Buying prediction through Machine Learning

Data Science assignment
Harvard Data Science

By Michel Brok
July 2020

Management Summary

Context
For companies that are relying on digital marketing activities to create awareness and consideration but not selling their goods online, it is much more difficult to understand the impact of online to offline (ROPO-effect) purchase behaviour. But the internet does influence the 'path of purchase' with consumers that are seeking for information, also directly from manufacturer and brand websites.

To get a better understanding of the 'path to purchase', this company that is subject of this assignment, implemented a survey on their website asking visitors if they are buying their products and what the most important reasons are. There is a connection being made with the Google Analytics software that gives insights in website behaviour. The next step is to understand what drives buying behaviour in order to optimise marketing activities.

Problem definition
The main question is: how to predict buying behaviour? The dataset being used for this assignment contains survey information together with some relevant website behaviour data.

Methodology

To answer the questions, a Machine Learning model will be created, trying to predict buying behaviour and answering the main questions of this assignment.

The steps being taking:

1: Importing the dataset
2: Exploratory Data Analysis (EDA)
3: Data preparation
4: Training the model
5: Evaluate model performance

Conclusions

The main question of this assignment is, **how can we predict buying behaviour?** To answer that question a 'Exploratory Data Analysis' has being done to get already a first view on the features and their chances to be the most important predictor. Already from this analysis, the 'NPS' score seemed to be the best candidate. After testing different models and selecting the right model, which was more difficult as the dataset is imbalanced, the 'xgbDart' model seems to best performing model with an accuracy of 0.63 and a Kappa value of 0.05. Also 'stacking' was being used to see if the performance could increase with the use of different models together. But based on the ‘ROC-curve’ comparing the two models, the ‘xgbDart’ was still the best performing model.

In the ‘variable importance plot’ is very clearly to see that ‘NPS’ is the most important variable which has a linear marginal effect on the outcome. The other variables seemed to be less effective after further analysis. The recommendation here to first start understanding how to improve the NPS-scores and using Machine Learning algorithm understanding the effects. Further recommendation would be to increase the amount of data and ensuring data quality to also see the effect of the other potential predictors of buying behaviour.

Table of contents

1 Introduction

In this assignment the main questions is: how can we predict buying behaviour with the features in the dataset. This is a binary classification problem (yes or no) with multiple features and therefore a supervised learning model. The file being used for this assignment is a csv-document with data collected from the period March till July 2020. The data collection has been done via Google Analytics (online webanalytics software) in combination with Hotjar (online survey software). Via the Hotjar tool, visitors of the website are mainly being asked if they are using the product. The reasons for doing this is that the digital marketing strategy is towards engagement and not selling product. This data is being combined with website behaviour data that is being collected with the Google Analytics software.

This data is an export coming directly from the system where every row contains information about each respondent. The 'userid' variable contains a number that can't be related to a person and is inline with the privacy regulations.

2 Importing the dataset

In this section the libraries are being loaded which are needed for analysis and model building. The dataset will be imported and a check will be made to understand the data and if the import went correctly.

2.1 Loading the libraries

Loading the libraries first to make us of the functionalities.

```r
library(plyr)
library(dplyr)
library(dplyr)
library(ggplot2)
library(PerformanceAnalytics)
library(ggthemes)
library(corrplot)
library(car)
library(caret)
library(caretEnsemble)
library(foreach)
library(doParallel)
library(chron)
library(hms)
library(tidyr)
library(tidyverse)
library(mice)
library(lubridate)
library(randomForest)
library(gbm)
library(effects)
library(pdp)
library(cvms)
library(broom)
library(modelplotr)
library(visreg)
library(margins)
library(data.table)
```

2.2 Importing the dataset

The dataset is stored on the local computer and being imported via the read.csv function. The "na.string" option is used to convert the factor variables as much as possible to numeric vectors.

```
dataset <- read.csv("~/project/Own assignment/
dataset_assignment_v02.csv", sep=";", na.strings = "na")
str(dataset)

## 'data.frame':    970 obs. of  11 variables:
##  $ userid                  : Factor w/ 970 levels
"006b1483","00a5c05c",..: 133 534 477 178 3 696 207 471 341 751 ...
##  $ nps                     : Factor w/ 12 levels "(not set)","0",..:
4 12 10 11 1 4 12 1 9 9 ...
##  $ giving_product          : Factor w/ 2 levels "No","Yes": 2 2 1 1 2
2 2 2 2 1 ...
##  $ page_type               : Factor w/ 3 levels "Blog","Brand",..: 1
1 1 1 1 1 1 1 1 1 ...
##  $ source.used             : Factor w/ 14 levels "bingorganic",..: 2
4 10 4 1 9 9 10 10 4 ...
##  $ city                    : Factor w/ 327 levels "'s-Gravendeel",..:
245 91 235 16 16 138 16 188 266 114 ...
##  $ avg_session_dur         : Factor w/ 531 levels
"00:00:00","00:00:02",..: 50 252 524 518 516 515 512 506 493 472 ...
##  $ avg_session_quality     : num  2.9 40.4 48.2 52.2 34.1 41.1 22.7
```

```
60.5 6.3 57.5 ...
##  $ avg_time_page          : Factor w/ 182 levels
"00:00:00","00:00:01",..: 38 77 29 6 178 8 15 6 4 19 ...
##  $ registration_completed: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pages_session          : num  1 1 4 2.5 1 1.5 1.5 1 1 3 ...
```

```
class(dataset)
```

```
## [1] "data.frame"
```

```
n_distinct(dataset)
```

```
## [1] 970
```

```
summary(dataset)
```

```
##       userid           nps      giving_product   page_type
##  006b1483:  1   (not set):382   No :210         Blog   :386
##  00a5c05c:  1   8        :165   Yes:760         Brand  :298
##  00aca954:  1   7        :155                   Product:286
##  0102d6dd:  1   10       : 84
##  01b0007b:  1   6        : 63
##  0259b967:  1   9        : 50
##  (Other) :964   (Other)  : 71
##         source.used         city     avg_session_dur
avg_session_quality
##  googlecpc    :399   Amsterdam:110   00:00:00: 42    Min.   : 0.00
##  googleorganic:374   Rotterdam: 61   00:00:03: 26    1st Qu.: 2.90
##  direct       : 92   Utrecht  : 30   00:02:32:  9    Median : 6.60
##  exm          : 36   (not set): 25   00:00:38:  8    Mean   :18.89
##  referral     : 28   The Hague: 22   00:00:04:  7    3rd Qu.:33.65
##  bingorganic  : 23   Eindhoven: 18   00:00:34:  7    Max.   :76.90
##  (Other)      : 18   (Other)  :704   (Other) :871
##   avg_time_page registration_completed pages_session
##  00:00:00: 77   Min.   :0              Min.   : 0.00
##  00:00:06: 46   1st Qu.:0              1st Qu.: 1.00
##  00:00:07: 43   Median :0              Median : 1.50
##  00:00:03: 38   Mean   :0              Mean   : 1.86
##  00:00:05: 38   3rd Qu.:0              3rd Qu.: 2.50
##  00:00:04: 37   Max.   :0               Max.   :11.67
##  (Other) :691
```

The dataset contains 970 observations with 11 variables or features. These 970 observations are unique, meaning that every row contains a new observation. The class of the data set is a data frame. Almost all variables are “factors” which is important to understand for the steps being taken exploring them and later on to manipulate for the machine learning model. The data that is collected contains different elements around website behaviour and questions if people are giving the product and what NPS (NPS stands for Net Promotor Score, a score that helps in understanding how satisfied customers are) score is.

The first step is taken with importing the dataset and exploring the first results. The next step is a more detailed exploration of the data (EDA).

3 Exploratory Data Analysis (EDA)

After the first steps of loading the libraries and important the data, a more deeper analysis will being done in this section to understand the data better and prepare for the modeling phase.

3.1 Checking for missing values

Before the analysis starts, a check will be done if values are missing with the “anyNa” and “sapply” function.

```
anyNA(dataset)

## [1] FALSE

sapply(dataset, {function(x)
  any(is.na(x))})

##                  userid                      nps
giving_product
##                   FALSE                    FALSE
FALSE
##               page_type              source.used
city
##                   FALSE                    FALSE
FALSE
##         avg_session_dur    avg_session_quality
avg_time_page
```

```
##                   FALSE                   FALSE
FALSE
## registration_completed          pages_session
##                   FALSE                   FALSE
```

In the variables no missing values are being found. The analysis will continue and starts with exploring the most important variable, the buyers data.

3.2 Features of the dataset

Before the analysis starts, what are the most important features and how do they look like? The “glimse” function will be used to get a first view on the dataset.

```
glimpse(dataset)

## Rows: 970
## Columns: 11
## $ userid                <fct> 22439367, 91a3a75b, 84bc3cb5,
3015c55f, 00aca9…
## $ nps                   <fct> 10, 9, 7, 8, (not set), 10, 9, (not
set), 6, 6…
## $ giving_product        <fct> Yes, Yes, No, No, Yes, Yes, Yes,
Yes, Yes, No,…
## $ page_type             <fct> Blog, Blog, Blog, Blog, Blog, Blog,
Blog, Blog…
## $ source.used           <fct> cpcfacebook, direct, googleorganic,
direct, bi…
## $ city                  <fct> Rotterdam, Eersel, Raalte,
Amsterdam, Amsterda…
## $ avg_session_dur       <fct> 00:00:57, 00:05:16, 00:38:25,
00:33:31, 00:32:…
## $ avg_session_quality   <dbl> 2.9, 40.4, 48.2, 52.2, 34.1, 41.1,
22.7, 60.5,…
## $ avg_time_page         <fct> 00:00:37, 00:01:21, 00:00:28,
00:00:05, 00:24:…
## $ registration_completed <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0…
## $ pages_session         <dbl> 1.00, 1.00, 4.00, 2.50, 1.00, 1.50,
1.50, 1.00…
```

Userid
This feature contains data about the userid with purpose to create a record per respondent and avoid duplicates. This feature will later not being used as it is not relevant for the goal of this assignment

NPS
The NPS-feature contains data about the NPS-score being given. NPS stands for Net Promotor Score and is being used to understand if customers would recommend the product or service to someone else. In this case, the NPS-question is if people are recommending the website to someone else. When a score is giving of 9 or 10, then people are called a “promotor”, meaning that they will probably actively will promote the product or service to someone else. People giving a 7 or 8 are called " passives, they are satisfied but not very enthusiastic and they could be vulnerable for competitive offers. Detractors are the one giving 1-6 scores, they can potentially damage the brand with negative word-of-mouth source. This feature also “not set” results meaning that respondents didn’t give a NPS-score.

Giving_product
The feature in the dataset contains information about buyers vs non-buyers. This is the target feature.

Page_type
This feature contains data about the type of pages are being visited on the website.

source_used
These are the channels that respondents used to enter the website.

City
As it says, the city that the respondents are located in.

Avg_session_duration
The session duration is being measured in Google Analytics which is web-analytics software. The sessions duration measures how long a person within a session in average is on the website. This is an important metric is it is often being used to measure the engagement of the website.

Avg_session_quality
The average sessions quality feature is a number being given by the Google Analytics software how high the quality of a user is on the website source.

Avg_time_on_page
This feature measure how long a visitors on the website is visiting a webpage.

Registration_completed
This feature containts data if a respondent registered to the e-mail-program of the website.

Pages_session
The pages per session is a Google Analytics metric that show how many pages are being visited. Also this is an important metric for consumer engagement.

With a view on the features are their data, this section continues with the analysis of the features, starting with the most important one, the buyers data.

3.2.1 Buyers vs non-buyers

More information about people that are buying or not buying, is in the "giving_product" variable. The "dplyr" functionalities with "pipe operator" is being used to get quickly the information together in one overview.

```
dataset %>%
group_by(giving_product) %>%
tally(sort = TRUE) %>%
mutate(percent = n/sum(n) * 100)
```

```
## # A tibble: 2 x 3
##   giving_product     n percent
##   <fct>          <int>   <dbl>
## 1 Yes              760    78.4
## 2 No               210    21.6
```

From the 970 unique observations, 760 people answer with "yes" on the question "are you giving the product". In this assignment we consider this as a person buying the product and ofcourse the other way around. The buyers-group is the biggest group with 78%, 22% of the people are not giving the product and we consider them as "non-buyers". As mentioned earlier, all records are unique observations so these are being handled as such.

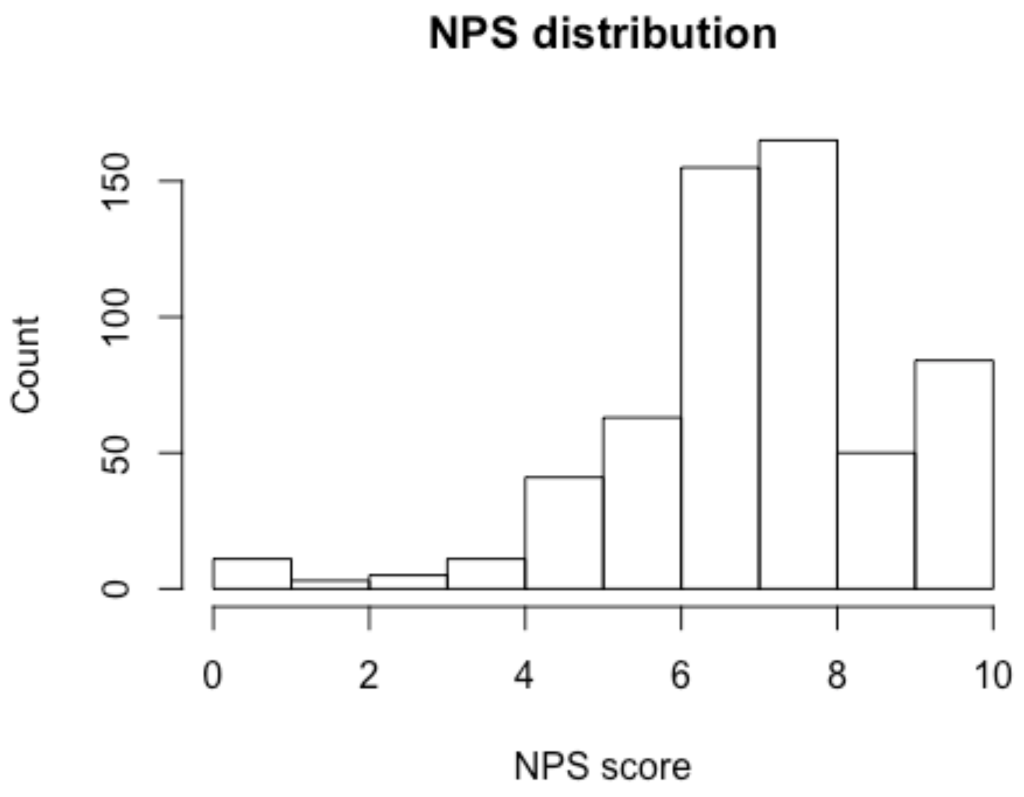
3.2.2 What is the NPS-score telling?

