TMDB Box Office Prediction

Gabriel Lapointe

June 9, 2019

Contents

1 Business Overview

1.1 Context

In a world… where movies made an estimated $41.7 billion in 2018, the film industry is more popular than ever. For some movies, it's "You had me at 'Hello.' " For others, the trailer falls short of expectations and you think "What we have here is a failure to communicate."

1.2 Problem

With metadata on over 7,000 past films from The Movie Database, we have to predict their overall worldwide box office revenue.

1.3 Objective

The objective is to determine:

- What movies make the most money at the box office?
- How much does a director matter?
- How much does the budget matter?

We have to determine the movie revenue ($) in function of the budget ($) and in function of if the movie had a director as a member of the crew when they made it. Then, we have to identify let's say a top 10 in descending order of the predicted movie revenues.

2 Dataset Information

The train and test datasets come from the Kaggle competition TMDB Box Office Prediction. The objective is to know from the train dataset:

- The column names
- The number of columns
- The number of rows
- The number of values that are `NA` and empty

Knowing this information gives hints on how to analyse the data and how the features could be correlated. If there are any NA or empty values, we have to understand the meaning of NA and empty in their context. This holds also for budgets of 0$.

2.1 General Information

```r
source("Source/DatasetInformation.R")
source("Source/DataPreparation.R")
source("Source/DataTransformation.R")

library(ggplot2)
library(gridExtra)
library(dplyr)
library(tictoc)

train <- read.csv("Dataset/train.csv",
                  header = TRUE,
                  stringsAsFactors = FALSE,
                  row.names="id")

test <- read.csv("Dataset/test.csv",
                  header = TRUE,
                  stringsAsFactors = FALSE,
                  row.names="id")

set.seed(1234)

## Remove scientific notation (e.g. E-005).
options(scipen = 999)

PrintDatasetInformation(train)
```

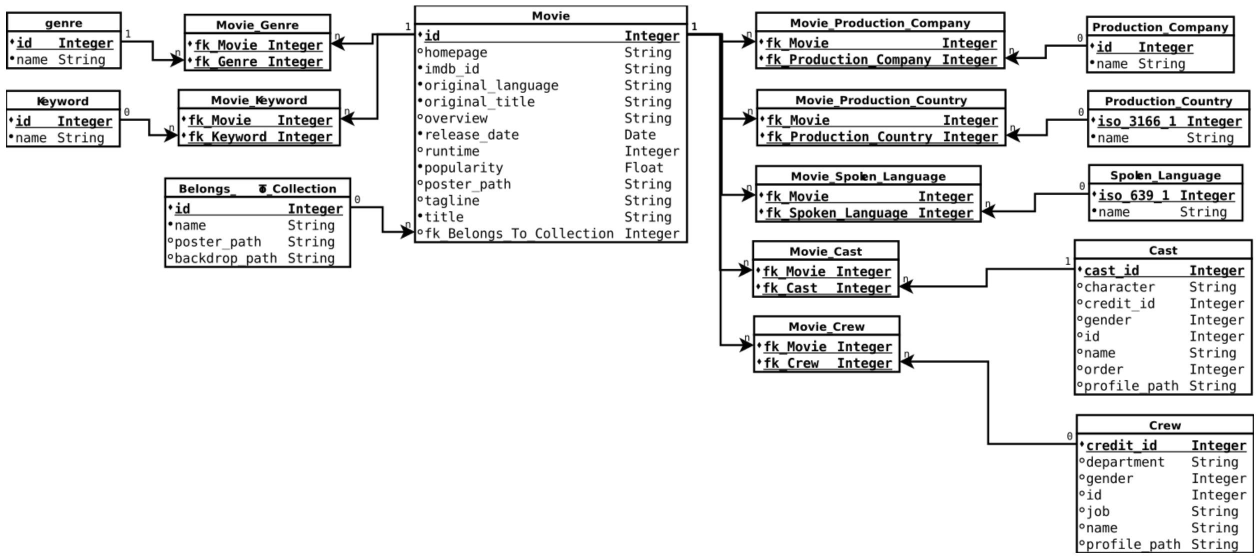
```
Number of rows:  3000
Number of columns:  22

                           Type Number_of_NA Number_of_empty
belongs_to_collection character            0            2396
budget                  integer            0               0
genres                character            0               7
homepage              character            0            2054
imdb_id               character            0               0
original_language     character            0               0
original_title        character            0               0
overview              character            0               8
popularity              numeric            0               0
poster_path           character            0               1
production_companies  character            0             156
production_countries  character            0              55
release_date          character            0               0
runtime                 integer            2              NA
spoken_languages      character            0              20
status                character            0               0
tagline               character            0             597
title                 character            0               0
Keywords              character            0             276
cast                  character            0               0
crew                  character            0               3
revenue                 integer            0               0
```

```
PrintDatasetInformation(test)
```

```
Number of rows:  4398
Number of columns:  21

                           Type Number_of_NA Number_of_empty
belongs_to_collection character            0            3521
budget                  integer            0               0
genres                character            0              16
homepage              character            0            2978
imdb_id               character            0               0
original_language     character            0               0
original_title        character            0               0
```

```
overview             character          0           14
popularity             numeric          0            0
poster_path          character          0            1
production_companies character          0          258
production_countries character          0          102
release_date         character          0            1
runtime                integer          4           NA
spoken_languages     character          0           42
status               character          0            2
tagline              character          0          863
title                character          0            3
Keywords             character          0          393
cast                 character          0            0
crew                 character          0            9
```

Here is the relational model of the dataset that is described in more details in sub-sections below.



2.2 Possible Algorithms

We know that the objective is to predict the revenue of every movie in the test dataset. We also know that:

- The output is the revenue of every movie.
- The output is a real number.
- The accuracy of the prediction is important over the speed (which should not be neglected)

Therefore, we need algorithms in the supervised learning and regression category. In this analysis, we consider the following models considering that the speed has to be taken in consideration:

- Linear regression
- Random Forest
- Extreme Gradient Boosting Trees

2.3 Code Book

2.3.1 Movie Collection Information

The feature `Belongs_to_collection` contains the following properties as a JSON array:

- id: The movie collection ID (Integer)
- name: The movie collection name (string)
- poster_path: The movie collection poster image path(string)
- backdrop_path: The movie collection backdrop path (string)

The value of this feature is empty if the movie is not in a collection (e.g.The movie collection Back to the future 1, 2, and 3). We deduce that:

- A movie is in its own collection if and only if the feature `belongs_to_collection` is empty.
- A movie cannot be in more than one collection.
- A collection may contain many movies.

2.3.2 Movie Genres Information

The feature `genres` contains the following properties as a JSON array:

- id: The genre ID (integer)
- name: The name of the genre

According to the dataset, we deduced that:

- A movie can have one or many genres.
- A genre can contain one or many movies.
- A movie can have no genre meaning that it is not classified.

2.3.3 Production Companies Information

The feature `production_companies` contains the following properties as a JSON array:

- iso_3166_1: The production company country abbreviation (e.g. US)
- name: The production company name

According to the dataset, we deduced that:

- A movie can be produced by one or many companies.
- A production company can have produced one or many movies.
- A movie that is not associated to at least one production company could be produced by amators or particular producer producing his own movies.

2.3.4 Production Countries Information

The feature `production_countries` contains the following properties as a JSON array:

- iso_3166_1: The production country abbreviation (e.g. US) where the movie has been produced
- name: The production country name

According to the dataset, we deduced that:

- A movie can be produced in one or many countries.
- A production country can contain one or many movies.
- A movie that is not associated to at least one production country could be ???

2.3.5 Spoken Languages Information

The feature `spoken_languages` contains the following properties as a JSON array:

- iso_639_1: The spoken language country abbreviation (e.g. US)
- name: The spoken language name (could contain non ascii characters)

According to the dataset, we deduced that:

- In a movie, one or many languages can be spoken.
- A spoken language can be used in one or many movies.
- A movie that is not associated to at least one spoken language could be silent movies like Charlie Chaplin or Mr. Bean.

2.3.6 Keywords Information

The feature `Keywords` contains the following properties as a JSON array:

- id: The keyword ID (Integer)
- name: The keyword (String)

According to the dataset, we deduced that:

- One or many keywords can be used for a movie.
- One or many movies can be associated to a keyword.
- A movie that is not associated to at least one keyword could be ???.

2.3.7 Cast Information

The feature `cast` contains the following properties as a JSON array:

- cast_id: The cast ID corresponding to one character playing in the movie.
- character: The character name in the movie (could be empty)
- credit_id: The credit ID as an hexadecimal string
- gender: The character is a female (1), a male (2) or other (0)
- id: The ID of the actor playing in the movie or of an event occuring in the movie
- name: The name of the actor playing in the movie or of the event occuring in the movie
- order: The order of displaying in the credit at the end of the movie
- profile_path: Avatar of the actor of the movie as an image path

Note that an actor can play in one or many movies and a cast can contain one or many actors.

2.3.8 Crew Information

The feature `crew` contains the following properties as a JSON array:

- credit_id: The credit ID as an hexadecimal string representing a member of the crew
- department: Department name in which the member of the crew was working (e.g. Directing, Writing, Sound, etc.)
- gender: The member of the crew is a female (1), a male (2) or other (0)
- id: The ID of the member of the crew
- job: The job name of the member of the crew (e.g. Director, Producer, Writer, Screenplay, etc.)
- name: Name of the member of the crew
- profile_path: Profile image path of the member of the crew

Note that a member of the crew can be hired to help making one or many movies and a movie can contain one or many members of the crew.

3 Data Preparation

The objective of preparing the data is to clean the dataset and make the dataset workable in order to visualize the data. Preparing the dataset is represented by the following steps:

1. Detect and fix values that seem to be wrong in their context.
2. Replace the NA or empty values in the dataset by meaningful values.
3. Determine and remove columns that will not help us on the visualisation of data.
4. Add new features from existing ones.

3.1 JSON Standard Validation

The objective is to detect features that contain invalid keys and/or values and fix them with valid values without modifying the context of the data.

If we take a closer look to the JSON strings, we notice that their keys and values of type string are all surrounded by single quotes. Here is an example taken from the first observation of the feature `belongs_to_collection`: `[{'id': 313576, 'name': 'Hot Tub Time Machine Collection', 'poster_path': '/iEhb00TGPucF0b4joM1ieyY026U.jpg', 'backdrop_path': '/noeTVcgpBiD48fDjFVic1Vz7ope.jpg'}]`. This is not respecting the JSON standard which requires double quotes for string values. The library `jsonlite` is validating this standard and we get a validation error. In order to fix that, we have to replace all single quotes in all JSON strings of the dataset by double quotes. However, we have to assume that there may have single or double quotes in the string value itself.

Here is a list of syntax rules that we verify:

1. All keys have to be surrounded by double quotes `"` followed by a colon `:`, a space and its mapped value (e.g. `"id": 1`).
2. All string values have to be surrounded by double quotes `"` (e.g. `"string value"`).
3. Special characters like double quotes ("), backslashes \ and squares `#` must not be used.
4. The value `None` (e.g. `"backdrop_path": None`), `[]`, `N/A` or empty must not be used as a value. The value `null` has to be used instead.
5. Each element of an array has to start with a bracket `{` followed by a double quote `"` (e.g. `{"id"`).
6. Each element of an array has to end with a bracket `}` (e.g. `"name": "Comedy"}`).
7. Each mapping (key, value) has to be separated by a comma `,` (e.g. `"id": 18, "name": "Drama"` or `"name": null, "id": 2` or `"name": "US", "id": 1`).

For example, there are some actors having a nickname in their character name in the cast. Those are written between double quotes (e.g. in the cast of Rocky Balboa: `'cast_id': 17, 'character': 'Adrianna "Adrian" Pennino'`). Because of such a case, we have to replace all double quotes by single quotes on first step.

```r
## The feature Keywords is the only one whose name is starting with an uppercase.
## To be coherent with all of the other feature names, it has to start with a lowercase.
colnames(train)[which(names(train) == "Keywords")] <- "keywords"
colnames(test)[which(names(test) == "Keywords")] <- "keywords"

features_to_fix <- c("belongs_to_collection", "genres", "production_companies", "production_countries",
for(feature_to_fix in features_to_fix)
{
    train[, feature_to_fix] <- FixJSONStandardErrors(train[, feature_to_fix])
    test[, feature_to_fix] <- FixJSONStandardErrors(test[, feature_to_fix])
}
```

3.2 Incorrect Values

An incorrect value is a value that is invalid in the feature context. For example, the feature `crew` having an observation that gives the keywords instead or the crew. It can be seen as a misplaced value. We also validate the coherence between observations of a same feature. It could occur that for the feature `genres` that all observations except one start with the `id` whereas one of them starts with `name` instead. This is an incoherence between this observation and the others.

Since there are too many error possibilities, we establish the following rules for every observation in every feature containing JSON strings:

- The JSON string in the feature `belongs_to_collection` has to start with the property `id` like `[{"id":` *or* be empty.
- The JSON string in the feature `genres` has to start with the property `id` like `[{"id":` *or* be empty.
- The JSON string in the feature `production_companies` has to start with the property `name` like `[{"name":` *or* be empty.
- The JSON string in the feature `production_countries` has to start with the property `iso_3166_1` like `[{"iso_3166_1":` *or* be empty.
- The JSON string in the feature `spoken_languages` has to start with the property `iso_639_1` like `[{"iso_639_1":` *or* be empty.
- The JSON string in the feature `keywords` has to start with the property `id` like `[{"id":` *or* be empty.
- The JSON string in the feature `cast` has to start with the property `cast_id` like `[{"cast_id":` *or* be empty.
- The JSON string in the feature `crew` has to start with the property `credit_id` like `[{"credit_id":` *or* be empty.

```r
features_to_validate <- list(belongs_to_collection = "id",
                             genres = "id",
                             production_companies = "name",
                             production_countries = "iso_3166_1",
                             spoken_languages = "iso_639_1",
                             keywords = "id",
                             cast = "cast_id",
                             crew = "credit_id")

PrintNumberOfInvalidValuesDetected(train, features_to_validate)
```

```
Number of detected invalid values in the feature 'belongs_to_collection': 0
Number of detected invalid values in the feature 'genres': 0
Number of detected invalid values in the feature 'production_companies': 0
Number of detected invalid values in the feature 'production_countries': 0
Number of detected invalid values in the feature 'spoken_languages': 0
Number of detected invalid values in the feature 'keywords': 0
Number of detected invalid values in the feature 'cast': 0
Number of detected invalid values in the feature 'crew': 0
```

```r
PrintNumberOfInvalidValuesDetected(test, features_to_validate)
```

```
Number of detected invalid values in the feature 'belongs_to_collection': 0
Number of detected invalid values in the feature 'genres': 0
Number of detected invalid values in the feature 'production_companies': 0
Number of detected invalid values in the feature 'production_countries': 0
Number of detected invalid values in the feature 'spoken_languages': 0
Number of detected invalid values in the feature 'keywords': 0
Number of detected invalid values in the feature 'cast': 0
Number of detected invalid values in the feature 'crew': 0
```

3.3 NA Values

The only feature containing `NA` in the train dataset is the movie runtime (6 movies among 7398). We consider the value `0` the same as the `NA` value because having a movie runtime of 0 minutes is impossible. We get the 2 movies from the train dataset where their runtime is `NA` and then we replace their value by a valid one taken from another source (e.g. IMDB).

```r
print(train[is.na(train$runtime), c("title", "release_date", "runtime")])
```

```
             title release_date runtime
1336                10/29/07      NA
2303 Happy Weekend      3/14/96      NA
```

```r
## Source: https://www.imdb.com/title/tt1107828/
train[1336, "runtime"] <- 130

## Source: https://www.german-films.de/filmarchive/browse-archive/view/detail/film/happy-weekend/index.
train[2303, "runtime"] <- 94

print(test[is.na(test$runtime), c("title", "release_date", "runtime")])
```

```
                            title release_date runtime
3244    La caliente niña Julietta      3/20/81      NA
4490  Pancho, el perro millonario       6/6/14      NA
4633      Nunca en horas de clase      11/3/78      NA
6818 Miesten välisiä keskusteluja       1/4/13      NA
```

```r
## Source: https://www.imdb.com/title/tt0082131/
test[244, "runtime"] <- 93

## Source: https://www.imdb.com/title/tt3132094/
test[1490, "runtime"] <- 91

## Source: https://www.filmaffinity.com/es/film267495.html
test[1633, "runtime"] <- 100

## Source: https://www.imdb.com/title/tt2192844/
test[3818, "runtime"] <- 90
```

##Empty Values Many of the features having at least one empty value are explained the following ways:

- `belongs_to_collection`: The movie does not belong to a collection of movies.
- `homepage`: The movie does not have a homapage.
- `poster_path`: The movie does not have a film poster. We assume that either the movie has a very low budget and get low revenue or they do not have this information or oversight the poster when inserting the movie in the database.
- `overview`: The movie does not have an overview. We assume that the movie is not enough popular to take the time to give an overview of the movie or it could be an oversight.
- `spoken_languages`: The movie could be a silent movie like Charlie Chaplin or Mr. Bean.
- `production_companies`: A movie self-made could be the reason why a movie does not use a production company.
- `tagline`: The does not have a tagline. We assume that a movie with a great tagline is a movie that we remember. We expect that the movie popularity will be higher. Movie without taglines are less popular and then the revenue could be lower.
- `Keywords`: No keywords describe the movie.
- `genres`: We assume that a movie not associated to genres is non classifed.
- `title`: The empty movie title could be an oversight because the original title is taken instead in some rare cases.

We assume that the empty values in the following features are oversights in the database or they just do not have the information on them:

- `production_countries`: It is impossible that a movie is produced nowhere. It could be possible that there is no information found about the production country.
- `crew`: It is impossible that a movie has been made by itself. It has to have at least one member of the crew like the producer.
- `status`: There are 7396 / 7398 movies that are relased. The 2 other movies do not have a status. In this case, we assume that they are released.
- `release_date`: There are 7397 / 7398 movies that have a release date.

We have to replace this empty date by a real date because we extract parts of the date in the data visualisation and transformation section. From the IMDB source the empty release date is replaced by the following one:

```
test$release_date[test$release_date == ""] <- "5/1/00"
```

3.4 Zero Values

Some movies budget are 0 in both datasets and need to be replaced by a significative value. In such a case, we use the median which gives better results for the train dataset than the mean. We are not using a linear regression model because the budget depends on many variables such as the popularity, the number of members in the crew, the number of characters in the casting, the production company and surely others that we do not have in our dataset.

```
cat("There are", length(train$budget[train$budget == 0]), "movies with 0 of budget. \n\n")
```

```
There are 812 movies with 0 of budget.
```

```
train$budget[train$budget == 0] <- median(train$budget[train$budget > 0])
test$budget[test$budget == 0] <- median(test$budget[test$budget > 0])
```

3.5 Useless Features

The objective is to remove features assuming that they will not be useful for the prediction.

The first one is the movie `status` because all movies in both dataset are released (except 2 of them that we assumed to be released).

```
train$status <- NULL
test$status <- NULL
```

The `imdb_id` and `poster_path` are also useless because they do not give any useful information on the movie revenue.

```
train$imdb_id <- NULL
train$poster_path <- NULL

test$imdb_id <- NULL
test$poster_path <- NULL
```

Since there are only 8 movies having no overview in the dataset, it is useless to see what is the revenue in function of if the movie has or not an overview. The same logic applies to the `original_title` and `title` features.

```
train$overview <- NULL
train$original_title <- NULL

test$overview <- NULL
test$original_title <- NULL
```

Since we want to know which production companies have made the best movies revenue, it becomes usaless to keep the production countries.

```r
train$production_countries <- NULL
test$production_countries <- NULL
```

We keep the `title` only for the next section for informational purposes. It will be removed at the end of the next section.

4 Data Visualization and Transformation

The first objective is to verify if we reject or not our assumptions by visualizing the data using scatter plots or bar charts. This will help us to know which features are helpful or useless on the predictions.

The second objective is to transform the train dataset in order to obtain a matrix of real and integer values for the predictions purposes. Each time the conclusion according to their chart will show that the feature has a significant impact on the movie revenue, the string (including the ones containing JSON) feature is transformed to a numeric feature.

