Choose Your Own: Retail Store Final Project

HarvardX Data Science Program Professional Certificate

Mauricio Patón

January, 2021

Contents

1 Final Report

1.1 Introduction

The Capstone Course of the HarvardX Data Science Program Professional Certificate offered in edX requires, along with the MovieLens project a Choose Your Own Project.

In this case, the student needs to select a database of his interest in order to perform a Machine Learning Project in which to predict a defined metric.

For this work, a Retail Shop dataset has been selected as the Case Study to work with. Identifying behaviors of customers could lead into marketing strategies that allow to link specific products to a particular target of customers. In addition, it is important for a store to obtain a rough estimate of their revenue based on the customers that might purchase items in the store.

A good model should provide a reasonably accurate prediction to the end-user. In order to measure the accuracy of a recommendation, different metrics can be used, such as the Root Mean Squared Error (RMSE), the Mean Squared Error (MSE). In this work, the RMSE will be the metric assigned to measure how close (or not) the estimation of how much a given user is going to spend.

1.2 Methodology: Data Exploration and Preparation

In this section, an exploration of the data is conducted. As previously stated, the data corresponding to a Retail Shop dataset was downloaded. The data downloaded contains three csv files, one for customer information, one that contains information regarding the products categories and subcategories and another file that contains information regarding the transactions.

The files are loaded into R using the following code:

```
# Load csv data for initial exploration
customer <- read.csv("https://raw.githubusercontent.com/maupagas/RetailStore_edX/main/retail-store-files
transactions <- read.csv("https://raw.githubusercontent.com/maupagas/RetailStore_edX/main/retail-store-
product_info <- read.csv("https://raw.githubusercontent.com/maupagas/RetailStore_edX/main/retail-store-
```

1.2.1 Preliminary data exploration

Once the data is loaded, the exploratory analysis can start. Before proceeding with any modeling, the first step consists of understanding the data at hand. Therefore, the functions from the basic package *summary*, *str* and *head* were executed.

```
# Data Exploration
summary(product_info)
```

```
##  prod_cat_code               prod_cat prod_sub_cat_code          prod_subcat
##  Min.   :1.000   Bags              :2   Min.   : 1.00     Mens            : 3
##  1st Qu.:2.500   Books             :6   1st Qu.: 3.00     Women           : 3
##  Median :4.000   Clothing          :3   Median : 5.00     Kids            : 2
##  Mean   :3.739   Electronics       :5   Mean   : 6.13     Academic        : 1
##  3rd Qu.:5.000   Footwear          :3   3rd Qu.:10.00     Audio and video: 1
##  Max.   :6.000   Home and kitchen:4   Max.   :12.00     Bath            : 1
##                                                         (Other)         :12
```

```
summary(customer)
```

```
##   customer_Id             DOB       Gender     city_code
##  Min.   :266783   27-12-1988:   7    :   2   Min.   : 1.000
##  1st Qu.:268912   01-08-1975:   5   F:2753   1st Qu.: 3.000
##  Median :271028   17-09-1982:   5   M:2892   Median : 5.000
##  Mean   :271037   20-03-1972:   5            Mean   : 5.473
##  3rd Qu.:273180   23-03-1982:   5            3rd Qu.: 8.000
##  Max.   :275265   01-02-1970:   4            Max.   :10.000
##                   (Other)   :5616            NA's   :2
```

```
summary(transactions)
```

```
##  transaction_id         cust_id            tran_date     prod_subcat_code
##  Min.   :3.269e+06   Min.   :266783   13-07-2011:   35   Min.   : 1.000
##  1st Qu.:2.494e+10   1st Qu.:268935   21-12-2013:   33   1st Qu.: 3.000
##  Median :5.009e+10   Median :270980   22-11-2011:   33   Median : 5.000
##  Mean   :5.007e+10   Mean   :271022   23-10-2011:   33   Mean   : 6.149
##  3rd Qu.:7.533e+10   3rd Qu.:273114   25-09-2011:   33   3rd Qu.:10.000
##  Max.   :9.999e+10   Max.   :275265   25-08-2012:   32   Max.   :12.000
##                                       (Other)   :22854
##  prod_cat_code        Qty               Rate              Tax
##  Min.   :1.000   Min.   :-5.000   Min.   :-1499.0   Min.   :  7.35
##  1st Qu.:2.000   1st Qu.: 1.000   1st Qu.:  312.0   1st Qu.: 98.28
##  Median :4.000   Median : 3.000   Median :  710.0   Median :199.08
##  Mean   :3.764   Mean   : 2.432   Mean   :  636.4   Mean   :248.67
```

```
##  3rd Qu.:5.000   3rd Qu.: 4.000   3rd Qu.: 1109.0   3rd Qu.:365.71
##  Max.   :6.000   Max.   : 5.000   Max.   : 1500.0   Max.   :787.50
##
##    total_amt               Store_type
##  Min.   :-8270.9   e-Shop        :9311
##  1st Qu.:  762.5   Flagship store:4577
##  Median : 1754.7   MBR           :4661
##  Mean   : 2107.3   TeleShop      :4504
##  3rd Qu.: 3569.2
##  Max.   : 8287.5
##
```

```
head(transactions)
```

```
##   transaction_id cust_id  tran_date prod_subcat_code prod_cat_code Qty  Rate
## 1    80712190438  270351 28-02-2014                1             1  -5  -772
## 2    29258453508  270384 27-02-2014                5             3  -5 -1497
## 3    51750724947  273420 24-02-2014                6             5  -2  -791
## 4    93274880719  271509 24-02-2014               11             6  -3 -1363
## 5    51750724947  273420 23-02-2014                6             5  -2  -791
## 6    97439039119  272357 23-02-2014                8             3  -2  -824
##       Tax total_amt Store_type
## 1 405.300 -4265.300     e-Shop
## 2 785.925 -8270.925     e-Shop
## 3 166.110 -1748.110   TeleShop
## 4 429.345 -4518.345     e-Shop
## 5 166.110 -1748.110   TeleShop
## 6 173.040 -1821.040   TeleShop
```

```
n_users <- n_distinct(transactions$cust_id)
n_transactions <- n_distinct(transactions$transaction_id)
```

An initial exploration reveals that there are 5506 users and 20878 transactions. From an initial exploratory analysis we observe that there are 6 categories of products and 12 subcategories.

The date is also written as factors. Therefore some conversions are required to be able to make better use of the data.

Therefore, we will do the following:

- Convert the data to a datetime using the *lubridate* package.
- Convert categories and subcategories to factors.
- Convert the year of birth of the customer into age.
- Ensure that the columns have the same name in all variables.
- Fix missing values and NA's in the dataset.

```
# Convert time factors into date
date_1 <- as.Date(transactions$tran_date, "%d-%m-%Y")
date_2 <- as.Date(transactions$tran_date, "%d/%m/%Y")
transactions$tran_date <- ifelse(is.na(date_1), date_2, date_1)

# Fix the date into appropriate format
transactions$tran_date <- as.Date(transactions$tran_date, origin = "1970-01-01")

# Convert categories as factors in both variables
transactions$prod_subcat_code <- as.factor(transactions$prod_subcat_code)
transactions$prod_cat_code <- as.factor(transactions$prod_cat_code)
```

```r
# Convert product into factors
product_info$prod_sub_cat_code <- as.factor(product_info$prod_sub_cat_code)
product_info$prod_cat_code <- as.factor(product_info$prod_cat_code)
```

Now, we are going to merge data between variables. For that, we need to make sure that all columns have the same name between dataframes.

```r
# Merge data

# First, have the same column name for both variables
colnames(product_info)[3] <- "prod_subcat_code"
colnames(customer)[1] <- "cust_id"

# Once columns have the same name, we can join the data
transactions.new <- transactions %>%
                            left_join(product_info,
                                      by = c('prod_subcat_code',
                                             'prod_cat_code')) %>%
                            left_join(customer, by = 'cust_id') %>%
                            unite("prod_category",
                                  c(prod_cat, prod_subcat),
                                  sep = " ",
                                  remove = F) # Keep the original variables

# Obtain the age of the customer
transactions.new$cust_age <- year(transactions.new$tran_date) -
                             year(as.Date(transactions.new$DOB, "%d-%m-%Y"))
```

Looking at the gender column, it appears that there are 9 customers of them do not belong to any gender. Therefore, this needs to be fixed. One option would be to eliminate those entries. The other option however, is to assign a gender based on the probability that the genders appear in the data base. The same occur for the city code. Two entries do not have a city code. We will sample randomly to fit the missing values

```r
## Fix the Genders
summary(transactions.new$Gender)
```

```
##           F     M
##     9 11233 11811
```

```r
nonGender <- which(transactions.new$Gender == "" )

# Replace by sampling randomly according to the proportion in the sample
transactions.new$Gender[nonGender] <- sample(transactions.new$Gender,
                                             length(nonGender),
                                             replace = T)
transactions.new$Gender <- as.factor(transactions.new$Gender)

## Avoid having NA's in the variable city_code
nonCity <- which(is.na(transactions.new$city_code))
transactions.new$city_code[nonCity] <- sample(transactions.new$city_code,
                                              length(nonCity),
                                              replace = T)
```

Once all has been fixed, we can order the columns as we wish with the following code.

```
# Now we can reorder the data properly...
transactions.ord <- transactions.new[ , c("cust_id", "DOB", "cust_age", "Gender",
                                          "city_code", "transaction_id", "tran_date",
                                          "prod_subcat_code","prod_cat_code",
                                          "prod_category", "prod_cat", "prod_subcat",
                                          "Store_type", "Qty", "Rate", "Tax", "total_amt")]
```

We need also a dataset for future work that aggregates all the information *by customer* instead of by transaction.

```
## Aggregate also data by customers
cust.desc <- transactions.ord %>% filter(total_amt > 0) %>%
  group_by(cust_id) %>%
  summarise(n = n(),
            age = mean(cust_age),
            city_code = mean(city_code),
            Gender = Gender,
            amount = sum(total_amt)) %>%
            unique()
```

1.3 Methodology: Exploratory Graphs

First, let's do some exploratory graphics to understand the transaction occurring in our retail store.

1.3.1 Amount spent in the store

Let's see in Figure 1 how is the amount of money spent in the store:

```
# Distribution plot of ALL transactions
transactions %>% ggplot(aes(total_amt)) +
                        geom_histogram(bins = 20, color = "black", fill = "steelblue") +
                        xlab("Amount paid") +
                        ylab("Number of transactions")
```

Figure 1 shows that the amount spent in returns is relatively low compared to the proportion that correspond to sales. We will ignore therefore for this project those products that consist of returns. Let's make a histogram then of only the sales by filtering the positive transactions. This can be seen in Figure 1.

```
# Distribution plot of only sales
transactions %>% filter(total_amt > 0) %>%
                 ggplot(aes(total_amt)) +
                 geom_histogram(bins = 20, color = "black", fill = "steelblue") +
                 xlab("Amount paid") +
                 ylab("Number of transactions")
```

Figure 1 shows that most of the majority of the transactions correspond to expenditures of less than 2000 $. However, there are a small number of transactions that are over 6000 dollars.