The NPS-scores being measures tells something about the performance of the website. In this section the analysis is being done how the NPS-scores are being distributed and if there's a relation between NPS-scores and buying behaviour.

NPS scores given
Quick view on the NPS data and the distribution of NPS-scores. First the data needs to be

converted from factor, to character to numeric to get a decent graph. The “hist” function is being used to get a plot.

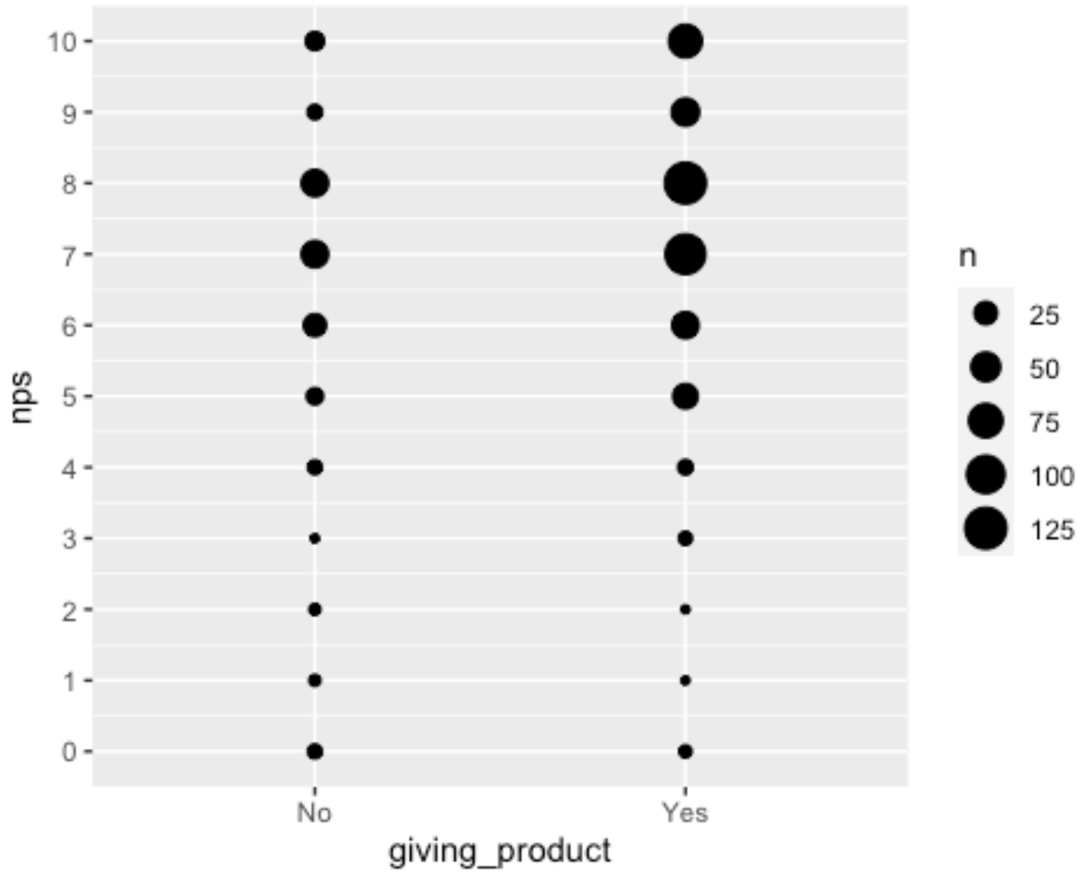
```
dataset$nps <- as.character(dataset$nps)
dataset$nps <- as.numeric(dataset$nps)

hist(dataset$nps, main= "NPS distribution", xlab = "NPS score", ylab =
"Count")
```



First view shows that most of the scores are being between 7 and 8. Also a couple of zeros are being given and tens. Looking to this distribution, there are no strange scores and outliers.

NPS scores buyers vs non-buyers
To get a view on the differences of NPS-scores per group, the ‘ggplot’-functionality is being used where the size of the point represents the count per NPS-score.

```r
dataset %>%
  select(giving_product, nps) %>%
  filter(nps >= 0) %>%
  group_by(giving_product) %>%
  count(nps) %>%
  ggplot(aes(x=giving_product, y=nps)) +
  geom_point(aes(size=n)) +
  scale_y_continuous(breaks = seq(0,10, 1),
                     limit=c(0,10))
```



In the graph above, in both of the groups the most scores are being given between 7 and 8. This was also to see in the previous graph with no distinction between buyers and non-buyers. But in this graph, more clearly is to see that the higher NPS-scores are giving by the group of buyers. Which could also has to do with the fact that the group of buyers is much bigger than the group of non-buyers (78% of buyers and 22% of non-buyers). Therefore the average per group will be calculated.

Average NPS-scores per group
Because ‘NA’s’ are being detected in the dataset, the ‘na.omit’ option is being used to filter out the NA’s to get the average of NPS-scores.

```
nps <- dataset$nps
nps <- na.omit(dataset$nps)
mean(nps, na.rm = TRUE)

## [1] 7.391156

str(nps)

##  num [1:588] 10 9 7 8 10 9 6 6 6 6 ...
##  - attr(*, "na.action")= 'omit' int [1:382] 5 8 11 14 15 16 18 23
29 31 ...

length(nps)

## [1] 588
```

The average NPS score is 7.4 for all groups with the NA’s being excluded, resulting in a smaller group of observation (588 instead of 970). To get the average scores per group, the dplyr “pipe operator” is being used again with filter this time on higher NPS-scores then 0.

```
dataset %>%
  select(giving_product, nps) %>%
  filter(nps >= 0) %>%
  group_by(giving_product) %>%
  summarize(mean_nps = mean(nps))

## # A tibble: 2 x 2
##   giving_product mean_nps
##   <fct>             <dbl>
## 1 No                 6.82
## 2 Yes                7.58
```

The average NPS-scores giving by the buyers and non-buyers shows a smaller difference, with 6.8 for non-buyers and 7.6 for buyers. The difference is not that big with 20%. For a normal customer satisfaction, scores between 7 and 8 are not that bad, but to use the power of NPS, a score of 9-10 is needed. To see the the difference between the amount of promotors for buyers and non-buyers, this group will separately being analysed.

Amount of promotors between buyers and non-buyers

The difference between buyers and non-buyers NPS scores seems to be small. In the graph where the NPS-scores per group was being visualised, was to see that the buyers group seems to give higher scores. The "promotors" (NPS 9 and higher) are being searched with the filter option (nps equal or bigger then 9).

```
promotors <-dataset %>%
  select(giving_product, nps) %>%
  filter(nps >=9) %>%
  group_by(giving_product) %>%
  tally()
promotors
```

```
## # A tibble: 2 x 2
##   giving_product     n
##   <fct>          <int>
## 1 No                19
## 2 Yes              115
```

When only looking at the promotors (NPS score 9 and 10) it seems that the buying group is much bigger, but the buying-group in general is also bigger then the non-buying group. This is clearly to see in the simply graph below, with the count of NPS-scores per group.

```
non_promotors <- dataset %>%
  select(giving_product, nps) %>%
  filter(nps >= 0) %>%
  group_by(giving_product) %>%
  tally()
non_promotors
```

```
## # A tibble: 2 x 2
##   giving_product     n
##   <fct>          <int>
## 1 No               147
## 2 Yes              441
```

To see the difference relatively, the data is being put together with the “merge” function and combines the data from the previous two tibbles, the “promotors” and “non-promotors”. Based on the total counts the percentage has been calculated.

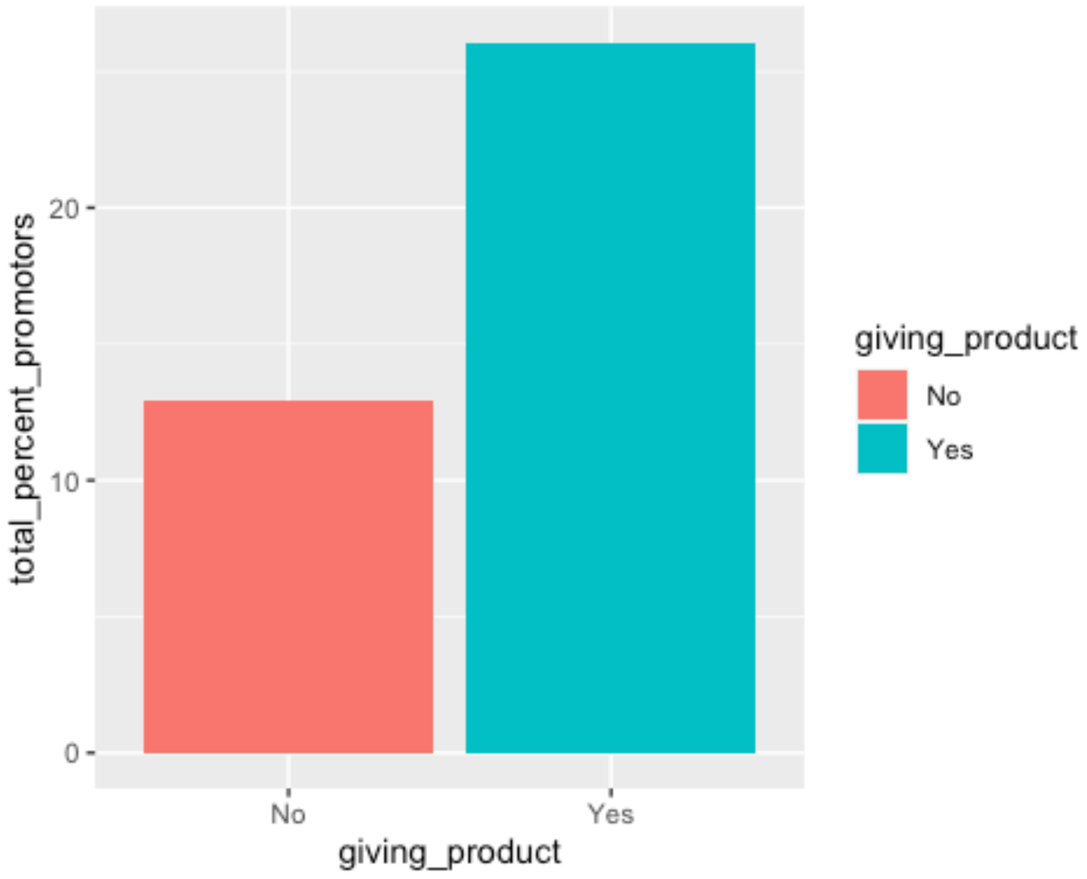
```
promotors_non_promotors <- merge(promotors, non_promotors, by =
"giving_product", all = TRUE, suffix = c(".promotors",
".non_promotors")) %>% mutate(total_percent_promotors = n.promotors /
n.non_promotors *100)
promotors_non_promotors
```

```
##   giving_product n.promotors n.non_promotors total_percent_promotors
## 1             No          19             147                12.92517
## 2            Yes         115             441                26.07710
```

Also relatively the amount of promotors under the buying-group is bigger then with the non-buyers. In the buying group, 26% are “promotors” giving a NPS-score with 9 or higher, whilst only 13% of promotors can be found in the non-buying group. Here we could say that there is a bigger difference between buyers and non-buyers when looking to NPS score.

The table with the scores are difficult to read, a graph will be made from the scores with the “ggplot” functionality. As it is a dataframe, it first need to converted to a list.

```
lp <- as.tbl(promotors_non_promotors)

## Warning: `as.tbl()` is deprecated as of dplyr 1.0.0.
## Please use `tibble::as_tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was
generated.

lp <- list(promotors_non_promotors)
names(lp) <- c("promotors_non_promotors")
blp <- rbindlist(lp, id="id", use.names = TRUE)
ggplot() +
  geom_bar(data = blp, aes(x= giving_product,
y=total_percent_promotors, fill = giving_product), stat = 'identity')
```



3.2.3 Which cities are buyers coming from?

After analysing the NPS-feature and to understand how it's impacting buyers and non-buyers, in this section the "city" feature will be analysed. First, a quick view on the "city" feature.

```
str(dataset$city)

##  Factor w/ 327 levels "'s-Gravendeel",..: 245 91 235 16 16 138 16
188 266 114 ...

head(dataset$city, n=50)
```

```
##  [1] Rotterdam       Eersel          Raalte          Amsterdam
##  [5] Amsterdam       Hilversum       Amsterdam       Menaldum
##  [9] Steenbergen     Groningen       Amsterdam       The Hague
## [13] Shinagawa City  Bleiswijk       Amsterdam       Shinjuku City
## [17] Beneden-Leeuwen Amsterdam       Amsterdam       Emmen
## [21] Rotterdam       Paramaribo      Tilburg         Ede
## [25] Nieuwegein      Schijndel       Amsterdam       Kerkrade
## [29] Haarlem         Heinkenszand    Weert           Rotterdam
## [33] Loon op Zand    Pattaya City    Utrecht         Hilversum
## [37] Best            Oldenzaal       Utrecht         Sprang-Capelle
## [41] Amsterdam       Hilversum       Venray          Zoetermeer
## [45] Bussum          Oss             Pijnacker       Bilthoven
## [49] Doetinchem      Enschede
## 327 Levels: 's-Gravendeel 's-Gravenzande ... Zwolle
```

```r
length(dataset$city)
```

```
## [1] 970
```

Inspecting the city dataset, there are 970 observations with 327 levels, meaning that there are 327 different observations. Also here is to see that there is a "(not set)" observation, which means the city was not given in. The "not set" records will converted to NA for analysis purposes. First, how much "not set" vectors are there?

```r
dataset %>%
  select(city) %>%
  mutate(as.character(city)) %>%
  filter(city == "(not set)") %>%
  count(city)
```

```
##        city  n
## 1 (not set) 25
```

