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(tictoc)

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

```r
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)
```

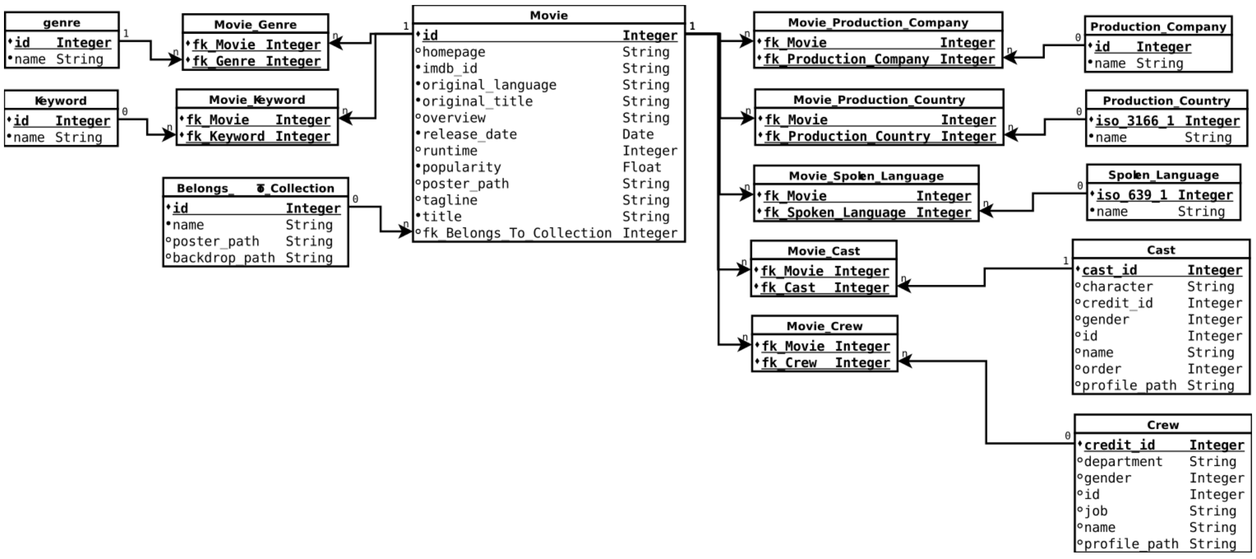
```
Number of rows:  3000
Number of columns:  22

                           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
```

```r
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:

```r
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.

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

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

```r
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).

```r
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.

```r
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.

```r
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 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:

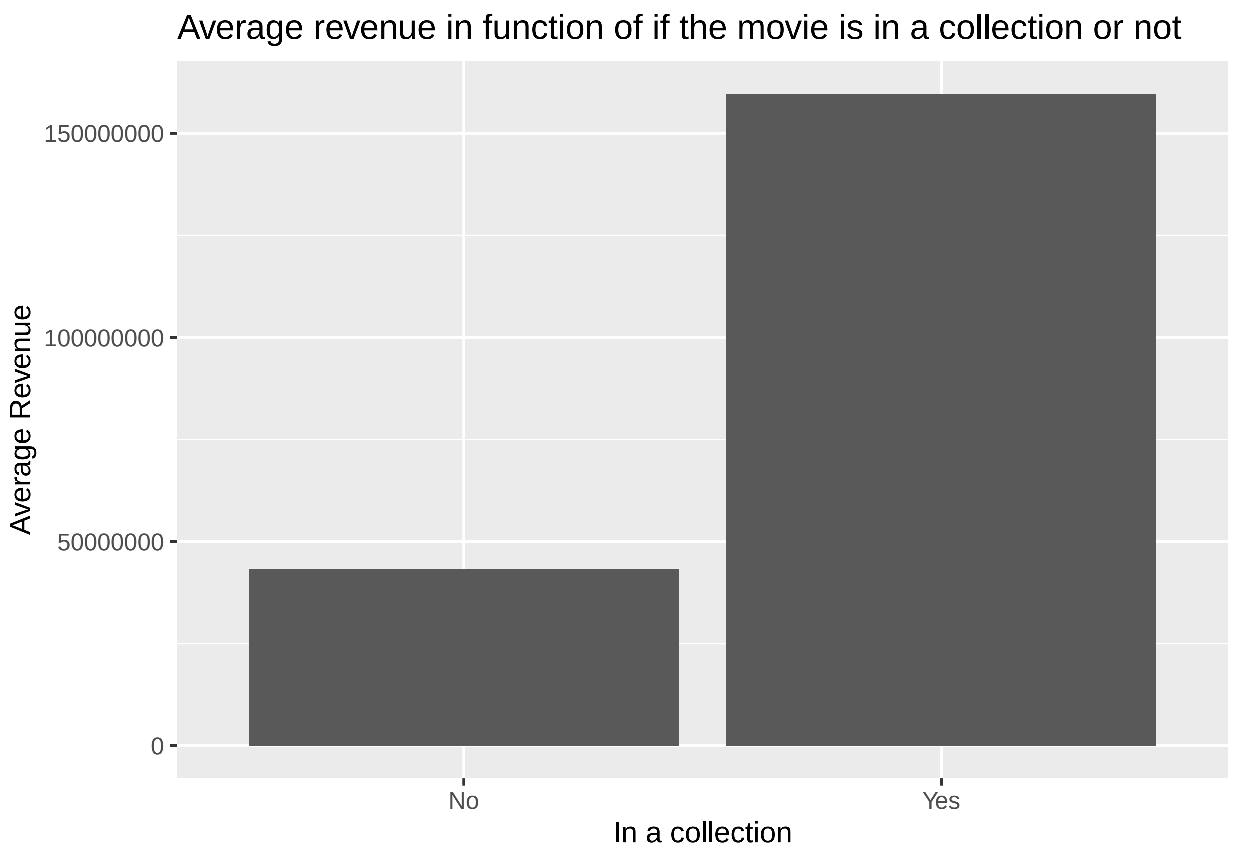
- the movie was enough good to gain a large revenue that the producer(s) decided to make a second movie and so on as a series.
- the same or another producer decided to do a remake of the movie many years after the original one (e.g. Karate kid, Superman).

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

```
train$is_in_collection <- unlist(lapply(train$belongs_to_collection, function(collection) ifelse(collec
train$belongs_to_collection <- NULL

test$is_in_collection <- unlist(lapply(test$belongs_to_collection, function(collection) ifelse(collecti
test$belongs_to_collection <- NULL
```

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

```
library(ggplot2)
library(dplyr)

train %>%
    group_by(is_in_collection) %>%
    summarise(mean = mean(revenue)) %>%
    ggplot(aes(x = factor(is_in_collection), y = mean)) +
          geom_bar(stat = "identity") +
          ggtitle("Average revenue in function of if the movie is in a collection or not") +
          labs(x = "In a collection", y = "Average Revenue") +
          scale_x_discrete(labels = c("No", "Yes"))
```



There is a significant difference where the revenue of movies in a collection is much greater than the ones not in a collection.

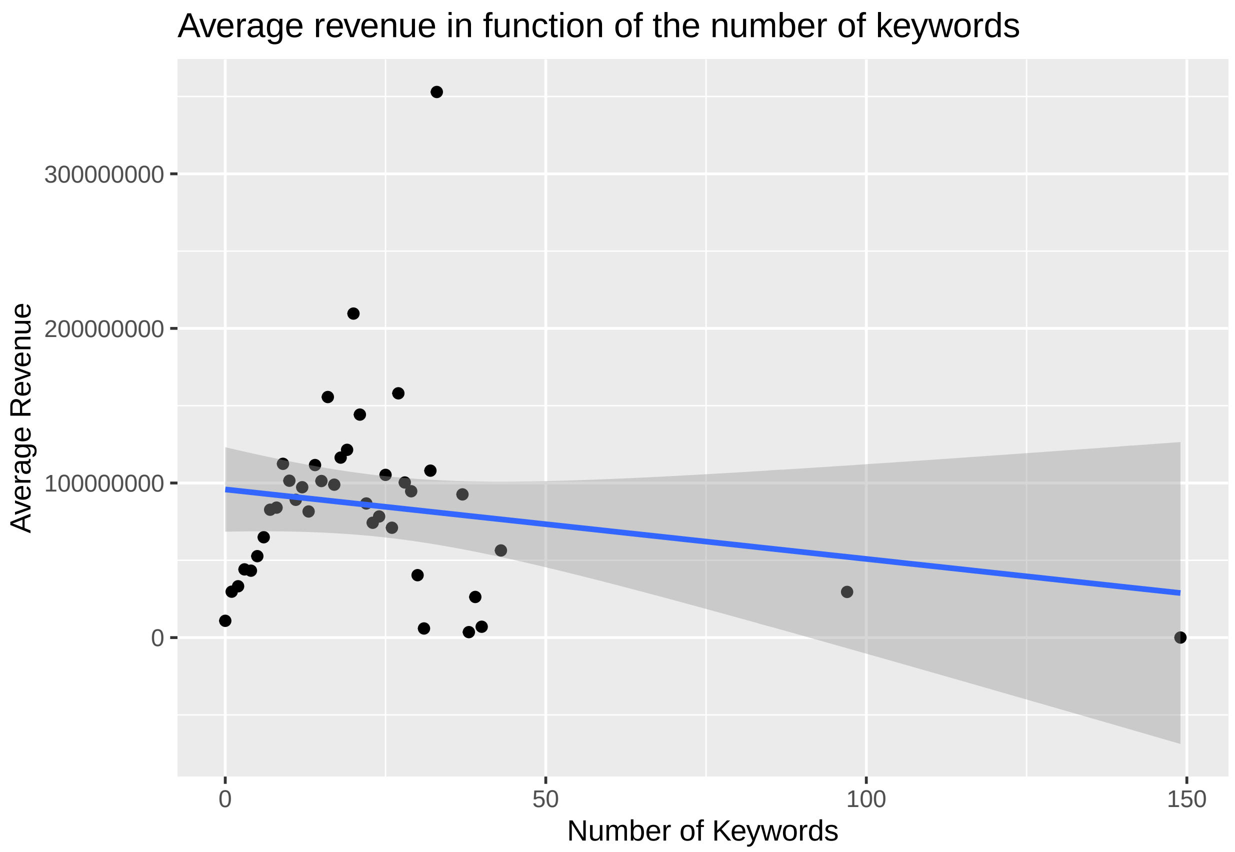
4.2 Movie Keywords

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`.

```r
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 revenue in function of the number of keywords.