1.3.2 Gender distribution

The bias on the gender distribution (if any) when purchasing items in the store can be observed in Figure 3:

```
# Gender distribution
transactions.ord %>% filter(total_amt>0) %>%
    group_by(cust_age, Gender) %>% summarise(n = n()) %>%
    ggplot(aes(cust_age, n, fill = Gender)) +
           geom_bar(stat = "identity", color = "black", position = "dodge") +
```

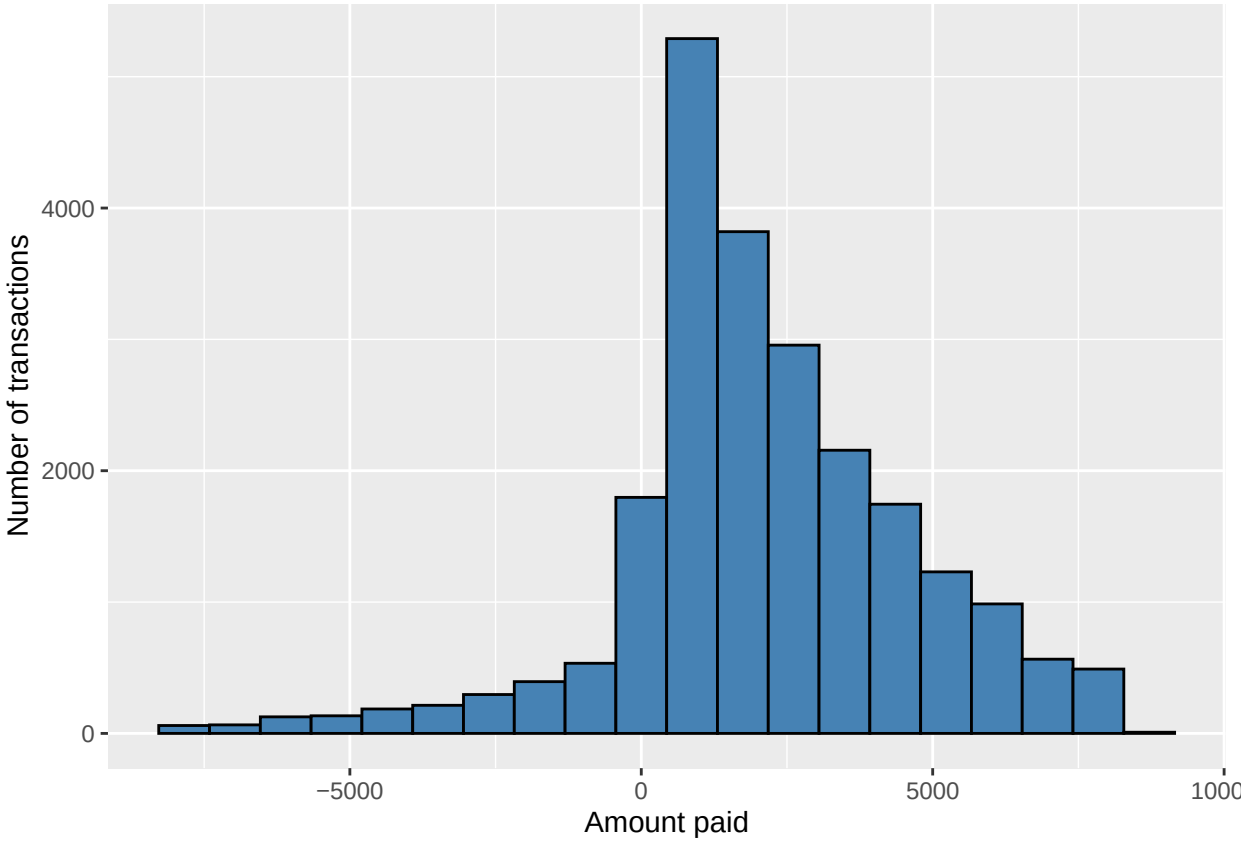


Figure 1: Amount paid per transaction

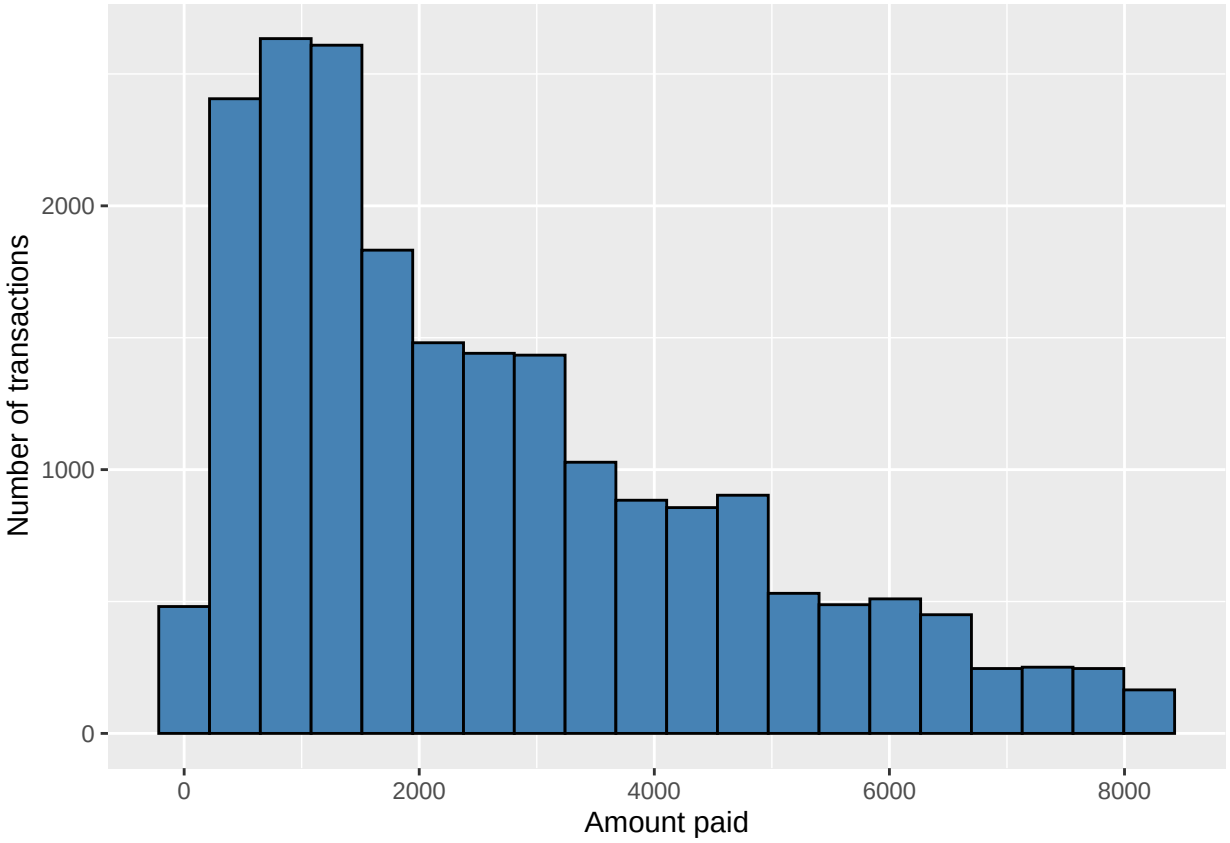


Figure 2: Amount paid per transaction

```
        scale_y_continuous(breaks = seq(0, 600, 100), labels = number) +
        xlab("Customer Age") + ylab("Number of customers")
```

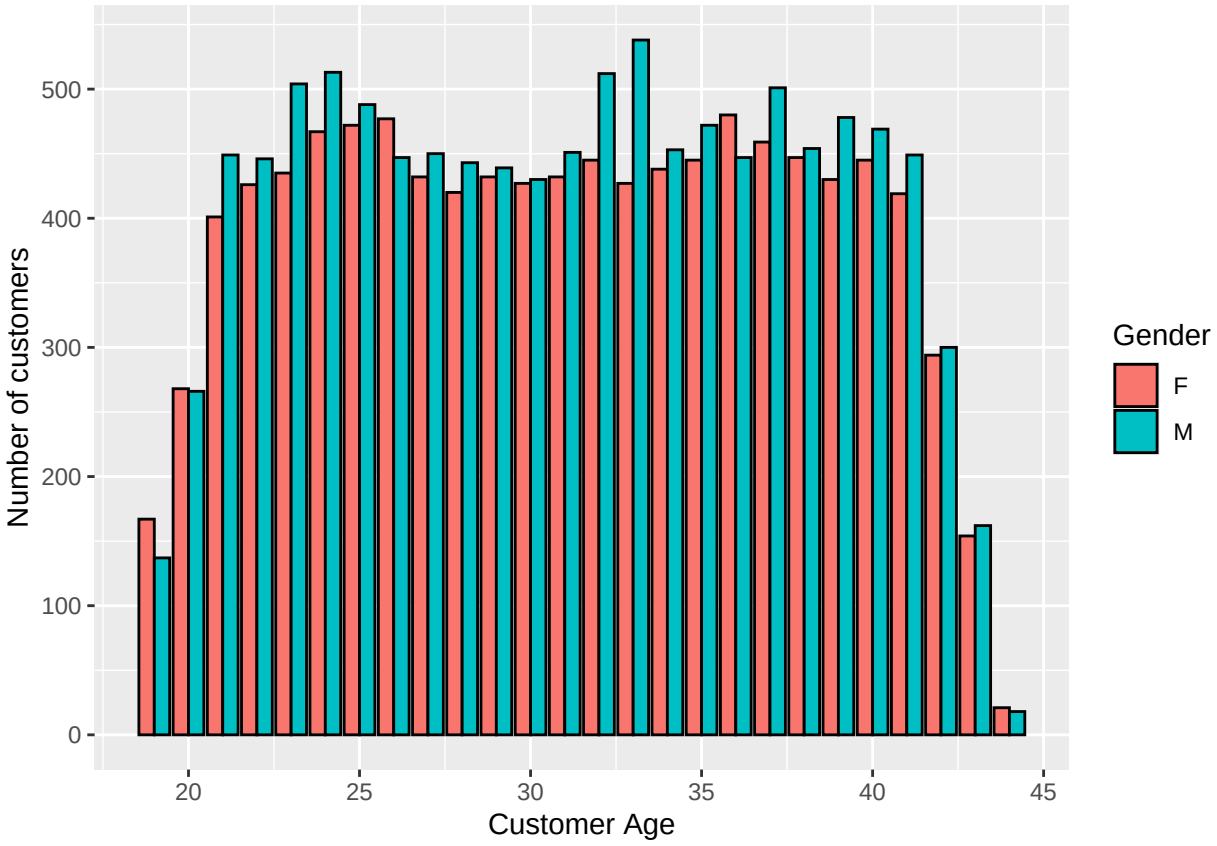


Figure 3: Amount paid per gender.

Figure 3 shows that here is some majority of sex in some ages (e.g. 32-33 years old), however it does not seem to be much big of a difference. The customer age range provided in the data comprises from 20-45 years old. Therefore, this is a signal of somehow a bias in the whole sample, since it does not capture teenagers nor 45+ people purchases. The latter ones can even have a significant purchasing power, but however, it is not available in the data provided.

1.3.3 Average of amount spent over time

Another important aspect to evaluate is the amount of money spent throughout the years and also to observe how much is spent each month, to observe if there are seasonality trends. The average amount spent each month per retail store is shown in Figure 4:

```
# Plot per month
transactions.ord %>% filter(total_amt > 0) %>%
                     mutate(year = year(tran_date), month = month(tran_date)) %>%
                     ggplot(aes(as.factor(month), total_amt, fill = Store_type)) +
                            geom_boxplot() +
                            scale_x_discrete(breaks = 1:12,
                                             labels = c("Jan", "Feb", "Mar",
                                                        "Apr", "May", "Jun",
```

```
                                                    "Jul", "Aug", "Sep",
                                                    "Oct", "Nov", "Dic"))+
                        xlab("Month") +
                        ylab("Average Amount")
```

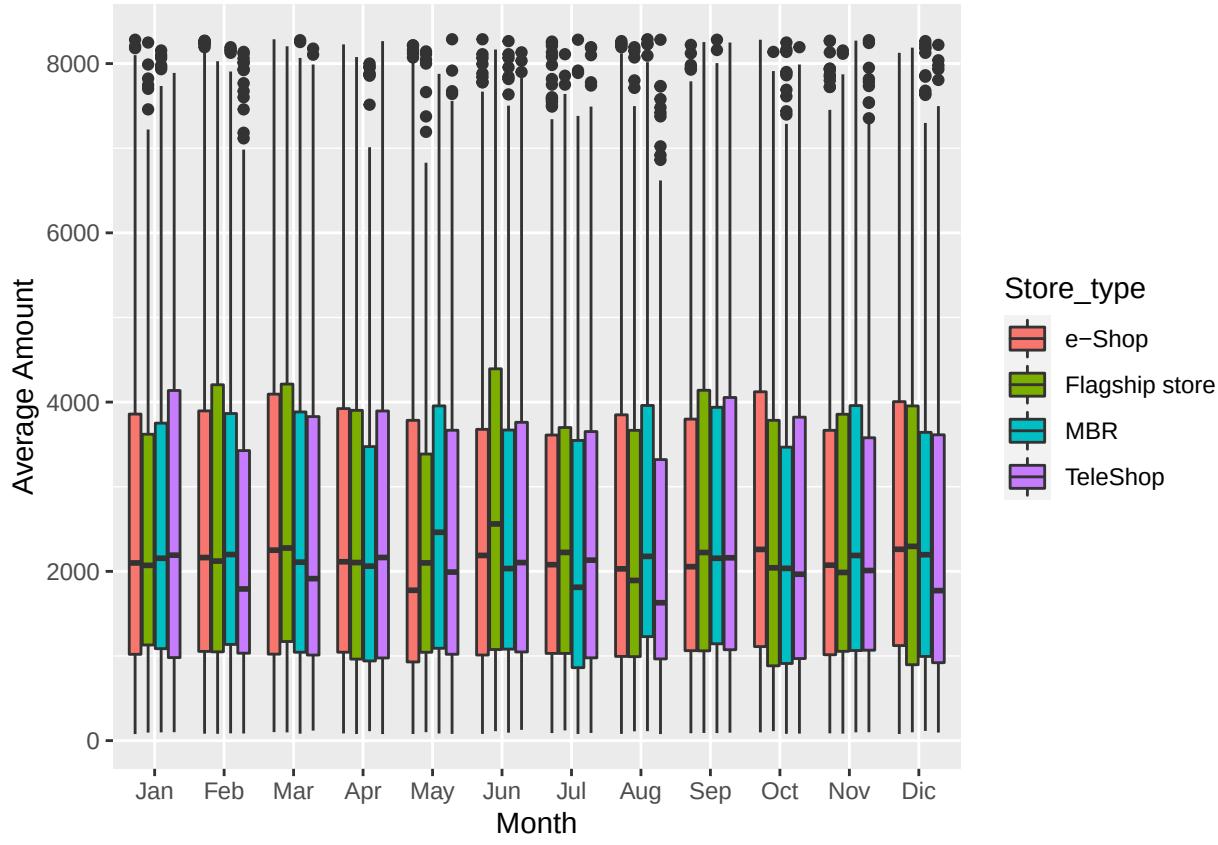


Figure 4: Average amount paid per store type.