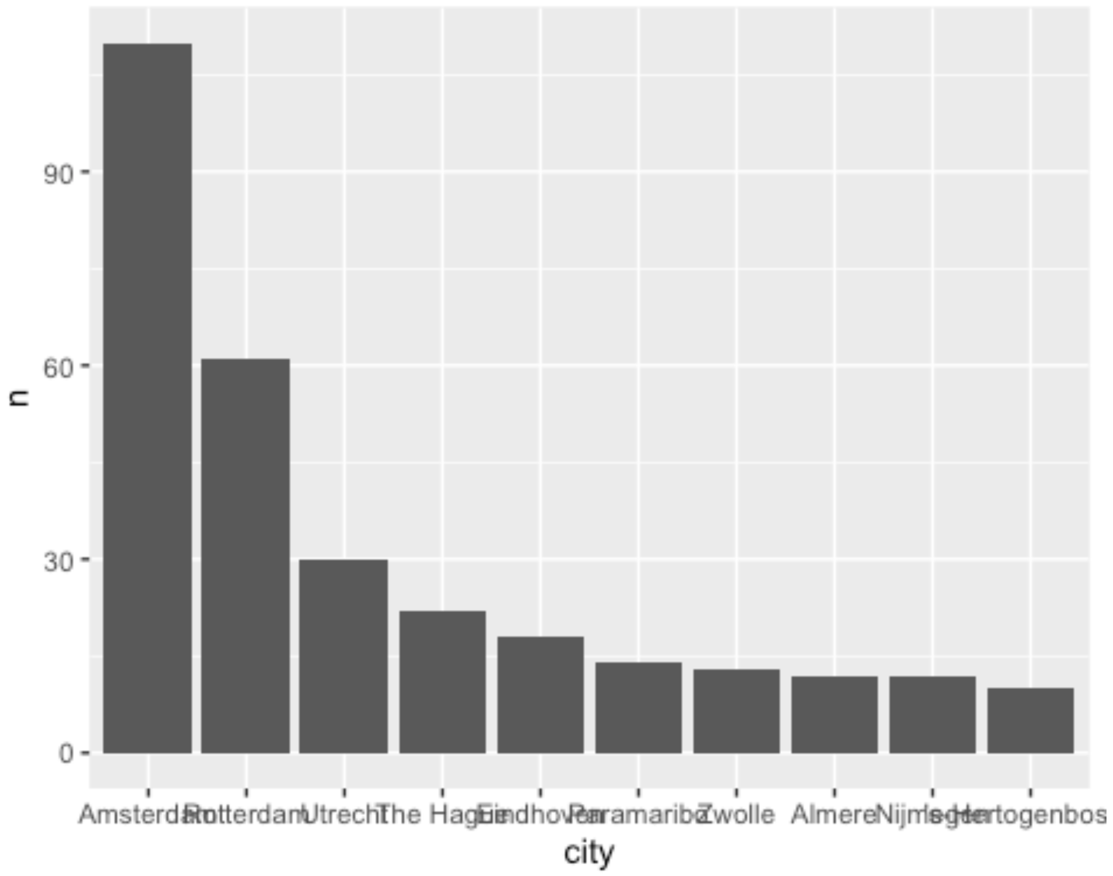
There are 25 "(not set)" variables that needs to be removed. Actually those will not be removed but converted to NA. This with the purpose to keep the complete overview on the dataset and later on for analysis purposes the NA's can easily be "hide" from the dataset.

```
dataset$city [dataset$city == "(not set)"] <- NA
```

After changing the ‘not set)’ values to NA, further exploration will be done on the city feature.

Cities with most respondents
Creating a first view on where respondent are coming from without filtering on buyer vs non-buyer. Therefore the ‘pipe’ function is being used to filter the right data and ggplot for creating the graph.

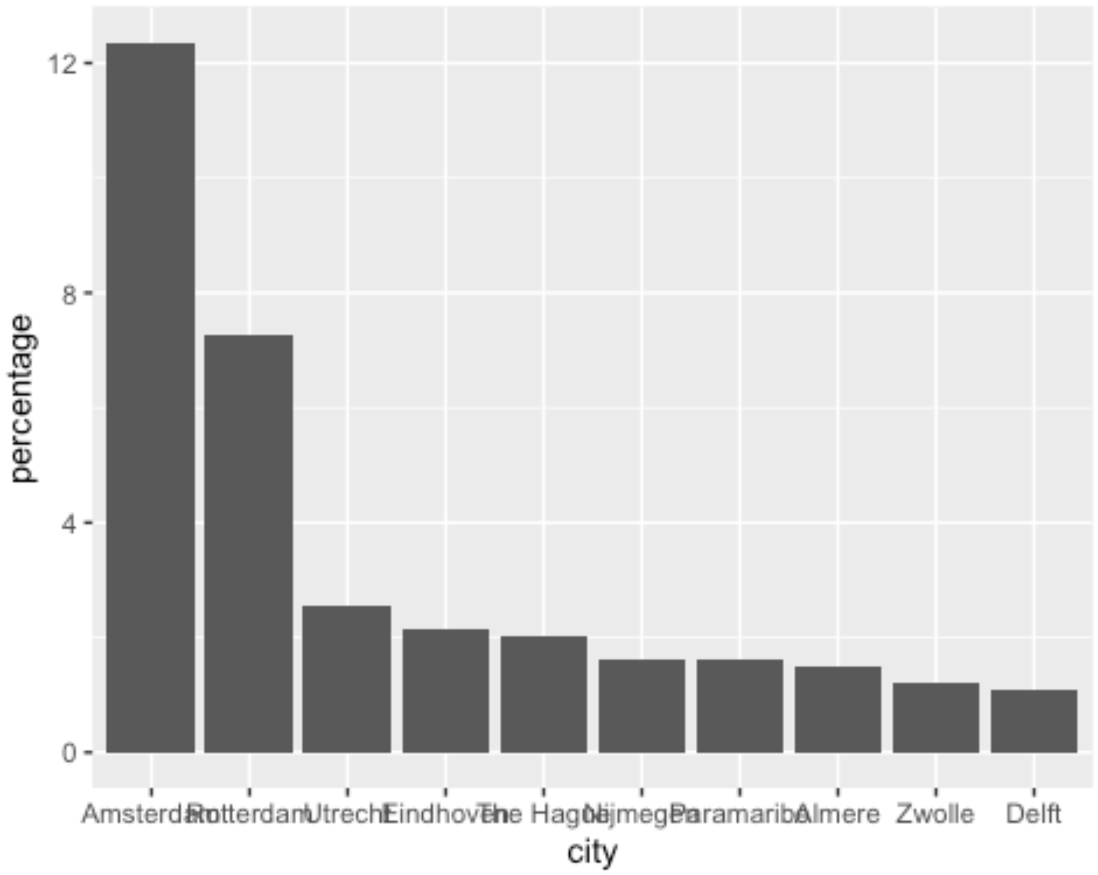
```
dataset %>%
  select(city) %>%
  na.omit(city) %>%
  count(city, sort = TRUE) %>%
  mutate(percentage = n / sum(n) * 100) %>%
  slice(1:10) %>%
  mutate(city = fct_reorder(city, n, .desc = TRUE)) %>%
  ggplot() +
  geom_bar(aes( x= city, y = n), stat = “identity”)
```



Most of the buyers are coming from Amsterdam, Rotterdam and Utrecht which are also the biggest cities in the Netherlands. Funny fact is that there are buyers from Paramaribo which is not a country the company operates in.

Buyers and the biggest cities
Now filtering will be done on the group of buyers and the cities that they come from. With the dplyr-functionality the right selection will be made to filter the data. The 'na.omit' option is used to exlude the NA's wich where being converted from the '(not set)' observation. The 'count' function counts the records and the 'mutate' function convert the count to percentage. With the 'slice' function only the 10 records will be showed. With the 'fct_reorder' function the right order will be created to prepare for displaying the graph with the ggplot-function.

```
dataset %>%
  select(city, giving_product) %>%
  group_by(giving_product) %>%
  filter(giving_product == "Yes") %>%
  na.omit(city) %>%
  count(city, sort = TRUE) %>%
  mutate(percentage = n / sum(n) * 100) %>%
  slice(1:10) %>%
  mutate(city = fct_reorder(city, n, .desc = TRUE)) %>%
  ggplot() +
  geom_bar(aes( x= city, y = percentage), stat = "identity")
```

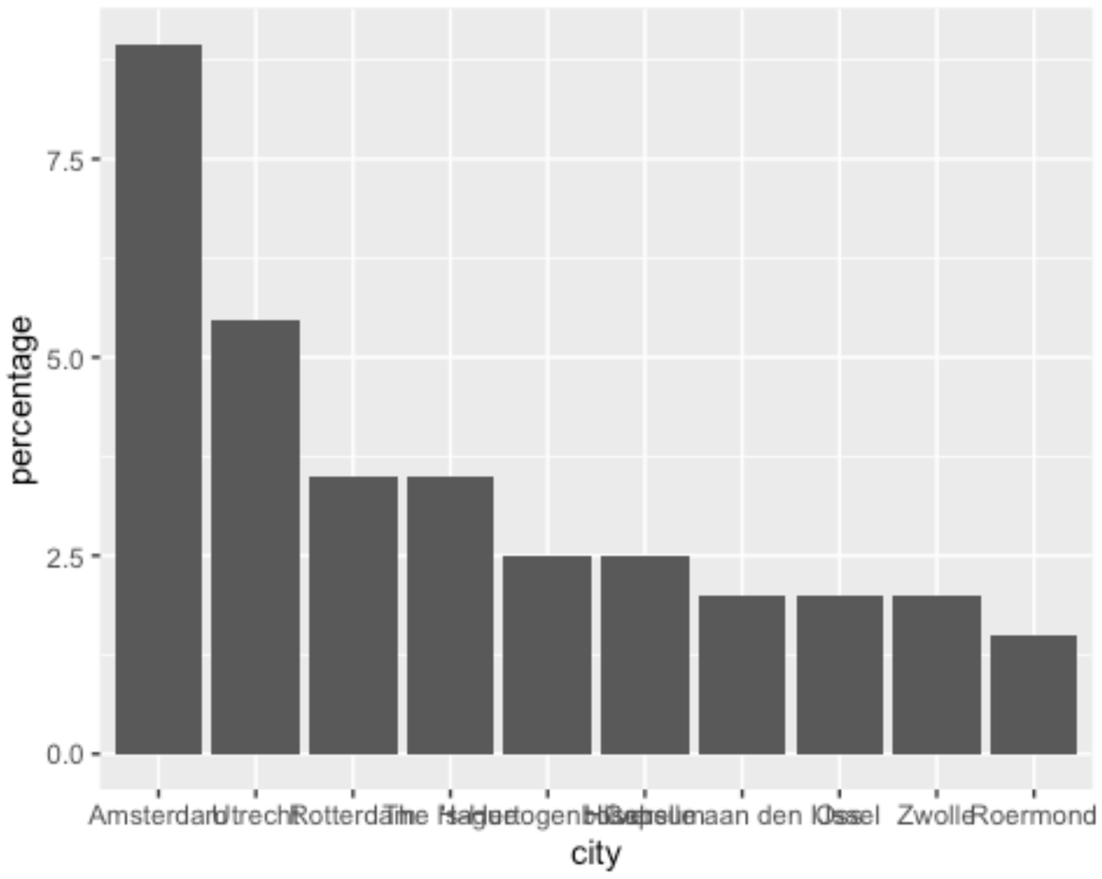


After filtering the data, a similar view exist as the unfiltered version. Also here the biggest cities are similar with the biggest cities in the Netherland. There is one difference compared with the full view which is the city of “Eindhoven” that changed places with “The Hague”.

Non-buyers and the biggest cities

Here a repetition of cities with the non-buying group.

```r
dataset %>%
  select(city, giving_product) %>%
  filter(giving_product == "No") %>%
  na.omit(city) %>%
  count(city, sort = TRUE) %>%
  mutate(percentage = n / sum(n) * 100) %>%
  slice(1:10) %>%
  mutate(city = fct_reorder(city, n, .desc = TRUE)) %>%
  ggplot() +
  geom_bar(aes( x= city, y = percentage), stat = "identity")
```



Also for the non-buyers no big difference in the cities, also the amount of participants get that small that there is no much to say about cities.

3.2.4 Website behaviour: number of pages per session

The pages per session is a standard metric in Google Analytics and counts the amount of pages visited in a session. A session is a group of interactions which takes place within a given timeframe. The amount of pages visited could be a possible metric for user engagement on a website, especially for the more content and blog oriented websites.

Average pages per session
The analysis starts with the average pages per session splitted for buyers and non-buyers.

```
dataset %>%
  select(giving_product, pages_session) %>%
  group_by(giving_product) %>%
  summarise(mean_pages_session = mean(pages_session))
```

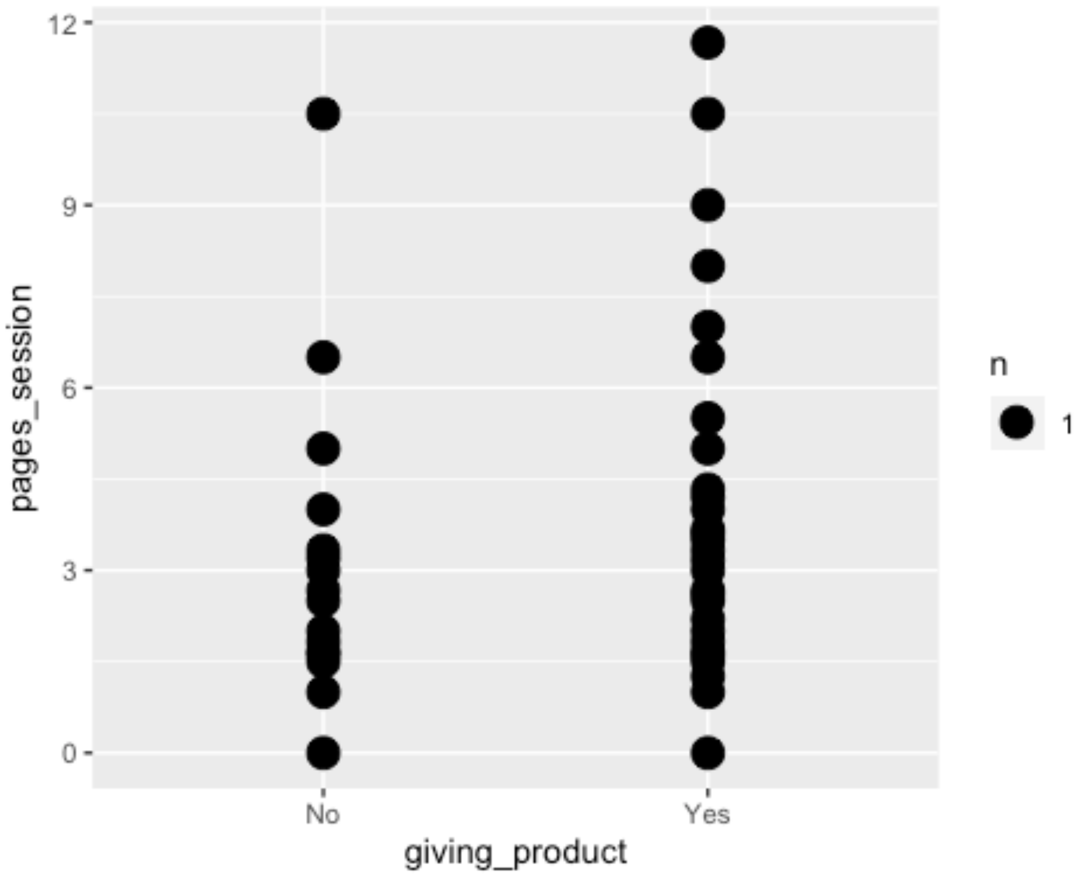
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   giving_product mean_pages_session
##   <fct>                       <dbl>
## 1 No                           1.85
## 2 Yes                          1.86
```

The result is an equal average of pages visited for the buyers and the non-buyers group. This says not much on the difference, therefore a deeper analysis will be done on the pages visited.

Distribution of count of pages visited
A analysis will be done on the count instead on the average to see if there are outliers affecting the average.

