Kickstarter Project

Data Science: Capstone

Rémi Fauve

2020-06-17

Contents

Preamble

This study was done as part of the HarvardX online course "Data Science: Capstone" (PH125.9x).

The data set used for this project was created by Mickaël Mouillé (License CC BY-BC-SA 4.0), and is available on Kaggle at this link.

The following libraries are required to execute the code of this report:

```
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if (!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if (!require(lubridate))
  install.packages("lubridate", repos = "http://cran.us.r-project.org")
if (!require(rpart))
  install.packages("rpart", repos = "http://cran.us.r-project.org")
if (!require(rpart.plot))
  install.packages("rpart.plot", repos = "http://cran.us.r-project.org")
if (!require(kableExtra))
  install.packages("kableExtra", repos = "http://cran.us.r-project.org")
if (!require(doParallel))
  install.packages("doParallel", repos = "http://cran.us.r-project.org")
if (!require(gridExtra))
  install.packages("gridExtra", repos = "http://cran.us.r-project.org")
#(.packages())
```

1 Introduction

Crowdfunding is a common practice nowadays, but the first website to engage in this business model emerged less than 20 years ago (ArtistShare in 2001). More crowdfunding sites appeared on the web, the most famous ones being the American companies IndieGoGo (2008), Kickstarter (2009) or GoFundMe (2010). Crowdfunding grew quickly into a fully mature form of alternative finance, cumulating over 34 billion USD raised worldwide in 2015. The popular success of crowdfunding is due to the simplicity for both projects' creators and backers, in contrast with the administrative burden of traditional investment strategies.

However, the drawback of becoming a common practice is that the supply (the backers) cannot match the demand (the projects' creators): not every submitted project can be funded. Projects' creators being therefore in competition, they now need to optimize the marketing side of their project to increase their chance of ending funded. This is where data science can provide quick tips, if not a model to forecast the pledged money, or even better, the optimal settings for a maximal gain.

In this study, data on all the projects registered on the Kickstarter plateform, up to 2018, is imported and explored to extract general insights on the success rate or the amount of pledged money depending on the different variables in the data. Then, different machine learning models are trained and their performance on the prediction of the pledged money per project are compared.

2 Data importation

2.1 Raw data description

The data is accessible on Kaggle, but Kaggle's API appears to only work with Python. The archived data is then manually downloaded, stored and unzipped in the "./DATA/orig" repertory. The original data set is formated as a CSV file. It is imported in RStudio as a dataframe, with blank cells replaced with *NA*.

```
data_ks <-
  read.csv(file = "./DATA/orig/ks-projects-201801.csv",
           na.strings = c("", " ", "NA"))

nrow(data_ks)
```

```
## [1] 378661
```

```
ncol(data_ks)
```

```
## [1] 15
```

The dimensions of the data are, as expected, around 379,000 rows and 15 columns, so the data has been correctly imported. Another way to test if the data is corrupted is to investigate the number and the nature of unique values for variables that are supposed to display a very limited number of different values, as shown below (for example, the variables `state` or `currency` should only contains specific strings).

Before searching for insight in the data set, each variable needs to be checked for errors, inconsistencies or *NAs*.

2.2 Integrity check

Every variable is checked.

2.2.1 `ID`

The `ID` variable should have a distinct value for each row, as each row should be a different project. No further checking is required for this variable.

```
# no NA
anyNA(data_ks$ID)
```

```
## [1] FALSE
```

```
# does every row have a different ID ?
length(unique(data_ks$ID)) == nrow(data_ks)
```

```
## [1] TRUE
```

2.2.2 `name`

The `name` variable should only contain characters. 4 projects with missing names are discarded.

```
# no NA
anyNA(data_ks$name)
```

```
## [1] TRUE
```

```
sum(is.na(data_ks$name))
```

```
## [1] 4
```

```r
# NA discarded
data_ks <- data_ks %>%
  filter(!is.na(name))
```

It appears that around 5,000 projects have same names.

```r
# does every row have a different name ?
length(unique(data_ks$name)) == nrow(data_ks)
```

```
## [1] FALSE
```

```r
dupl_names <- as.data.frame(sort(unique(data_ks$name[duplicated(data_ks$name)])))
colnames(dupl_names) <- "name"

# number of projects with same names
data_ks %>%
  mutate(dupl = (name %in% dupl_names$name)) %>%
  group_by(dupl) %>%
  summarise(nb = n())
```

```
## # A tibble: 2 x 2
##   dupl      nb
##   <lgl>  <int>
## 1 FALSE 373536
## 2 TRUE    5121
```

To check if projects with the same name are different projects, their launch date are compared. It appears that every project with the same name have a different launch date, so they can be considered as different projects.

```r
date_check <- data_ks %>%
  filter(name %in% dupl_names$name) %>%
  group_by(name) %>%
  mutate(diff_launched = all(!duplicated(launched)))

# does every project with a duplicated name have a different launch date ?
all(date_check$diff_launched)
```

```
## [1] TRUE
```

2.2.3 `main_category` and `category`

The `main_category` and `category` variables should only contains specific strings, which can be later treated as factors. Different projects can very well be within the same `main_category` and `category`. Each of the 170 combinations of `main_category` and `category` are examined and no inconsistencies have been noticed.

```r
# no NA
anyNA(data_ks$category)
```

```
## [1] FALSE
```

```r
anyNA(data_ks$main_category)
```

```
## [1] FALSE
```

```r
# number of distinct categories
nrow(data_ks %>%
  select(main_category, category) %>%
```

```
  group_by(main_category) %>%
  filter(!duplicated(category)))
```

```
## [1] 170
```

```
# character to factors
data_ks$category <- as.factor(data_ks$category)
data_ks$main_category <- as.factor(data_ks$main_category)
```

2.2.4 launched and deadline

The launched and deadline variables should only contain strings that can be converted to dates. Of course, the deadline should always be a later date than the launch date (launched).

```
# no NA
anyNA(ymd_hms(data_ks$launched))
```

```
## [1] FALSE
```

```
anyNA(ymd(data_ks$deadline))
```

```
## [1] FALSE
```

```
# is the deadline strictly superior to the launch date for every project?
all(ymd(data_ks$deadline) > ymd_hms(data_ks$launched))
```

```
## [1] TRUE
```

```
# character to date (hours, minutes and seconds are not relevant information)
data_ks$launched <- date(ymd_hms(data_ks$launched))
data_ks$deadline <- ymd(data_ks$deadline)
```

2.2.5 goal and pledged

The goal and pledged variables should only contain null or positive numbers.

```
# no NA
anyNA(data_ks$goal)
```

```
## [1] FALSE
```

```
anyNA(data_ks$pledged)
```

```
## [1] FALSE
```

```
all(data_ks$goal > 0)
```

```
## [1] TRUE
```

```
all(data_ks$pledged >= 0)
```

```
## [1] TRUE
```

2.2.6 country and currency

The country and currency variables should only contain specific strings, which can be later treated as factors. The error message N,0" appears on around 3,700 projects in the country variable, but it can be corrected using currency as a hint, each country having their own national currency (except European countries). The 186 remaining projects with missing country are discarded.

```r
# no NA
anyNA(data_ks$country)
```

```
## [1] FALSE
```

```r
anyNA(data_ks$currency)
```

```
## [1] FALSE
```

```r
# number of projects with N,0\" error
nrow(data_ks %>%
  filter(country == "N,0\""))
```

```
## [1] 3797
```

```r
currencies <- as.data.frame(data_ks %>%
  filter(country != "N,0\"") %>%
  select(country, currency) %>%
  group_by(country) %>%
  filter(!duplicated(currency)) %>%
  arrange(currency) %>%
  ungroup())

# guessing country from currency (except for European countries)
for (i in 1:nrow(data_ks)) {
  if (data_ks$country[i] == "N,0\"" & data_ks$currency[i] != "EUR")
    data_ks$country[i] <- currencies[currencies$currency == data_ks$currency[i], 1]
}

# number of projects with N,0\" error
nrow(data_ks %>%
  filter(country == "N,0\""))
```

```
## [1] 186
```

```r
# only European countries can't be straightforwardly guessed
unique(data_ks$currency[data_ks$country == "N,0\""])
```

```
## [1] "EUR"
```

```r
# N,0\" discarded
data_ks <- data_ks %>%
  filter(country != "N,0\"")

# character to factors
data_ks$country <- as.factor(data_ks$country)
data_ks$currency <- as.factor(data_ks$currency)
```

2.2.7 state

The `state` variable should only contain specific strings, which can be later treated as factors.

```r
# no NA
anyNA(data_ks$state)
```

```
## [1] FALSE
```

```r
unique(data_ks$state)
```

```
## [1] "failed"     "canceled"   "successful" "live"       "undefined"
## [6] "suspended"
```

```
# character to factors
data_ks$state <- as.factor(data_ks$state)
```

2.2.8 backers

The backers variable should only contain positive integer numbers.

```
# no NA
anyNA(data_ks$backers)
```

```
## [1] FALSE
```

```
# is the number of backers superior or equal to 0 ?
all(data_ks$backers >= 0)
```

```
## [1] TRUE
```

2.2.9 usd.pledged, usd_pledged_real and usd_goal_real

The usd.pledged, usd_pledged_real and usd_goal_real variables should only contain positive numbers. In addition to missing values, a quick look into the USD converted amounts revealed some inconsistencies with the pledged values, and between usd.pledged and usd_pledged_real. These variables were added by the creator of this data set, using the Fixer.io API. The usd.pledged variable is then discarded.

```
# no NA
anyNA(data_ks$usd.pledged)
```

```
## [1] TRUE
```

```
anyNA(data_ks$usd_pledged_real)
```

```
## [1] FALSE
```

```
anyNA(data_ks$usd_goal_real)
```

```
## [1] FALSE
```

```
# many missing values
sum(is.na(data_ks$usd.pledged))
```

```
## [1] 3611
```

```
all(data_ks$usd_pledged_real >= 0)
```

```
## [1] TRUE
```

```
all(data_ks$usd_goal_real > 0)
```

```
## [1] TRUE
```

```
# usd.pledged discarded
data_ks <- data_ks %>%
  select(-usd.pledged)
```

2.3 Final variables

For these last editions of the data set, the goal is to dispose of a maximum of relevant variables for later data exploration.

2.3.1 Time spans

The `time_span`, in days, between the launch date and the deadline is calculated. The launch date and the deadline are renamed, for clarity sake.

```
data_ks <- data_ks %>%
  mutate(
    start_date = launched,
    end_date = deadline,
    time_span =
      round(as.numeric(interval(start_date, end_date), "days"), 0)) %>%
  select(-launched, -deadline)
```

It appears that some launch dates are corrupted, being set at 1970-01-01, which corresponds to the Unix epoch (date-time "0" for informatic time stamps). Also, the Kickstarter platform obviously didn't exist back in 1970 (created in 2009)... Thus, these corrupted `start_date` are discarded.

Additional variables could be looked at, such as:

- the day of the week of the launch date (resp. the deadline) (`start_wday`, resp. `end_wday`)
- the day of the month of the launch date (resp. the deadline) ? (`start_day`, resp. `end_day`)
- the month of the launch date (resp. the deadline) (`start_month`, resp. `end_month`)
- the quarter of the launch date (resp. the deadline) (`start_quarter`, resp. `end_quarter`)
- are both dates in different months ? (`diff_month`)
- are both dates in different quarters ? (`diff_quarter`)
- are both dates in different years ? (`diff_year`)

2.3.2 USD equivalents

Calculating a USD equivalent for the `goal` and `pledged` variables will allow easier comparisons. Because of the lack of information on the method used to calculate the `usd_pledged_real` and `usd_goal_real` variables (an account is required to use the Fixer.io API, as the creator of the original data set did), the exchange rates were rather extracted from this site as monthly averages [1] for comparison sake.

```
load(file = "./DATA/currency_rates.Rda")
```

It was decided to apply the rate of exchange to the goal and pledged money at the deadline date of each project, as it makes more sense to evaluate an amount of money once it has been gathered. Once the USD equivalents are calculated, the status of each project can be simply expressed as a new logical self-explaining `goal_reached` variable.

```
data_ks <- data_ks %>%
  rowwise() %>%
  mutate(goal_USD =
           goal * currency_rates[
             which(currency_rates$Date == floor_date(end_date,
                                                     unit = "month")),
             as.character(currency)],
         pledged_USD =
           pledged * currency_rates[
             which(currency_rates$Date == floor_date(end_date,
                                                     unit = "month")),
```

[1] A compromise had to be found between the high variation of exchange rates over time and the lack of information of the precise date for each pledging.

```
            as.character(currency)]) %>%
  ungroup()

data_ks <- data.frame(data_ks)

data_ks <- data_ks %>%
  mutate(goal_reached = pledged_USD >= goal_USD) %>%
  select(-goal, -pledged)

summary(data_ks$pledged_USD)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##        0       31      625     9063     4050 20338986
```

```
summary(data_ks$usd_pledged_real)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##        0       31      625     9063     4051 20338986
```

It seems that the values calculated here are very close to those obtained with the exchange rates from the Fixer.io API. Due to the lack of details on the method used to choose the exchange rates with the Fixer.io API, the former USD equivalents for the goal and the money pledged are discarded.

From now on, every money amount will be given in USD equivalent.

```
data_ks <- data_ks %>%
  select(-usd_pledged_real, -usd_goal_real)
```

2.3.3 Categories

Some `category` values can be attributed to different `main_category`, so it is their association that really carries the "category description" of a given project. Then, this "true" category description should be its own variable.

```
data_ks %>%
  filter(category == "Anthologies") %>%
  group_by(main_category) %>%
  filter(!duplicated(main_category)) %>%
  select(main_category, category) %>%
  knitr::kable("latex", align = "r") %>%
  kable_styling()
```

main_category	category
Comics	Anthologies
Publishing	Anthologies

```
# new variable
data_ks$true_category <- paste0(as.character(data_ks$main_category),
                                " | ",
                                as.character(data_ks$category))

data_ks$true_category <- as.factor(data_ks$true_category)
```

Additional variables could be looked at, such as:

- are `main_category` and `category` different ? (`diff_cat`)

2.3.4 State of the project

A project can be in an other state that the binary "successful" (goal reached) or "failed" (goal not reached), and when the states are compared with the variable `goal_reached`, it is clear that the state of the project does not only depend on the goal being reached or not, and that the `state` variable is not reliable.

```
table(data_ks %>%
  select(state, goal_reached))
```

```
##             goal_reached
## state         FALSE   TRUE
##   canceled    38073    697
##   failed     197693      6
##   live         2358    441
##   successful      5 133947
##   suspended    1552    292
##   undefined    1777   1623
```

The goal of this study is to predict the amount of money pledged by a project up to its deadline. Then, the `state` variable can be discarded, but the projects whose the deadline is prior to the creation of the original data set (2018-01-02) should also be discarded, as they are ongoing and may gather more money before their deadline.

```
data_ks <- data_ks %>%
  filter(end_date <= ymd("2018-01-02")) %>%
  select(-state)
```

2.3.5 Name

The name of the project is a major marketing hook. Its content and format may have an influence on the amount of pledged money. Here, only the format of the name of the project will be investigated, as the study of its content would require a lengthy semantic analysis.

Additional variables could be looked at, such as:

- is it an original name ? (`orig_name`)
- only lower (resp. upper) case letters ? (`name_lcase`, resp. `name_ucase`)
- number of words (`name_nb_word`)
- number of characters (`name_nb_char`)
- any special character (other than letters and numbers) ? (`name_spec_char`)

2.3.6 Non-relevant variables

The variable `ID` does not carry any relevant information, as it is just an index number. The variable `backers` also does not carry any relevant information, as the amount of money pledged by a single backer can vary significantly, and this information (amount per backer) is missing. Both variables are discarded.

```
data_ks <- data_ks %>%
  select(-ID, -backers)
```

3 Data exploration

Now that the data set is cleaned and consistent with its column names, it can be explored. The goal is to predict the amount of money one can expect to receive, regarding the caracteristics of said project.

3.1 Goal and pledged money

First, the distribution of the pledged money is investigated. More than 50,000 projects launched on Kickstarter since its creation received no money at all (around 14% overall), and around 7,500 projects received 1 USD or less (2%), but around 300 projects managed to gather more than 1 million USD (>0.1%).

```
# what percentage of projects received no money ?
sum(data_ks$pledged_USD == 0)
```

```
## [1] 51851
```

```
# how many projects received 1 USD or less ?
sum(data_ks$pledged_USD <= 1) - sum(data_ks$pledged_USD == 0)
```

```
## [1] 7688
```

```
# how many projects received more than 1 million USD ?
sum(data_ks$pledged_USD > 1e6)
```

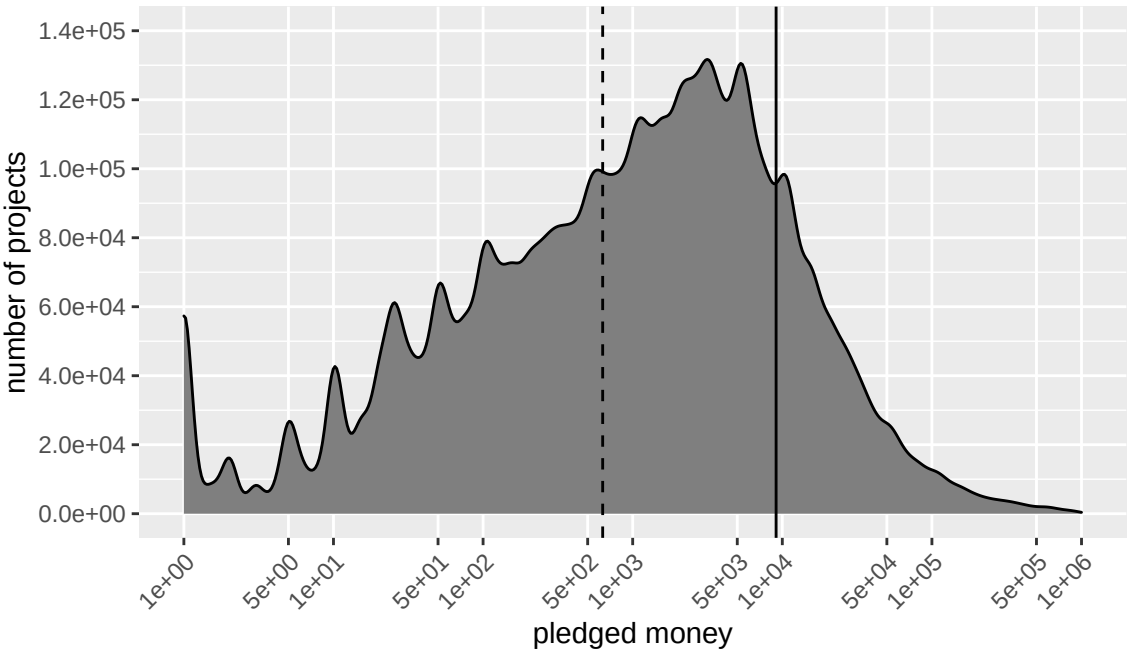
```
## [1] 276
```

```
# maximum amount pledged
max(data_ks$pledged_USD)
```

```
## [1] 20338986
```

By plotting the distribution density of the money pledged, it appears far from uniform (as expected). The median and the mean are represented respectively by the dashed and the solid black lines. The mean is shifted toward high values, since they are up to 4 orders of magnitude superior to the median. Since the values of the `goal_USD` and `pledged_USD` variables spread accross more than 7 orders of magnitude, the median gives a much more robust information about the distributions than the mean, which is more sensible to large values.

The distribution has an approximate bell shape, stretched toward the lower values. Local maxima can be interpreted as the rounded final amount of pledged money (like precisely 5 USD, 10 USD, 50 USD, 100 USD, etc.), when exactly the goal amount (which is generally a round value) is pledged, or when a single backer gave money (generally a round amount).



One obvious variable that may influence the amount of money pledged by a given project is its goal. Indeed, if the goal is set at 1,000 USD, why would anyone give more than a fraction of 1000 USD, or simply more one the goal is reached ? It is expected that the goal distribution will be shifted toward high values, since it may not be worth it to launch a crowdfunding for less than a thousand USD. Nontheless, around 6,000 projects had their goal set to 100 USD or less (1.6% of all projects), and around 1,000 to more than 1 million USD (0.2%).

```
# number of projects whose goal is 100 USD or less
sum(data_ks$goal_USD <= 100)
```

```
## [1] 6036
```

```
# number of projects whose goal is more than 1 million USD
sum(data_ks$goal_USD > 1e6)
```

```
## [1] 1077
```

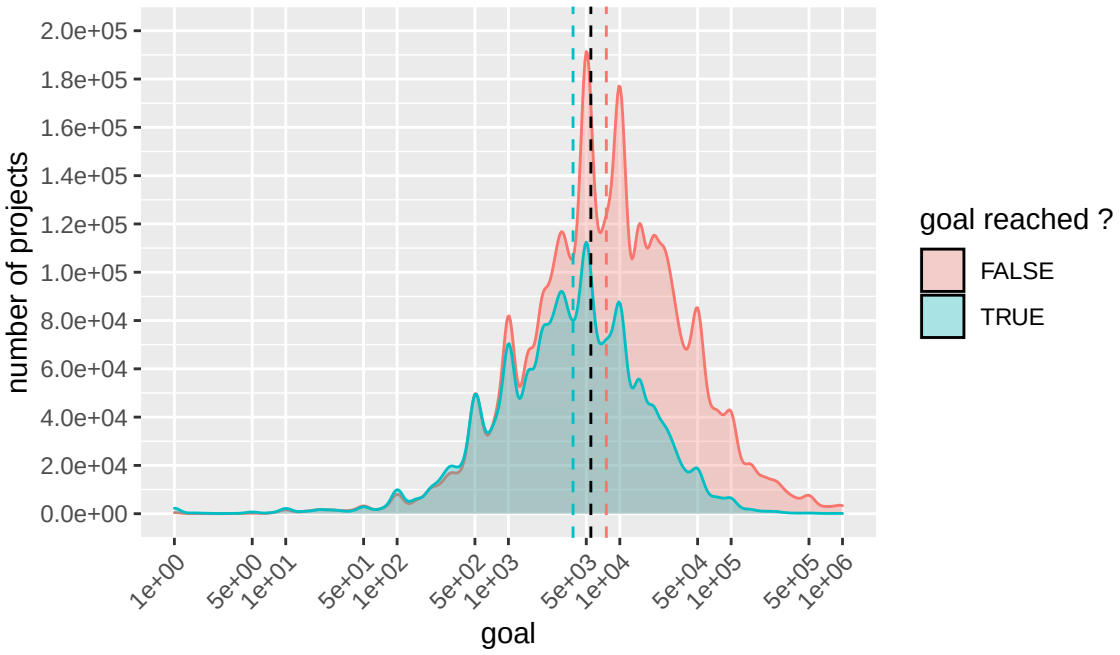
```
# percentage of successfull projects
mean(data_ks$goal_reached) * 100
```

```
## [1] 36.3722
```

```
# maximum goal amount
max(data_ks$goal_USD)
```

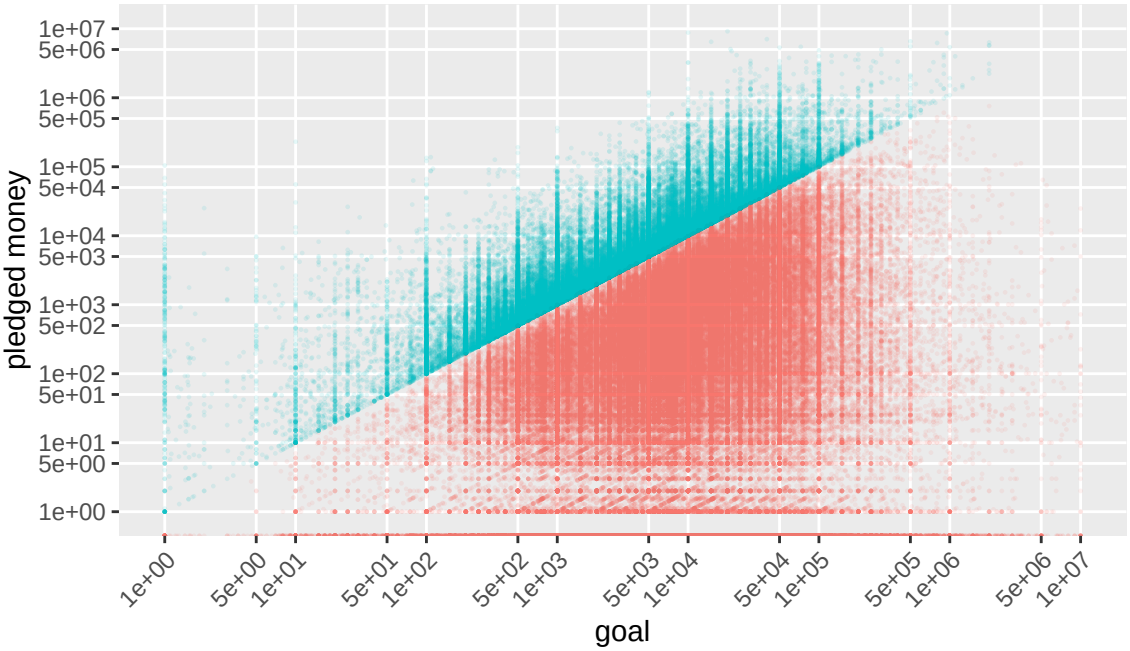
```
## [1] 165330000
```

The distribution of the goal is plotted with the distinction of wether the goal has been reached or not. The number of projects that fail to reach their goal is, overall, significantly higher than the number of projects that reached their goal. Regardless of that, the two distributions are quite similar (common local maxima), and their medians (colored dashed lines) present a ratio of around 2 (4,000 USD to 7,000 USD), with a global median (black dashed line) around 5,000 USD, and only seem close because of the logarithmic x-axis. The distribution of the goals which have been reached is shifted toward the low values, which is expected: the lowest the goal, the easiest it is to reach it. Local maxima generaly correspond to rounded up goal values.