Figure 5 shows the amount of money spent each year:

```
# Plot per year
transactions.ord %>% filter(total_amt > 0) %>%
                     mutate(year = year(tran_date), month = month(tran_date)) %>%
                     ggplot(aes(as.factor(year), total_amt, fill = prod_cat)) +
                            geom_boxplot() +
                            xlab("Year") +
                            ylab("Average Amount")
```

Yearly expenditures appear to be quite stable over the years, as per Figure 5. There is not a real trend of increasing or decreasing consistently over the years. Therefore, finding a time bias should be complicated.

We can also see the total amount of money spent in products yearly in Figure 6:

```
# See the amount of money spent per category per year
transactions.ord %>% filter(total_amt > 0) %>%
                     mutate(year = year(tran_date), month = month(tran_date)) %>%
                     group_by(prod_cat, year) %>%
```

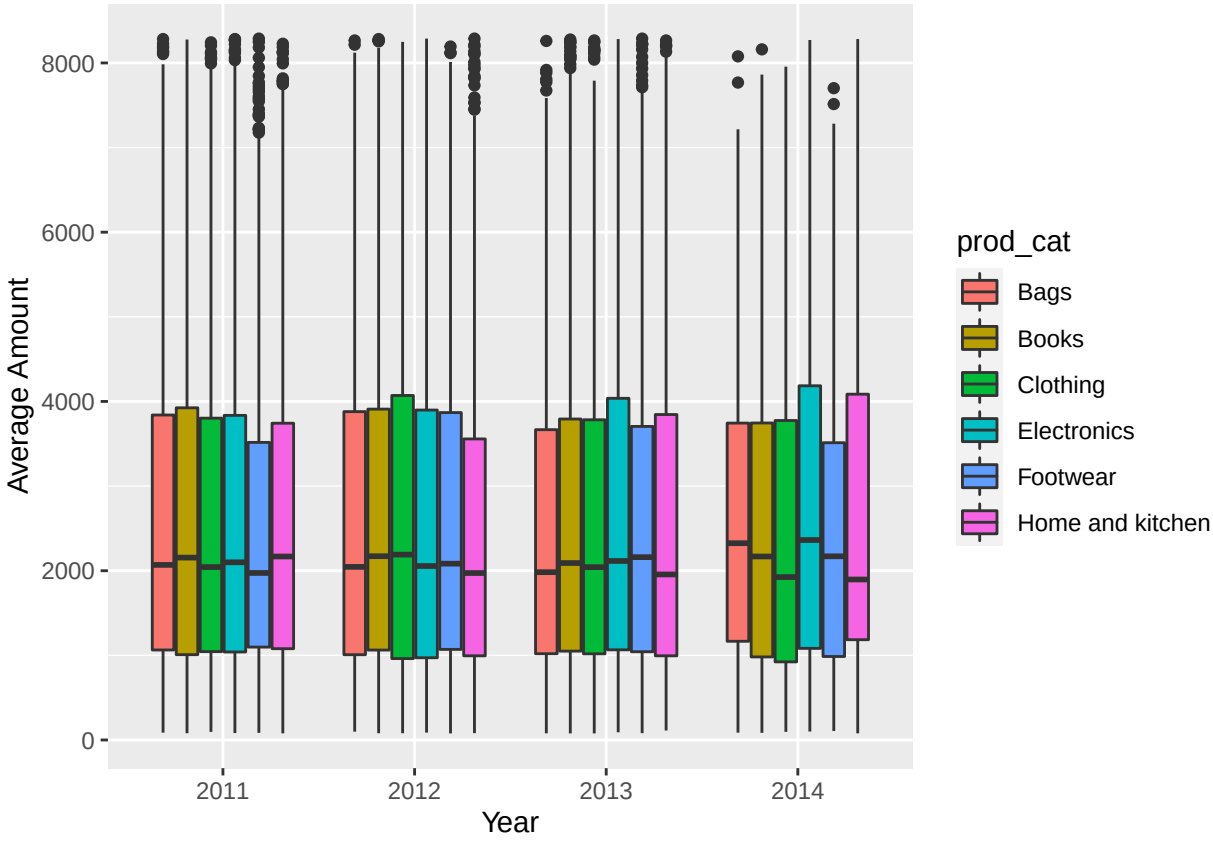


Figure 5: Amount paid per category.

```
                summarise(n = n(), total_amount = sum(total_amt)) %>%
                ggplot(aes(x = year, y = total_amount, fill = prod_cat)) +
                       geom_bar(stat="identity", color = "black",
                                position = "dodge") +
                       ylab("Total Amount Spent") +
                       xlab("Year") +
                       scale_y_continuous(labels = number)
```

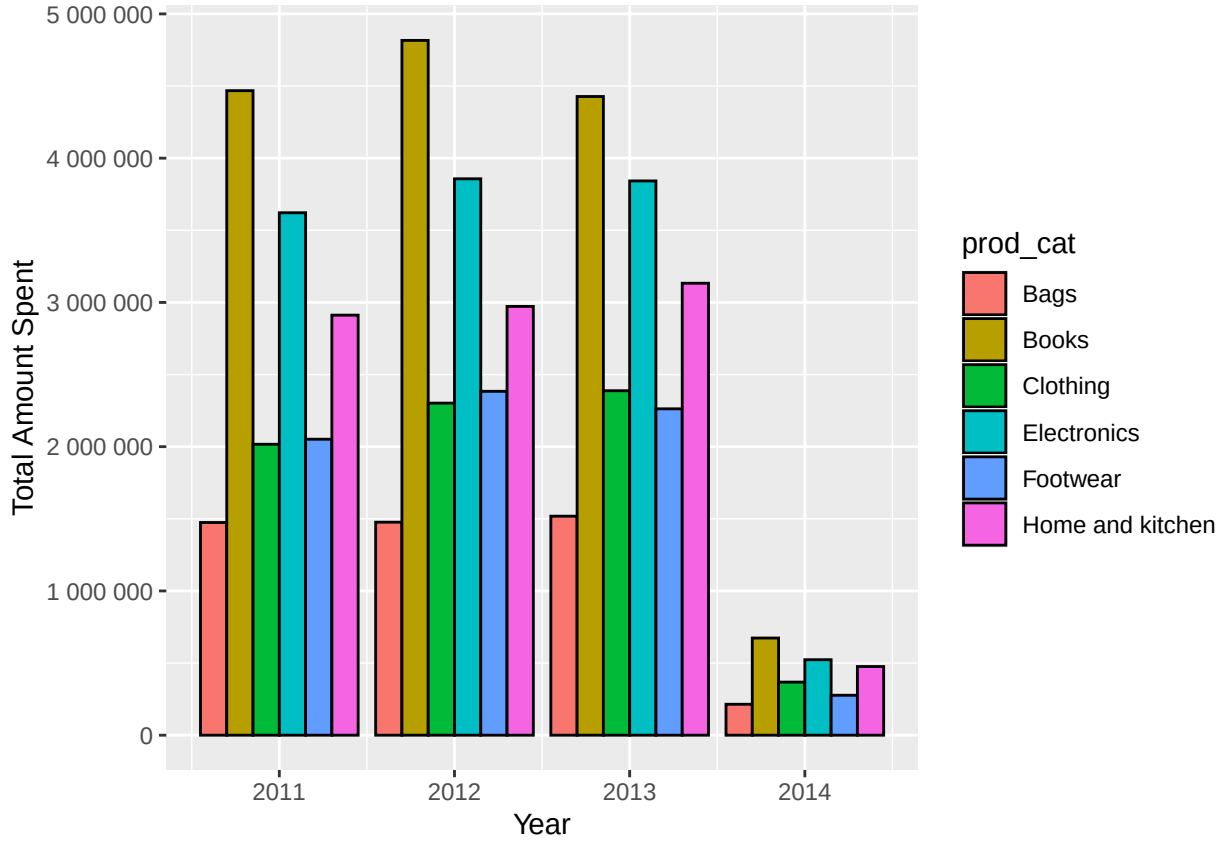


Figure 6: Total amount spent per product category each year.

Figure 6 shows significant differences between total amount of money spent per category. Therefore, the category could be a bias that could impact the amount of money to be spent. Let's see how people spends this money during the year in Figure 7:

```
# See the amount of money spent per category per month
transactions.ord %>% filter(total_amt > 0) %>%
                     mutate(year = year(tran_date), month = month(tran_date)) %>%
                     group_by(prod_cat, month) %>%
                     summarise(n = n(), total_amount = sum(total_amt)) %>%
                     ggplot(aes(x = month, y = total_amount, fill = prod_cat)) +
                            geom_bar(stat = "identity", color = "black",
                                     position = "dodge") +
                            ylab("Total Amount Spent") +
                            xlab("Month") +
                            scale_x_continuous(breaks = 1:12,
```

```
                                        labels = c("Jan", "Feb", "Mar",
                                                   "Apr", "May", "Jun",
                                                   "Jul", "Aug", "Sep",
                                                   "Oct", "Nov", "Dic")) +
                              scale_y_continuous(breaks = seq(0, 1.5e6, 2.5e5),
                                                 labels = number)
```

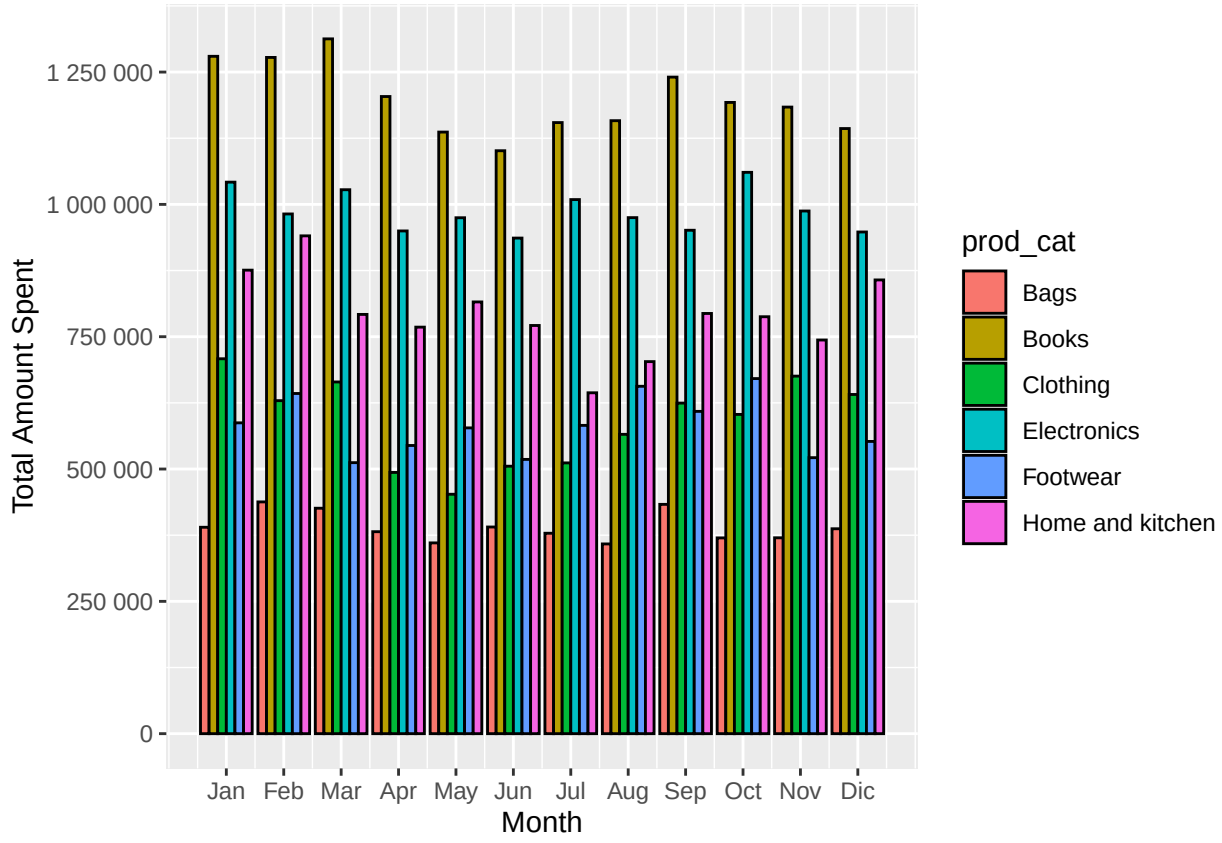


Figure 7: Total amount spent per product category each month.