```r
train %>%
    group_by(number_of_keywords) %>%
    summarise(mean = mean(revenue)) %>%
    ggplot(aes(x = number_of_keywords, y = mean)) +
           geom_point() +
           geom_smooth(method = lm) +
           ggtitle("Average revenue in function of the number of keywords") +
           labs(x = "Number of Keywords", y = "Average Revenue")
```



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.

4.3 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.

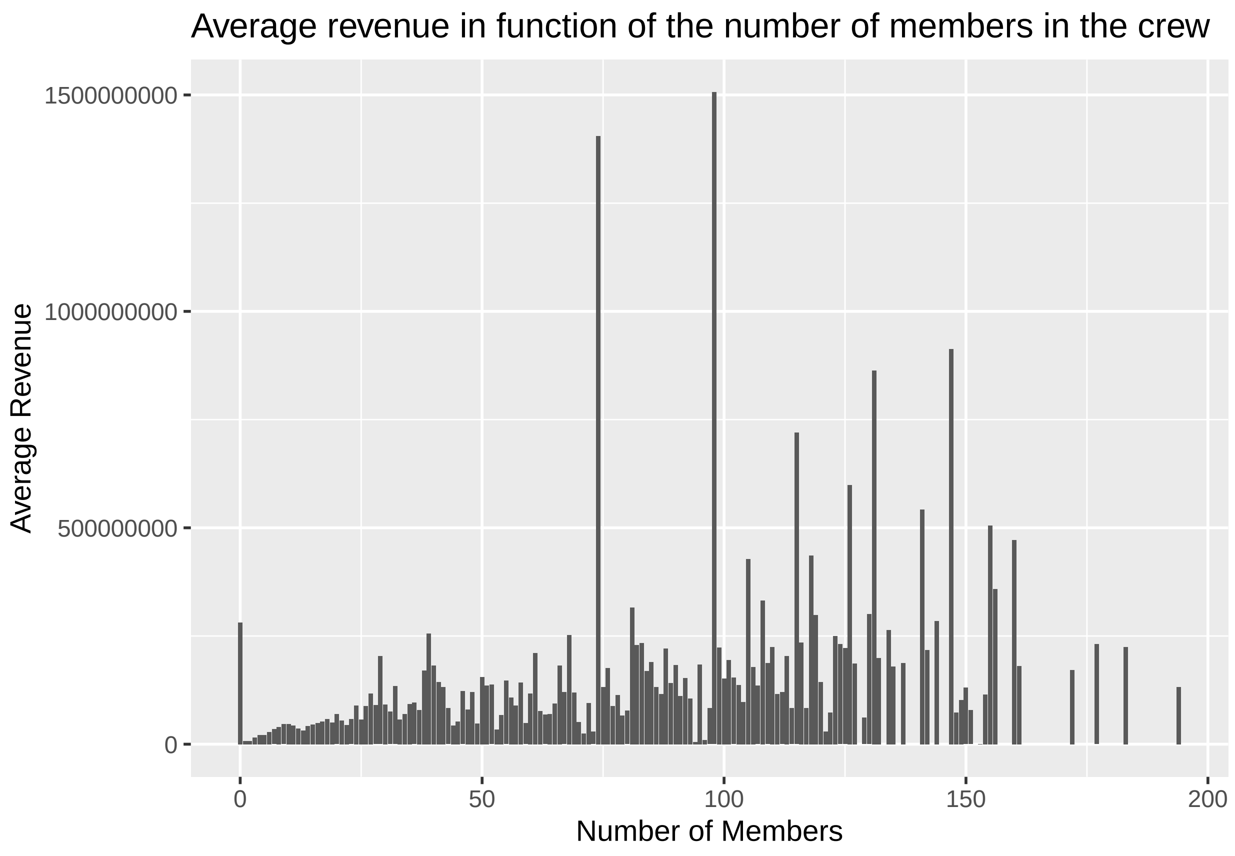
```
train$number_of_directors <- unlist(lapply(train$crew, CountMembersInCrewByJobType, job.type = "Directo
test$number_of_directors <- unlist(lapply(test$crew, CountMembersInCrewByJobType, job.type = "Director")
```

We 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.

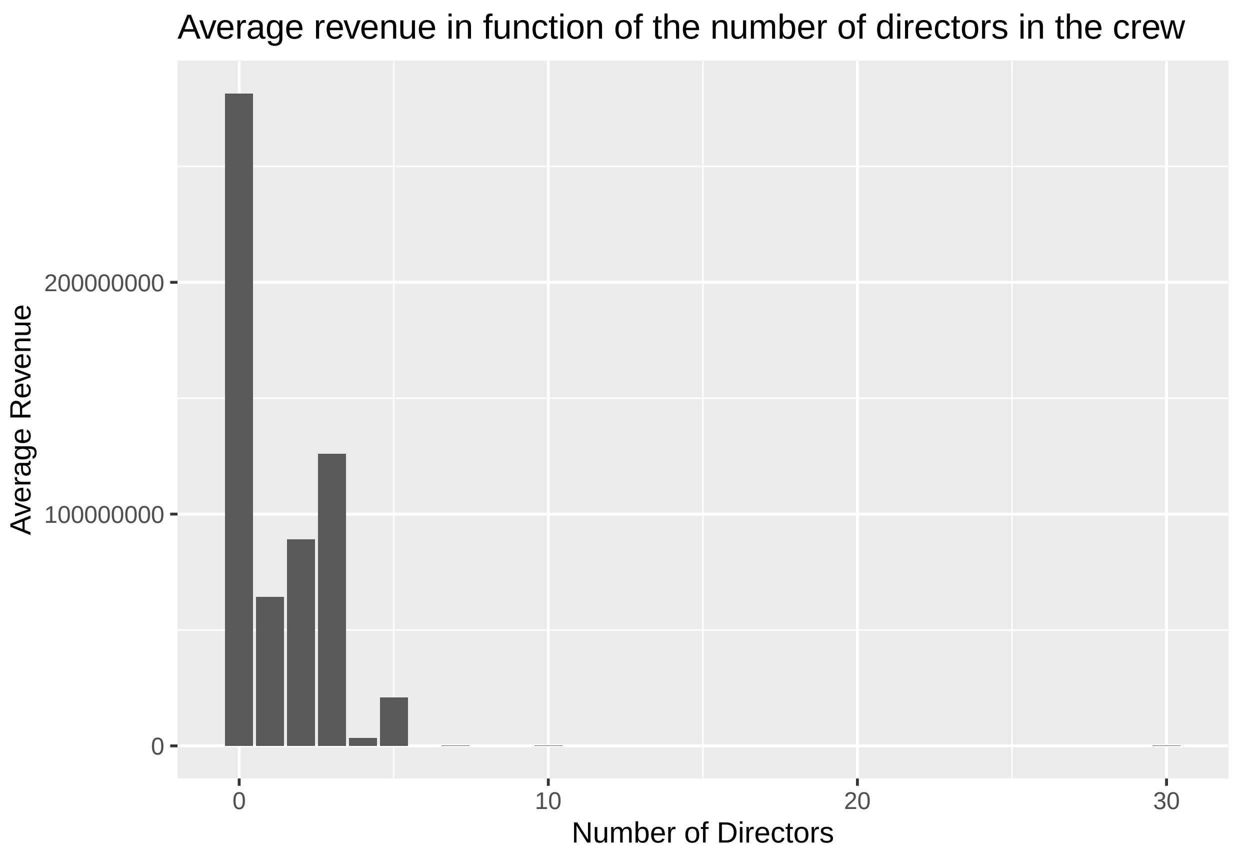
```
train$number_of_crew_members <- unlist(lapply(train$crew, CountJSONArrayInFeature))
train$crew <- NULL

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 directors.

```r
train %>%
    group_by(number_of_crew_members) %>%
    summarise(mean_revenue = mean(revenue)) %>%
    ggplot(aes(x = number_of_crew_members, y = mean_revenue)) +
          geom_bar(stat = "identity") +
          ggtitle("Average revenue in function of the number of members in the crew") +
          labs(x = "Number of Members", y = "Average Revenue")
```



```r
train %>%
    group_by(number_of_directors) %>%
    summarise(mean_revenue = mean(revenue)) %>%
    ggplot(aes(x = number_of_directors, y = mean_revenue)) +
          geom_bar(stat = "identity") +
          ggtitle("Average revenue in function of the number of directors in the crew") +
          labs(x = "Number of Directors", y = "Average Revenue")
```



According to this bar chart, having directors in the crew have a significant negative impact on the movie revenues compared to crew having no directors. However, without knowing the number of movies per number of directors, we cannot know the reasons behind this.

```
directors <- train %>%
    group_by(number_of_directors) %>%
    summarise(mean_revenue = mean(revenue),
              number_of_movies = n())
print(directors)
```

```
# A tibble: 9 x 3
  number_of_directors mean_revenue number_of_movies
                <dbl>        <dbl>            <int>
1                   0   281448176.               16
2                   1    64250338.             2815
3                   2    89000934.              146
4                   3   125950718.               14
5                   4     3373166                 3
6                   5    20872362.                2
7                   7      132416.                2
8                  10      100345                 1
9                  30        7171                 1
```

```
train$number_of_directors <- NULL
test$number_of_directors <- NULL
```

The number of movies is negligible for crews having more than 3 directors. We see that only 16 movies with crews having no directors have been made but generated big average revenues 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 no directors in the crew implies bigger revenues. Furthermore, most of movies (93.8333333% of them) have been made with one director in the crew.