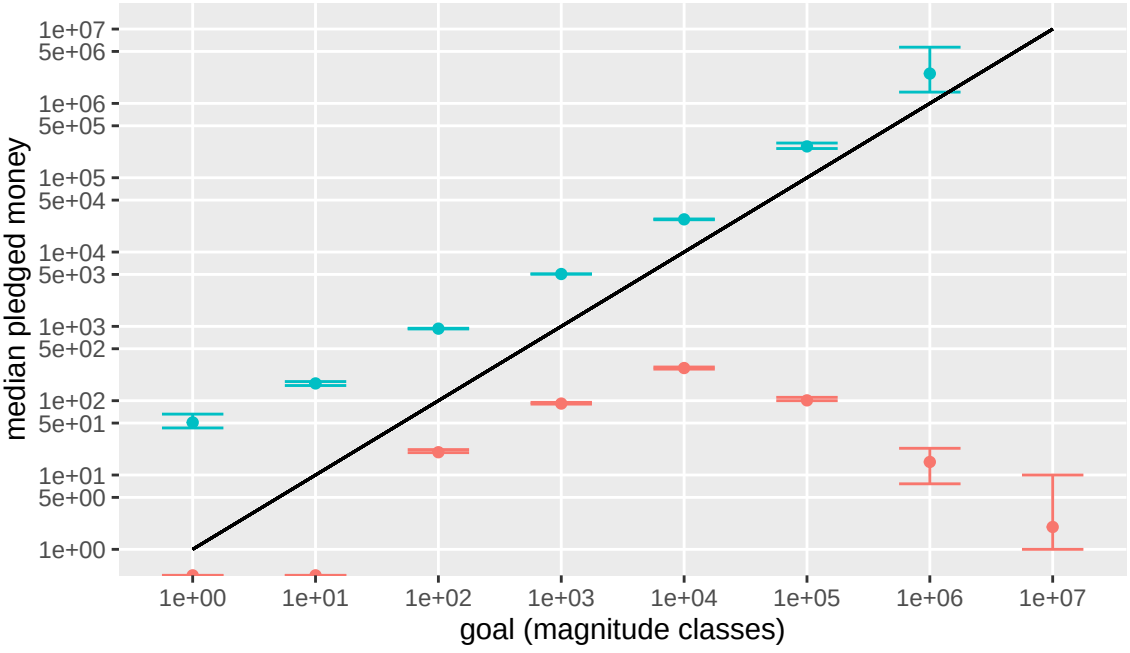
Both `goal_USD` and `pledged_USD` distributions are plotted one against the other in the following plot. This graph is very interesting, as it shows many different phenomena:

- the thin "red" line, on the very bottom border of the graph, represents the projects that received, no matter the goal, no money whatsoever.
- the points are distributed on vertical and horizontal lines, which is explained by people's tendency to give rounded up amount of money (100 USD rather than 100.02 USD, exchange rates set aside).
- when the goal is reached, the amount pledged is then on or above the first bisector (by construction).
- many projects received way more money than their goal (for example, projects with goals at 1 USD receiving up to 100,000 USD !).
- a non-obvious gap is noticeable just below the first bisector. This may be explained by people's increased will to donate money if it allows the project to reach its goal.
- by looking at the "dense" part in the middle of the graph, it appears that many projects tend to receive between 10 and 100,000 USD, for goals set between 100 and 100,000 USD.

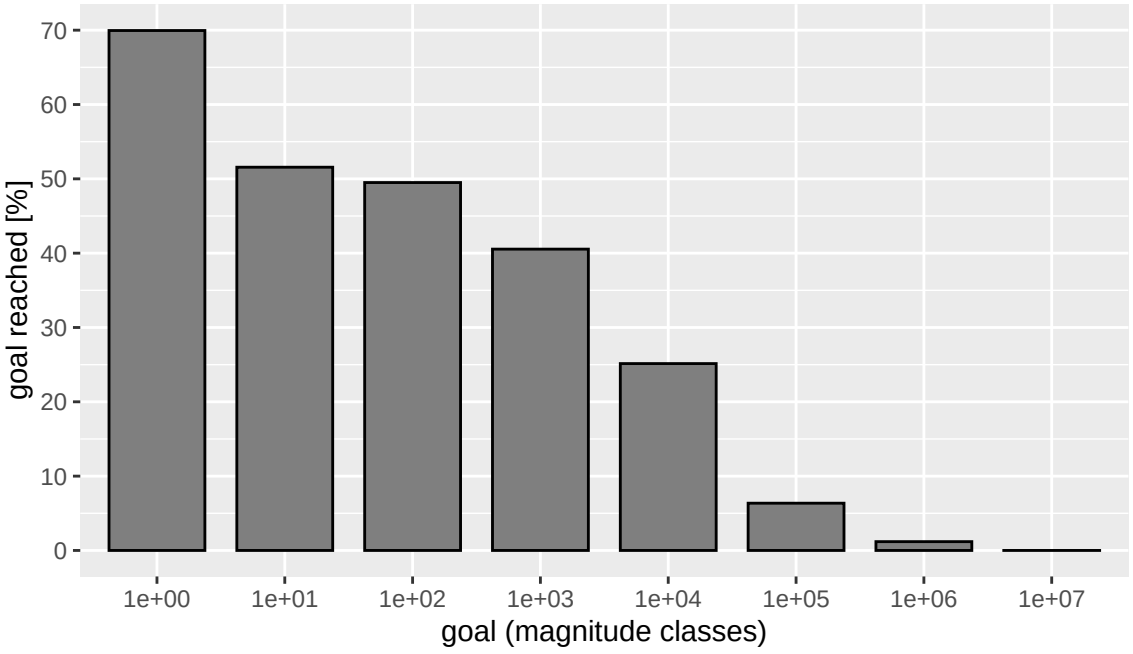


To refine our insights, the `goal_USD` variable are distributed in magnitude classes. The median of the pledged money per goal class is calculated, along its 95% confidence interval, and plotted below. The black line corresponds to the first bisector. This graph is very insightful, when combined with the previous graph. Up to a goal of 100 USD, one can expect to gain up to one more order of magnitude ! Also, projects with a goal of 10 USD or less, are either completely funded, or not at all.

Regarding the projects that did not reach their goal, and assuming the annonced goal is the only influent variable, one may be tempted to advise futur projects' creator to set their goal at 10,000 USD, as it appears to be the case where the highest amount of money is gathered, the goal being reached or not. Obviously, there are other influent parameters, and, as plotted in the previous graph, plenty of projects with a 10,000 USD goal gathered way less than 100 USD, if no money at all.



Even if the `goal_reached` variable is not a directly relevant variable for this study (because, it is not a parameter one can set before launching a crowdfunding), it is important to notice that the percentage of projects whose goal was reached decreases when the goal increases (as plotted in the graph below).



3.2 Name

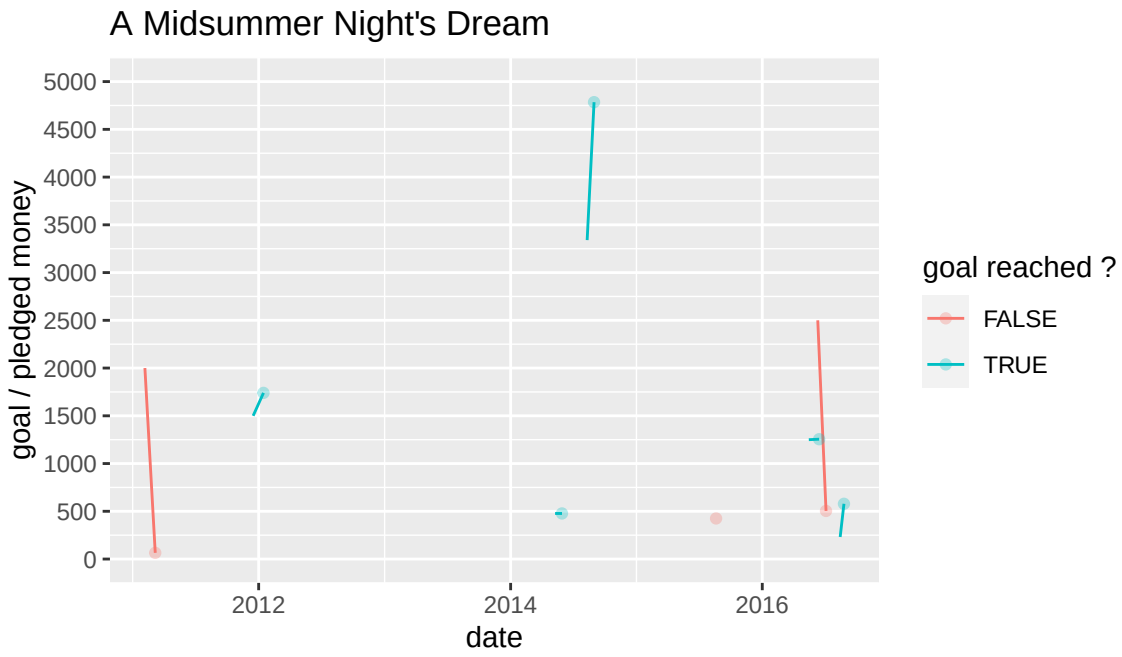
The name of the project is a major marketing hook, but running a semantic analysis to identify the best way to name your project is a whole project on its own. So in this study, we are focused only on its format.

3.2.1 Original project

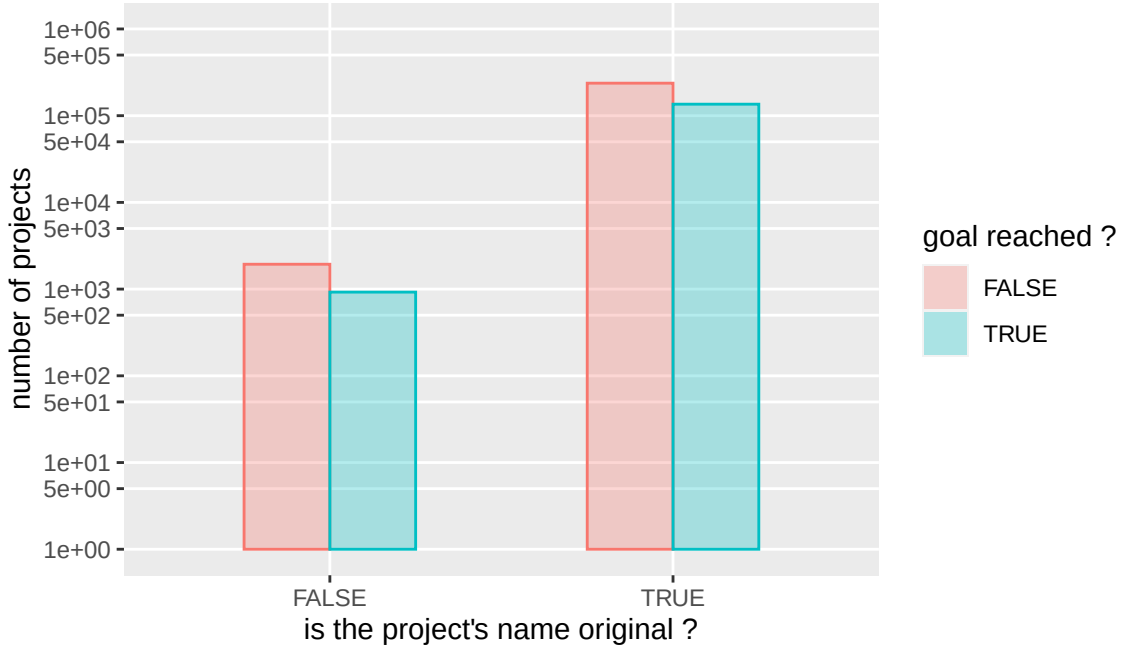
Before going into the format investigation, it appears that some project have exactly the same name, and it may be because they are either part of a series of periodic projects, or because they are multiple attempts to get funded for the same idea. This variable and its influence on the money pledged for each project is very difficult to explore, without doing it on a case by case basis.

For example, 8 projects, from 2011 to 2016, with the name "A Midsummer Night's Dream", plotted each by a segment, in the graph below, going from the goal at the launch date, to the pledged amount at the deadline (point). It is far from obvious to get general insights from this kind of information. One can assume that these projects belong to a series, but information is lacking about the author (same or different for each attempt ?), and the date distribution is not regular enough to imply that these projects only are part of a periodic series.

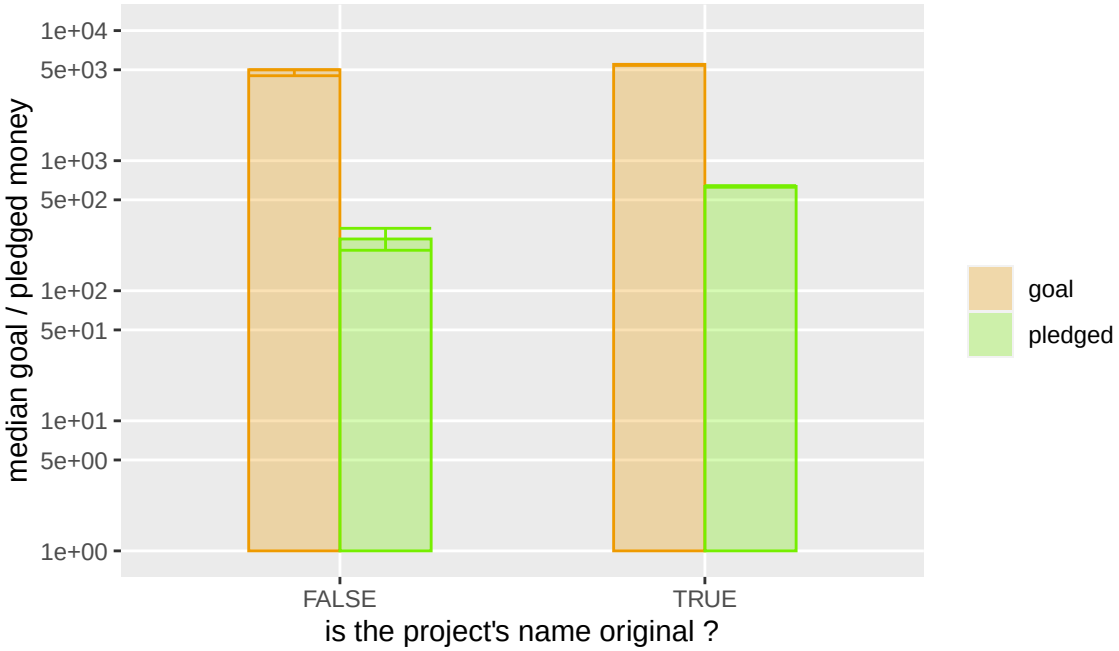
```
## Warning: Removed 1 rows containing missing values (geom_segment).
```



The `orig_name` variable is very case depend, and the general insights obtained from this variable are very tenuous. Only a few thousand projects (around 0.7% overall) have an unoriginal name, and their ratio of goal reached is approximately the same as as projects with unique names.



However, the median pledged money for projects with unoriginal names is around half the median pledged money for projects with unique names, while the median goal is roughly the same for both sides. But, because this variable is too much case dependant, it does not seem wise to draw a general insight from these statistics.

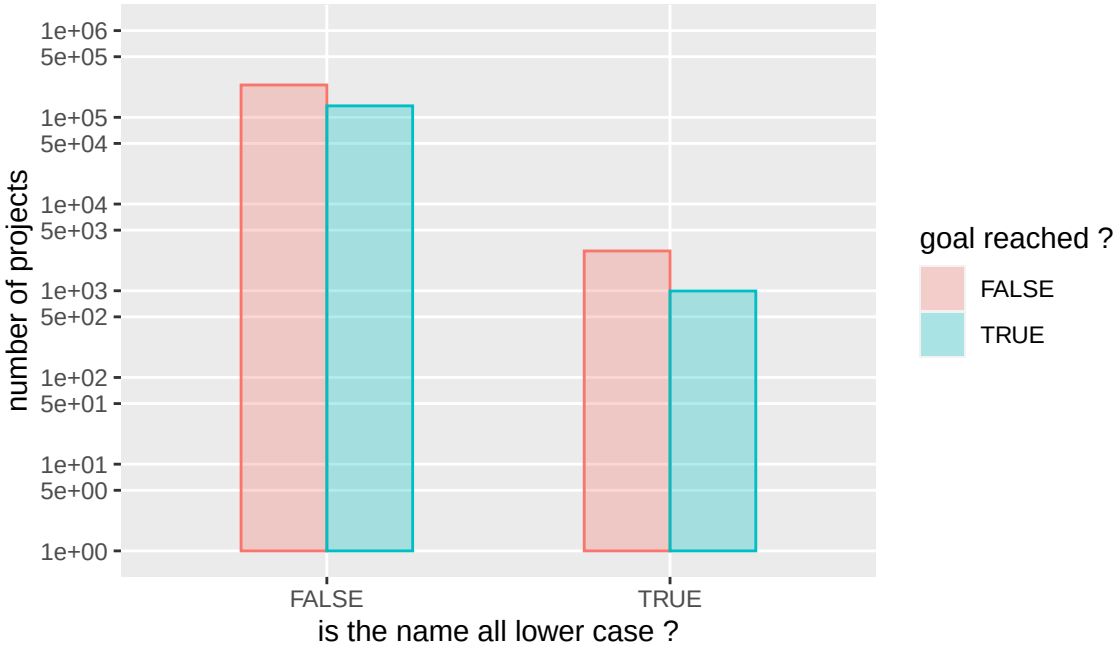


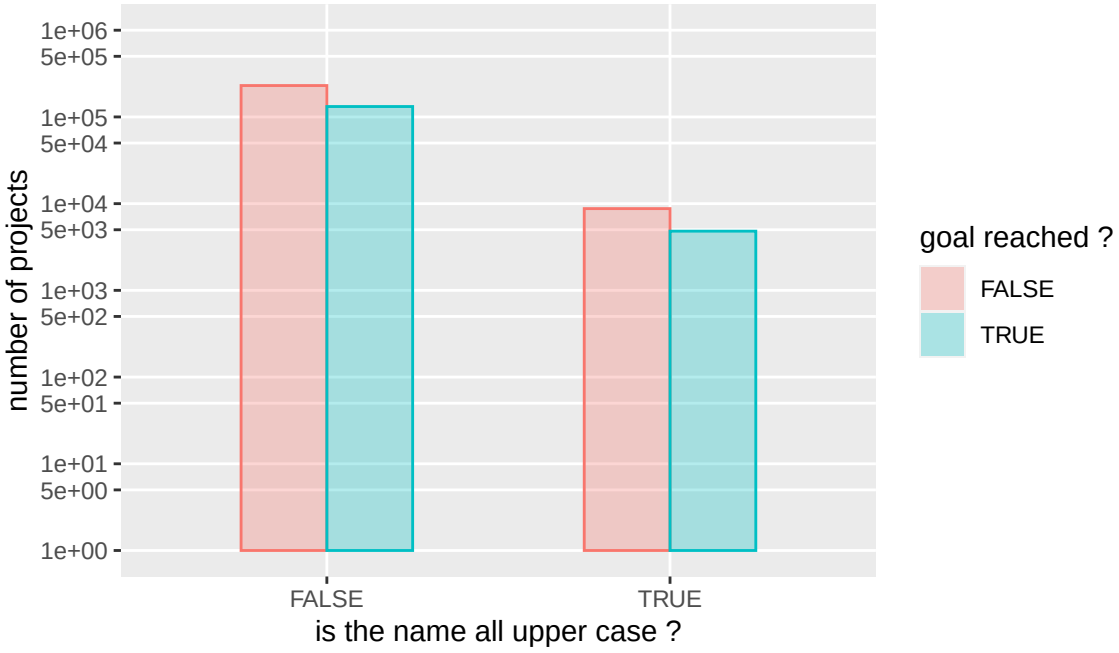
3.2.2 Lower and upper case, and special characters

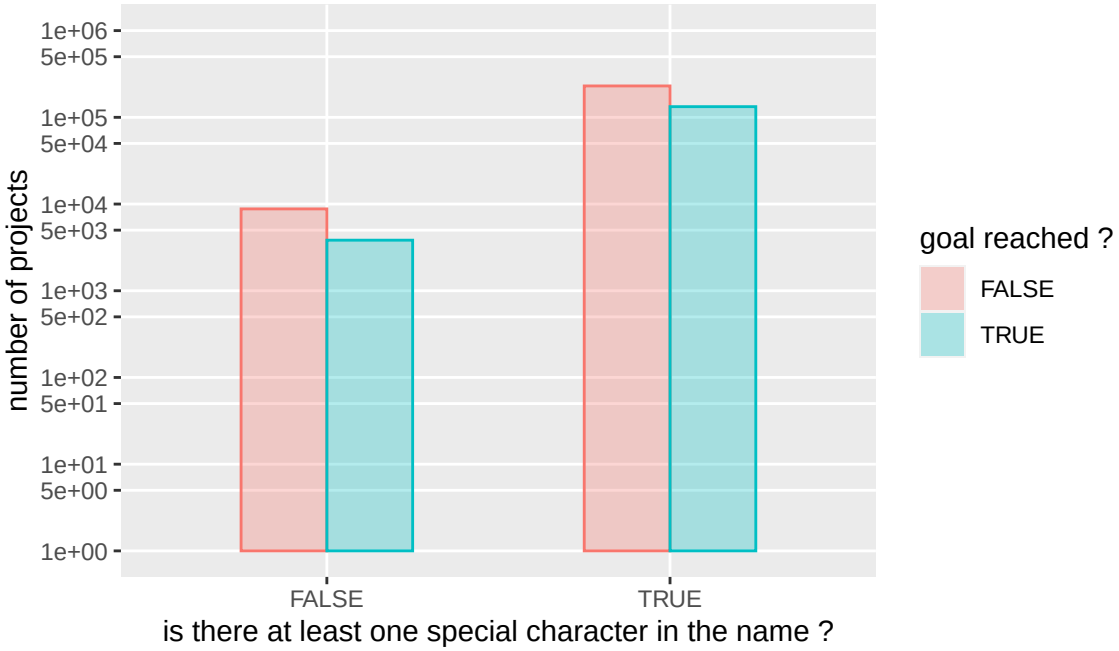
A clever use of lower case or upper case letters and special character can enhance the name of the project, by making it more punchy (with a specific word all in upper case letters following by an exclamation point, for example). To avoid a study on a case-by-case basis, only three boolean variables are investigated:

- "is the name all lower case ?" (`name_lcase`),
- "is the name all upper case ?" (`name_ucase`),
- "is there at least one special character in the name ?" (`name_spec_char`).

By comparing the occurences of each boolean values, with the distinction of wether the project's goal has been reached or not, it appears that very few project's names are either all lower case (1.0%) or all upper case (3.5%), and very few project's names contain no special character (3.3%).