```
dataset %>%
  select(giving_product, pages_session) %>%
  group_by(giving_product) %>%
  count(pages_session) %>%
  ggplot(aes(x = giving_product, y= pages_session))+
  geom_count()
```



For the buyers and non-buyers the most pages visited centers between 1.5 and 3. For the group of buyers also more pages are visited that the non-buying group. Meaning that the pages visited could have an impact on the fact is someone buys the product.

3.2.5 Website behaviour: average session duration

Convert factor to numeric
To calculate the average, first the conversion needs to be done from factor to numeric to apply arithmetic calculations. This will be done with the "as.character, as.numeric" function. As this formats leads to NA's, the "hms" function convert the original time format to second.

```
dataset$avg_session_dur <- as.character(dataset$avg_session_dur)
dataset$avg_session_dur <- hms(dataset$avg_session_dur)
dataset$avg_session_dur <- as.numeric(dataset$avg_session_dur)
str(dataset)
```

```
## 'data.frame':    970 obs. of  11 variables:
##  $ userid                 : Factor w/ 970 levels
"006b1483","00a5c05c",..: 133 534 477 178 3 696 207 471 341 751 ...
##  $ nps                    : num  10 9 7 8 NA 10 9 NA 6 6 ...
##  $ giving_product         : Factor w/ 2 levels "No","Yes": 2 2 1 1 2
2 2 2 2 1 ...
##  $ page_type              : Factor w/ 3 levels "Blog","Brand",..: 1
1 1 1 1 1 1 1 1 1 ...
##  $ source.used            : Factor w/ 14 levels "bingorganic",..: 2
4 10 4 1 9 9 10 10 4 ...
##  $ city                   : Factor w/ 327 levels "'s-Gravendeel",..:
245 91 235 16 16 138 16 188 266 114 ...
##  $ avg_session_dur        : num  57 316 2305 2011 1971 ...
##  $ avg_session_quality    : num  2.9 40.4 48.2 52.2 34.1 41.1 22.7
60.5 6.3 57.5 ...
##  $ avg_time_page          : Factor w/ 182 levels
"00:00:00","00:00:01",..: 38 77 29 6 178 8 15 6 4 19 ...
##  $ registration_completed: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pages_session          : num  1 1 4 2.5 1 1.5 1.5 1 1 3 ...
```

Average sessions duration

After the conversion from factors to numeric, the calculations can be done on the average session duration. Within the "summarize" function the number of seconds will be converted to minutes.

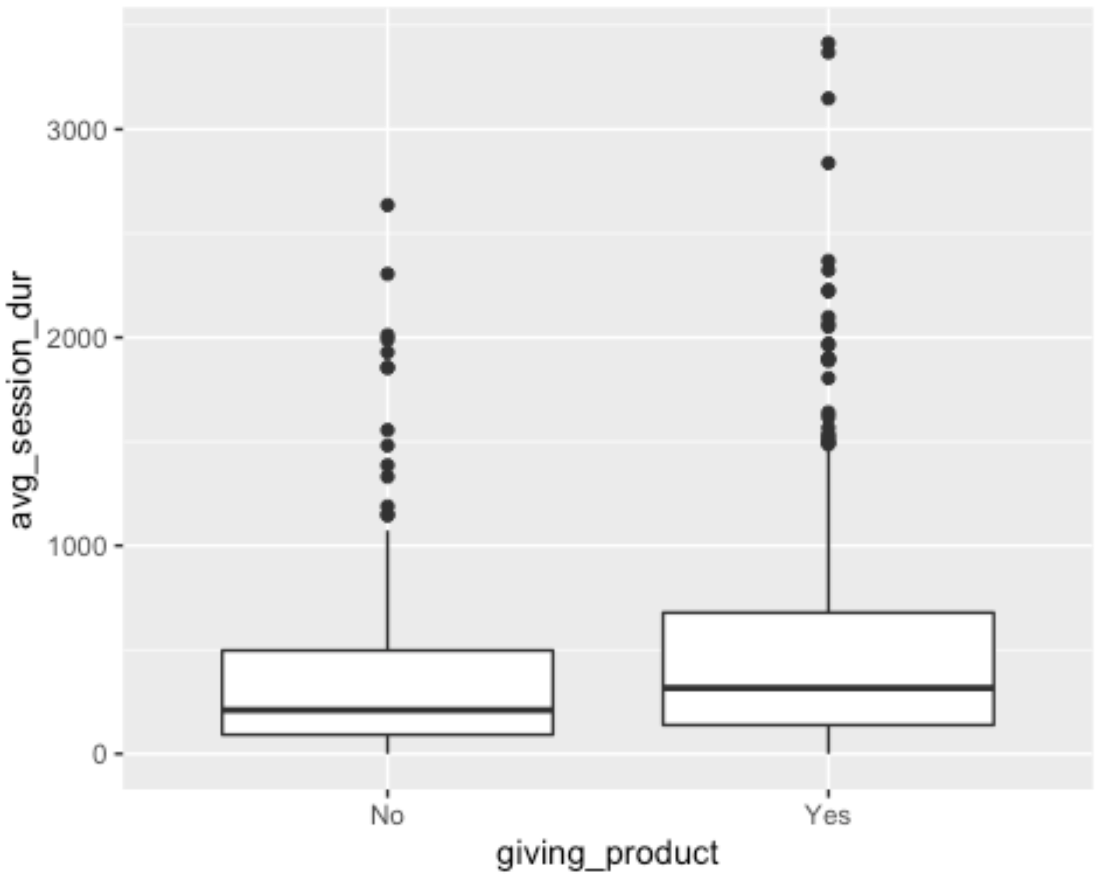
```
dataset %>%
  select(giving_product, avg_session_dur) %>%
  group_by(giving_product) %>%
  summarize(average_min = mean(avg_session_dur / 60))

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 2 x 2
##   giving_product average_min
##   <fct>                <dbl>
## 1 No                    5.66
## 2 Yes                   5.96
```

The buyers in average are spending slightly more time on the website than the non-buyers. The same analysis will be done on the distribution.

Distribution average session duration

The average scores on the average sessions duration for buyers and non-buyers were close-by, an analysis will be done on the distribution on the count of average session duration.

```
dataset %>%
  select(giving_product, avg_session_dur) %>%
  group_by(giving_product) %>%
  count(avg_session_dur) %>%
  ggplot(aes(x=giving_product, y=avg_session_dur)) +
  geom_boxplot()
```



Looking at the counts, it seems that for the buyers, the amount of longer average session duration is higher than for the non-buyers. There is another potential indicator of consumer engagement, which is the time on site/page. This measures how long a visitors stays on a page.

3.2.6 Website behaviour: average time on page

The "average_time_page" feature is almost similar as the "average session duration" metric. The difference is that the "average time on page" only measures the length of a page visited.

Also the "average time on page" is a factor which needs to be converted to numeric for calculation purposes.

```
dataset$avg_time_page <- as.character(dataset$avg_time_page)
dataset$avg_time_page <- hms(dataset$avg_time_page)
dataset$avg_time_page <- as.numeric(dataset$avg_time_page)
str(dataset)
```

```
## 'data.frame':    970 obs. of  11 variables:
##  $ userid                 : Factor w/ 970 levels
"006b1483","00a5c05c",..: 133 534 477 178 3 696 207 471 341 751 ...
##  $ nps                    : num  10 9 7 8 NA 10 9 NA 6 6 ...
##  $ giving_product         : Factor w/ 2 levels "No","Yes": 2 2 1 1 2
2 2 2 2 1 ...
##  $ page_type              : Factor w/ 3 levels "Blog","Brand",..: 1
1 1 1 1 1 1 1 1 1 ...
##  $ source.used            : Factor w/ 14 levels "bingorganic",..: 2
4 10 4 1 9 9 10 10 4 ...
##  $ city                   : Factor w/ 327 levels "'s-Gravendeel",..:
245 91 235 16 16 138 16 188 266 114 ...
##  $ avg_session_dur        : num  57 316 2305 2011 1971 ...
##  $ avg_session_quality    : num  2.9 40.4 48.2 52.2 34.1 41.1 22.7
60.5 6.3 57.5 ...
##  $ avg_time_page          : num  37 81 28 5 1441 ...
##  $ registration_completed: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pages_session          : num  1 1 4 2.5 1 1.5 1.5 1 1 3 ...
```

```
dataset %>%
  select(giving_product, avg_time_page) %>%
  group_by(giving_product) %>%
  summarize(mean_time_page_seconds = mean(avg_time_page ))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 2 x 2
##   giving_product mean_time_page_seconds
##   <fct>                           <dbl>
## 1 No                               56.8
## 2 Yes                              60.8
```

There is a very small difference in the average time of visiting webpages between the buying and non-buying group. The difference between the groups is in average 4 seconds.

3.2.7 Website behaviour: session quality

The session quality is an estimate based on Google's machine learning to understand how close a Google Analytics session was to transactions. As transaction are less relevant for this website, it is a less important metric but could still be relevant for this analysis. Google creates a number for every session or session, with a range between 0 and 100 where 0 is no transaction and 100 is probably a transaction.

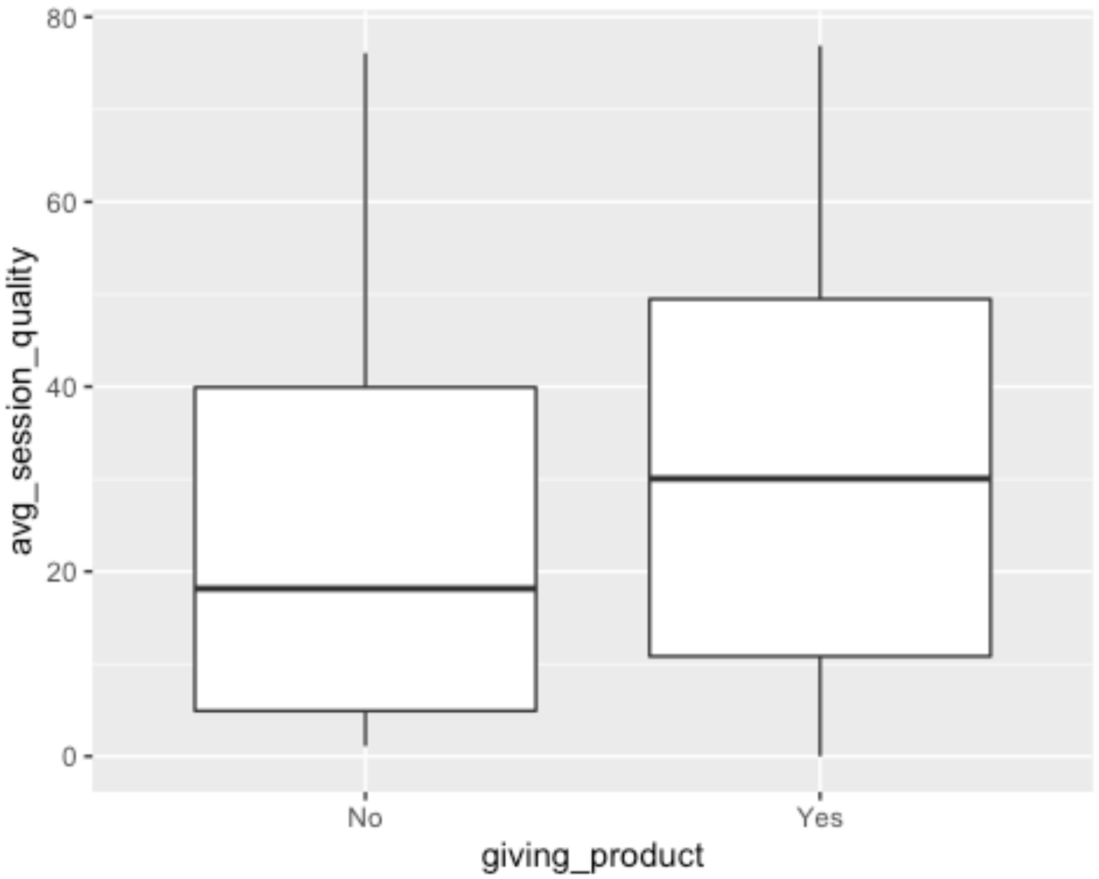
Average session quality

```
dataset %>%
  select(giving_product, avg_session_quality) %>%
  group_by(giving_product) %>%
  summarise(mean = mean(avg_session_quality))

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 2 x 2
##   giving_product  mean
##   <fct>          <dbl>
## 1 No              18.8
## 2 Yes             18.9
```

It seems that the average of the 'average session quality' is similar for the buying and non-buying group. Both are almost 20 which means that also both groups are at least far away behaviour that estimates if a purchase would be done.

Distribution of average session quality
Similar to other website metrics, a deep analysis will be done on the count to see if there is a bigger difference between the two groups.