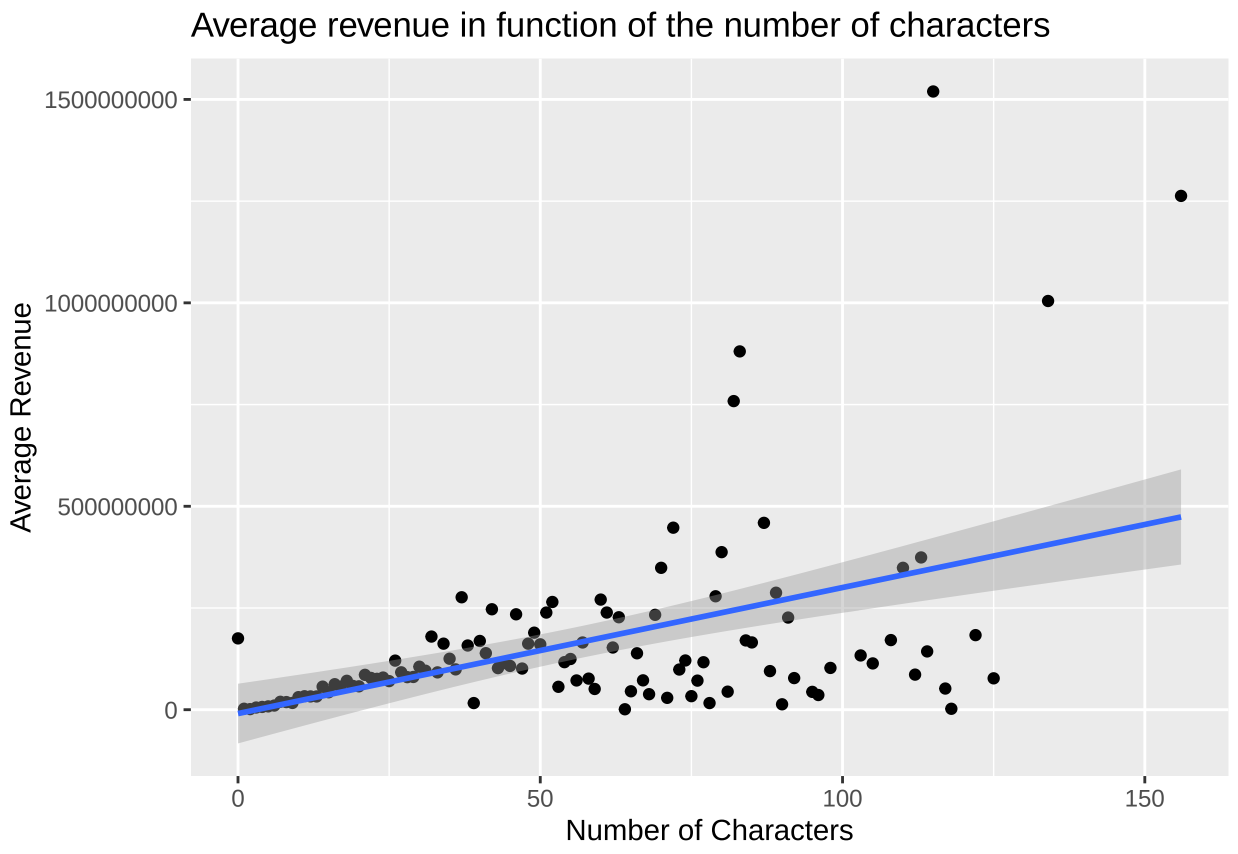
4.4 Movie Casting

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 revenue in function of the number of characters playing in the movie.

```
train %>%
    group_by(number_of_characters) %>%
    summarise(mean = mean(revenue)) %>%
    ggplot(aes(x = number_of_characters, y = mean)) +
           geom_point() +
           geom_smooth(method = lm) +
           ggtitle("Average revenue in function of the number of characters") +
           labs(x = "Number of Characters", y = "Average Revenue")
```



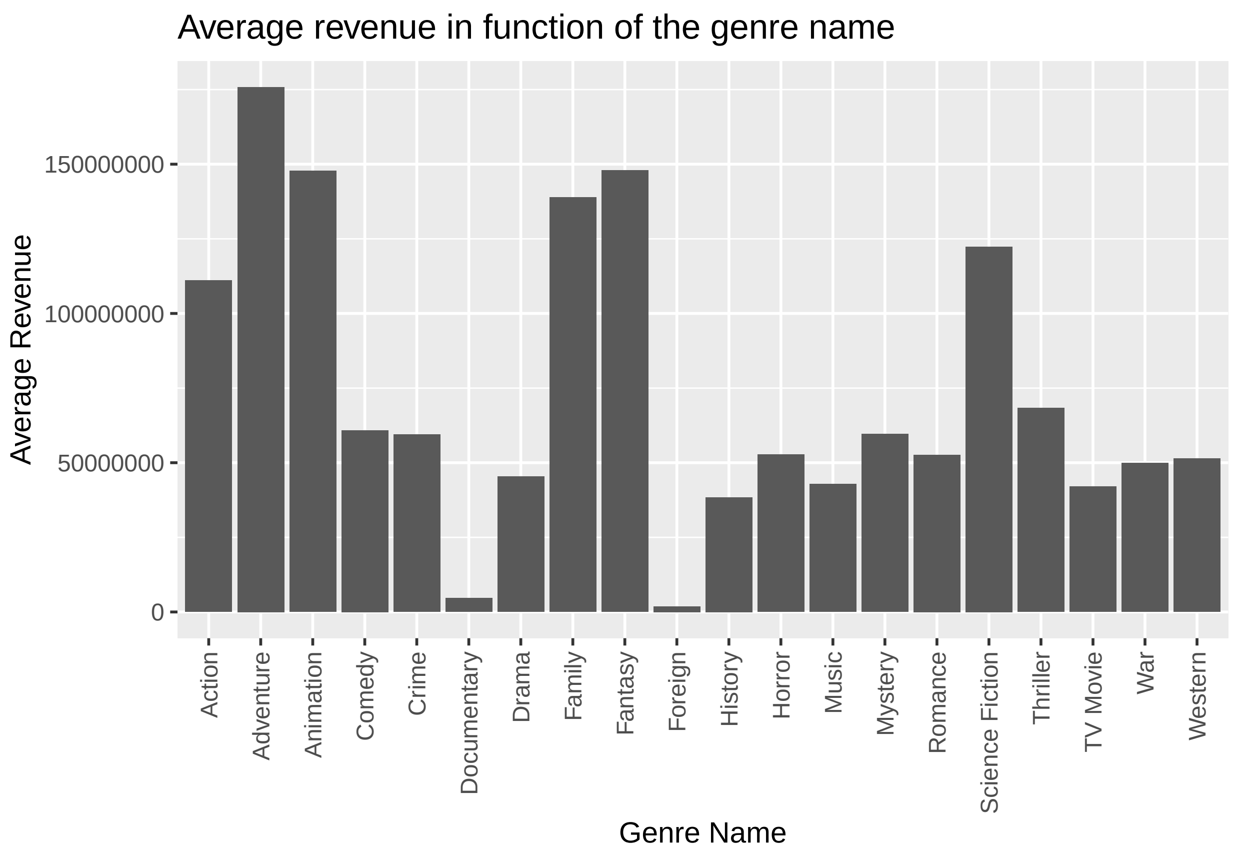
We have to consider the budget allowed because it is possible that many actors play in a low budget movie. These actors are paid with a lower salary or they are inexperienced which could justify their low salary. Another reason could be that many of these actors are playing a very short among of time in the movie. This may explain why sometimes with many characters the revenue is still low.

4.5 Movie Genres

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

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

```r
## Steps 1 and 2: Append the data extracted.
movies.genres <- AppendExtractedData(train, "genres", subfeatures.to_exclude = "id")

## Step 3: Grouping the data and getting the means.
movies.genres <- movies.genres %>%
    group_by(name) %>%
    summarise(mean_revenue = mean(revenue))
movies.genres$id <- seq(1, nrow(movies.genres))

## Step 4: Bar chart of the average revenue based on the genre.
movies.genres %>%
    ggplot(aes(x = name, y = mean_revenue)) +
          geom_bar(stat = "identity") +
          ggtitle("Average revenue in function of the genre name") +
          labs(x = "Genre Name", y = "Average Revenue") +
          theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))
```



According to these bar charts, both of them show that the following genres generate the highest revenue:

- Adventure
- Science-Fiction
- Fantasy
- Action
- Family
- Animation

And the ones generating the lowest revenue:

- Foreign
- Documentary
- TV Movies
- History

```
## Step 5: Keep the genre with the best revenue average if a movie is associated to many genres.
print(movies.genres)
```

```
# A tibble: 20 x 3
   name            mean_revenue    id
   <chr>                  <dbl> <int>
 1 Action           111043447.     1
 2 Adventure        175809499.     2
 3 Animation        147734092.     3
 4 Comedy            60875743.     4
 5 Crime             59491277.     5
 6 Documentary        4638009.     6
 7 Drama             45406074.     7
 8 Family           138897795.     8
 9 Fantasy           147965933.     9
10 Foreign             1874198.    10
11 History            38413232.    11
12 Horror             52709071.    12
13 Music              42870898.    13
14 Mystery            59633964.    14
15 Romance            52705007.    15
16 Science Fiction   122367176.    16
17 Thriller           68336641.    17
18 TV Movie           42000000     18
19 War                49915868.    19
20 Western            51370635.    20
```

```r
train$best_genre <- ExtractBestRevenueFromGenres(train, movies.genres)
train$genres <- NULL
head(train$best_genre, 10)
```

```
 [1]  4  8  7 17  1  2 17  6  2  4
```