1e+06
5e+05
1e+05
5e+04
1e+04
5e+03
1e+03
5e+02
1e+02
5e+01
1e+01
5e+00
1e+00
number of projects
FALSE
TRUE
is the name all lower case ?
goal reached ?
FALSE
TRUE

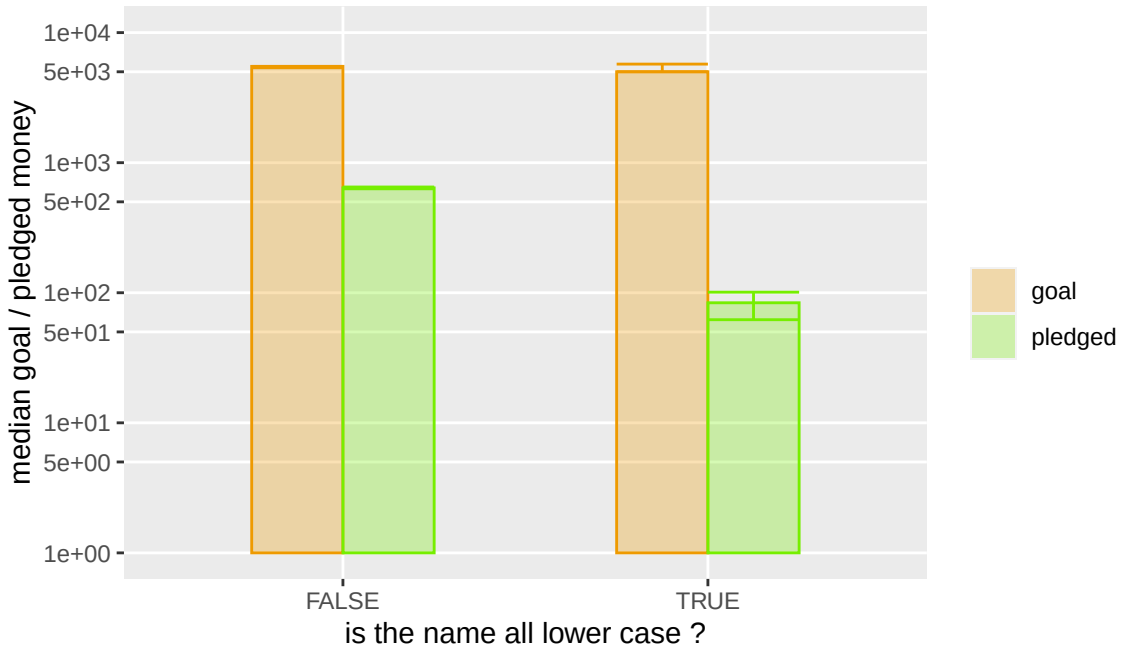

1e+06
5e+05
1e+05
5e+04
1e+04
5e+03
1e+03
5e+02
1e+02
5e+01
1e+01
5e+00
1e+00
number of projects
FALSE
TRUE
is the name all upper case ?
goal reached ?
FALSE
TRUE

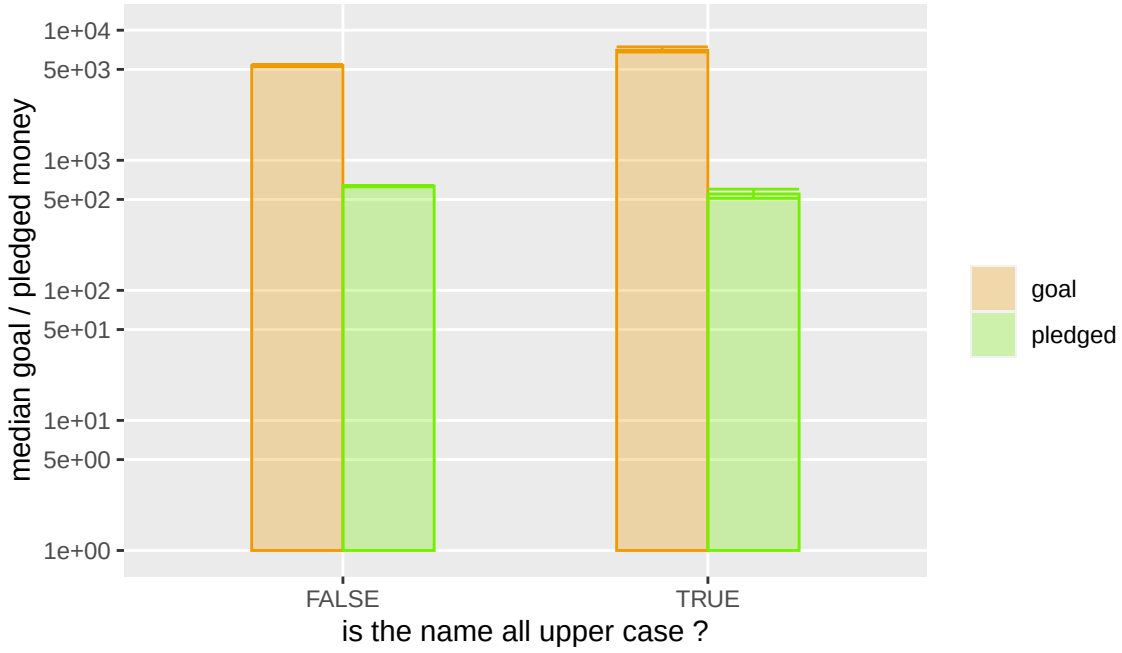


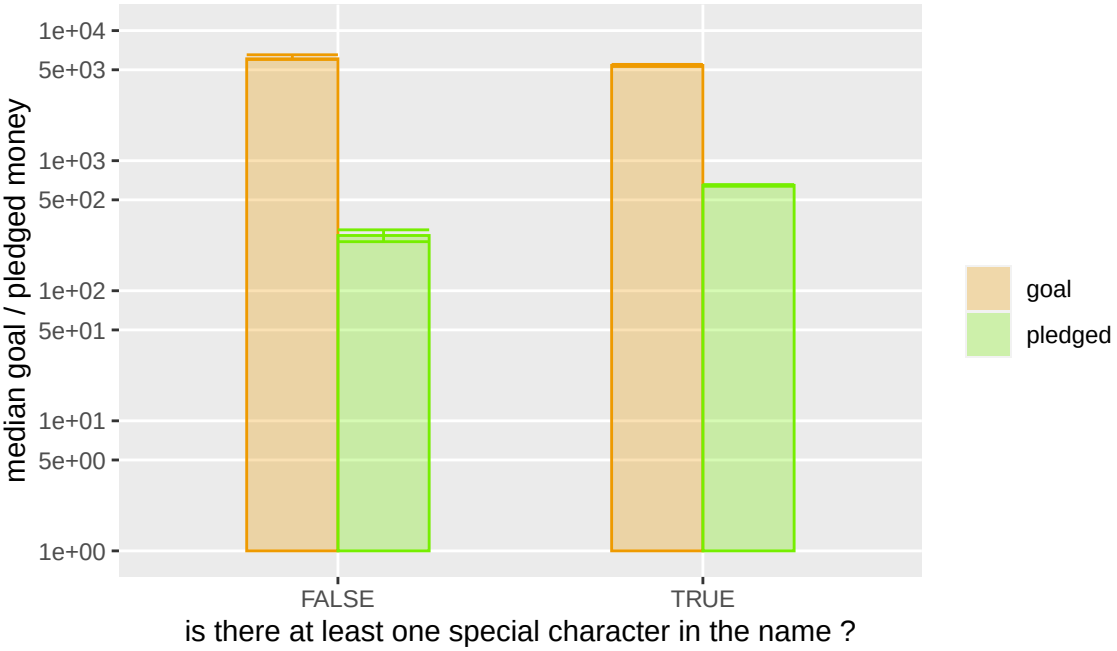
From a statistical only point of view (see graphs below), projects whose name is all lower case tend to receive almost ten times less than otherwise; projects whose name is all upper case tend to receive a little less than otherwise; projects whose name does not contain any special character receive around two times less than otherwise.

One can assume that many names follow basic grammatical rules (upper case letter at the beginning, followed by lower case letters), then making the large majority mixed-case name, not clearly captured by those variable. The `name_spec_char` variable may also not be optimal, as a simple comma, point, double-point or hyphen is a special character, and a case-by-case basis may be necessary to explore the effects of the presence of individual (or combined) specific special characters.

These variables cannot capture great details about the names, but it should incite project creators to put at least one upper case letter, and some special characters in their project's name.


1e+04
5e+03
1e+03
5e+02
1e+02
5e+01
1e+01
5e+00
1e+00
median goal / pledged money
FALSE
TRUE
is the name all lower case ?
goal
pledged

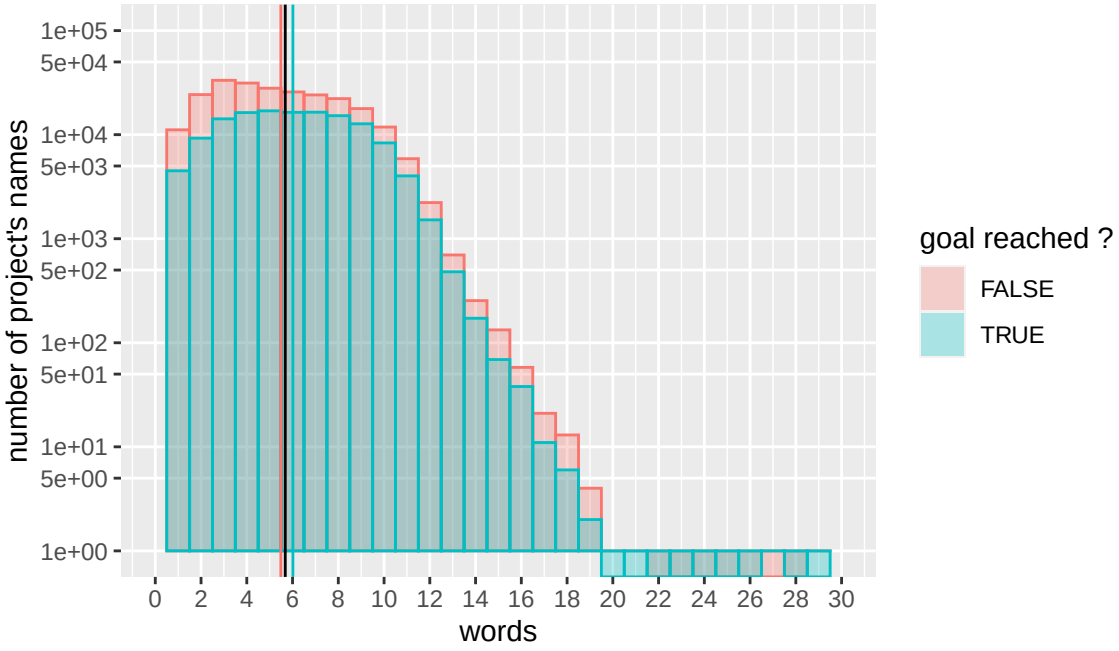

1e+04
5e+03
1e+03
5e+02
1e+02
5e+01
1e+01
5e+00
1e+00
median goal / pledged money
FALSE
TRUE
is the name all upper case ?
goal
pledged

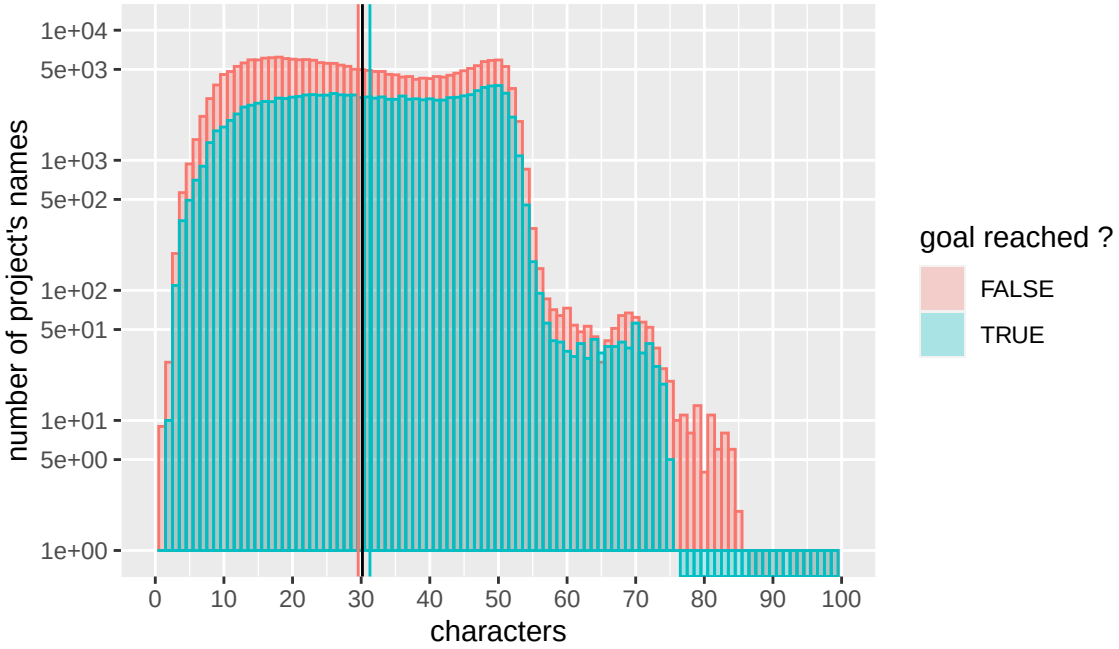


3.2.3 Length of the string

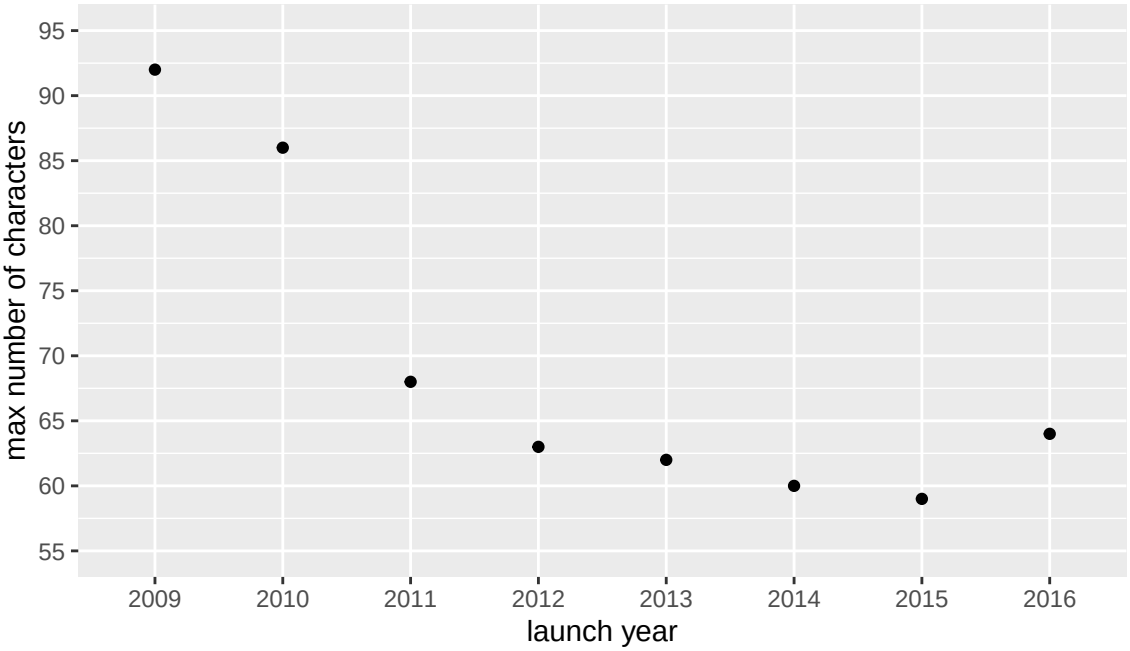
The length of the name's string may vary, as sometimes, it is solely the name of the product, and sometimes it stands as a little description of the product. By plotting the distribution of the length of the string used for the name, either by counting the words or the characters, it appears, as expected, that not all projects have the same number of words (around 6 in average) nor the same number of characters (around 30 in average). These graphs present interesting features:

- the distributions of the number of words per name, wether the goal was reached or not, are quite similar, except for lower number of words, which may indicates that a too short name may not be appealing enough (for example, giving just the name of the product).
- the distribution of the number of characters per name is a little more complex: projects that did not reach their goal follow approximately the same trend as the projects that did reach their goal, except for names with more than 75 characters, for which almost none reached their goal.
- the distribution of the number of characters per name shows two plateaus, one between 10 and 50 characters, and one between 55 and 75 characters. This may be explained by a "boring" effect for too long names, but the switchover between the two plateaus seems too sharp. Another hypothesis may be that Kickstarter may have change at some point the maximum number of characters for the name.





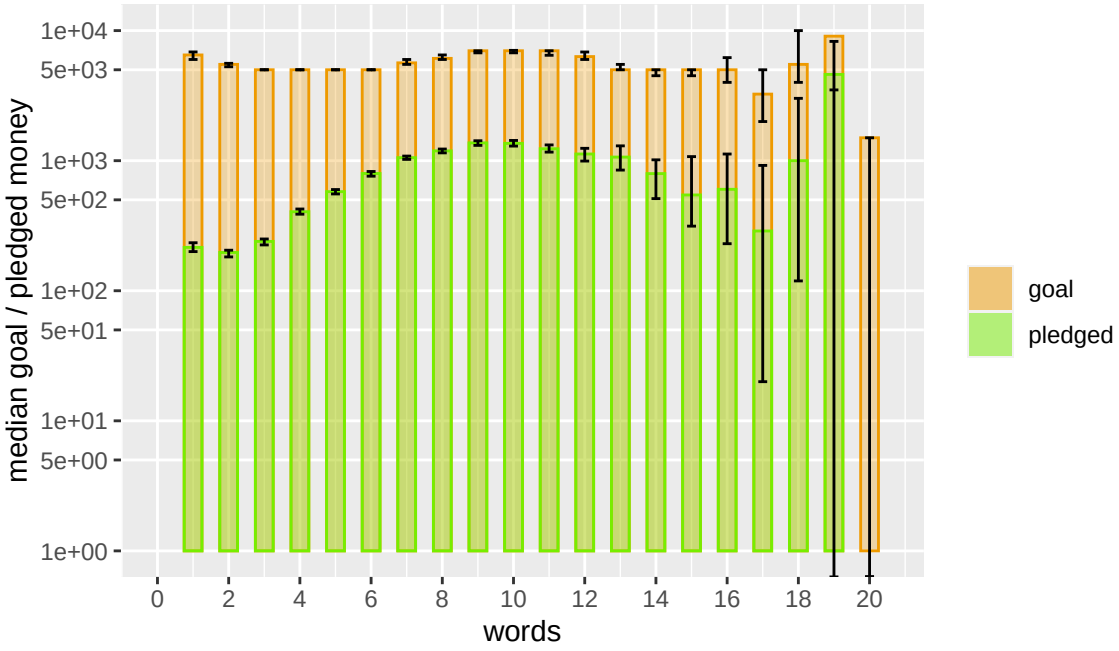
On the last point, it appears that, by plotting the maximum number of characters in the names per year, people stopped using name with more than 70 characters after 2011. This may explain the presence of the two disctinct plateaus.

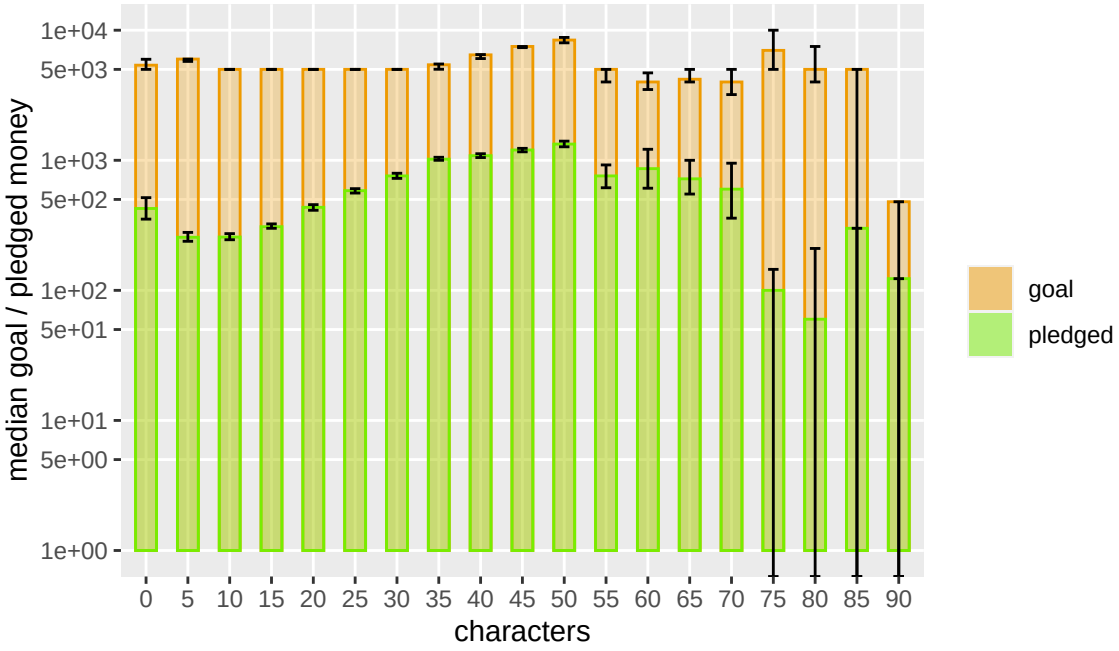


When comparing the distribution of the median pledged money per number of words or characters, they both seem to present the same features:

- the general trend is that the median pledged money increase with the length of the name until a threshold (around 9 words or 55 characters).
- after that threshold, the uncertainties start increasing significantly, and the median lose its meaning (logarithmic y-axis).
- projects with a very short name (1 word, or less than 5 characters!) oppose this trend, by having a slightly higher median pledged money.

The goal being roughly the same regardless of the length of the name, one should try to build a name that reached 50 character in 10 words, based on these statistics.



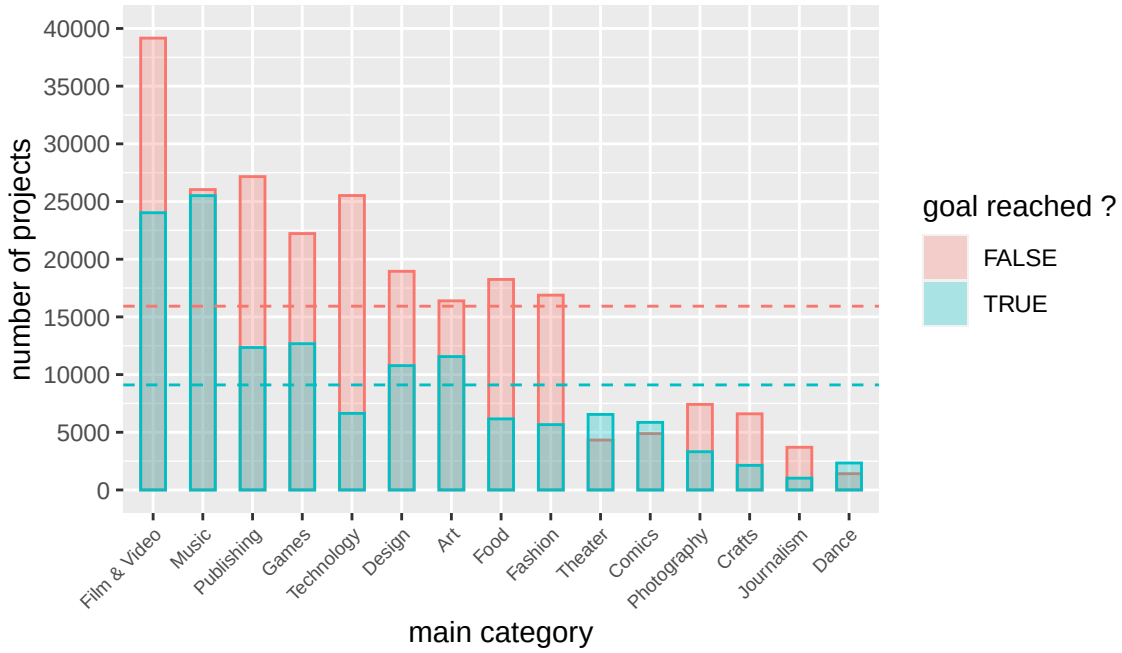


3.3 Category

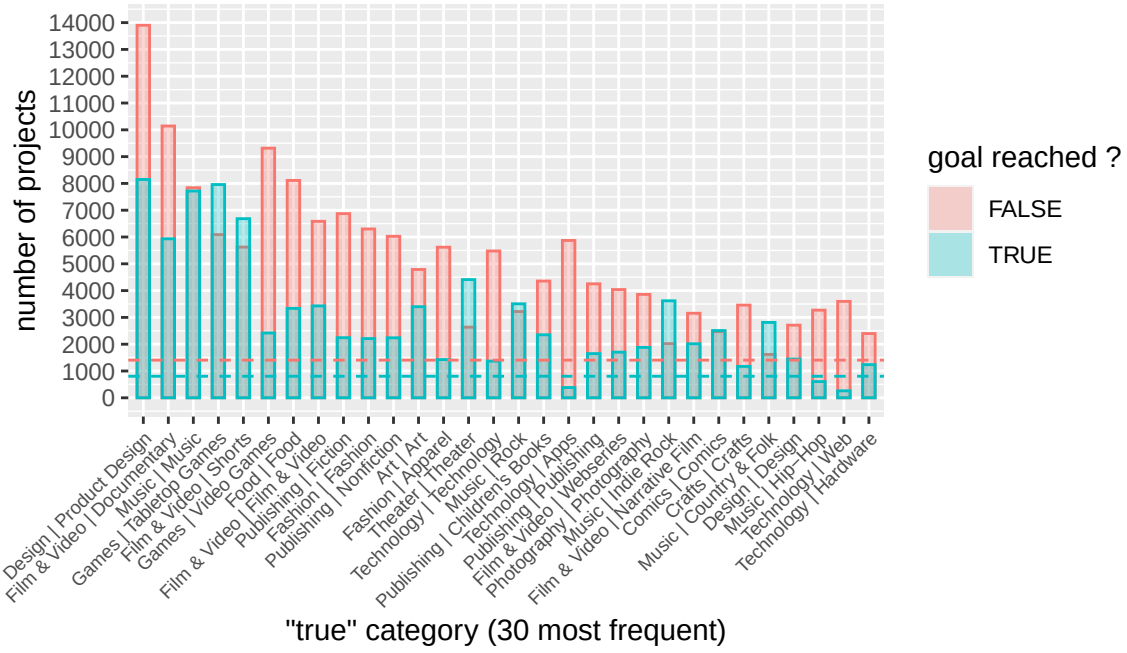
The different projects must indicate their category with two variables, `main_category`, which is general, and `category` which is more specific. As shown in a previous section, some `category` values can be shared between different `main_category` values, so the `true_category` variable was created (concatenation of the two variables), to allow a truer representation of the all the categories.

