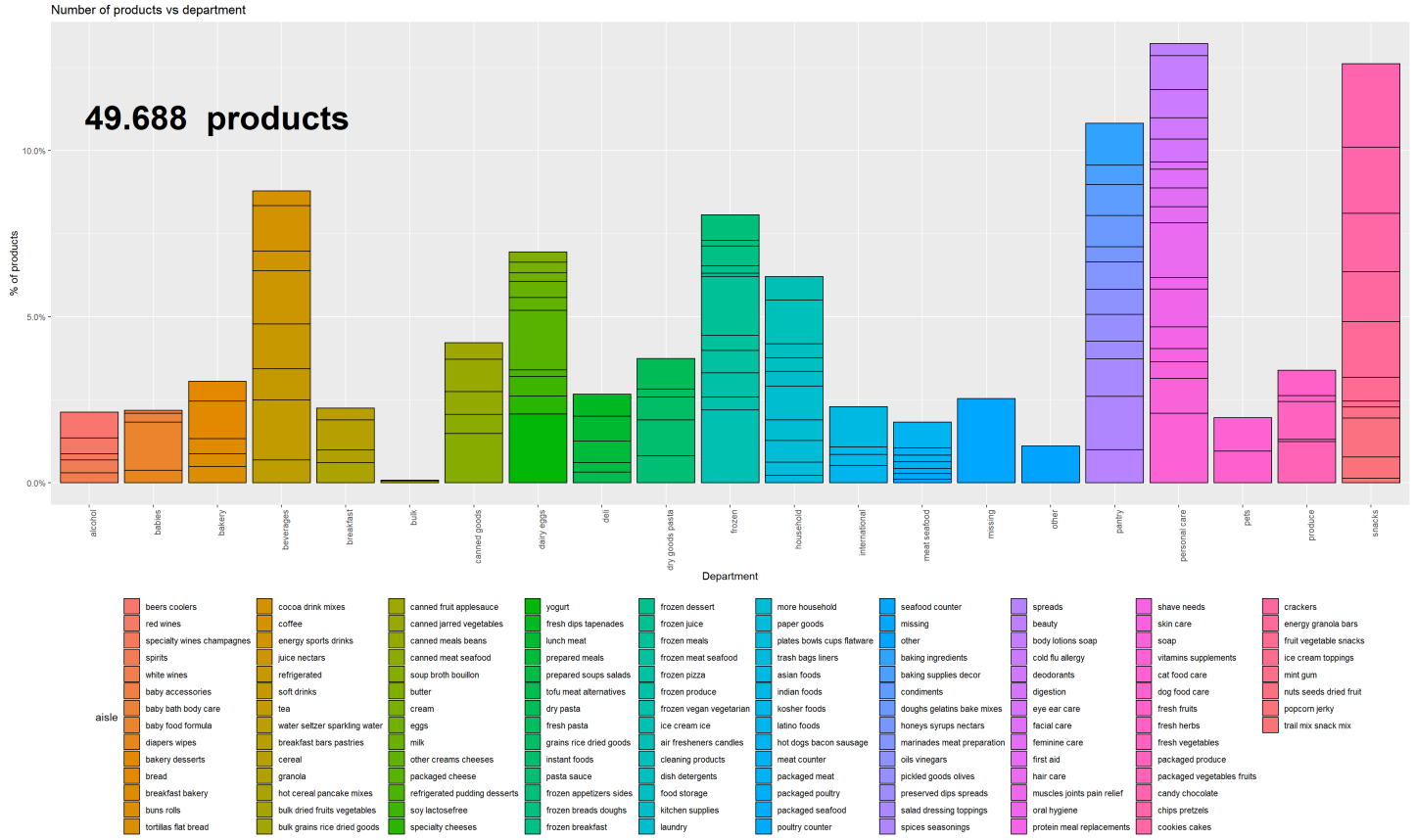
```
## Joining, by = "department_id"
```

```
rm(dfAisles, dfDepartments, dfOrderProductsPrior, dfOrders)

# Exploratory analisys ------------------------------------------------------------

# All Products and departments
dfProducts <- dfProducts %>% arrange(department, aisle)
dfProducts$aisle <- factor(dfProducts$aisle, levels=unique(dfProducts$aisle))

print(paste('Number of departments:', length(unique(dfProducts$department))))
```

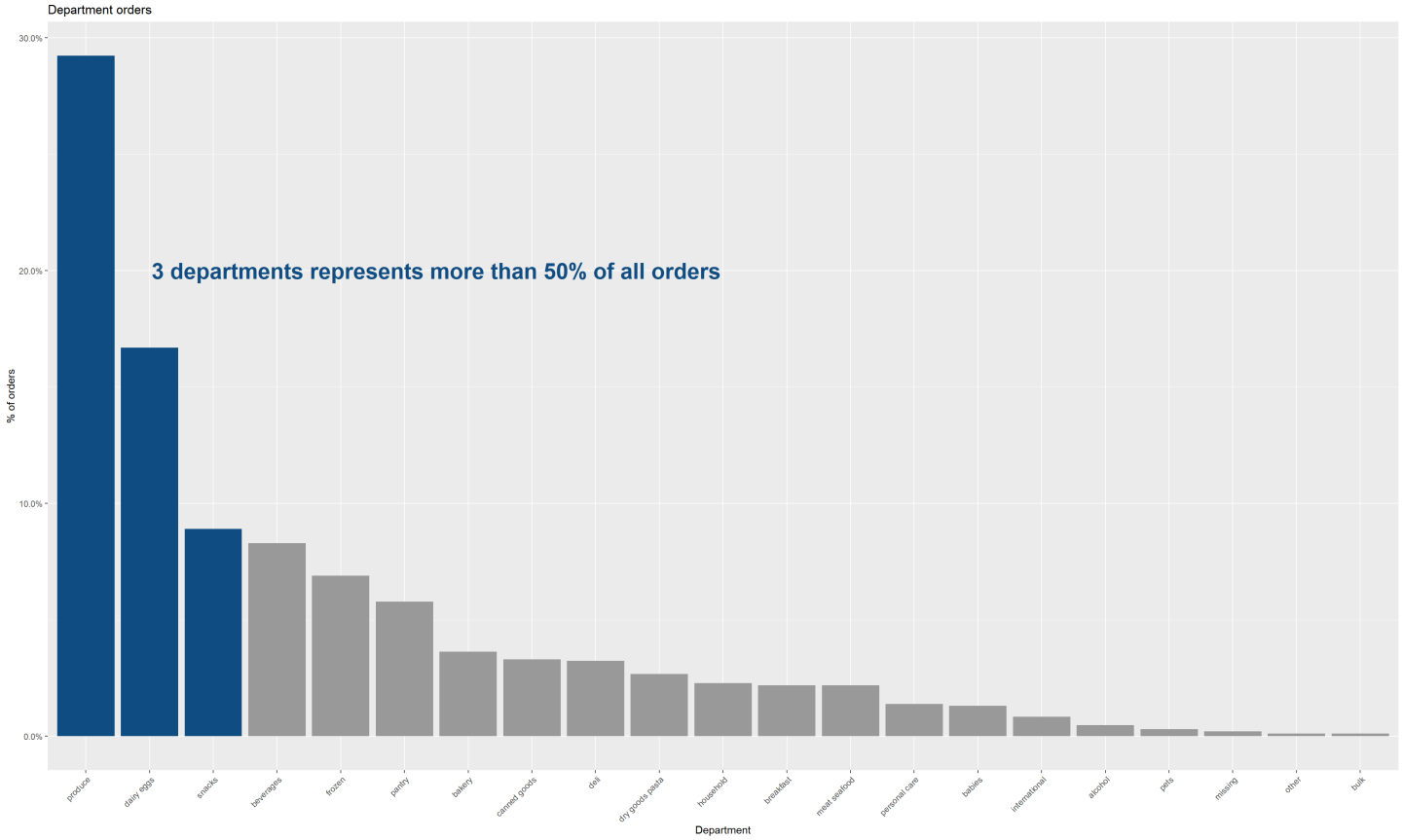
```
## [1] "Number of departments: 21"
```

```
dfProducts %>%
  group_by(department,aisle) %>%
  summarize(contar = n()) %>%
  mutate(perc = contar / length(dfProducts$product_id)) %>%
  select(-contar) %>%
  ggplot() + geom_bar(aes(department, perc, fill=aisle), stat="identity", colour="black") +
  labs(title="Number of products vs department", x="Department", y="% of products") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1), legend.position = 'bottom') +
  scale_y_continuous(labels = percent) + guides(fill=guide_legend(ncol=10)) +
  annotate("text", x = 3, y = 0.11, label = paste(format(length(unique(dfProducts$product_id)), big.mark='.')," products"), fontface = 2, size = 12)
```

```
## Warning in prettyNum(.Internal(format(x, trim, digits, nsmall, width, 3L, : 'big.mark' and 'decimal.mark' are both '.', which could be confusing
```

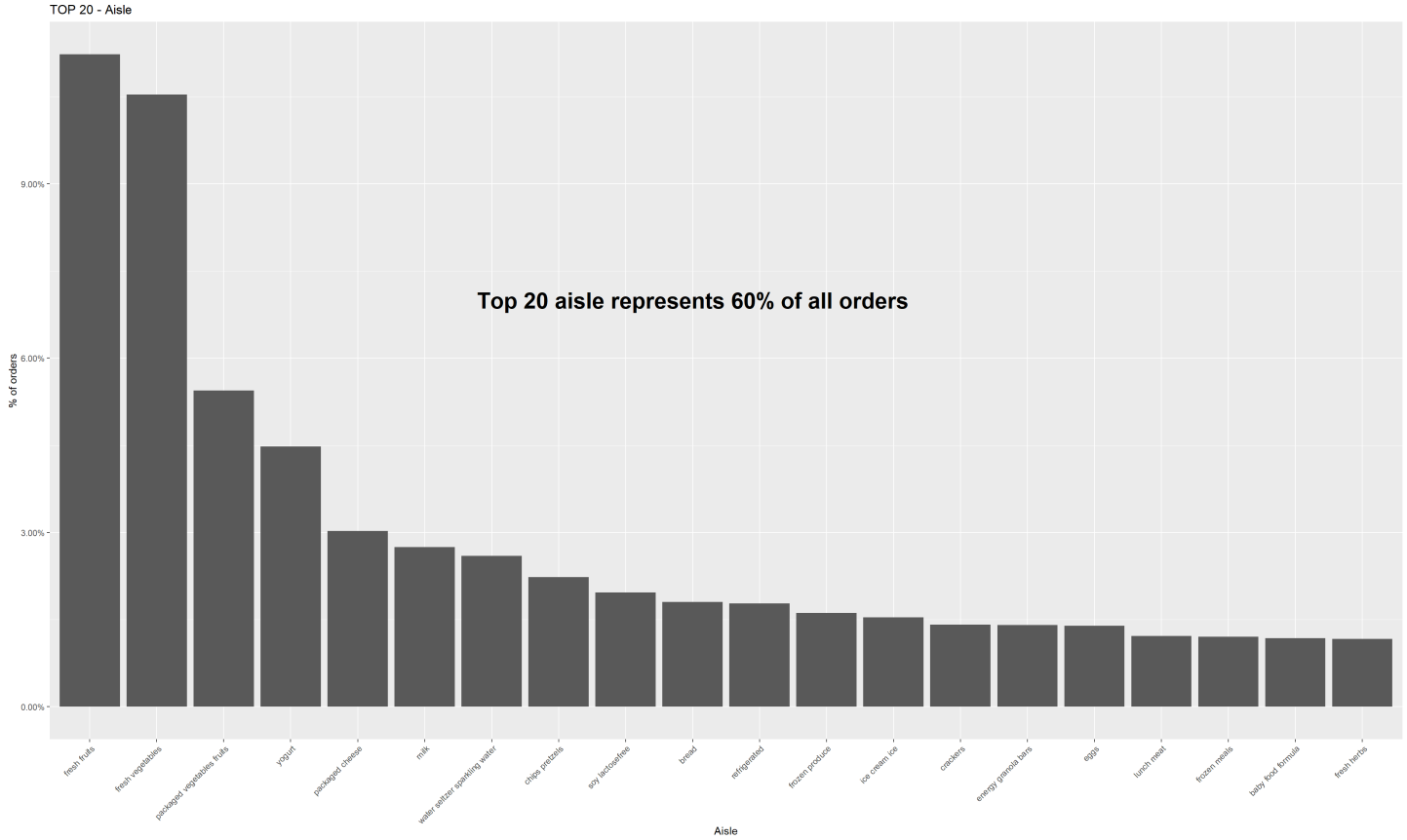



```
# Department orders
df %>%
  group_by(department) %>%
  summarize(contar = n()) %>%
  mutate(perc = contar / sum(contar)) %>%
  mutate(highlight = ifelse(department == 'produce' | department == 'dairy eggs' | department == 'snacks', F,T)) %>%
  ggplot + geom_bar(aes(x=reorder(department,-perc), y=perc, fill=highlight), stat="identity") +

  labs(title="Department orders", x="Department", y="% of orders") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none") +
  scale_y_continuous(labels = percent) +
  annotate("text", x = 6.5, y = 0.2, label = "3 departments represents more than 50% of all orders", fontface = 2, size = 8, color="#0f4c81") +
  scale_fill_manual(values=c("#0f4c81", "#999999"))
```