There might be some seasonality in products such as books as seen in Figure 7, when their sales spike in September and during the first months of the year (January-March).

1.3.4 Amount of money spent per category at each age

Let's see in 8 how much is spent per product category to observe is some products are more favored than others per age group.

```
# Alternative plot by lines
transactions.ord %>% filter(total_amt>0) %>%
                     group_by(cust_age, prod_cat) %>%
                     summarise(n = n(), total_amount = sum(total_amt)) %>%
                     ggplot(aes(x = cust_age, y = total_amount, color = prod_cat)) +
                            geom_point() + geom_line() +
                            ylab("Total Amount Spent") +
                            xlab("Customer Age") +
```

```
                        scale_y_continuous(labels = number) +
                        scale_color_discrete(name = "Product Category")
```

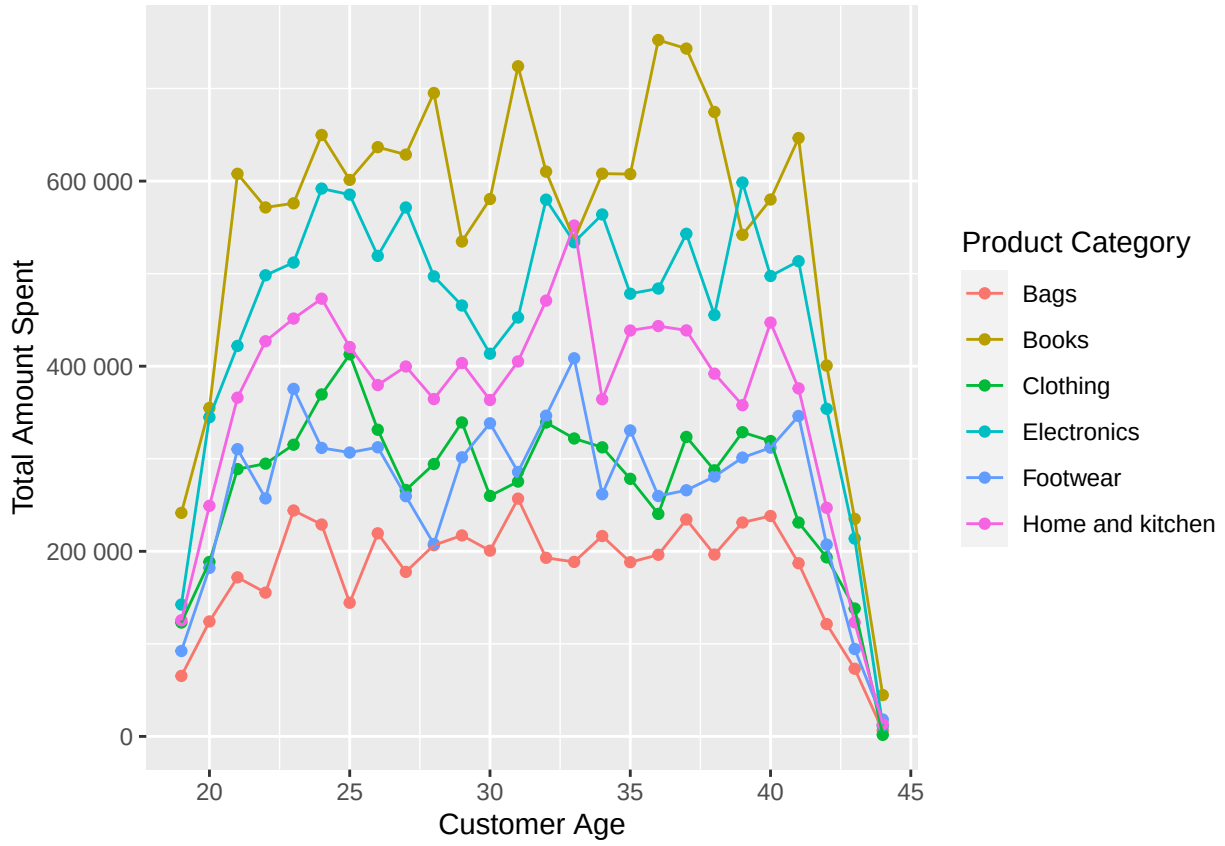


Figure 8: Total amount spent per age.

As seen previously in Figure 7, Figure 8 also confirms that there are not big differences per age. Only a change of trend is seen in people of 32-33 years old who spend more money in home and kitchen than in books. This could be explained by the age they get married or have kids, resulting into moving to another house, with the subsequent increase in these costs.

1.3.5 Users Histogram

One important aspect when considering a retail store is to know if our customers are normally new, or if they are recurrent. We can see this in Figure 9.

```
summary.transactions <- transactions.ord %>% filter(total_amt>0) %>%
                                             group_by(cust_id) %>%
                                             summarise(n = n(),
                                                       amount  = sum(total_amt))

avg_transaction <- mean(summary.transactions$amount)
avg_n_usr <- mean(summary.transactions$n)

summary.transactions %>% ggplot(aes(n)) +
                                geom_histogram(binwidth = 1,
```

```
                        fill = "steelblue",
                        color = "black") +
  scale_x_continuous(breaks = 1:11) +
  geom_vline(xintercept = avg_n_usr,
             col = "maroon",
             lty = 2,
             lwd = 1) +
  xlab("Number of times a user repeats") +
  ylab("Number of users")
```

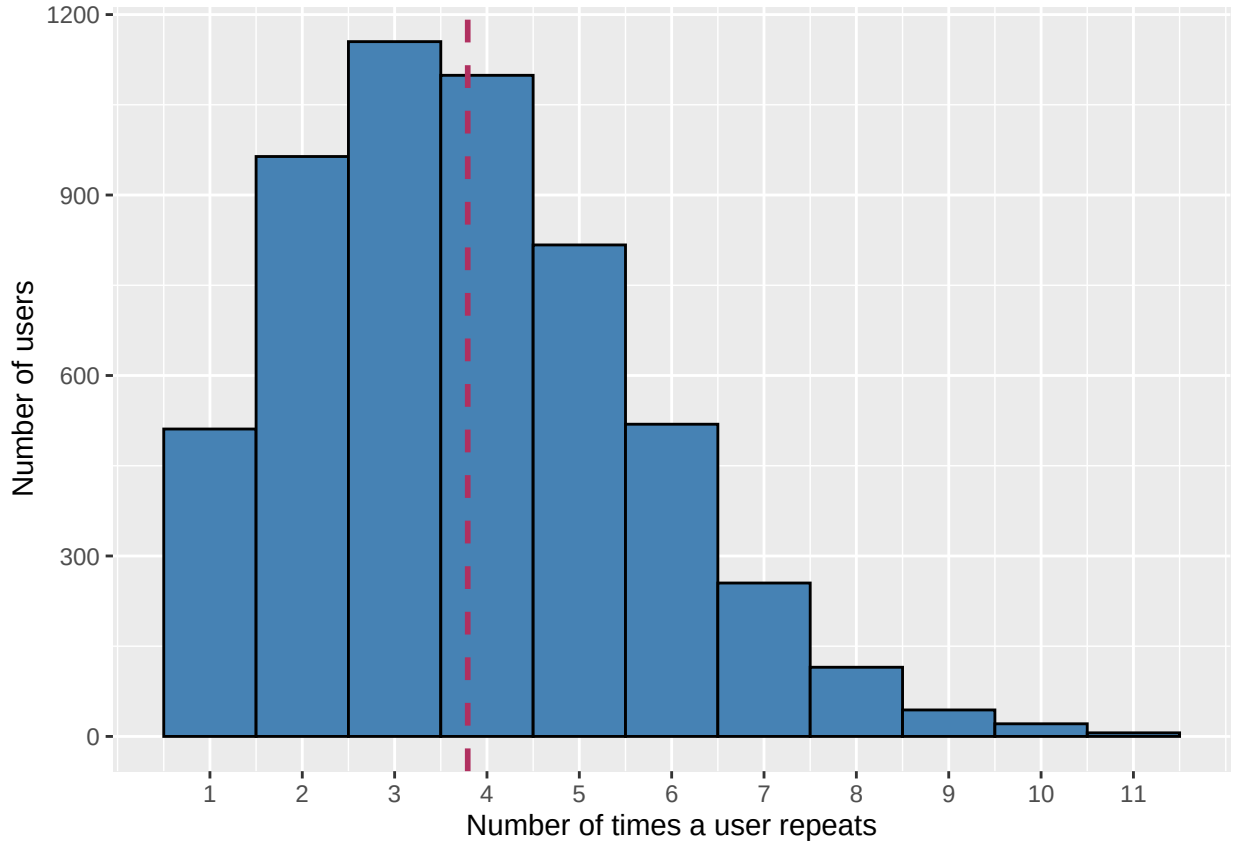


Figure 9: Number of times a user purchases in the store.

Figure 9 shows that customers return to the store within 4 years an average of 3.7915002. It is also important to see how much money is each user spending in the store. We can see that in Figure 10.

```
summary.transactions %>% ggplot(aes(amount)) +
  geom_histogram(bins = 30,
                 fill = "steelblue",
                 color = "black") +
  # facet_wrap(~Gender) +
  # scale_x_continuous(breaks = 1:11) +
  geom_vline(xintercept = avg_transaction,
             col = "maroon",
             lty = 2,
             lwd = 1) +
```

```
                    xlab("Amount spent by user") +
                    ylab("Number of users")
```