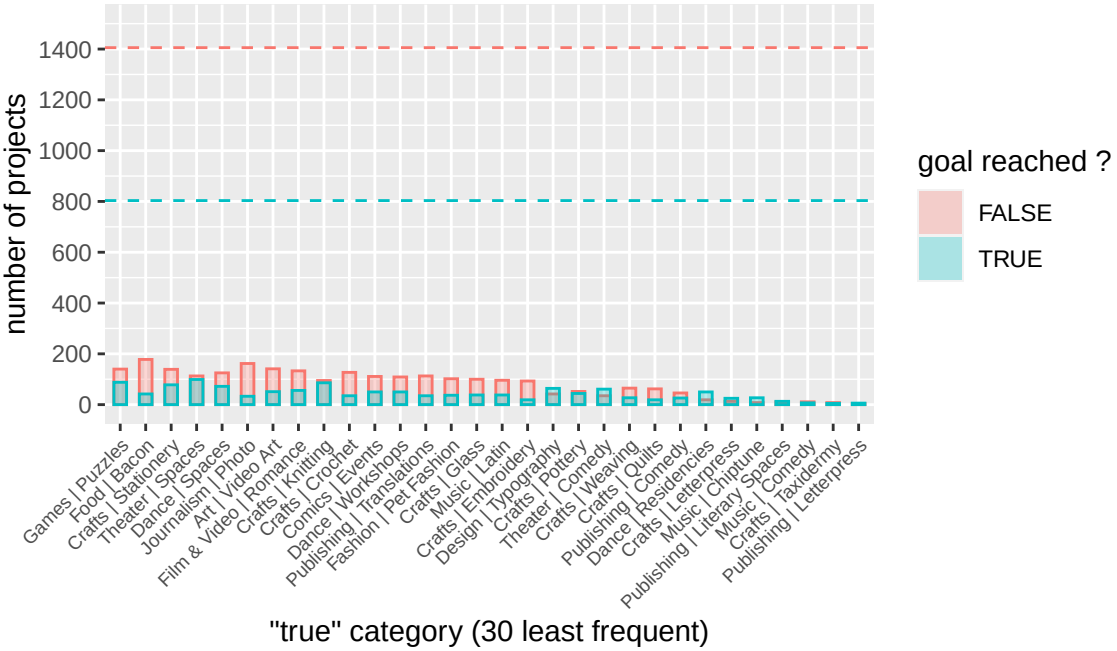
First, the distributions of the percentage of projects per main category, with the distinction of wether the goal has been reached or not, are plotted. The dashed lines represent the percentage of projects if the distribution was uniform among the main categories. The general trend seems to be that, for the most frequent main categories, chances of reaching one's goal is around less than 50%. The two most frequent main categories

("Film & Video" and "Music") represent around 30% of the total projects.

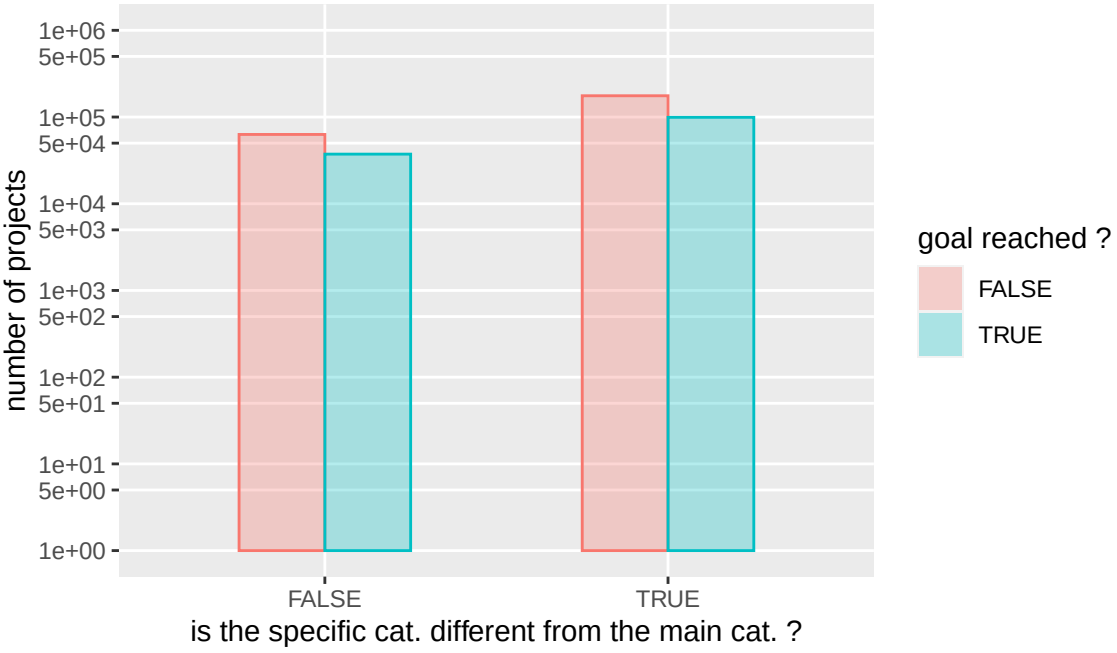


Next, the distribution of the percentage of projects per "true" category, with the distinction of wether the goal has been reached or not, is plotted (only the 30 most and least frequent "true" categories). As previously, the dashed lines represent the percentage of projects if the distribution was uniform among the "true" categories (they are much lower because of the higher number of specific categories). The same general trend can be identified here (less than 50% chance to reach one's goal), but since the total number of categories is higher, many more exceptions (more than 50% chance to reach one's goal) can be spotted.

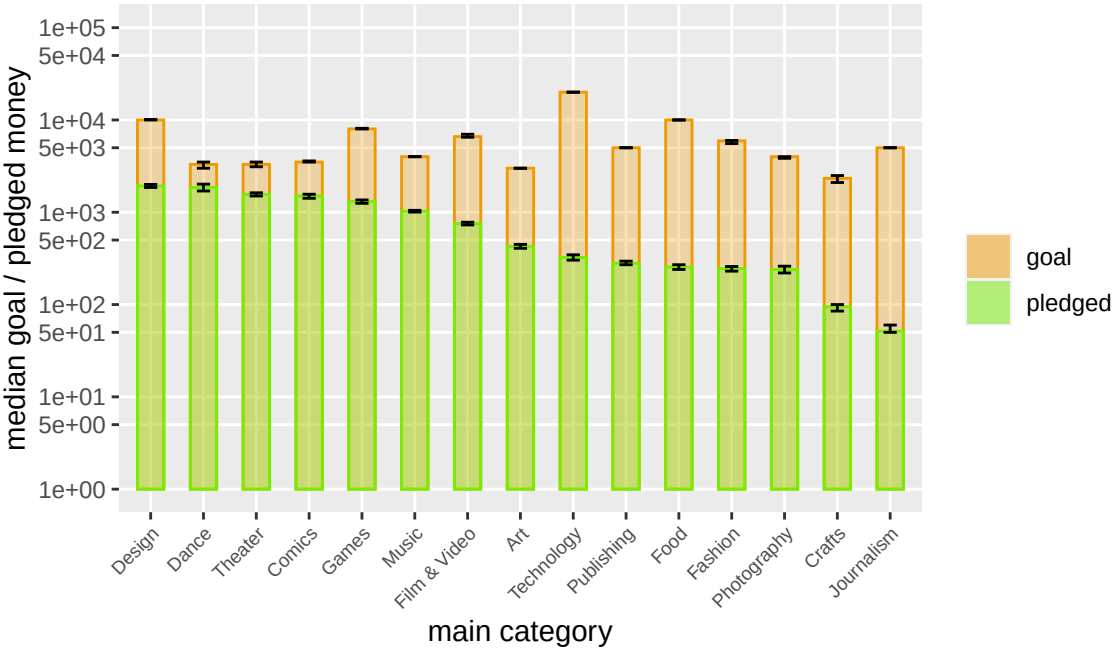




One thing to notice is that among the 30 most frequent categories, there are many projects with no specific categorie ("Music | Music", for example), which is not the case for the 30 least frequent categories. This seems to indicate that people would sometimes rather not choose a specific category (around 25 % of all projects). But the ratio of goals reached is nearly the same (around 50 %).

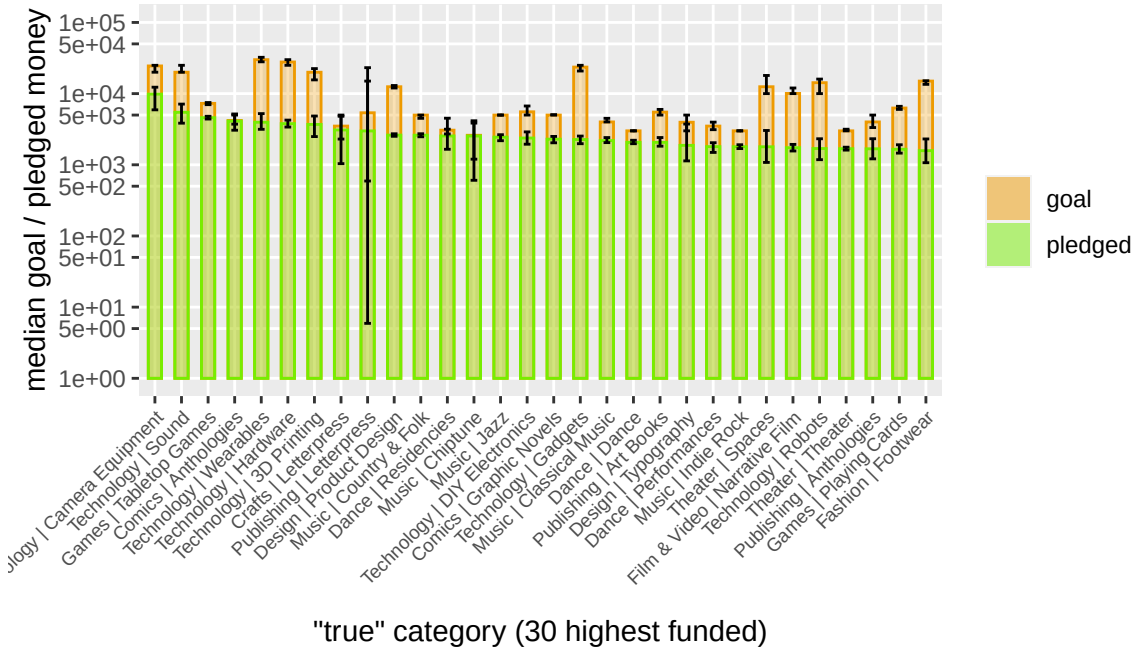


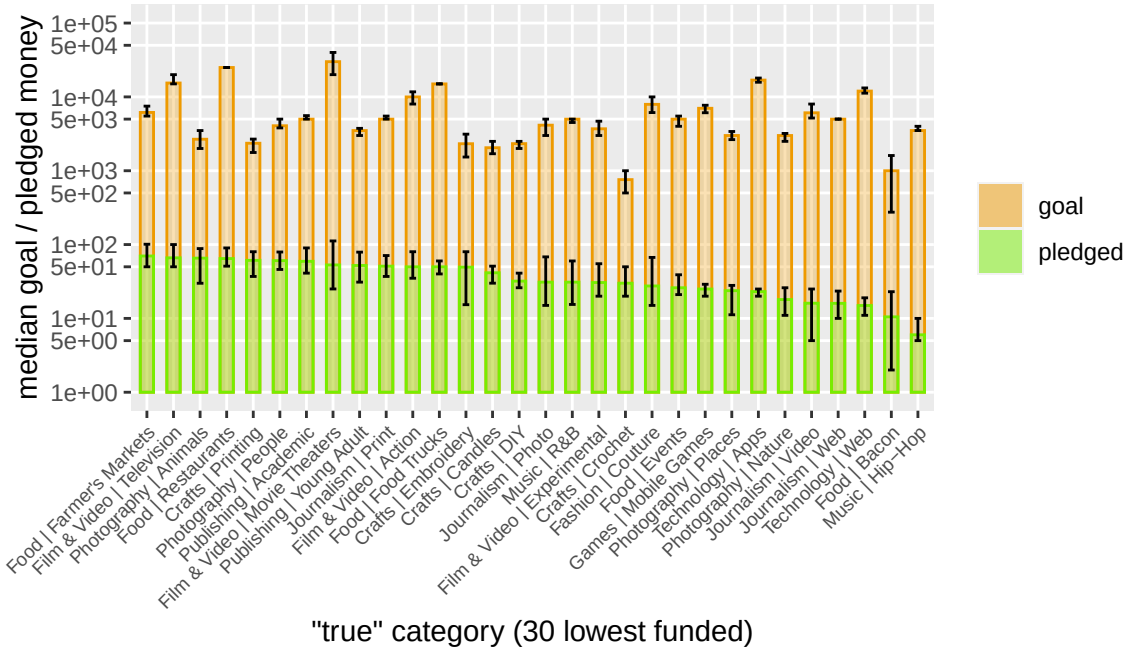
When comparing the distribution of the median pledged money among main categories, their order is overturned compared to their frequency: "Film & Video" and "Music" only receive a few hundred USD, up to a thousand USD, whereas "Dance" and "Design" receive around twice as much.



The "true" categories are even more overturned, as very specific categories are present in the 30 most and least pledged (except "Dance | Dance" and "Theater | Theater").

No clear insight, except for the preference (so far) of the backers regarding the categories, can be outlined from these statistics. These statistics only provide insight on the forecasted success rate of the category of one's project.





3.4 Calendar

3.4.1 Time span

A project can be active from 1 day to 3 months, and of course, this may influence the amount of money pledged once the deadline is reached.

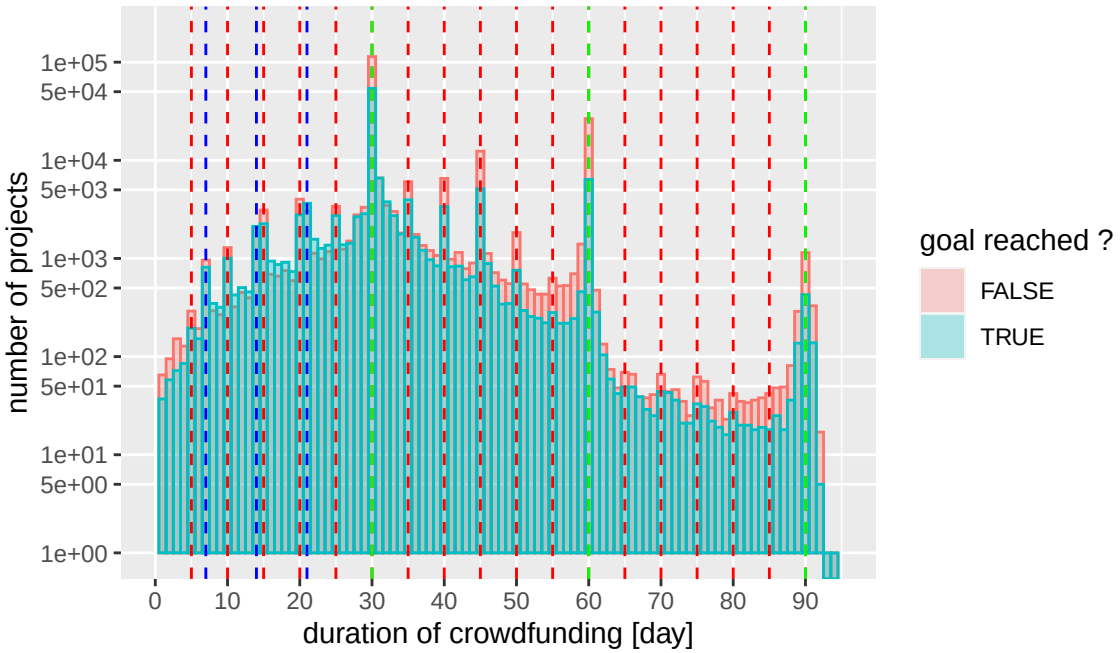
```
min(data_ks$time_span)
```

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

```
max(data_ks$time_span)
```

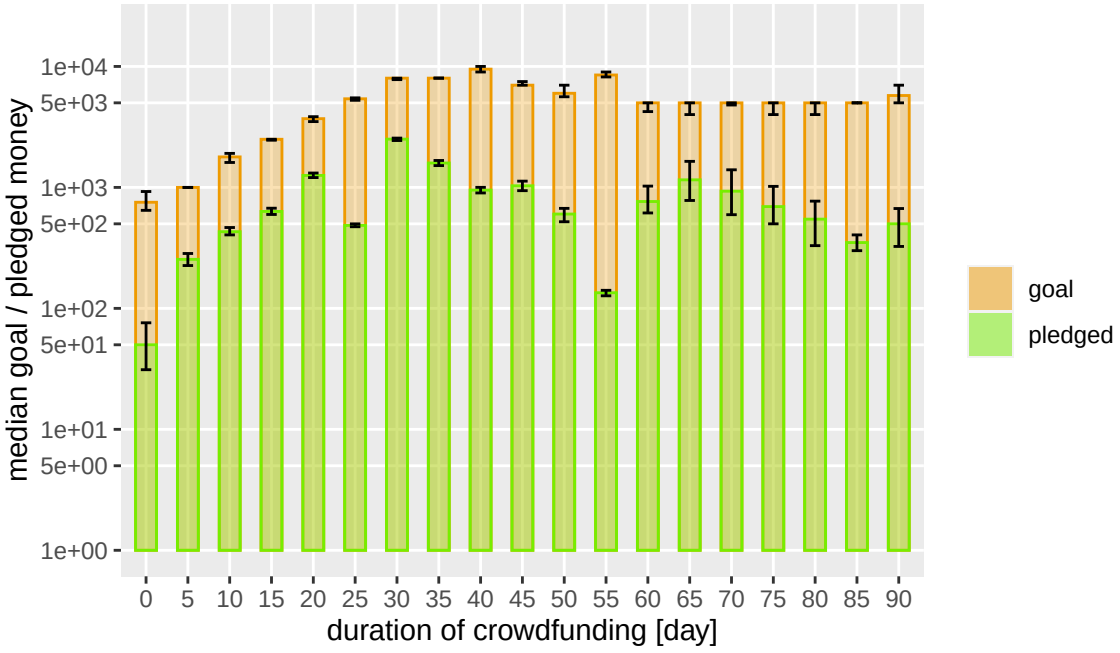
```
## [1] 92
```

The distribution of the time spans, with the distinction wether the goal has been reached or not, is plotted. It seems that round period of time a generaly chosen (weeks (blue dashed lines), 5-days (red dashed lines) or months (green dashed lines)). The majority of the projects lasts less than two months, but the chances of reaching the goal is higher before the first month. This may be explained if one assumes that the smaller the time span, the lower the goal, making it's goal easier to reach.



As expected, the goal is lower during the first month, making it easier to reach. If the duration of the crowdfunding is too short, the median pledged money is significantly low A peculiar feature is noticeable: for the five days before the end of a month (25 days, 55 days and 85 days), a significant drop in the pledged money is visible. This feature remains unexplained.

The best time span appears to be a full month (plus a day or two, to avoid the unexplained drop just below 30 days).

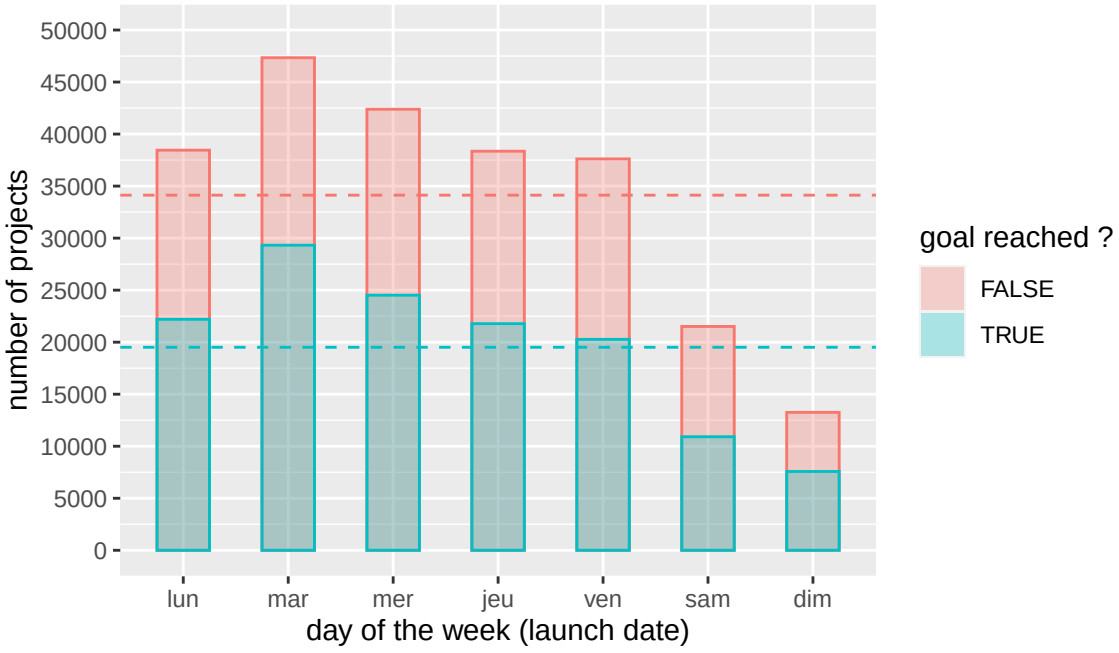


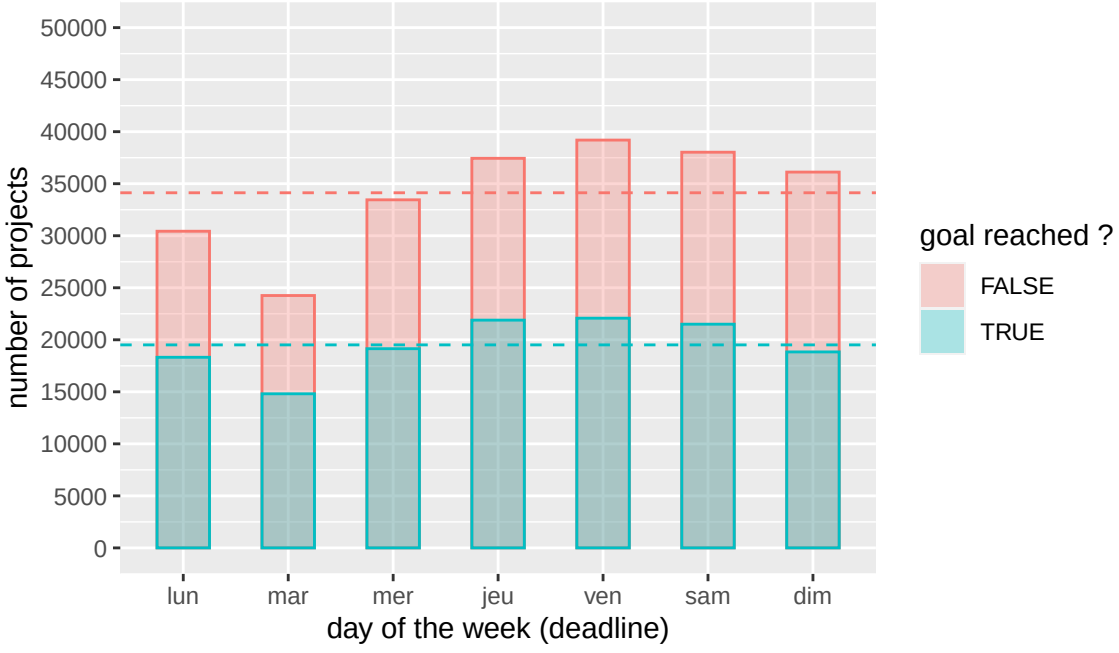
3.4.2 Day difference (week days)

The distributions of the number of projects per week day, with the distinction of wether the goal has been reached or not, are plotted. The dashed lines represent the number of projects if the distribution was uniform among the week days.

It appears that Tuesday is, somehow, the day of the week when people launch their project on Kickstarter. Also, since they are generaly not working days, the week-end shows an expected low amount of project launched.