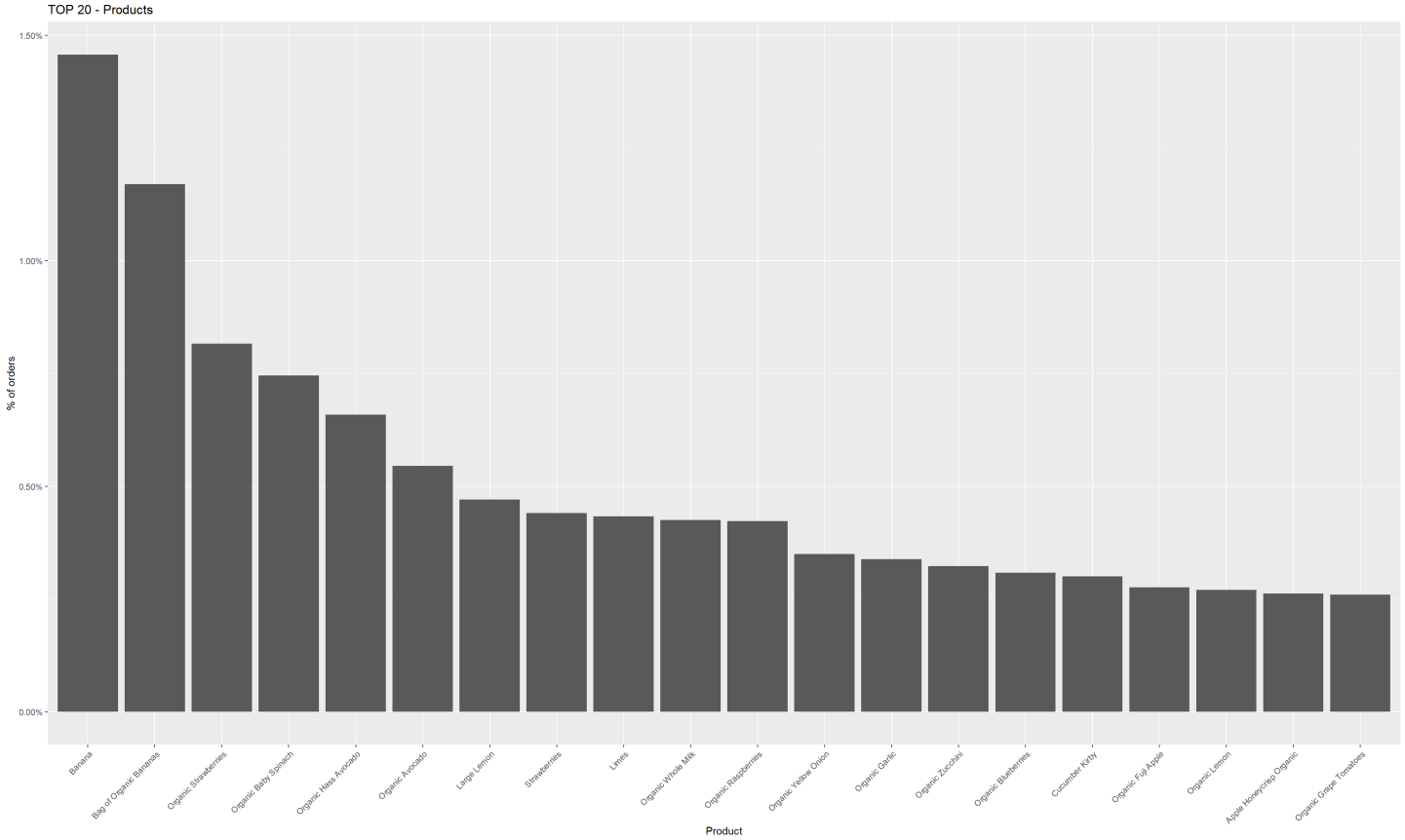

```
# TOP 20 aisle
df %>%
  group_by(aisle) %>%
  summarize(contar = n()) %>%
  mutate(perc = contar / sum(contar)) %>%
  top_n(20) %>%
  ggplot + geom_bar(aes(x=reorder(aisle,-perc), y=perc), stat="identity") +
    labs(title="TOP 20 - Aisle", x="Aisle", y="% of orders") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    scale_y_continuous(labels = percent) +
    annotate("text", x = 10, y = 0.07, label = "Top 20 aisle represents 60% of all orders", fontface = 2, size = 8,)
```

```
## Selecting by perc
```

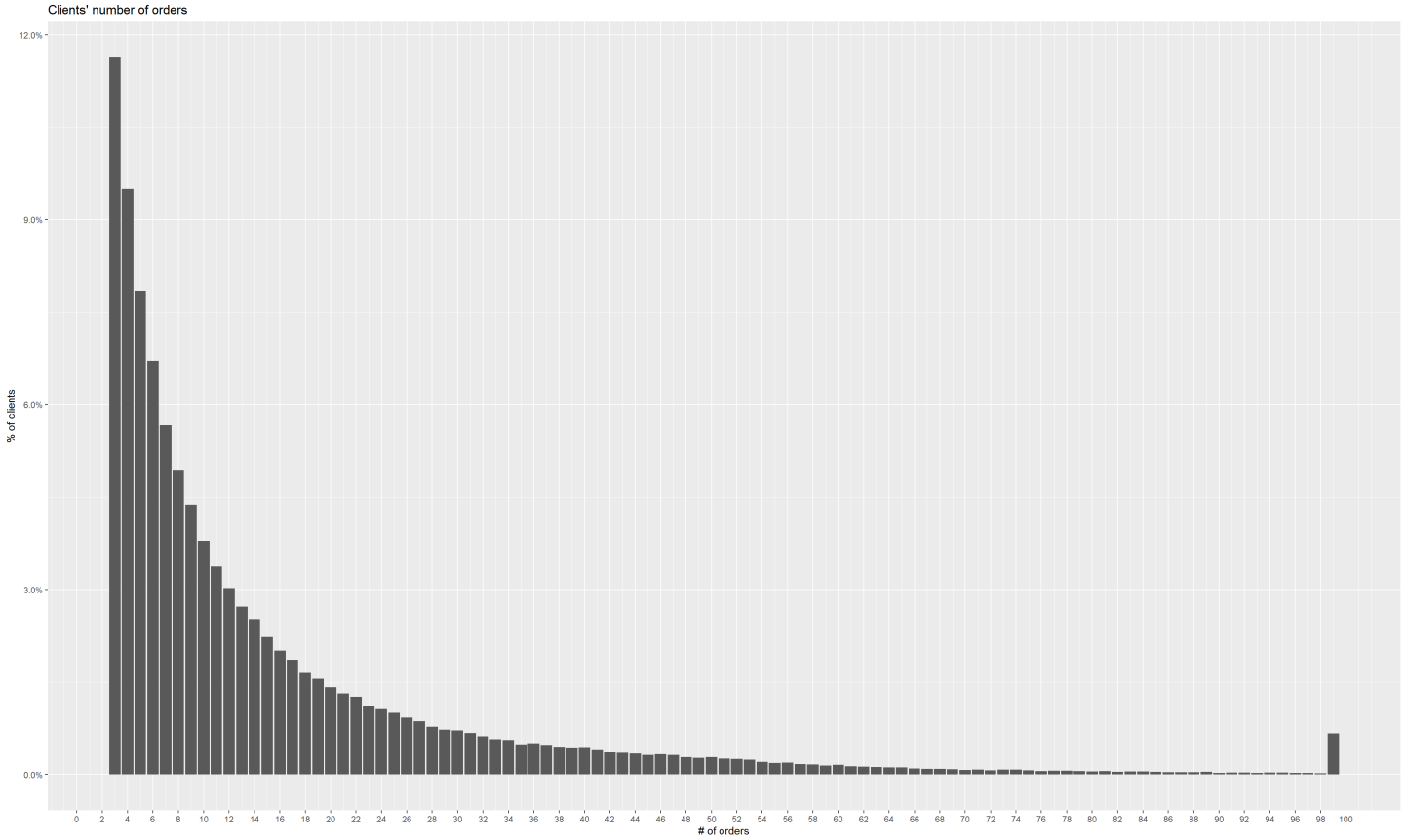



```
# Top 20 products
df %>%
  group_by(product_name) %>%
  summarize(contar = n()) %>%
  mutate(perc = contar / sum(contar)) %>%
  top_n(20) %>%
  ggplot + geom_bar(aes(x=reorder(product_name,-perc), y=perc), stat="identity") +
  labs(title="TOP 20 - Products", x="Product", y="% of orders") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_y_continuous(labels = percent)
```

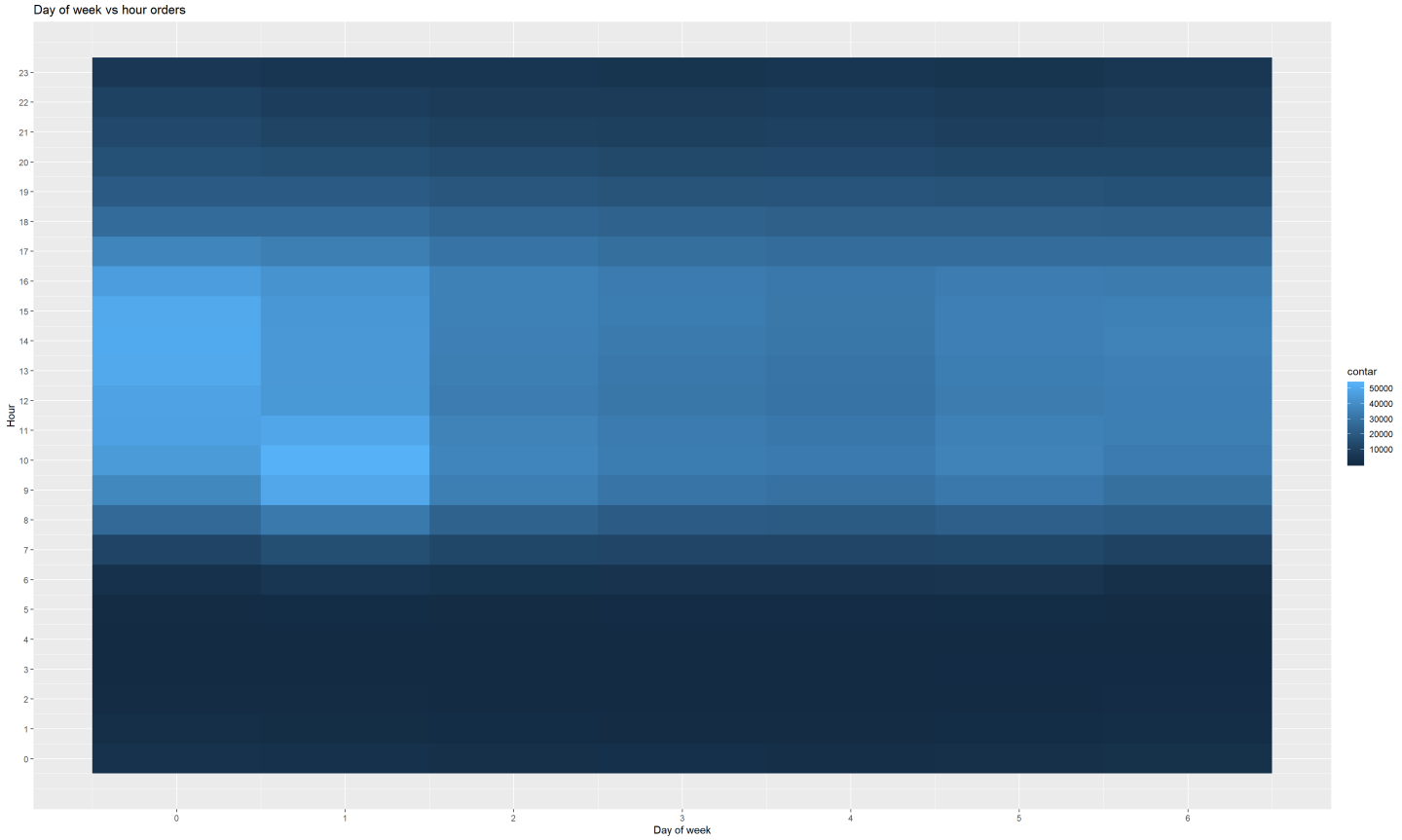
```
## Selecting by perc
```




```
# Clients' number of orders
df %>%
  select(order_id, user_id) %>%
  group_by(user_id) %>%
  distinct() %>%
  summarize(numOrdens = n()) %>%
  group_by(numOrdens) %>%
  summarise(contar = n()) %>%
  mutate(perc = contar / sum(contar)) %>%
  ggplot + geom_bar(aes(x=numOrdens, y=perc), stat="identity") +
  labs(title="Clients' number of orders", x="# of orders", y="% of clients") +
  scale_y_continuous(labels = percent) + scale_x_continuous(breaks = seq(0, 100, by = 2))
```



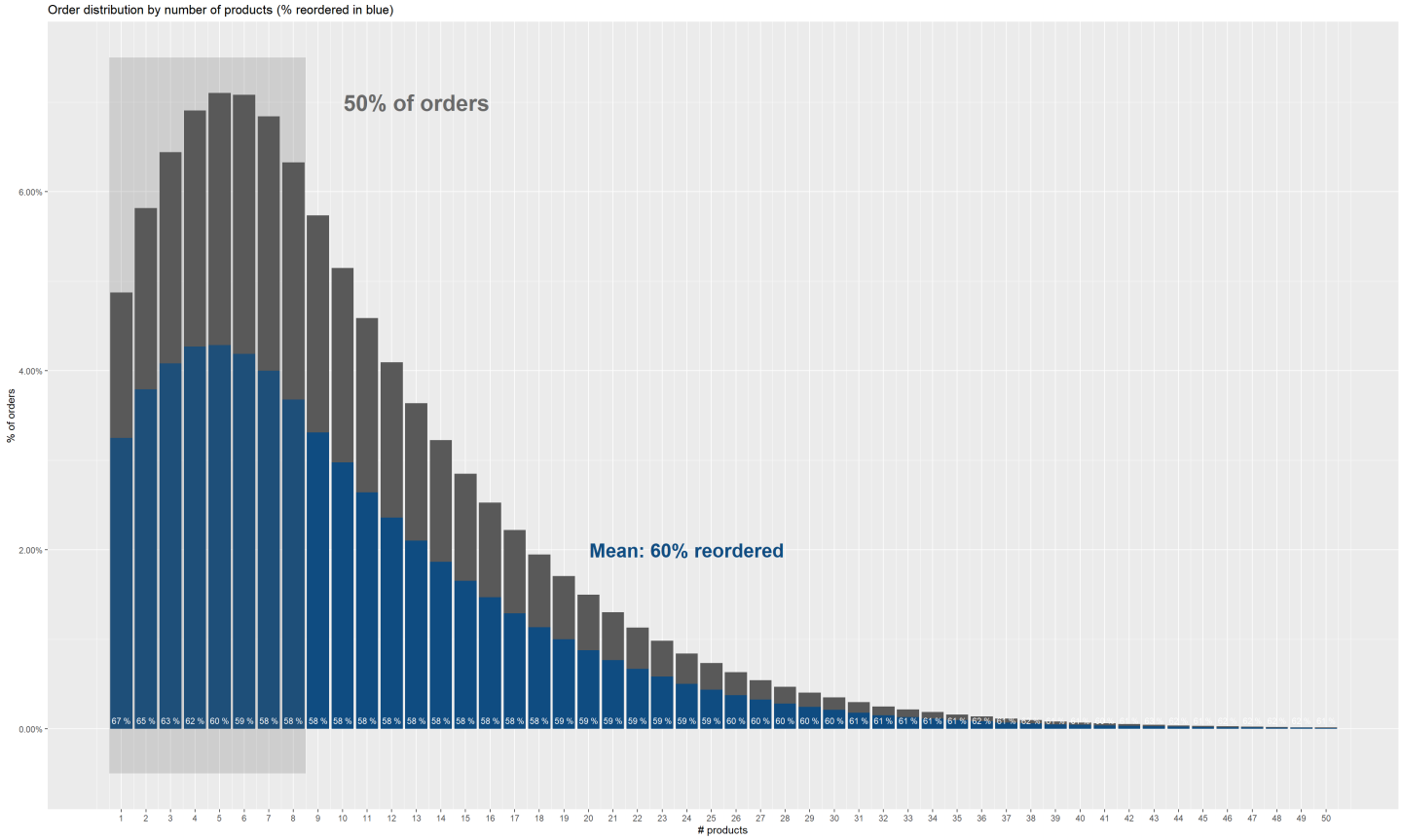

```
# The minimum number of orders a client did was 3 on dataset. Some few clients ordered 100 times.

# Day of week vs hour orders
df %>%
  select(order_id, order_dow, order_hour_of_day) %>%
  distinct() %>%
  group_by(order_dow, order_hour_of_day) %>%
  summarise(contar = n()) %>%
  ggplot() + geom_tile(aes(order_dow, order_hour_of_day, fill=contar)) +
  scale_x_continuous(breaks = seq(0, 6, by = 1)) + scale_y_continuous(breaks = seq(0, 23, by = 1)) +
  labs(title="Day of week vs hour orders", x="Day of week", y="Hour")
```




```
# We can see that most clients order between 13h and 15h, on saturday and between 9h and 10h
# on saunday.

# Products by add order
df %>%
  group_by(add_to_cart_order, product_name) %>%
  summarize(contar = n()) %>%
  mutate(perc = contar / sum(contar)) %>%
  arrange(add_to_cart_order,desc(contar)) %>%
  filter(add_to_cart_order <= 5) %>%
  top_n(5) %>%
  ggplot + geom_bar(aes(x=reorder(product_name,-perc), y=perc, fill=product_name), stat="identity") +
    facet_grid(rows=vars(add_to_cart_order)) +
    labs(title="TOP 5 - Products by add order", x="Products", y="% of orders") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none") +
    scale_y_continuous(labels = percent) +
    scale_fill_manual(values=c("#ffef99", "#ffeb7f", "#8dc63f", "#065535", "#ff5f5f", "white"))
```

```
## Selecting by perc
```


```
# Order distribution by number of products
df %>%
  group_by(order_id) %>%
  summarize(PorcReordered = sum(reordered) / n(), numProdutosNaOrdem = n()) %>%
  group_by(numProdutosNaOrdem) %>%
  summarise(qtOrdens = n(), PorcReorderedFull = mean(PorcReordered)) %>%
  mutate(perc = qtOrdens / sum(qtOrdens)) %>%
  mutate(PorcReordered = PorcReorderedFull * perc) %>%
  filter(numProdutosNaOrdem <= 50) %>%
  ggplot + geom_bar(aes(x=numProdutosNaOrdem, y=perc), stat="identity") +
  geom_bar(aes(x=numProdutosNaOrdem, y=PorcReordered), stat="identity", fill = "#0f4c81") +
  labs(title="Order distribution by number of products (% reordered in blue)", x="# products", y="% of orders") +
  scale_y_continuous(labels = percent) + scale_x_continuous(breaks = seq(1, 50, by = 1)) +
  annotate("rect", xmin = 0.5, xmax = 8.5, ymin = -0.005, ymax = .075, alpha= .2) +
  annotate("text", x = 13, y = 0.07, label = "50% of orders", fontface = 2, size = 8, color="#696969") +
  annotate("text", x = 24, y = 0.02, label = "Mean: 60% reordered", fontface = 2, size = 7, color="#0f4c81") +
  geom_text(aes(label = paste(format(PorcReorderedFull*100, digits=0),"%"), x=numProdutosNaOrdem, y=0), vjust=-0.8, color="white", size=3)
```