4.1 Movie Revenues

Let's get a general picture on the revenue values and verify if the values are normally distributed.

```
      Min.    1st Qu.     Median       Mean    3rd Qu.       Max. 
         1    2379808   16807068   66725852   68919204 1519557910 
```

```
                                          title release_date    revenue
1                                  The Avengers      4/25/12 1519557910
2                                     Furious 7       4/1/15 1506249360
3                       Avengers: Age of Ultron      4/22/15 1405403694
4                          Beauty and the Beast      3/16/17 1262886337
5                Transformers: Dark of the Moon      6/28/11 1123746996
6                         The Dark Knight Rises      7/16/12 1084939099
7  Pirates of the Caribbean: On Stranger Tides      5/14/11 1045713802
8                                  Finding Dory      6/16/16 1028570889
9                           Alice in Wonderland       3/3/10 1025491110
10                                     Zootopia      2/11/16 1023784195
```

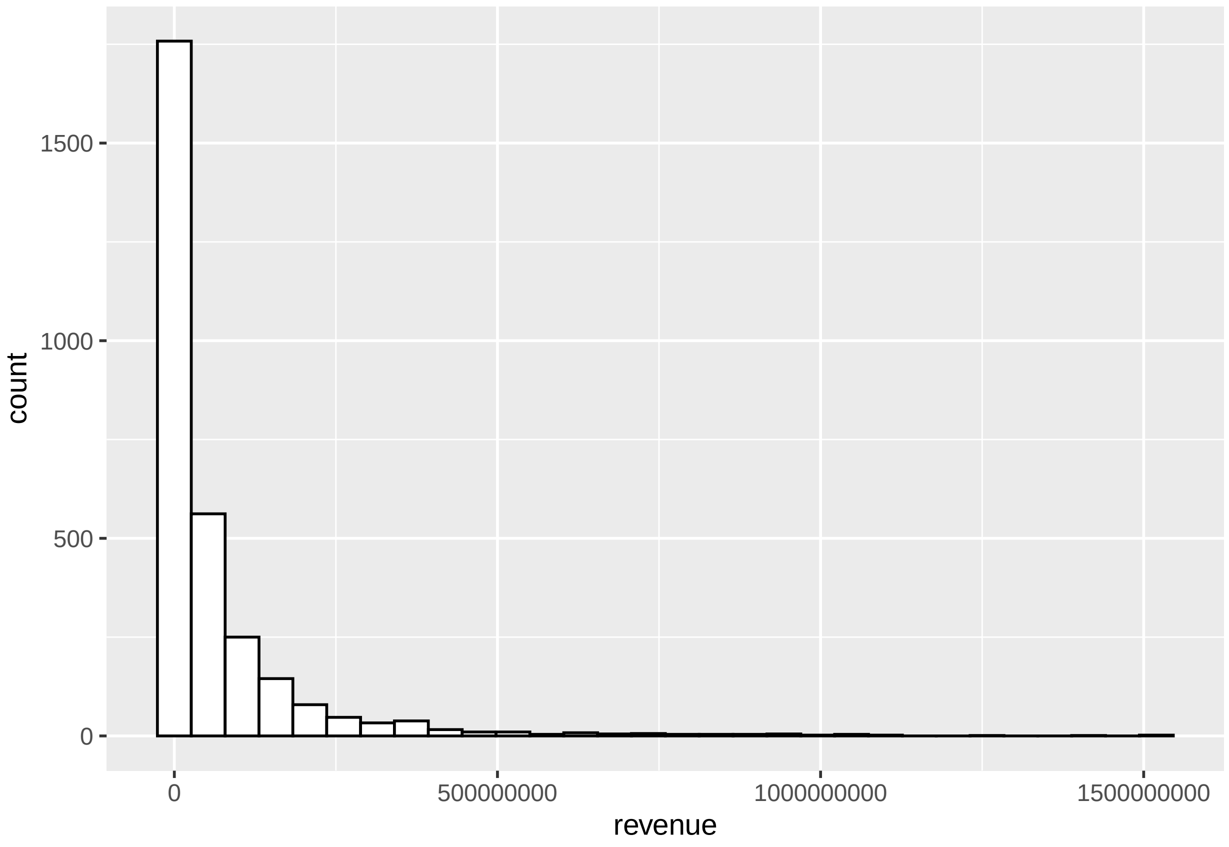
```
                        title release_date revenue
2991             East of Eden       3/9/55       5
2992          American Adobo      9/29/01       4
2993                    Saamy       5/5/03       3
2994              All at Once       6/5/14       3
2995                Borsalino      5/19/70       3
2996                Tere Naam      8/15/03       2
2997 The Wind in the Willows     10/16/96       1
2998             Mute Witness      9/28/95       1
2999                  Missing       1/1/07       1
3000          The Merry Widow      8/26/25       1
```

There is a huge gap between the mean and the median implying that the revenue is skewed. We also note that the movie `All at Once` generated 3$ of revenue on the Box-Office. According to IMDB, the movie generated 3 514 780$ worldwide. Another example is the indian movie `Saamy` which generated 3$ of revenue. Again, according to IMDB, the movie generated 510 000 000 indian rupees gross (world) which correspond to 7 406 475 US dollars today (according to the exchange rate, the revenue seems to correspond to around 11 million dollars in 2003). The following questions would need to be answered:

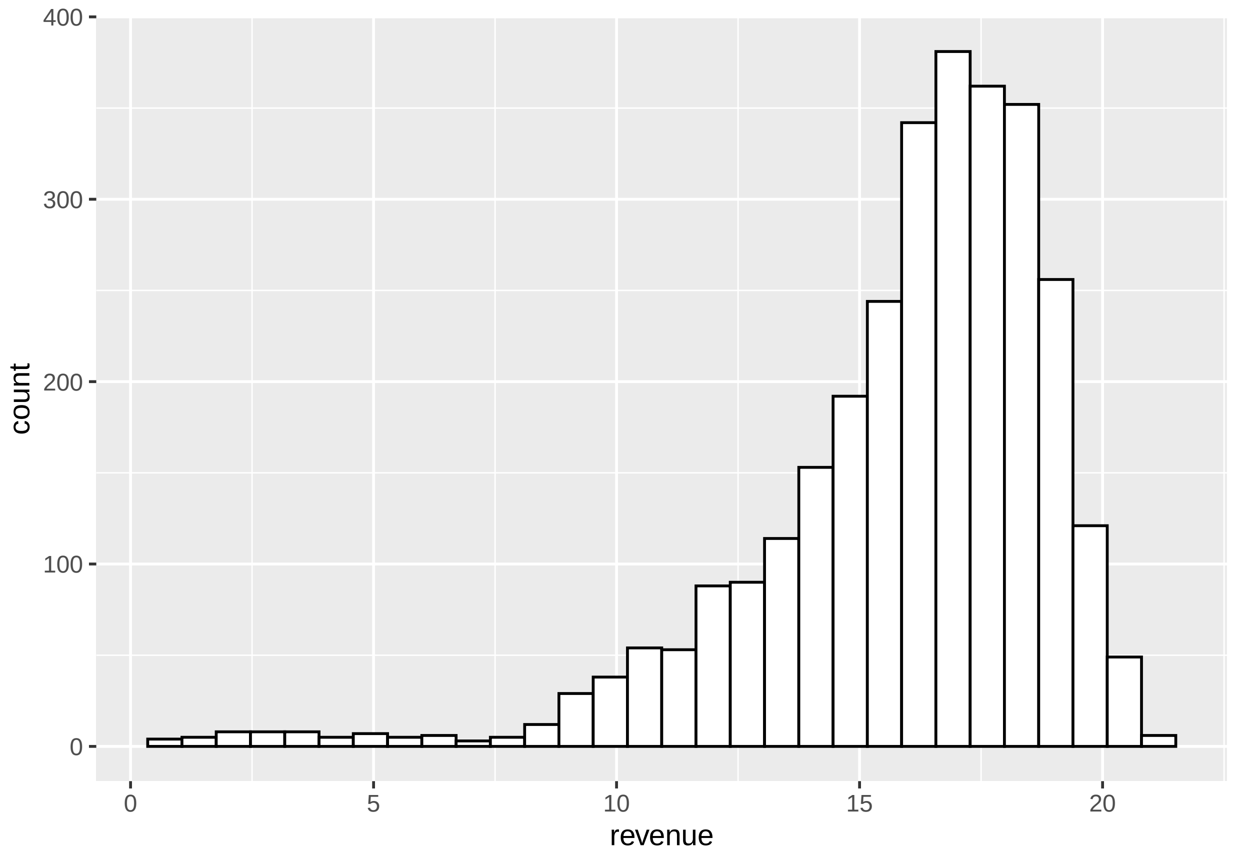
1. Are there incorrect revenues?
2. Are some revenues expressed in million of US dollars?
3. Are all revenues converted in US dollars?
4. Are all revenues converted in function of the money exchange rate when the dataset has been built or when the movie has been released?

Answers to these questions would be very useful to know how to convert them and then get much better predictions. At least knowing money devise would have helped.

Let's see now how the revenue distribution behaves in order to know if the revenue is skewed or not. If so, we will adjust it with the logarithm.



We see that the revenue is highly skewed to the left. Let's see how the log distribution of the revenue behaves:



The log distribution reduced significantly the skewness of the revenue, hence we keep the log revenue instead.

4.2 Movie Collection

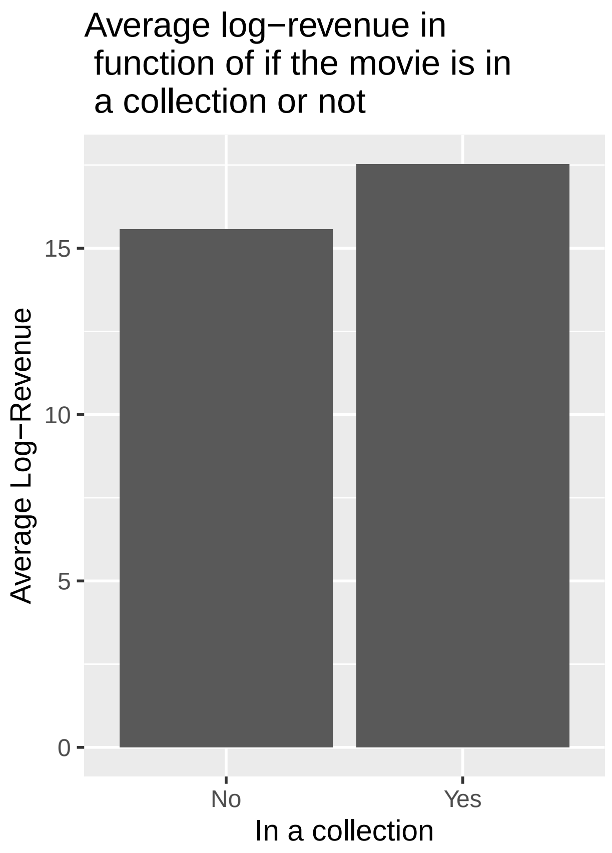
The objective is to know if the income of a movie belonging in a collection is higher than a movie that is not in a collection. A movie that belongs in a collection could mean that:

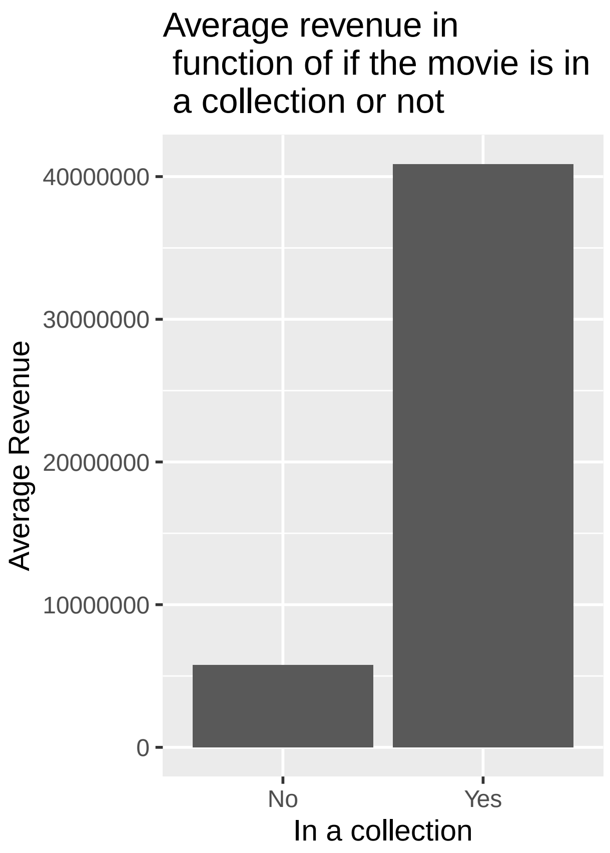
- the movie was enough good to gain a large revenue that the producer(s) decided to make a second movie and so on as a series.
- the same or another producer decided to do a remake of the movie many years after the original one (e.g. Karate kid, Superman).

For both assumptions, we expect a greater revenue for movies in a collection.

```
train$is_in_collection <- unlist(lapply(train$belongs_to_collection, HasTheFeature))
train$belongs_to_collection <- NULL

test$is_in_collection <- unlist(lapply(test$belongs_to_collection, HasTheFeature))
test$belongs_to_collection <- NULL
```

We build a bar chart to represent the average log-revenue in function of if the movie is in a collection or not.





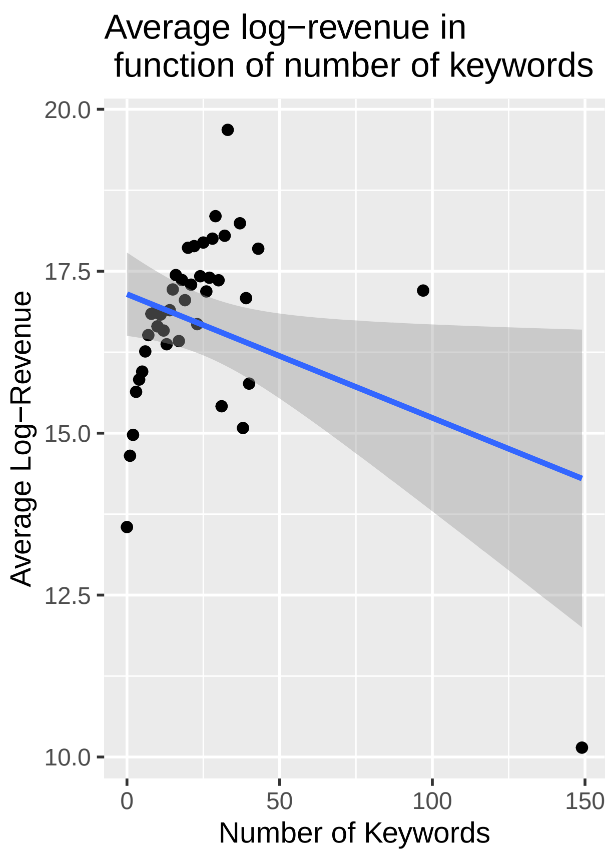
The difference between movies in a collection and the ones not in a collection is small. However, the average revenue is slightly bigger for movies belonging in a collection.

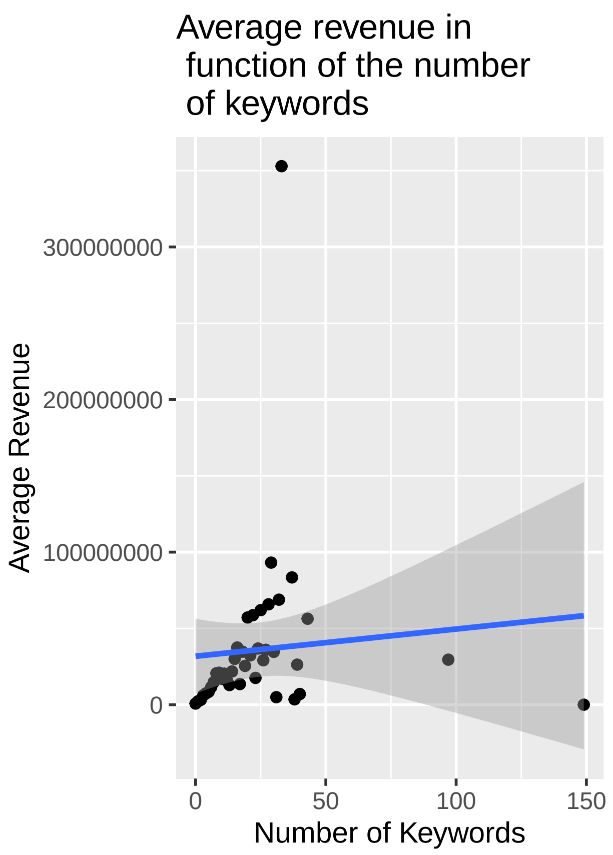
4.3 Movie Keywords

From the feature `keywords`, we only need to know the number of keywords used in order to facilitate the search of the movie. We assume that a movie associated to many keywords could help increasing its revenue because it is easier to search and find. However, it may also depend on the language the keywords are because in a foreign language it could be harder to know the spelling of the keyword and then to search with it. It depends also on the precision of the keywords. For example, the movie `Casino Royale` keywords `James Bond`, `007`, `Digit` and `casino` are more precise than `bank`, `money` and `terrorist`.

```
train$number_of_keywords <- unlist(lapply(train$keywords, CountJSONArrayInFeature))
train$keywords <- NULL

test$number_of_keywords <- unlist(lapply(test$keywords, CountJSONArrayInFeature))
test$keywords <- NULL
```

We build a scatter plot to represent the average log-revenue in function of the number of keywords.





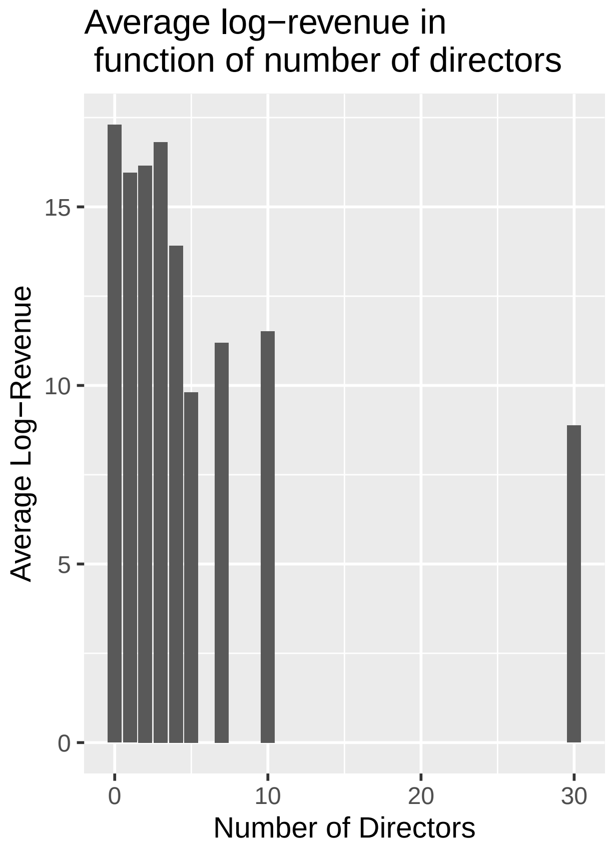
Having keywords helps on the movie revenue but having no keywords or too many of them have a negative impact. According to the scatter plot, movies with big revenues have around 20 - 25 keywords.

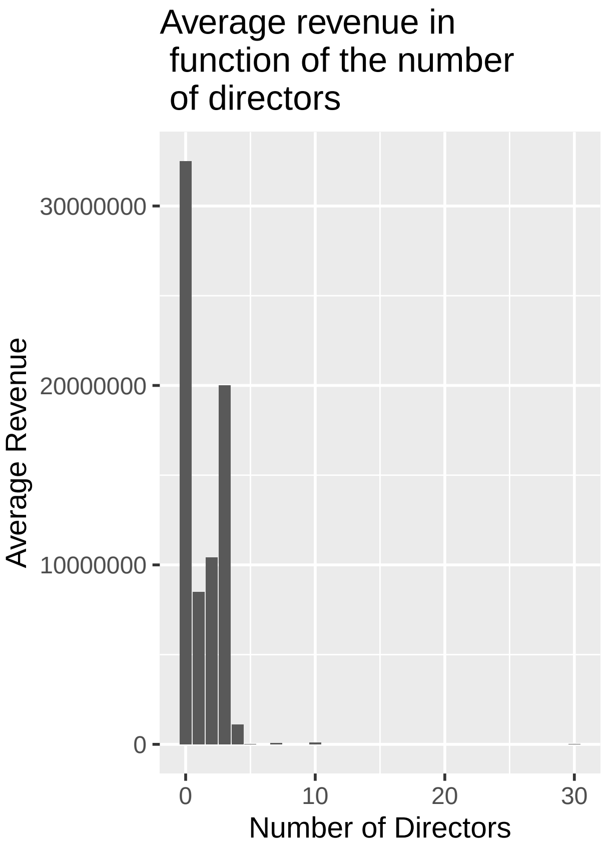
4.4 Members of the Crew

From the feature `crew`, we want to know if having a director as a member of the crew has significant impacts on the movie revenue. The director has the role to choose the cast and crew members (make generally the decisions), they have to be creative in order to ensure the movie is realized within the budget. However, the director depends on the budget to make the movie and also on his competences.

```
train$number_of_directors <- unlist(lapply(train$crew, CountMembersInCrewByJobType, job.type = "Directo
test$number_of_directors <- unlist(lapply(test$crew, CountMembersInCrewByJobType, job.type = "Director"
```

We build a bar chart in order to represent the average log-revenue in function of the number of directors.





```
# A tibble: 9 x 3
  number_of_directors mean_revenue number_of_movies
                <dbl>        <dbl>            <int>
1                   0        17.3                16
2                   1        16.0              2815
3                   2        16.2               146
4                   3        16.8                14
5                   4        13.9                 3
6                   5         9.81                2
7                   7        11.2                 2
8                  10        11.5                 1
9                  30         8.88                1
```

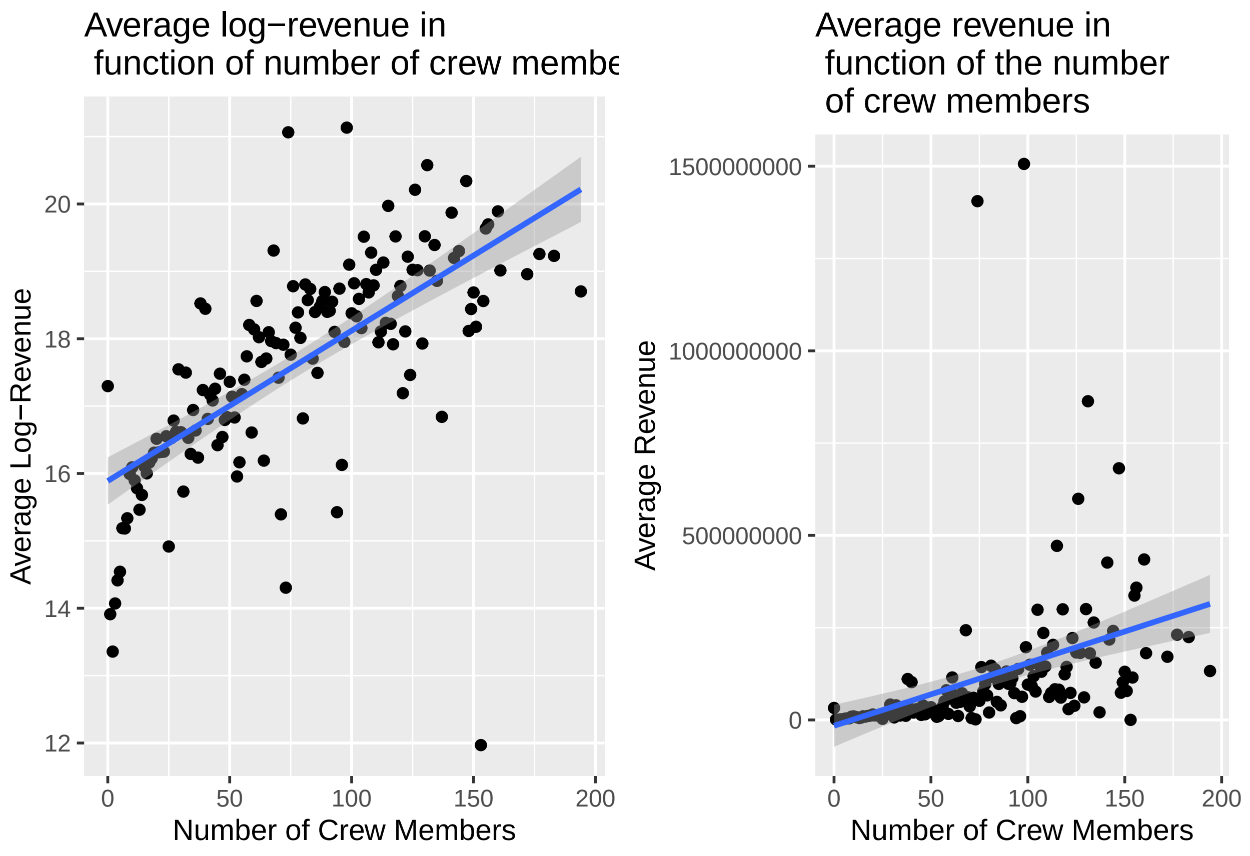
According to this bar chart, the difference between crew having at least a director and crew having no directors is not significant except for 5 and more directors. However, without knowing the number of movies per number of directors, we cannot know the reasons behind this.

We see that only 16 movies with crews having no directors have been made compared to the others having at least a director. Is this an enough large sample to conclude that the revenue will be much better if the crew has no director, since it represents only 0.5333333% of the movies in the train dataset? On a mathematical point of view, the sample is not enough large to conclude but on a data driven point of view, those movies show that having or not directors in the crew does not matter. Furthermore, most of movies (93.8333333% of them) have been made with one director in the crew.

We also expect that a director will not be alone in the crew members. Some of the members have certainly written the movie script in order to be produced. The producer has the role to select the script and ensure the schedule and budget are respected. In order to improve the sequences of a movie and make the movie in its finished state, an editor is important. In summary, we assume that the number of members in the crew has an impact on the revenue. We expect that a small crew will make low revenue movies whereas a large crew will make bigger revenue movies.

```
directors <- NULL
train$number_of_crew_members <- unlist(lapply(train$crew, CountJSONArrayInFeature))
train$crew <- NULL

test$number_of_crew_members <- unlist(lapply(test$crew, CountJSONArrayInFeature))
test$crew <- NULL
```

We build a bar chart in order to represent the average revenue in function of the number of crew members.