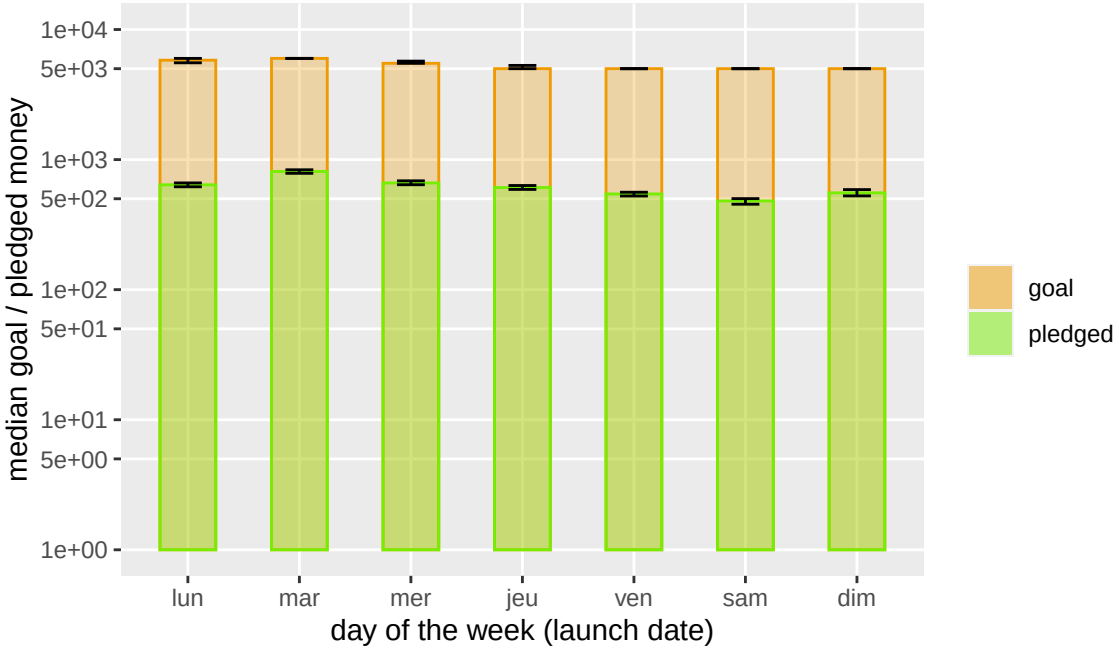
The graph with the deadline week days is reversed, with larger amount of projects being stopped at the end of the week, rather than on a Monday or a Tuesday.

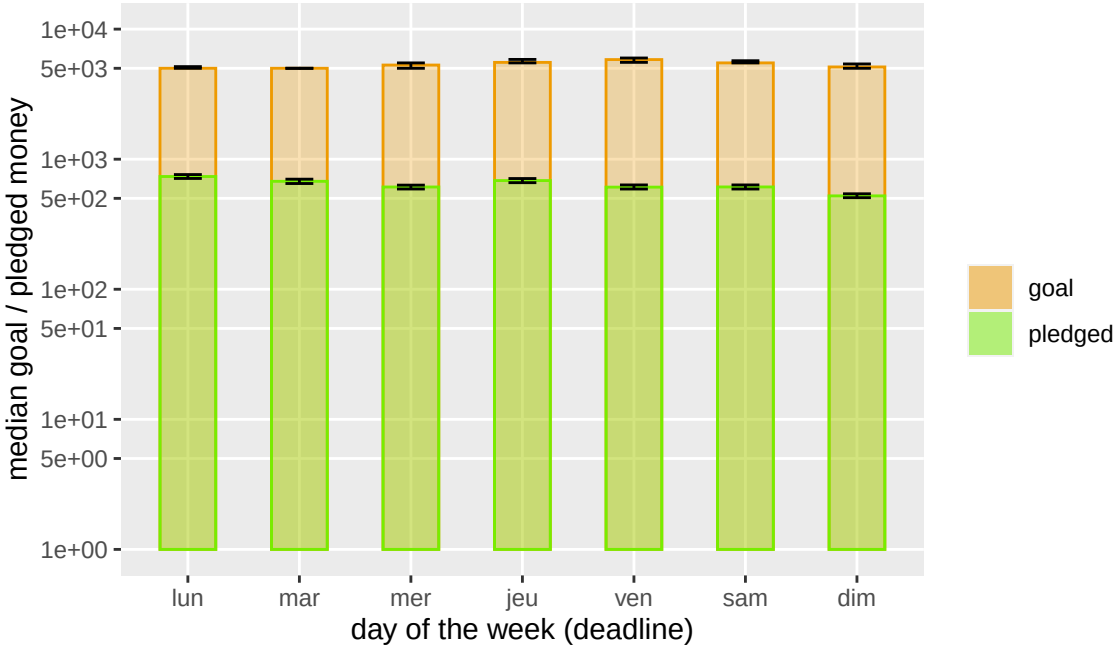




The distribution of the goal and pledged money per week day, on the other hand, does not shown any general trend (except Tuesday as launch day, with a few thousands USD than projects launched any other days).

The worst days seem to be a launch on a Saturday and a deadline on a Sunday, and the best days seem to be a launch on a Tuesday and a deadline on a Monday. Now we know !





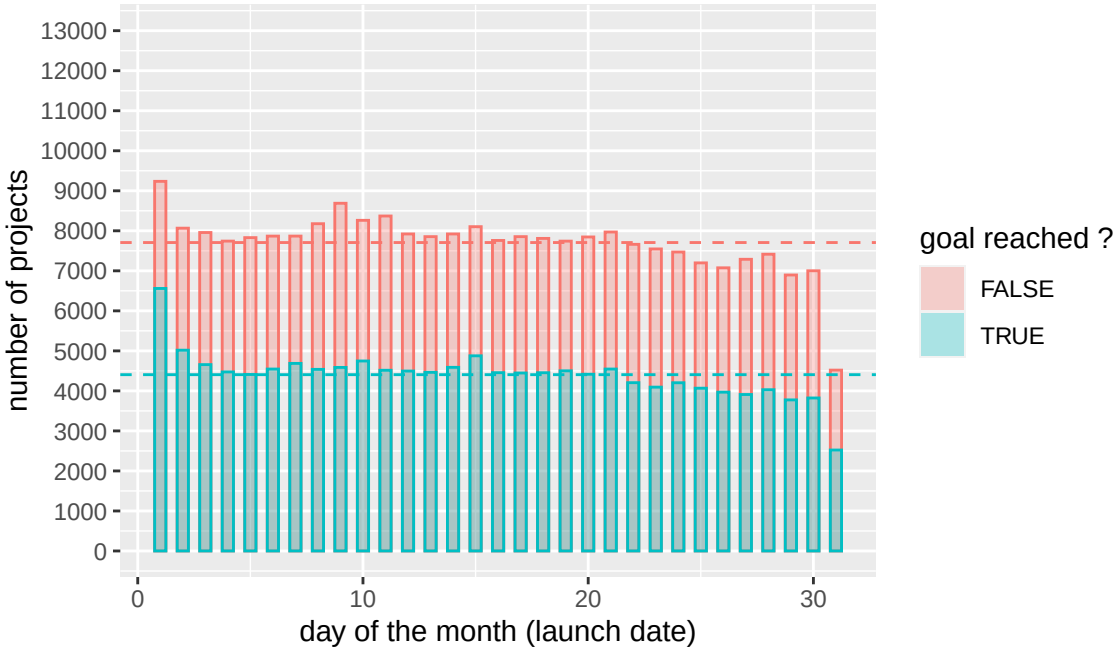
3.4.3 Day difference (month days)

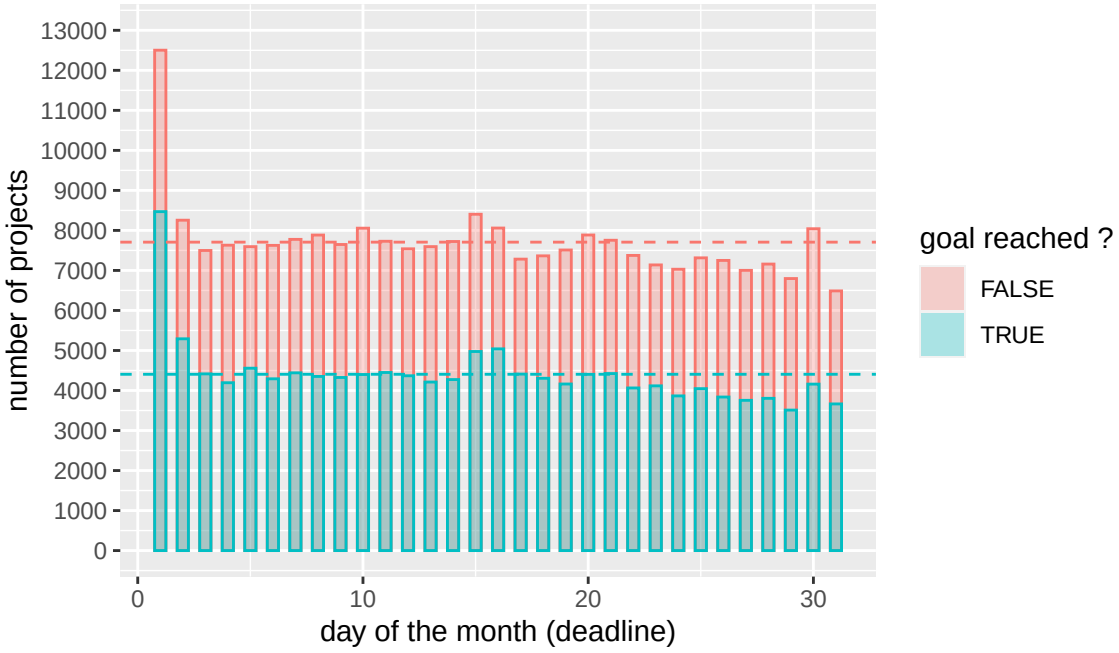
The same investigation can be conducted on month days. The distributions of the number of projects per month day, with the distinction of wether the goal has been reached or not, are plotted. The dashed lines represent the number of projects if the distribution was uniform among the month days.

It appears that the hightlights of the month are the very first days, the middle days and the very last days, for both launch dates and deadlines.

More projects than average are launched in the first or the middle days of the month, and less than average avec the 20th day of the month. The very low number of projects on the 31st day may be explain by only half of the months have 31 days.

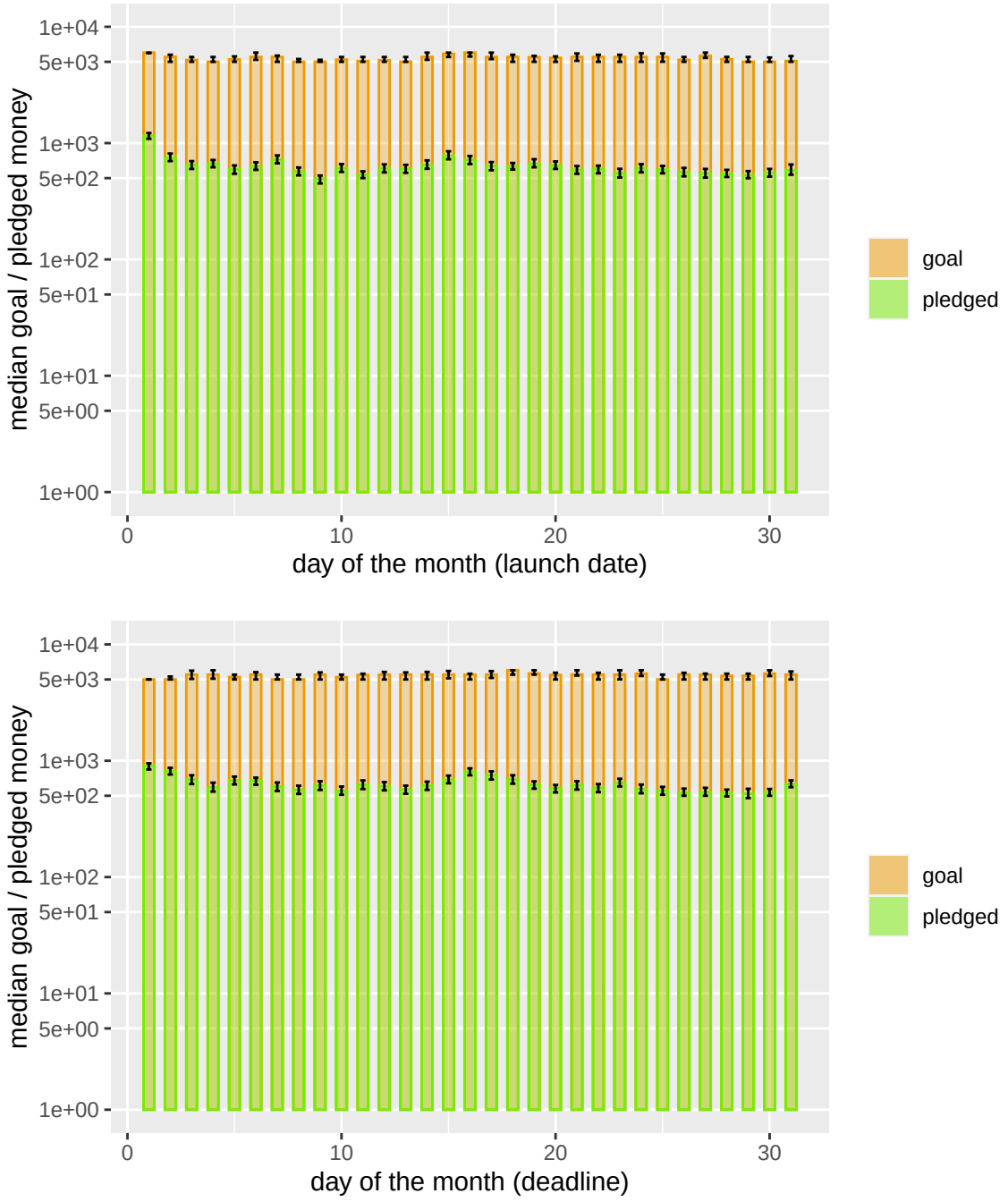
For the deadlines, the highlights are even more prononced (start, middle and end of the month).





The distribution of the goal and pledged money per month day does have the same highlights, with projects pledging more money on the start, the middle and the end of the month.

Statistically, the most lucrative day, for either the launch date or deadline, is the first day of the month.



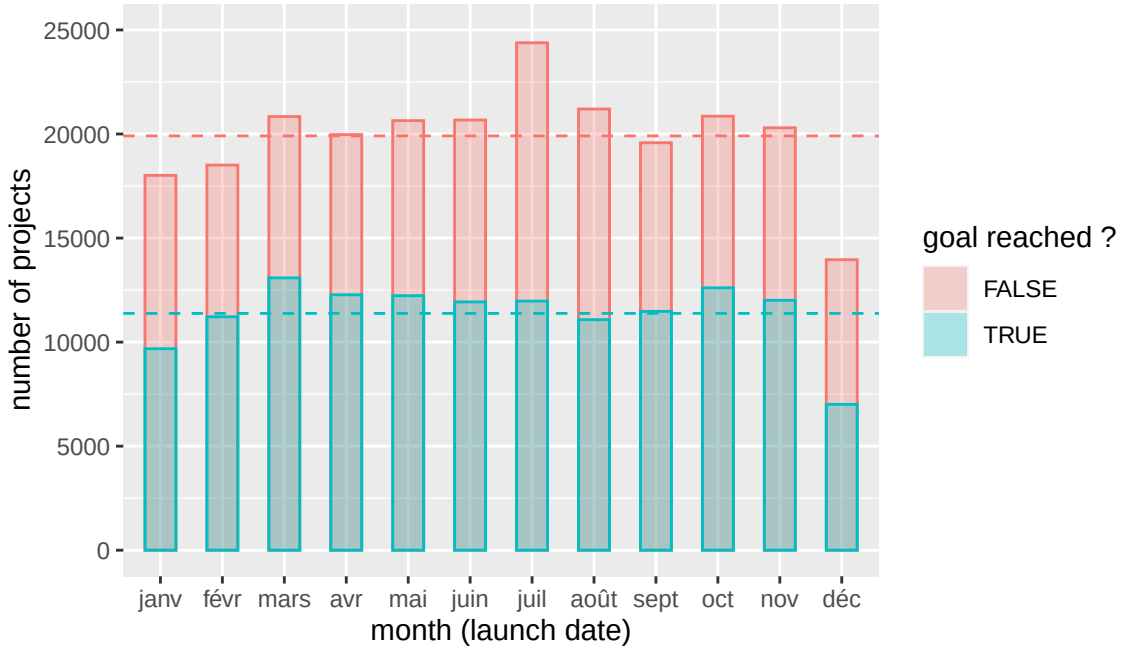
3.4.4 Month difference

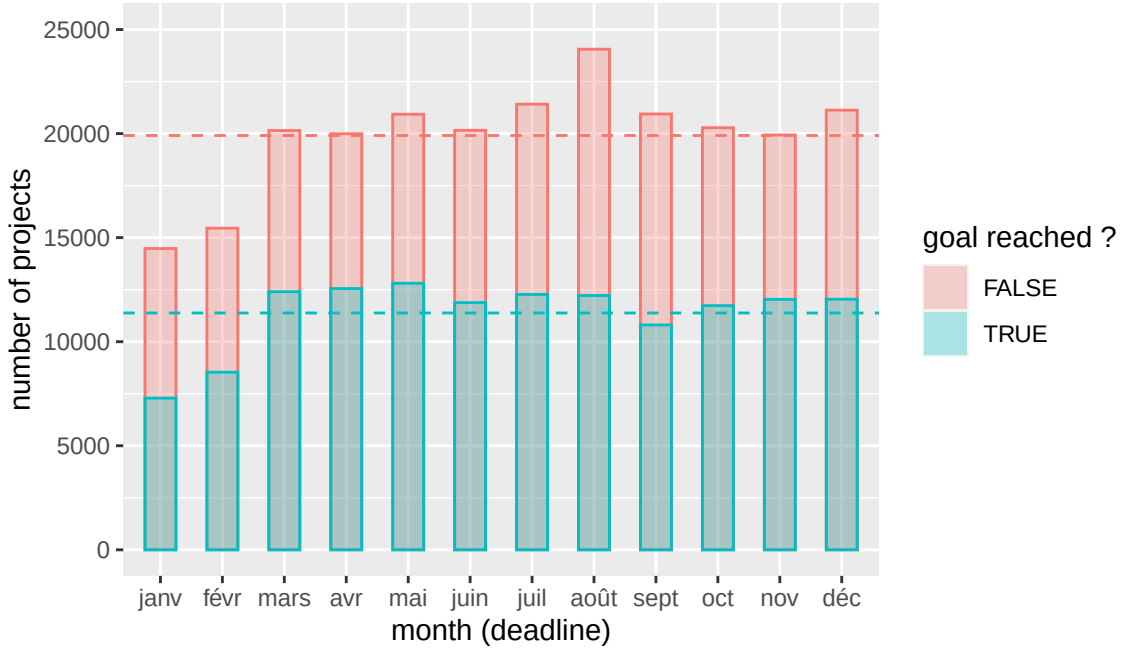
The distributions of the number of projects per month, with the distinction of wether the goal has been reached or not, are plotted. The dashed lines represent the number of projects if the distribution was uniform among the months.

It appears that fewer projects are launched in December, and less projects reach their goal when launched in January, and many projects are launched in July, but a large proportion of them fails to reach their goal.

Less projects have their deadline on January or February, and many projects have their deadline on August,

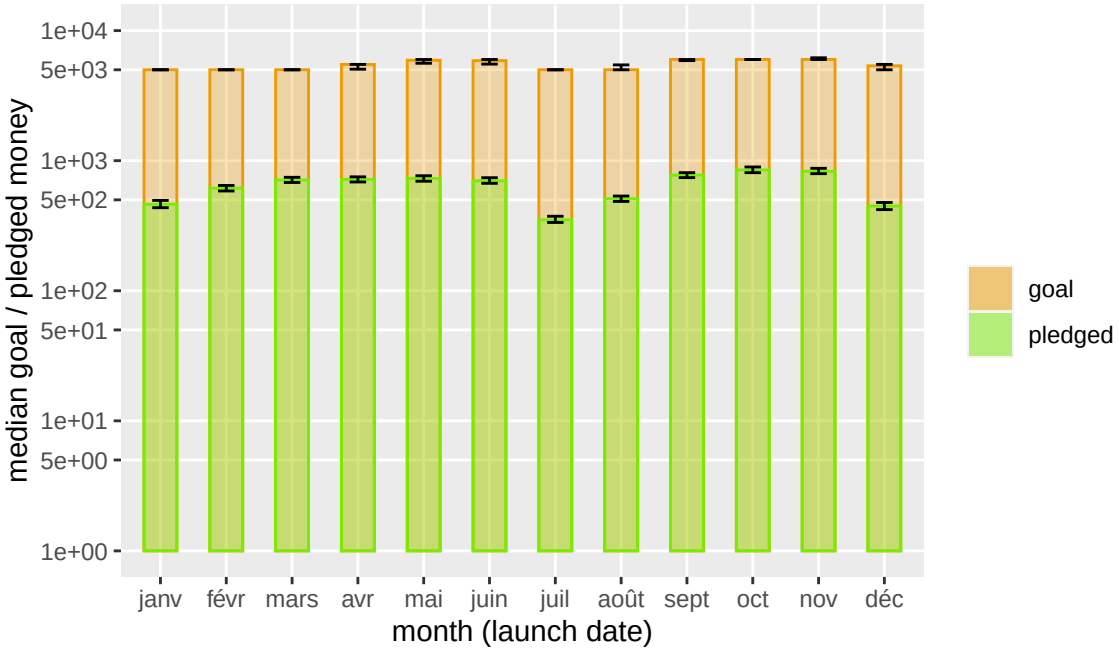
but a large proportion of them fails to reach their goal.

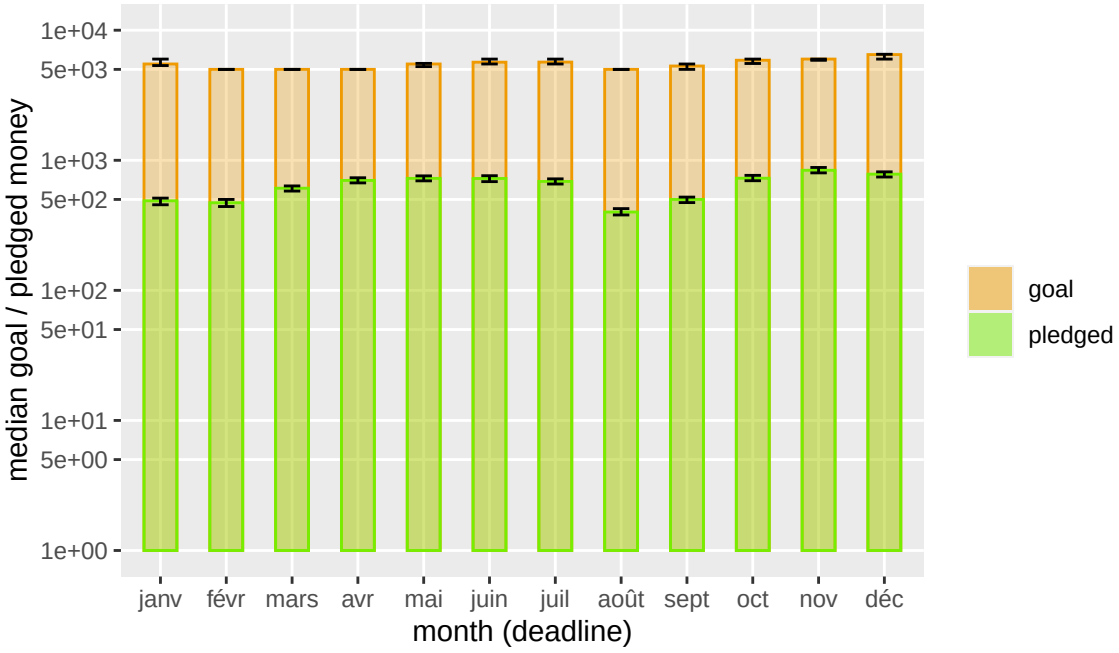




The distribution of the goal and pledged money per month, on the other hand, shows amost the same trend: projects launched or ended on December, January (winter holiday), July and August (summer holiday) pledge less money than on other months.