```
dataset %>%
  select(giving_product, avg_session_quality) %>%
  group_by(giving_product) %>%
  count(avg_session_quality) %>%
  ggplot(aes(x=giving_product, y=avg_session_quality)) +
  geom_boxplot()
```



In the boxplot a small difference is being presented where the buying group seems to have slightly higher session quality than the non-buying group.

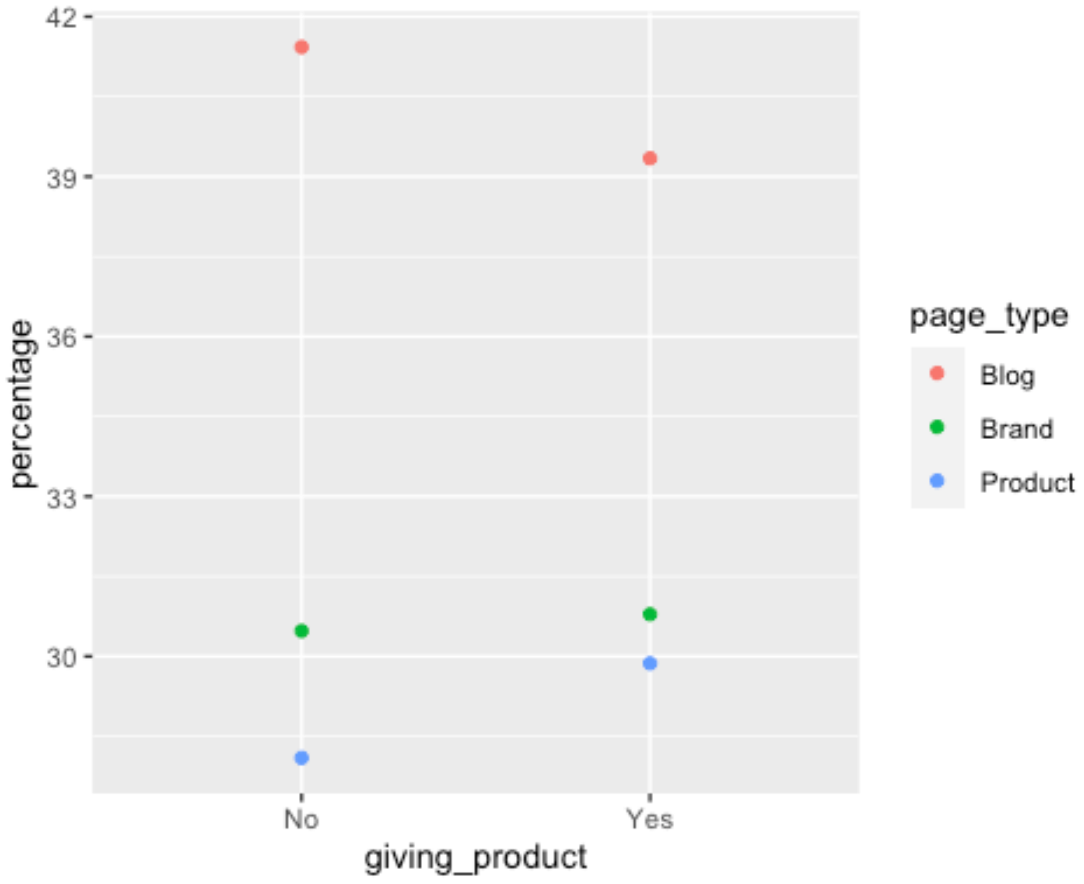
3.2.8 Website behaviour: page type

The ‘page type’ feature contains more information on the type of pages being visited on the website. There are basically 3 different type of pages:

1) Blog: these pages are containing written content by employees, customers and experts, not related to a product.
2) Brand: these pages are “branded” pages and containing information about the brands and are more commercial then blog pages.
3) Product: these are product pages with mostly more detailed information about the products and what their benefits are.

To understand the effect of pages on buying behaviour, the dplyr-functionality is being used together with a plot within the ggplot-functionality.

```
dataset %>%
  select(page_type, giving_product) %>%
  group_by(giving_product) %>%
  count(page_type, sort = TRUE) %>%
  mutate(percentage = n / sum(n) * 100) %>%
ggplot(aes(x = giving_product, y= percentage, color = page_type))+
  geom_point()
```



The ‘blog pages’ are in both cases the most visited and even more by the non-buyers. Also ‘brand’ and ‘product’ pages are showing similar results for both of the groups.

3.2.9 Website behaviour: source used

In this section the ‘sources’ will be further analysed. A ‘source’ is a type of channel that is being used by the visitor to land on the website. Most common channels are for instance Google Search, Facebook or other social media channels.

Most sources being used buyers
With the ‘filter’ function within ‘dyplr’, the right selection will be done on the most used sourced by the buying group. With the ‘mutate’ function the count of buyers will be converted to a percentage to get a better view.

```
dataset %>%
  select(giving_product, source.used) %>%
  filter(giving_product == "Yes") %>%
  group_by(giving_product) %>%
  count(source.used, sort = TRUE) %>%
  mutate(percentage = n / sum(n) * 100)
```

```
## # A tibble: 14 x 4
## # Groups:   giving_product [1]
##    giving_product source.used            n percentage
##    <fct>          <fct>              <int>      <dbl>
##  1 Yes            googlecpc            321     42.2
##  2 Yes            googleorganic        287     37.8
##  3 Yes            direct                70      9.21
##  4 Yes            exm                   29      3.82
##  5 Yes            referral              22      2.89
##  6 Yes            bingorganic           16      2.11
##  7 Yes            cpcfacebook            4      0.526
##  8 Yes            facebookcpm            4      0.526
##  9 Yes            duckduckgoorganic      2      0.263
## 10 Yes            cpcsocial              1      0.132
## 11 Yes            famblenddisplay        1      0.132
## 12 Yes            website                1      0.132
## 13 Yes            yahooorganic           1      0.132
## 14 Yes            youtubecpm             1      0.132
```

In the overview above, very clearly two channels are being used the most by buyers: the paid advertising links in Google (CPC) and the organic links (organic), follow by direct traffic and the e-mail program.

Most sources being used non-buyers
The same analysis will be done for the non-buying group with the use of the same methodology as for the buying group.

```
dataset %>%
  select(giving_product, source.used) %>%
  filter(giving_product == "No") %>%
  group_by(giving_product) %>%
  count(source.used, sort = TRUE) %>%
  mutate(percentage = n / sum(n) * 100)
```

```
## # A tibble: 8 x 4
## # Groups:   giving_product [1]
##   giving_product source.used            n percentage
##   <fct>          <fct>              <int>      <dbl>
## 1 No             googleorganic         87     41.4
## 2 No             googlecpc             78     37.1
## 3 No             direct                22     10.5
## 4 No             bingorganic            7      3.33
## 5 No             exm                    7      3.33
## 6 No             referral               6      2.86
## 7 No             duckduckgoorganic      2      0.952
## 8 No             website                1      0.476
```

The non-buying group are also using the Google Search channel the most but slightly different then the buying group. Most used channel is Google Organic, which are the organic links (not paid) in Google.

3.2.10 EDA conclusion

All features have been analyses which helps in the next section, data preparation. A good analysis is necessary to provide more context in order to prepare for building the machine learning model. Besides that it is also helpful to interpret the results in the right way.

In total there are 8 features being analysed (predictors) and the outcome, buying vs non-buying. Looking towards correlation between these features and buying behaviour, there seems to be a light connection between the hight of NPS-scores and buying behaviour. Especially for the 'promotors', which are people given NPS-scores higher then 9, are better represented in the buying-group then in the non-buying group.

4: Data preparation

In the previous sections all features have been analysed and a first view on the potential predictors has been created. In the 'data preparation' stage, a couple of steps will be taken:

- Dropping not relevant features
- Converting still existing factors to numeric
- Checking and removing NA's
- Checking for imbalance

4.1 Dropping not relevant features

First the features that are not relevant will be dropped. The three features that will be dropped are:

- 1 registration_completed
- 2 city
- 4 User ID

To prevent the original dataset get's lost, first a copy will be created with the name "dataset_ml".

The 'registration' feature will be dropped because it contains no information. The 'city' feature is not relevant based on the analysis being done. And the 'userid' feature is not relevant for building the machine learning model.

```
dataset_ml <- dataset
dataset_ml$registration_completed <- NULL
dataset_ml$city <- NULL
dataset_ml$userid <- NULL
str(dataset_ml)

## 'data.frame':    970 obs. of  8 variables:
##  $ nps                 : num  10 9 7 8 NA 10 9 NA 6 6 ...
##  $ giving_product      : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 2
2 2 2 1 ...
##  $ page_type           : Factor w/ 3 levels "Blog","Brand",..: 1 1 1
1 1 1 1 1 1 ...
##  $ source.used         : Factor w/ 14 levels "bingorganic",..: 2 4
10 4 1 9 9 10 10 4 ...
##  $ avg_session_dur     : num  57 316 2305 2011 1971 ...
##  $ avg_session_quality: num  2.9 40.4 48.2 52.2 34.1 41.1 22.7 60.5
6.3 57.5 ...
##  $ avg_time_page       : num  37 81 28 5 1441 ...
##  $ pages_session       : num  1 1 4 2.5 1 1.5 1.5 1 1 3 ...
```

4.2 Converting factors to numeric

There are still factors available in the dataset which are more difficult to handle in building the Machine Learning model. Therefore factors will be converted to numeric. The "unclass" function is being used to convert the factor first, then 'as.numeric' is used to convert to numeric.

Sources used

Converting ‘sources’ used to numeric. This would mean that the information on what type of sources, is being lost because it get’s a number. This information can be retrieved in the original dataset incase the ‘source used’ would be a relevant predicting feature.

```r
unclass_sources <- unclass(dataset_ml$source.used)
dataset_ml$source.used <- as.numeric(unclass_sources)
str(dataset_ml)
```

```
## 'data.frame':    970 obs. of  8 variables:
##  $ nps                : num  10 9 7 8 NA 10 9 NA 6 6 ...
##  $ giving_product     : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 2
2 2 2 1 ...
##  $ page_type          : Factor w/ 3 levels "Blog","Brand",..: 1 1 1
1 1 1 1 1 1 1 ...
##  $ source.used        : num  2 4 10 4 1 9 9 10 10 4 ...
##  $ avg_session_dur    : num  57 316 2305 2011 1971 ...
##  $ avg_session_quality: num  2.9 40.4 48.2 52.2 34.1 41.1 22.7 60.5
6.3 57.5 ...
##  $ avg_time_page      : num  37 81 28 5 1441 ...
##  $ pages_session      : num  1 1 4 2.5 1 1.5 1.5 1 1 3 ...
```

Pages

Converting the ‘pages’ factor to numeric.

```r
unclass_pages <- unclass(dataset_ml$page_type)
dataset_ml$page_type <- as.numeric(unclass_pages)
str(dataset_ml)
```

```
## 'data.frame':    970 obs. of  8 variables:
##  $ nps                : num  10 9 7 8 NA 10 9 NA 6 6 ...
##  $ giving_product     : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 2
2 2 2 1 ...
##  $ page_type          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ source.used        : num  2 4 10 4 1 9 9 10 10 4 ...
##  $ avg_session_dur    : num  57 316 2305 2011 1971 ...
##  $ avg_session_quality: num  2.9 40.4 48.2 52.2 34.1 41.1 22.7 60.5
6.3 57.5 ...
```

```
##   $ avg_time_page       : num  37 81 28 5 1441 ...
##   $ pages_session       : num  1 1 4 2.5 1 1.5 1.5 1 1 3 ...
```

4.3 Removing NA's

In the overview above, in the NPS features, NA's are visible. Simply removing them will affect the outcomes of the analysis and prediction. The 'mice' function is used to impute missing values. This function helps to replace missing values with plausible values, in this case based on the 'mean'.

```
dataset_na <- mice(dataset_ml, m=1, maxit = 5, method = "mean", seed =
500)

##
##   iter imp variable
##    1   1  nps
##    2   1  nps
##    3   1  nps
##    4   1  nps
##    5   1  nps

full_na <- complete(dataset_na, 1)
dim(full_na)

## [1] 970   8

str(full_na)

## 'data.frame':    970 obs. of  8 variables:
##   $ nps                 : num  10 9 7 8 7.39 ...
##   $ giving_product      : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 2
2 2 2 1 ...
##   $ page_type           : num  1 1 1 1 1 1 1 1 1 1 ...
##   $ source.used         : num  2 4 10 4 1 9 9 10 10 4 ...
##   $ avg_session_dur     : num  57 316 2305 2011 1971 ...
##   $ avg_session_quality: num  2.9 40.4 48.2 52.2 34.1 41.1 22.7 60.5
6.3 57.5 ...
##   $ avg_time_page       : num  37 81 28 5 1441 ...
##   $ pages_session       : num  1 1 4 2.5 1 1.5 1.5 1 1 3 ...
```

Checking for missing NA's

```
(na_count_full_na <- data.frame(sapply(full_na,
function(y)sum(length(which(is.na(y)))))))

##
sapply.full_na..function.y..sum.length.which.is.na.y.....
## nps
0
## giving_product
0
## page_type
0
## source.used
0
## avg_session_dur
0
## avg_session_quality
0
## avg_time_page
0
## pages_session
0
```

No missing values detected anymore.

4.4 Class imbalance

In the EDA was to see that the amount of buyers is bigger than non-buyers. A check below will be done what the proportion is of buyers and non-buyers in the test and training set.

```
options(digits = 2)
prop.table(table(dataset$giving_product))

##
##   No  Yes
## 0.22 0.78
```

There is a 'imbalance' problem with more buyers than non-buyers in the dataset. But imbalance becomes a bigger problem once the group that is important to the prediction, is much smaller than the rest of the group. When training the model the class imbalance will be taken care of with the 'upsampling' function.

5: Training the model

In this section the model will be trained by creating a test and training set. Different machine learning models will be tested at the same time to handle the classification problem.

5.1 Creating test and training set

The 'CreateDataPartion' function will be used to create the test and training set. For this assignment a ratio of 75% for the trainings set has been chosen.

```
set.seed(123)
y <-full_na$giving_product
my_index <- createDataPartition(y, times = 1, p = 0.75, list = FALSE)
x_train <- full_na[my_index, ]
x_test <- full_na[-my_index, ]
str(x_train)
```

```
## 'data.frame':    728 obs. of  8 variables:
##  $ nps                 : num  9 7 10 7.39 6 ...
##  $ giving_product      : Factor w/ 2 levels "No","Yes": 2 1 2 2 1 2
1 2 2 2 ...
##  $ page_type           : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ source.used         : num  4 10 9 10 4 9 10 9 4 9 ...
##  $ avg_session_dur     : num  316 2305 1962 1640 1145 ...
##  $ avg_session_quality: num  40.4 48.2 41.1 60.5 57.5 8.5 62.5 34.6
51.3 18.7 ...
##  $ avg_time_page       : num  81 28 7 5 18 10 19 9 207 98 ...
##  $ pages_session       : num  1 4 1.5 1 3 1 1.5 2.5 4.33 2.5 ...
```

```
class(y)
```

```
## [1] "factor"
```

5.2 Increase performance

To boost speed, first parallel processing will be activated.

```
registerDoParallel(3)
getDoParWorkers()

## [1] 3

set.seed(123)
```

5.3 Cross-validation

Before the actual modelling starts, cross-validation is activated. Cross-validation is used to prevent overfitting. The normal cross-validation is activated with standard the 10-fold validation. As there is an imbalance in the data set, especially for the predicted value("giving_product") with 78% of buyers, the "up-sampling" function will be used.

```
my_control <- trainControl(method = "repeatedcv", repeats = 5, number
= 10, savePredictions = 'final', classProbs = TRUE, allowParallel =
TRUE, sampling = "up", index = createResample(x_train$giving_product))
```

5.4 Training the model

Methods as LDA and QDA are not working well with many predictors in a dataset, same counts for kNN and local regression. In this case there is a binairy classification problem but with multiple predictors and therefore the following models are selected to run the model:

- Multinom
- RF
- svmRadial
- Adaboost
- xgbDart

As the dataset is imbalanced, the "Kappa" metric is used to select the optimal model instead of 'Accuracy'. The option "PreProcess" is used to center and scale the data.