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

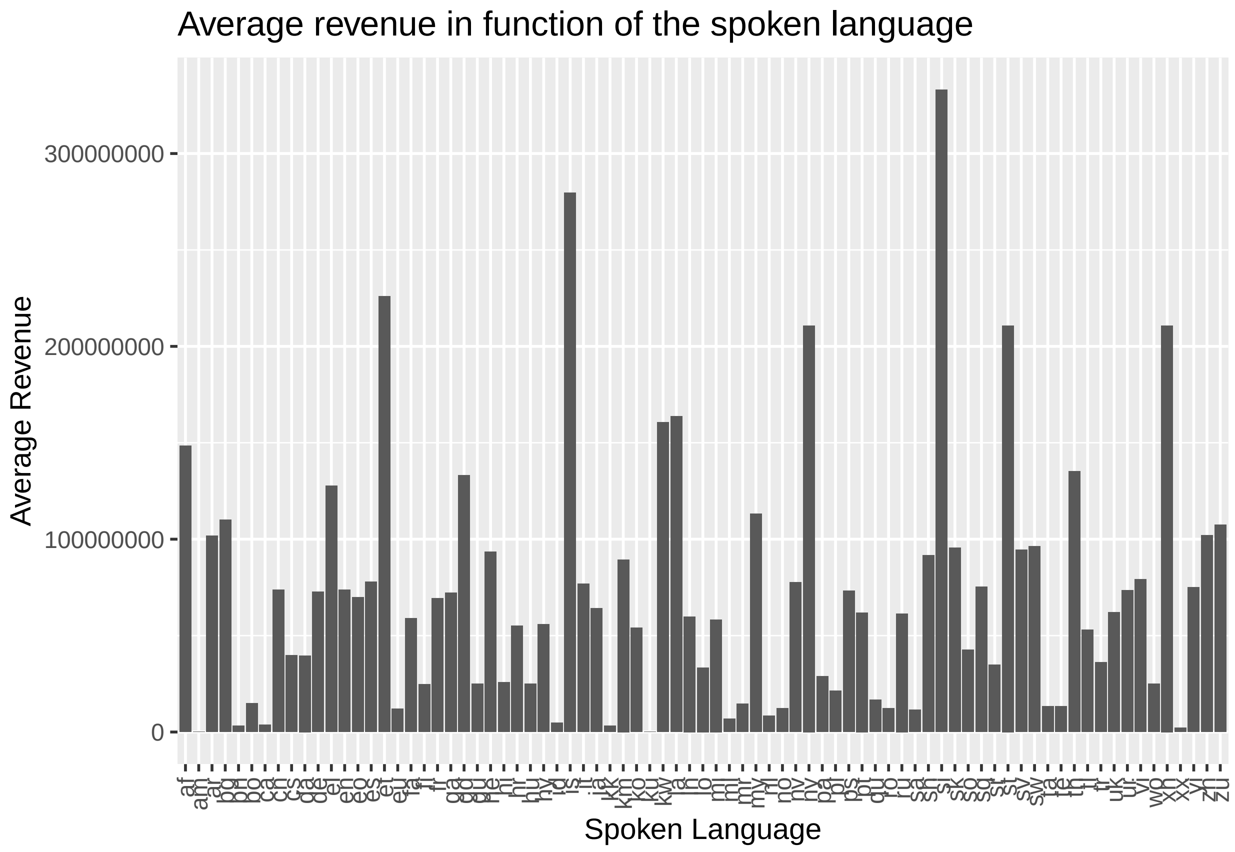
4.6 Spoken Languages

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.

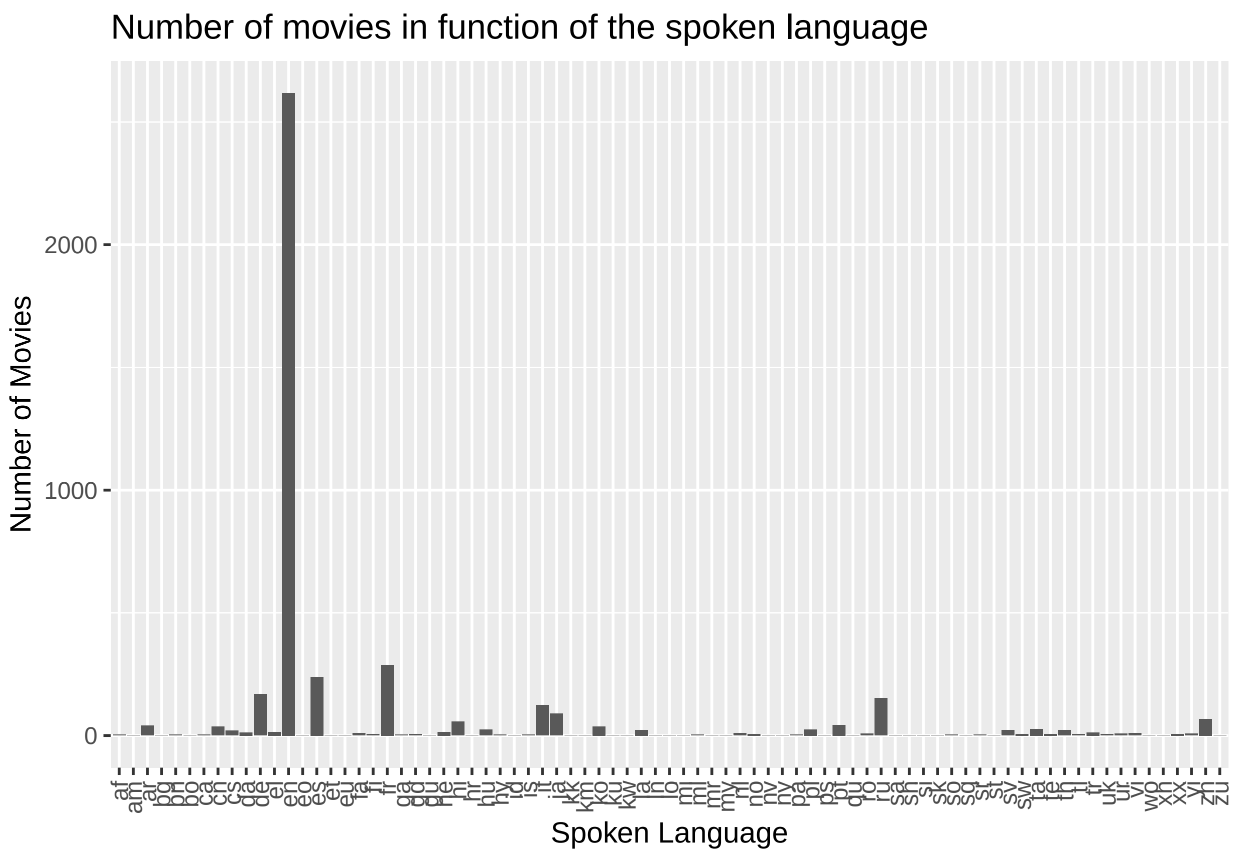
The objective is to visualize the average revenue for every spoken language. Here are the steps to achieve the objective:

1. Extract in a data frame the spoken language IDs `iso_639_1` for every movie in the dataset with the movie ID, popularity and 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 revenue.
4. Show a bar chart of the average revenue in function of the spoken language.
5. Show a bar chart of the number of movies in function of the spoken language.

```r
## Steps 1 and 2: Append the extracted data.
movies.spoken_languages <- AppendExtractedData(train, "spoken_languages", subfeatures.to_exclude = "name

## Step 3: Grouping the data and getting the mean and number of movies.
movies.spoken_languages <- movies.spoken_languages %>%
    group_by(iso_639_1) %>%
    summarise(mean_revenue = mean(revenue),
              number_of_movies = n())

## Step 4: Bar chart of the average revenue based on the spoken language.
movies.spoken_languages %>%
    ggplot(aes(x = iso_639_1, y = mean_revenue)) +
          geom_bar(stat = "identity") +
          ggtitle("Average revenue in function of the spoken language") +
          labs(x = "Spoken Language", y = "Average Revenue") +
          theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))
```



```r
## Step 5: Bar chart of the number of movies based on the spoken language.
movies.spoken_languages %>%
    ggplot(aes(x = iso_639_1, y = number_of_movies)) +
        geom_bar(stat = "identity") +
        ggtitle("Number of movies in function of the spoken language") +
        labs(x = "Spoken Language", y = "Number of Movies") +
        theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))
```



```
movies.spoken_languages <- NULL
train$spoken_languages <- NULL
test$spoken_languages <- NULL
```

We see that the English spoken language represents % of the movies in this dataset. For the reasons we mentioned, we will not consider this feature to help calculating the predictions of the revenue.