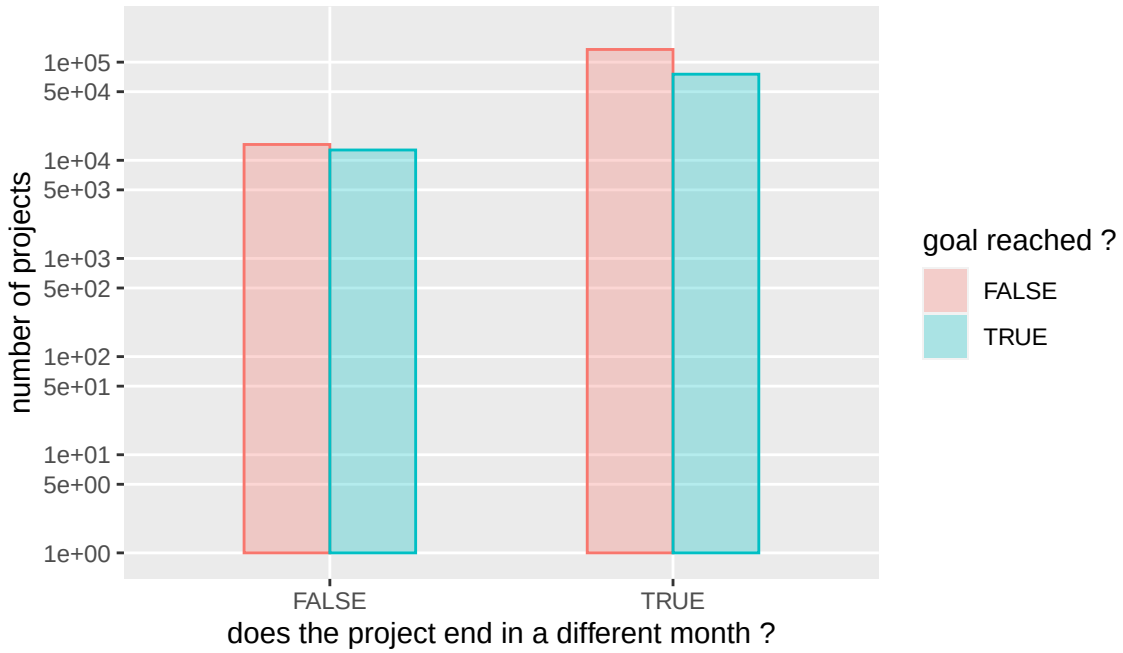
Statistically, the most lucrative month periods, are March to June, and September to November.



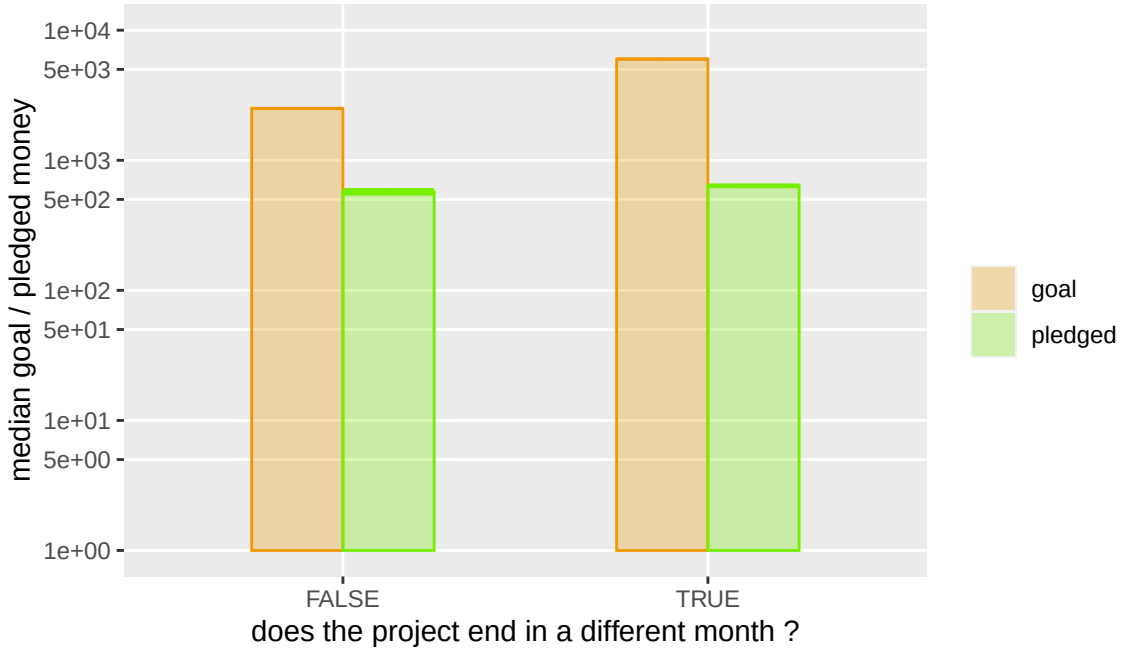


That being stated, should your project (if its duration is below 31 days) be included in a single month, or overlaping different months ?

It seems that generaly, projects are overlaping different months, though the chances of reaching its goal are better for projects launched and ended on the same month.



However, the median pledged money is approximately the same in both case.



3.4.5 Quarter difference

The investigation on the quarters lead to the same conclusions than the previous section (on months), so, for clarity sake, the results are not present in this report.

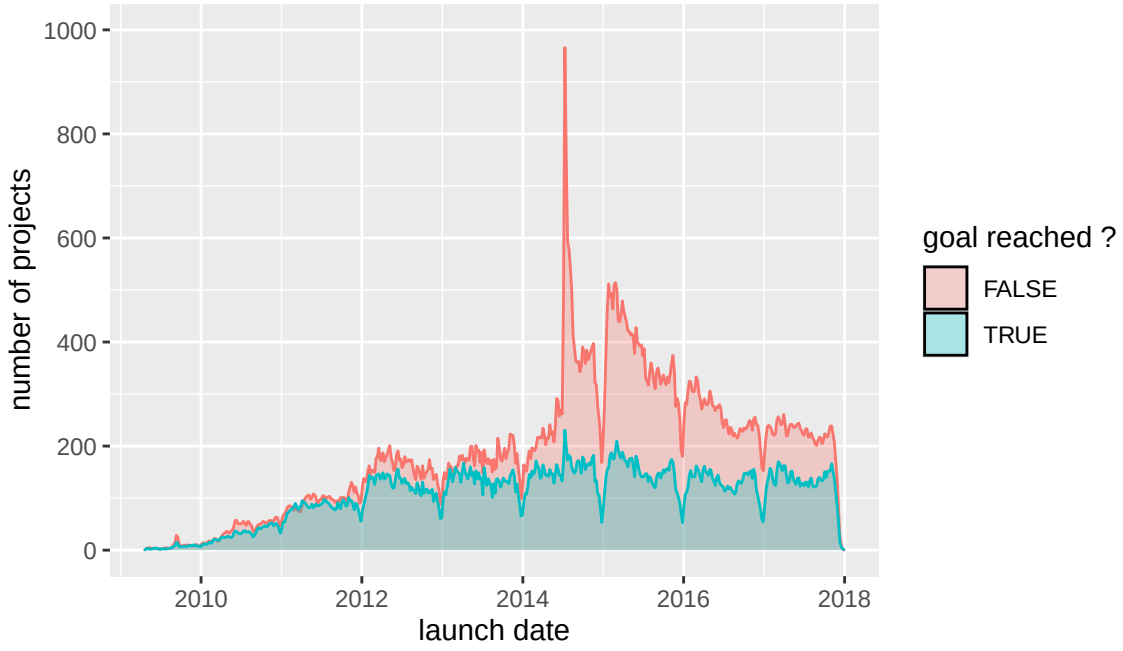
3.4.6 Global evolution

The global evolution over time should confirm some previous findings, and may also provide more general insights about the fluctuations of the "market" of projects on Kickstarter.

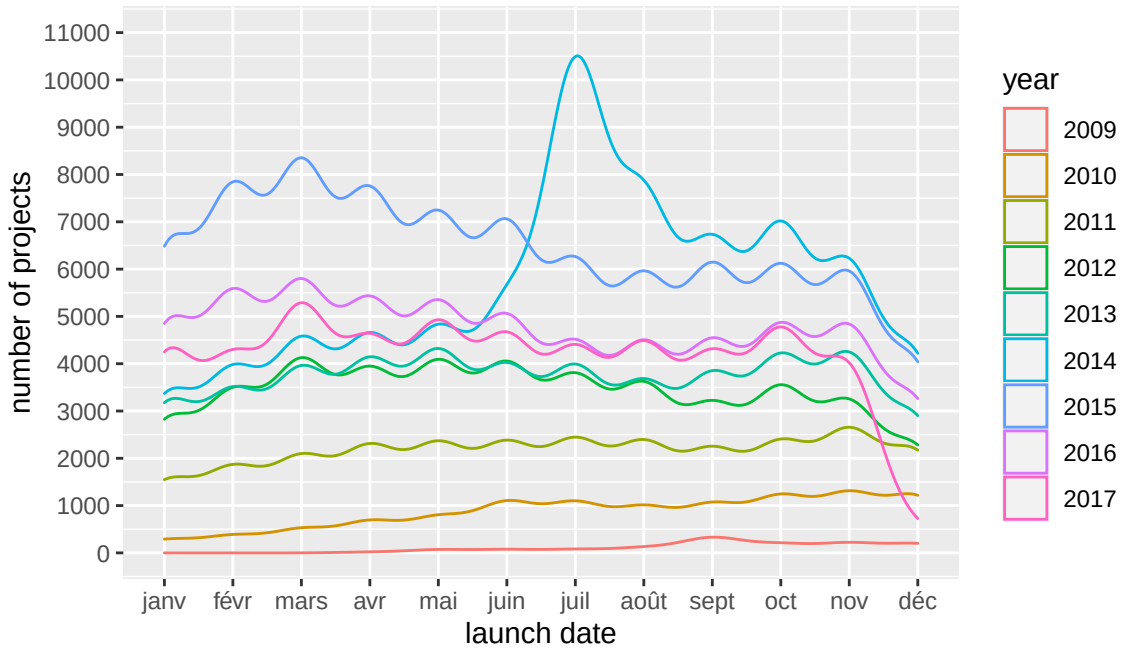
The number of projects launched per day, with the distinction wether the goal has been reached or not (plotted below), shows an increase of the total number of projects during the first 2-3 years, and a steady number of projects that reached their goal (between 100 and 200 per day). The number of projects that did not reached their goal peaked in 2014, and slowly decreased to reached a range between 200 and 300 per day.

Both distributions shows a drop in the number of projects launched around New Years, and, especially after 2014, the number of projects launched presents a weaker drop in the middle of the years, as shown when investigating projects' distribution among months.

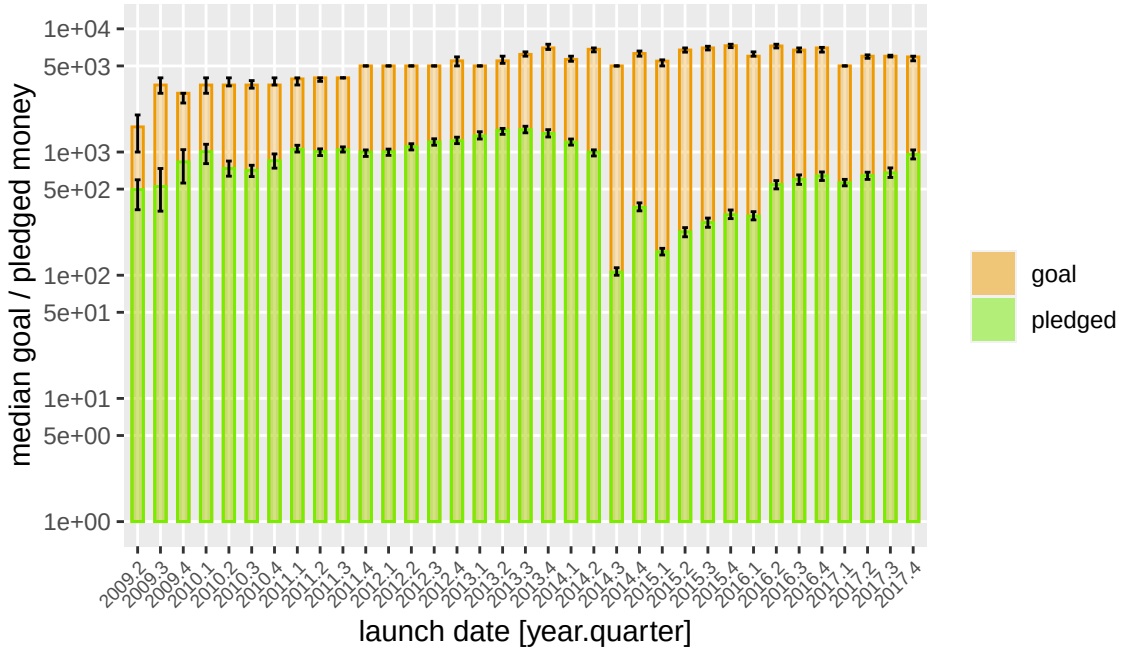
The same graph with the deadline instead of the launch date (not plotted here) presents the same general features, with a global shift of around a month.



By superposing the different years, it appears, as shown when investigating projects' distribution between days of the month, that projects are prefentially launched at the start of the month.

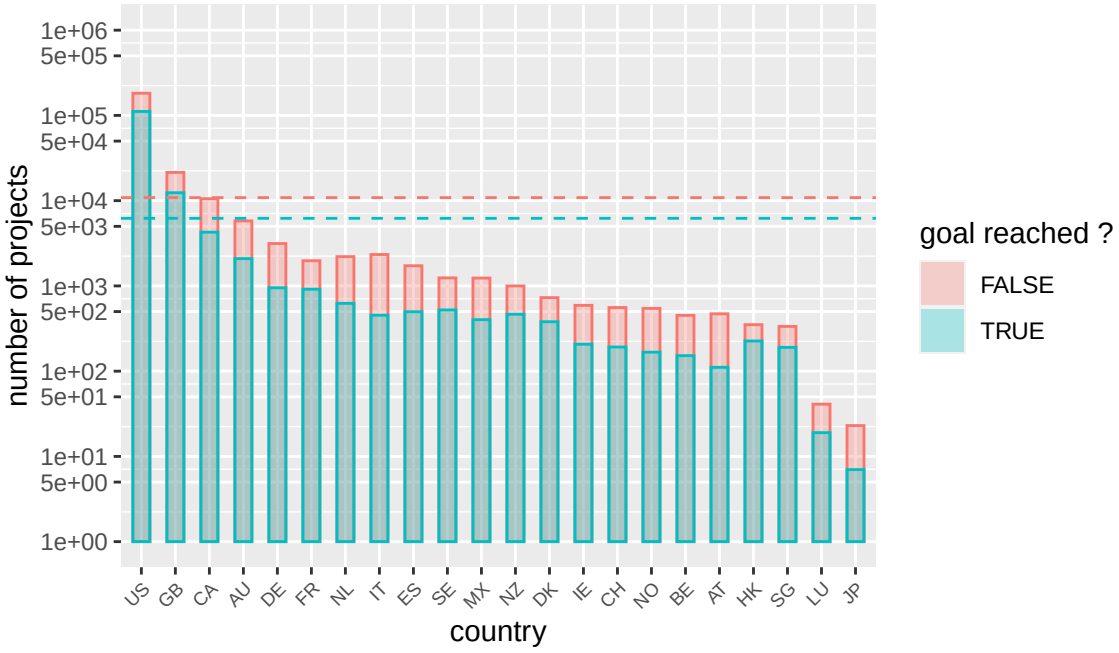


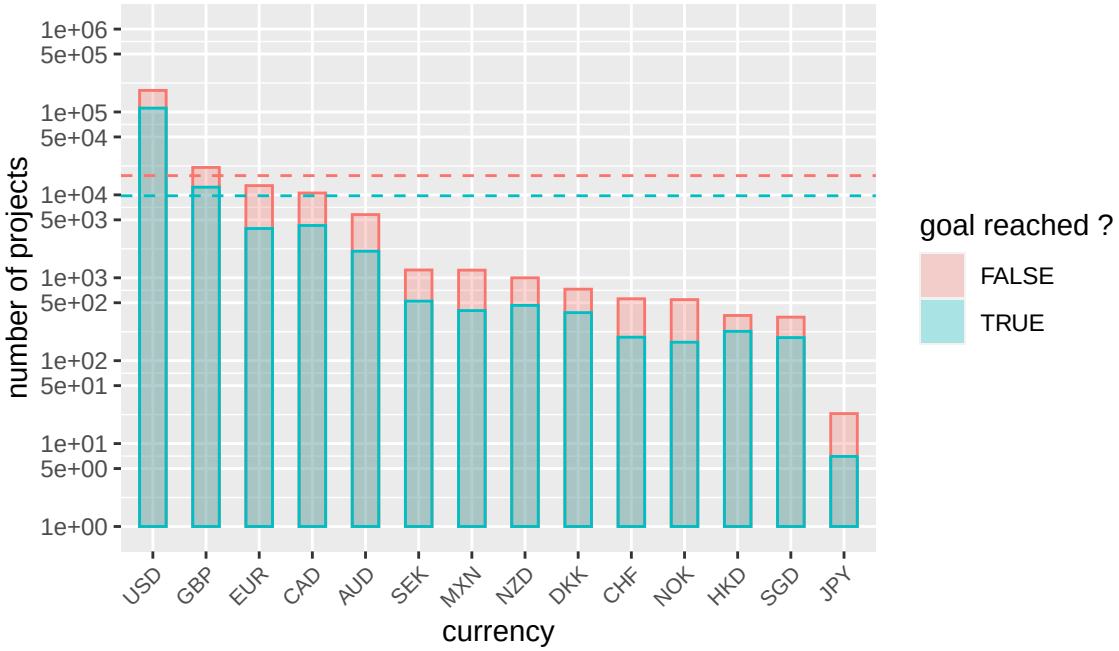
The median pledged money is strongly affected by the 2014 peak for projects' number, with a sharp drop at the same quarter, and a slow catching-up afterward, as the total number of projects decreases. This shows that despite the strong increase of number of projects on the Kickstarter platform, the number of projects that reached their goal did not increase, then mecanically decreasing the median pledged money (a lot of projects receive no, or next to no, money at all).



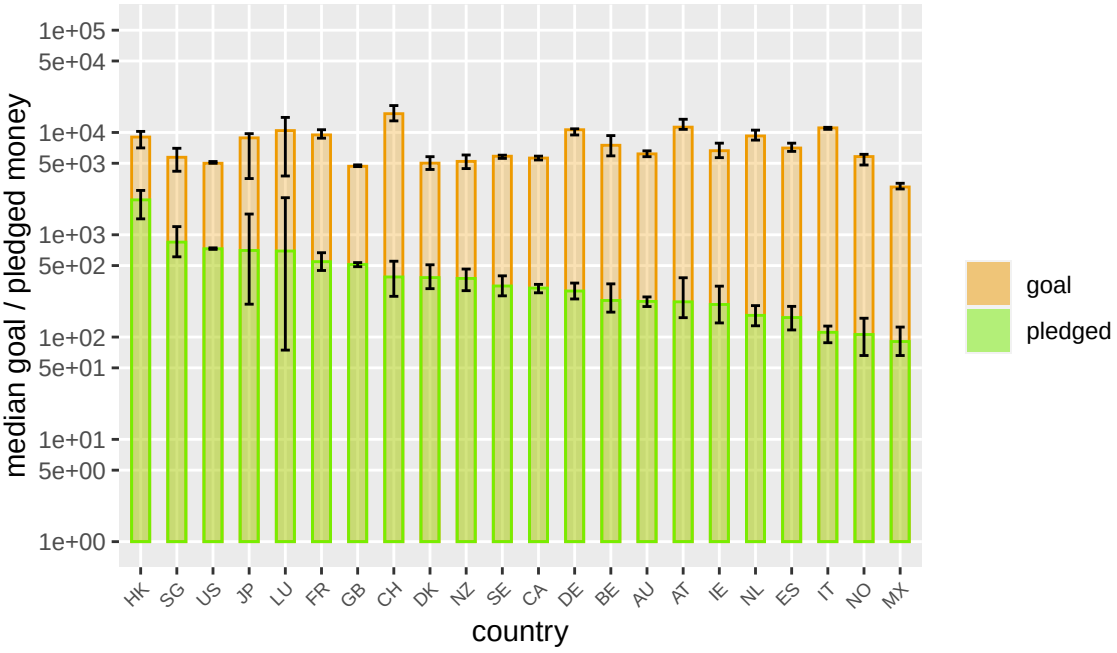
3.5 Country

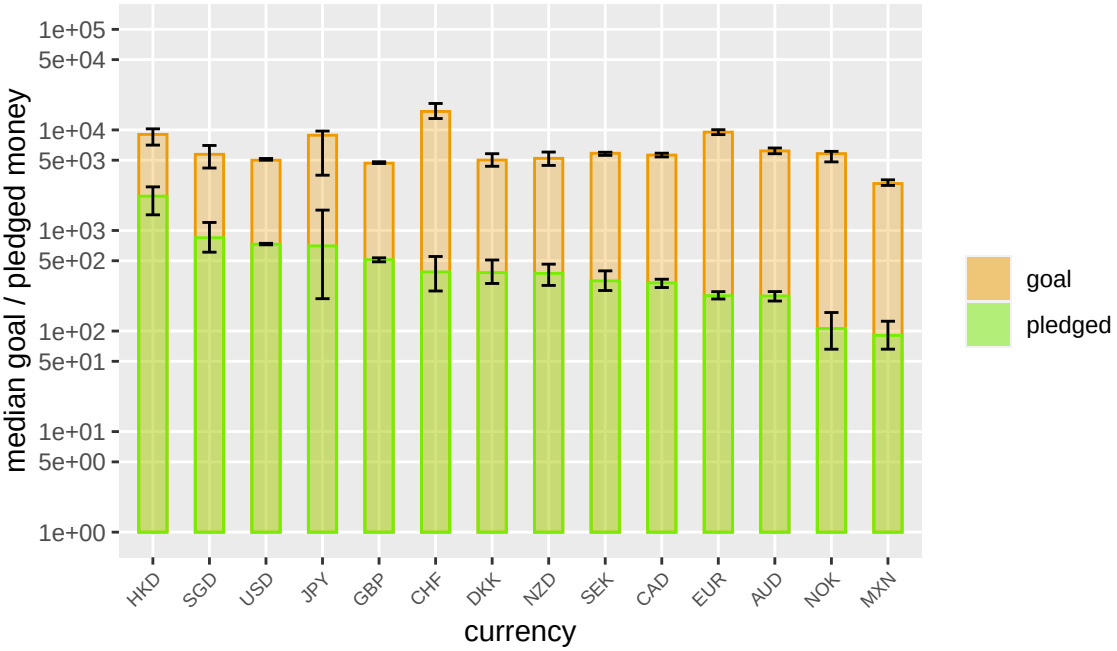
It appears that, regarding either the country or the currency of the projects, they are coming primarily from the United-States and Great-Britain.



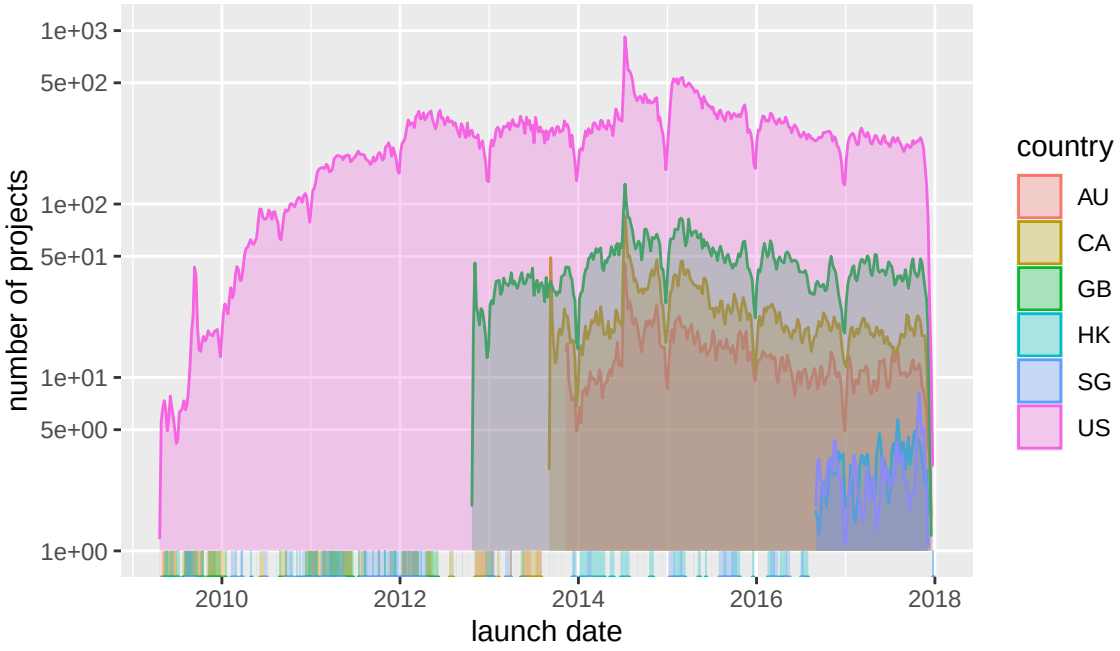


However, the two countries with the highest median pledged money are Hong-Kong and Singapor.





By comparing the distributions of the number of projects per country, spreaded along their launch date, it appears that Kickstarter opened its plateform to Great-Britain in 2012, Canada and Australia in 2013, and Hong-Kong and Singapor only in 2016. This may explain the 2014 peak in total number of projects, and the rate of goal reached for projects launched from Hong-Kong and Singapor.