```
modeltypes <- list(multinom = caretModelSpec(method = "multinom"), rf
= caretModelSpec(method = "rf"), svmRadial = caretModelSpec(method =
"svmRadial"), adaboost = caretModelSpec(method = "adaboost"), xgbDart
= caretModelSpec(method = “xgbDART"))
```

```
models <- caretList(giving_product ~ ., data = x_train, tuneList =
modeltypes, metric = "Kappa", continue_on_fail = FALSE, preProcess =
c("center", "scale"), trControl = my_control)


multinom

Penalized Multinomial Regression


728 samples

  7 predictor

  2 classes: 'No', 'Yes'


Pre-processing: centered (7), scaled (7)

Resampling: Cross-Validated (10 fold, repeated 5 times)

Summary of sample sizes: 728, 728, 728, 728, 728, 728, ...

Addtional sampling using up-sampling prior to pre-processing


Resampling results across tuning parameters:


  decay  Accuracy  Kappa

  0e+00  0.55      0.045

  1e-04  0.55      0.034

  1e-01  0.56      0.046


Kappa was used to select the optimal model using the largest value.

The final value used for the model was decay = 0.1.


$rf

Random Forest
```

```
728 samples

  7 predictor

  2 classes: 'No', 'Yes'


Pre-processing: centered (7), scaled (7)

Resampling: Cross-Validated (10 fold, repeated 5 times)

Summary of sample sizes: 728, 728, 728, 728, 728, 728, ...

Addtional sampling using up-sampling prior to pre-processing


Resampling results across tuning parameters:


  mtry  Accuracy  Kappa

  2     0.72       0.0035

  4     0.71       0.0019

  7     0.69      -0.0194


Kappa was used to select the optimal model using the largest value.

The final value used for the model was mtry = 2.


$svmRadial

Support Vector Machines with Radial Basis Function Kernel


728 samples

  7 predictor

  2 classes: 'No', 'Yes'


Pre-processing: centered (7), scaled (7)

Resampling: Cross-Validated (10 fold, repeated 5 times)
```

```
Summary of sample sizes: 728, 728, 728, 728, 728, 728, ...

Addtional sampling using up-sampling prior to pre-processing


Resampling results across tuning parameters:


  C     Accuracy  Kappa

  0.25  0.57      0.0083

  0.50  0.61      0.0317

  1.00  0.62      0.0430


Tuning parameter 'sigma' was held constant at a value of 0.22

Kappa was used to select the optimal model using the largest value.

The final values used for the model were sigma = 0.22 and C = 1.


$adaboost

AdaBoost Classification Trees


728 samples

  7 predictor

  2 classes: 'No', 'Yes'


Pre-processing: centered (7), scaled (7)

Resampling: Cross-Validated (10 fold, repeated 5 times)

Summary of sample sizes: 728, 728, 728, 728, 728, 728, ...

Addtional sampling using up-sampling prior to pre-processing


Resampling results across tuning parameters:
  nIter  method         Accuracy  Kappa

   50    Adaboost.M1    0.72      -0.00453

   50    Real adaboost  0.75       0.01738

  100    Adaboost.M1    0.72       0.00489

  100    Real adaboost  0.75       0.00335

  150    Adaboost.M1    0.72      -0.00089

  150    Real adaboost  0.75       0.01833


Kappa was used to select the optimal model using the largest value.

The final values used for the model were nIter = 150 and method = Real adaboost.


$xgbDart

eXtreme Gradient Boosting


728 samples

  7 predictor

  2 classes: 'No', 'Yes'


Pre-processing: centered (7), scaled (7)

Resampling: Cross-Validated (10 fold, repeated 5 times)

Summary of sample sizes: 728, 728, 728, 728, 728, 728, ...

Addtional sampling using up-sampling prior to pre-processing


Resampling results across tuning parameters:


  max_depth  eta  rate_drop  skip_drop  subsample  colsample_bytree  nrounds  Accuracy  Kappa

  1          0.3  0.01       0.05       0.50       0.6                50      0.58       1.1e-02

  1          0.3  0.01       0.05       0.50       0.6               100      0.58       2.7e-02
```

```
1         0.3  0.01      0.05      0.50      0.6             150     0.58      1.7e-02
1         0.3  0.01      0.05      0.50      0.8              50     0.58      2.5e-02
1         0.3  0.01      0.05      0.50      0.8             100     0.57      1.7e-02
1         0.3  0.01      0.05      0.50      0.8             150     0.58     -2.1e-03
1         0.3  0.01      0.05      0.75      0.6              50     0.57      2.2e-02
1         0.3  0.01      0.05      0.75      0.6             100     0.58      2.9e-02
1         0.3  0.01      0.05      0.75      0.6             150     0.58      1.7e-02
1         0.3  0.01      0.05      0.75      0.8              50     0.57      2.2e-02
1         0.3  0.01      0.05      0.75      0.8             100     0.57      1.2e-02
1         0.3  0.01      0.05      0.75      0.8             150     0.57      7.7e-03
1         0.3  0.01      0.05      1.00      0.6              50     0.56      4.5e-03
1         0.3  0.01      0.05      1.00      0.6             100     0.55     -1.3e-02
1         0.3  0.01      0.05      1.00      0.6             150     0.56     -1.5e-02
1         0.3  0.01      0.05      1.00      0.8              50     0.58      2.5e-02
1         0.3  0.01      0.05      1.00      0.8             100     0.56     -9.4e-03
1         0.3  0.01      0.05      1.00      0.8             150     0.57     -5.9e-03
1         0.3  0.01      0.95      0.50      0.6              50     0.58      2.9e-02
1         0.3  0.01      0.95      0.50      0.6             100     0.58      1.3e-02
1         0.3  0.01      0.95      0.50      0.6             150     0.57     -5.0e-03
1         0.3  0.01      0.95      0.50      0.8              50     0.58      3.0e-02
1         0.3  0.01      0.95      0.50      0.8             100     0.59      2.1e-02
1         0.3  0.01      0.95      0.50      0.8             150     0.58      1.7e-02
1         0.3  0.01      0.95      0.75      0.6              50     0.58      2.2e-02
1         0.3  0.01      0.95      0.75      0.6             100     0.57     -2.5e-03
1         0.3  0.01      0.95      0.75      0.6             150     0.58     -4.0e-03
1         0.3  0.01      0.95      0.75      0.8              50     0.58      1.9e-02
1         0.3  0.01      0.95      0.75      0.8             100     0.58     -6.8e-05
1         0.3  0.01      0.95      0.75      0.8             150     0.58     -7.1e-04
```

```
1          0.3  0.01       0.95       1.00       0.6                50      0.58       1.2e-02
1          0.3  0.01       0.95       1.00       0.6               100      0.56      -1.5e-02
1          0.3  0.01       0.95       1.00       0.6               150      0.56      -6.0e-03
1          0.3  0.01       0.95       1.00       0.8                50      0.56       3.6e-04
1          0.3  0.01       0.95       1.00       0.8               100      0.56      -9.1e-03
1          0.3  0.01       0.95       1.00       0.8               150      0.55      -2.5e-02
1          0.3  0.50       0.05       0.50       0.6                50      0.58       3.8e-02
1          0.3  0.50       0.05       0.50       0.6               100      0.58       4.0e-02
1          0.3  0.50       0.05       0.50       0.6               150      0.57       3.1e-02
1          0.3  0.50       0.05       0.50       0.8                50      0.60       4.6e-02
1          0.3  0.50       0.05       0.50       0.8               100      0.60       4.3e-02
1          0.3  0.50       0.05       0.50       0.8               150      0.58       3.6e-02
1          0.3  0.50       0.05       0.75       0.6                50      0.58       4.7e-02
1          0.3  0.50       0.05       0.75       0.6               100      0.58       4.2e-02
1          0.3  0.50       0.05       0.75       0.6               150      0.57       3.9e-02
1          0.3  0.50       0.05       0.75       0.8                50      0.62       7.1e-02
1          0.3  0.50       0.05       0.75       0.8               100      0.62       7.0e-02
1          0.3  0.50       0.05       0.75       0.8               150      0.61       6.6e-02
1          0.3  0.50       0.05       1.00       0.6                50      0.60       7.8e-02
1          0.3  0.50       0.05       1.00       0.6               100      0.56       3.5e-02
1          0.3  0.50       0.05       1.00       0.6               150      0.58       3.9e-02
1          0.3  0.50       0.05       1.00       0.8                50      0.62       6.9e-02
1          0.3  0.50       0.05       1.00       0.8               100      0.61       6.4e-02
1          0.3  0.50       0.05       1.00       0.8               150      0.62       6.6e-02
1          0.3  0.50       0.95       0.50       0.6                50      0.58       9.0e-03
1          0.3  0.50       0.95       0.50       0.6               100      0.56      -1.6e-02
1          0.3  0.50       0.95       0.50       0.6               150      0.58      -4.1e-03
1          0.3  0.50       0.95       0.50       0.8                50      0.58       1.7e-02
```

```
1         0.3 0.50      0.95      0.50      0.8              100     0.58      1.9e-02
1         0.3 0.50      0.95      0.50      0.8              150     0.58      7.0e-03
1         0.3 0.50      0.95      0.75      0.6               50     0.57      1.1e-02
1         0.3 0.50      0.95      0.75      0.6              100     0.56     -1.9e-02
1         0.3 0.50      0.95      0.75      0.6              150     0.57     -1.6e-02
1         0.3 0.50      0.95      0.75      0.8               50     0.58      2.9e-02
1         0.3 0.50      0.95      0.75      0.8              100     0.58      1.7e-02
1         0.3 0.50      0.95      0.75      0.8              150     0.58      1.2e-02
1         0.3 0.50      0.95      1.00      0.6               50     0.58      1.9e-02
1         0.3 0.50      0.95      1.00      0.6              100     0.57      1.4e-02
1         0.3 0.50      0.95      1.00      0.6              150     0.57      9.2e-03
1         0.3 0.50      0.95      1.00      0.8               50     0.57      1.5e-02
1         0.3 0.50      0.95      1.00      0.8              100     0.56     -4.1e-03
1         0.3 0.50      0.95      1.00      0.8              150     0.56     -5.5e-03
1         0.4 0.01      0.05      0.50      0.6               50     0.55      1.0e-04
1         0.4 0.01      0.05      0.50      0.6              100     0.57      4.0e-03
1         0.4 0.01      0.05      0.50      0.6              150     0.57     -9.9e-03
1         0.4 0.01      0.05      0.50      0.8               50     0.57      8.8e-03
1         0.4 0.01      0.05      0.50      0.8              100     0.58      1.0e-02
1         0.4 0.01      0.05      0.50      0.8              150     0.58      2.4e-02
1         0.4 0.01      0.05      0.75      0.6               50     0.58      1.1e-02
1         0.4 0.01      0.05      0.75      0.6              100     0.57     -3.3e-03
1         0.4 0.01      0.05      0.75      0.6              150     0.57     -4.1e-03
1         0.4 0.01      0.05      0.75      0.8               50     0.57      4.1e-03
1         0.4 0.01      0.05      0.75      0.8              100     0.57      1.6e-03
1         0.4 0.01      0.05      0.75      0.8              150     0.57     -9.7e-04
1         0.4 0.01      0.05      1.00      0.6               50     0.58      2.4e-02
1         0.4 0.01      0.05      1.00      0.6              100     0.58      1.6e-02
```

```
  1          0.4  0.01       0.05       1.00       0.6                150       0.57      -2.9e-03
  1          0.4  0.01       0.05       1.00       0.8                 50       0.57       8.9e-03
  1          0.4  0.01       0.05       1.00       0.8                100       0.56      -1.0e-02
  1          0.4  0.01       0.05       1.00       0.8                150       0.57      -1.1e-02
  1          0.4  0.01       0.95       0.50       0.6                 50       0.58       2.0e-02
  1          0.4  0.01       0.95       0.50       0.6                100       0.56       4.3e-03
  1          0.4  0.01       0.95       0.50       0.6                150       0.58       1.3e-03
  1          0.4  0.01       0.95       0.50       0.8                 50       0.58       1.9e-02
  1          0.4  0.01       0.95       0.50       0.8                100       0.58       1.4e-02
  1          0.4  0.01       0.95       0.50       0.8                150       0.58       8.0e-03
  1          0.4  0.01       0.95       0.75       0.6                 50       0.57      -1.6e-03
  1          0.4  0.01       0.95       0.75       0.6                100       0.57       7.7e-04
  1          0.4  0.01       0.95       0.75       0.6                150       0.56      -2.8e-02
  1          0.4  0.01       0.95       0.75       0.8                 50       0.57       1.0e-02
  1          0.4  0.01       0.95       0.75       0.8                100       0.57       7.5e-05
  1          0.4  0.01       0.95       0.75       0.8                150       0.58      -2.6e-03
  1          0.4  0.01       0.95       1.00       0.6                 50       0.57      -4.5e-03
  1          0.4  0.01       0.95       1.00       0.6                100       0.56      -2.4e-02
  1          0.4  0.01       0.95       1.00       0.6                150       0.56      -3.0e-02
  1          0.4  0.01       0.95       1.00       0.8                 50       0.56      -5.6e-03
  1          0.4  0.01       0.95       1.00       0.8                100       0.56      -5.2e-03
  1          0.4  0.01       0.95       1.00       0.8                150       0.55      -2.4e-02
  1          0.4  0.50       0.05       0.50       0.6                 50       0.58       4.5e-02
  1          0.4  0.50       0.05       0.50       0.6                100       0.57       4.4e-02
  1          0.4  0.50       0.05       0.50       0.6                150       0.60       4.9e-02
 [ reached getOption("max.print") -- omitted 321 rows ]


Tuning parameter 'gamma' was held constant at a value of 0
Tuning parameter 'min_child_weight' was held constant at a value of 1

Kappa was used to select the optimal model using the largest value.