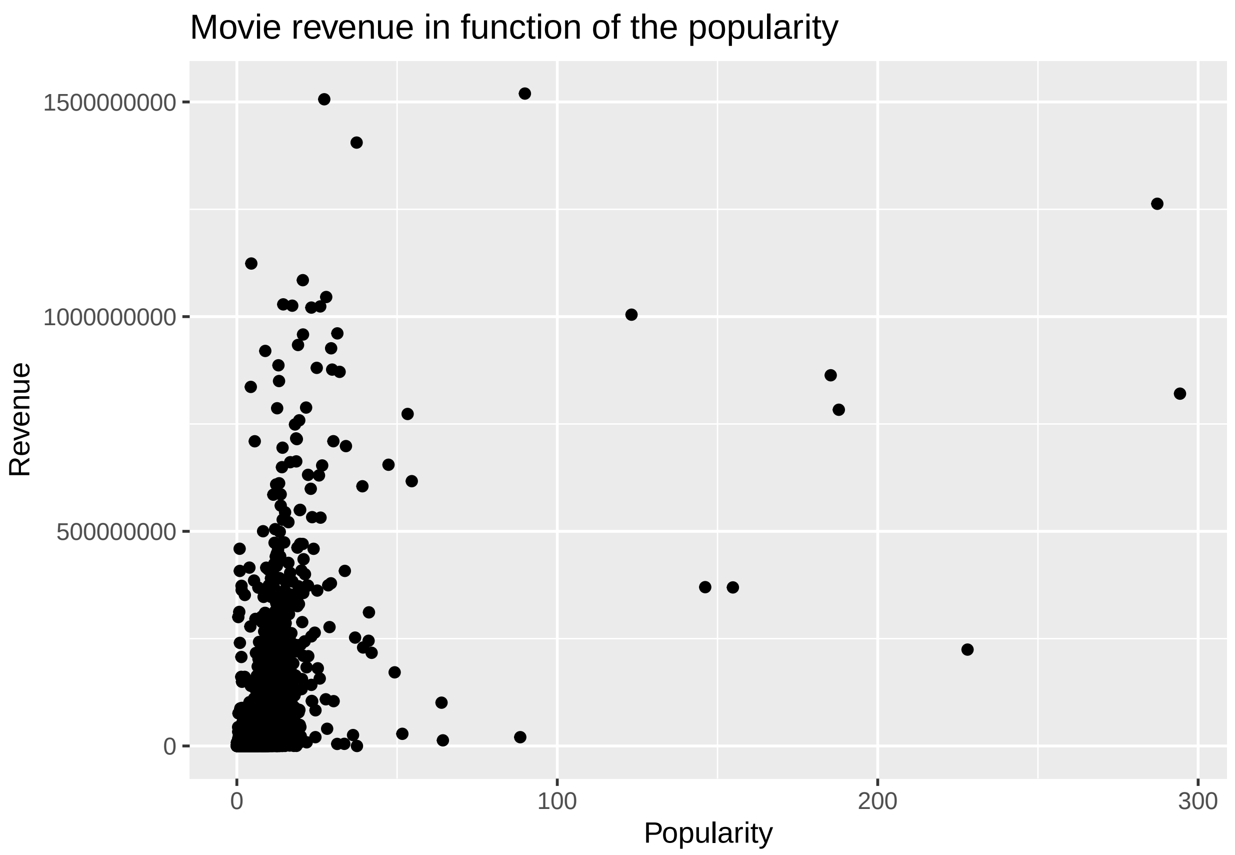
4.7 Movie Popularity

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 revenue in function of the popularity.

```
train %>%
    ggplot(aes(x = popularity, y = revenue)) +
           geom_point() +
           ggtitle("Movie revenue in function of the popularity") +
           labs(x = "Popularity", y = "Revenue")
```



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 revenue and their popularity.

```
head(train[order(train$revenue, decreasing = TRUE), c("title", "popularity", "revenue")], n = 10)
```

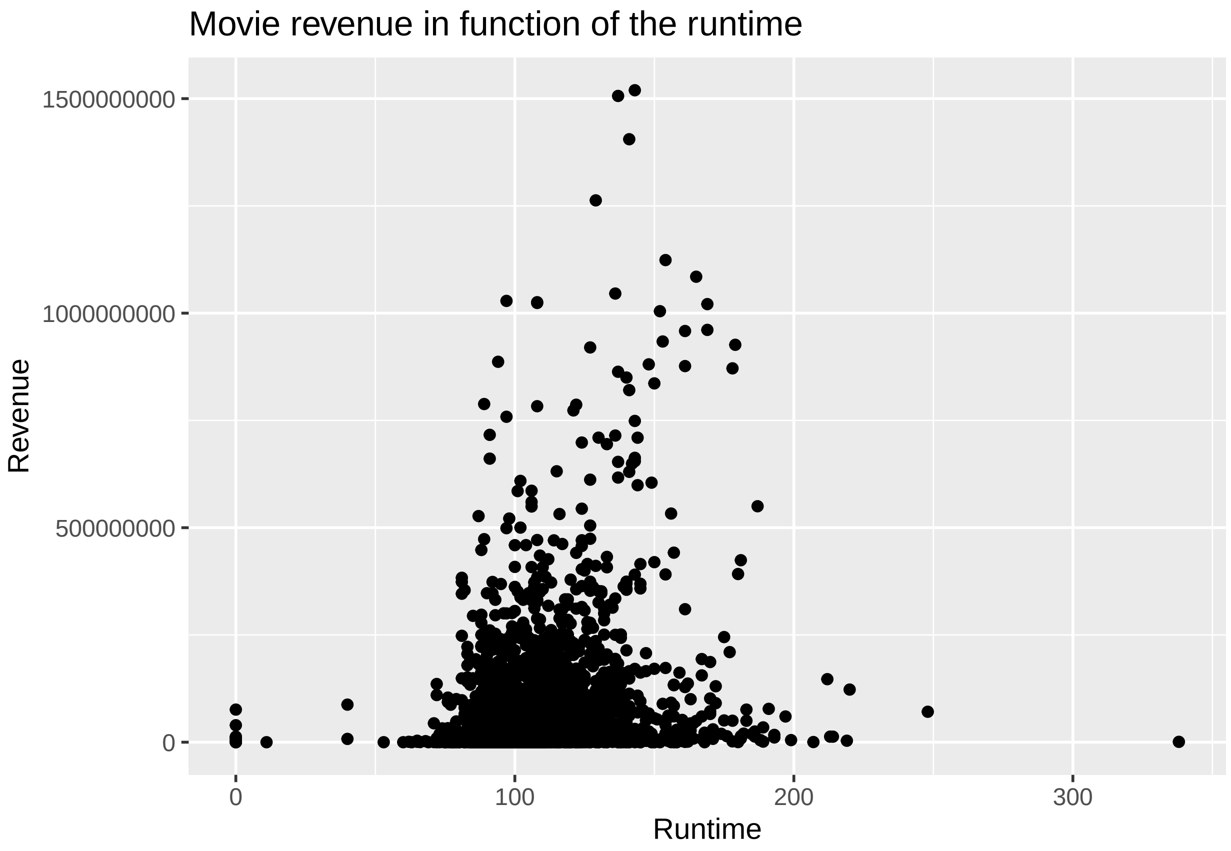
```
                                      title popularity    revenue
1127                           The Avengers  89.887648 1519557910
1762                              Furious 7  27.275687 1506249360
2771                Avengers: Age of Ultron  37.379420 1405403694
685                    Beauty and the Beast 287.253654 1262886337
2323           Transformers: Dark of the Moon   4.503505 1123746996
907                   The Dark Knight Rises  20.582580 1084939099
2136 Pirates of the Caribbean: On Stranger Tides  27.887720 1045713802
2563                           Finding Dory  14.477677 1028570889
882                       Alice in Wonderland  17.285093 1025491110
735                                Zootopia  26.024868 1023784195
```

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 1262886337 (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 1123746996 (ranked 5th among the 3000 movies) and has a low popularity of 4.503505.

4.8 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.

```
train %>%
    ggplot(aes(x = runtime, y = revenue)) +
           geom_point() +
           ggtitle("Movie revenue in function of the runtime") +
           labs(x = "Runtime", y = "Revenue")
```

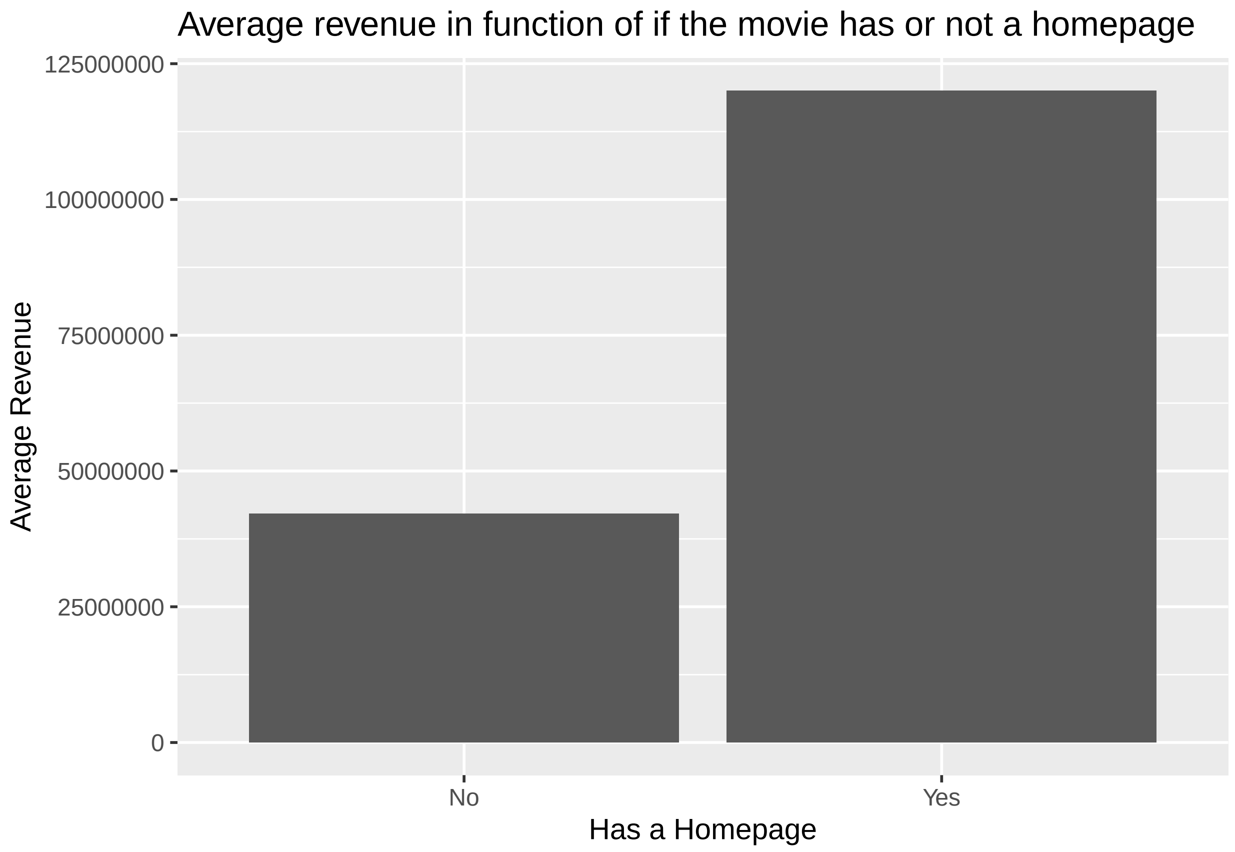


According to this scatter plot, runtimes between 90 minutes and 180 minutes obtain better revenue. Out of that range, the revenue is lower.