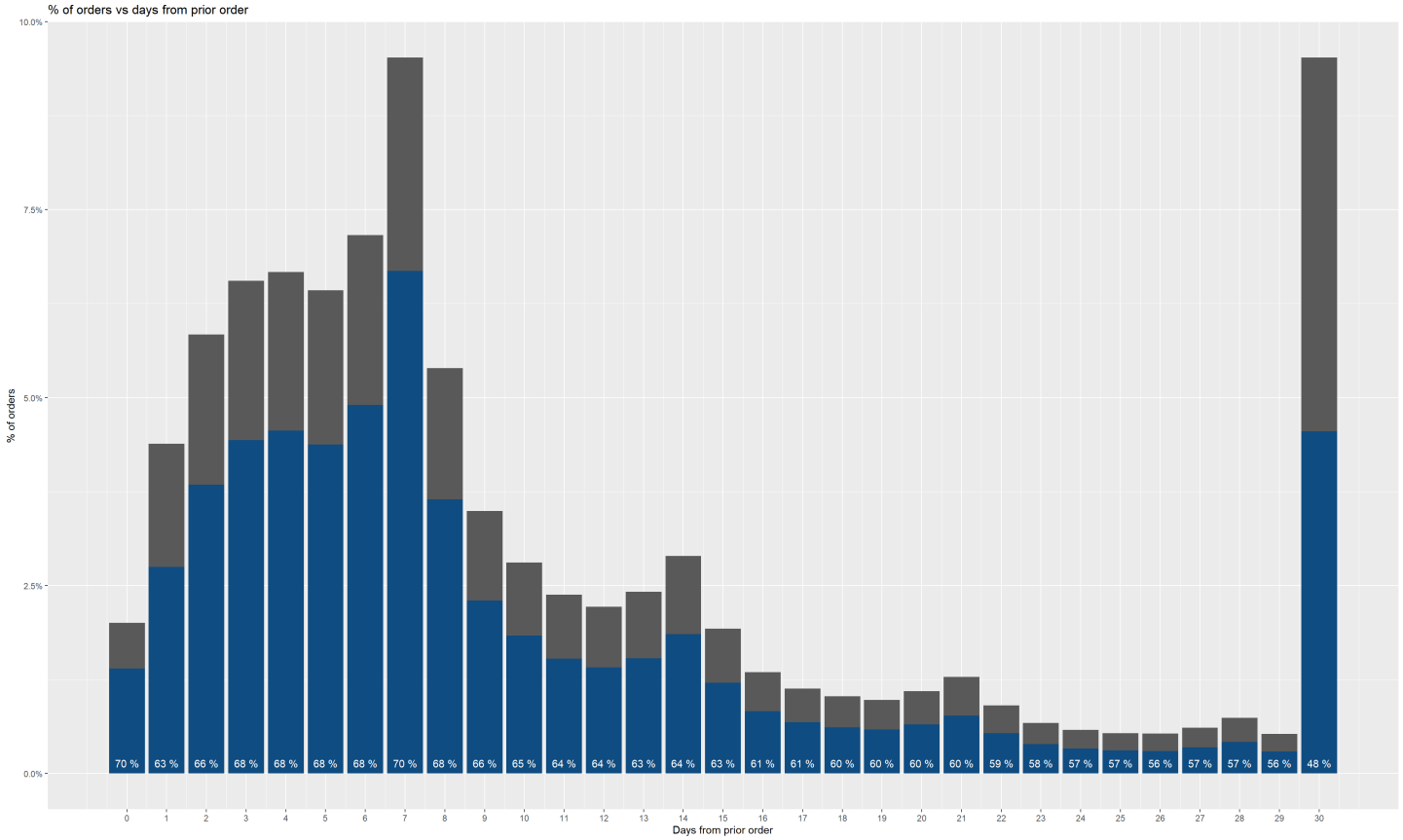



```
# Orders vs prior order in days
df %>%
  group_by(order_id, days_since_prior_order) %>%
  summarise(PorcReordered = sum(reordered) / n()) %>%
  group_by(days_since_prior_order) %>%
  summarise(qtOrdens = n(), PorcReorderedFull = mean(PorcReordered)) %>%
  mutate(perc = qtOrdens / sum(qtOrdens)) %>%
  mutate(PorcReordered = PorcReorderedFull * perc) %>%
  ggplot + geom_bar(aes(x=days_since_prior_order, y=perc), stat="identity") +
  geom_bar(aes(x=days_since_prior_order, y=PorcReordered), stat="identity", fill = "#0f4c81") +
  geom_text(aes(label = paste(format(PorcReorderedFull*100, digits=0),"%"), x=days_since_prior_order, y=0), vjust=-0.8, color="white") +
  labs(title="% of orders vs days from prior order", x="Days from prior order", y="% of orders") +
  scale_y_continuous(labels = percent) + scale_x_continuous(breaks = seq(0, 30, by = 1))
```

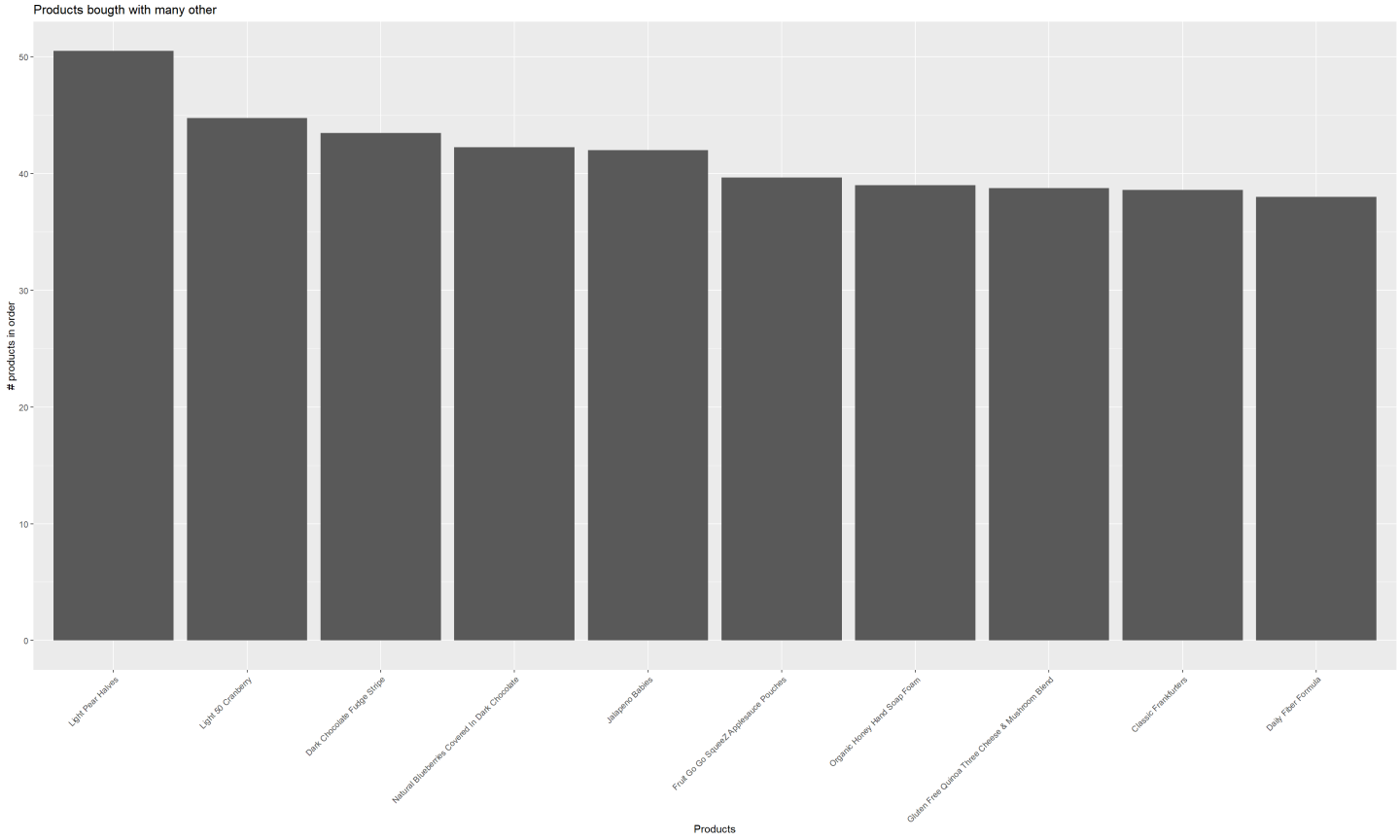
```
## Warning: Removed 1 rows containing missing values (position_stack).
```

```
## Warning: Removed 1 rows containing missing values (position_stack).
```

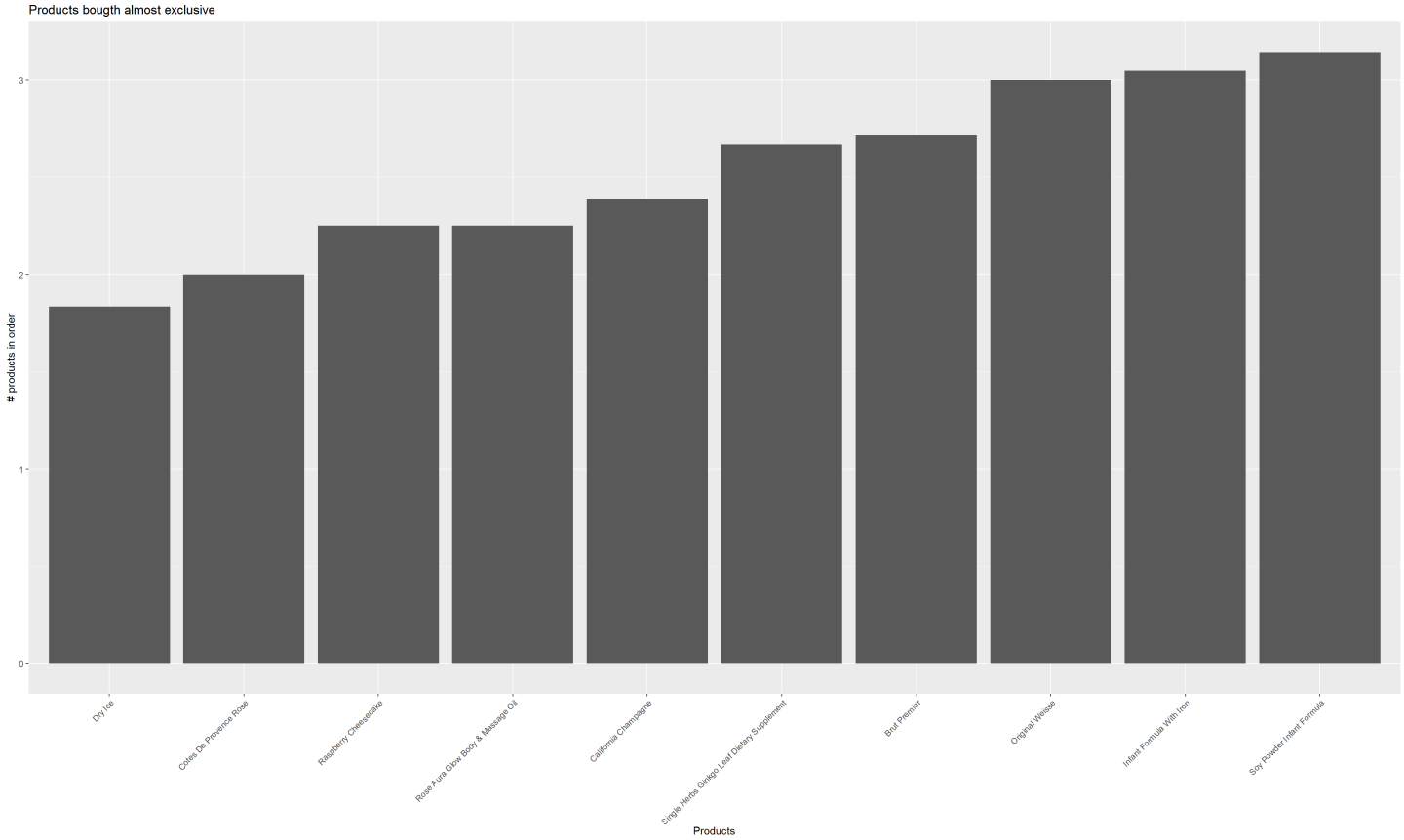
```
## Warning: Removed 1 rows containing missing values (geom_text).
```