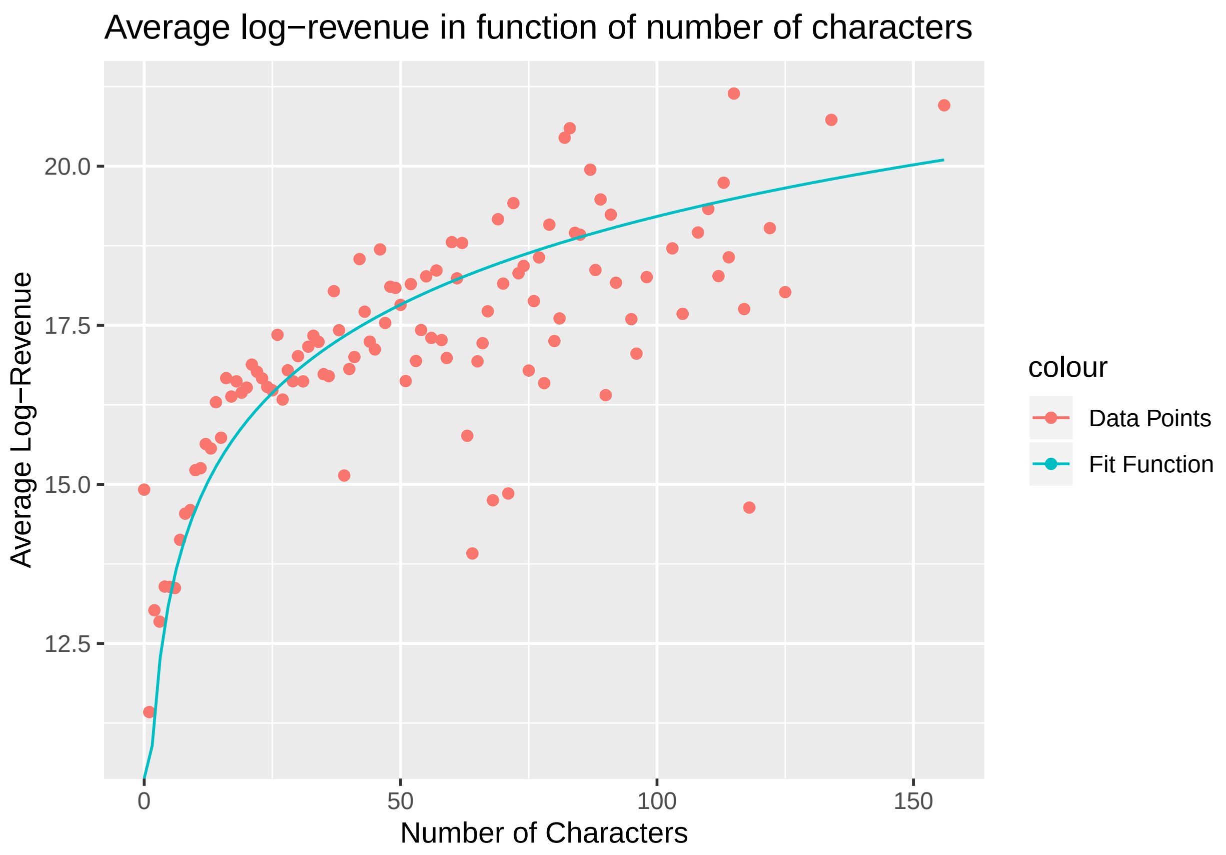


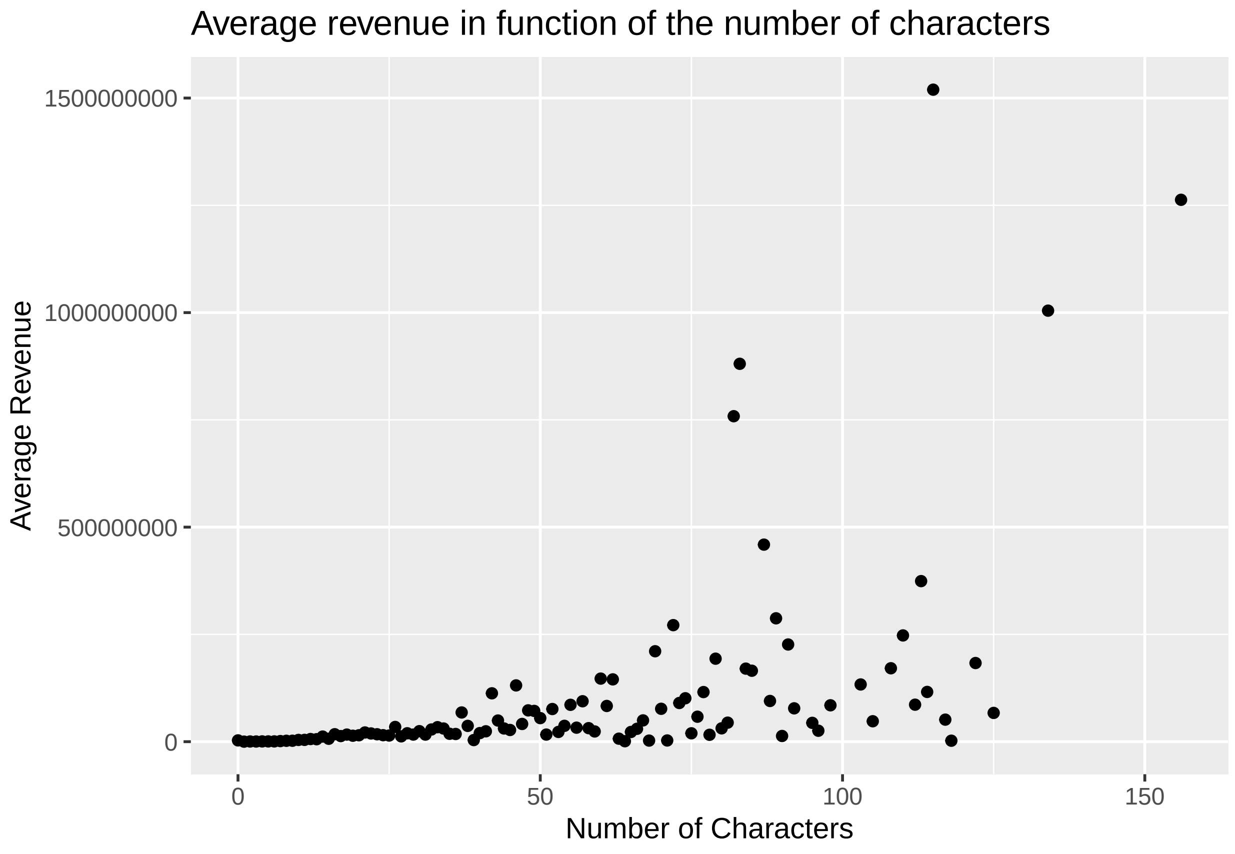
4.5 Movie Casting

For the feature `cast`, we know that more characters are playing in a movie, more large must be the budget. This is also based on the popularity of the actors and how much they ask to play in the movie. We assume that most of the time, the revenue is increasing as the number of actors increases.

```
train$number_of_characters <- unlist(lapply(train$cast, CountJSONArrayInFeature))
train$cast <- NULL

test$number_of_characters <- unlist(lapply(test$cast, CountJSONArrayInFeature))
test$cast <- NULL
```

We build a scatter plot to represent the average log-revenue in function of the number of characters playing in the movie.





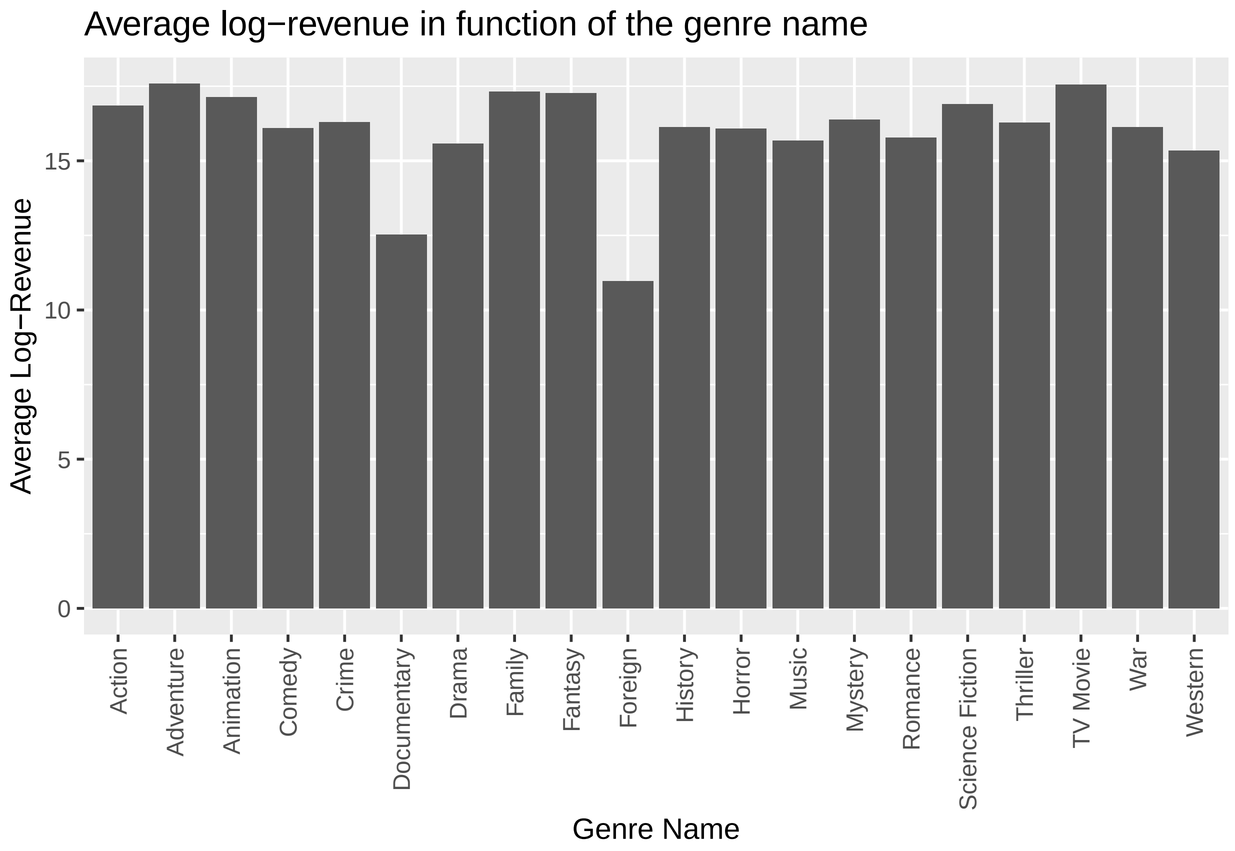
We have to consider the budget allowed because it is possible that many actors play in a low budget movie.

These actors are paid with a lower salary or they are inexperienced which could justify their low salary. Another reason could be that many of these actors are playing a very short amount of time in the movie. This may explain why sometimes with many characters the revenue is still low. However, the curve $f(x) = 2\ln(x) + 10$ fits well with the data points and could be used to model them. However, it does not mean that this model will fit well with the test data points.

4.6 Movie Genres

Nowadays, superheros movies (e.g. Avengers, Superman, Iron man, etc.), action/adventure and animation movies are really popular and their incomes are generally big. These movies are classified as science-fiction, action, animation, adventure and fantasy. We assume that those genres of movies generate more revenues than the others. The objective is to visualize the average log-revenue for every genre. Here are the steps to achieve this objective:

1. Extract in a data frame the genre names for every movie in the dataset with the log-revenue.
2. Append this data frame to the one containing all movies.
3. Group the data frame by genre name where we keep the average log-revenue.
4. Show a bar chart of the average log-revenue in function of the genre name.
5. Keep the genre name with the best average log-revenue if a movie is associated with many genre names.



According to this bar chart, it shows that the following genres generate the lowest average log-revenues with a significant difference compared to the others.

```
# A tibble: 20 x 3
   name            mean_revenue    id
   <chr>                  <dbl> <int>
 1 Adventure               17.6     2
 2 TV Movie                17.6    18
 3 Family                  17.3     8
 4 Fantasy                 17.3     9
 5 Animation               17.1     3
 6 Science Fiction         16.9    16
 7 Action                  16.9     1
 8 Mystery                 16.4    14
 9 Crime                   16.3     5
10 Thriller                16.3    17
11 War                     16.1    19
12 History                 16.1    11
13 Comedy                  16.1     4
14 Horror                  16.1    12
15 Romance                 15.8    15
16 Music                   15.7    13
17 Drama                   15.6     7
18 Western                 15.3    20
19 Documentary             12.5     6
20 Foreign                 11.0    10
```

The last step is to extract the best average log-revenue per genre and keep it in our dataset.

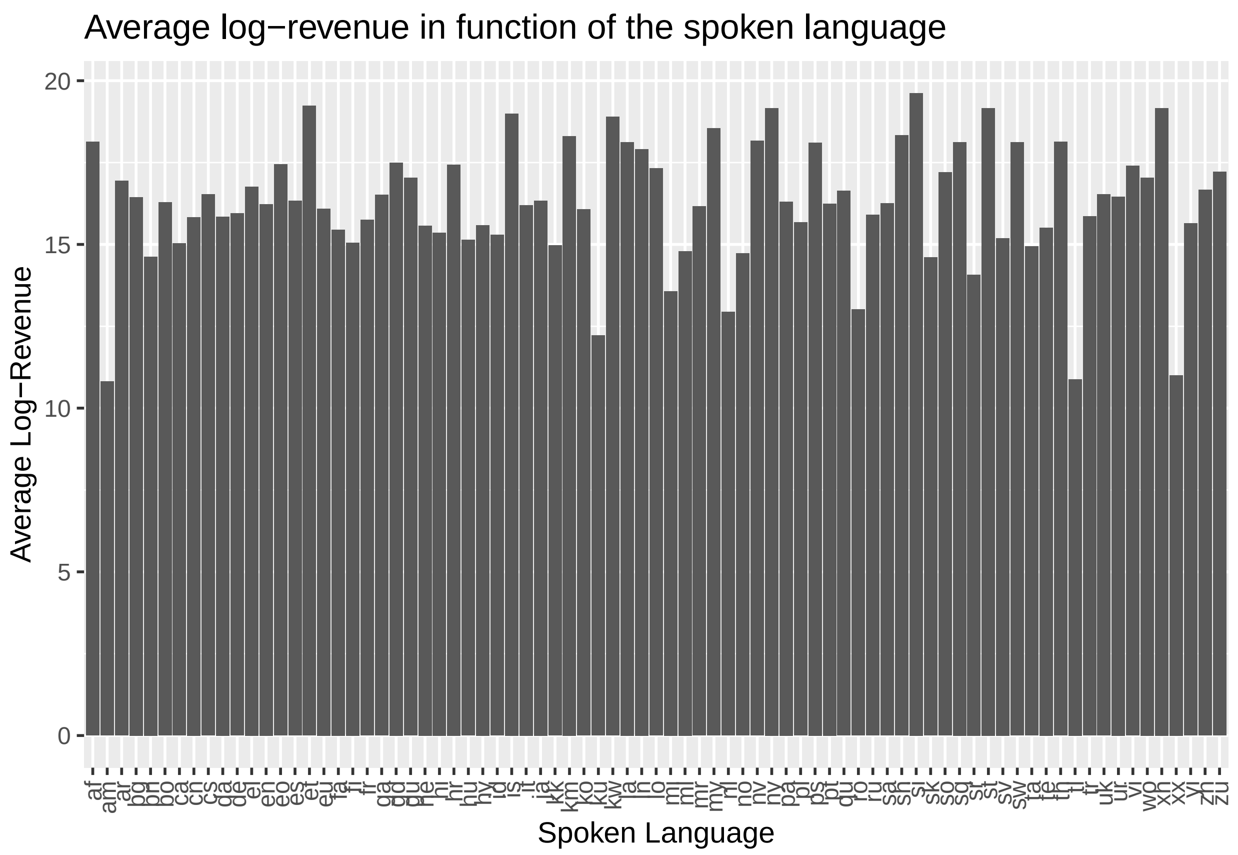
```
## Step 5: Keep the genre with the best revenue average if a movie is associated with many genres.
train$best_genre <- ExtractBestRevenueFromGenres(train, movies.genres)
train$genres <- NULL

test$best_genre <- ExtractBestRevenueFromGenres(test, movies.genres)
movies.genres <- NULL
test$genres <- NULL
```

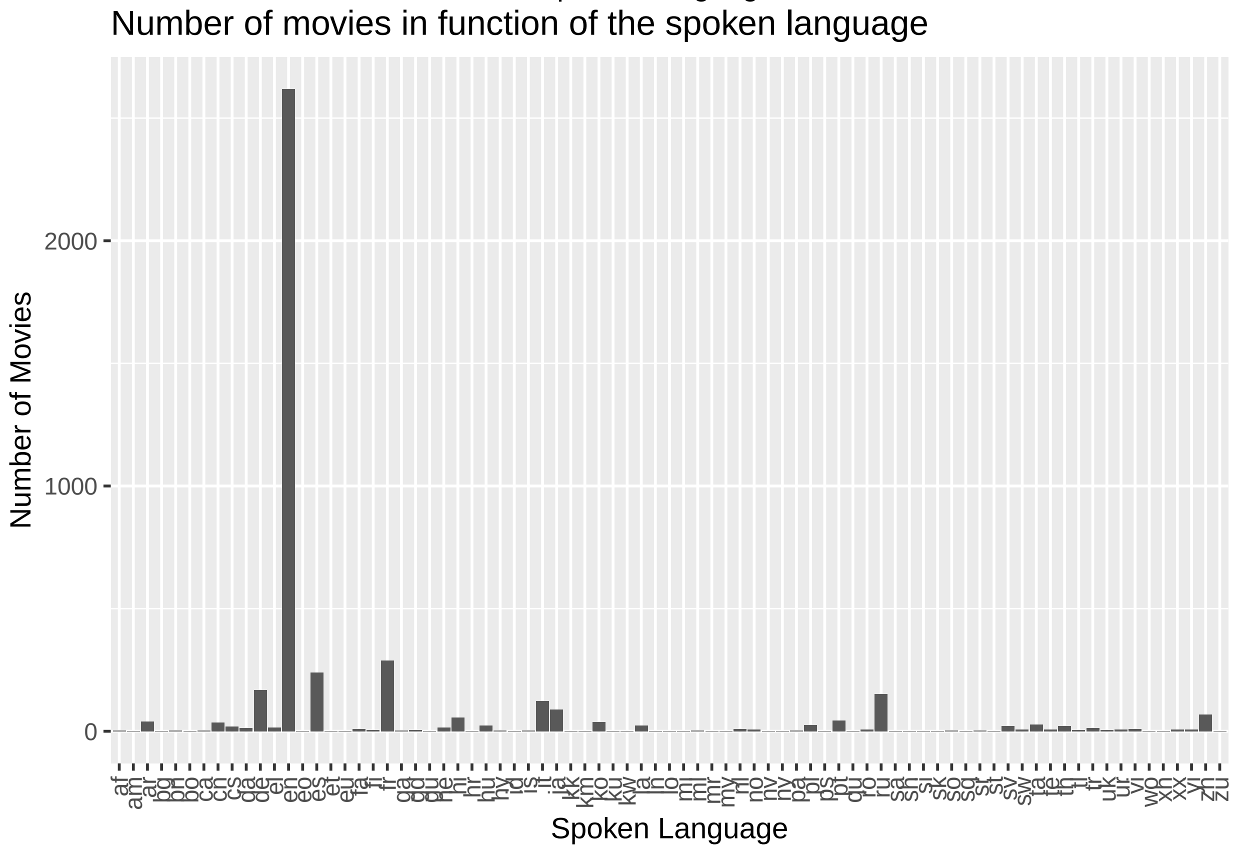
4.7 Spoken Languages

One can assume that movies in which actors speaks in English have the best revenue because it is a well-known language around the world. However, if a spoken language (e.g. Sinhalese) is used in only one movie and another spoken language is used in this movie (e.g. English) where the revenue is huge, the average revenue associated to the Sinhalese spoken language will be huge. In such case, the results could be biaised and mislead to predict the revenue.

The objective is to visualize the average log-revenue for every spoken language. Here are the steps to achieve the objective:

1. Extract in a data frame the spoken language IDs `iso_639_1` for every movie in the dataset with the log-revenue.
2. Append this data frame to the one containing all movies.
3. Group the data frame by spoken language where we keep the average log-revenue.
4. Show a bar chart of the average log-revenue in function of the spoken language.
5. Show a bar chart of the number of movies in function of the spoken language.





```
Percentage of the number of movies spoken in English: 87.27 %
```

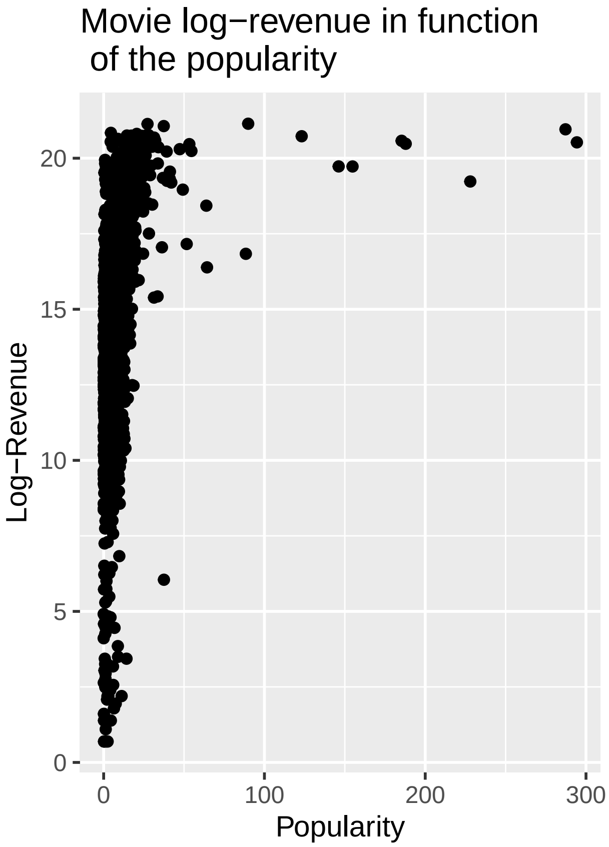
We will not consider this feature to help calculating the predictions of the revenue.

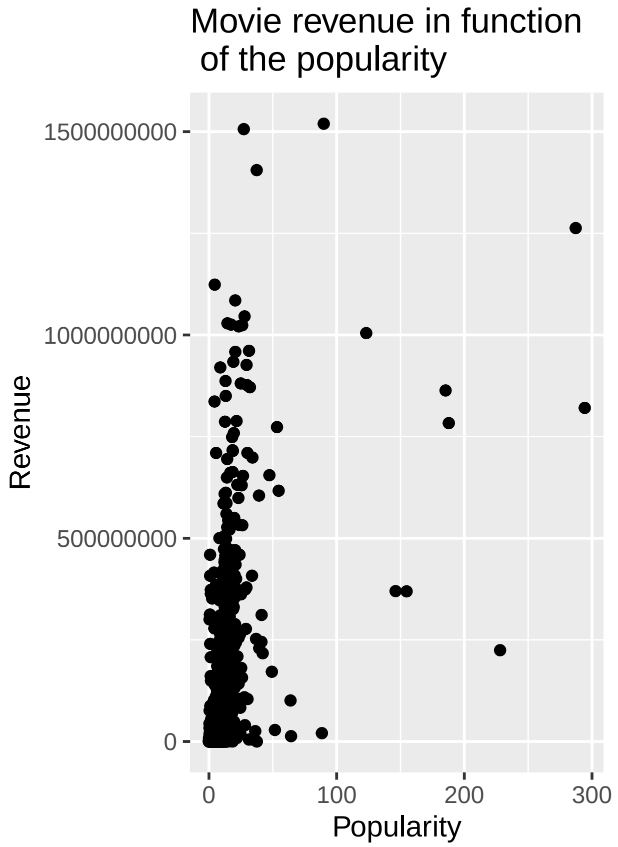
4.8 Movie Popularity

The popularity is a value that gets updated daily and takes a number of things into account like views, number of user ratings/watchlist/favourite additions and release date (Source: https://www.themoviedb.org/talk/56e614a2c3a3685aa4008121).