The final values used for the model were nrounds = 50, max_depth = 1, eta = 0.4, gamma = 0,
subsample = 0.75, colsample_bytree

 = 0.8, rate_drop = 0.5, skip_drop = 0.05 and min_child_weight = 1.



attr(,"class")

[1] "caretList"
```

Looking at the results, 'Multinom' seems to has a lower accuracy but a better Kappa. The 'rf', 'svmradial' and 'adaboost' have a higher accuracy then 'Multinom' but a lower Kappa. Then the 'xgbDart' is with a slighty higher accuracy and Kappa probably the best solution. First, the evaluation will be done on the test set.

6 Evaluation model performance

This assignment is a binary classification problem, meaning that the model should be evaluated based on accuracy instead of 'RMSE' which is used for regression problems. As the dataset is imbalanced, the 'Accuracy' metric is not the best metric to evaluate the model with. Therefore the 'Kappa' value and the 'ROC-curve' will used to select the best performing model.

6.1 ConfusionMatrix results

After training the model, the performance will be checked on the test data. This will be done via the 'Confusion-Matrix' combined with the 'predict' functionality. The results will be evaluated per algorithm.

6.1.1 svmRadial

```
confusionMatrix(predict(model_list$svmRadial, x_test, type = "raw",
return_table = TRUE, dnn = c("Predicted", "Target")),
x_test$giving_product)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No   22  74
##        Yes  30 116
##
##                Accuracy : 0.57
##                  95% CI : (0.505, 0.633)
##     No Information Rate : 0.785
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.026
##
##  Mcnemar's Test P-Value : 2.48e-05
##
##             Sensitivity : 0.4231
##             Specificity : 0.6105
##          Pos Pred Value : 0.2292
##          Neg Pred Value : 0.7945
##              Prevalence : 0.2149
##          Detection Rate : 0.0909
##    Detection Prevalence : 0.3967
##       Balanced Accuracy : 0.5168
##
##        'Positive' Class : No
##
```

The ‘svmRadial’ has accuracy of 0.6 and a Kappa of 0.021.

6.1.2 Multinom

```
confusionMatrix(predict(model_list$multinom, x_test, type = "raw"),
x_test$giving_product)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No   24  74
##        Yes  28 116
##
##                Accuracy : 0.579
##                    95% CI : (0.514, 0.641)
##       No Information Rate : 0.785
##       P-Value [Acc > NIR] : 1
##
##                     Kappa : 0.055
##
##    Mcnemar's Test P-Value : 8.36e-06
##
##               Sensitivity : 0.4615
##               Specificity : 0.6105
##            Pos Pred Value : 0.2449
##            Neg Pred Value : 0.8056
##                Prevalence : 0.2149
##            Detection Rate : 0.0992
##      Detection Prevalence : 0.4050
##         Balanced Accuracy : 0.5360
##
##          'Positive' Class : No
##
```

The ‘multinom’ algorithm has a lower accuracy with 0.57 and a higher Kappa with 0.066.

6.1.3 RF

```
confusionMatrix(predict(model_list$rf, x_test, type = "raw"),
x_test$giving_product)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No    6  19
##        Yes  46 171
##
##                  Accuracy : 0.731
##                    95% CI : (0.671, 0.786)
##       No Information Rate : 0.785
##       P-Value [Acc > NIR] : 0.98067
```

```
##
##                   Kappa : 0.019
##
##  Mcnemar's Test P-Value : 0.00126
##
##             Sensitivity : 0.1154
##             Specificity : 0.9000
##          Pos Pred Value : 0.2400
##          Neg Pred Value : 0.7880
##              Prevalence : 0.2149
##          Detection Rate : 0.0248
##    Detection Prevalence : 0.1033
##       Balanced Accuracy : 0.5077
##
##        'Positive' Class : No
##
```

The algorithm ‘rf’ has a relatively high accuracy with 0.71 and a low Kappa with 0.004.

6.1.4 Adaboost

```
confusionMatrix(predict(model_list$adaboost, x_test, type = "raw"),
x_test$giving_product)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No    3   9
##        Yes  49 181
##
##                Accuracy : 0.76
##                  95% CI : (0.701, 0.813)
##     No Information Rate : 0.785
##     P-Value [Acc > NIR] : 0.845
##
##                   Kappa : 0.014
##
```

```
##  Mcnemar's Test P-Value : 3.04e-07
##
##             Sensitivity : 0.0577
##             Specificity : 0.9526
##          Pos Pred Value : 0.2500
##          Neg Pred Value : 0.7870
##              Prevalence : 0.2149
##          Detection Rate : 0.0124
##    Detection Prevalence : 0.0496
##       Balanced Accuracy : 0.5052
##
##        'Positive' Class : No
##
```

The ‘adaboost’ model has a high acuracy with 0.74 and a negative Kappa with -0.04.

6.1.5 xgbDart

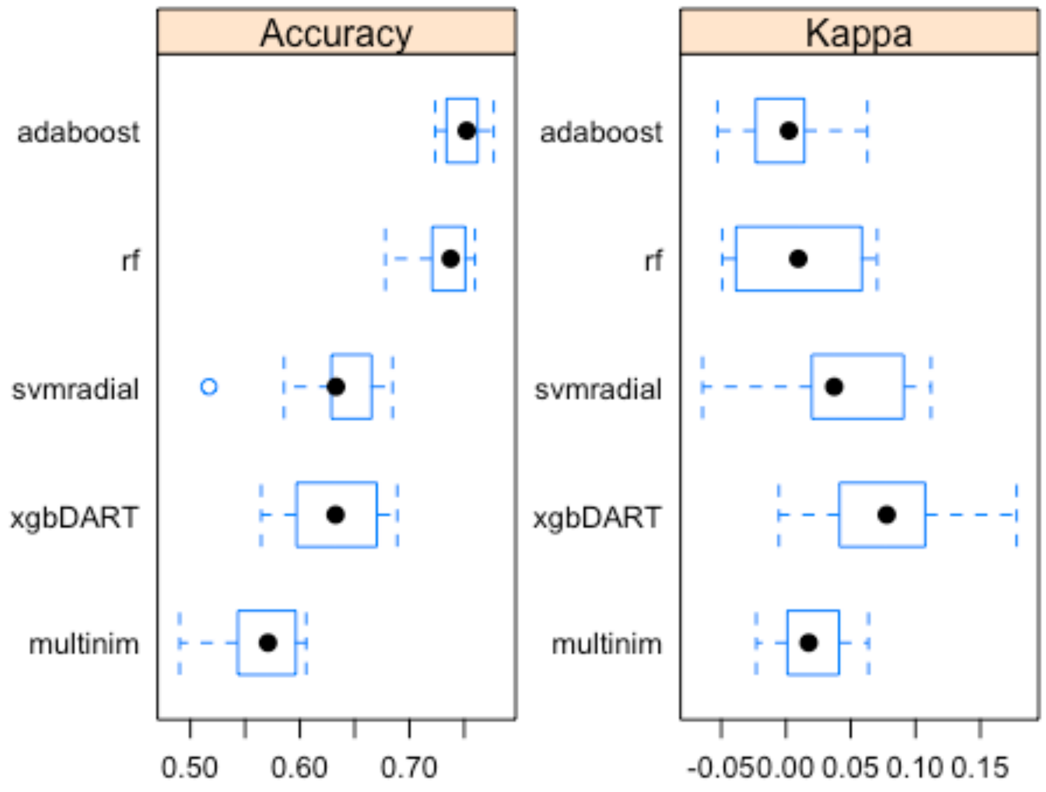
```
confusionMatrix(predict(model_list$xgbDART, x_test, type = "raw"),
x_test$giving_product)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No   18  46
##        Yes  34 144
##
##                Accuracy : 0.669
##                  95% CI : (0.606, 0.728)
##     No Information Rate : 0.785
##     P-Value [Acc > NIR] : 1.000
##
##                   Kappa : 0.096
##
##  Mcnemar's Test P-Value : 0.219
##
##             Sensitivity : 0.3462
##               Specificity : 0.7579
##            Pos Pred Value : 0.2812
##            Neg Pred Value : 0.8090
##                Prevalence : 0.2149
##            Detection Rate : 0.0744
##      Detection Prevalence : 0.2645
##         Balanced Accuracy : 0.5520
##
##          'Positive' Class : No
##
```

The best performing model when looking at Kappa is 'xgbDart' with an accuracy of 0.63 and a Kappa value of 0.05.

To visualise the model performance, a 'list' will be created and used as input for the 'bwplot.

```
model_overview <- list(multinim = model_list$multinom, rf =
model_list$rf, svmradial = model_list$svmRadial, adaboost =
model_list$adaboost, xgbDART = model_list$xgbDART)
res <- resamples(model_overview)
scales <- list(x =list(relation="free"), y=list(relation="free"))
bwplot(res, scales=scales)
```



After visualising the results, it makes more clear that the ‘xgbDart’ is the best performing model with the highest Kappa-value and also a not to low accuracy.

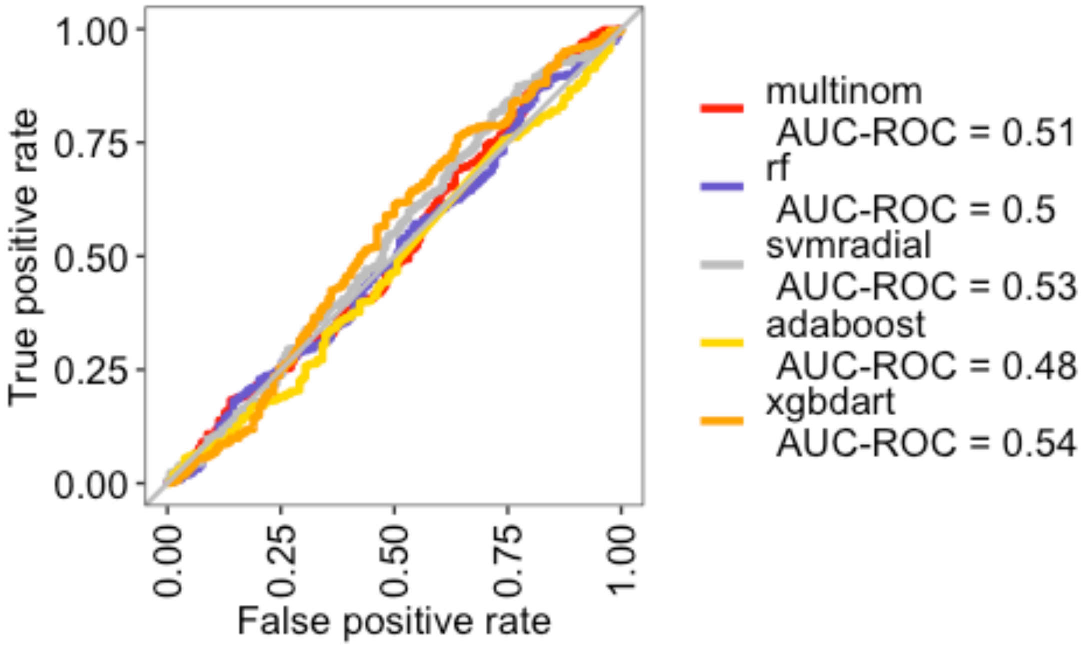
6.2 ROC-curve

The results of the Confusion-matrix and the ‘bwplot’ gave already a good view on the best performing model. A last evaluation will be done with the ROC-curve. The ROC-curve shows the trade-off between specificity and sensitive. Classifiers that give curves closer to the top-left corner, indicating a better performance (source).

The ROC-curve is created by first adding the different models to a list via the ‘evalm’ function.

```
library(MLeval)
evalm(list(model_list$multinom, model_list$rf, model_list$svmRadial,
model_list$adaboost, model_list$xgbDART), gnames = c('multinom', 'rf',
'svmradial', 'adaboost', 'xgbdart'))
```

```
## ***MLeval: Machine Learning Model Evaluation***

## Input: caret train function object
```



A good performing model gives a curve closer to the top-left corner. Analysing the ROC-curve gives quite a similar view as the results coming from ConfusionMatrix results, also here the 'xgbDart' is the best performing model but the 'multinom' model comes quite close.

6.3 Model improvement with stacking

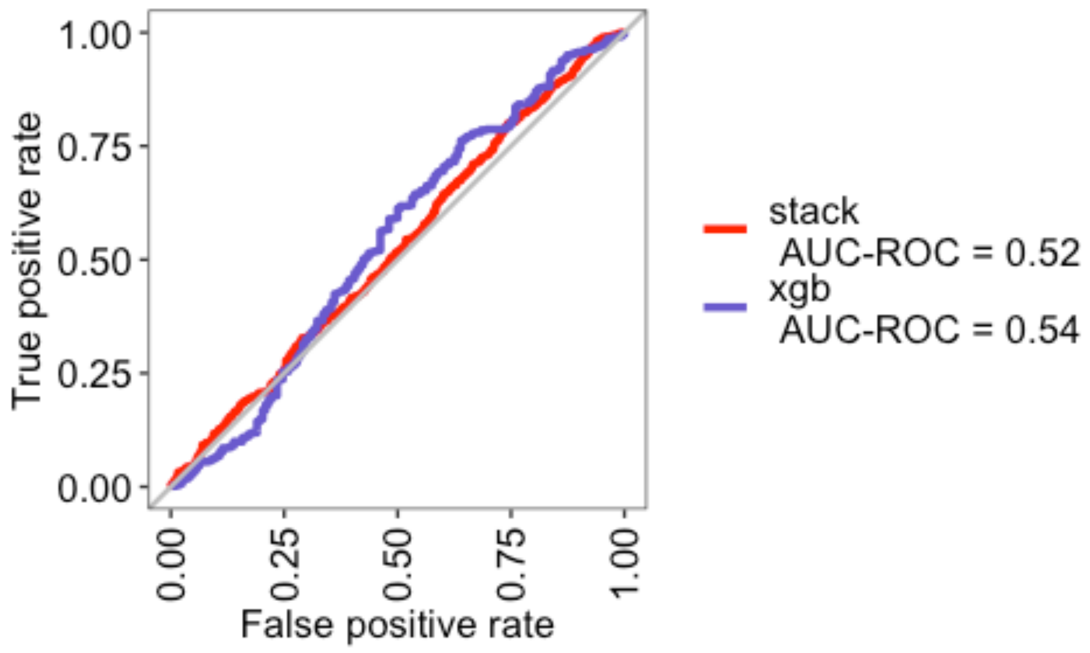
Stacking (source) is a technique to combine different models together to improve the performance of the algorithm. Also this functionality is part of the 'Caret' package. In this case, the 'caretstack' function is used to combine the different algorithms. The previous models trained, with 'model_list' will be taken as input.

```
stackcontrol <- trainControl(method = "repeatedcv", number = 10,
repeats = 3, savePredictions = TRUE, classProbs = TRUE)
stack_model <- caretStack(model_list, method="glm", metric="Kappa",
trControl = stackcontrol)
print(stack_model)
```

```
## A glm ensemble of 5 base models: multinom, rf, svmRadial, adaboost,
xgbDART
##
## Ensemble results:
## Generalized Linear Model
##
## 2665 samples
##    5 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 2399, 2398, 2399, 2399, 2398, 2399, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.78      0
```

The accuracy goes up, but Kappa scores a 0. Let's compare the ROC-curve of the stacked model with the best performing model, 'xgbDart'.