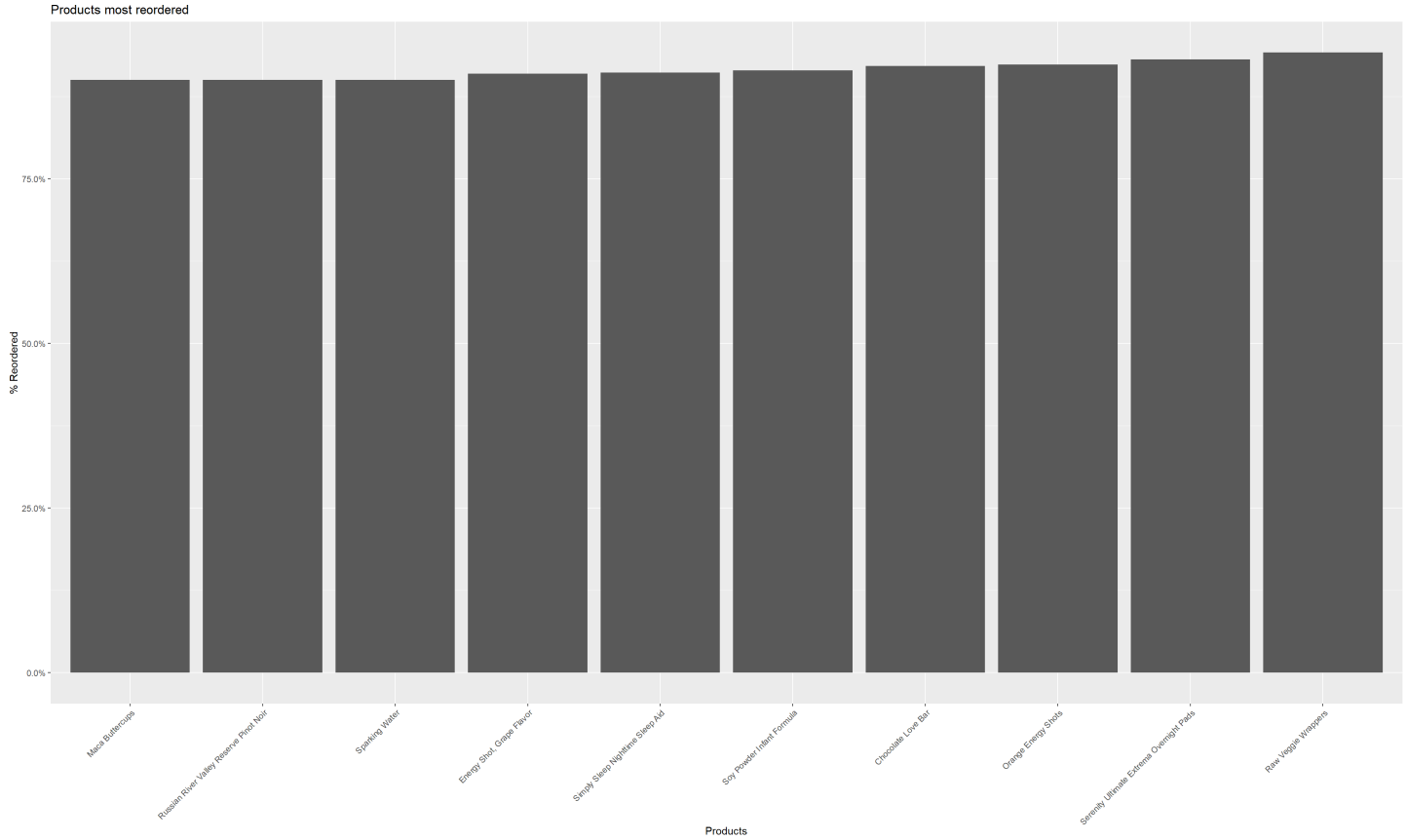

```
# We have to peaks. The first weekkly (7 days) and the other monthly (30 days).

# Products bougth with many other
df %>%
  group_by(order_id) %>%
  summarize(numProdutosNaOrdem = n()) %>%
  left_join(df,by='order_id') %>%
  group_by(product_name) %>%
  summarise(numProdutosNaOrdemMedio = mean(numProdutosNaOrdem), numProductsSold = n()) %>%
  filter(numProductsSold > 2) %>%
  top_n(10, numProdutosNaOrdemMedio) %>%
  ggplot + geom_bar(aes(x=reorder(product_name,-numProdutosNaOrdemMedio), y=numProdutosNaOrdemMedio), stat="identity") +
    labs(title="Products bougth with many other", x="Products", y="# products in order") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) + scale_y_continuous(labels = comma)
```




```
# Products bougth almost exclusive
df %>%
  group_by(order_id) %>%
  summarize(numProdutosNaOrdem = n()) %>%
  left_join(df,by='order_id') %>%
  group_by(product_name) %>%
  summarise(numProdutosNaOrdemMedio = mean(numProdutosNaOrdem), numProductsSold = n()) %>%
  filter(numProductsSold > 2) %>%
  top_n(-10, numProdutosNaOrdemMedio) %>%
  ggplot + geom_bar(aes(x=reorder(product_name,numProdutosNaOrdemMedio), y=numProdutosNaOrdemMedio), stat="identity") +
  labs(title="Products bougth almost exclusive", x="Products", y="# products in order") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) + scale_y_continuous(labels = comma)
```




```
# Products most reordered
df %>%
  group_by(product_name) %>%
  summarise(numReordered = sum(reordered), numProductsSold = n()) %>%
  filter(numProductsSold > 2) %>%
  mutate(PorcReordered = numReordered / numProductsSold) %>%
  top_n(10, PorcReordered) %>%
  ggplot + geom_bar(aes(x=reorder(product_name,PorcReordered), y=PorcReordered), stat="identity") +
  labs(title="Products most reordered", x="Products", y="% Reordered") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) + scale_y_continuous(labels = percent)
```




```
# Reordered - quantile and men
dfTemp <- df %>%
  group_by(product_name) %>%
  summarise(numReordered = sum(reordered), numProductsSold = n()) %>%
  filter(numProductsSold > 2) %>%
  mutate(PorcReordered = numReordered / numProductsSold)

quantile(dfTemp$PorcReordered)
```

```
##        0%       25%       50%       75%      100%
## 0.0000000 0.2142857 0.3808166 0.5303560 0.9411765
```

```
print(paste("Mean: ",mean(dfTemp$PorcReordered)))
```

```
## [1] "Mean:  0.369625043133485"
```

```
rm(dfTemp)

# Market Basket Analysis

# Generate a csv dataset
#dfSparse <- df %>%
#  group_by(order_id) %>%
#  select(order_id, product_name) %>%
#  mutate(product_name2 = paste0(product_name, collapse = ",")) %>%
#  select(order_id,product_name2) %>%
#  distinct()
#dfSparse$order_id <- NULL
#write.csv(dfSparse,"market_basket_transactions.csv", quote = FALSE, row.names = FALSE)
#rm(dfSparse)

# Import as transaction
tr <- read.transactions('market_basket_transactions.csv', format = 'basket', sep=',')
```

```
## Warning in asMethod(object): removing duplicated items in transactions
```

```
summary(tr)
```

```
## transactions as itemMatrix in sparse format with
##  3214875 rows (elements/itemsets/transactions) and
##  281851 columns (items) and a density of 3.520976e-05
##
## most frequent items:
##                 Banana Bag of Organic Bananas   Organic Strawberries   Organic Baby Spinach   Organic Hass Avocado               (Other)
##                 460485                 367637                 252559                 232235                 205061              30386140
##
## element (itemset/transaction) length distribution:
## sizes
##      1      2      3      4      5      6      7      8      9     10     11     12     13     14     15     16     17     18     19     20     2
1     22     23     24     25     26     27     28     29     30     31     32     33     34     35     36     37     38     39     40     41     42
43     44     45     46     47     48     49     50     51     52     53     54     55     56     57     58     59     60     61     62     63     6
4     65     66     67     68     69     70     71     72     73     74     75     76     77     78     79     80     81     82     83     84     85
86     87     88     89     90     91     92     93     94     95     96     97     98     99    100    101    102    105    110    111    112    11
3    114    119    139    150
## 173997 198879 213972 224853 227409 225230 216030 200441 182058 163733 145878 129947 115507 101932  89900  79491  69553  61292  53686  46683  4078
9  35385  30654  26430  22919  19838  17001  14613  12638  10863   9192   7861   6780   5783   5026   4267   3586   3090   2604   2184   1828   1556
1371   1141   1014    898    729    645    513    459    376    336    242    226    205    176    166    116    109     84     83     78     69
54     51     44     37     30     33     27     14     13     26     16     17     11     11     11      8      6      9      2      3      4
5      2      2      4      5      1      2      1      1      2      3      1      1      4      2      1      1      4      2      1      2      1
1      1      2      1      1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   4.000   8.000   9.924  13.000 150.000
##
## includes extended item information - examples:
##                        labels
## 1                          #2
## 2           #2 Coffee Filters
## 3 #2 Cone White Coffee Filters
```

```
# Calculate association rules based on min support and confidence
regrasAssociacao <- apriori(tr, parameter = list(supp=0.001, conf=0.8,maxlen=10))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
##         0.8    0.1    1 none FALSE            TRUE       5   0.001      1     10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 3214
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[281851 item(s), 3214875 transaction(s)] done [20.45s].
## sorting and recoding items ... [1750 item(s)] done [0.45s].
## creating transaction tree ... done [4.07s].
## checking subsets of size 1 2 3 4 done [1.23s].
## writing ... [231 rule(s)] done [0.00s].
## creating S4 object  ... done [0.69s].
```

```
# 231 rules created. Let's take a look on the top 10.
inspect(regrasAssociacao[1:10])
```

```
##      lhs                                              rhs                           support     confidence lift     count
## [1]  {Medium Pulp}                                 => {Country Stand Juice}         0.001146856 1.0000000  871.9487 3687
## [2]  {Country Stand Juice}                         => {Medium Pulp}                 0.001146856 1.0000000  871.9487 3687
## [3]  {One-Ply}                                     => {Paper Towels Choose-A-Sheet} 0.001074692 1.0000000  930.4993 3455
## [4]  {Paper Towels Choose-A-Sheet}                 => {One-Ply}                     0.001074692 1.0000000  930.4993 3455
## [5]  {One-Ply}                                     => {Mega Rolls}                  0.001074692 1.0000000  912.7981 3455
## [6]  {Mega Rolls}                                  => {One-Ply}                     0.001074692 0.9809767  912.7981 3455
## [7]  {Paper Towels Choose-A-Sheet}                 => {Mega Rolls}                  0.001074692 1.0000000  912.7981 3455
## [8]  {Mega Rolls}                                  => {Paper Towels Choose-A-Sheet} 0.001074692 0.9809767  912.7981 3455
## [9]  {2 Huge Rolls = 5 Regular Rolls  Towels/Napkins} => {Select-A-Size Paper Towels}  0.001219954 1.0000000  605.4379 3922
## [10] {2 Huge Rolls = 5 Regular Rolls  Towels/Napkins} => {White}                       0.001219954 1.0000000  474.2403 3922
```

```
# Remove redundant rules
regrasSubset <- which(colSums(is.subset(regrasAssociacao, regrasAssociacao)) > 1) # get subset rules in vector
length(regrasSubset)
```

```
## [1] 178
```

```
regrasAssociacaoSubSet <- regrasAssociacao[-regrasSubset]

# We removed 178 rules, remaning 53. Avarage support of 0.0011 and confidence from 0.8 to 1
inspect(regrasAssociacaoSubSet)
```

```
##      lhs                                          rhs                                  support     confidence lift      count
## [1]  {2 Huge Rolls = 5 Regular Rolls  Towels/Napkins} => {Select-A-Size Paper Towels}          0.001219954 1.0000000  605.43785  3922
## [2]  {2 Huge Rolls = 5 Regular Rolls  Towels/Napkins} => {White}                              0.001219954 1.0000000  474.24030  3922
## [3]  {98% Fat Free}                                 => {Gluten Free}                         0.001034566 0.9904705  248.16763  3326
## [4]  {Chocolate Chip Walnut}                        => {Cookies}                             0.001086512 1.0000000  679.82132  3493
## [5]  {Organic Snack Mix Bunnies Snack Mix}          => {Organic}                             0.001049185 1.0000000   46.87773  3373
## [6]  {Party Size}                                   => {Simply Naked}                        0.001107975 0.9024576  537.47467  3562
## [7]  {Party Size}                                   => {Pita Chips}                          0.001107975 0.9024576  520.50381  3562
## [8]  {Ginger Root Beer}                             => {Naturally Flavored Zero Calorie Soda} 0.001273144 1.0000000  596.45176  4093
## [9]  {Ginger Root Beer}                             => {Caffeine Free}                       0.001273144 1.0000000  496.65920  4093
## [10] {Roja}                                         => {Hot}                                 0.001120728 1.0000000  674.11931  3603
## [11] {Roja}                                         => {Salsa}                               0.001120728 1.0000000  577.59163  3603
## [12] {Americano}                                    => {Prosciutto}                          0.001294918 1.0000000  356.57442  4163
## [13] {Fat Free}                                     => {Milk}                                0.001211867 0.8171141   63.41540  3896
## [14] {Select-A-Size Paper Towels}                   => {White}                               0.001581088 0.9572505  453.96675  5083
## [15] {No Salt Added}                                => {Organic Tomato Paste}                0.001194448 0.9149392  141.90541  3840
## [16] {Kalamata}                                     => {Olives}                              0.001168319 1.0000000  652.63398  3756
## [17] {Kalamata}                                     => {Organic}                             0.001164586 0.9968051   46.72796  3744
## [18] {97% Fat Free}                                 => {Gluten Free}                         0.001212489 0.9723123  243.61800  3898
## [19] {Citrus}                                       => {Organic Raw}                         0.001331311 0.9891380  577.85843  4280
## [20] {Citrus}                                       => {Kombucha}                            0.001331311 0.9891380  546.29015  4280
## [21] {Jalapeno Lime}                                => {Tortilla Chips}                      0.001307982 1.0000000  275.29329  4205
## [22] {Almondmilk Creamer}                           => {Vanilla}                             0.001609705 1.0000000  354.52966  5175
## [23] {Pitted}                                       => {Olives}                              0.001164898 0.9067797  591.79522  3745
## [24] {Pitted}                                       => {Organic}                             0.001251371 0.9740920   45.66323  4023
## [25] {Brown Rice}                                   => {Tortillas}                           0.001299273 0.8217588  200.32240  4177
## [26] {and Pear Baby Food}                           => {Mango}                               0.001149034 1.0000000  524.36389  3694
## [27] {Happy Baby Spinach}                           => {Mango}                               0.001149034 1.0000000  524.36389  3694
## [28] {Clasico}                                      => {Tortilla Chips}                      0.001531319 1.0000000  275.29329  4923
## [29] {Country Buttermilk}                           => {Bread}                               0.001862281 1.0000000  173.77703  5987
## [30] {Fruit Spread}                                 => {Strawberry}                          0.001347175 0.8537355   78.05292  4331
## [31] {Whole Peeled}                                 => {Tomatoes}                            0.001851394 1.0000000  418.93081  5952
## [32] {Apricot & Banana Stage 2 Baby Food}           => {Peach}                               0.001478440 1.0000000  263.06153  4753
## [33] {Hope}                                         => {Hummus}                              0.001728528 1.0000000  405.20229  5557
## [34] {Original Recipe}                              => {Hummus}                              0.001728528 0.9940966  402.81022  5557
## [35] {Reduced Fat}                                  => {2% Milkfat}                          0.002277227 0.9824208  156.38543  7321
## [36] {Reduced Fat}                                  => {Milk}                                0.002277849 0.9826892   76.26552  7323
## [37] {Crispy Wheat}                                 => {Crackers}                            0.002190754 1.0000000  190.54499  7043
## [38] {Italian (Flat)}                               => {Parsley}                             0.002594502 1.0000000  223.03837  8341
## [39] {New England Grown}                            => {Parsley}                             0.002595746 0.9449666  210.76380  8345
## [40] {Organic Milk Reduced Fat}                     => {2% Milkfat}                          0.003844940 1.0000000  159.18375 12361
## [41] {Lowfat}                                       => {Yogurt}                              0.003067615 0.9642159  128.48518  9862
## [42] {Lowfat}                                       => {Strawberry}                          0.002608500 0.8199061   74.96007  8386
## [43] {Organic Butterhead (Boston)}                  => {Butter}                              0.003595785 1.0000000  165.11093 11560
## [44] {Strained Low-Fat}                             => {Yogurt}                              0.003604495 1.0000000  133.25354 11588
## [45] {Bibb) Lettuce}                                => {Butter}                              0.003736382 1.0000000  165.11093 12012
## [46] {Flat Parsley}                                 => {Bunch}                               0.004006377 1.0000000   74.04816 12880
## [47] {Coconut}                                      => {Yogurt}                              0.003605117 0.9501558  126.61162 11590
## [48] {Super Spinach! Baby Spinach}                  => {Baby Bok Choy}                       0.004280104 1.0000000  186.17530 13760
## [49] {Sweet Baby Kale}                              => {Baby Bok Choy}                       0.004280104 1.0000000  186.17530 13760
## [50] {YoKids Squeezers Organic Low-Fat Yogurt}      => {Strawberry}                          0.005730549 1.0000000   91.42518 18423
## [51] {Vitamin D}                                    => {Milk}                                0.008704226 0.9758334   75.73345 27983
## [52] {Organic Red Radish}                           => {Bunch}                               0.008047902 1.0000000   74.04816 25873
## [53] {Bag}                                          => {Clementines}                         0.011584276 0.8654892   41.38749 37242
```

```
# Transforme in dataframe to posterior use
dfAssociationRules = DATAFRAME(regrasAssociacaoSubSet)
dfAssociationRules$LHS <- str_sub(dfAssociationRules$LHS,2,str_length(dfAssociationRules$LHS)-1)
dfAssociationRules$RHS <- str_sub(dfAssociationRules$RHS,2,str_length(dfAssociationRules$RHS)-1)
dfAssociationRules <- dfAssociationRules %>% mutate(Rule1=0, Rule2=0)
dfAssociationRules <- dfAssociationRules %>% group_by(LHS) %>% summarise(n=n()) %>% left_join(dfAssociationRules)
```