One can assume that more a movie is popular, more its revenue is greater. However, a movie could be viewed by a large number of people where most of them do not like the movie enough to pay to see it. People could add the movie in the watchlist and never watch it. If we go further, a movie could be very popular because of its great trailer or ads. What if the trailer contains the best parts of the movie just to attract as many people as possible? When people will watch it, they will be dissapointed because they were expecting much more of the movie. In conclusion, we assume that the popularity could be misleading on the prediction of the revenue of a movie.

We build a scatter plot to represent the log-revenue in function of the popularity.





According to this scatter plot, we see some movies where their popularity is greater than 100 with a revenue less than the revenue of some movies having their popularity less than 50. Let's see a top 10 of movies with the greatest log-revenue and their popularity.

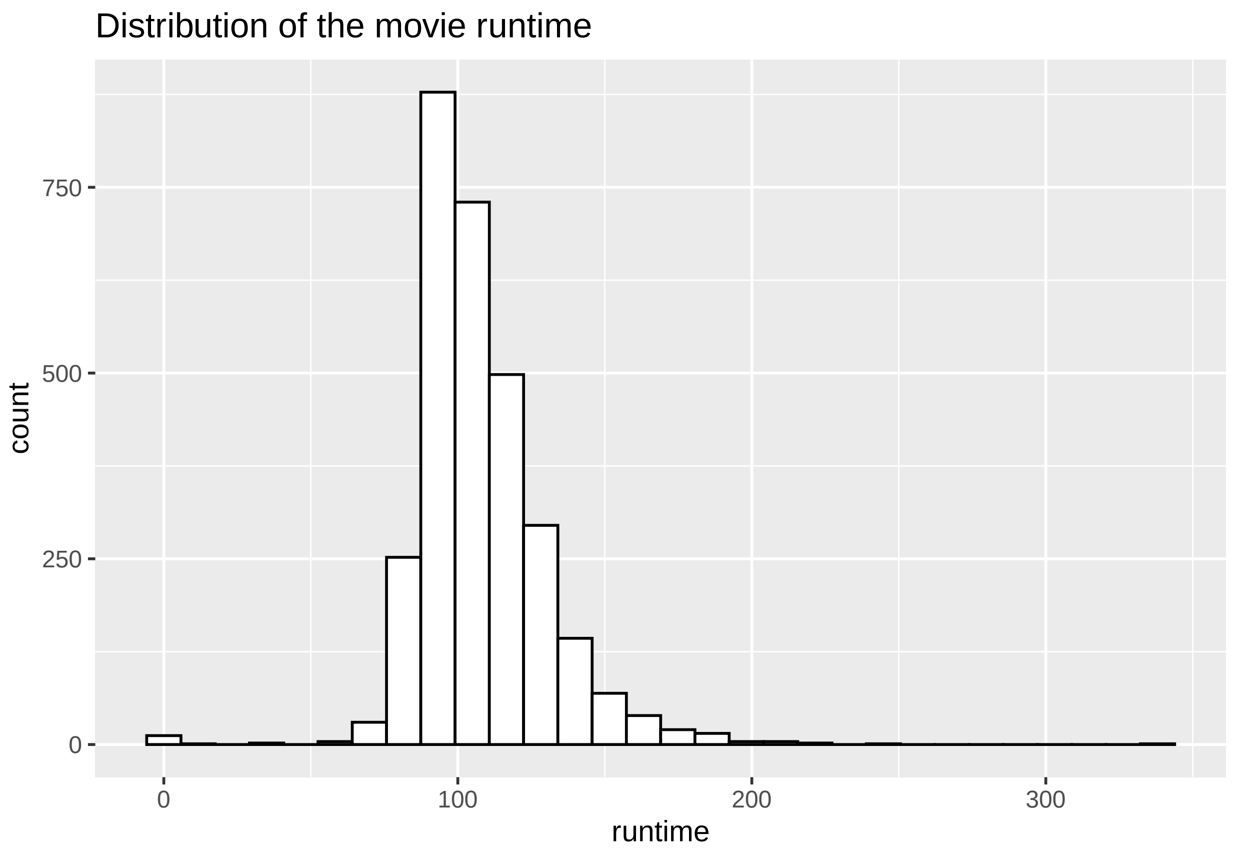
```
                                           title popularity  revenue
1127                                The Avengers  89.887648 21.14169
1762                                   Furious 7  27.275687 21.13289
2771                     Avengers: Age of Ultron  37.379420 21.06359
685                         Beauty and the Beast 287.253654 20.95667
2323               Transformers: Dark of the Moon   4.503505 20.83993
907                        The Dark Knight Rises  20.582580 20.80479
```

```
2136 Pirates of the Caribbean: On Stranger Tides  27.887720 20.76797
2563                               Finding Dory  14.477677 20.75144
882                        Alice in Wonderland  17.285093 20.74844
735                                   Zootopia  26.024868 20.74677
```

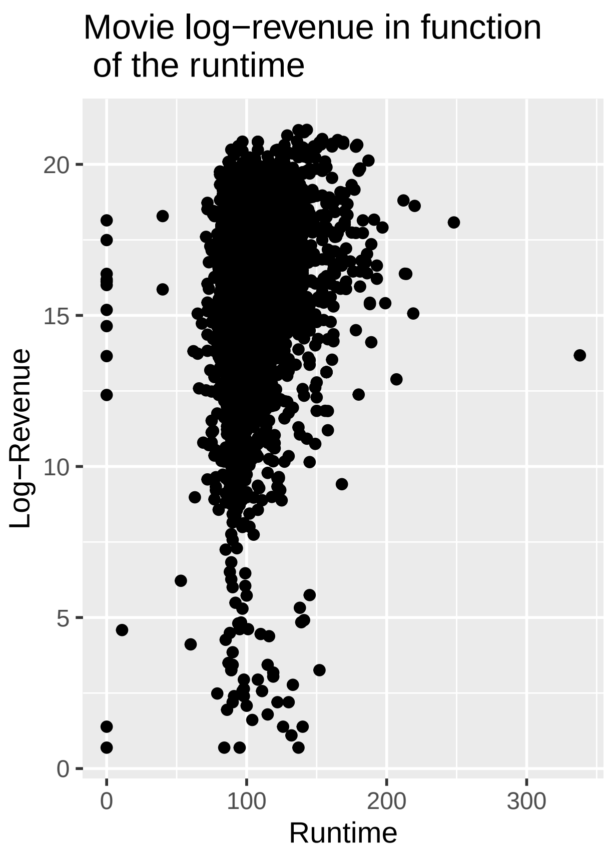
As our assumption stated, the popularity could be misleading on the prediction of the revenue. A good example of this is the movie *Beauty and the Beast* with a revenue of 20.9566657 (ranked 4th among the 3000 movies) and a great popularity of 287.253654 whereas the movie *Transformers: Dark of the Moon* generated a revenue of 20.8399345 (ranked 5th among the 3000 movies) and has a low popularity of 4.503505.

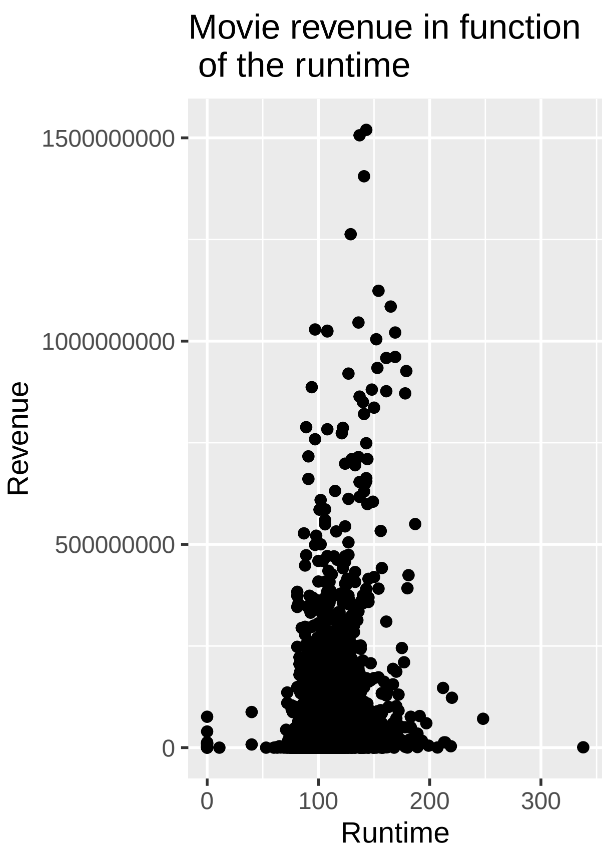
4.9 Movie Runtime

Generally, a movie should not be too long or too short. A too long movie may become boring or make people watching the movie in 2 parts or more because it takes too much time in a day to watch the full movie. For too short movies, the story could be shorten where there is no conclusion, skip some part of the story or not long enough because the movie is so good. For both cases, we expect that the imapcts on the revenue is negative. Before showing the revenue in function of the runtime, we have to verify if the runtime is skewed or not.



Let's see now if our assumptions hold..

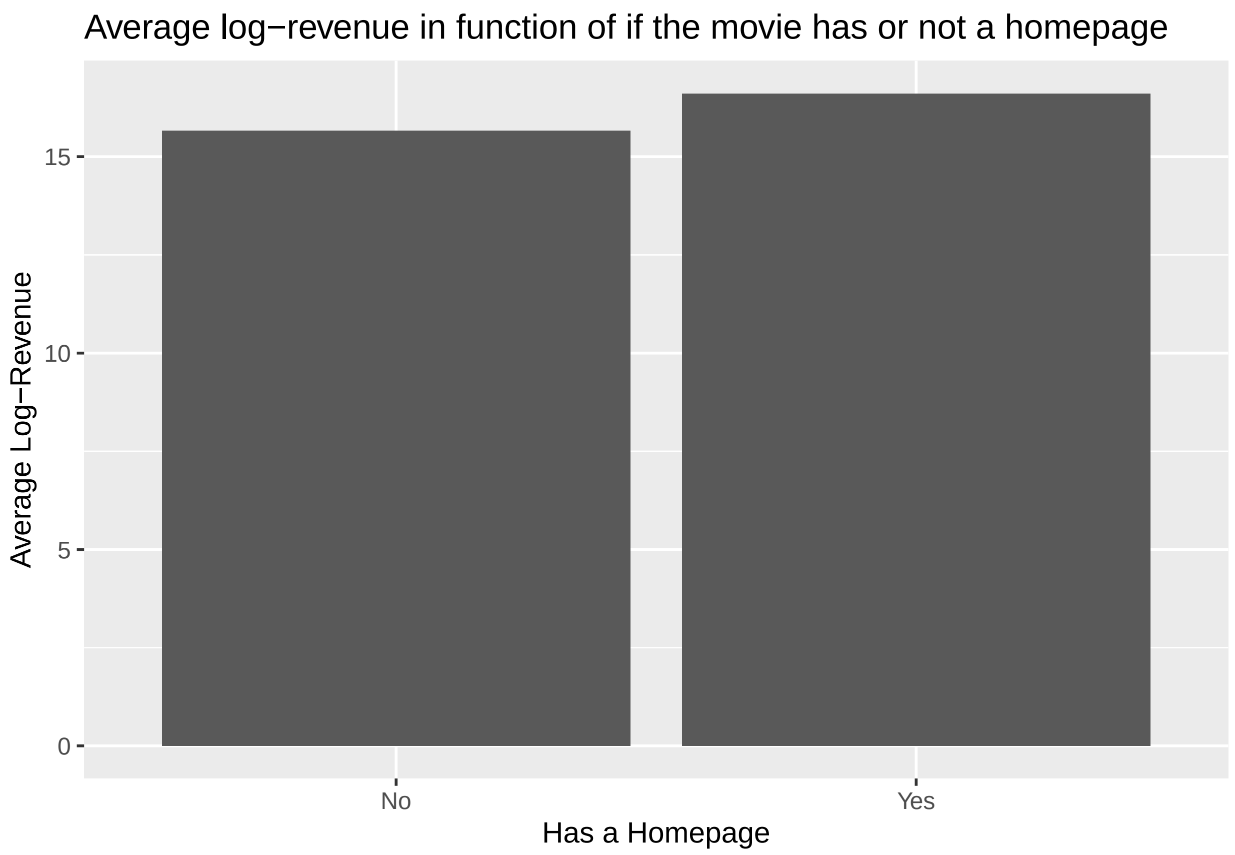




According to this scatter plot, runtimes between 90 minutes and 180 minutes obtain better revenues. Out of that range, the revenue is lower.

4.10 Movie Homepage

Homepage for movies makes them more visible and accessible to people because they can get more details (news, new movies coming up, critics, trailers) on movies and get an overview of the list of movies (e.g. Marvel's movies). This is a good way to attract people and make them watch the movies. For example, if someone wants to see what Disney's movies are coming up soon, he search for Disney's movies and will quickly find the Disney's website.

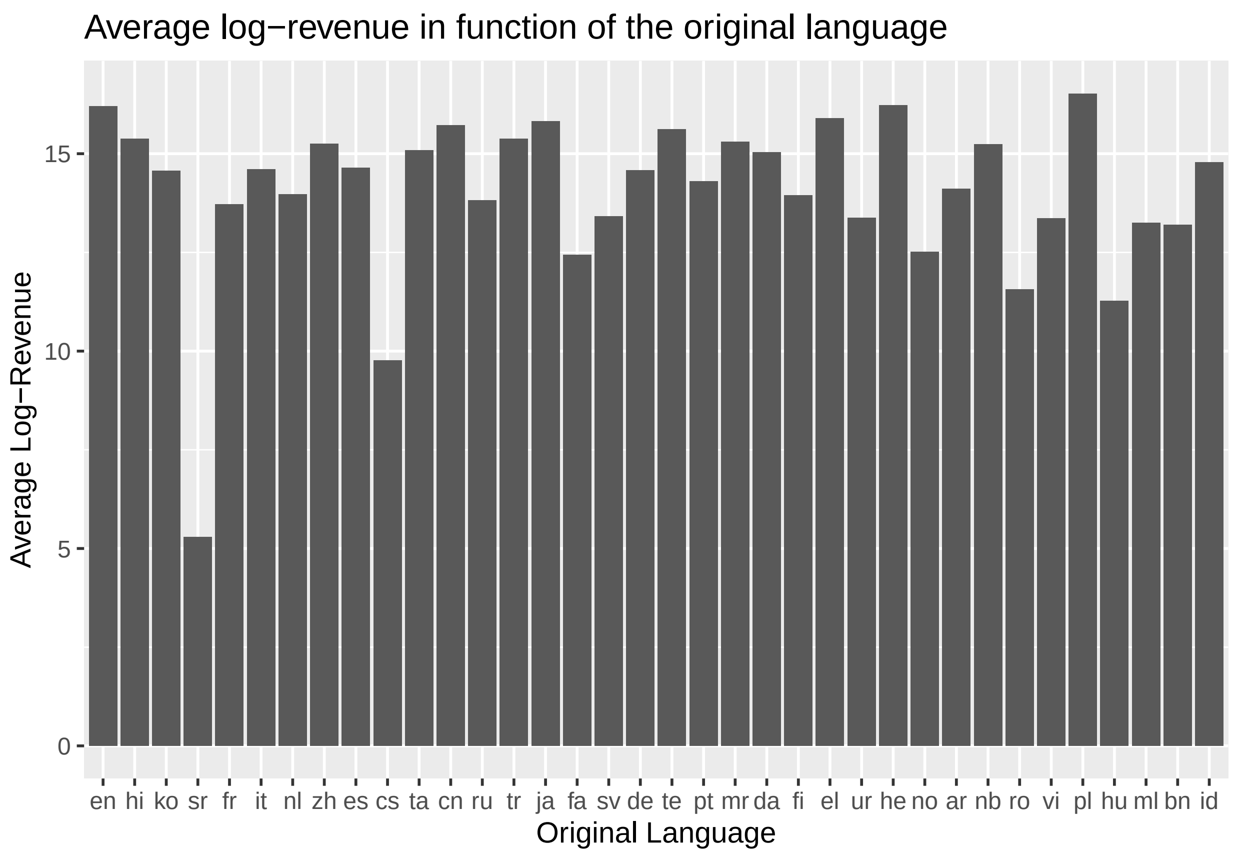


According to this bar chart, the log-revenue is slightly bigger for movies having a homepage than the ones that do not. This feature should not be vrey helpful on the model prediction.

4.11 Movie Original Language

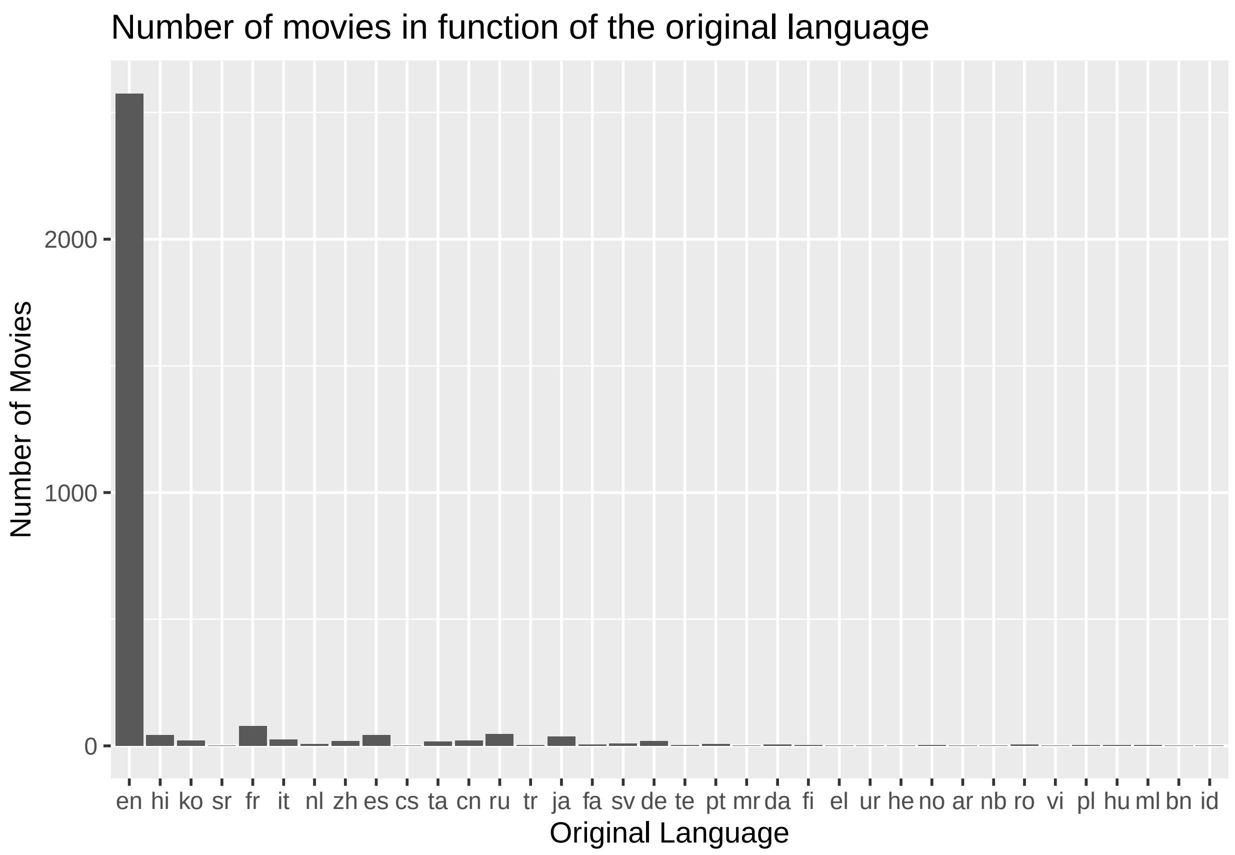
We know that some languages are more spoken around the world than others. One can assume that English language spoken in movies is the most popular and has the best revenue average because it is a well-know language around the world. Movies where the actors speaks foreign languages that are spoken in only a country or in a part of a country will need to be translated and may not be known. We expect that this has a negative impact on the revenue.

However, we saw many good Chinese movies (specially with martial arts) with great Chinese actors (e.g. Jackie Chan, Donnie Yen, Jet Li) that got known around the world. Knowing that, we assume that Chinese movies have generated among the best revenue.



```
# A tibble: 5 x 3
  original_language_id mean_revenue number_of_movies
                 <int>        <dbl>            <int>
1                    1         16.2             2575
2                    2         15.4               42
3                    9         14.6               43
4                   13         13.8               47
5                    5         13.7               78
```

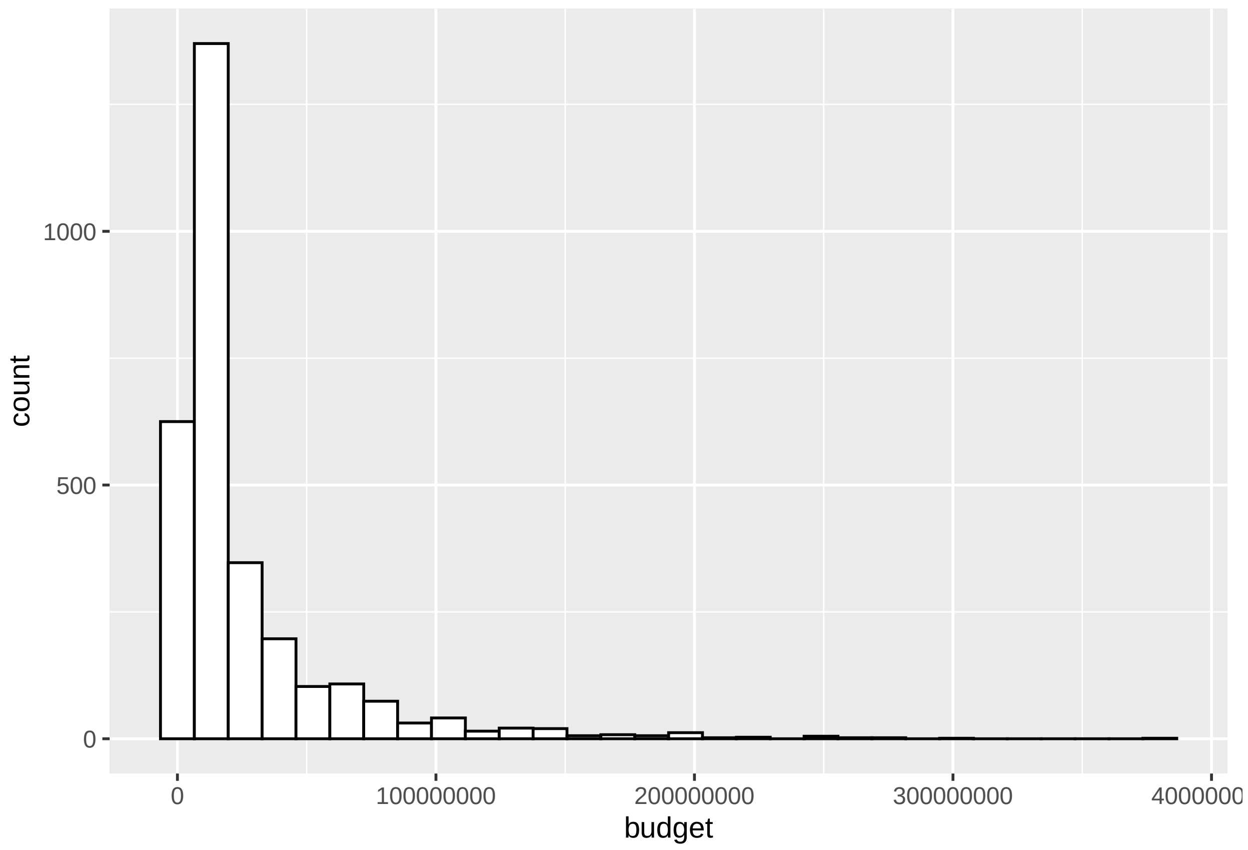
Let's see the number of movies by original language.