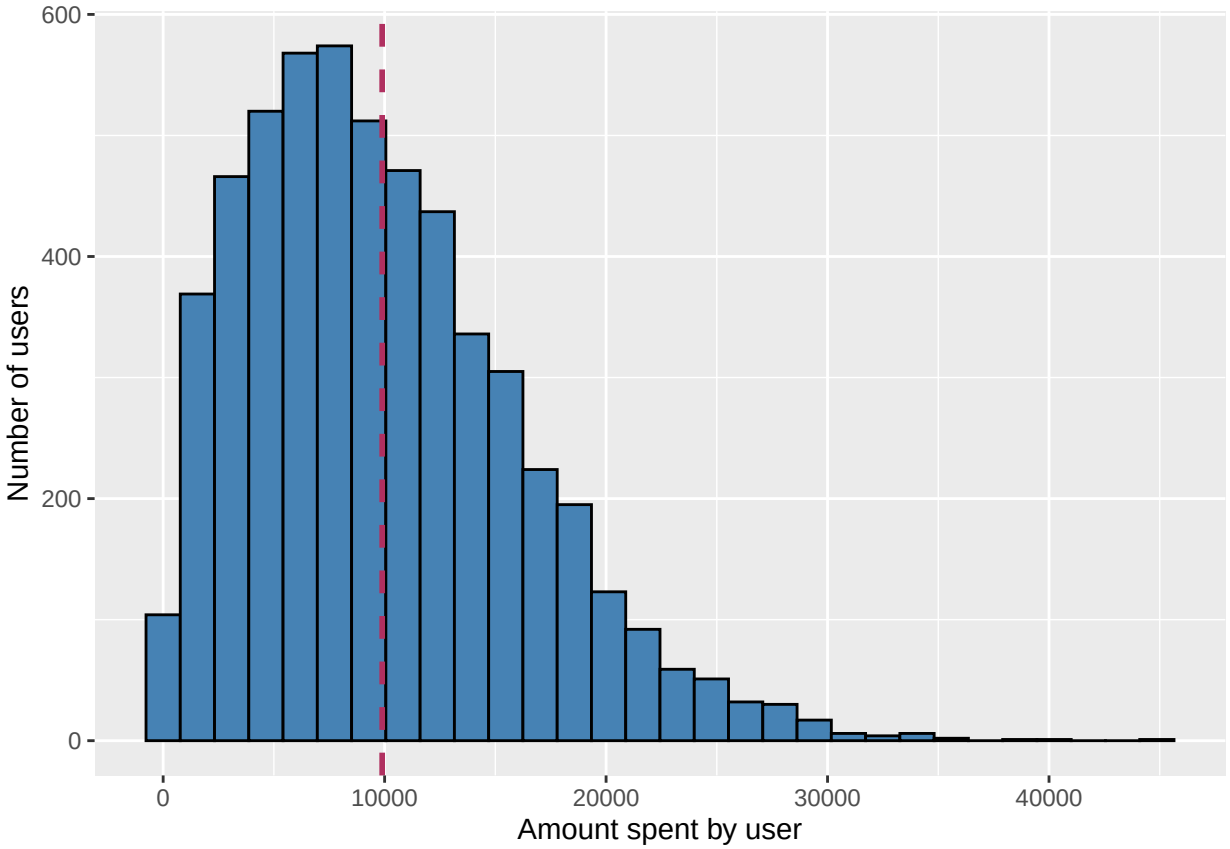


Figure 10: Histogram of the total amount of money spent in the store

Even in the average that each user spends in the store is of 9890, we see in Figure 10 that there are few users that spend over 30000. Is there any possibility to move the curve to the right?

1.4 Model Development

Looking at the data, we do not seem to see a particular bias over time or by city region or anything similar. If any, it appears to occur a bias on the group of the item.

We are going to try to attempt different regression models in order to predict the amount that a given customer can spend in the store

Let's first prepare the data by grouping the transactions by customer ID and collect the age, the city code, the gender and the total amount spent by the customer. Remember that for simplicity purposes, we are only using those transactions that are buys.

We will use the data aggregated by customer for our predictions.

```
#Obtain the average amount
avg_amount <- mean(cust.desc$amount)

# Initialize the tibble
rmse_results <- tibble()
```

The average amount of the customer spent in our retail store is 9890 dollars.

1.4.1 K-Nearest Neighbors

The first algorithm we are going to use is k-nearest neighbors, to try to evaluate a regression using as predictors the number of times going to the shop, the age and the city code. In order to be able to use k-nearest neighbors, we need to create dummy variables for the factors, and also to scale numerical variables between 0 and 1. Otherwise, the value of the variables has an impact in the prediction of the algorithm.

Therefore, let's create first the dummies and add it to the customer description data frame:

```
# Prepare dummy variables for the k-nearest-neighbors
dms1 <- dummy("Gender", as.data.frame(cust.desc), sep = "_")
```

```
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts = FALSE):
## non-list contrasts argument ignored
```

```
dms2 <- dummy("city_code", as.data.frame(cust.desc), sep = "_")
```

```
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts = FALSE):
## non-list contrasts argument ignored
```

```
dms2 <- dms2[,-11]

#Explore that the number of the variables is the correct one
head(dms2)
```

```
##   city_code_1 city_code_2 city_code_3 city_code_4 city_code_5
## 1           0           0           0           1           0
## 2           0           0           0           0           0
## 3           0           0           1           0           0
## 4           0           1           0           0           0
## 5           0           0           0           0           0
## 6           0           0           0           0           1
##   city_code_5.66666666666667 city_code_6 city_code_7 city_code_8 city_code_9
## 1                          0           0           0           0           0
## 2                          0           0           0           0           0
## 3                          0           0           0           0           0
## 4                          0           0           0           0           0
## 5                          0           0           0           0           1
```

```
## 6                          0           0           0           0           0
```

```
# Create dummy variables needed for the k-nearest neighbors
cust.desc.dummies <- cbind(as.data.frame(cust.desc), dms1, dms2)
head(cust.desc.dummies)
```

```
##   cust_id  n      age city_code Gender    amount Gender_F Gender_M city_code_1
## 1  266783  4 38.25000         4      M  8952.710        0        1           0
## 2  266784  3 21.00000        10      F  5694.065        1        0           0
## 3  266785  7 27.00000         3      F 28442.700        1        0           0
## 4  266788  4 39.75000         2      F  6092.970        1        0           0
## 5  266794 11 40.81818         9      F 28117.830        1        0           0
## 6  266799  3 43.00000         5      F  8778.120        1        0           0
##   city_code_2 city_code_3 city_code_4 city_code_5 city_code_5.66666666666667
## 1           0           0           1           0                          0
## 2           0           0           0           0                          0
## 3           0           1           0           0                          0
## 4           1           0           0           0                          0
## 5           0           0           0           0                          0
## 6           0           0           0           1                          0
##   city_code_6 city_code_7 city_code_8 city_code_9
## 1           0           0           0           0
## 2           0           0           0           0
## 3           0           0           0           0
## 4           0           0           0           0
## 5           0           0           0           1
## 6           0           0           0           0
```

Now, we need to re-scale the data to make sure that the age and the number of times going to the shop is re-scaled between 0 and 1

```
# Scale the data for the K-Nearest Neighbors algorithm
cust.desc.dummies$age.s <- rescale(cust.desc.dummies$age)
cust.desc.dummies$n.s   <- rescale(cust.desc.dummies$n)
```

For k-nearest neighbors we are going to create a train set, a validation set and a test set. We will use 60% in our test set in this case. Let's use the caret package to do this:

```
# We try now k-nearest neighbors
tr.id <- createDataPartition(cust.desc.dummies$amount, p = 0.6, list = F)
tr <- cust.desc.dummies[tr.id, ]
temp <- cust.desc.dummies[-tr.id, ]

v.id <- createDataPartition(temp$amount, p = 0.5, list = F)
val  <- temp[v.id, ]
test <- temp[-v.id, ]
```

Since K-nearest neighbors is an algorithm dependent on the value of k (the number of neighbors selected), we need to cover a range with that. Therefore, it is better to write two functions to handle this. We select columns 7 to 20 because they are the ones that include the Gender in dummy variables and also the city code. The results of the RMSE for different values of k is show in 11.

```
# Function to calculate the RMSE with a given K
func.knn.reg <- function(tr_predictor, val_predictor,
                         tr_target, val_target, k){
  library(FNN)
  res <- knn.reg(tr_predictor, val_predictor,
                 tr_target, k, algorithm = "brute")
```

```
  rmserror <- RMSE(res$pred, val_target)
  # cat(paste("RMSE for k = ", toString(k), ": ", rmserror, "\n", sep = ""))
  rmserror
}

# Function to calculate the RMSE with a given range of k's
func.knn.reg.multi <- function(tr_predictor, val_predictor,
                               tr_target, val_target,
                               start_k, end_k){
  rms_errors <- vector()
  for (k in start_k:end_k){
    rms_error <- func.knn.reg(tr_predictor, val_predictor,
                              tr_target, val_target, k)
    rms_errors <- c(rms_errors, rms_error)
  }
  # Plot the error
  plot(rms_errors, type ='o', xlab = "k", ylab = "RMSE")
  return(rms_errors)
}

# Run the k-nearest neighbors for a big range
rmse.knn.gencity.val <- func.knn.reg.multi(tr[,7:20], val[,7:20],
                                           tr$amount, val$amount, 1, 30)
```

The minimum RMSE using the k-neighbors with these predictors is 4299.7161039. We will store it in the table using this code.

```
# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "K-Nearest Neighbors - City and Gender Predictors
                                 (Validation Set)",
                                 RMSE = min(rmse.knn.gencity.val)))
```

Let's do the same for the validation set to confirm that there is no overfitting. The results of the RMSE for different values of k is show in 12.

```
# Run the k-nearest neighbors for a big range - for the test set
rmse.knn.gencity.test <- func.knn.reg.multi(tr[,7:20], val[,7:20],
                                           tr$amount, val$amount, 1, 30)
```

```
# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "K-Nearest Neighbors - City and Gender Predictors
                                 (Test Set)",
                                 RMSE = min(rmse.knn.gencity.test)))
```

The validation and the test set provide similar RMSE, which indicates no overfitting. However, the RMSE is very high, compared to the mean amount spent of 9890.3646914 dollars. Therefore, we should also consider the addition of some alternative predictors, such as the product category, which indicated some bias. We need to add the product category (the 6 major ones) and add them also as dummy variables. Let's prepare the data first following this code:

```
# Try to perform k-nearest neighbors adding the category and the year of the product
transactions.grouped <- transactions.ord %>% filter(total_amt > 0) %>%
       group_by(cust_id) %>% select(cust_id, tran_date, prod_cat, total_amt)
```

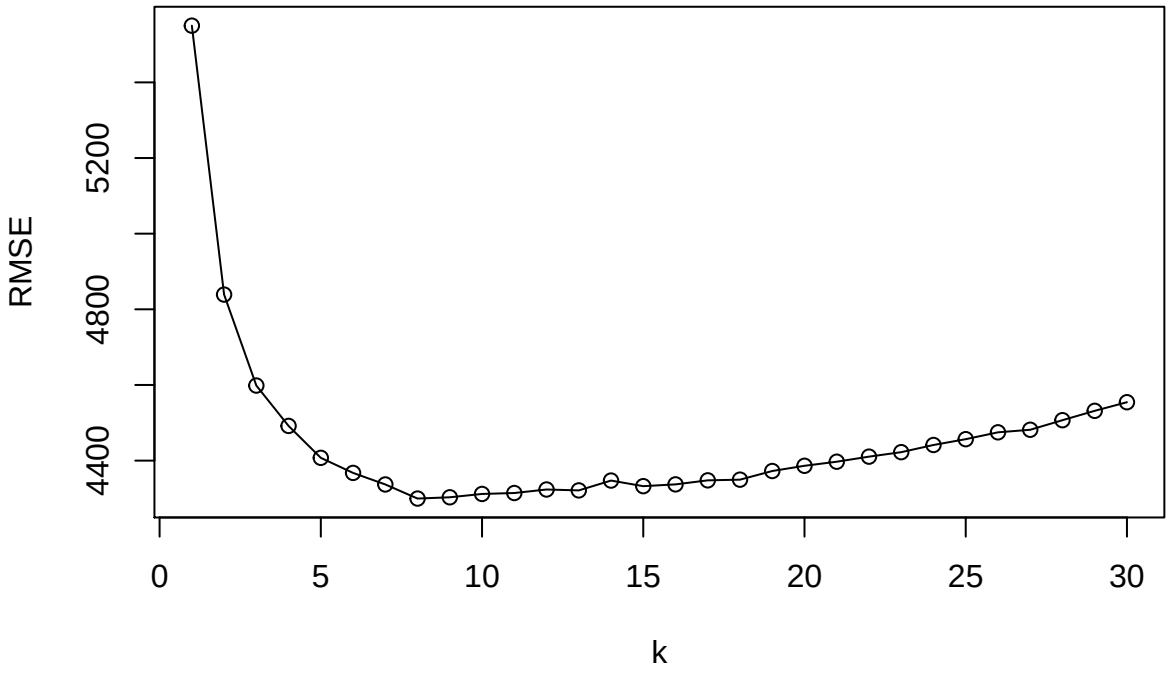


Figure 11: Result of RMSE for the k-nearest neighbors algorithm *on the validation set* using the gender and the city as predictors.