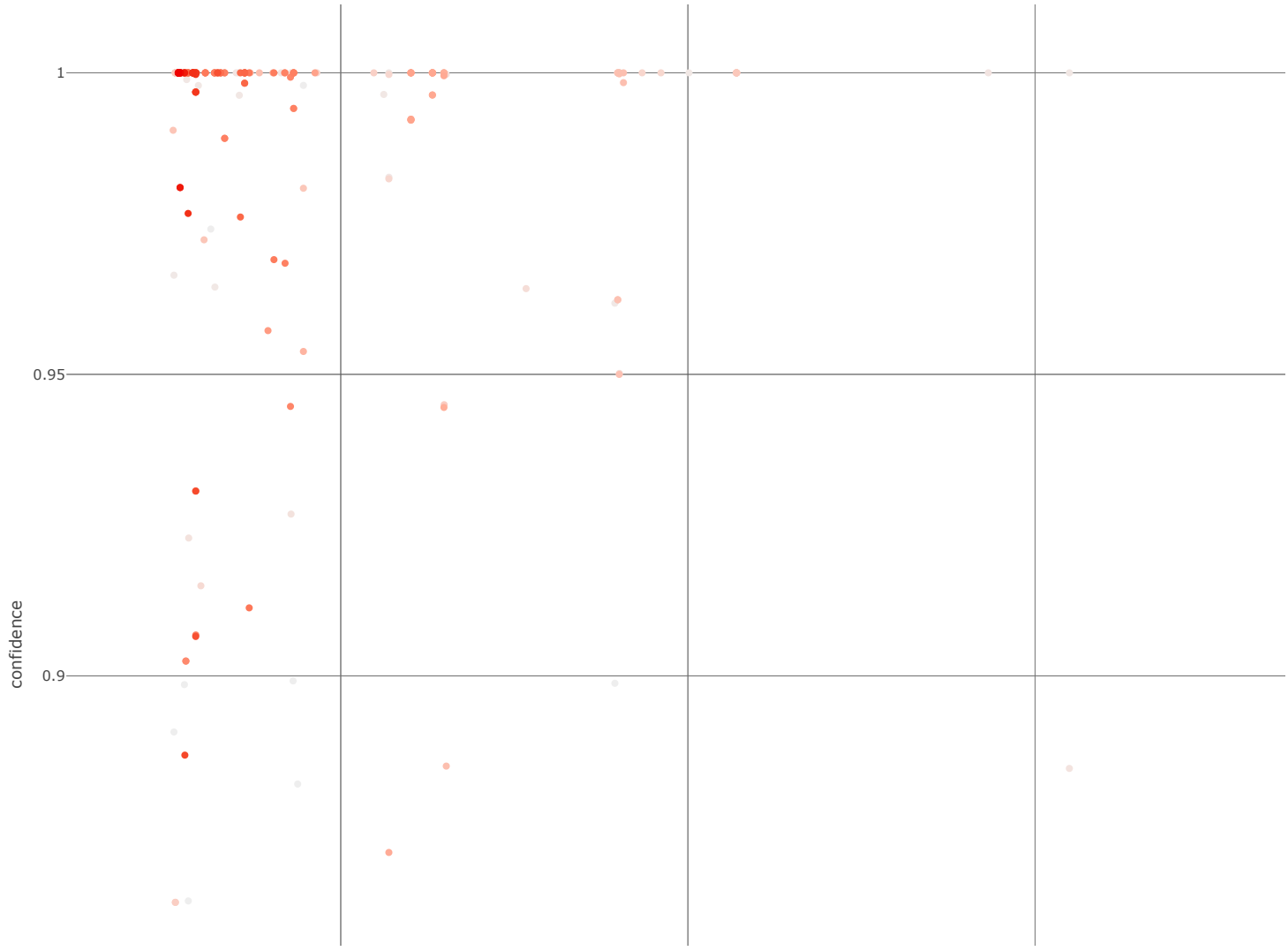
```
## Joining, by = "LHS"
```

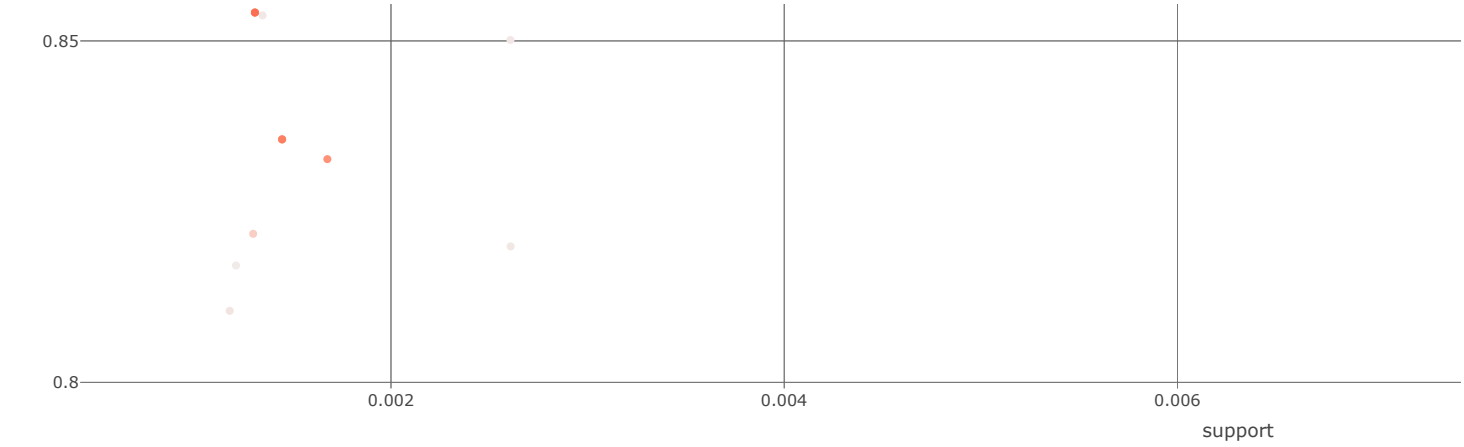
```
item = 1
while (item <= length(dfAssociationRules$LHS)){
  dfAssociationRules[item,'Rule1']=dfAssociationRules[item,'RHS']
  if (dfAssociationRules[item,2] == 2){
    dfAssociationRules[item,'Rule2']=dfAssociationRules[item+1,'RHS']
    dfAssociationRules[item + 1,'Rule1']=dfAssociationRules[item,'RHS']
    dfAssociationRules[item + 1,'Rule2']=dfAssociationRules[item+1,'RHS']
    item = item + 1
  }
  item = item + 1
}

dfTemp <- dfAssociationRules %>% select(RHS, Rule1, Rule2) %>% distinct()
dfTemp <- dfTemp %>% group_by(RHS) %>% summarise(n=n()) %>% left_join(dfTemp)
```

```
## Joining, by = "RHS"
```

```
dfTemp <- dfTemp[!(dfTemp$n==2 & dfTemp$Rule2==0),]
names(dfTemp)[1] <- "product_name"
dfTemp$n <- NULL

dfAssociationRulesFinal <- dfAssociationRules %>% select(LHS, Rule1, Rule2) %>% distinct()
names(dfAssociationRulesFinal)[1] <- "product_name"
dfAssociationRulesFinal <- bind_rows(dfAssociationRulesFinal,dfTemp)

# Data Viz
subRegras<-regrasAssociacao[quality(regrasAssociacao)$confidence>0.4]
# Plot SubRules
# The majority rules are located up left and  support is smaller than 0.005 bps.
plot(subRegras, engine = "htmlwidget")
```

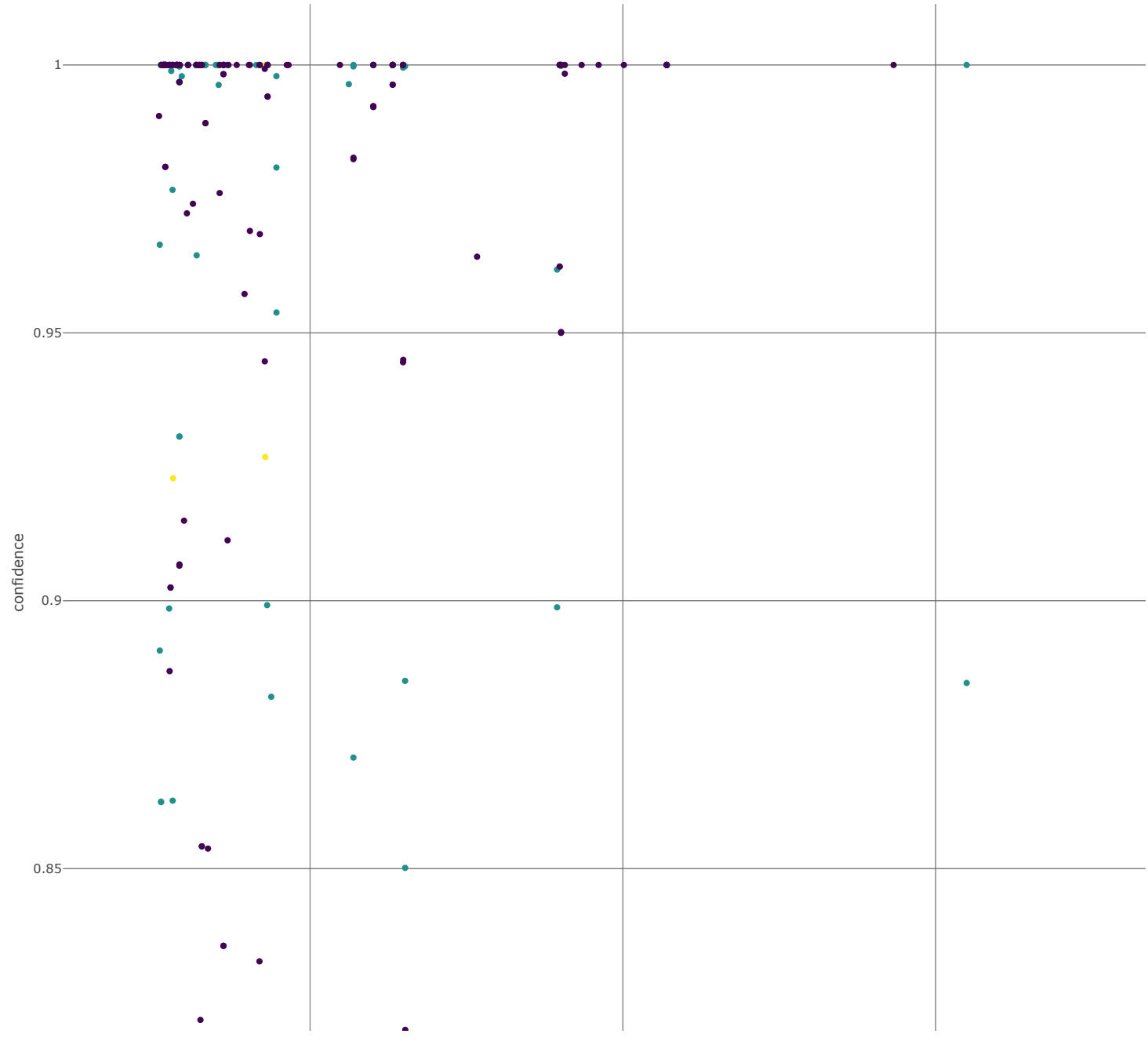
```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```

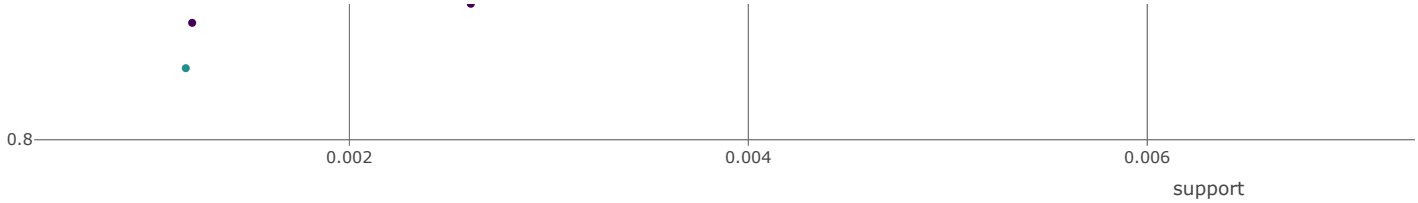