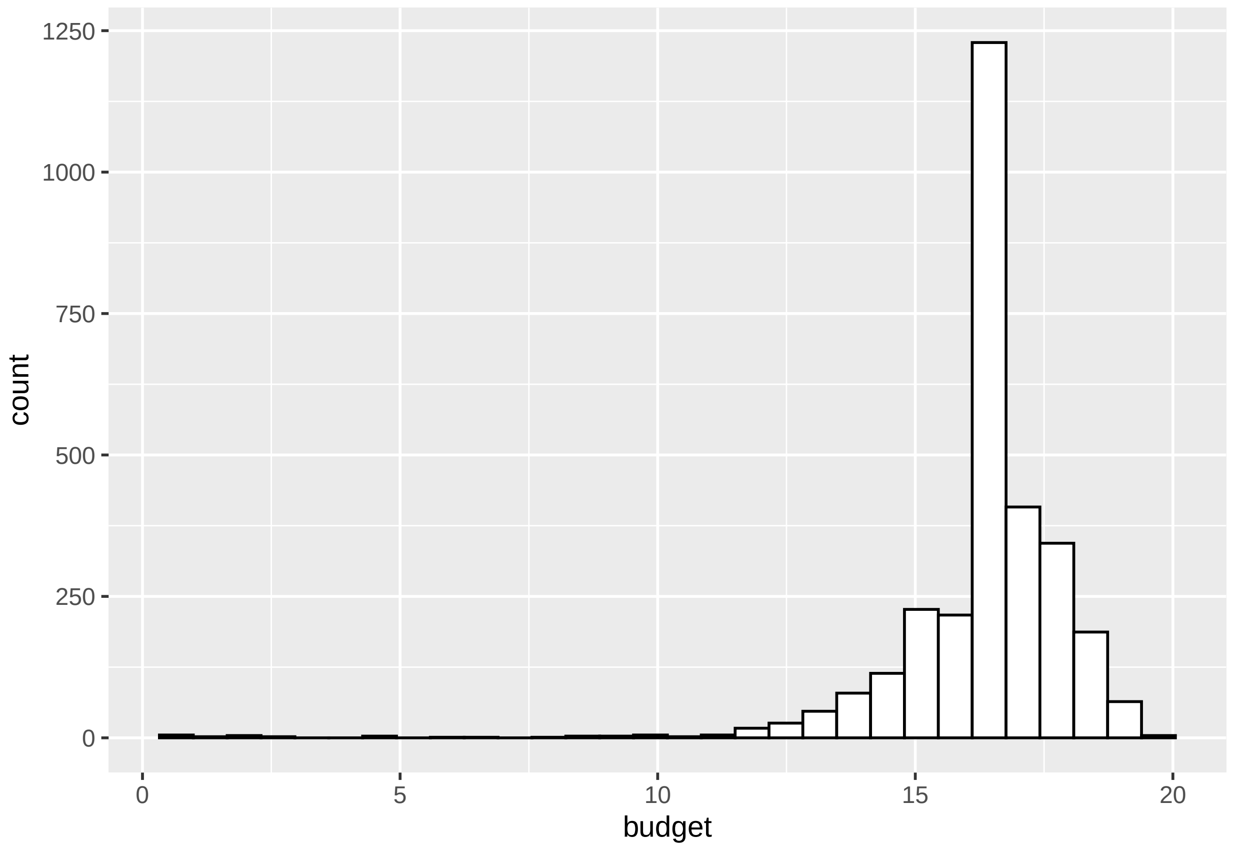
As we expected, English language spoken in movies dominates and have generated the best average log-revenue.

4.12 Movie Budget

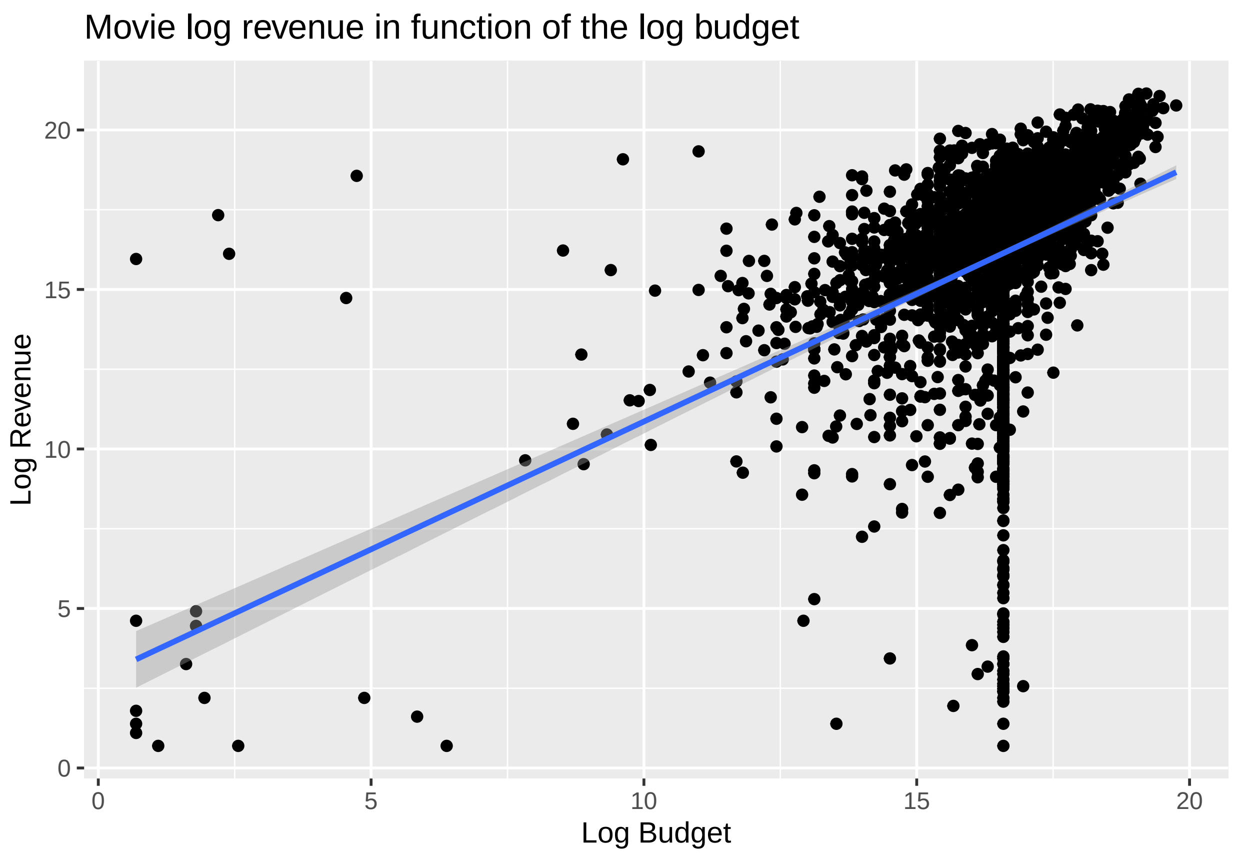
The budget is always an important constraint to consider before making a movie. If the budget allowed is low, we expect that the revenue will be also low but most of the time, it is much greater than the budget. Let's see the distribution of the budget:



The distribution of the budget has a high skewness to the left and need to be adjusted using a log distribution instead.



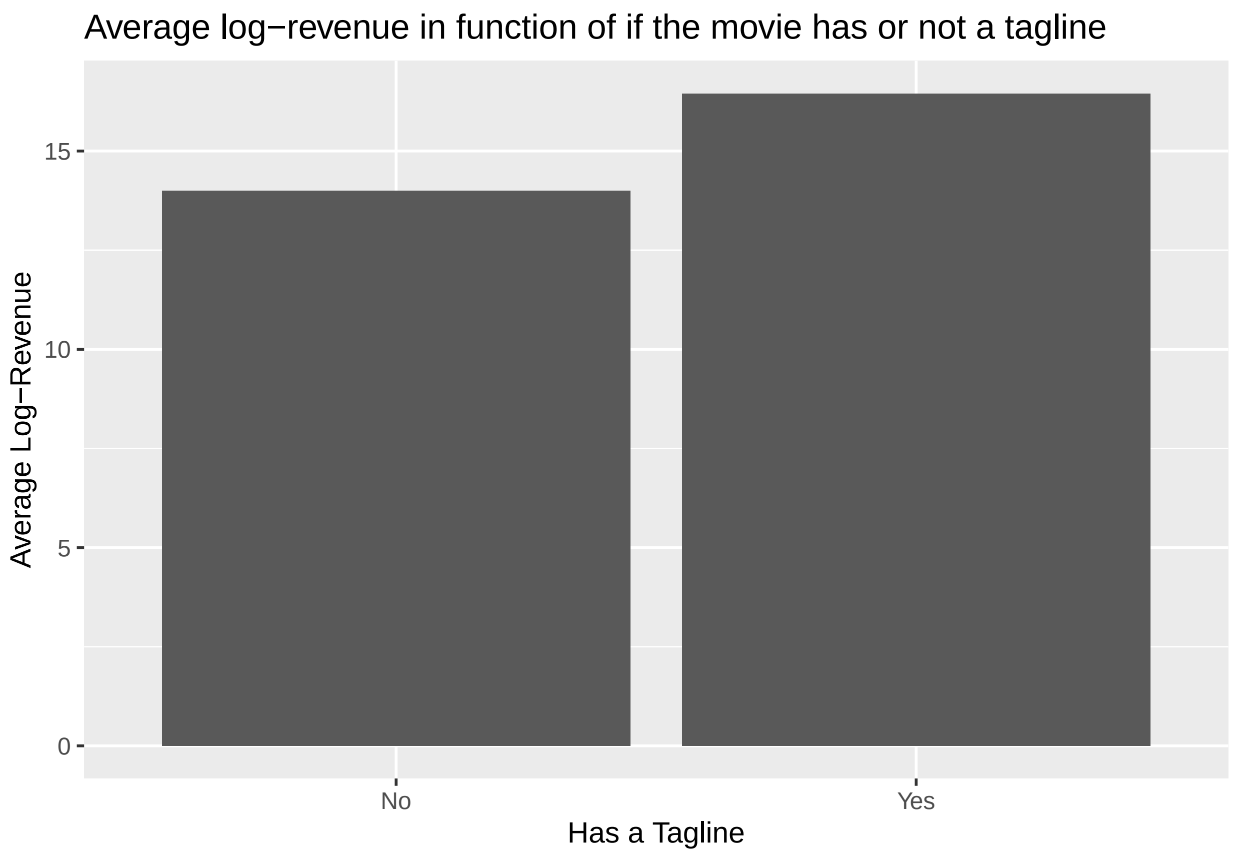
Now that the budget is adjusted to remove the high skewness, let's see how is the log revenue in function of the log budget.



Our assumptions holds most of the time and the scatter plot describes a linear increasing of the revenue in function of the budget. Therefore, the budget matters and has a significant impact on the revenues.

4.13 Movie Tagline

A movie having a tagline can help the people remembering that movie because they are catchy and help selling the movie on a marketing point of view. Thus, we assume that movies having no tagline have their average revenue lower than the ones having a tagline.



According to this bar chart, our assumption holds. Movies having a tagline have a better revenue than the ones having no tagline.

4.14 Release Date

We know that making million or billion dollars was harder as we get back over the years because mainly of the inflation and life cost. Therefore, it makes sense that the revenue was considerably lower in the '80s and older than nowadays. We expect that the revenue will be much lower in the '50s, '60's until the '90s than the 2000s and 2010s.

We assume also that a movie released on the weekend (mainly Friday or Saturday) generates more revenue because people are generally not working the next day. They will see the movie in the evening but we cannot check that assumption because the time is not provided in this dataset.

Another assumption is that releasing a movie on Summer generates more revenue because young students going to high school or primary school can see the movie anytime during the daylight. Indeed, the school normally ends at the middle/end of June and starts on end of August or beginning of September.

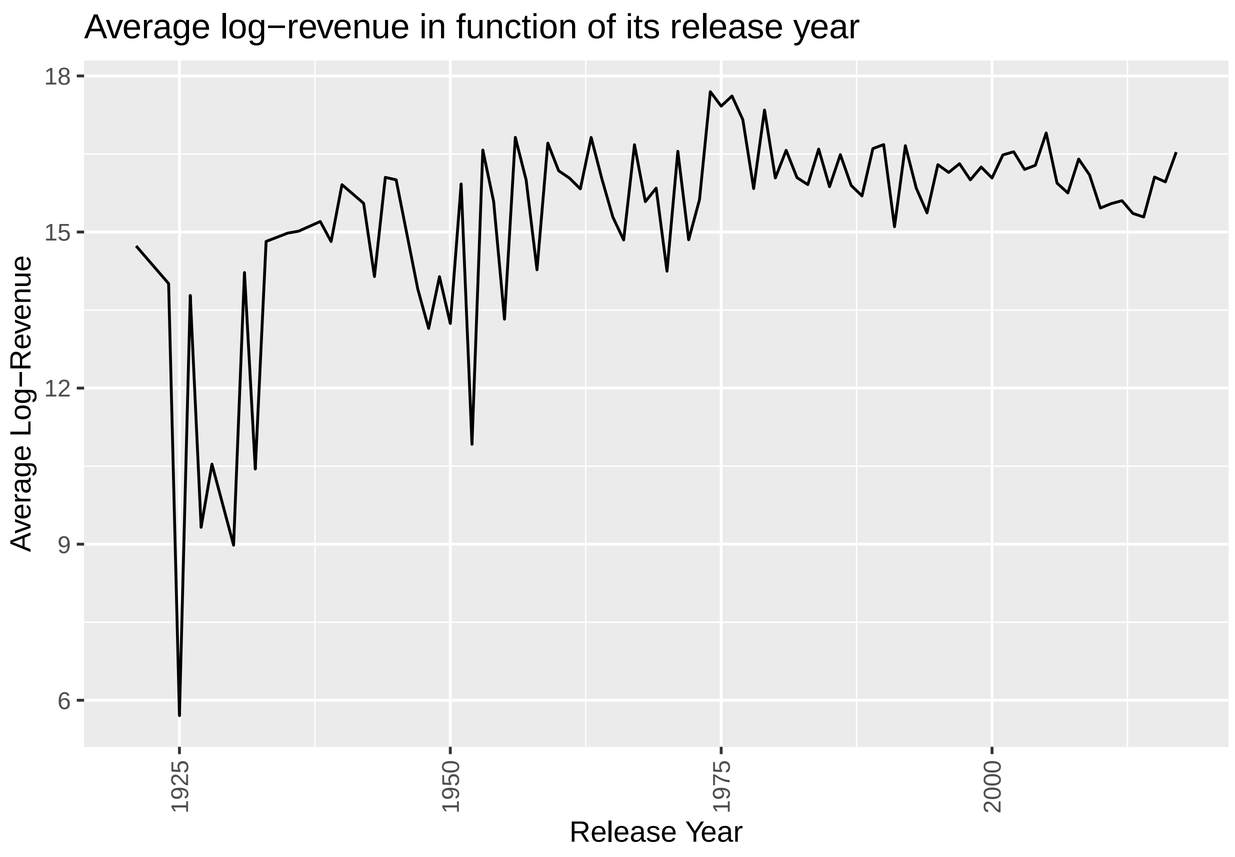
The objective is to determine if the year, month or/and weekday the movie was released have a considerable impact on the revenue. Here are the steps to achieve this objective:

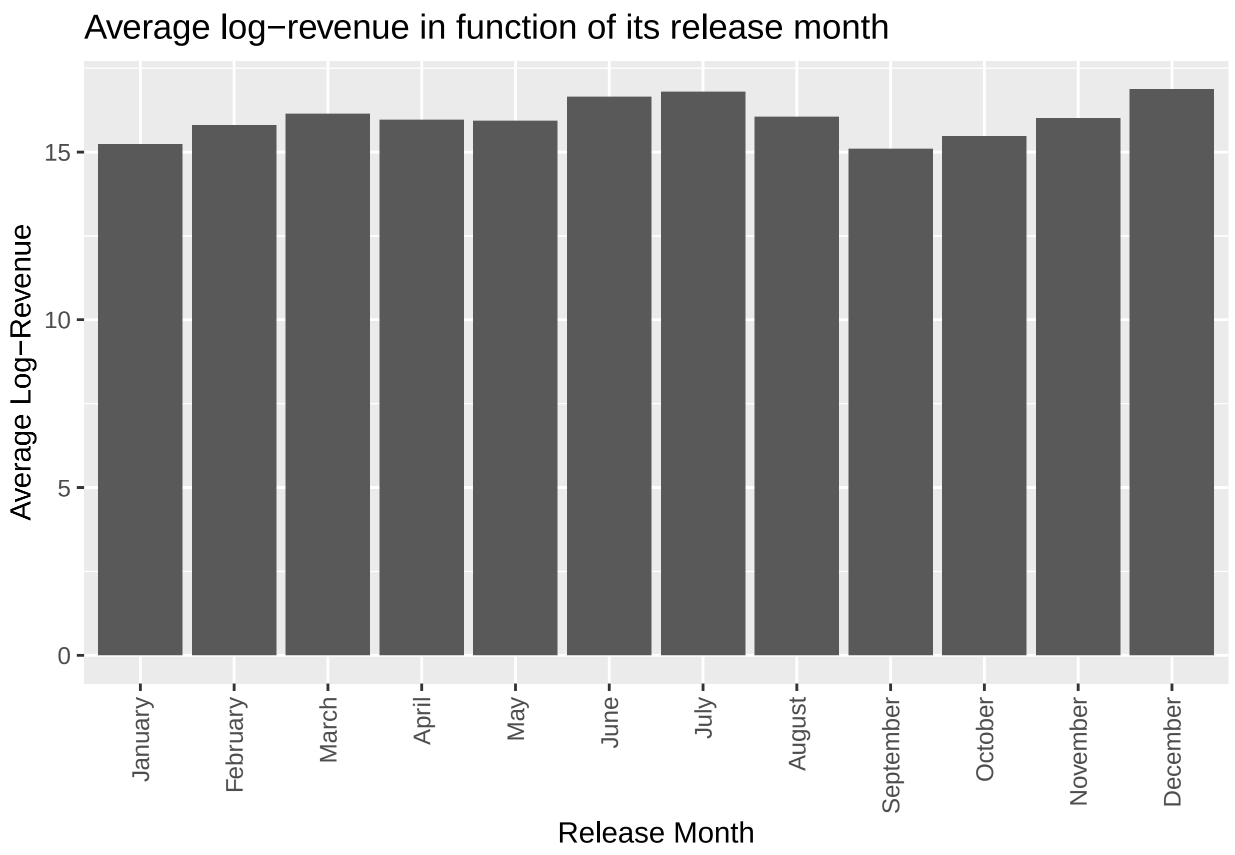
1. Convert the release date from string to date with a more lisible format `YYYY-MM-DD`.
2. Extract the year, month and weekday in a data frame for every movie release date.
3. Show bar charts of the revenue in function of the release year, release month and release week-day.
4. Show line charts of the number of movies in function of the release year, release month and release weekday.

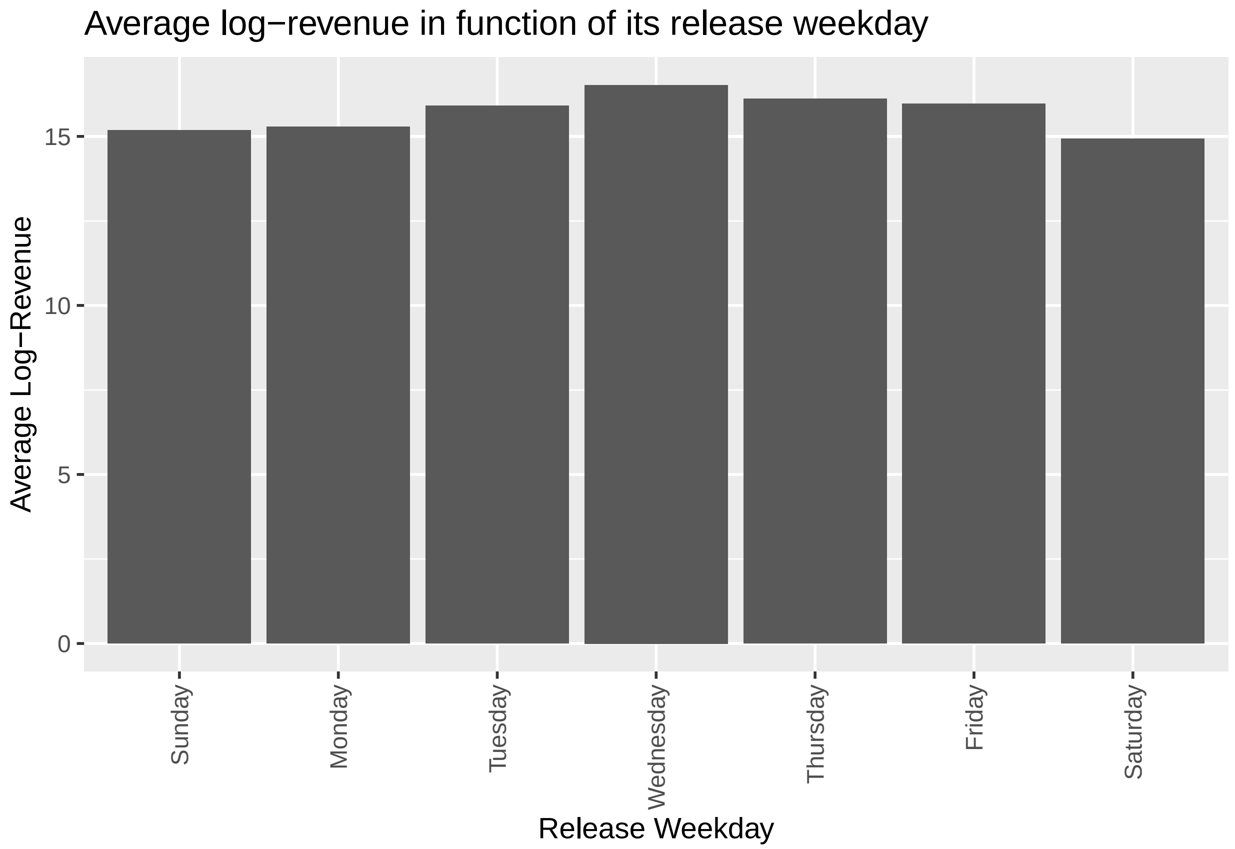
```
## Step 1: Convert the release date from string to date with a more lisible format YYYY-MM-DD.
train$release_date <- as.Date(train$release_date, format = "%m/%d/%y")
```

```r
test$release_date <- as.Date(test$release_date, format = "%m/%d/%y")

## Step 2: Extract the year, month and weekday in a data frame for every movie release date.
library(lubridate)

train$release_year <- year(train$release_date)
train$release_year[train$release_year > 2019] <- train$release_year[train$release_year > 2019] - 100
train$release_month <- month(train$release_date)
train$release_weekday <- wday(train$release_date)
train$release_date <- as.numeric(as.POSIXct(train$release_date, format = "%m/%d/%y", origin = "1900-01-0
#train$release_date <- (year(train$release_date) * 366) + (train$release_month * 31) + day(train$releas

test$release_year <- year(test$release_date)
test$release_year[test$release_year > 2019] <- test$release_year[test$release_year > 2019] - 100
test$release_month <- month(test$release_date)
test$release_weekday <- wday(test$release_date)
test$release_date <- as.numeric(as.POSIXct(test$release_date, format = "%m/%d/%y", origin = "1900-01-01
#test$release_date <- (year(test$release_date) * 366) + (test$release_month * 31) + day(test$release_da
```

Let's see how is the average log-revenue per year, month and weekdays.