```
evalm(list( stack_model$ens_model, model_list$xgbDART), gnames =
c('stack', 'xgb'))
```

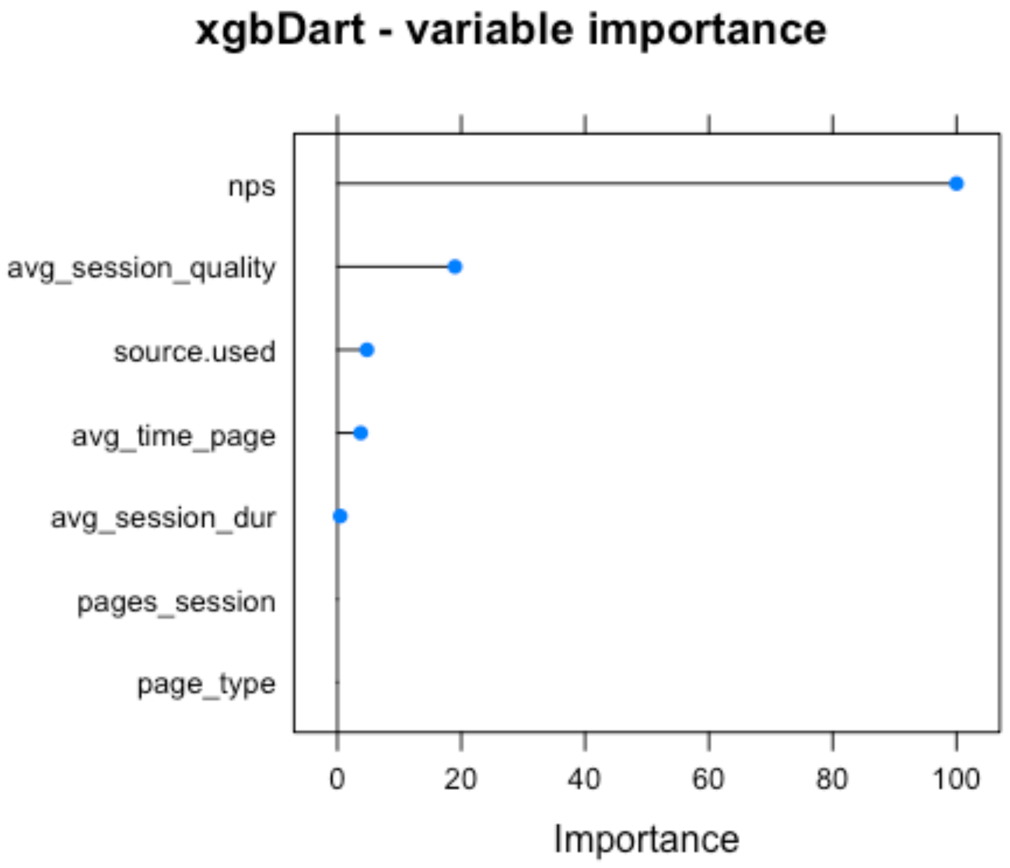


The ‘stacked’ model doesn’t perform better then the ‘xgbDart’ model. Therefore the analysis will continue with the ‘xgbDart’ model.

6.4 Variable importance

Now that the right model is choses, let’s have a look at the variable importance via the ‘varImp’ functionality.

```
plot(varImp(object= model_list$xgbDART), main="xgbDart - variable
importance")
```



It seems that ‘NPS’ is the most important variable for predicting buying behaviour, follow by the ‘average sessions quality’ and ‘average time on page’. This is more or less inline also with the analysis that was already being done on the different features.

6.5 Effect of features on predicted outcome

Now that the right model and the most important variables are known, last exercise is to understand the marginal effect that these variables have on the predicted outcome. In the ‘variable plot’ above was to see that the most important variables are 1) NPS, 2) average sessions quality and 3) average time on page. These variables will be deeper analysed

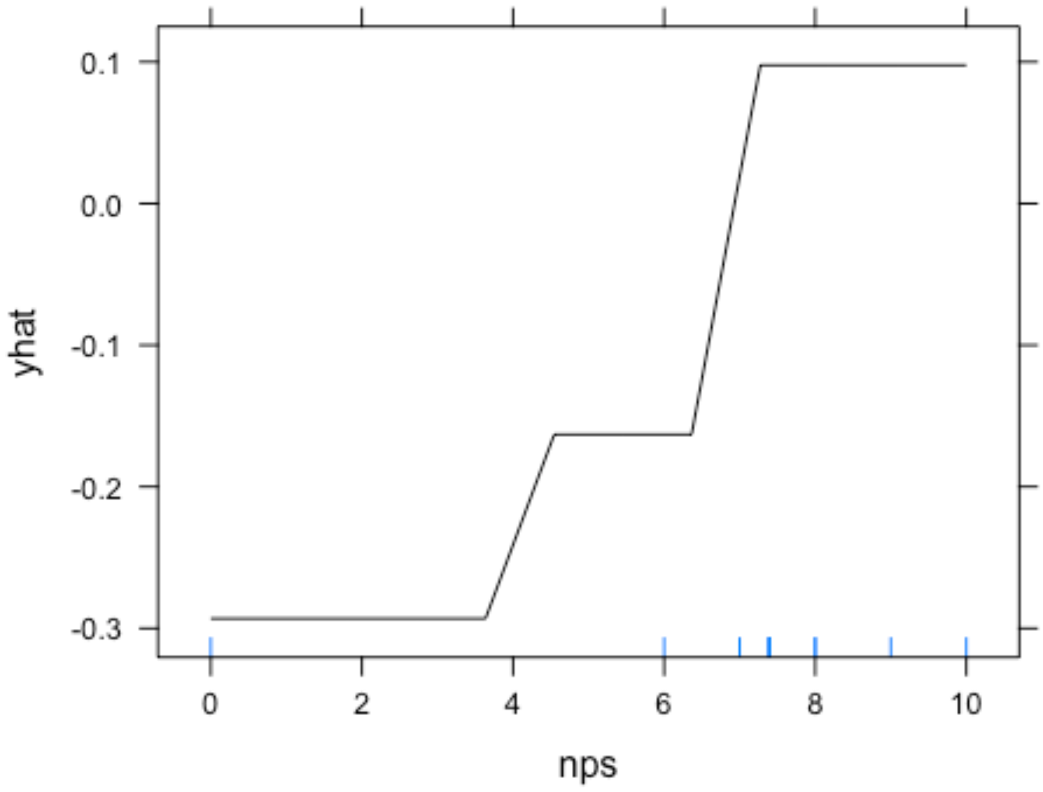
First a new model will be trained but now only with ‘xgbDart’ model. Based on that model, a plot will be created with the ‘partial’ function that creates a plot that visualises the probability of buying behaviour of that feature.

```
model_x <- train(giving_product ~ ., data = x_train, method =
"xgbDART", metric = "Kappa", trControl = my_control)
```

6.5.1 NPS impact

Based on the model above only trained with the ‘xgbDart’ algorithm, a plot will be created understanding the effect of NPS on the marginal effect on buying behavior.

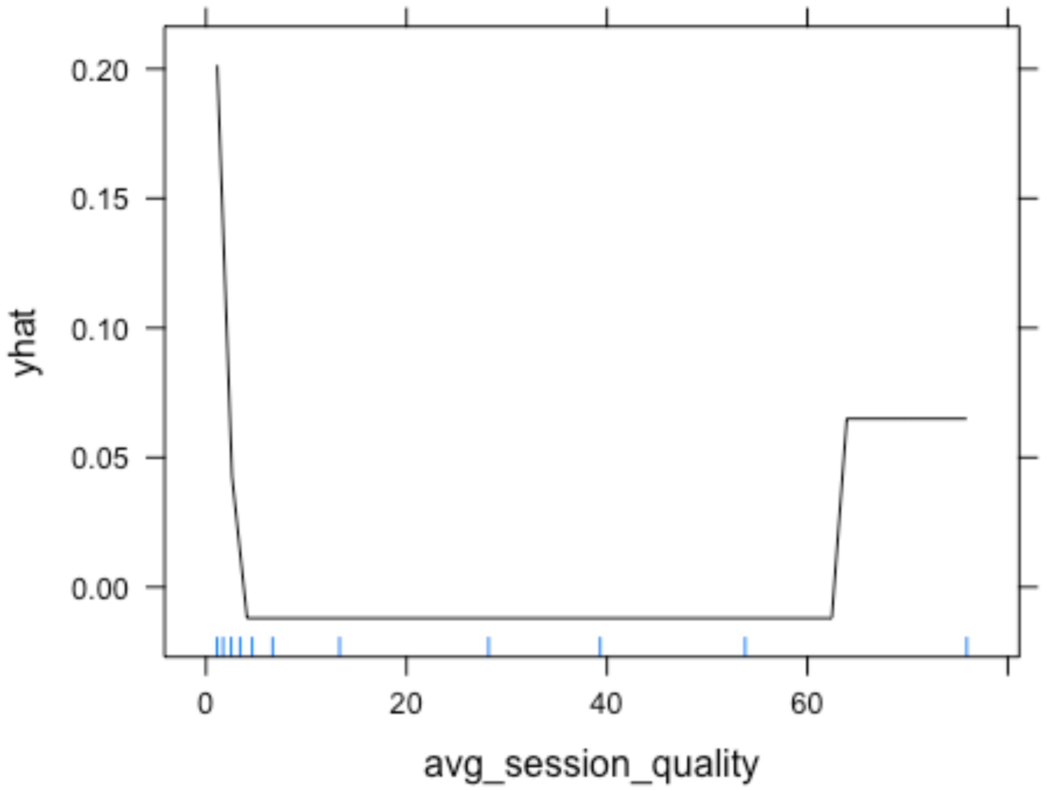
```
plot_modelx <- partial(model_list$xgbDART, pred.var = c("nps"), probs
= T, which.class = 2, plot = TRUE, rug = TRUE)
plot_modelx
```



In this plot is to see that the marginal effect goes up once the NPS-score is increasing. Higher NPS-scores means a higher marginal effect on buying behaviour.

6.5.2 Average session quality

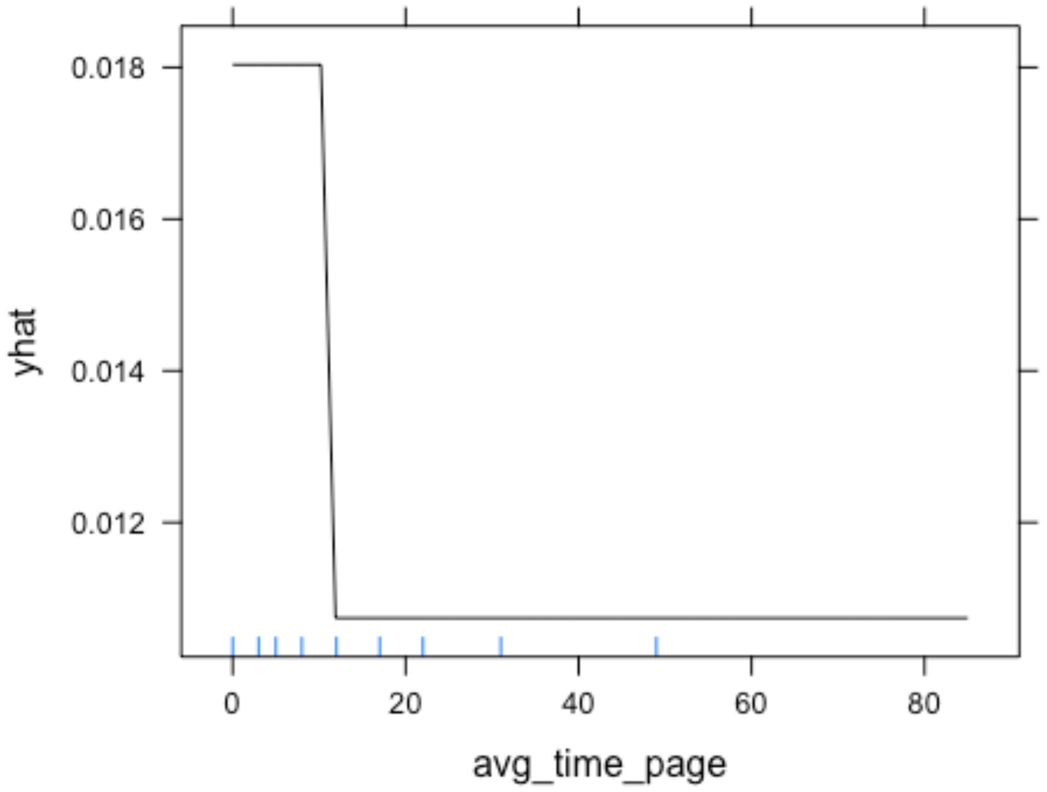
```
plot_modelx <- partial(model_list$xgbDART, pred.var =
"avg_session_quality", probs = T, trim.outliers = TRUE, which.class =
2, plot = TRUE, rug = TRUE)
plot_modelx
```



For the ‘avg_session_quality’ the result is different then for the ‘NPS-score’. Here there is not a clear linear effect between variable and marginal effect. It seems that with lower quality scores the marginal effect is higher, this is changing once the score is going up.

6.5.3 Average time on page

```
plot_modelx <- partial(model_list$xgbDART, pred.var = "avg_time_page",
probs = T, trim.outliers = TRUE, which.class = 2, plot = TRUE, rug =
TRUE)
plot_modelx
```



Also the ‘average time on page’ shows that within the lower time spend on pages, a correletion is visible with the marginal effect is has.

6.6 Conclusion

With different evaluation methods used, the right model is chosen which is the ‘xgbDart’. As the dataset is imbalanced it was much more difficult to choose the right model as accuracy is less relevant. Therefore the ‘Kappa value’ and the ‘ROC-curve" where more important the the ’Confusion-Matrix’ results. The ‘xgbDart’ model leverages ‘Gradient Boosting’ (source). The general idea behind this is that instances, which are hard to predict correctly (“difficult” cases) will be focused on during learning, so that the model learns from past mistakes. The disadvantage of this model that is relatively slow.

With the use of the right model, the most important variables where being searched for. In the 'variable importance' plot was to see that the 'NPS-score' was the most powerful variables, followed by the 'session quality' and 'average time on page'. With the use of the 'partial' function that visualises the marginal effect on the variable on buying behaviour, very clearly was to see that only the 'NPS-score' is the feature that can influences buying behaviour.

7 Final conclusion

The main question of this assignment is, **how can we predict buying behaviour?** To answer that question a 'Exploratory Data Analysis' has being done to get already a first view on the features and their chances to be the most important predictor. Already from this analysis, the 'NPS' score seemed to be the best candidate. After testing different models and selecting the right model, which was more difficult as the dataset is imbalanced, the 'xgbDart' model seems to best performing model with an accuracy of 0.63 and a Kappa value of 0.05. Also 'stacking' was being used to see if the performance could increase with the use of different models together. But based on the 'ROC-curve' comparing the two models, the 'xgbDart' was still the best performing model.

In the 'variable importance plot' is very clearly to see that 'NPS' is the most important variable which has a linear marginal effect on the outcome. The other variables seemed to be less effective after further analysis. The recommendation here to first start understanding how to improve the NPS-scores and using Machine Learning algorithm understanding the effects. Further recommendation would be to increase the amount of data and ensuring data quality to also see the effect of the other potential predictors of buying behaviour.