```
# The color represents the number of itens in the rule
plot(subRegras,method="two-key plot", engine = "htmlwidget")
```

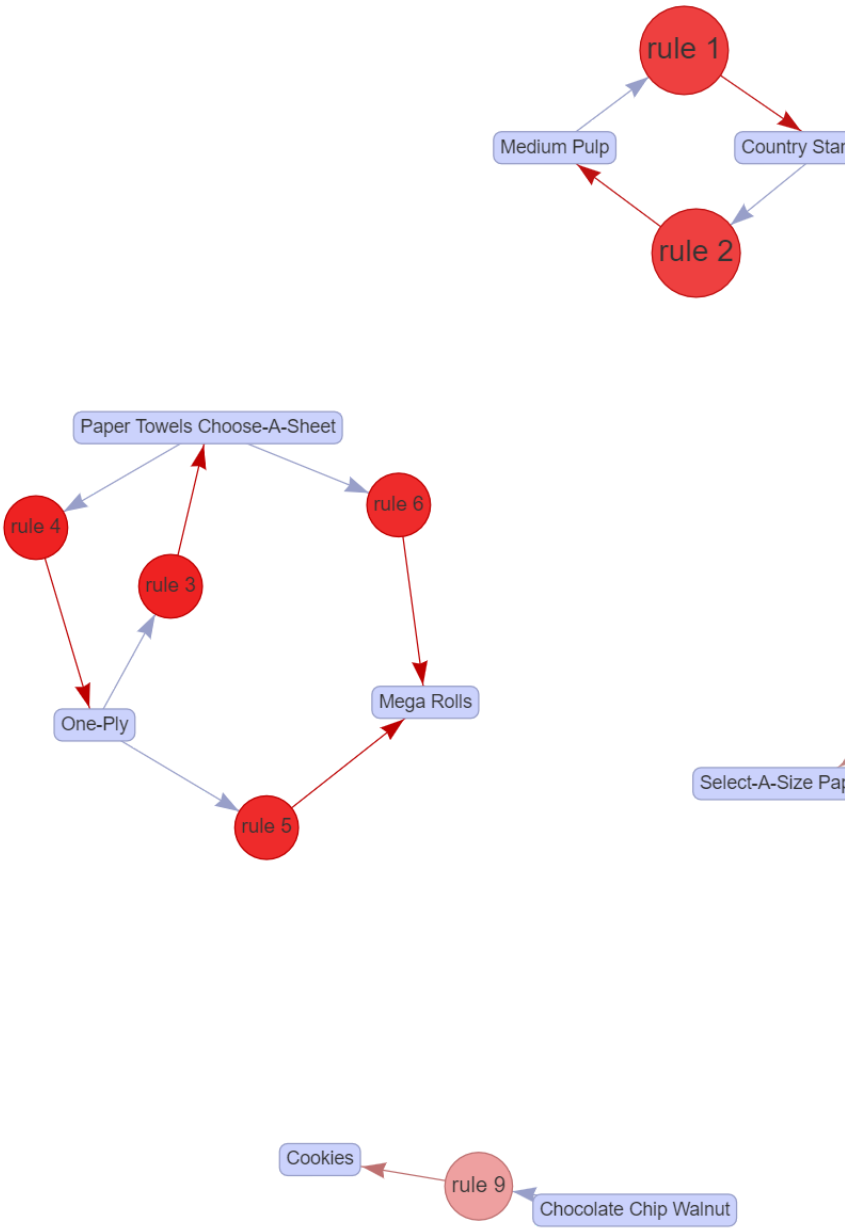
```
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



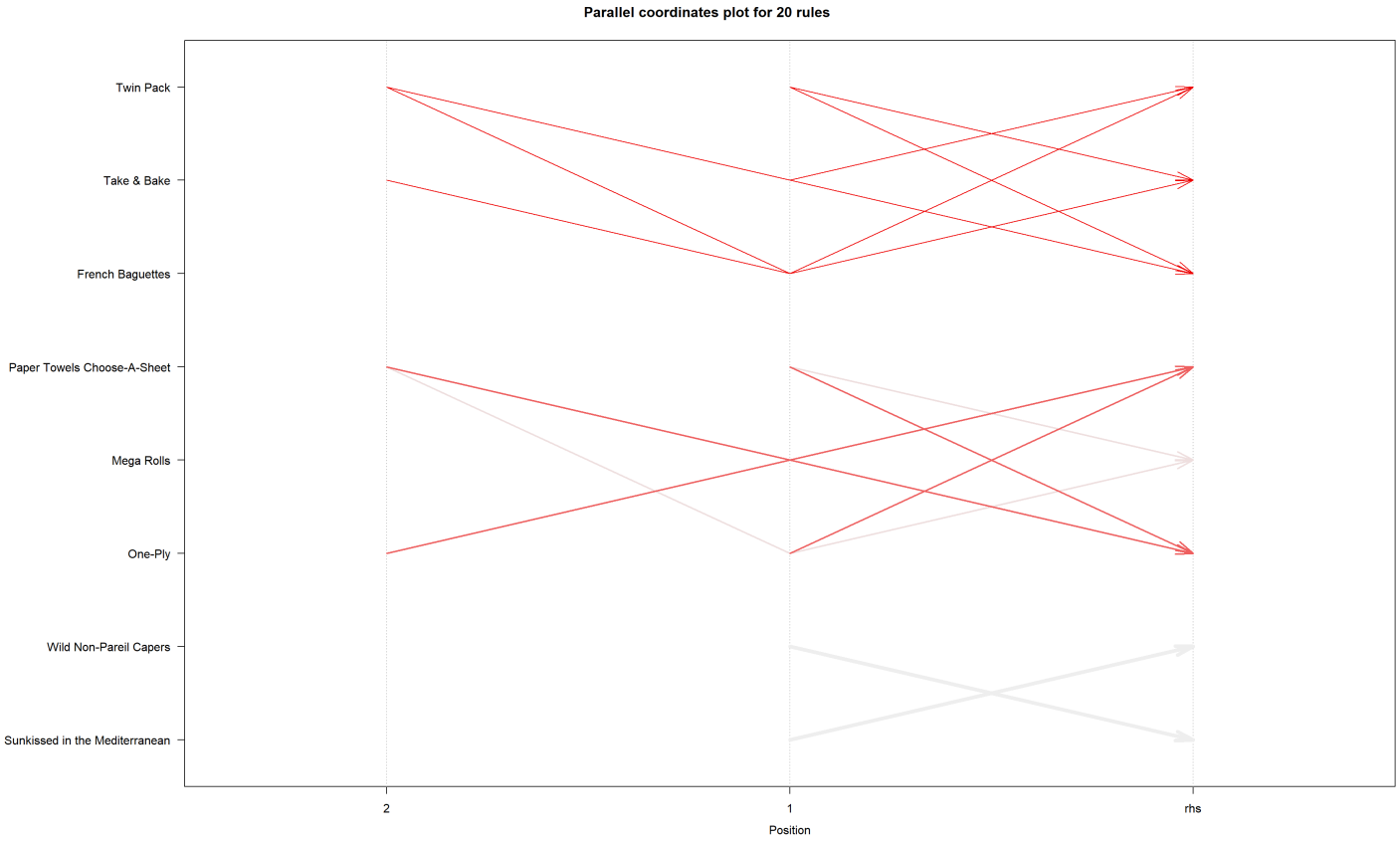



```
# This graph shows the rules as a chain. It is possible to see the entire rule.
top10subRules <- head(subRegras, n = 10, by = "confidence")
plot(top10subRules, method = "graph",  engine = "htmlwidget")
```