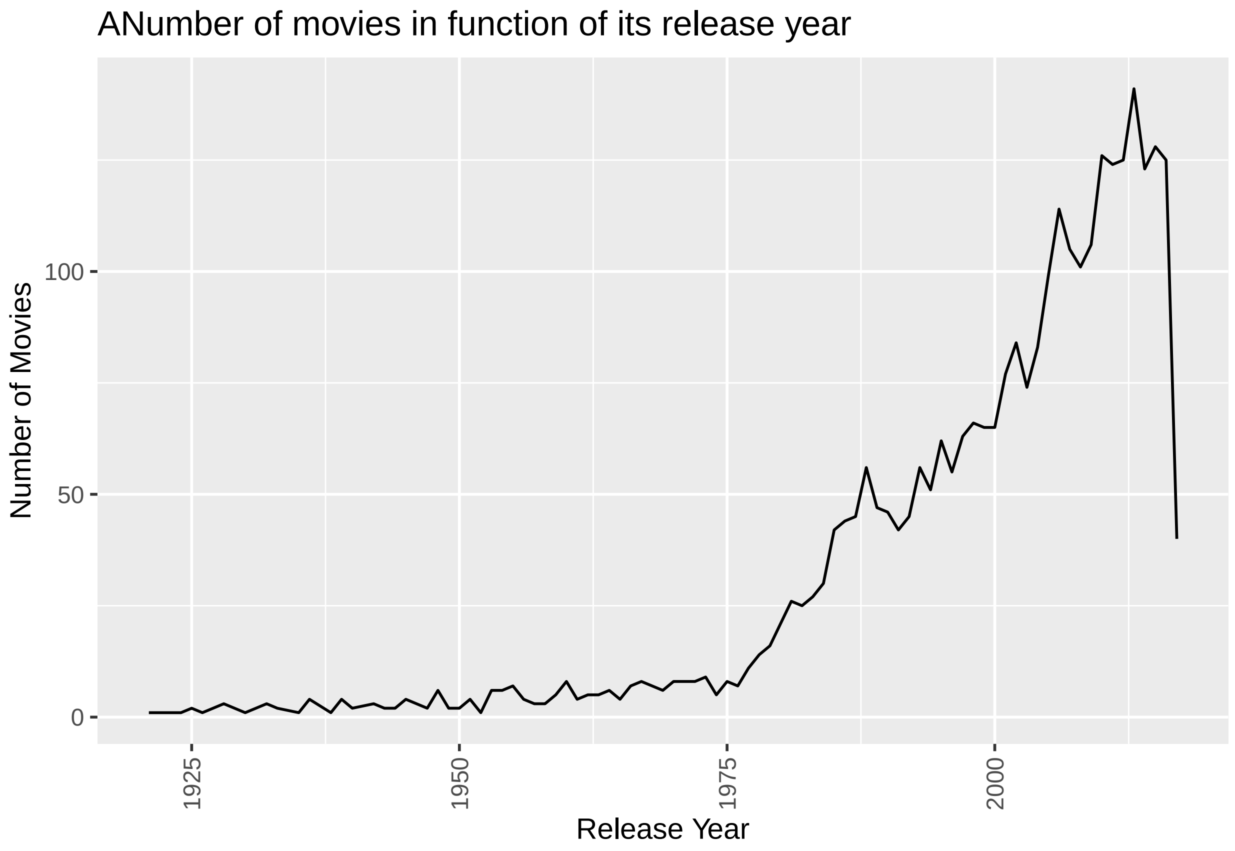




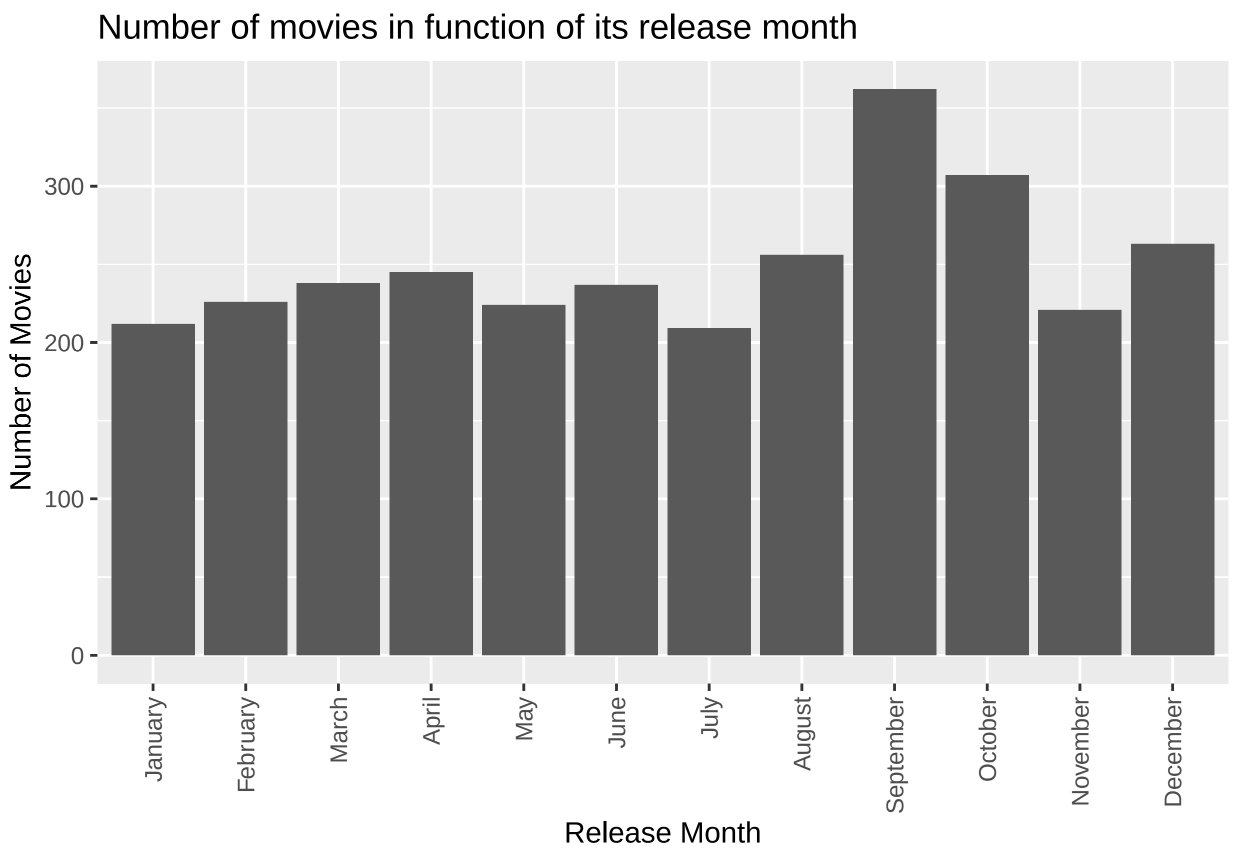
According to these bar charts, we see that:

- Generally, more recent are the movies, better are the revenue. However, we see that around the year 1975, the average log-revenues were as good as today which is suprising.
- Months of May, June and July got better revenue (specially June) which may be caused by the summer time as our assumption states.
- In December, the revenue is also great probably because of Christmas where children are on holiday vacations the last part of the month.
- Movies relased on Wednesday got better revenue followed by Thuesday. This is suprising because it is in the middle of the week.

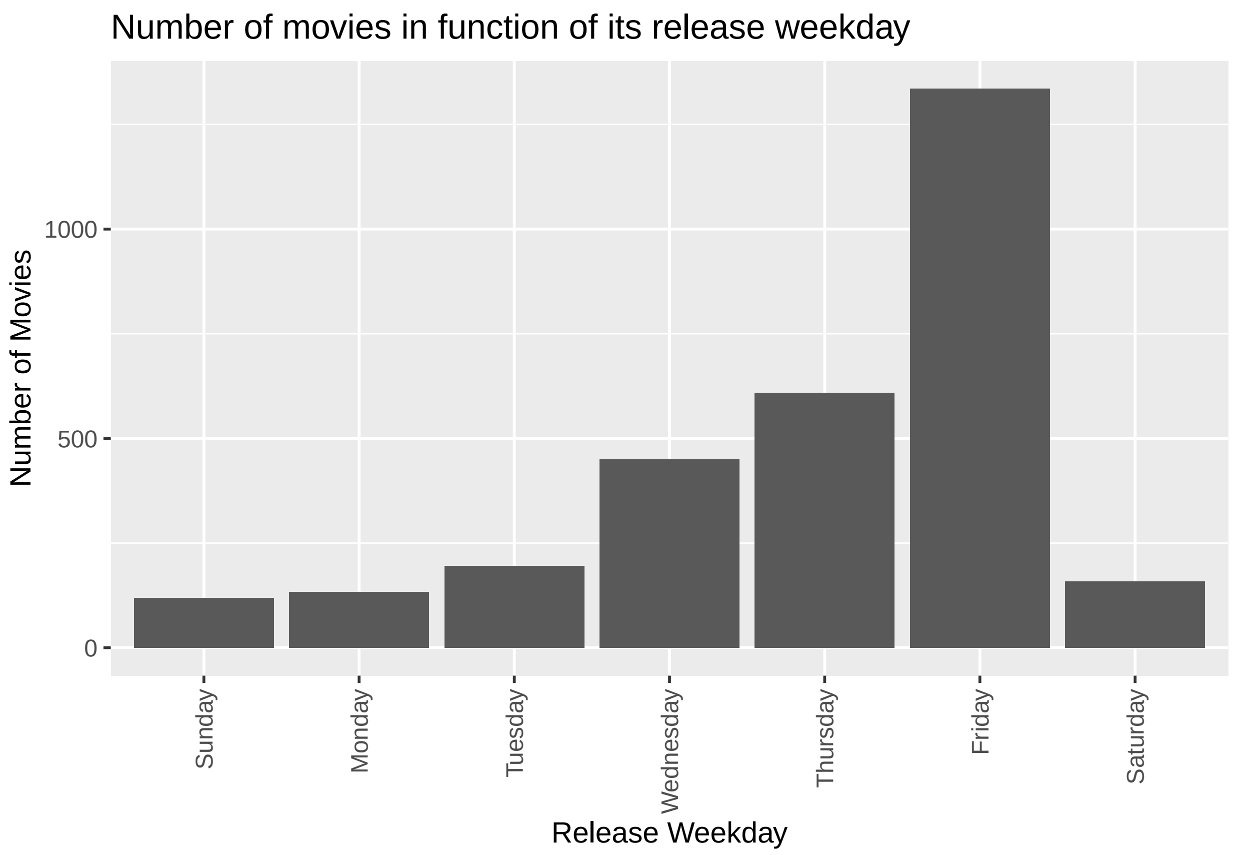
The number of movies per year, month and weekdays could be a factor to consider because it could explain the suprising insights we got.



There are a huge difference on the number of movies made betwween 1975 and older, and 2000 and 2013. The good average revenue we saw around 1975 is explained by few (around 10 and less) movies that got big revenues whereas more than 100 have been made from the early 2000. In these 100 movies, some of them got enough low revenues to low down the average considerably. This is why the average revenues seem to be similar since 1950.



In order, most of movies are released on September, October and December. This is explained by the back-to-school period, Halloween and Christmas. However, more than 200 movies have been released every month and September get among the lowest average revenues.



Friday dominates with more than 1250 movies released and got good average revenues. Having a much larger sample of movies on Friday increases the chances that some of them generated among the biggest and the lowest revenues.

4.15 Production Companies

A production company is generally the one that sets the budget of a movie and makes decisions on hiring directors, actors and writters (crew and casting). Production companies can team up together to work on bigger movies involving many great actors. Thus, we expect that making a movie without a production company will generate a very low revenue. For companies teaming up with others contribute on the budget. Some well-known production companies are known to produce great movies. For example, *Walt Disney Pictures* is known for animated and family movies where The beauty and the beast got a great revenue. Others like *Marvel Studios*, *Warner Bros*, *Paramount Pictures* and *Twentieth Century Fox Film Corporation* are also well-known and generated great movies as well.

In this dataset, we see that a movie can have been made by many production companies. In this case, each of them could have contributed on the budget. However, the contribution for each of them is not known. We only know the budget allowed to make the movie. We could divide the budget by the number of production companies involved but we assume that this will not be accurate.

We have to be careful about the budget because some companies like *Film It Suda* has a budget of 4 but a revenue of 24270441 which is completly absurd. Also, we could consider only the number of movies made by a production company to sort the companies from the most important to the less important. Here again, what if a company is new (made its first movie), invested a high percentage of the budget and this movie generated among the best revenue? In such case, considering only the number of movies is not sufficient.

The objective is to determine which production companies made the best revenue in function of the budget and the number of movies. As we saw in the analysis of the budget, in general, the revenue increases as the budget increases.

In order to obtain the ranking of the production companies, we use the score function defined as:

$$S(\bar{b}, n_m) = \frac{\ln(\bar{b}+1)n_m}{n_t}$$

where

- n_m is the number of movies made per production company and $n_m > 0$ is an integer.
- $\bar{b}$ is the average budget allowed per production company and $\bar{b} > 0$ is a real number.
- n_t is the number of movies in the train dataset and $n_t > 0$ is an integer.

The following steps have to be done in order to achieve the objective:

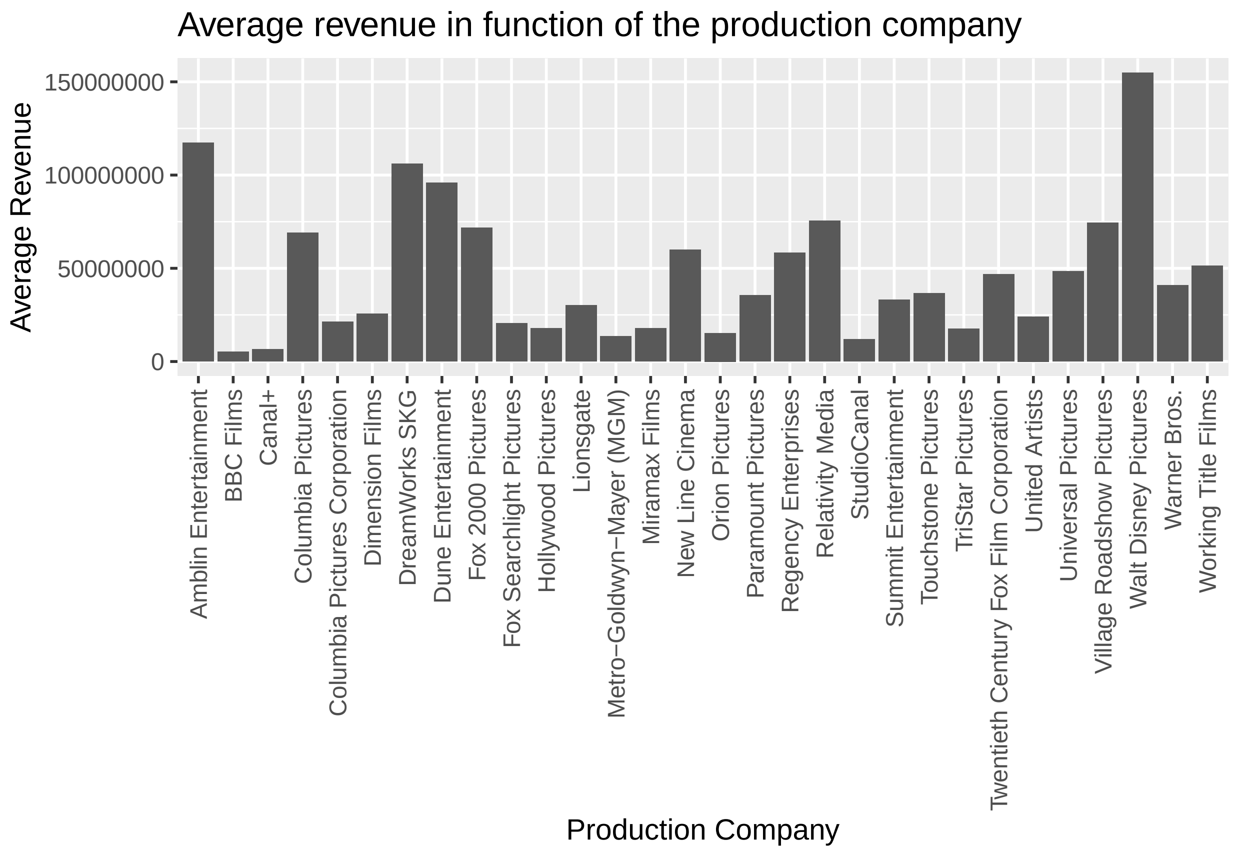
1. Extract in a data frame the production company names for every movie in the dataset with the budget and revenue.
2. Append this data frame to the one containing all movies.
3. Group the data frame by production company name where we keep the average revenue, average budget and number of movies.
4. Calculate the score for every production company and add it to the data frame.
5. Keep the production company with the highest score for every movie of this dataset.
6. Show a bar chart of the average revenue in function of the production company.

```
## Steps 1 and 2: Append the data extracted in a data frame.
train.production.companies <- AppendExtractedData(train, "production_companies")
test.production.companies <- AppendExtractedData(test, "production_companies")

## Steps 3 and 4: Grouping the data and getting the means and calculate the score for every production
train.production.companies <- GetScoreByProductionCompanyName(train, train.production.companies)
test.production.companies <- GetScoreByProductionCompanyName(test, test.production.companies)
print(train.production.companies, n = 25)
```

```
# A tibble: 3,695 x 5
   name                        mean_revenue mean_budget number_of_movies score
   <chr>                              <dbl>       <dbl>            <int> <dbl>
 1 Warner Bros.                        17.5        17.0              202 1.14
 2 Universal Pictures                  17.7        17.0              188 1.07
 3 Paramount Pictures                  17.4        16.8              161 0.900
 4 Twentieth Century Fox F~            17.7        16.9              138 0.779
 5 Columbia Pictures                   18.1        17.3               91 0.523
 6 Metro-Goldwyn-Mayer (MG~            16.4        16.3               84 0.456
 7 New Line Cinema                     17.9        17.2               75 0.430
 8 Walt Disney Pictures                18.9        17.8               62 0.369
 9 Touchstone Pictures                 17.4        17.0               63 0.358
10 Columbia Pictures Corpo~            16.9        16.7               61 0.339
11 TriStar Pictures                    16.7        16.8               53 0.296
12 Relativity Media                    18.1        17.5               48 0.280
13 Canal+                              15.7        16.5               46 0.253
14 United Artists                      17.0        15.8               44 0.231
15 Miramax Films                       16.7        16.4               40 0.219
16 Village Roadshow Pictur~            18.1        17.9               36 0.214
17 Regency Enterprises                 17.9        17.4               31 0.180
18 Dune Entertainment                  18.4        17.5               30 0.175
19 Working Title Films                 17.8        16.9               30 0.169
20 BBC Films                           15.5        16.4               30 0.164
21 DreamWorks SKG                      18.5        17.7               27 0.159
22 StudioCanal                         16.3        16.7               28 0.156
```

```
23 Fox Searchlight Pictures         16.8        16.0                29 0.155
24 Lionsgate                        17.2        16.3                28 0.153
25 Fox 2000 Pictures                18.1        17.1                25 0.142
# ... with 3,670 more rows
```

```
## Step 5: Keep the production company with the highest score for every movie of this dataset.
train.production.companies.best <- ExtractBestScoreFromProductionCompanies(train, train.production.compa
test.production.companies.best <- ExtractBestScoreFromProductionCompanies(test, test.production.compani

## The feature production_company_names is used only for the graph purposes.
train$production_company_scores <- train.production.companies.best
train$production_companies <- NULL

test$production_company_scores <- test.production.companies.best
test$production_companies <- NULL

## Step 6: Show a bar chart of the average revenue in function of the production company.
train.production.companies %>%
    top_n(30) %>%
    ggplot(aes(x = name, y = exp(mean_revenue) + 1)) +
           geom_bar(stat = "identity") +
           ggtitle("Average revenue in function of the production company") +
           labs(x = "Production Company", y = "Average Revenue") +
           theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))
```



```
train.production.companies.best <- NULL
test.production.companies.best <- NULL
train$production_company_names <- NULL
```

```r
train$title <- NULL
test$title <- NULL
```

According to the production company best score for every movie, `Marvel Studios`followed by `Revolution Sun Studios` and `DC Entertainment` are the top 3 production companies where their movies generated the best average revenue.

5 Models

Now that we have observed how the movie revenue behave in function of our features, we start the prediction phase. This phase consists to create a model based on our train dataset in order to train our model and then apply it on the test dataset to obtain the predicted revenue. We know that regression algorithms are suitable for the problem we have to solve. Thus, we start with a linear regression model and then we compared with the Extreme gradient boosting trees and Random forest models.

Before starting to build a model from our train and test datasets, we need to define variables in order to prepare for the regression algorithms that will follow:

- $n_c \in \mathbb{N}_*$ be the number of features (columns) in the dataset.
- $n_r \in \mathbb{N}_*$ be the number of movies (rows) in the dataset.
- $X \in M_{n_r,n_c}(\mathbb{R}^+)$ be the matrix representing a dataset.
- $x_i(x_{i,1}, x_{i,2}, \dots, x_{i,n_c})$ be the movie i in the dataset X where $x_i \in \mathbb{R}^{n_c}$.
- $y = (y_1, y_2, \dots, y_{n_r})$ be the movie revenue to predict in the test dataset where every $y_i \in \mathbb{R}$.

In the train dataset, the number of features is $n_c = 15$ and the number of movies is $n_r = 3000$. The movie revenue known in the train set only is $y = (y_1, y_2, \dots, y_{3000}) = (12314651, 95149435, 13092000, \dots, 82087155)$.