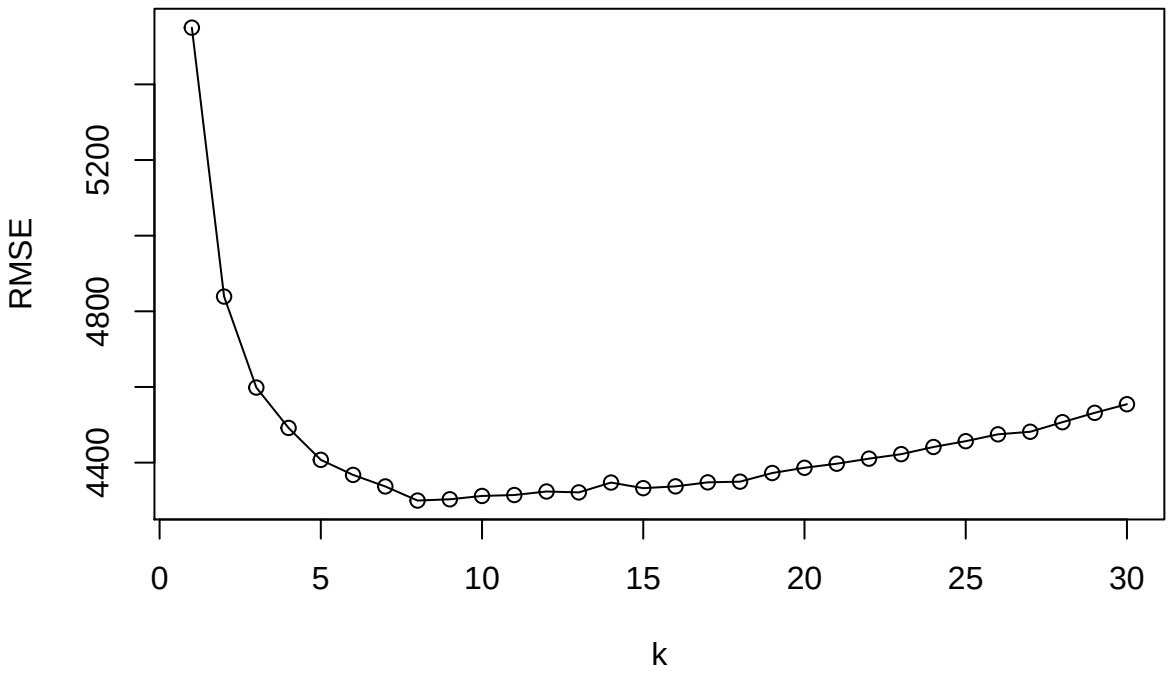


Figure 12: Result of RMSE for the k-nearest neighbors algorithm *on the test set* using the gender and the city as predictors.

```r
# Create dummies for the product category
dummy.transactions <- dummy.data.frame(as.data.frame(transactions.grouped),
                                       names = "prod_cat", sep = " ")
colnames(dummy.transactions) <- str_remove(colnames(dummy.transactions),
                                           "prod_cat ")
colnames(dummy.transactions) <- str_replace_all(colnames(dummy.transactions), "-", ".")
colnames(dummy.transactions) <- str_replace_all(colnames(dummy.transactions), " ", ".")

head(dummy.transactions)
```

```
##   cust_id  tran_date Bags Books Clothing Electronics Footwear Home.and.kitchen
## 1  270384 2014-02-20    0     0        0           1        0                 0
## 2  267750 2014-02-20    0     0        0           0        0                 1
## 3  275023 2014-02-20    0     1        0           0        0                 0
## 4  269345 2014-02-20    0     1        0           0        0                 0
## 5  268799 2014-02-20    0     1        0           0        0                 0
## 6  270787 2014-02-20    0     1        0           0        0                 0
##   total_amt
## 1  8270.925
## 2  4508.400
## 3  2594.540
## 4  4153.695
## 5  2033.200
## 6  3226.600
```

```r
# Group the dummy variables created by customer id
dummy.trans.grouped <- dummy.transactions %>%  group_by(cust_id) %>%
                            summarise(n = n(), bags = sum(Bags),
                                      Books = sum(Books), Clothing = sum(Clothing),
                                      Electronics = sum(Electronics),
                                      Footwear = sum(Footwear),
                                      Home = sum(Home.and.kitchen))


# Add to the customer description data frame the dummy variables for the product category
final.cust <- cust.desc %>% left_join(dummy.trans.grouped, by = c("cust_id", "n"))
head(final.cust)
```

```
## # A tibble: 6 x 12
## # Groups:   cust_id [6]
##   cust_id     n   age city_code Gender amount  bags Books Clothing Electronics
##     <int> <int> <dbl>     <dbl> <fct>   <dbl> <int> <int>    <int>       <int>
## 1  266783     4  38.2         4 M       8953.     0     1        2           0
## 2  266784     3  21          10 F       5694.     0     2        0           1
## 3  266785     7  27           3 F      28443.     1     1        0           0
## 4  266788     4  39.8         2 F       6093.     1     1        0           0
## 5  266794    11  40.8         9 F      28118.     2     2        2           3
## 6  266799     3  43           5 F       8778.     0     1        0           1
## # ... with 2 more variables: Footwear <int>, Home <int>
```

Once the new data is prepared as final customer data ("final.cust"), we need to create the dummy variables for k-nearest neighbors and scale the other numerical variables from 0 to 1.

```r
## Let's prepare the data again to apply K-nearest neighbors
dms3 <- dummy("Gender", as.data.frame(cust.desc), sep = "_")
dms4 <- dummy("city_code", as.data.frame(cust.desc), sep = "_")
```

```
dms4 <- dms4[,-11]

# Merge new dummy variables
final.cust.dms <- cbind(as.data.frame(final.cust), dms3, dms4)

# Scale from 0 to 1 the other numerical variables
final.cust.dms$age.s <- rescale(final.cust.dms$age)
final.cust.dms$n.s   <- rescale(final.cust.dms$n)
```

Now we can attempt again k-nearest neighbors with our new predictors. We use the same distribution for validation, train and test set as before. The RMSE for the validation set is shown in Figure **??**:

```
# We try now k-nearest neighbors again
tr.id <- createDataPartition(final.cust.dms$amount, p = 0.6, list = F)
tr <- final.cust.dms[tr.id, ]
temp <- final.cust.dms[-tr.id, ]

v.id <- createDataPartition(temp$amount, p = 0.5, list = F)
val  <- temp[v.id, ]
test <- temp[-v.id, ]

# Try the K-Nearest Neighbors now
rmse.knn.prod.val <- func.knn.reg.multi(tr[,c(7:12)], val[,c(7:12)],
                                        tr$amount, val$amount, 1, 30)
```

```
# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "K-Nearest Neighbors - Product as Predictors
                                 (Validation Set)",
                                 RMSE = min(rmse.knn.prod.val)))
```

It does appear to substantially from using the city and the gender. Let's try to attempt it on the test set to ensure that we are not overfitting the model. The RMSE for the test set is shown in Figure **??**:

```
rmse.knn.prod.test <- func.knn.reg.multi(tr[,c(7:12)], test[,c(7:12)],
                                         tr$amount, test$amount, 1, 30)
```

```
# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "K-Nearest Neighbors - Product as Predictors
                                 (Test Set)",
                                 RMSE = min(rmse.knn.prod.test)))
```

Looks like our K-nearest neighbors works much better when only considering the type of product and does not overfit.

The RMSE is on the test set is 4055.1589243, which is still somehow high considering what is the average amount. However, to see also how good might be the model, it is interesting to compare it with the Linear Regression

1.4.2 Linear Regression

Let's try the simplest model to compare how good or not is our k-nearest neighbors algorithm tested. For the linear method we will use a 70% train set and a 30% test set, since we will not use a validation one. Since the data does not require to be scaled, we will come back to the *final.cust* variable which does not contain dummy variables nor scaled ones,

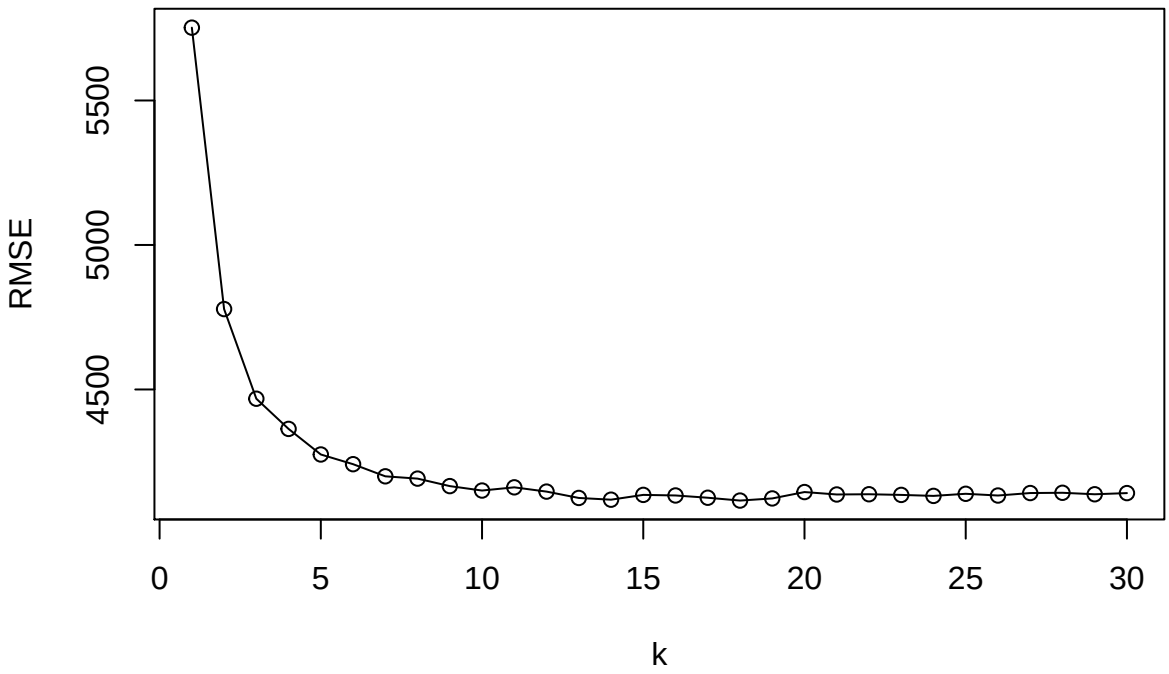


Figure 13: Result of RMSE for the k-nearest neighbors algorithm *on the validation set* using the product category as predictor.