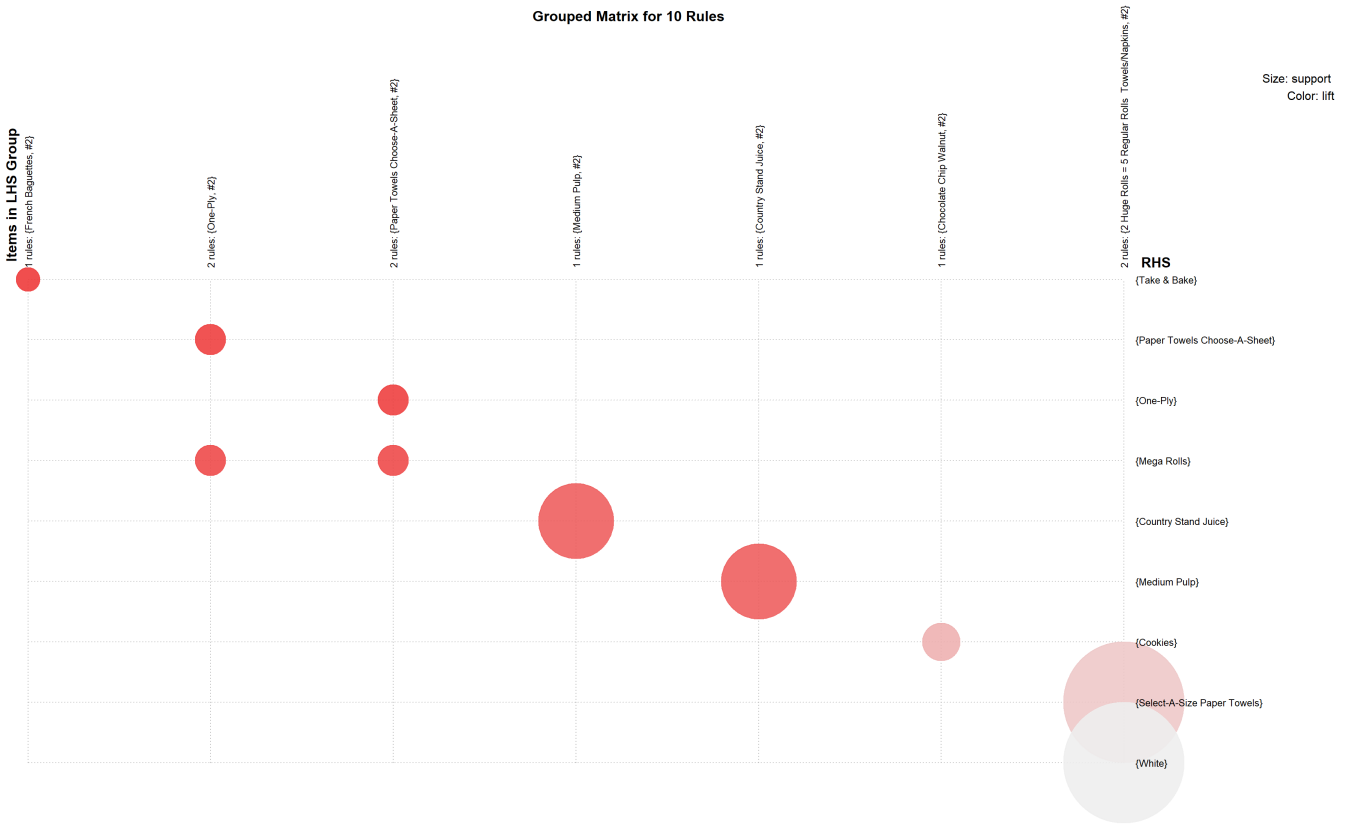
Select by id



```
# This graph shows as buying a product follows to another.
# 2 - The most recent addition
# 1 - The item I had
subRegras2<-head(subRegras, n=20, by="lift")
plot(subRegras2, method="paracoord")
```




```
# Some rules such as Hue Rolls -> Paper Towels have great support and lift
plot(top10subRules, method = "grouped")
```




```
rm(tr, regrasAssociacao, regrasAssociacaoSubSet, subRegras, subRegras2, top10subRules, regrasSubset)

# Model-------------------------------------------------------------------------------------

# Creating indicators for products
# % Reordered
dfTemp <- df %>% group_by(product_id) %>% summarise(numReorderedMeanPrd = sum(reordered)/n())
dfProducts <- left_join(dfProducts, dfTemp)
```

```
## Joining, by = "product_id"
```

```
# Number products in orders the product is included mean/std
dfTemp <- df %>%
  group_by(order_id) %>%
  summarize(numPrdOrdem = n()) %>%
  left_join(df) %>%
  group_by(product_name) %>%
  summarise(numPrdOrderMeanPrd = mean(numPrdOrdem),
            numPrdOrderStdPrd = sd(numPrdOrdem))
```

```
## Joining, by = "order_id"
```

```
dfProducts <- left_join(dfProducts, dfTemp)
```

```
## Joining, by = "product_name"
```

```
# Day of week, hour, add to cart, order number mean/std
dfTemp <- df %>%
  group_by(product_id) %>%
  summarize(
    dayOfWeekMeanPrd = mean(order_dow),
    dayOfWeekStdPrd = sd(order_dow),
    hourDayMeanPrd = mean(order_hour_of_day),
    hourDayStdPrd = sd(order_hour_of_day),
    addCartMeanPrd = mean(add_to_cart_order),
    addCartStdPrd = sd(add_to_cart_order),
    orderNumberMeanPrd = mean(order_number),
    orderNumberStdPrd = sd(order_number))
dfProducts <- left_join(dfProducts, dfTemp)
```

```
## Joining, by = "product_id"
```

```
# Days since prior order
dfTemp <- df %>% filter(is.na(days_since_prior_order) == FALSE) %>%
  group_by(product_id) %>%
  summarize(
    daysPriorOrderMeanPrd = mean(days_since_prior_order),
    daysPriorOrderStdPrd = sd(days_since_prior_order))
dfProducts <- left_join(dfProducts, dfTemp)
```

```
## Joining, by = "product_id"
```

```
# How many times de product was sold
dfTemp <- df %>% group_by(product_id) %>% summarise(NumSoldPrd = n())
dfProducts <- left_join(dfProducts, dfTemp)
```

```
## Joining, by = "product_id"
```

```
# How many users bought the product
dfTemp <- df %>% group_by(product_id, user_id) %>% summarise(n = n()) %>%
                 group_by(product_id) %>% summarise(NumUserBougthPrd=n())
dfProducts <- left_join(dfProducts, dfTemp)
```

```
## Joining, by = "product_id"
```

```
# Creating indicators for users
# % Reordered mean/std
dfUsers <- df %>% group_by(user_id, order_id) %>% summarise(n = sum(reordered)/n()) %>%
  group_by(user_id) %>% summarise(numReorderedMeanUser = mean(n), numReorderedMStdUser = sd(n))
# Dow of last order
dfUsers <- dfOrderProductsTrain %>% select(user_id, order_dow) %>% distinct() %>% right_join(dfUsers)
```

```
## Joining, by = "user_id"
```

```
names(dfUsers)[2] <- "DowLastOrderUser"
# Hour of last order
dfUsers <- dfOrderProductsTrain %>% select(user_id, order_hour_of_day) %>% distinct() %>% right_join(dfUsers)
```

```
## Joining, by = "user_id"
```

```
names(dfUsers)[2] <- "HourLastOrderUser"
# Days since prior order of last order
dfUsers <- dfOrderProductsTrain %>% select(user_id, days_since_prior_order) %>% distinct() %>% right_join(dfUsers)
```

```
## Joining, by = "user_id"
```

```
names(dfUsers)[2] <- "DaysSincePriorLastOrderUser"
# Number products in orders the user does
dfTemp <- df %>%
  group_by(order_id) %>%
  summarize(numPrdOrdem = n()) %>%
  left_join(df) %>%
  group_by(user_id) %>%
  summarise(numPrdMeanUser = mean(numPrdOrdem), numPrdStdUser = sd(numPrdOrdem))
```

```
## Joining, by = "order_id"
```

```
dfUsers <- left_join(dfUsers, dfTemp)
```

```
## Joining, by = "user_id"
```

```
# Day of week, hour, add to cart, order number avegare/std
dfTemp <- df %>%
  group_by(user_id) %>%
  summarize(
    dayOfWeekMeanUser = mean(order_dow),
    dayOfWeekStdUser = sd(order_dow),
    hourDayMeanUser = mean(order_hour_of_day),
    hourDayStdUser = sd(order_hour_of_day),
    addCartMeanUser = mean(add_to_cart_order),
    addCartStdUser = sd(add_to_cart_order),
    orderNumberMeanUser = mean(order_number),
    orderNumberStdUser = sd(order_number))
dfUsers <- left_join(dfUsers, dfTemp)
```

```
## Joining, by = "user_id"
```

```
# Days since prior order average/std
dfTemp <- df %>% filter(is.na(days_since_prior_order) == FALSE) %>%
  group_by(user_id) %>%
  summarize(
    daysPriorOrderMeanUser = mean(days_since_prior_order),
    daysPriorOrderStdUser = sd(days_since_prior_order))
dfUsers <- left_join(dfUsers, dfTemp)
```

```
## Joining, by = "user_id"
```

```
# How many times the client has ordered
dfTemp <- df %>% group_by(order_id,user_id) %>% summarise(NumPrd = n()) %>% group_by(user_id) %>% summarise(NumOrderedUser=n())
dfUsers <- left_join(dfUsers, dfTemp)
```

```
## Joining, by = "user_id"
```

```
# How many products the user bought
dfTemp <- df %>% group_by(user_id) %>% summarise(NumPrdUser = n())
dfUsers <- left_join(dfUsers, dfTemp)
```

```
## Joining, by = "user_id"
```

```
# Gerando o dataset final
dfFinal <- df %>% select(user_id, product_id) %>% distinct()
dfFinal <- left_join(dfFinal,dfProducts, by='product_id')
dfFinal <- left_join(dfFinal,dfUsers, by='user_id')

# User-product indicators
# Number that user bougth the product
dfTemp <- df %>% group_by(user_id, product_id) %>% summarise(PU_NumUserBoutghPrd = n())
dfFinal <- left_join(dfFinal, dfTemp, by=c('user_id','product_id'))
# Number or orders user did after bougth the product by the last time
dfTemp <- df %>% group_by(user_id, product_id) %>% summarise(order_number = max(order_number))
dfTemp <- df %>% group_by(user_id) %>% summarise(order_numberMAX = max(order_number)) %>%
  left_join(dfTemp, by='user_id') %>% mutate(PU_OrdersSinceLastBougth = order_numberMAX-order_number)
dfTemp$order_number <- NULL
dfTemp$order_numberMAX <- NULL
dfFinal <- left_join(dfFinal, dfTemp, by=c('user_id','product_id'))
# Number or days passed after bougth the product by the last time: days and normalized by user mean
dfTemp2 <- df
dfTemp2$days_since_prior_order <- replace_na(dfTemp2$days_since_prior_order,0)
dfTemp <- dfTemp2 %>% group_by(user_id, product_id) %>% summarise(daysProduct = sum(days_since_prior_order))
dfTemp <- dfTemp2 %>% group_by(user_id, order_id) %>% summarise(n = sum(days_since_prior_order)/n()) %>%
  group_by(user_id) %>% summarise(daysProductMAX = sum(n)) %>%
  left_join(dfTemp, by='user_id') %>% mutate(PU_DaysSinceLastBougth = daysProductMAX-daysProduct)
rm(dfTemp2)
dfTemp$daysProductMAX <- NULL
dfTemp$daysProduct <- NULL

dfTemp <- left_join(dfTemp,dfUsers[,c('user_id','daysPriorOrderMeanUser')], by='user_id')
dfTemp <- dfTemp %>% mutate(PU_DaysSinceLastBougthNorm = PU_DaysSinceLastBougth/daysPriorOrderMeanUser)
dfTemp$daysPriorOrderMeanUser <- NULL
dfFinal <- left_join(dfFinal, dfTemp, by=c('user_id','product_id'))
# Boolean if product was ever reorderd by user
dfTemp <- df %>% group_by(user_id, product_id) %>% summarise(PU_PrdEverReorderedUser = max(reordered))
dfFinal <- left_join(dfFinal, dfTemp, by=c('user_id','product_id'))

# Add Market Basket Analysis for products
dfFinal <- left_join(dfFinal, dfAssociationRulesFinal, by='product_name')
dfFinal$Rule1 <- replace_na(dfFinal$Rule1,0)
dfFinal$Rule2 <- replace_na(dfFinal$Rule2,0)
rm(dfTemp)

# Verify if the product was bougth in last order by user (boolean)
dfUsersPrdTrain <- dfOrderProductsTrain %>% select(user_id, product_id) %>% distinct() %>%
  mutate(boughtLastOrder=1)
dfFinal <- left_join(dfFinal, dfUsersPrdTrain, by=c('user_id', 'product_id'))

# Remove NAs
dfFinal <- dfFinal %>% replace(is.na(.), 0)
# Remove unnecessary columns
dfFinal$product_name <- NULL
dfFinal$aisle <- NULL
dfFinal$department <- NULL

# Normalize dataset
for (item in names(dfFinal[,-c(1,2,42,43,44)])){
  X <- dfFinal[[item]]
  dfFinal[[item]] <- (X - min(X)) / (max(X) - min(X))
}

# Transform to factor
dfFinal$boughtLastOrder <- as.factor(dfFinal$boughtLastOrder)
dfFinal$Rule1 <- as.factor(dfFinal$Rule1)
dfFinal$Rule2 <- as.factor(dfFinal$Rule2)

# Generate train/test users
dfUsersTrain <- dfOrderProductsTrain %>% select(user_id) %>% distinct()
trainIndex <- createDataPartition(dfUsersTrain$user_id, p = .7, list = FALSE, times = 1)
trainSet <- dfUsersTrain[trainIndex]
testSet  <- dfUsersTrain[-trainIndex]

trainSet <- dfFinal %>% filter(user_id %in% trainSet)
testSet <- dfFinal %>% filter(user_id %in% testSet)

# Remove unnecessary columns
trainSet$user_id <- NULL
trainSet$product_id <- NULL
testSet$user_id <- NULL
testSet$product_id <- NULL

rm(df, dfFinal, dfOrderProductsTrain, dfUsers, dfProducts, trainIndex, dfUsersPrdTrain, dfUsersTrain, item, X, dfAssociationRules, dfAssociationRule
sFinal)
```

```
# Train models
#write.csv(trainSet,"trainSet.csv", quote = FALSE, row.names = FALSE)
#write.csv(testSet,"testSet.csv", quote = FALSE, row.names = FALSE)
trainSet <- fread("trainSet.csv")
testSet <- fread("testSet.csv")
trainSet$boughtLastOrder <- as.factor(trainSet$boughtLastOrder)
testSet$boughtLastOrder <- as.factor(testSet$boughtLastOrder)
trainSet$Rule1 <- as.factor(trainSet$Rule1)
trainSet$Rule2 <- as.factor(trainSet$Rule2)
testSet$Rule1 <- as.factor(testSet$Rule1)
testSet$Rule2 <- as.factor(testSet$Rule2)

# As it is an educational project and for computacional reasons, we will reduce the train dataset, so we can
# run models faster. We will take the opportunity to balance the train dataset to 50/50.
dfTemp <- trainSet %>% filter(boughtLastOrder==1) %>% sample_n(15000)
trainSet <- trainSet %>% filter(boughtLastOrder==0) %>% sample_n(15000) %>% bind_rows(dfTemp)
trainSet <- sample_n(trainSet, 30000, replace=FALSE)

dfTemp <- testSet %>% filter(boughtLastOrder==1) %>% sample_n(5000)
testSetB <- testSet %>% filter(boughtLastOrder==0) %>% sample_n(5000) %>% bind_rows(dfTemp)
testSetB <- sample_n(testSetB, 10000, replace=FALSE)

testSet <- sample_n(testSet, 10000, replace=FALSE)

rm(dfTemp)

# Define cross validation
ctrl <- trainControl(method = "cv", number=5)
variaveisModelo <- as.formula(boughtLastOrder ~ .)

library(doParallel)
```