In order to obtain the movie revenue Y in the test dataset, we need to apply on each movie x_i a function f such that $y_i = f(x_i) + \epsilon(x_i)$ or equivalently

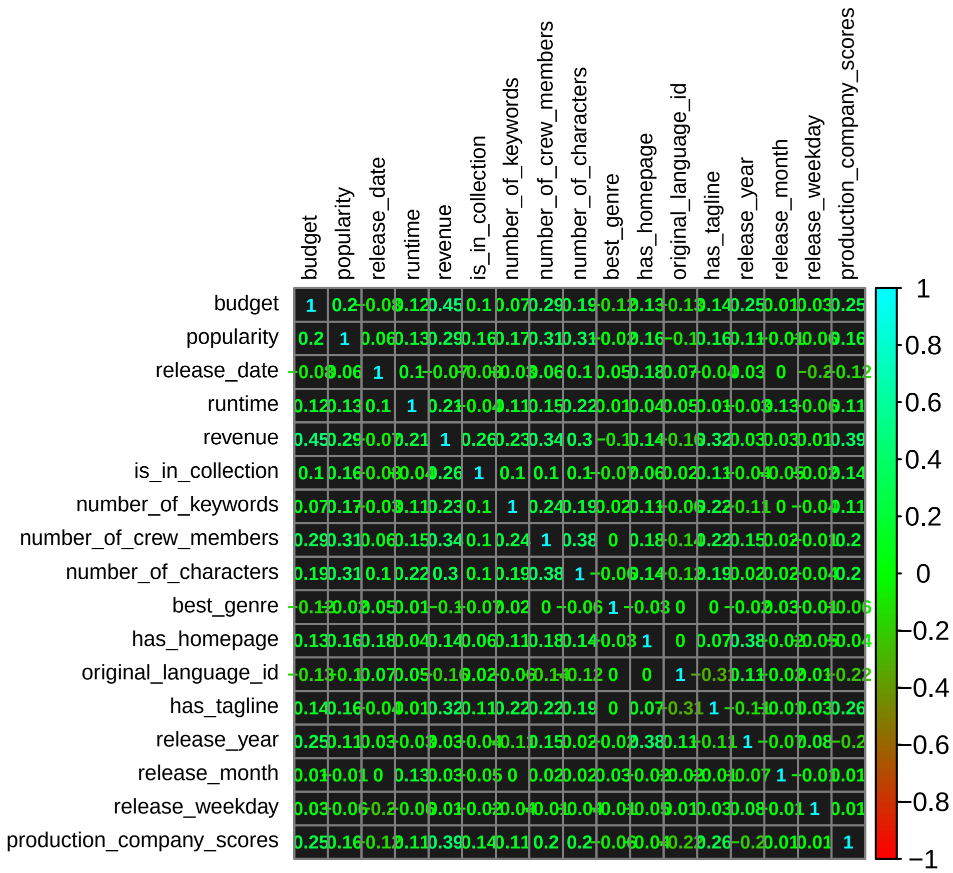
$$y_i - f(x_i) = \epsilon(x_i)$$

where $\epsilon(x_i)$ is the residuals function for $i = 1, 2, \dots, 4398$. Note that $f(x_i)$ is our predictor function for the test dataset X where $n_r = 4398$ and $n_c = 15$.

5.1 Linear Regression Model

The objective is to build a linear regression model knowing how the features correlate together and which ones have the strongest correlation with the revenue.

```r
#train$id <- NULL
#movies.id <- test$id
#test$id <- NULL

train.correlation <- cor(train)

library(corrplot)
corrplot(train.correlation, method = "number", bg = "grey10",
         addgrid.col = "gray50", tl.cex=0.7, tl.col = "black", number.cex = 0.55,
         col = colorRampPalette(c("red", "green", "cyan"))(100))
```



We observe that the strongest correlation is between the budget (x_1) and the revenue (y) with $r(x_1, y) = 0.71$ which is good comparing to the other features.

Let's fit a log linear regression equation $\log(f(x_i)+1) = \beta_0+\beta_1 \log(x_i+1)$ to the points $(x_{1,1}, y_1), (x_{2,1}, y_2), \ldots, (x_{3000,1}, y_{3000})$ and get the equation coefficients $\beta_0, \beta_1 \in \mathbb{R}$.

```
tic()
budget.fit <- lm(formula = log(train$revenue + 1) ~ train$budget)
running_time <- toc()
```

```
0.007 sec elapsed
```

```
linear_regression.running_time <- running_time$toc - running_time$tic
summary(budget.fit)
```

```

Call:
lm(formula = log(train$revenue + 1) ~ train$budget)

Residuals:
     Min       1Q   Median       3Q      Max
-2.29408 -0.03241  0.05703  0.10670  1.06034

Coefficients:
             Estimate Std. Error t value            Pr(>|t|)
(Intercept)  1.724451   0.039649   43.49 <0.0000000000000002 ***
train$budget 0.066085   0.002409   27.43 <0.0000000000000002 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2291 on 2998 degrees of freedom
Multiple R-squared:  0.2007,    Adjusted R-squared:  0.2004
F-statistic: 752.7 on 1 and 2998 DF,  p-value: < 0.00000000000000022
```

Using the estimated intercept value and the estimated budget coefficient, we found that the regression equation is

$$\log(f(x_i) + 1) = 5.23149 + 0.64850 \log(x_i + 1).$$

To calculate the residuals, we use the RMSLE (states for Root Mean Squared Log Error) function

$$RMSLE(y, f(x_j)) = RMSE(\log(y + 1), \log(f(x_j + 1))) = \sqrt{\frac{1}{n_r} \sum_{i=1}^{n_r} (\log(f(x_{i,j}) + 1) - \log(y_i + 1))^2}.$$

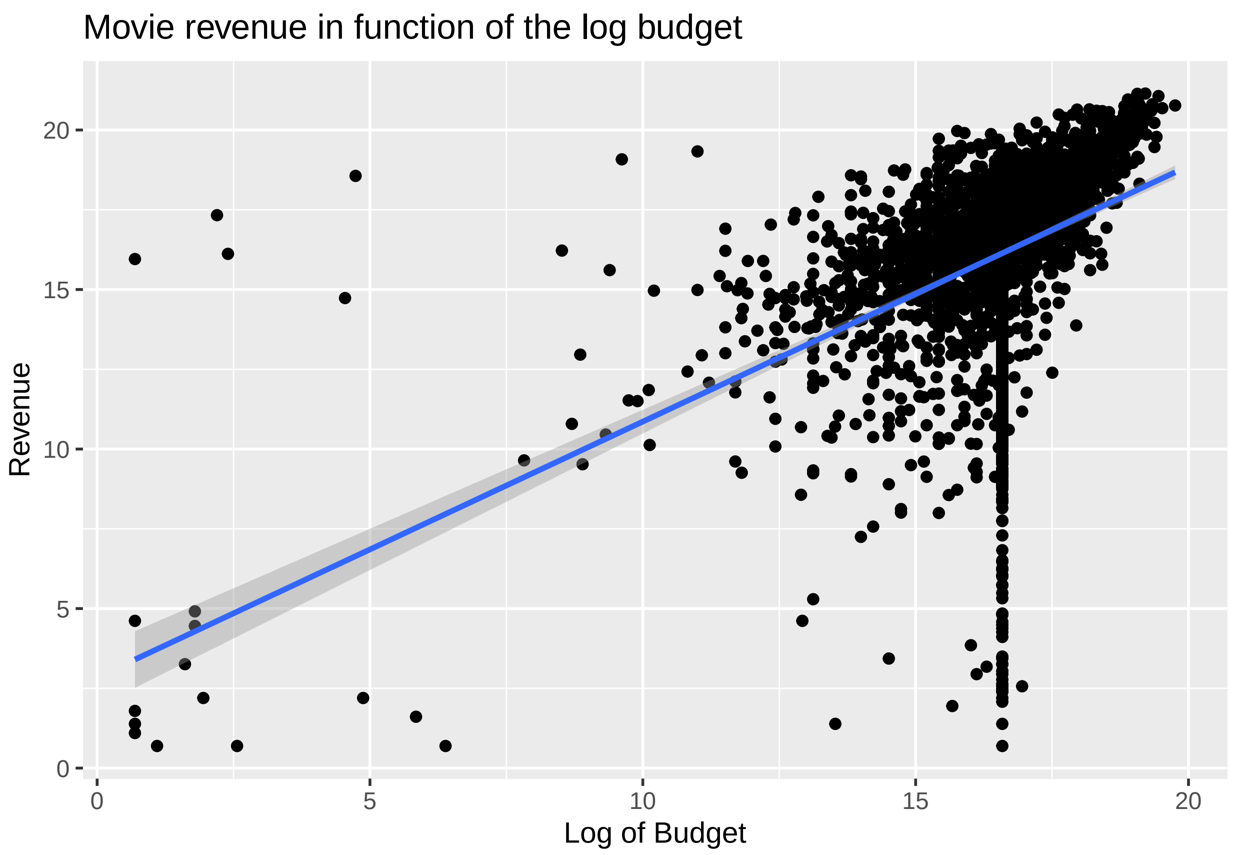
Then, we apply the RMSLE on our predicted revenues $f(X)$ and the revenues y given in the train dataset.

```r
budget.fit.coefficients <- coef(budget.fit)
revenue.estimated <- budget.fit.coefficients["(Intercept)"] + budget.fit.coefficients["train$budget"] *

library(Metrics)
linear_regression.score = rmse(revenue.estimated, log(train$revenue + 1))
cat("RMSLE =", linear_regression.score, "\n\n")
```

```
RMSLE = 0.2290108
```

This regression model penalizes outliers (points that are far from the linear model). Movies with low budget and generating big revenues are not following this linear model and then are penalized. The same applies with big budget allowed to movies that generated low revenues.

```r
train %>%
    ggplot(aes(x = budget, y = revenue)) +
           geom_point() +
           geom_smooth(method = lm) +
           ggtitle("Movie revenue in function of the log budget") +
           labs(x = "Log of Budget", y = "Revenue")
```



5.2 Extreme gradient Boosting Trees Model

The extreme gradient boosting trees model is part of an ensemble set of algorithms using the gradient descent to minimize the loss function and adjusting the residuals with weak learners (Boosting phase). For mathematical details about how the algorithm works, refer to this paper.

The weak learners could be seen as humans pulling a huge structure. Having more humans to pull the heavy structure will increase the total strength and then the structure will move quicker. Of course some of these humans are stronger than others. This is the same for weak learners. In our case, the algorithm uses regression trees as weak learners. Some of these trees could be better depending on how they are built.

5.2.1 Model Preparation Phase

The objective is to initialize the values of the parameters of the `xgboost` model and then find the optimal combination of them in order to minimize the loss function Root Mean Squared Log Error (RMSLE). Here is the list of parameters we know:

- `booster`: We will use the regression trees, hence `gbtree`.
- `objective`: The learning objective function representing the loss function to minimize. Since the suared log error loss objective function is not available yet for R, we use the linear regression objective function `reg:linear`.
- `eval_metric`: The score function used. In our case it is RMSLE but we can use `rmse` where we take the logarithm of the true and predicted revenues instead.

We have to determine the optimal value of the following parameters:

- `eta`: Step size shrinkage of new weights on regulation in order to help preventing overfitting. Corresponds to the eta of this paper.

- `gamma`: A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split. Corresponds to the gamma of this paper.
- `lambda`: L2 Regulation term on weights in order to reduce the overfitting. Corresponds to the lambda of this paper.
- `subsample`: For each tree, a subsample (a fraction in the dataset p_r where $p_r \in]0, 1]$) of the movies is taken randomly.
- `colsample_bytree`: For each tree, a subsample (a fraction in the dataset p_c where $p_c \in]0, 1]$) of the features is taken randomly.
- `min_child_weight`: Defines the minimum sum of weights of all observations required in a child.
- `nrounds`: The number of regression trees
- `max_depth`: The maximum depth of the regression trees

This gives a total of 8 parameters to fine-tune which is to many to apply a grid search algorithm on all of them. Indeed, if we pick n candidates for every parameter, this gives n^8 possible combinations. Therefore, we will use as the initial state the default parameters given by the cross-validation function `xgb.cv`. Then, we apply (in order) the following steps where each of them has to be tested with the RMSLE score:

1. Find the best `eta` and number of trees by reducing the stepsize `eta` and increasing `nrounds` if not enough.
2. Find the best `gamma`.
3. Find the best L2 regulation `lambda`.
4. Find the best L1 regulation `alpha`.
5. Find the best sub-sample ratio of movies per tree with `subsample`.
6. Find the best sub-sample ratio of features per tree with `colsample_bytree`.
7. Find the best minimum sum of weights with `min_child_weight`.
8. Find the best maximum depth of regression trees with `max_depth`.

```r
## Default parameters in xgb.cv.
# parameters <- list(booster          = "gbtree",
#                    eta              = 0.3,
#                    gamma            = 0,
#                    lambda           = 1,
#                    alpha            = 0,
#                    subsample        = 1,
#                    colsample_bytree = 1,
#                    min_child_weight = 1,
#                    max_depth        = 6)

## Optimized parameters.
parameters <- list(booster          = "gbtree",
                   objective        = "reg:linear",
                   eval_metrics     = "rmse",
                   eta              = 0.1,
                   gamma            = 0,
                   lambda           = 4,
                   alpha            = 0,
                   subsample        = 0.95,
                   colsample_bytree = 1,
                   min_child_weight = 1,
                   max_depth        = 4)

cv.number_of_folds = 10
```

5.2.2 Cross Validation

The objective is to train our model on the train set using our optimal parameters and analyse where it predicts well versus where it does not predict correctly. According to that analysis, we will adjust the parameters and if necessary, add features that could help or remove features that are noise.

We proceed to a 10-fold cross-validation to get the optimal number of trees and rmlse score. We use random subsamples representing 80% of the training set. Since we don't have too many movies (3000) and features (17), we can use 10 folds instead of 5 folds. Thus, the training set will be split in 10 test samples where each test sample has 300 movies.

Since the RMSLE evaluation metrics and the loss function associated with the RMSLE evaluation metrics function are not yet supported in the `xgboost` package of R, we have to define them manually. In the vector representation, the objective function is defined as

$$F(\hat{y}, y) = \frac{1}{2}(\ln(\hat{y}+1) - \ln(y+1))^2.$$

This is the RMSE (Root Mean Squared Error) loss function where $f(x_i) = \ln(\hat{y}+1)$ and $y = \ln(y+1)$. We keep only the predicted revenues part $\frac{\partial F(\hat{y},y)}{\partial \hat{y}}$ of the gradient $\nabla F(\hat{y}, y) = \left(\frac{\partial F(\hat{y},y)}{\partial \hat{y}}, \frac{\partial F(\hat{y},y)}{\partial y}\right)$:

$$\begin{aligned}\frac{\partial F(\hat{y}, y)}{\partial \hat{y}} &= \frac{1}{2}\frac{\partial(\ln(\hat{y}+1) - \ln(y+1))^2}{\partial \hat{y}} \\ &= \frac{\ln(\hat{y}+1) - \ln(y+1)}{\hat{y}+1}.\end{aligned}$$

We deduce the second partial derivative over predicted revenues from the first one:

$$\begin{aligned}\frac{\partial^2 F(\hat{y}, y)}{\partial \hat{y}^2} &= \frac{\partial \frac{\ln(\hat{y}+1) - \ln(y+1)}{\hat{y}+1}}{\partial \hat{y}} \\ &= \frac{1 - \ln(\hat{y}+1) + \ln(y+1)}{(\hat{y}+1)^2}.\end{aligned}$$

We can now create our customized objective function returning the gradient and the hessian of F.

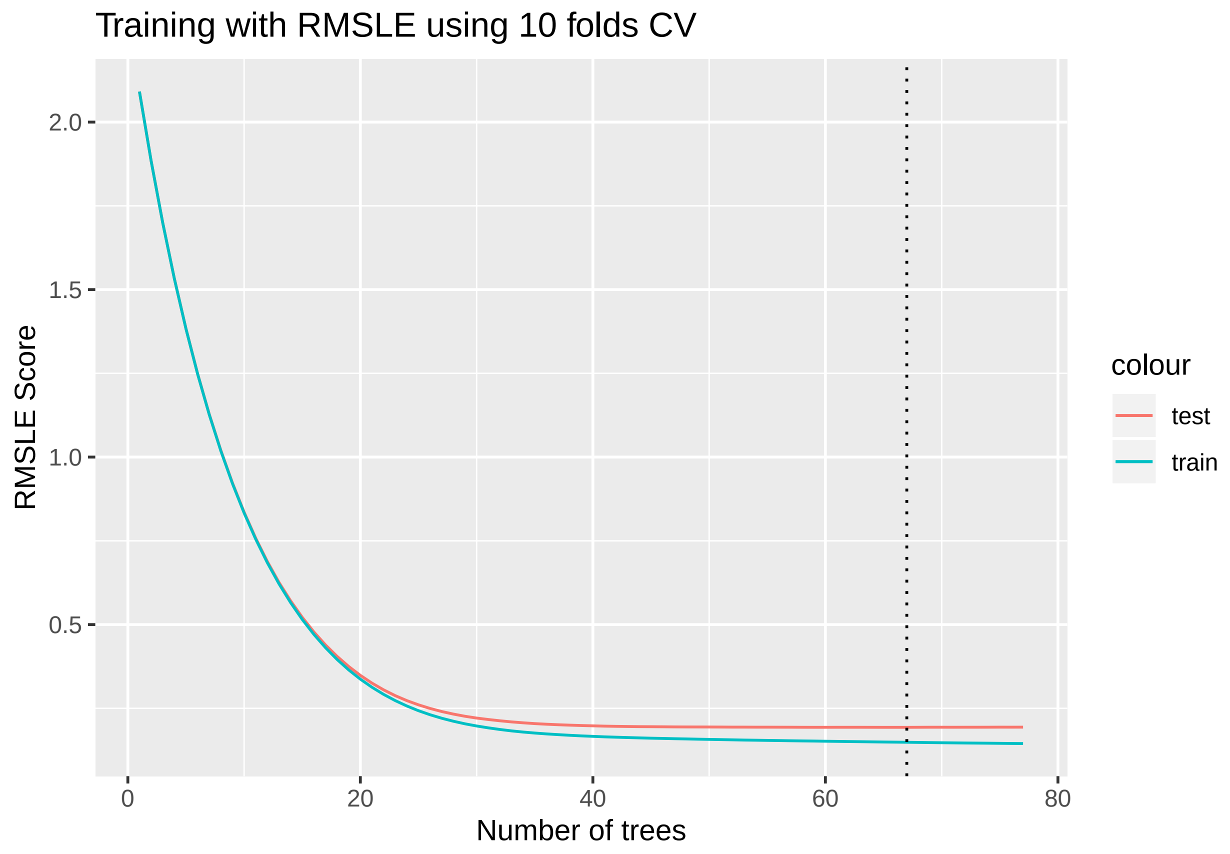
```
SquaredLogLossObjectiveFunction <- function(predicted, train.matrix)
{
    labels <- getinfo(train.matrix, "label")

    predicted <- 0.5 * (log(predicted + 1) - log(labels + 1)) * (log(predicted + 1) - log(labels + 1))
    gradient <- (log(predicted + 1) - log(labels + 1)) / (predicted + 1)
    hessian <- (1 - log(predicted + 1) + log(labels + 1)) / ((predicted + 1) * (predicted + 1))

    return(list(gradient, hessian))
}

RootMeanSquaredLogError <- function(predicted, train.matrix)
{
    labels <- getinfo(train.matrix, "label")

    return(list("rmsle", rmsle(predicted, labels)))
}
```

We can cross-validate with our train dataset by using the RMSE evaluation metrics function where we use $\ln(y+1)$ instead of y. This being said, the budget and production companies score features should be transformed the same way to make them on the same scale as the revenue.

```r
revenues <- log(train$revenue + 1)
train$revenue <- NULL

train$production_company_scores <- log(train$production_company_scores + 1)
test$production_company_scores <- log(test$production_company_scores + 1)

library(xgboost)
train.matrix <- xgb.DMatrix(data.matrix(train), label = revenues)

model.cv <-xgb.cv(data = train.matrix,
                  nfold = cv.number_of_folds,
                  #obj = SquaredLogLossObjectiveFunction,
                  #feval = RootMeanSquaredLogError,
                  params = parameters,
                  nrounds = 200,
                  early_stopping_rounds = 10,
                  maximize = FALSE,
                  verbose = FALSE)

print(model.cv)
```

```
##### xgb.cv 10-folds
 iter train_rmse_mean train_rmse_std test_rmse_mean test_rmse_std
    1       2.0911354   0.0006475000      2.0912440   0.006318196
    2       1.8849747   0.0005759790      1.8851136   0.005948050
    3       1.6995426   0.0005094892      1.6999382   0.005980166
    4       1.5327339   0.0004342102      1.5333326   0.005638785
    5       1.3828664   0.0004004518      1.3836120   0.005439185
    6       1.2481579   0.0003743254      1.2492621   0.005545814
    7       1.1271217   0.0003621254      1.1284152   0.005595810
    8       1.0184092   0.0003480226      1.0200570   0.005605460
    9       0.9207985   0.0004311051      0.9229246   0.005464684
   10       0.8333087   0.0004444748      0.8358167   0.005223376
   11       0.7548803   0.0004700198      0.7578980   0.005142945
   12       0.6846339   0.0005127476      0.6882305   0.005104663
   13       0.6217232   0.0005946595      0.6259726   0.004980310
   14       0.5655703   0.0006584269      0.5709431   0.005128153
   15       0.5154475   0.0007266211      0.5216885   0.005305459
   16       0.4707969   0.0007893370      0.4779076   0.005599254
   17       0.4310821   0.0008671320      0.4392120   0.005833855
   18       0.3957967   0.0009472885      0.4049665   0.006298454
   19       0.3646233   0.0009313525      0.3747886   0.006814665
   20       0.3370931   0.0010088312      0.3483635   0.007274900
   21       0.3128447   0.0011000325      0.3252139   0.007828361
   22       0.2915927   0.0011555343      0.3051260   0.008506196
   23       0.2730524   0.0013105283      0.2878221   0.009034656
   24       0.2569250   0.0014164153      0.2730266   0.009822022
   25       0.2428705   0.0014906387      0.2603975   0.010562593
   26       0.2308125   0.0015082514      0.2496153   0.011303991
   27       0.2204107   0.0015861233      0.2404754   0.011958199
   28       0.2114404   0.0016924143      0.2327720   0.012435648
   29       0.2037549   0.0017145730      0.2262786   0.012955533
   30       0.1972569   0.0018144294      0.2209534   0.013399804
   31       0.1917169   0.0018421742      0.2166624   0.013755782
   32       0.1868282  0.0018818365      0.2128300   0.014177041
   33       0.1826417  0.0019109028      0.2094598   0.014526005
   34       0.1791402  0.0019031795      0.2067641   0.015115969
   35       0.1761682  0.0019520350      0.2043926   0.015340610
   36       0.1736325  0.0019977475      0.2025580   0.015754621
   37       0.1715134  0.0020237743      0.2010224   0.016146353
   38       0.1694940  0.0021387557      0.1997330   0.016413102
   39       0.1677155  0.0021669804      0.1986377   0.016629183
   40       0.1662899  0.0021673700      0.1977872   0.016771365
   41       0.1648905  0.0022521787      0.1968315   0.016693963
   42       0.1637280  0.0022433716      0.1961932   0.016933139
   43       0.1627185  0.0023107366      0.1956703   0.017121665
   44       0.1617382  0.0024222295      0.1952913   0.017093236
   45       0.1607950  0.0024111678      0.1951126   0.017162707
   46       0.1601298  0.0023907873      0.1948695   0.017283479
   47       0.1593282  0.0024519127      0.1945717   0.017310884
   48       0.1586571  0.0024524477      0.1944177   0.017256660
   49       0.1578789  0.0023826755      0.1943043   0.017173432
   50       0.1572301  0.0024324534      0.1942881   0.017287115
   51       0.1566128  0.0024564473      0.1940667   0.017220151
   52       0.1559459  0.0024642697      0.1938804   0.017487624
   53       0.1552655  0.0023521728      0.1938569   0.017488120
   54       0.1547621  0.0023421098      0.1937122   0.017607466
   55       0.1542157  0.0022938084      0.1936698   0.017565653
   56       0.1537074  0.0023146166      0.1936161   0.017685177
   57       0.1532081  0.0023407177      0.1935130   0.017699865
   58       0.1527294  0.0023060981      0.1933921   0.017803568
   59       0.1522776  0.0022845395      0.1932760   0.017834599
   60       0.1517429  0.0022765290      0.1932977   0.017862730
   61       0.1512514  0.0023200477      0.1933945   0.017943107
   62       0.1507875  0.0023457521      0.1934173   0.018040545
   63       0.1503546  0.0024037032      0.1933976   0.018113993
   64       0.1499019  0.0024382379      0.1933017   0.018138301
   65       0.1494485  0.0024730615      0.1932071   0.018132353
   66       0.1489999  0.0024912568      0.1932525   0.018082728
   67       0.1486774  0.0025243316      0.1931991   0.018150247
   68       0.1482024  0.0025363999      0.1933929   0.018112317
   69       0.1476718  0.0024462442      0.1935416   0.018063999
   70       0.1473036  0.0024575816      0.1935637   0.018042557
   71       0.1468694  0.0024575991      0.1934670   0.018114356
   72       0.1464816  0.0024201629      0.1935824   0.018021860
   73       0.1461551  0.0024842424      0.1935348   0.018092585
   74       0.1458346  0.0024251513      0.1936504   0.018110370
   75       0.1454313  0.0025107438      0.1937828   0.018031873
   76       0.1450883  0.0024398848      0.1936894   0.017977396
   77       0.1447799  0.0024525627      0.1937808   0.018099092
 iter train_rmse_mean train_rmse_std test_rmse_mean test_rmse_std
Best iteration:
 iter train_rmse_mean train_rmse_std test_rmse_mean test_rmse_std
   67       0.1486774    0.002524332      0.1931991    0.01815025
```

```
xgboost.score <- model.cv$evaluation_log$test_rmse_mean[model.cv$best_iteration]
cv.plot.title <- paste("Training with RMSLE using", cv.number_of_folds, "folds CV")
print(ggplot(model.cv$evaluation_log, aes(x = iter)) +
          geom_line(aes(y = test_rmse_mean, colour = "test")) +
          geom_line(aes(y = train_rmse_mean, colour = "train")) +
          geom_vline(xintercept = model.cv$best_iteration, linetype="dotted") +
          ggtitle(cv.plot.title) +
          labs(x = "Number of trees", y = "RMSLE Score"))
```



We want to see which features generated gains when used by the regression trees models.