4.9 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.

```
train$has_homepage <- unlist(lapply(train$homepage, function(homepage) ifelse(homepage == "", 0, 1)))
train$homepage <- NULL

test$has_homepage <- unlist(lapply(test$homepage, function(homepage) ifelse(homepage == "", 0, 1)))
test$homepage <- NULL

train %>%
    group_by(has_homepage) %>%
    summarise(mean_revenue = mean(revenue)) %>%
    ggplot(aes(x = factor(has_homepage), y = mean_revenue)) +
          geom_bar(stat = "identity") +
          ggtitle("Average revenue in function of if the movie has or not a homepage") +
          labs(x = "Has a Homepage", y = "Average Revenue") +
          scale_x_discrete(labels = c("No", "Yes"))
```



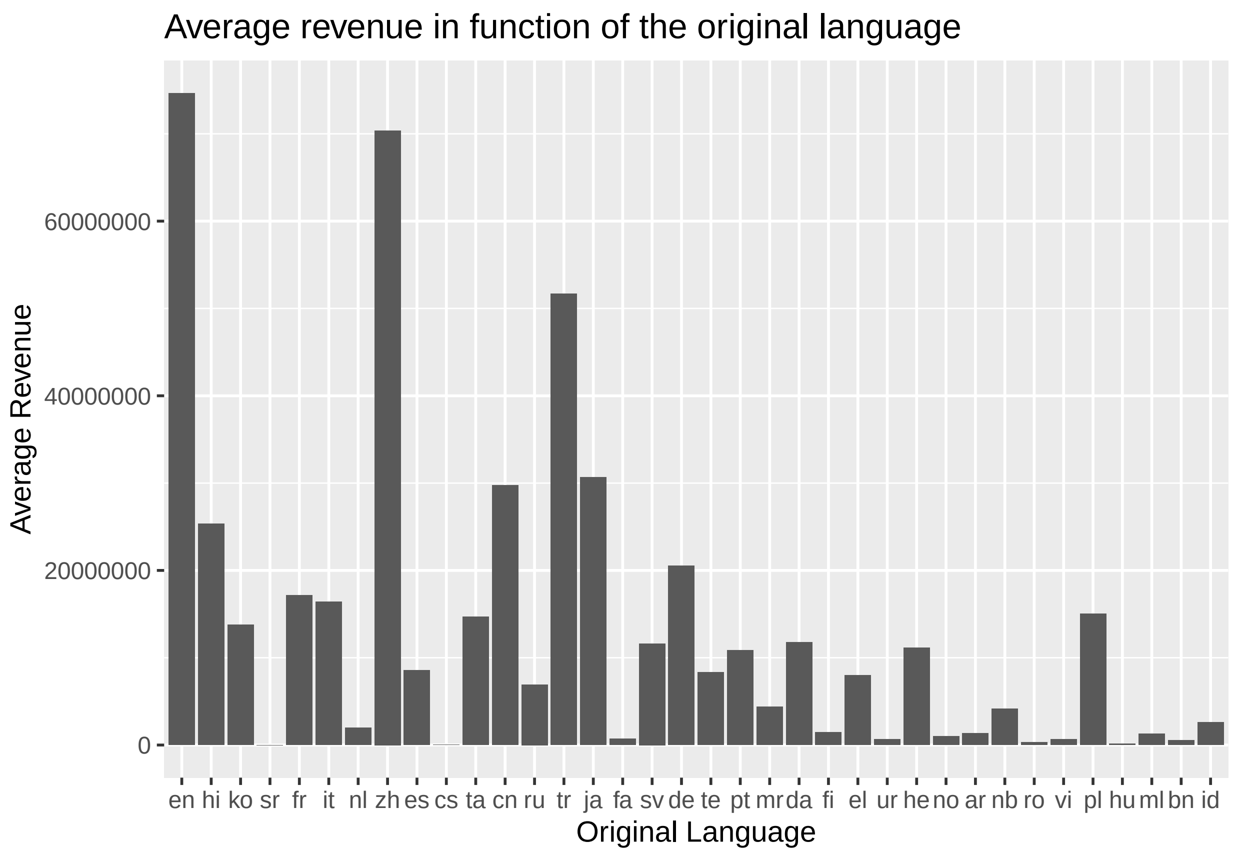
According to this bar chart, the revenue is considerably greater for movies having a homepage than the ones that do not.

4.10 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.

```
train.original_languages.names <- unique(train$original_language)
train.original_languages <- data.frame(id = seq(1, length(train.original_languages.names)),
                                       name = train.original_languages.names)
train$original_language_id <- unlist(lapply(train$original_language, ExtractOriginalLanguageIDFromName,
test.original_languages.names <- unique(test$original_language)
test.original_languages <- data.frame(id = seq(1, length(test.original_languages.names)),
                                      name = test.original_languages.names)
test$original_language_id <- unlist(lapply(test$original_language, ExtractOriginalLanguageIDFromName, o

train %>%
    group_by(original_language_id) %>%
    summarise(mean_revenue = mean(revenue)) %>%
    ggplot(aes(x = factor(original_language_id), y = mean_revenue)) +
          geom_bar(stat = "identity") +
          ggtitle("Average revenue in function of the original language") +
          labs(x = "Original Language", y = "Average Revenue") +
          scale_x_discrete(labels = train.original_languages$name)
```



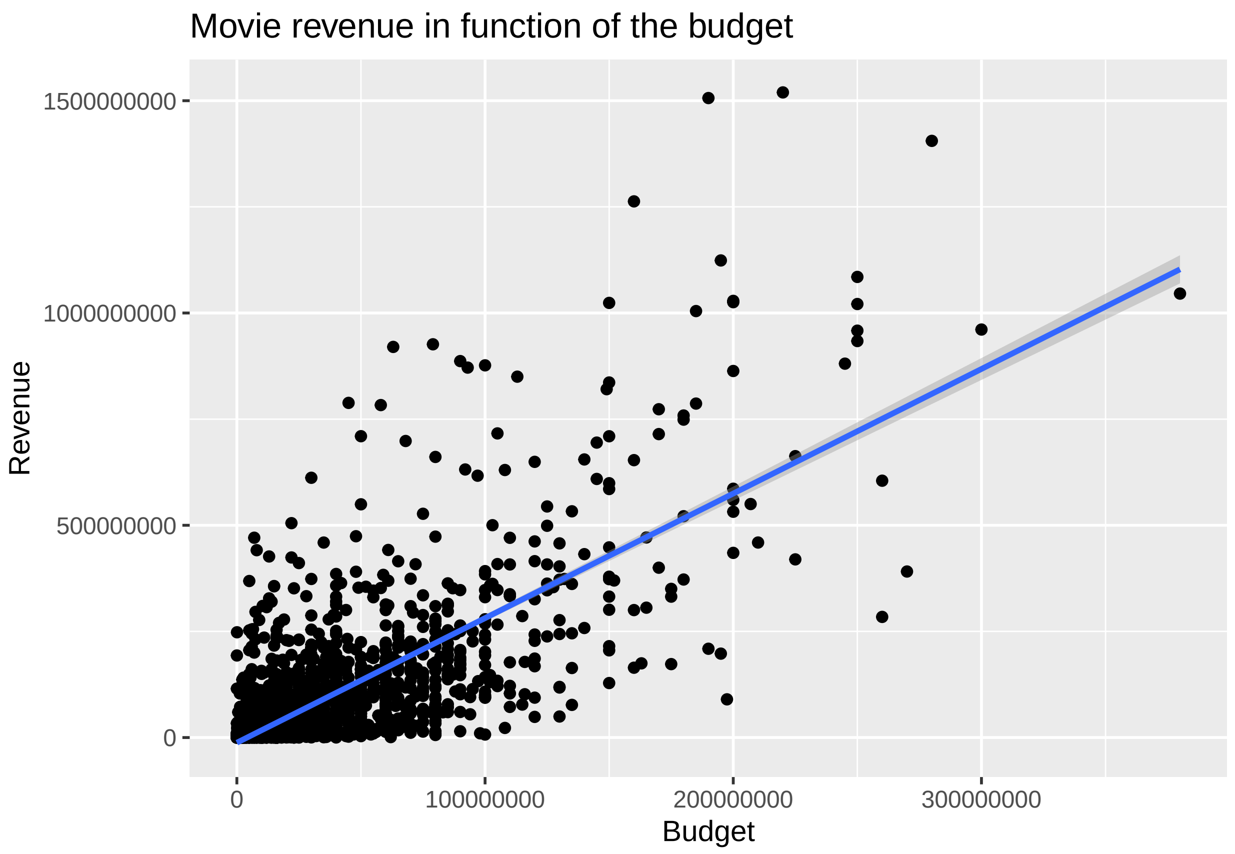
```
original_languages <- NULL
original_languages.names <- NULL
train$original_language <- NULL
test$original_language <- NULL
```

As we expected, English and Chinese movies have generated the best average revenue.