3.6 Summary

For this study, the exploration of the data was restricted to the objective set at the start, the pledged money at the end of a project, and to the influence of the different variables suggested at the end of the data importation section. Cross influence between multiple variables (such as the countries and the launch dates) were generally ignored, for conciseness sake, regarding the high number of variables and the (even higher) number of possible combinations and cross interactions. These neglected investigations could have revealed other insights on the data, but we choose to stop the exploration here, in order to submit this report in time !

The data exploration allows us to reduce the number of the variable to only keep the most relevant variables to make prediction on the pledged money for a given project.

```
# selected variables
data_ks <- data_ks %>%
  select(name_nb_char, name_nb_word, name_spec_char, name_lcase,
         main_category, true_category, diff_cat,
         time_span, start_wday, end_wday, start_day, end_day, start_month, end_month,
         country,
         goal_USD, pledged_USD)
```

4 Modelling approach

4.1 Goal

The objective function for this project is simply the Root-Mean Square Error (RMSE) between the actual pledged money y and the predicted pledged money $\hat{y}$.

$$RMSE = \sqrt{\frac{1}{N}\sum(y - \hat{y})^2}$$

```
RMSE_fct <- function(test, pred) {
  sqrt(mean((test - pred) ^ 2))
}
```

The chosen modeling approaches are residuals' explanation and regression tree. The ensemble of these algorithms will also be evaluated.

To avoid overfitting, the data set is divided into a training set (`train_set_final`) and a testing set (`test_set_final`). The models are trained on the training set, and the best one will then make predictions for the projects on the testing set. To compare the performance of each model, and optimize their potential tuning parameters without overfitting on the training set, the training is itself divided into a training set (`train_set`) anda testing set (`test_set`).

```
# some variable must be parsed into classified variables, to ensure common
# variables values in both train and test sets (necessary for certain models)
# (little foreshadowing: log is applied on goal_USD and pledged_USD (1 is added
# to avoid -Inf values))
data_ks <- data_ks %>%
  mutate(goal_log = log10(1 + goal_USD),
         pledged_log = log10(1 + pledged_USD),
         goal_log_class = cut(goal_log,
                              breaks = seq(0, 10, 0.001),
                              include.lowest = TRUE))

# seed for reproducible results
set.seed(2486)

# test_set_final contains 10% of the data
test_index <- createDataPartition(data_ks$pledged_USD,
                                  times = 1,
                                  p = 0.1,
                                  list = FALSE)
train_set_final <- data_ks[-test_index, ]
temp <- data_ks[test_index, ]
remove(test_index)

# train & test should have common variables values
test_set_final <- temp %>%
  semi_join(train_set_final, by = "name_nb_char") %>%
  semi_join(train_set_final, by = "name_nb_word") %>%
  semi_join(train_set_final, by = "name_spec_char") %>%
  semi_join(train_set_final, by = "name_lcase") %>%
  semi_join(train_set_final, by = "main_category") %>%
  semi_join(train_set_final, by = "true_category") %>%
  semi_join(train_set_final, by = "diff_cat") %>%
  semi_join(train_set_final, by = "time_span") %>%
```

```
  semi_join(train_set_final, by = "start_wday") %>%
  semi_join(train_set_final, by = "end_wday") %>%
  semi_join(train_set_final, by = "start_day") %>%
  semi_join(train_set_final, by = "end_day") %>%
  semi_join(train_set_final, by = "start_month") %>%
  semi_join(train_set_final, by = "end_month") %>%
  semi_join(train_set_final, by = "country") %>%
  semi_join(train_set_final, by = "goal_log_class")

# Add rows removed from final test set back into final train set
removed <- anti_join(temp,
                     test_set_final)
train_set_final <- rbind(train_set_final, removed)

# subsets
# test_set contains 10% of train_set_final
test_index <- createDataPartition(train_set_final$pledged_USD,
                                  times = 1,
                                  p = 0.1,
                                  list = FALSE)
train_set <- train_set_final[-test_index, ]
temp <- train_set_final[test_index, ]
remove(test_index)

# train & test should have common variables values
test_set <- temp %>%
  semi_join(train_set, by = "name_nb_char") %>%
  semi_join(train_set, by = "name_nb_word") %>%
  semi_join(train_set, by = "name_spec_char") %>%
  semi_join(train_set, by = "name_lcase") %>%
  semi_join(train_set, by = "main_category") %>%
  semi_join(train_set, by = "true_category") %>%
  semi_join(train_set, by = "diff_cat") %>%
  semi_join(train_set, by = "time_span") %>%
  semi_join(train_set, by = "start_wday") %>%
  semi_join(train_set, by = "end_wday") %>%
  semi_join(train_set, by = "start_day") %>%
  semi_join(train_set, by = "end_day") %>%
  semi_join(train_set, by = "start_month") %>%
  semi_join(train_set, by = "end_month") %>%
  semi_join(train_set, by = "country") %>%
  semi_join(train_set, by = "goal_log_class")

# Add rows removed from final test set back into final train set
removed <- anti_join(temp,
                     test_set)
# print(nrow(removed))
train_set <- rbind(train_set, removed)
remove(temp, removed)

# final ratios (close to 10%)
nrow(test_set) / nrow(train_set)
```

```
## [1] 0.110828
```

```r
nrow(test_set_final) / nrow(train_set_final)
```

```
## [1] 0.110826
```

4.2 Residuals' explanation

4.2.1 Naive approach

First, the naive approach for residuals' explanation consists in predicting that any amount of pledged money is just the average over all amounts of pledged money μ, and errors ϵ are supposed to be residual random noise:

$$\hat{y} = \mu + \epsilon$$

As presented at the start of the data exploration section, the `pledged_USD` variable covers more than 7 orders of magnitude, which incited us to use the median instead of the mean, for a robust description of its distribution. Another naive approach is tested, in which the predictions are the median m over every amount of pledged money:

$$\hat{y} = m + \epsilon$$

```r
# naive model (mean)
mod_naive_avg <- function(train, test) {
  mu_pledged <- mean(train$pledged_USD)
  test <- test %>%
    mutate(pred = mu_pledged)
  test$pred
}

# naive model (median)
mod_naive_med <- function(train, test) {
  m_pledged <- median(train$pledged_USD)
  test <- test %>%
    mutate(pred = m_pledged)
  test$pred
}
```

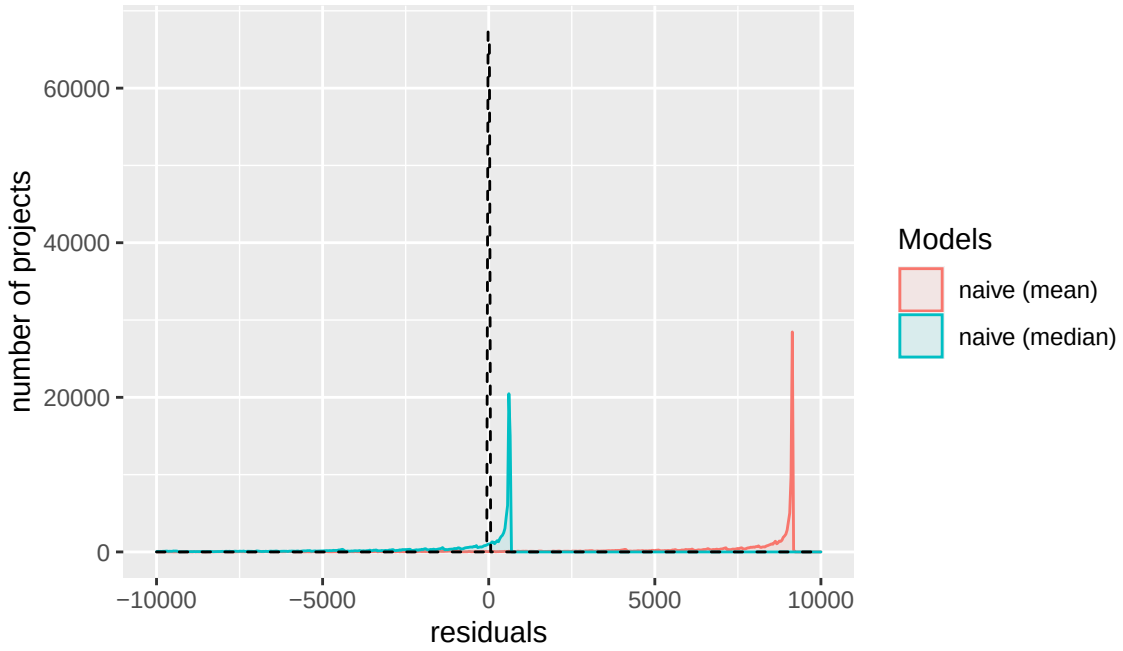
```r
res <- data.frame()

y_naive_avg <- mod_naive_avg(train_set, test_set)
res[1, 1] <- RMSE_fct(test_set$pledged_USD, y_naive_avg)
y_naive_med <- mod_naive_med(train_set, test_set)
res[1, 2] <- RMSE_fct(test_set$pledged_USD, y_naive_med)
```

If this model generates accurate predictions, residuals should be truly random, and it can be verified by checking its distribution, which therefore should be normal and centered at zero. If it is not the case, it means that the residuals still contain dormant information that have not been grasped by our model, and it could be improved. This test can be performed visually, by plotting the distribution of the residuals, but a metric should ease the comparison between models.

The Kolmogorov-Smirnov (KS) test has been chosen for this task. The KS test gives the maximum difference between the residuals' cumulative distribution and a normal cumulative distribution (centered at zero). A derivative metric is also evaluated: the relative error between the result of the KS test on the residuals, and the threshold under which the hypothesis of the residuals' distribution being a normal distribution cannot be rejected, with a 95% level of confidence. This second metric will be a clear indication on how much the residuals' distribution is different from a normal distribution.

By plotting the residuals' distribution for the naive approaches, it is obvious that they are significantly not normal (the dashed line represents the ideal normal distribution). Both approaches present a strongly

asymetric distribution. Also, the naive approach based on the mean is several orders of magnitude distant from zero.



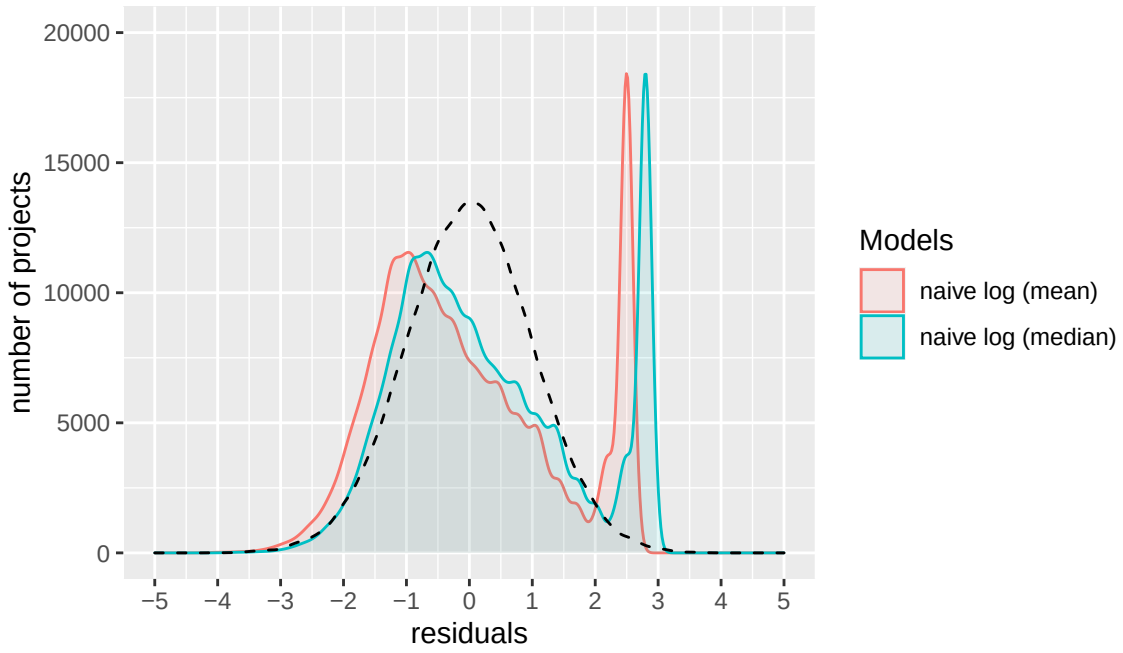
To bypass the large magnitude span of this variable, the derivative variable `pledged_log` is used (1 is added to avoid `-Inf` values for projects that pledged no money):

$$pledged\_log = log_{10}\left(pledged\_USD + 1\right)$$

```
# naive model (log mean)
mod_naive_avg_log <- function(train, test) {
  mu_pledged <- mean(train$pledged_log)
  test <- test %>%
    mutate(pred = mu_pledged)
  test$pred
}

# naive model (log median)
mod_naive_med_log <- function(train, test) {
  m_pledged <- median(train$pledged_log)
  test <- test %>%
    mutate(pred = m_pledged)
  test$pred
}
```

With the `pledged_log` variable, the graph is way closer to a normal distribution (both the mean or the median are very close to zero). However, the peaks around 3 are likely to represent the projects that receive no money (positive residuals mean an overestimation of the predictions).

Both residuals' distributions (mean and median) are very different from a normal distribution, which leaves room for improvement, brought through explicit biais' introduction. To avoid dealing with redondant models, the median based approach is discarded, as it provides similar results than the mean based approach for the `pledged_log` variable.



As expected, the naive approach is a limited model, and a lot of information remain dormant in the residuals, which can not be considered as random noise in this case.

Models	RMSE	KS	err_KS [%]
Residuals_naive	1.44083	0.096501	1204.99

4.2.2 One biais

One way to improve our model is to explicitely take into account the effect of certain variables as biaises. Data exploration revealed that the amount of pledged money can change following the different variables selected as the end of the data exploration.

To compare the influence of these variables v, 16 different models, one for each variable, will be tested. Each of these models will use an hyperparameter λ, to regularise the biaises b_v.

```
lambda_opt <- function(train, test, fct_mod) {
  # This function gives the optimal lambda (and RMSE) for given
  # train/test sets and a model. The lambda associated with the lowest RMSE
  # (with a given threshold) is designated as optimal.

  # threshold
  prec <- 0.1

  # initialization
  l_new <- 0
  i <- 2

  y <- fct_mod(train, test, l_new)
  rmse_l_new <- RMSE_fct(test$pledged_log, y)

  out <- data.frame(l = l_new,
                    rmse = rmse_l_new)
```

```
  # loop while lambda's transformation is above the threshold
  while (abs(i) > prec) {
    # short-term storage of last values (lambda, RMSE) for comparison
    l_old <- l_new
    rmse_l_old <- rmse_l_new

    # new lambda
    l_new <- l_old + i

    # RMSE with new lambda
    y <- fct_mod(train, test, l_new)
    rmse_l_new <- RMSE_fct(test$pledged_log, y)

    if (rmse_l_new > rmse_l_old) {
      # change in the "direction" and "amplitude" of the modification of lambda
      i <- -i / 2
    }
    # long-term storage of values (lambda, RMSE)
    out <- rbind(out,
                 c(l_new,
                   rmse_l_new))
  }
  # return penultimate values (lambda, RMSE), which the minimum RMSE reached
  out[nrow(out) - 1, ]
}
```

The different models follow the formula:

$$\hat{y}_v = \mu + b_v + \epsilon$$

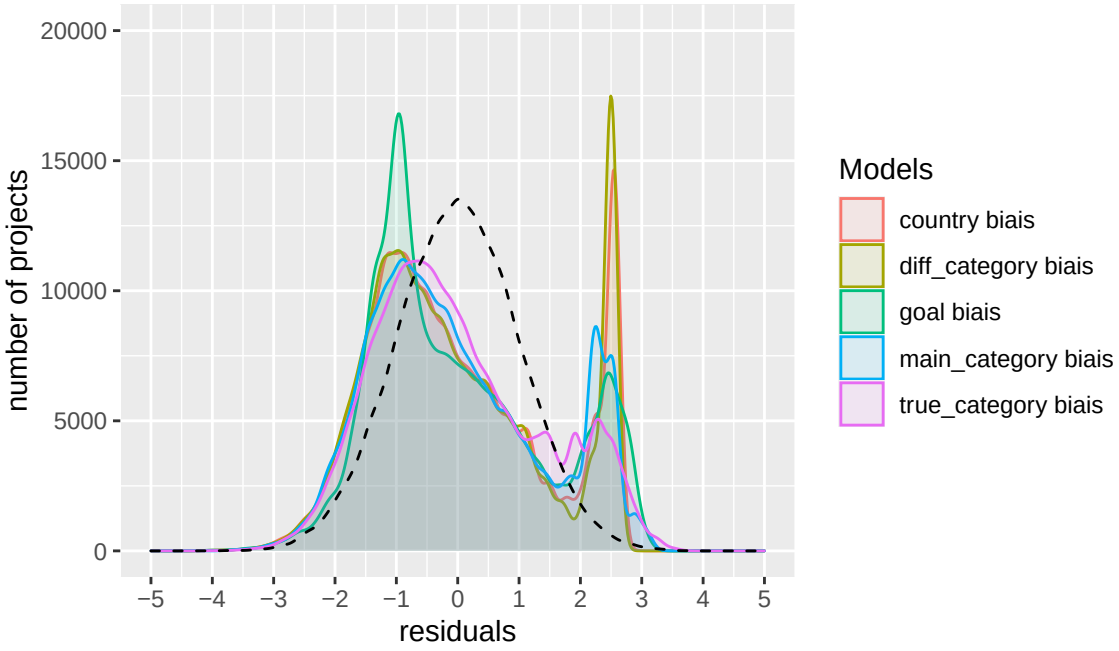
where

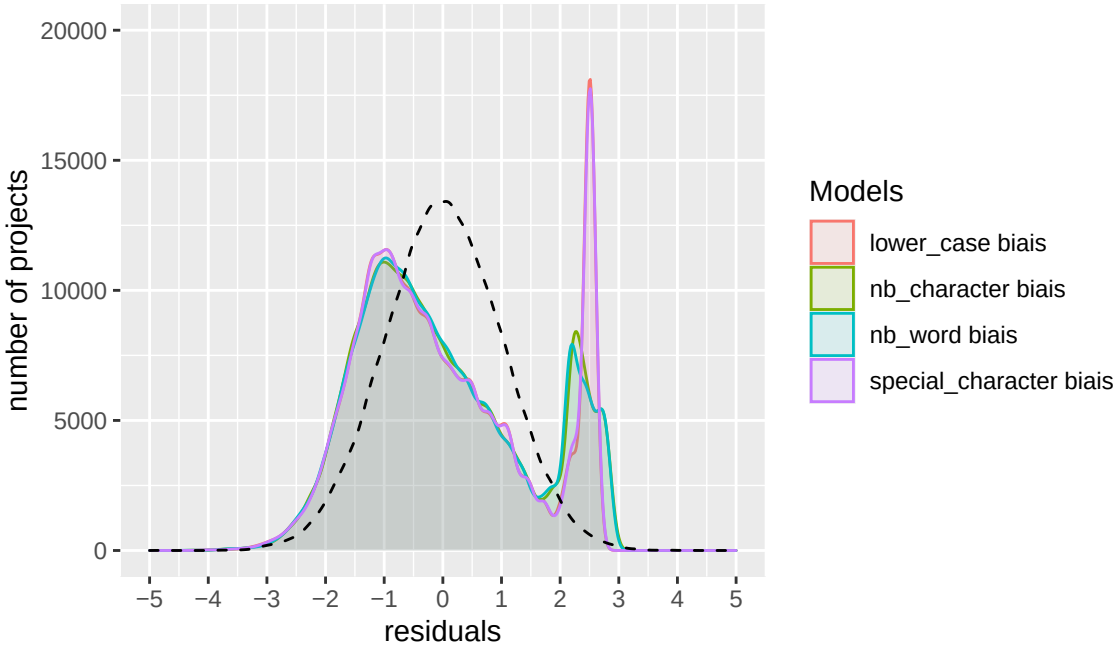
$$b_v = \frac{1}{\lambda + N} \sum^{N} (y - \mu)$$

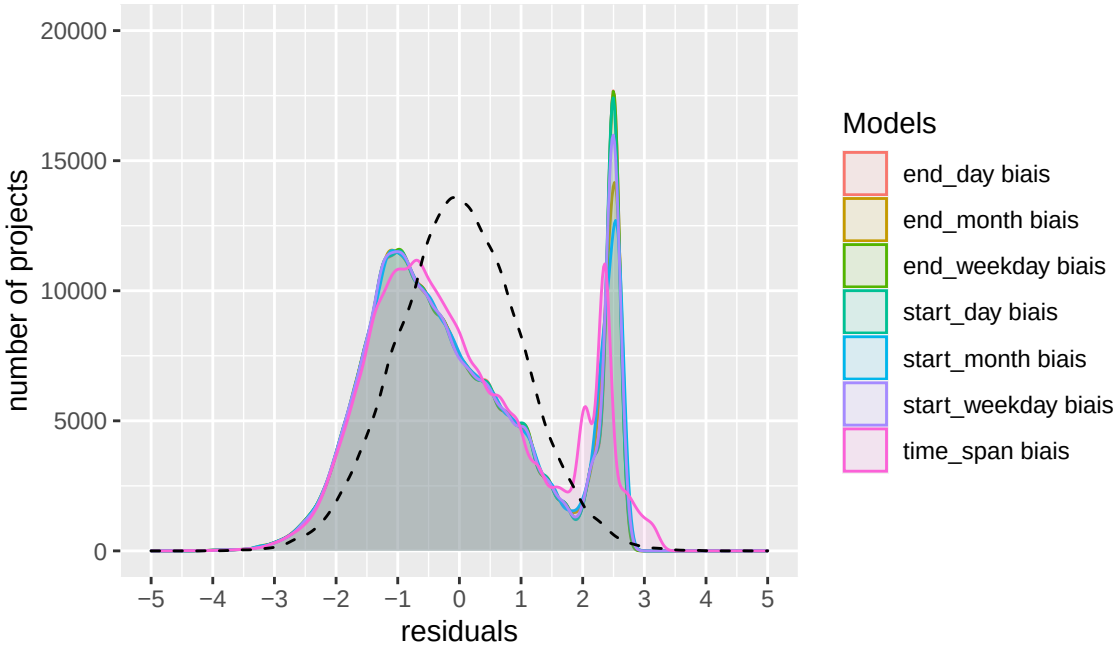
```
mod_b_goal <- function(train, test, lambda) {
  mu_pledged <- mean(train$pledged_log)
  goal_biais <- train %>%
    group_by(goal_log_class) %>%
    summarise(b_goal = sum(pledged_log - mu_pledged) / (n() + lambda))

  test <- test %>%
    left_join(goal_biais, by = "goal_log_class") %>%
    mutate(pred = mu_pledged + b_goal)

  test$pred
}
```

These models are trained on the `train_set` and the RMSEs are calculated with the `test_set` data set.

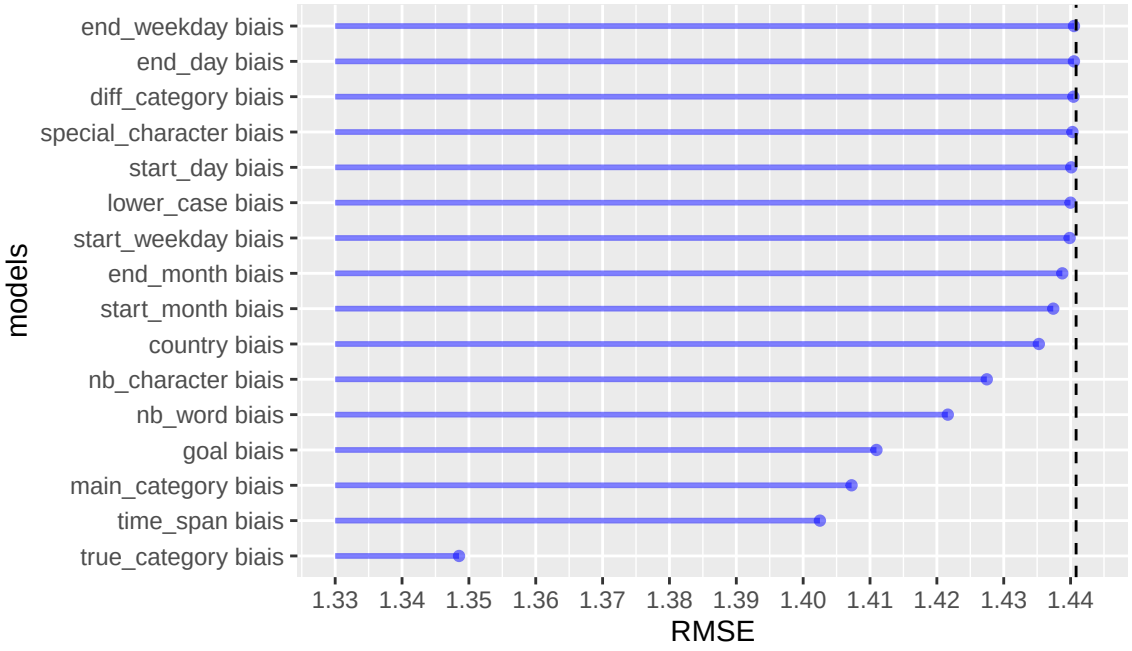
From only the graphical representation of the residuals' distributions (plotted in three distinct graphs to improve readability), it appears that some variables have nearly no impact, as the peak around 3 is not displaced or lowered. A few variables, such as the goal, the "true" category, the time span or the number of words in the name of the project, have an influence, as the peak around 3 is clearly smoothed out (although never completely).