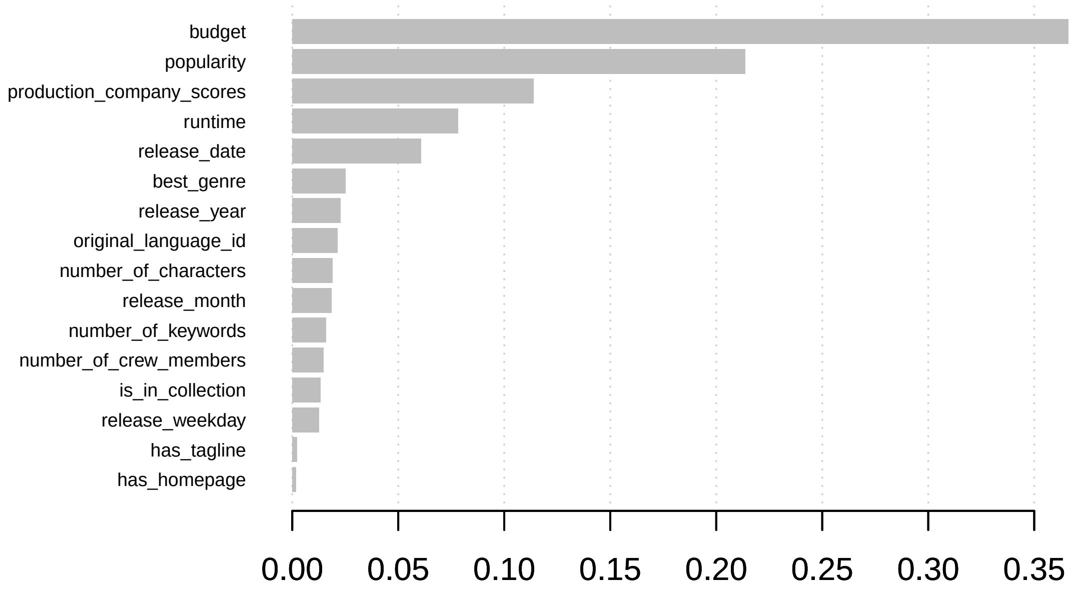
```
tic()
model <- xgb.train(data = train.matrix,
                   params = parameters,
                   nrounds = model.cv$best_iteration,
                   verbose = FALSE)
running_time <- toc()
```

```
0.315 sec elapsed
```

```
xgboost.running_time <- running_time$toc - running_time$tic

feature.names <- names(train)
importance_matrix <- xgb.importance(feature.names, model=model)

print(importance_matrix)
```

```
                      Feature        Gain       Cover   Frequency
 1:                    budget 0.366068193 0.215474187 0.174358974
 2:                popularity 0.213651235 0.146161338 0.155128205
 3: production_company_scores 0.113797601 0.082839954 0.098717949
 4:                  runtime 0.078333383 0.088973245 0.103846154
 5:             release_date 0.060759198 0.077771812 0.098717949
 6:               best_genre 0.025084404 0.039281719 0.048717949
 7:             release_year 0.022873800 0.048364432 0.039743590
 8:     original_language_id 0.021294658 0.062425559 0.051282051
 9:     number_of_characters 0.019028998 0.048713210 0.041025641
10:            release_month 0.018539501 0.043345495 0.052564103
11:       number_of_keywords 0.016031921 0.036333745 0.035897436
12:    number_of_crew_members 0.014638952 0.028239479 0.038461538
13:         is_in_collection 0.013285122 0.034734574 0.015384615
14:          release_weekday 0.012628530 0.033161453 0.029487179
15:              has_tagline 0.002152635 0.005933576 0.006410256
16:             has_homepage 0.001831870 0.008246223 0.010256410
```

```r
xgb.plot.importance(importance_matrix)
```



5.2.3 Predictions

The objective is to apply our optimal model to our test dataset in order to calculate and obtain the predictions on the movies revenue in a CSV file. Since we will obtain our predictions as $\hat{Y} = \ln(\hat{y}+1)$, we have to transform them to $\hat{y} = \exp(\hat{Y}) + 1$ in order to get the real predicted movie revenues.

```r
test.matrix <- xgb.DMatrix(data.matrix(test))

prediction.test <- as.integer(exp(predict(model, test.matrix)) + 1)
xgboost.submission <- data.frame(id = seq(3001, 3000 + nrow(test)),
                              revenue = prediction.test)
write.csv(xgboost.submission, "Submissions/Submission_XGBoost.csv", row.names = FALSE)
```

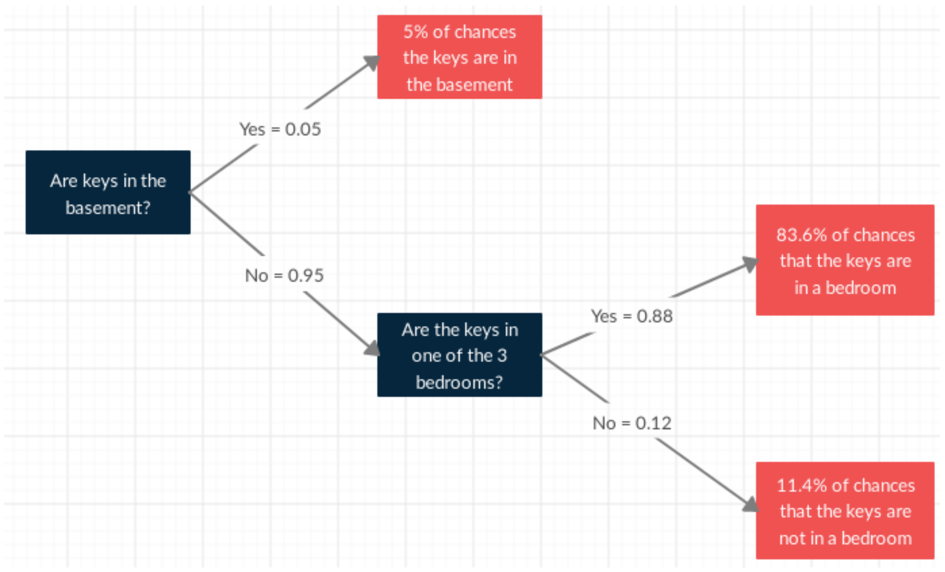
5.3 Random Forest Model

The Random forest model is part of the ensemble set of models like the gradient boosting trees one. This model uses decision trees as weak learners (see example here) in order to build its forest of n decision trees. Each decision tree in the forest uses a random subset of features on each question. Only a random subset of the training data points is used to answer a question. The purpose is to not use the same source of data but to increase the diversity in order to get more robust overall predictions. Then, the average of all decision tree estimates in the forest is taken as the prediction. The Random forest model is explained in details here.

To make decision trees intuitive for everyone, we will describe what it represents in our everyday life. Let's say that you are searching for your keys. You do not know yet but your keys are on a small desk close to your bed. Since you are pretty sure that they are in the house, then you start your researches in the house. However, they can at any place in the house making the model not enough accurate. You need the help of weak learners (decision trees) in order to facilitate your search. Right now, this can be seen as a model without any trees.

Suppose that your house has a basement and is a one-story house. The question you are asking yourself is: Are my keys in the basement? You are pretty sure that they are not in the basement. Thus, you just limit the range of your researches and then gain accuracy. However, it is still not accurate because there are too many places to look for. So you are asking yourself a second question: Are my keys in one of the 3 bedrooms? According to your past experiences, most of the time your let your keys in one of the 3 bedrooms. This limits the researches range to 3 rooms in the house. The weak learner is now a 2-level decision tree. Going further will give a very accurate result but if your habits change in the future, it could lead to the wrong room at the third level. This is the equivalent way to say that your model overfits.

Here is the decision tree representing our example where we quantified the probabilities for each decision:


Are keys in the basement?
Yes = 0.05
5% of chances the keys are in the basement
No = 0.95
Are the keys in one of the 3 bedrooms?
Yes = 0.88
83.6% of chances that the keys are in a bedroom
No = 0.12
11.4% of chances that the keys are not in a bedroom

5.3.1 Preparing Models

The objective is to initialize the values of the parameters of the `randomForest` model and then find the optimal combination of them in order to minimize the loss function Root Mean Squared Log Error (RMSLE). Here is the list of parameters to determine:

- `ntree`: The number of decision trees to grow.
- `mtry`: Number of variables randomly sampled as candidates at each split. Default: mtry = number of features / 3.
- `nodesize`: Minimum size of terminal nodes. Default: 5
- `maxnodes`: Maximum number of terminal nodes trees in the forest can have.
- `nPerm`: Number of times the Out-Of-Bag (OOB) data are permuted per tree for assessing variable importance.

This gives a total of 4 parameters to fine-tune which may be slow to apply a grid search algorithm on all of them. Indeed, if we pick n candidates for every parameter, this gives n^4 possible combinations. Therefore, we will use as the initial state the default parameters.

5.3.2 Cross Validation

The objective is to train our model on the train set using our optimal parameters and analyse where it predicts well versus where it does not predict well. According to that analysis, we will adjust the parameters and if necessary, add features that could help or remove features that are noise. Note that the Random forest model applies the Mean Squared Error (MSE) loss function:

$$MSE(y, \hat{y}) = \frac{1}{n_r} \sum_{i=1}^{n_r} (y_i - \hat{y}_i)^2.$$

Using $\ln(y+1)$ and $\ln(\hat{y}+1)$ instead of y and $\hat{y}$ gives the MSLE loss function instead. It follows that applying the square root on the resulting MSLE gives the RMSLE.

```r
library(randomForest)

tic()
model <- randomForest(x = train,
                      y = revenues,
                      ntree = 100,
                      mtry = 5,
                      maxnodes = 30,
                      nodesize = 5,
                      nPerm = 1,
                      importance = TRUE)
running_time <- toc()
```

```
0.646 sec elapsed
```

```r
random_forest.running_time <- running_time$toc - running_time$tic

model$mse <- sqrt(model$mse)
results <- data.frame(number_of_trees = seq(1, model$ntree),
                      RMSLE = model$mse)
print(results)
```

```
    number_of_trees     RMSLE
1                 1 0.2199122
2                 2 0.2037298
3                 3 0.2068333
4                 4 0.2073705
5                 5 0.2066119
6                 6 0.2022403
7                 7 0.2003667
8                 8 0.1995367
9                 9 0.1992651
10               10 0.2016841
11               11 0.2044841
12               12 0.2039798
13               13 0.2049677
14               14 0.2030085
15               15 0.2013317
16               16 0.2004825
17               17 0.2004702
18               18 0.1999474
19               19 0.2009961
20               20 0.2007153
```

```
21                21 0.1998504
22                22 0.1991766
23                23 0.1995804
24                24 0.1995575
25                25 0.1994052
26                26 0.1997359
27                27 0.1996497
28                28 0.2003069
29                29 0.2004476
30                30 0.2006554
31                31 0.2005586
32                32 0.2000968
33                33 0.2002999
34                34 0.2001927
35                35 0.2002033
36                36 0.2003633
37                37 0.1999996
38                38 0.2002134
39                39 0.1994672
40                40 0.1994709
41                41 0.1989649
42                42 0.1990850
43                43 0.1990897
44                44 0.1991601
45                45 0.1987483
46                46 0.1993306
47                47 0.1994306
48                48 0.1991211
49                49 0.1990868
50                50 0.1993515
51                51 0.1994573
52                52 0.1992552
53                53 0.1993900
54                54 0.1995834
55                55 0.1991433
56                56 0.1991715
57                57 0.1991009
58                58 0.1992604
59                59 0.1994772
60                60 0.1998638
61                61 0.1997524
62                62 0.1996636
63                63 0.1996061
64                64 0.1996185
65                65 0.1993808
66                66 0.1994042
67                67 0.1991375
68                68 0.1992891
69                69 0.1991870
70                70 0.1992232
71                71 0.1992742
72                72 0.1993911
73                73 0.1994899
74                74 0.1994017
```

```
75                75 0.1992380
76                76 0.1991016
77                77 0.1991294
78                78 0.1990963
79                79 0.1990054
80                80 0.1988701
81                81 0.1989581
82                82 0.1991551
83                83 0.1991814
84                84 0.1992083
85                85 0.1990885
86                86 0.1990228
87                87 0.1990772
88                88 0.1989666
89                89 0.1988963
90                90 0.1987636
91                91 0.1987291
92                92 0.1988615
93                93 0.1988814
94                94 0.1989009
95                95 0.1987683
96                96 0.1988478
97                97 0.1988429
98                98 0.1986749
99                99 0.1987325
100              100 0.1986658
```

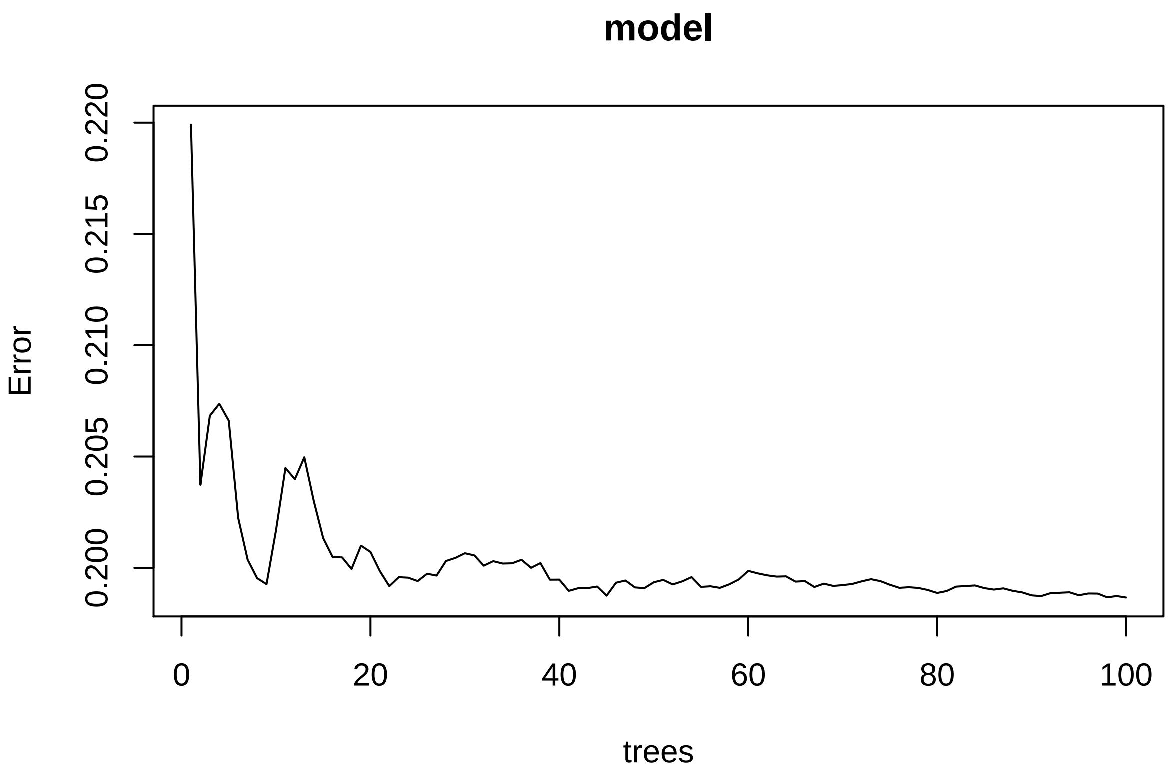
```r
random_forest.score <- results$RMSLE[which.min(results$RMSLE)]
cat("The number of trees minimizing the RMSLE is:", results$number_of_trees[which.min(results$RMSLE)],
```

```
The number of trees minimizing the RMSLE is: 100
```

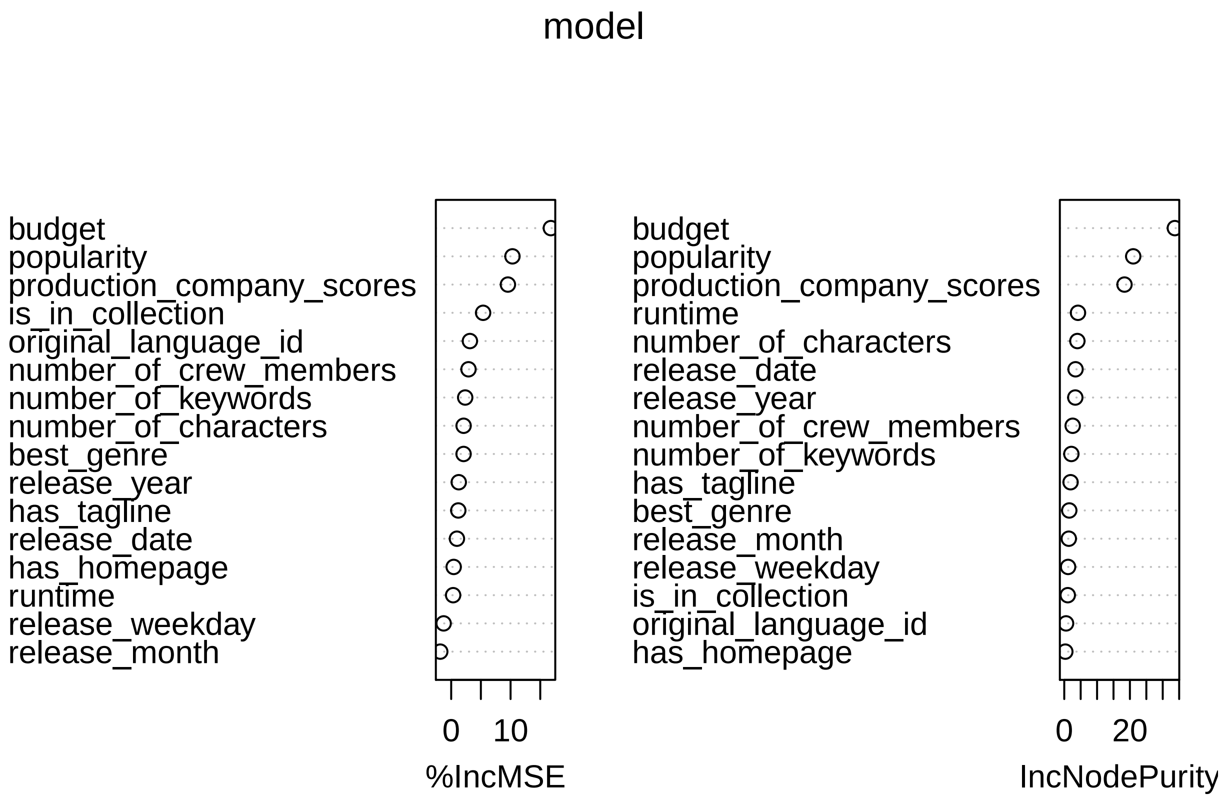
```r
cat("The minimum RMSLE is:", random_forest.score, "\n")
```

```
The minimum RMSLE is: 0.1986658
```

```r
plot(model)
```



```r
print(varImpPlot(model))
```



```
                           %IncMSE IncNodePurity
budget                  16.7834874    33.6887353
popularity              10.3139511    21.0087691
```

```
release_date                0.9661224     3.4302050
runtime                     0.3291503     4.1769590
is_in_collection            5.3757803     1.0720718
number_of_keywords          2.3588141     2.1584580
number_of_crew_members      2.9146555     2.5620870
number_of_characters        2.0917293     3.9957564
best_genre                  2.0896914     1.5263085
has_homepage                0.4239002     0.3216246
original_language_id        3.1528974     0.5509197
has_tagline                 1.1958174     1.9212420
release_year                1.2765553     3.3567306
release_month              -1.8395565     1.3863792
release_weekday            -1.2729894     1.1669922
production_company_scores   9.5511375    18.3454739
```

5.3.3 Predictions

The objective is to apply our optimal model to our test dataset in order to calculate and obtain the predictions on the movies revenue in a CSV file. Since we will obtain our predictions as $\hat{Y} = \ln(\hat{y} + 1)$, we have to transform them to $\hat{y} = \exp(\hat{Y}) + 1$ in order to get the real predicted movie revenues.

```r
predictions <- predict(model, newdata = test)

submission <- data.frame(id = seq(3001, 3000 + nrow(test)),
                         revenue = as.integer(exp(predictions) + 1))
write.csv(submission, "Submissions/Submission_RandomForest.csv", row.names = FALSE)
```

6 Conclusion

We conclude on the following points:

1. A comparison table of the models with their RMSLE score and runtime.
2. The top 10 of movies generating the biggest revenue.
3. From our best scores for predicting the revenues, we want to know why the model did not predict well for some movies?

6.1 Models Comparison Table

model	rmsle	running_time
Linear Regression	0.2290108	0.007
Extreme Gradient Boosting Trees	0.1931991	0.315
Random Forest	0.1986658	0.646

All of these models ran under one second, hence the best model is the extreme gradient boosting trees because of its lowest prediction errors.

6.2 Top Movie Revenues Predicted

We present the top 10 movies that generated the biggest predicted revenues:

```
      id revenue
1  3045      22
2  3097      22
3  3210      22
4  3217      22
5  3240      22
6  3261      22
7  3389      22
8  3459      22
9  3479      22
10 3628      22
11 3788      22
12 3879      22
13 3943      22
14 3954      22
15 3964      22
16 3986      22
17 4051      22
18 4136      22
19 4420      22
20 4551      22
21 4590      22
22 4764      22
23 4797      22
24 4847      22
25 4907      22
26 5001      22
27 5150      22
28 5295      22
29 5408      22
30 5558      22
31 5624      22
32 5644      22
33 5656      22
34 5662      22
35 5683      22
36 5864      22
37 6136      22
38 6139      22
39 6152      22
40 6199      22
41 6203      22
42 6546      22
43 6612      22
44 6663      22
45 6679      22
46 6681      22
47 6796      22
48 6868      22
49 6917      22
50 7184      22
51 7207      22
52 7363      22
53 7393      22
```

6.3 Retrospective on Predictions