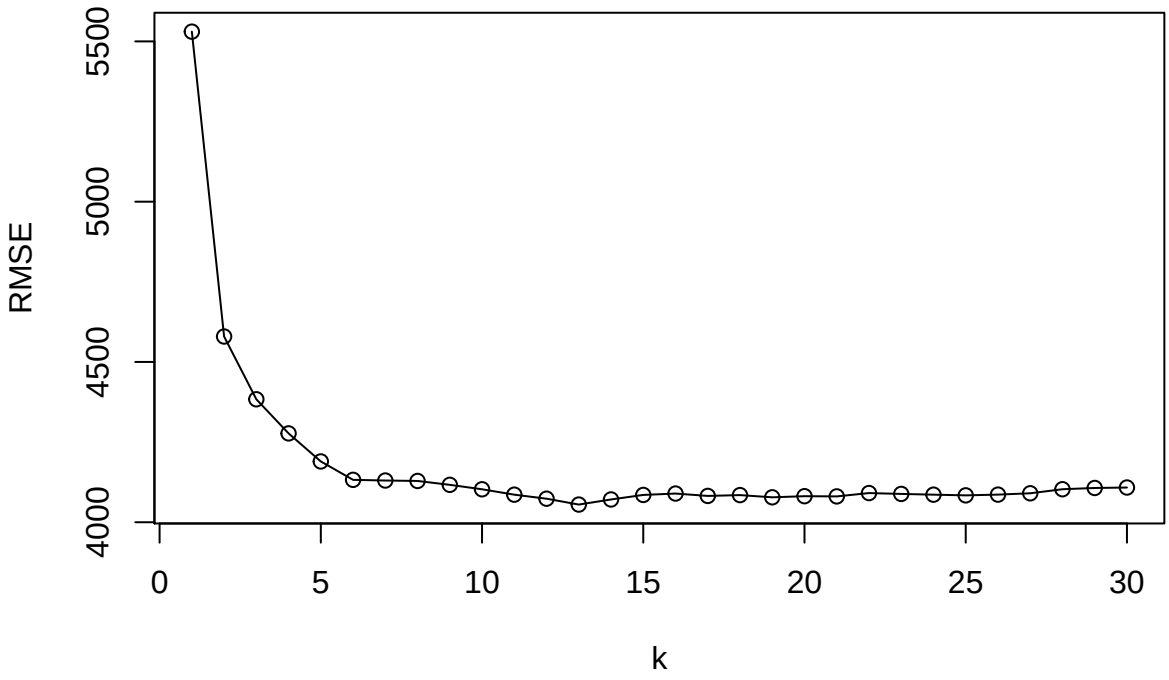


Figure 14: Result of RMSE for the k-nearest neighbors algorithm *on the test set* using the product category as predictor.

```r
# Create a train set
tr.id <- createDataPartition(final.cust$amount, p = 0.7, list = F)

# Ignore the first column because it is of no use
mod_lm <- lm(amount ~., data = final.cust[tr.id,-c(1,2)])
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```r
# see the linear model
summary(mod_lm)
```

```
##
## Call:
## lm(formula = amount ~ ., data = final.cust[tr.id, -c(1, 2)])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13150.5  -2545.3   -396.6   2292.8  17479.2
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -177.452    346.569  -0.512    0.609
## age            1.382      9.337   0.148    0.882
## city_code     24.786     21.733   1.140    0.254
## GenderM      -38.528    124.299  -0.310    0.757
## bags        2787.292    107.689  25.883   <2e-16 ***
## Books       2636.926     61.090  43.165   <2e-16 ***
## Clothing    2702.162     88.030  30.696   <2e-16 ***
## Electronics 2541.307     70.150  36.227   <2e-16 ***
## Footwear    2741.762     87.602  31.298   <2e-16 ***
## Home        2422.855     76.741  31.572   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3851 on 3846 degrees of freedom
## Multiple R-squared:  0.6105, Adjusted R-squared:  0.6096
## F-statistic: 669.8 on 9 and 3846 DF,  p-value: < 2.2e-16
```

```r
rmse.linear <- sqrt(mean(as.matrix(mod_lm$fitted.values -
                                   final.cust[-tr.id, "amount"])^2))


# Collect the results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Linear Mode (Test Set)",
                                 RMSE   = rmse.linear))
```

The summary of the data reveals the importance of the product category in our lineal model, as seen during the exploratory analysis. The RMSE is on the test set is 7658.0755371, which is substantially higher (more than 75%) to the one obtained with the k-nearest neighbors.

Let's explore other algorithms to see how much we can improve our prediction.

1.4.3 Random Trees

Let's try the random trees algorithm to see if it can be better or not than the k-nearest neighbors. From here onward, we will test the fitting on the model on the test set only. We will use again 70% for the test set and 30% for the test set. Let's plot the random trees in **??**:

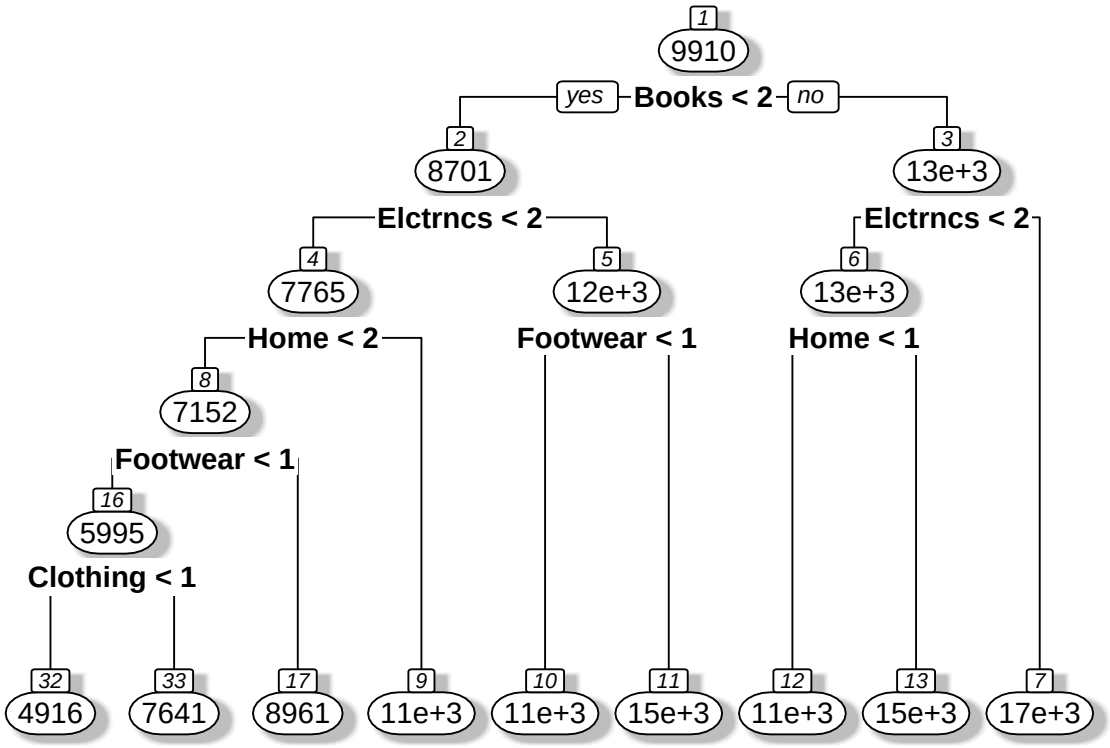
```
### Random Forest or Random Trees for regression....
library(rpart)
library(rpart.plot)

tr.id <- createDataPartition(final.cust$amount, p = 0.7, list = F)

store.fit <- rpart(amount ~., data = final.cust[tr.id,-c(1,2)])
store.fit
```

```
## n= 3856
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 3856 147103300000  9910.167
##    2) Books< 1.5 2883  93267990000  8701.147
##      4) Electronics< 1.5 2310  61068960000  7764.954
##        8) Home< 1.5 1962  46018670000  7151.505
##         16) Footwear< 0.5 1197  21596070000  5995.017
##           32) Clothing< 0.5 723   8945343000  4915.960 *
##           33) Clothing>=0.5 474  10524820000  7640.921 *
##         17) Footwear>=0.5 765  20316640000  8961.069 *
##        9) Home>=1.5 348  10149260000 11223.530 *
##      5) Electronics>=1.5 573  22012350000 12475.330
##       10) Footwear< 0.5 345  10904630000 10946.620 *
##       11) Footwear>=0.5 228   9081481000 14788.510 *
##    3) Books>=1.5 973  37134540000 13492.500
##      6) Electronics< 1.5 785  28349380000 12655.360
##       12) Home< 0.5 404  11678570000 10892.980 *
##       13) Home>=0.5 381  14085410000 14524.140 *
##      7) Electronics>=1.5 188   5937981000 16987.980 *
```

```
# Plot the random trees fit with the rpart plot
prp(store.fit, type = 2, nn = T,
    fallen.leaves = T, faclen = 4,
    varlen = 8, shadow.col ="gray")
```



Once we see the tree plot, we can decide to prune the tree, in order to get less variables involved.

```
# The representation gives 10 trees as the adequate number
plotcp(store.fit)
```

Once we have found the threshold value, we can do the fitting with the trees pruned, for a simplified model.

```
# See the cp table to get the first cp value below the cut-off point
store.fit$cptable
```

```
##           CP nsplit rel error    xerror       xstd
## 1 0.11353082      0 1.0000000 1.0002889 0.02675948
## 2 0.06924852      1 0.8864692 0.8872412 0.02506537
## 3 0.03331695      2 0.8172207 0.8182088 0.02300259
## 4 0.02791204      3 0.7839037 0.7928563 0.02249738
## 5 0.01935496      4 0.7559917 0.7775301 0.02221278
## 6 0.01757544      5 0.7366367 0.7625153 0.02186324
## 7 0.01445176      6 0.7190613 0.7440515 0.02175832
## 8 0.01377424      7 0.7046095 0.7280231 0.02081519
## 9 0.01000000      8 0.6908353 0.7057192 0.02021402
```

```
# Use the model of random trees with less branches for the new prediction
# Select tree below that value (10 in this case)
store.fitpruned <- prune(store.fit, cp = 0.01139806)

# Plot the new random trees pruned
prp(store.fitpruned, type = 2, nn = T,
    fallen.leaves = T, faclen = 4,
    varlen = 8, shadow.col ="gray")
```

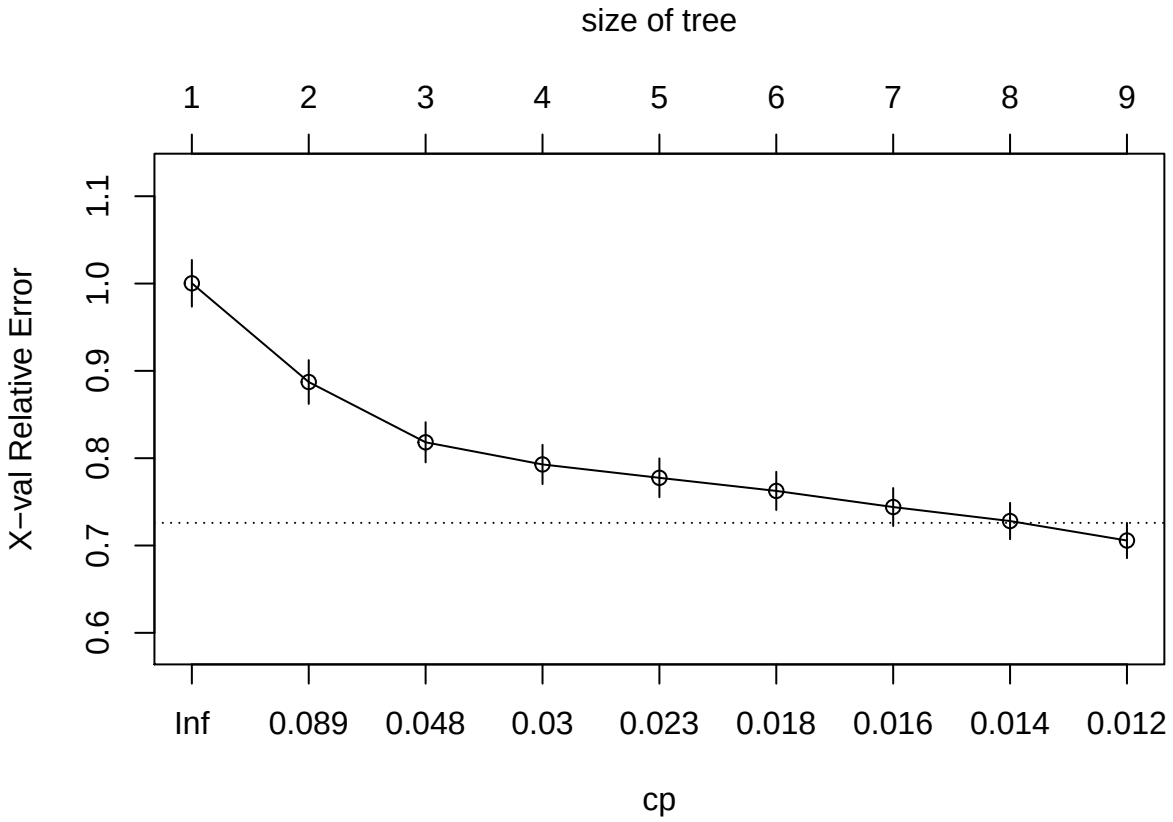
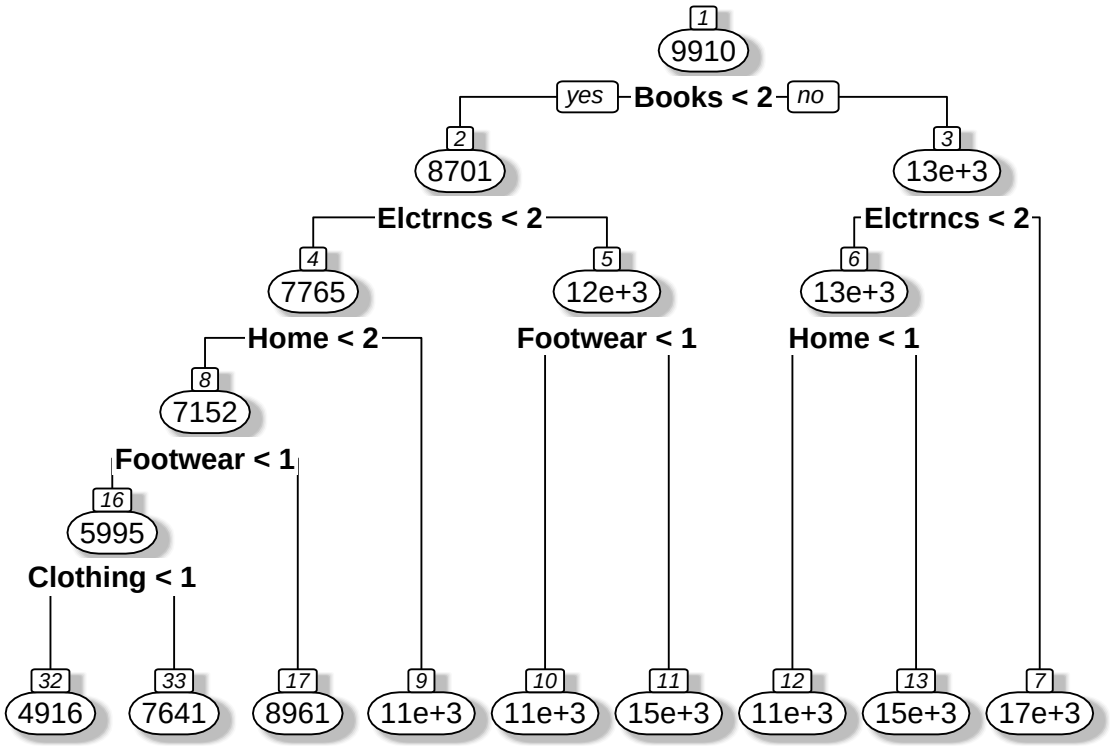