4.11 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.

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

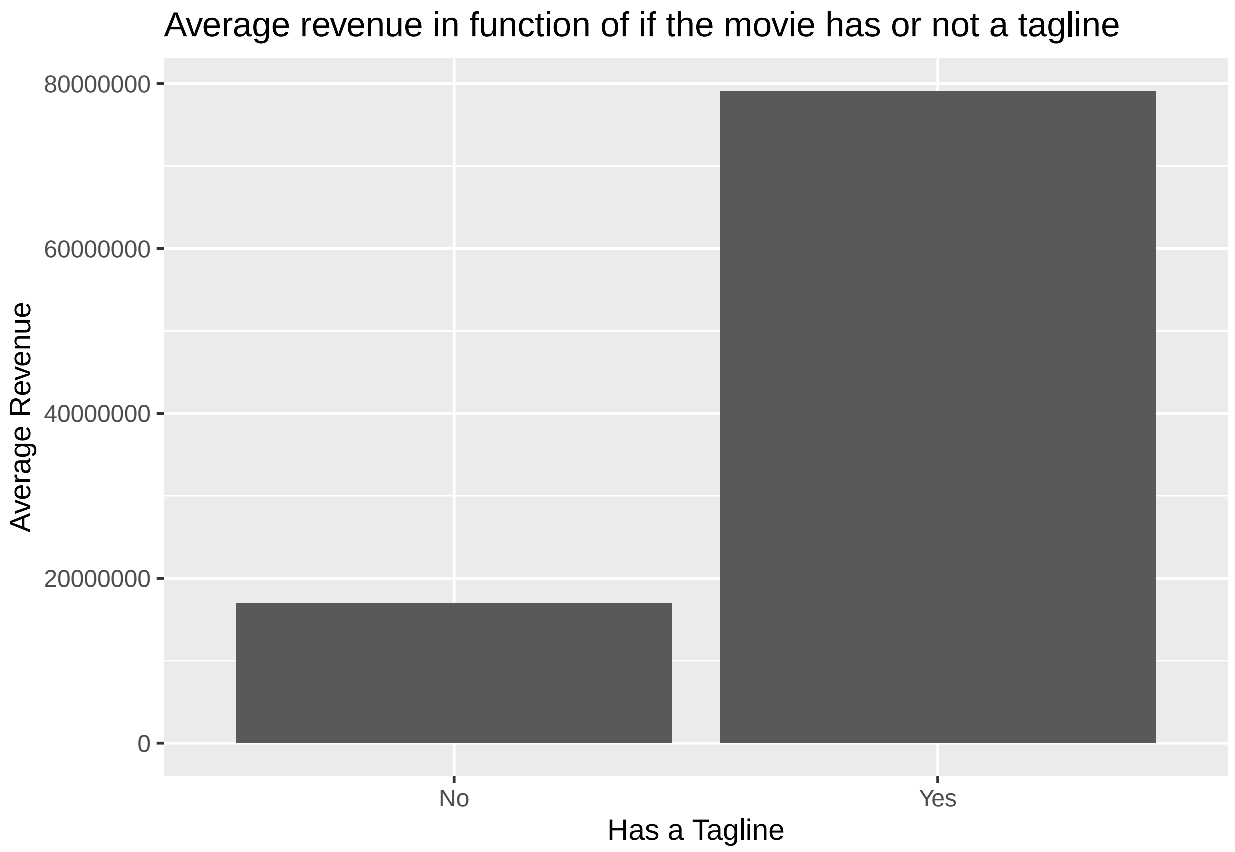


Our assumptions holds most of the time and the scatter plot describes a linear increasing of the revenue in function of the budget. Therefore, the budget matters and has impact on the movie revenue.

4.12 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.

```r
train$has_tagline <- unlist(lapply(train$tagline, function(tagline) ifelse(tagline == "", 0, 1)))
train$tagline <- NULL

test$has_tagline <- unlist(lapply(test$tagline, function(tagline) ifelse(tagline == "", 0, 1)))
test$tagline <- NULL

train %>%
    group_by(has_tagline) %>%
    summarise(mean_revenue = mean(revenue)) %>%
    ggplot(aes(x = factor(has_tagline), y = mean_revenue)) +
          geom_bar(stat = "identity") +
          ggtitle("Average revenue in function of if the movie has or not a tagline") +
          labs(x = "Has a Tagline", y = "Average Revenue") +
          scale_x_discrete(labels = c("No", "Yes"))
```



According to this bar chart, our assumption holds.

4.13 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.

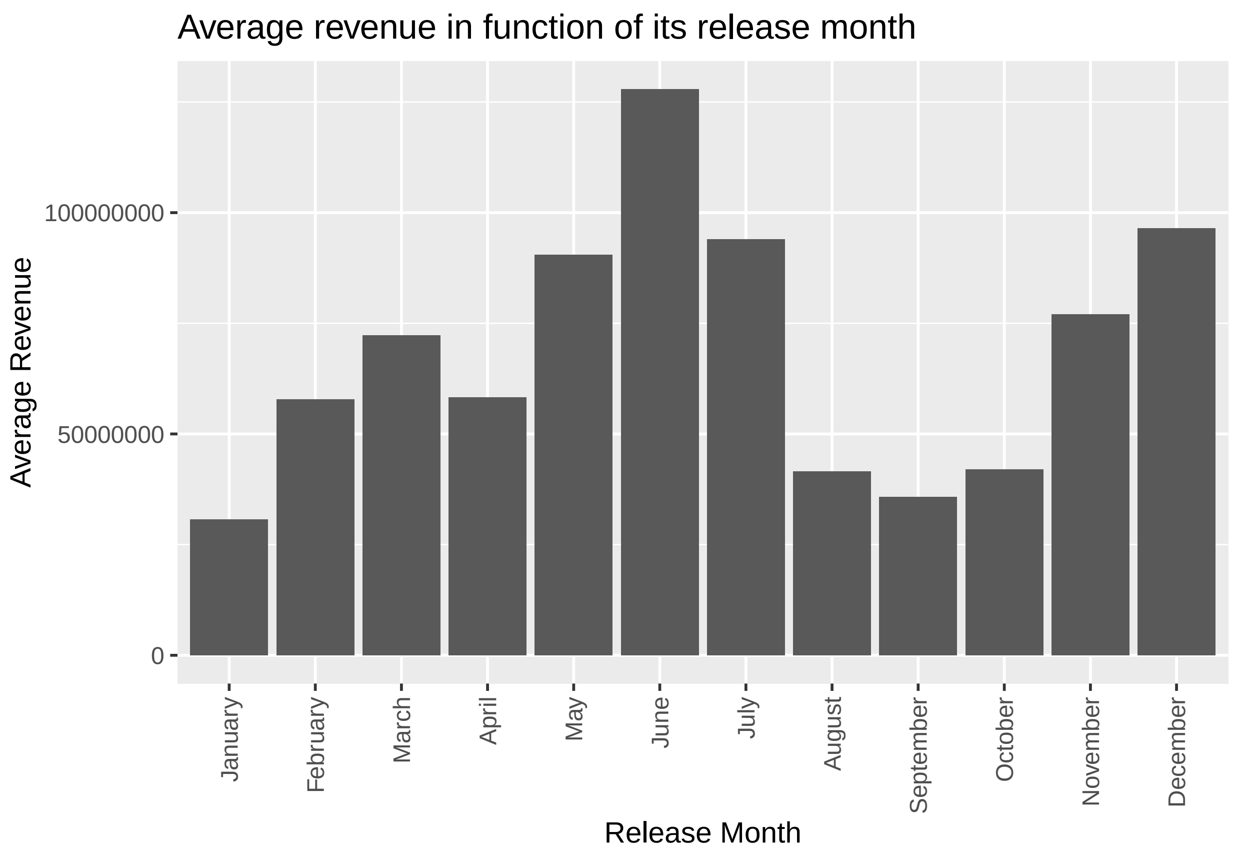
We assume also that a movie released on the weekend (mainly Friday or Saturday) generates more revenue because people are generally not working the next day. They will see the movie in the evening but we cannot check that assumption because the time is not provided in this dataset.

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.

```r
## 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")
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$release_date <- (year(train$release_date) * 366) + (train$release_month * 31) + day(train$release_

#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$release_date <- (year(test$release_date) * 366) + (test$release_month * 31) + day(test$release_date

## Step 3: Show bar charts of the revenue in function of the release year, release month and release we
# train %>%
#     group_by(release_year) %>%
#     summarise(mean_revenue = mean(revenue)) %>%
#     ggplot(aes(x = release_year, y = mean_revenue)) +
#           geom_bar(stat = "identity") +
#           ggtitle("Average revenue in function of its release year") +
#           labs(x = "Release Year", y = "Average Revenue") +
#           theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))

train %>%
    group_by(release_month) %>%
    summarise(mean_revenue = mean(revenue)) %>%
    ggplot(aes(x = factor(release_month), y = mean_revenue)) +
          geom_bar(stat = "identity") +
          ggtitle("Average revenue in function of its release month") +
          labs(x = "Release Month", y = "Average Revenue") +
          scale_x_discrete(labels = c("January", "February", "March", "April", "May", "June", "July", "
          theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))
```



```
# train %>%
#     group_by(release_weekday) %>%
#     summarise(mean_revenue = mean(revenue)) %>%
#     ggplot(aes(x = factor(release_weekday), y = mean_revenue)) +
#           geom_bar(stat = "identity") +
#           ggtitle("Average revenue in function of its release weekday") +
#           labs(x = "Release Weekday", y = "Average Revenue") +
#           scale_x_discrete(labels = c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday
#           theme(axis.text.x = element_text(angle = 90, hjust = 1, vjust = 0.5))
```

According to these bar charts, we see that:

- Generally, more recent are the released movies, better are the revenue specially the last year (2017).
- 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.