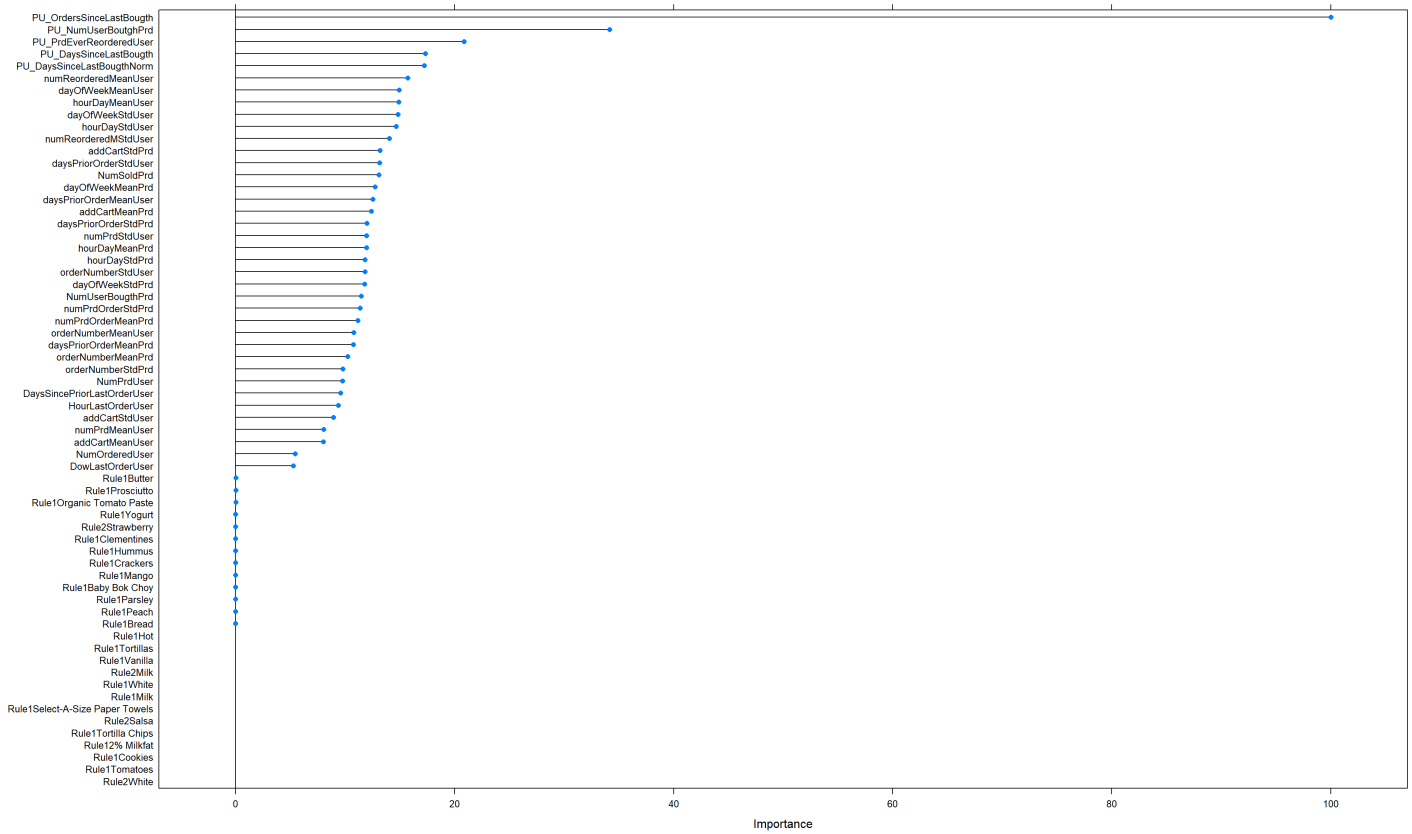
```
## Warning: package 'doParallel' was built under R version 3.6.3
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
cl <- makePSOCKcluster(5)
registerDoParallel(cl)


# Random Forest
modeloRF <- train(variaveisModelo, data=trainSet, method='rf', trControl=ctrl)

# Checking the most important parameters
plot(varImp(modeloRF))
```




```
# We can see that parameters regarind products and users are de most importants, while variables from
# market basket analysis are weak to predict

print(modeloRF)
```

```
## Random Forest
##
## 30000 samples
##    40 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 24000, 24000, 24000, 24000, 24000
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7233667  0.4467333
##   33    0.7427000  0.4854000
##   64    0.7418000  0.4836000
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 33.
```

```
previsoes <- data.frame(observado = testSet$boughtLastOrder,
                        previsto = predict(modeloRF, newdata = testSet))
confusionMatrix(previsoes$observado, previsoes$previsto,mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6702 2319
##          1  262  717
##
##                Accuracy : 0.7419
##                  95% CI : (0.7332, 0.7505)
##     No Information Rate : 0.6964
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2454
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7429
##                  Recall : 0.9624
##                      F1 : 0.8385
##              Prevalence : 0.6964
##          Detection Rate : 0.6702
##    Detection Prevalence : 0.9021
##       Balanced Accuracy : 0.5993
##
##        'Positive' Class : 0
##
```

```
# We got a good accurancy, but only because the test dataset is really umbalanced. The model made wrong
# prediciton for calss 1 (user ordered), making more mistakes than corrected forecasts. Let's use balanced
# test dataset and check accucary.

previsoes <- data.frame(observado = testSetB$boughtLastOrder,
                        previsto = predict(modeloRF, newdata = testSetB))
confusionMatrix(previsoes$observado, previsoes$previsto,mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3676 1324
##          1 1246 3754
##
##                Accuracy : 0.743
##                  95% CI : (0.7343, 0.7515)
##     No Information Rate : 0.5078
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.486
##
##  Mcnemar's Test P-Value : 0.1288
##
##               Precision : 0.7352
##                  Recall : 0.7469
##                      F1 : 0.7410
##              Prevalence : 0.4922
##          Detection Rate : 0.3676
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7431
##
##        'Positive' Class : 0
##
```

```
# Now, the overall accuracy have moved down, but the model was able to get more corrected predictions for
# Class 1. Let's try another model.

# Stochastic Gradient Boosting
modeloSGB <- train(variaveisModelo, data=trainSet, method='gbm', trControl=ctrl, verbose=FALSE)
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 39: Rule12% Milkfat has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 44: Rule1Cookies has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 46: Rule1Hot has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 49: Rule1Milk has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 54: Rule1Select-A-Size Paper Towels has no variati
on.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 55: Rule1Tomatoes has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 56: Rule1Tortilla Chips has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 57: Rule1Tortillas has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 58: Rule1Vanilla has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 59: Rule1White has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 61: Rule2Milk has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 62: Rule2Salsa has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 64: Rule2White has no variation.
```

```
print(modeloSGB)
```

```
## Stochastic Gradient Boosting
##
## 30000 samples
##    40 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 24000, 24000, 24000, 24000, 24000
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7392667  0.4785333
##   1                  100      0.7432000  0.4864000
##   1                  150      0.7429667  0.4859333
##   2                   50      0.7425667  0.4851333
##   2                  100      0.7434000  0.4868000
##   2                  150      0.7437333  0.4874667
##   3                   50      0.7444667  0.4889333
##   3                  100      0.7446667  0.4893333
##   3                  150      0.7449000  0.4898000
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
previsoes <- data.frame(observado = testSet$boughtLastOrder,
                        previsto = predict(modeloSGB, newdata = testSet))
confusionMatrix(previsoes$observado, previsoes$previsto,mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6813 2208
##          1  258  721
##
##                Accuracy : 0.7534
##                  95% CI : (0.7448, 0.7618)
##     No Information Rate : 0.7071
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2605
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7552
##                  Recall : 0.9635
##                      F1 : 0.8468
##              Prevalence : 0.7071
##          Detection Rate : 0.6813
##    Detection Prevalence : 0.9021
##       Balanced Accuracy : 0.6048
##
##        'Positive' Class : 0
##
```

```
# The result has improved, but it keeps the same problems, caused but predominance of classes 0.
# With balanced test dataset.

previsoes <- data.frame(observado = testSetB$boughtLastOrder,
                        previsto = predict(modeloSGB, newdata = testSetB))
confusionMatrix(previsoes$observado, previsoes$previsto,mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3760 1240
##          1 1301 3699
##
##                Accuracy : 0.7459
##                  95% CI : (0.7372, 0.7544)
##     No Information Rate : 0.5061
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4918
##
##  Mcnemar's Test P-Value : 0.2339
##
##               Precision : 0.7520
##                  Recall : 0.7429
##                      F1 : 0.7474
##              Prevalence : 0.5061
##          Detection Rate : 0.3760
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7459
##
##        'Positive' Class : 0
##
```

```
# The result improve a lot. Let's try anothe model.

# Extreme Gradient Boosting
modeloEGB <- train(variaveisModelo, data=trainSet, method='xgbLinear', trControl=ctrl)
print(modeloEGB)
```

```
## eXtreme Gradient Boosting
##
## 30000 samples
##    40 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 24000, 24000, 24000, 24000, 24000
## Resampling results across tuning parameters:
##
##   lambda  alpha  nrounds  Accuracy   Kappa
##   0e+00   0e+00   50      0.7341000  0.4682000
##   0e+00   0e+00  100      0.7315667  0.4631333
##   0e+00   0e+00  150      0.7253667  0.4507333
##   0e+00   1e-04   50      0.7331667  0.4663333
##   0e+00   1e-04  100      0.7294333  0.4588667
##   0e+00   1e-04  150      0.7271333  0.4542667
##   0e+00   1e-01   50      0.7345000  0.4690000
##   0e+00   1e-01  100      0.7290000  0.4580000
##   0e+00   1e-01  150      0.7257667  0.4515333
##   1e-04   0e+00   50      0.7331667  0.4663333
##   1e-04   0e+00  100      0.7294333  0.4588667
##   1e-04   0e+00  150      0.7261667  0.4523333
##   1e-04   1e-04   50      0.7331667  0.4663333
##   1e-04   1e-04  100      0.7292000  0.4584000
##   1e-04   1e-04  150      0.7263000  0.4526000
##   1e-04   1e-01   50      0.7344667  0.4689333
##   1e-04   1e-01  100      0.7287667  0.4575333
##   1e-04   1e-01  150      0.7262333  0.4524667
##   1e-01   0e+00   50      0.7362000  0.4724000
##   1e-01   0e+00  100      0.7306000  0.4612000
##   1e-01   0e+00  150      0.7270667  0.4541333
##   1e-01   1e-04   50      0.7370667  0.4741333
##   1e-01   1e-04  100      0.7309667  0.4619333
##   1e-01   1e-04  150      0.7273333  0.4546667
##   1e-01   1e-01   50      0.7394667  0.4789333
##   1e-01   1e-01  100      0.7315667  0.4631333
##   1e-01   1e-01  150      0.7270333  0.4540667
##
## Tuning parameter 'eta' was held constant at a value of 0.3
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 50, lambda = 0.1, alpha = 0.1 and eta = 0.3.
```

```
previsoes <- data.frame(observado = testSet$boughtLastOrder,
                        previsto = predict(modeloEGB, newdata = testSet))
confusionMatrix(previsoes$observado, previsoes$previsto,mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6649 2372
##          1  262  717
##
##                Accuracy : 0.7366
##                  95% CI : (0.7278, 0.7452)
##     No Information Rate : 0.6911
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2394
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7371
##                  Recall : 0.9621
##                      F1 : 0.8347
##              Prevalence : 0.6911
##          Detection Rate : 0.6649
##    Detection Prevalence : 0.9021
##       Balanced Accuracy : 0.5971
##
##        'Positive' Class : 0
##
```

```
# The result got worse.
# With balanced test sataset.

previsoes <- data.frame(observado = testSetB$boughtLastOrder,
                        previsto = predict(modeloEGB, newdata = testSetB))
confusionMatrix(previsoes$observado, previsoes$previsto,mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3690 1310
##          1 1260 3740
##
##                Accuracy : 0.743
##                  95% CI : (0.7343, 0.7515)
##     No Information Rate : 0.505
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.486
##
##  Mcnemar's Test P-Value : 0.3338
##
##               Precision : 0.7380
##                  Recall : 0.7455
##                      F1 : 0.7417
##              Prevalence : 0.4950
##          Detection Rate : 0.3690
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7430
##
##        'Positive' Class : 0
##
```

```
# Almost the same result as STB.

# Optimize the best model - Stochastic Gradient Boosting
ctrl <- trainControl(method = "cv", number=5)
grid <- expand.grid(interaction.depth = c(1, 10, 15),
                    n.trees = c(50, 300, 500),
                    shrinkage = c(.1, .3, .5),
                    n.minobsinnode = c(1, 5, 7))
modeloSGB <- train(variaveisModelo, data=trainSet, method='gbm', trControl=ctrl, tuneGrid=grid, verbose=FALSE)
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 39: Rule12% Milkfat has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 44: Rule1Cookies has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 46: Rule1Hot has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 49: Rule1Milk has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 54: Rule1Select-A-Size Paper Towels has no variati
on.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 55: Rule1Tomatoes has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 56: Rule1Tortilla Chips has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 57: Rule1Tortillas has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 58: Rule1Vanilla has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 59: Rule1White has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 61: Rule2Milk has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 62: Rule2Salsa has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 64: Rule2White has no variation.
```

```
print(modeloSGB)
```

```
## Stochastic Gradient Boosting
##
## 30000 samples
##    40 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 24000, 24000, 24000, 24000, 24000
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.minobsinnode  n.trees  Accuracy   Kappa
##   0.1         1                 1                50      0.7391000  0.4782000
##   0.1         1                 1               300      0.7440000  0.4880000
##   0.1         1                 1               500      0.7446000  0.4892000
##   0.1         1                 5                50      0.7386333  0.4772667
##   0.1         1                 5               300      0.7445333  0.4890667
##   0.1         1                 5               500      0.7440000  0.4880000
##   0.1         1                 7                50      0.7395000  0.4790000
##   0.1         1                 7               300      0.7444000  0.4888000
##   0.1         1                 7               500      0.7444000  0.4888000
##   0.1        10                 1                50      0.7472000  0.4944000
##   0.1        10                 1               300      0.7434000  0.4868000
##   0.1        10                 1               500      0.7410000  0.4820000
##   0.1        10                 5                50      0.7450333  0.4900667
##   0.1        10                 5               300      0.7435333  0.4870667
##   0.1        10                 5               500      0.7410667  0.4821333
##   0.1        10                 7                50      0.7466333  0.4932667
##   0.1        10                 7               300      0.7451333  0.4902667
##   0.1        10                 7               500      0.7404333  0.4808667
##   0.1        15                 1                50      0.7469667  0.4939333
##   0.1        15                 1               300      0.7427667  0.4855333
##   0.1        15                 1               500      0.7385000  0.4770000
##   0.1        15                 5                50      0.7455000  0.4910000
##   0.1        15                 5               300      0.7418667  0.4837333
##   0.1        15                 5               500      0.7376000  0.4752000
##   0.1        15                 7                50      0.7464667  0.4929333
##   0.1        15                 7               300      0.7420667  0.4841333
##   0.1        15                 7               500      0.7364667  0.4729333
##   0.3         1                 1                50      0.7427333  0.4854667
##   0.3         1                 1               300      0.7425000  0.4850000
##   0.3         1                 1               500      0.7413667  0.4827333
##   0.3         1                 5                50      0.7417333  0.4834667
##   0.3         1                 5               300      0.7436667  0.4873333
##   0.3         1                 5               500      0.7405333  0.4810667
##   0.3         1                 7                50      0.7424333  0.4848667
##   0.3         1                 7               300      0.7422000  0.4844000
##   0.3         1                 7               500      0.7417333  0.4834667
##   0.3        10                 1                50      0.7409333  0.4818667
##   0.3        10                 1               300      0.7194667  0.4389333
##   0.3        10                 1               500      0.7162333  0.4324667
##   0.3        10                 5                50      0.7423000  0.4846000
##   0.3        10                 5               300      0.7218667  0.4437333
##   0.3        10                 5               500      0.7173333  0.4346667
##   0.3        10                 7                50      0.7383667  0.4767333
##   0.3        10                 7               300      0.7184000  0.4368000
##   0.3        10                 7               500      0.7111333  0.4222667
##   0.3        15                 1                50      0.7371333  0.4742667
##   0.3        15                 1               300      0.7123333  0.4246667
##   0.3        15                 1               500      0.7080333  0.4160667
##   0.3        15                 5                50      0.7365333  0.4730667
##   0.3        15                 5               300      0.7155000  0.4310000
##   0.3        15                 5               500      0.7112333  0.4224667
##   0.3        15                 7                50      0.7353000  0.4706000
##   0.3        15                 7               300      0.7133333  0.4266667
##   0.3        15                 7               500      0.7093667  0.4187333
##   0.5         1                 1                50      0.7405333  0.4810667
##   0.5         1                 1               300      0.7396667  0.4793333
##   0.5         1                 1               500      0.7385667  0.4771333
##   0.5         1                 5                50      0.7421000  0.4842000
##   0.5         1                 5               300      0.7400667  0.4801333
##   0.5         1                 5               500      0.7395333  0.4790667
##   0.5         1                 7                50      0.7429000  0.4858000
##   0.5         1                 7               300      0.7408667  0.4817333
##   0.5         1                 7               500      0.7394667  0.4789333
##   0.5        10                 1                50      0.7269000  0.4538000
##   0.5        10                 1               300      0.6963333  0.3926667
##   0.5        10                 1               500      0.6904000  0.3808000
##   0.5        10                 5                50      0.7294000  0.4588000
##   0.5        10                 5               300      0.6955667  0.3911333
##   0.5        10                 5               500      0.6891333  0.3782667
```

```
##   0.5        10                 7               50      0.7294333  0.4588667
##   0.5        10                 7              300      0.6964000  0.3928000
##   0.5        10                 7              500      0.6925667  0.3851333
##   0.5        15                 1               50      0.7205000  0.4410000
##   0.5        15                 1              300      0.6911667  0.3823333
##   0.5        15                 1              500      0.6860333  0.3720667
##   0.5        15                 5               50      0.7185000  0.4370000
##   0.5        15                 5              300      0.6887333  0.3774667
##   0.5        15                 5              500      0.6891000  0.3782000
##   0.5        15                 7               50      0.7227000  0.4454000
##   0.5        15                 7              300      0.6943667  0.3887333
##   0.5        15                 7              500      0.6921000  0.3842000
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 50, interaction.depth = 10, shrinkage = 0.1 and n.minobsinnode = 1.
```

```
# Best parameters
modeloSGB$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 10      50                10       0.1              1
```

```
# Train the model with the best parameters
modeloSGB <- train(variaveisModelo, data=trainSet, method='gbm', trControl=ctrl, tuneGrid=modeloSGB$bestTune, verbose=FALSE)
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 39: Rule12% Milkfat has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 44: Rule1Cookies has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 46: Rule1Hot has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 49: Rule1Milk has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 54: Rule1Select-A-Size Paper Towels has no variati
on.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 55: Rule1Tomatoes has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 56: Rule1Tortilla Chips has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 57: Rule1Tortillas has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 58: Rule1Vanilla has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 59: Rule1White has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 61: Rule2Milk has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 62: Rule2Salsa has no variation.
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution = "bernoulli", : variable 64: Rule2White has no variation.
```

```
print(modeloSGB)
```

```
## Stochastic Gradient Boosting
##
## 30000 samples
##    40 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 24000, 24000, 24000, 24000, 24000
## Resampling results:
##
##   Accuracy   Kappa
##   0.7471667  0.4943333
##
## Tuning parameter 'n.trees' was held constant at a value of 50
## Tuning parameter 'interaction.depth' was held constant at a value of 10
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## Tuning parameter 'n.minobsinnode' was held constant at a value of 1
```

```
previsoes <- data.frame(observado = testSet$boughtLastOrder,
                        previsto = predict(modeloSGB, newdata = testSet))
confusionMatrix(previsoes$observado, previsoes$previsto,mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6721 2300
##          1  234  745
##
##                Accuracy : 0.7466
##                  95% CI : (0.738, 0.7551)
##     No Information Rate : 0.6955
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.2607
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##               Precision : 0.7450
##                  Recall : 0.9664
##                      F1 : 0.8414
##              Prevalence : 0.6955
##          Detection Rate : 0.6721
##    Detection Prevalence : 0.9021
##       Balanced Accuracy : 0.6055
##
##        'Positive' Class : 0
##
```

```
# Balanced test dataset

previsoes <- data.frame(observado = testSetB$boughtLastOrder,
                        previsto = predict(modeloSGB, newdata = testSetB))
confusionMatrix(previsoes$observado, previsoes$previsto,mode = "prec_recall")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3680 1320
##          1 1216 3784
##
##                Accuracy : 0.7464
##                  95% CI : (0.7378, 0.7549)
##     No Information Rate : 0.5104
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4928
##
##  Mcnemar's Test P-Value : 0.04082
##
##               Precision : 0.7360
##                  Recall : 0.7516
##                      F1 : 0.7437
##              Prevalence : 0.4896
##          Detection Rate : 0.3680
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7465
##
##        'Positive' Class : 0
##
```

```
# The final result is a little bit better than what we got before.
# In order to improve the performance, some more indicators related to User/Product
# may be created. Further, we could train the model full, with no cuts for computational
# reasons. Colaborative filtering techniques may be used as well.

stopCluster(cl)
```