Figure 15: Cutoff point of the random trees



Let's collect both predictions, for the non-pruned and the pruned random trees.

```
# Obtain the predictions of the original random tree model
preds <- predict(store.fit, final.cust[-tr.id,-c(1,2)])
rmse.trees <- sqrt(mean(as.matrix(preds - final.cust[-tr.id, "amount"])^2))
rmse.trees
```

```
## [1] 5207.116
```

```
# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Random Trees  (Test Set)",
                                 RMSE = rmse.trees))

# Obtain the predictions of the pruned random tree model
preds <- predict(store.fitpruned, final.cust[-tr.id,-c(1,2)])
rmse.trees.pruned <- sqrt(mean(as.matrix(preds - final.cust[-tr.id, "amount"])^2))

# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Random Trees Pruned (Test Set)",
                                 RMSE = rmse.trees.pruned))
```

The random trees seem to yield worse results than K-Nearest Neighbors. However, There are two libraries that can enhance the predictions from random trees. They are the **ipred** and the **gbm** for bagging and boosting. Let's see the predictions obtained by using random trees from these two methods.

```
# Let's use some techniques of bagging
library(ipred)
```

```
# Fit the model with random trees with bagging
bagging.fit <- bagging(amount ~., data = final.cust[tr.id,-c(1,2)])
prediction.t <- predict(bagging.fit,  final.cust[-tr.id,-c(1,2)])

rmse.bagging <- sqrt(mean(as.matrix(prediction.t - final.cust[-tr.id, "amount"])^2))

# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Random Trees - Bagging (Test Set)",
                                 RMSE = rmse.bagging))
```

With Bagging the RMSE becomes smaller than using the random trees alone. Let's use some techniques of boosting.

```
library(gbm)

# Fit the model with random trees with boosting
gbm.fit <- gbm(amount ~., data = final.cust[tr.id,-c(1,2)], distribution = "gaussian")

# Obtain the prediction values over the test set
prediction.gbm <- predict(gbm.fit,  final.cust[-tr.id,-c(1,2)])
#Get the rmse on the test set
rmse.gbm <- sqrt(mean(as.matrix(prediction.gbm - final.cust[-tr.id, "amount"])^2))

# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Random Trees - Boosting (Test Set)",
                                 RMSE = rmse.gbm))
```

With Boosting, it becomes even better and quite close to the k-nearest neighbors result.

1.4.4 Random Forests

Random trees by itself are okay, but they seem to be working well when using the bagging and the boosting methods. Let's try to use now a random forest. However, when using random forest, the previously data generated (final.cust) does not have sufficient predictors for the model to be executed.

Therefore, we revert into using the cust.desc.dummies data frame for the random forest model.

```
mod.rf <- randomForest(x = cust.desc.dummies[tr.id,-c(1,2,6)], y = cust.desc.dummies[tr.id, 6],
                       ntree = 1000,
                       xtest = cust.desc.dummies[-tr.id,-c(1,2,6)],
                       ytest = cust.desc.dummies[-tr.id, "amount"],
                       importance = T, keep.forest = T)


# Obtain the prediction values over the test set
prediction.rf <- predict(mod.rf, cust.desc.dummies[-tr.id,-c(1,2,6)])

#Get the RMSE on the test set
rmse.rf <- sqrt(mean(as.matrix(prediction.rf - cust.desc.dummies[-tr.id, "amount"])^2))

# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Random Forest (Test Set)",
```

```
                                    RMSE = rmse.rf))
```

The random forest appears to be able to slightly improve the prediction of the k-nearest neighbors with products as predictors

1.4.5 Neural Network

The last method to use is the use neural networks. Neural networks are a black box in terms of prediction. We know what it is going in and what it is going out, but not exactly how the model is working in between. We will use a source link from *fawda123* to visualize the neural network.

```
library(nnet)
library(devtools)

# Get the maximum amount value to scale the neural network
max_amount <- max(final.cust$amount)

# Perform the fitting of the neural network
fit_nn <-nnet(amount/max_amount ~., data = final.cust[tr.id, -c(1,2)],
              size = 6, decay = 0.1,
              maxit = 1000, linout = T)
```

```
## # weights:  73
## initial  value 1531.071339
## iter  10 value 82.917292
## iter  20 value 66.322812
## iter  30 value 48.239163
## iter  40 value 35.900212
## iter  50 value 29.733901
## iter  60 value 29.373280
## iter  70 value 29.050828
## iter  80 value 28.825914
## iter  90 value 28.740376
## iter 100 value 28.715281
## iter 110 value 28.684810
## iter 120 value 28.661045
## iter 130 value 28.646758
## iter 140 value 28.635645
## iter 150 value 28.616013
## iter 160 value 28.581935
## iter 170 value 28.529361
## iter 180 value 28.490324
## iter 190 value 28.469823
## iter 200 value 28.450298
## iter 210 value 28.441682
## iter 220 value 28.417890
## iter 230 value 28.374638
## iter 240 value 28.322117
## iter 250 value 28.293270
## iter 260 value 28.275461
## iter 270 value 28.265459
## iter 280 value 28.258886
## iter 290 value 28.256741
## iter 300 value 28.255615
## iter 310 value 28.254813
```

```
## iter 320 value 28.254508
## iter 330 value 28.254364
## iter 340 value 28.253970
## final  value 28.253556
## converged
```

```
# Download source package to plot neural network
source_url("https://gist.githubusercontent.com/fawda123/7471137/raw/f30f338ecf143c089af1b6a731edcabd0b1
```

```
#Plot the neural network
plot(fit_nn, max.sp = T)
```

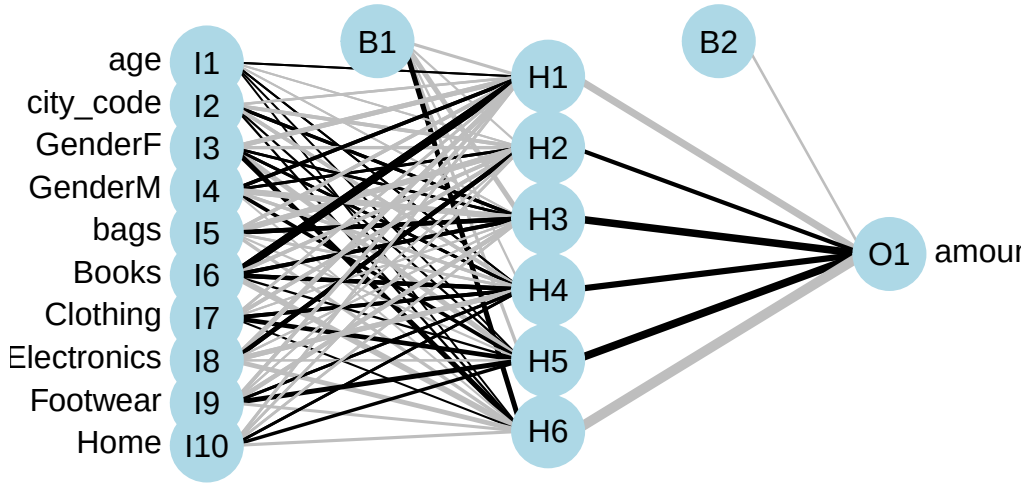


Figure 16: Neural network plot to predict the amount spent in the retail store

```
# Compute RMSE over the training set
rmse.nn.tr <- sqrt(mean(as.matrix(fit_nn$fitted.values*max_amount -
                                  final.cust[tr.id, "amount"])^2))


# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Neural Network (Train Set)",
                                 RMSE = rmse.nn.tr))

# Obtain the predictions over the test set
```

```r
pred <- predict(fit_nn, final.cust[-tr.id,-c(1,2)])

# Compute RMSE over the test set
rmse.nn.test <- sqrt(mean(as.matrix(pred*max_amount -
                                    final.cust[-tr.id, "amount"])^2, na.rm = T))

# Collect RMSE results
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Neural Network (Test Set)",
                                 RMSE = rmse.nn.test))
```

The result provided by the neural network is the best we already saw from any product.

1.5 Results

Table 1 contains the results of the RMSE on the train, the validation, and on the test set (when applicable):

```
knitr::kable(rmse_results,
             caption = "RMSE results for the different methods",
             digits = 1, align = c('l','c')) %>%
             kableExtra::kable_styling(latex_options = "hold_position")
```

Table 1: RMSE results for the different methods

Method	RMSE
K-Nearest Neighbors - City and Gender Predictors (Validation Set)	4299.7
K-Nearest Neighbors - City and Gender Predictors (Test Set)	4299.7
K-Nearest Neighbors - Product as Predictors (Validation Set)	4115.5
K-Nearest Neighbors - Product as Predictors (Test Set)	4055.2
Linear Mode (Test Set)	7658.1
Random Trees (Test Set)	5207.1
Random Trees Pruned (Test Set)	5207.1
Random Trees - Bagging (Test Set)	4851.3
Random Trees - Boosting (Test Set)	4184.6
Random Forest (Test Set)	3992.9
Neural Network (Train Set)	3816.7
Neural Network (Test Set)	3861.5

Table 1 shows that the best method to predict the results is the Neural Network (Train Set). However, algorithms such a k-nearest neighbors with k = X or the random trees with the bagging appear also as a good options.

Neural networks are kind of black boxes that do not allow the user to know what is really going inside them. For that reason, random trees might be preferred as they appear to be more transparent in the decision.

The RMSE seems somehow high for the target proposed. This might be due also to the quality of the data that seems already biased. No seasonality pattern is observed and not a significant difference between sex in terms of amount of money spent is observed. It is possible however that a better transformation of the data in a cluster could have been performed. In this case, the methods used for clustering (e.g. k-means) the customers ended up not giving any result and therefore, they were not shown.

1.6 Conclusions

We have used a dataset of an online retail store. The data is somehow limited as it only comprises 2011-2014 and there is a narrow range of ages represented in this dataset (20-45 years old). The classification methods applied in order to cluster the users in some groups failed and therefore, we went directly to train the model in order to minimize the RMSE.

The linear regression proved to be completely off the target. However, the use of other algorithms such as K-nearest neighbors, the random forests (particularly with the bagging) or the neural network managed to get much more accurate predictions on the amount a given customer is going to spend in the store.

1.6.1 Limitations and future considerations

It has been a very challenging project all along, but the learning process was satisfactory. Personally, I would benefit of a little bit more time to finish the project as I started wrapping everything up too late (human nature). Some formalities in the final report could not be addressed due to issues with RMarkdown (e.g. numbering of captions disappearing).

The work on this data would benefit from the following approaches: * Segmentation of users into clusters from a certain pattern * Transactions that include return could be added, and find a ratio between items purchased and returned by the customer * The predictors could be used in all the categories instead of using only the main ones. * The amount could be calculated per transaction, since the n has a big impact on the results.