20000
15000
10000
5000
0
number of projects
−5
−4
−3
−2
−1
0
1
2
3
4
5
residuals
Models
country biais
diff_category biais
goal biais
main_category biais
true_category biais


20000
15000
10000
5000
0
number of projects
−5
−4
−3
−2
−1
0
1
2
3
4
5
residuals
Models
lower_case biais
nb_character biais
nb_word biais
special_character biais



The RMSEs obtained from every single biais models are plotted below (the black dashed line represents the RMSE for the naive approach model). As expected, the variables whose residuals' distributions were the most transformed display the lowest RMSEs, confirming their relevancy regarding data explanation through biaises. The other variables show only a very tenuous RMSE improvement from the naive approach.



The new lowest RMSE is obtained with a single biais model based on the "true" category of the project. Since other variables demonstrated some influence on the RMSE of the predictions, the next step is to take into account multiple biaises in a single model.

Models	RMSE	KS	err_KS [%]
Residuals_naive	1.44083	0.096501	1204.991
Residuals_single_b	1.34852	0.067165	808.285

4.2.3 Multiple biaises

To compare the influence of the combined biaises b_{v_i}, two different models will be tested: one with only the 8 variables that displayed the lowest RMSEs for single biais models, and the other with all the 16 variables. Each of these models will use an hyperparameter λ, to regularise the biaises.

They follow the formula:

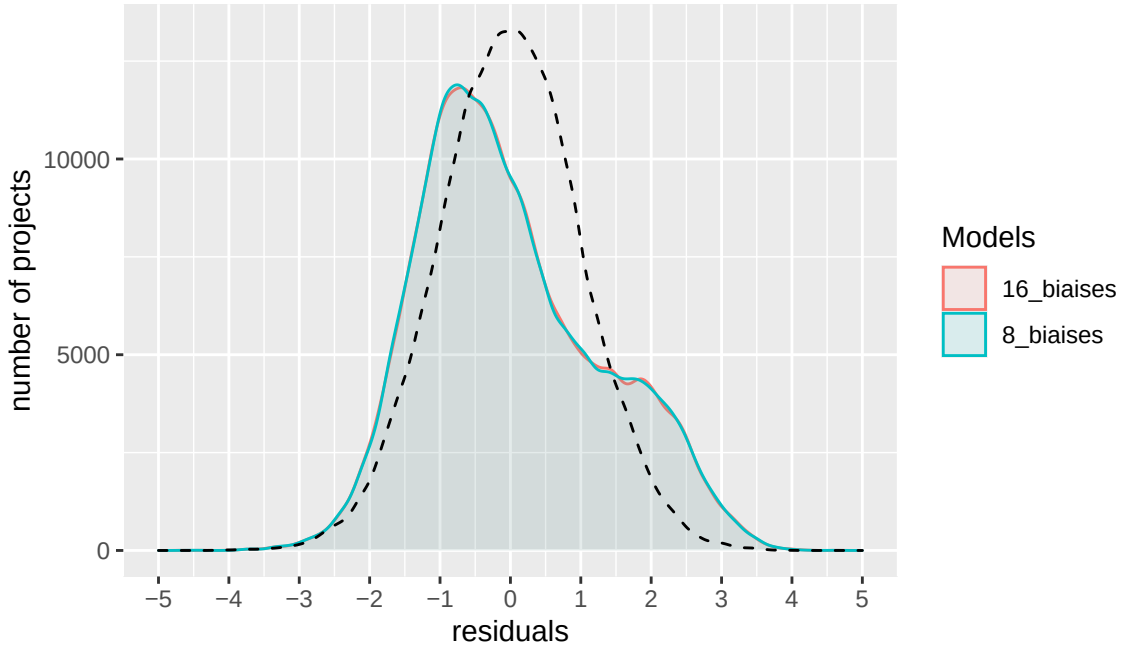
$$\hat{y}_{v_1,\ldots,v_{8,16}} = \mu + \sum_{i}^{8,16} b_{v_i} + \epsilon$$

where

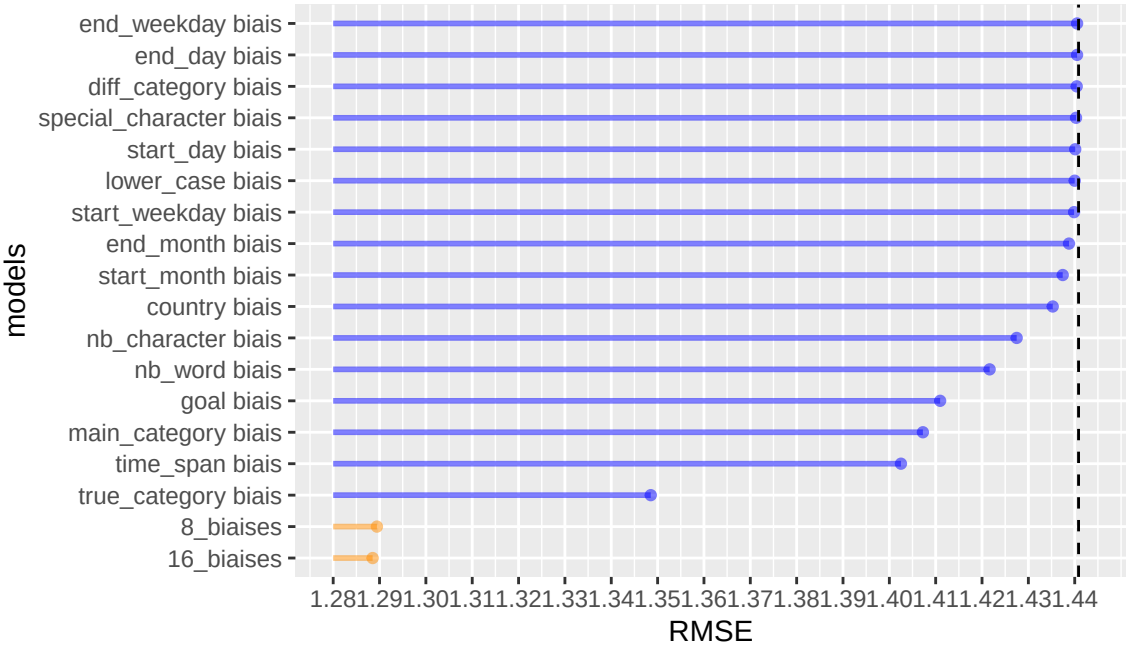
$$b_{v_i} = \frac{1}{\lambda + N} \sum^{N} \left(y - \mu - \sum_{j<i}^{8,16} b_{v_j} \right)$$

These models are trained on the `train_set` and the RMSEs are calculated with the `test_set` data set.

From only the graphical representation of the residuals' distributions, the peak around 3 is almost completely smoothed out, and the general shape comes closer to the bell shape of a normal distribution. However, the distributions show that the models still, at the same time, overestimate and underestimate some predictions, which means that some dormant information still lies in the data.



The model combining all the 16 variables gives a lower RMSE than the model combining only 8 variables. It is a significant improvement from the single biais models (lowest RMSE represented by the blue dashed line).



The results from the 16 combined biaises model are saved. It appears that, even if the RMSE is lower with multiple biaises, the KS test shows that somehow the single biais model has, according to this test, a residuals' distribution closer to a normal distribution. This might be explained by the longer tail toward positive residuals for the multiple biais model, which might result in a higher maximum difference between its residuals' distribution and a normal distribution.

Models	RMSE	KS	err_KS [%]
Residuals_naive	1.44083	0.096501	1204.991
Residuals_single_b	1.34852	0.067165	808.285
Residuals_multi_b	1.28944	0.067673	815.145

The residuals' explanation modeling approach could be further enhanced using matrix factorization on the residuals of the multiple biaises model. Unfortunately, this method is very computationaly expensive, and my calculation power is too limited to expect a result before the deadline for this study. So instead of perfecting the residuals' explanation models, another king of model is attempted.

4.3 Regression tree

A regression tree (RT) model consists in segmenting the predictor space into a number of simple "regions" of the objective numeric variable (`pledged_log`). Once the RT model is trained, the segmentation rules are used as explicit rules to make predictions. This kind of model has the advantage to be useful for interpretation, so it will be interesting to compare this approach with the residuals' explanation approach.

A RT model has a tuning paramter (complexity parameter), which will be optimized using cross-validation. The RT model is trained on the `train_set` and the RMSEs are calculated with the `test_set` data set.

```
cl <- makePSOCKcluster(6)
# parallel processing decreases significantly the calculation time,
# thus allowing to optimize evezn more the tuning parameter for the same
# period of time.

# time <- Sys.time() # 30 min
```

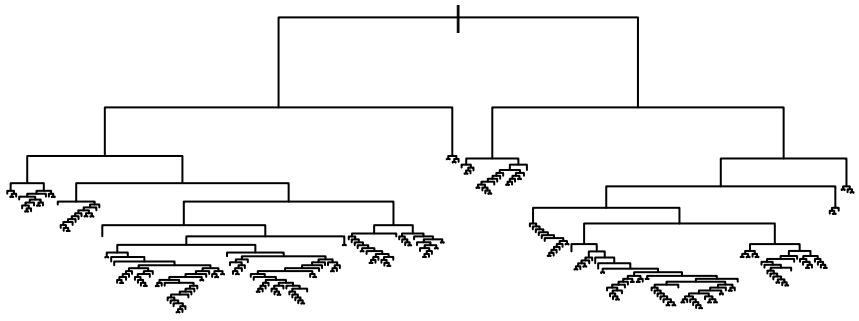
```r
# seed for reproducible results
set.seed(2486)

registerDoParallel(cl)

mod_tree <- train(pledged_log ~ .,
                  method = "rpart",
                  tuneGrid = data.frame(cp = seq(0, 0.001, len = 100)),
                  data = train_set[, -c(1, 17, 18, 20)])
stopCluster(cl)

# Sys.time() - time

y_tree <- predict(mod_tree, test_set, type = "raw")

res[1, 1] <- RMSE_fct(test_set$pledged_log, y_tree)
```

Unfortunately, the RT model with the optimized complexity parameter is very large, with around 300 leaves, so it won't be as interpretable as expected. By pruning the RT model (to the point of being barely readable), it appears that the prominent variables are the number of words in the project's name, the "true" category, the goal, the time span and the main category. This result is coherent with the residual's explanation approach.

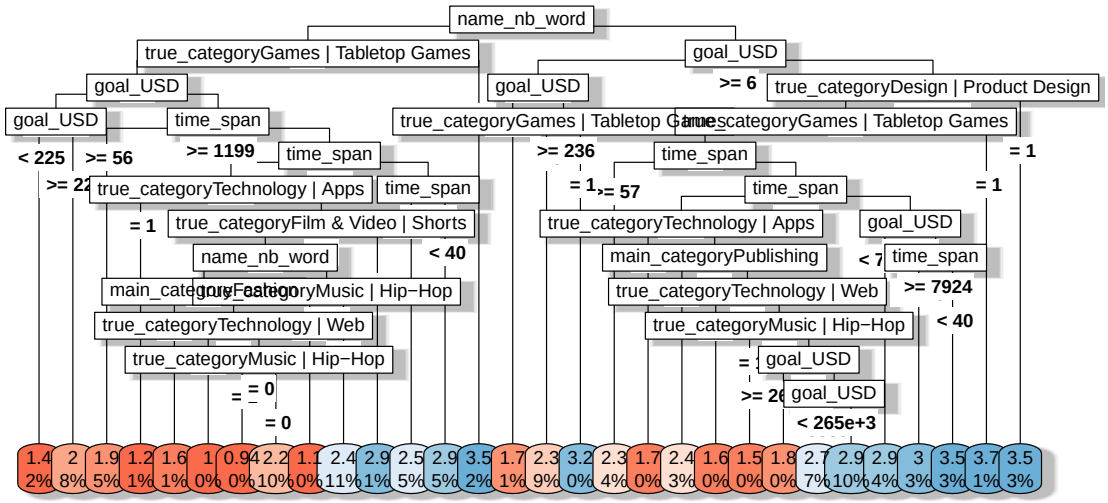
```r
# complexity parameter optimization's trace
# ggplot(mod_tree)

# number of leaves
sum(mod_tree$finalModel$frame$var == "<leaf>")
```

```
## [1] 286
```

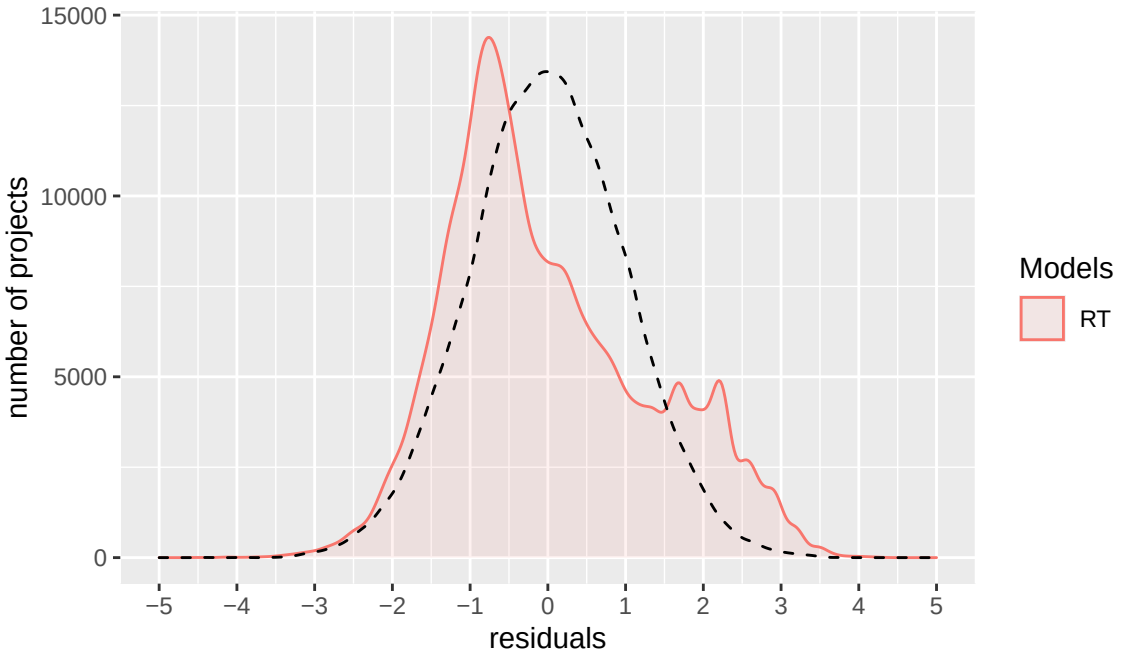
```r
# RT model overview
plot(mod_tree$finalModel)
```



```
# pruned RT model
rpart.plot(prune(mod_tree$finalModel, cp = 0.001),
           box.palette = "RdBu",
           shadow.col = "gray",
           tweak = 3,
           type = 5)
```



From only the graphical representation of the residuals' distributions, the peak around 3 is almost completely smoothed out, and the general shape is close to the distribution of the multiple biaises model in the residuals' explanation approach.



The RMSE obtained with the RT model is lower than the best single biais model, but the multiple biaises model has the lowest RMSE so far (and is 5 times quicker to train than the RT model).

Models	RMSE	KS	err_KS [%]
Residuals_naive	1.44083	0.096501	1204.991
Residuals_single_b	1.34852	0.067165	808.285
Residuals_multi_b	1.28944	0.067673	815.145
Regression_Tree	1.30454	0.090318	1121.377

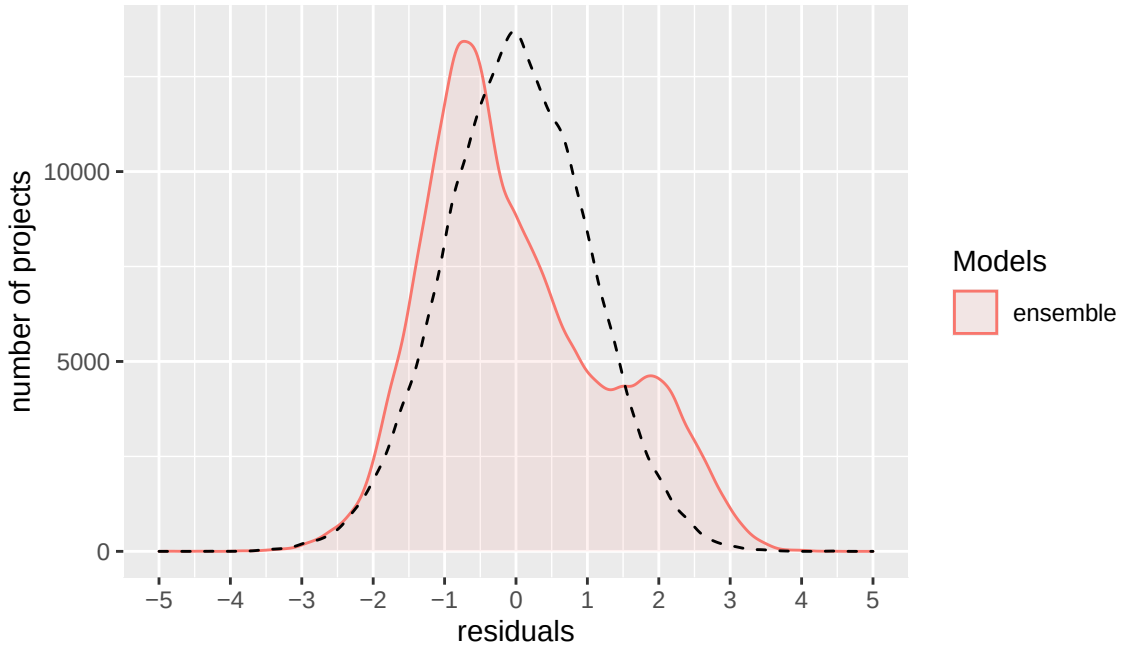
Another approach that was considered was the random forest approach, but since a single RT model took around 30 min (on 6 parallel threads) to be trained, we are looking at a 5 hours training for only 10 trees... The lack of calculation power makes the random forest approach incompatible with the upcoming deadline of this study.

4.4 Ensemble

Both previous approaches provided close RMSEs, and an ensemble prediction might provide an even lower RMSE, with no additional costly calculations.

```
y_ens <- as.data.frame(cbind(y_16, y_tree)) %>%
  mutate(ens = (y_16 + y_tree) / 2) %>%
  select(ens)

res[1, 1] <- RMSE_fct(test_set$pledged_log, y_ens$ens)
```

As expected, the mean of two similar distribution gives another similar distribution. No visible gains are detected on this graph.



The RMSE obtained with the ensemble model is indeed lower than the best model so far.

Models	RMSE	KS	err_KS [%]
Residuals_naive	1.44083	0.096501	1204.991
Residuals_single_b	1.34852	0.067165	808.285
Residuals_multi_b	1.28944	0.067673	815.145
Regression_Tree	1.30454	0.090318	1121.377
Ensemble (residuals' explanation + regression tree)	1.27758	0.084135	1037.770

Eventually, even if the RT model is more computational costly and provides a greater RMSE than a multiple biaises model, it still brings a little improvement when used in an ensemble model, with the multiple biaises model.

The selected model for this study is then the ensemble model averaging a multiple biaises model and a RT model.

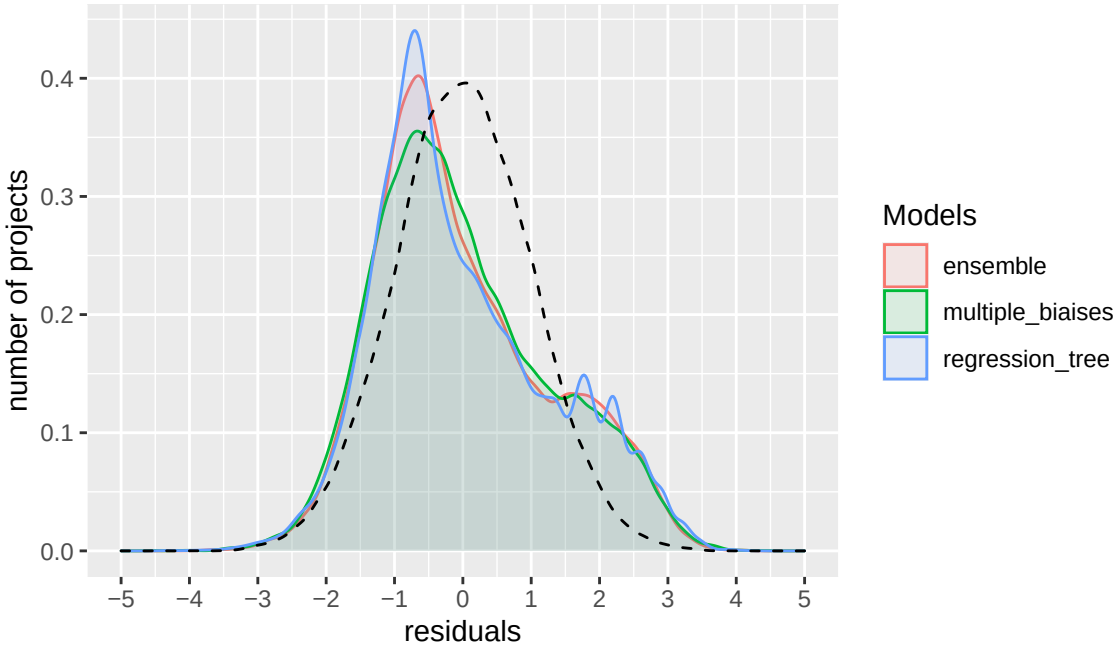
5 Results

For reasons presented in the previous section, an ensemble model averaging a multiple biaises model and a RT model. have been selected.

The selected model is now trained on the `train_set_final` data set, and the final RMSEs are calculated with the `test_set_final` data set.

```
res <- data.frame()

# residuals' explanation
l_opt <- lambda_opt(train_set_final, test_set_final, mod_b_16) # 4min
res[1, 1] <- l_opt[1, 2] # 1.28069
y_16 <- mod_b_16(train_set_final, test_set_final, l_opt[1, 1])

# regression tree
cl <- makePSOCKcluster(6)
# time <- Sys.time() # 30min (6 threads)
set.seed(2486)
registerDoParallel(cl)

mod_tree <- train(pledged_log ~ .,
                  method = "rpart",
                  tuneGrid = data.frame(cp = seq(0, 0.001, len = 100)),
                  data = train_set_final[, -c(1, 17, 18, 20)])
stopCluster(cl)
# Sys.time() - time

y_tree <- predict(mod_tree, test_set_final, type = "raw")
res[1, 2] <- RMSE_fct(test_set_final$pledged_log, y_tree) # 1.29838

# ensemble
y_ens <- as.data.frame(cbind(y_16, y_tree)) %>%
  mutate(ens = (y_16 + y_tree) / 2) %>%
  select(ens)

res[1, 3] <- RMSE_fct(test_set_final$pledged_log, y_ens$ens) # 1.27115
```

On the residuals' graph, the benefits of the ensemble model are showing, as the local maxima around 2 are smoothed out, and the global maximum is also corrected and smoothed by the average operation.



The final RMSE is presented in the table below. Its value is very close to the best RMSE during the training of the models, indicating that no overfitting occured.

Model	RMSE
Ensemble (residuals' explanation + regression tree)	1.27115

Since it is hard with only the RMSE to shape even a general idea about the accuracy of this model, another metric is calculated to show how accurate the final model is: the mean absolute error of the final model is around 10 USD, which is quite impressing, considering that the amount of pledged money spreads over seven orders of magnitude.

```
# mean absolute error (in USD)
10^mean(abs(resi$ensemble)) - 1
```

```
## [1] 9.91583
```

6 Conclusion

Even if crowdfunding is a trendy investment strategy, data exploration revealed the other side of the picture, with less than 40% of all projects getting funded, and around 14 % receiving no money at all. Many are called and few are chosen.

Data exploration provided usefull insights on the variables that influence the total gain of a project. The models tested on this data revealed what variables had the most influence on the pledged money. Both the data exploration and models can be enhanced, without reaching the case-by-case level yet. Some variables were deliberately overlooked, and the models were chosen partly for their "short" training time, in order to meet the deadline for this project.

The ensemble model, averaging the best testes models (multiple biaises and regression tree), gave a final RMSE value of 1.27115, which was the lowest value of all tested models. The mean absolute error with this model is around 10 USD, which tells how precise the predictions of this model are.