4.14 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:

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

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 $\bar{b} \geq 0$ is a real number.
- n_t is the number of movies in the train dataset and $n_t > 0$ an integer.

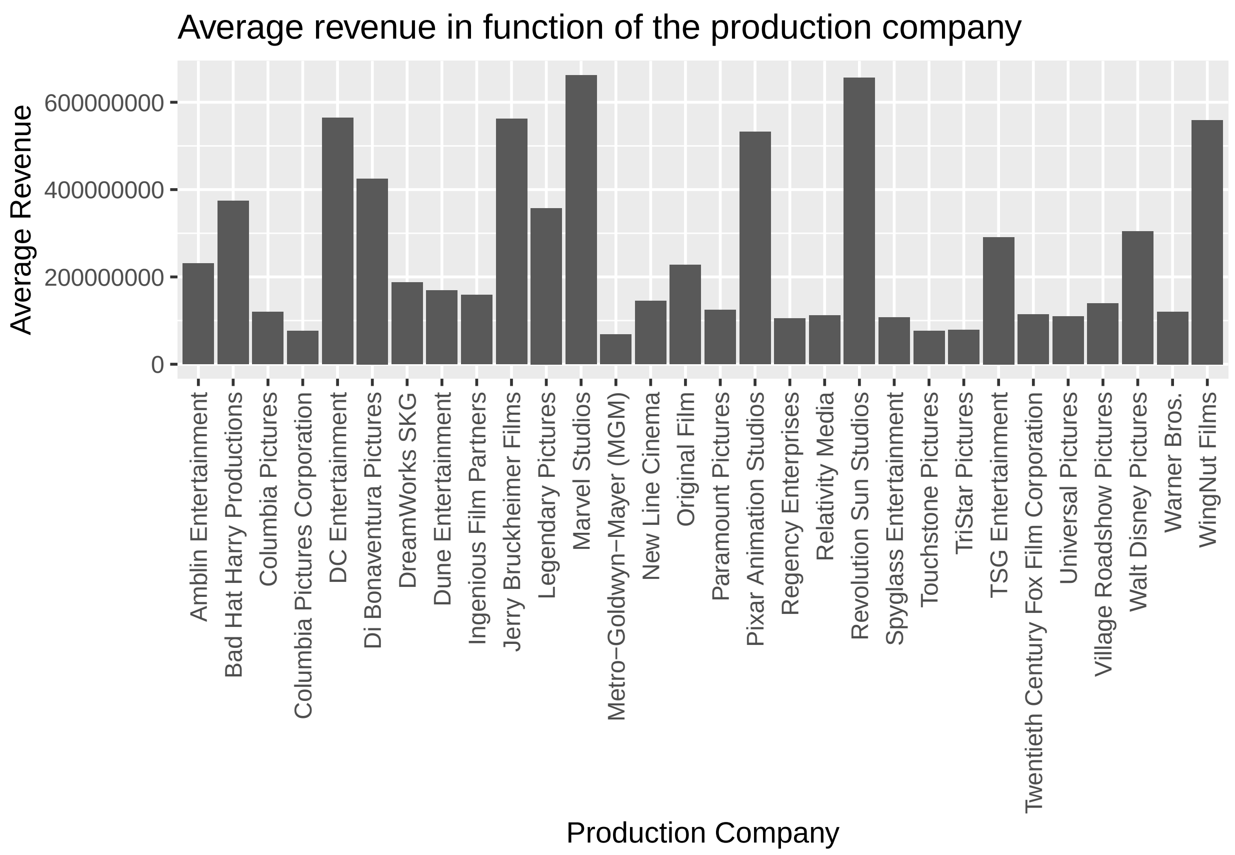
The following steps have to be done in order to achieve the objective:

1. Extract in a data frame the production company names for every movie in the dataset with the movie ID, 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.

```r
## 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.               120334674.   47085510.              202  3.17e6
 2 Universal Pictures         109670828.   37942995               188  2.38e6
 3 Paramount Pictures         124783064.   39026416.              161  2.09e6
 4 Walt Disney Pictures       303777083.   89950000                62  1.86e6
 5 Twentieth Century Fox~     113961916.   39814058.              138  1.83e6
 6 Columbia Pictures          120284417.   49960769.               91  1.52e6
 7 New Line Cinema            145159176.   43953333.               75  1.10e6
 8 Metro-Goldwyn-Mayer (~    68721456.   32192961.              84  9.01e5
 9 Relativity Media         111337606.   50994722.              48  8.16e5
10 Village Roadshow Pict~   139645381.   66666667.              36  8.00e5
11 Legendary Pictures       357564820.  125000000               19  7.92e5
12 Touchstone Pictures       76546547.   33245786.              63  6.98e5
13 Columbia Pictures Cor~    75911345.   33124592.              61  6.74e5
14 Dune Entertainment       168699729.   58166667.              30  5.82e5
15 Original Film            227934116.   79357143.              21  5.55e5
16 DreamWorks SKG           187110073.   60370370.              27  5.43e5
17 TriStar Pictures          78391982.   28950943.              53  5.11e5
18 Regency Enterprises      105011750.   42419355.              31  4.38e5
19 Revolution Sun Studios   655469383.  162000000                8  4.32e5
20 Jerry Bruckheimer Fil~   561781952.  184285714.               7  4.30e5
21 Amblin Entertainment     230780010.   55304348.              23  4.24e5
22 Pixar Animation Studi~   532351328.  146500000                8  3.91e5
23 WingNut Films            559200424   126000000                9  3.78e5
24 Di Bonaventura Pictur~   425142483.  125555556.               9  3.77e5
25 Marvel Studios           661596618.  136000000                8  3.63e5
# ... with 3,670 more rows
```

```r
## 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 = mean_revenue)) +
           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

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.
- $x_i(x_{i,1}, x_{i,2}, \ldots, x_{i,n_c})$ be the movie i in the dataset X where $x_i \in \mathbb{R}^{n_c}$.
- $y = (y_1, y_2, \ldots, y_{n_r})$ 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 $y = (y_1, y_2, \ldots, y_{3000}) = (12314651, 95149435, 13092000, \ldots, 82087155)$.

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

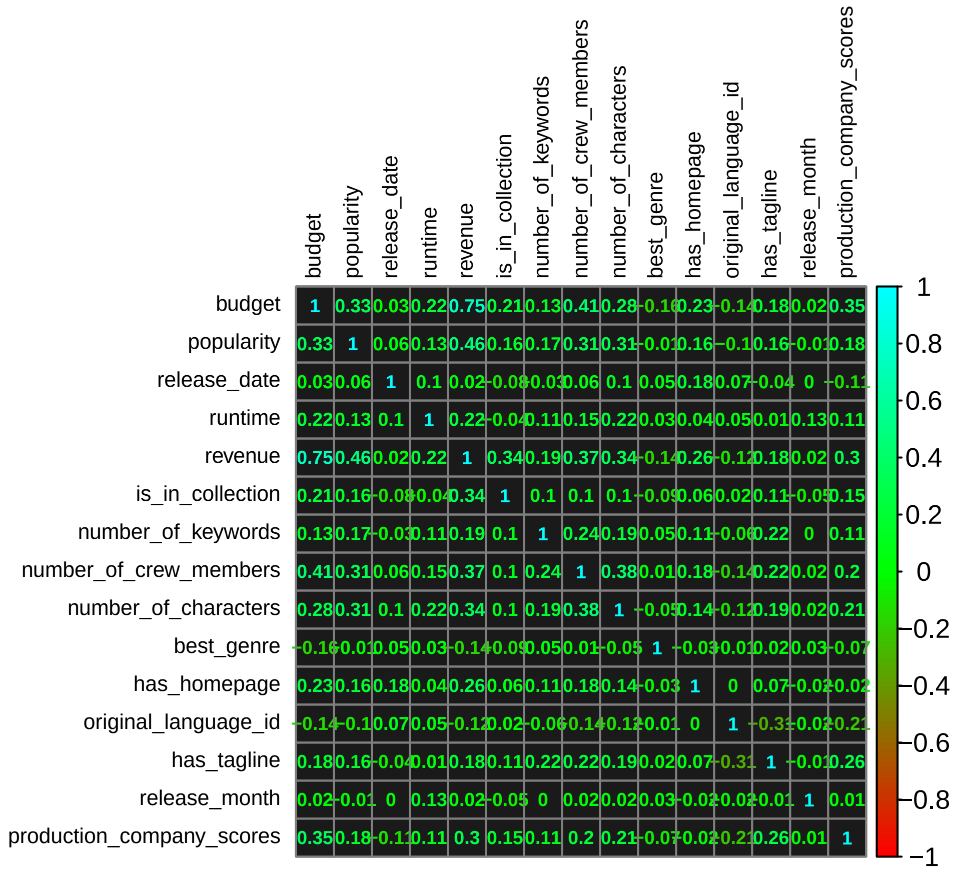
$$y_i - f(x_i) = \epsilon(x_i)$$

where $\epsilon(x_i)$ is the residuals function for $i = 1, 2, \ldots, 4398$. 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.

```
#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 $\beta_0, \beta_1 \in \mathbb{R}$.

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

```
0.006 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 
-15.4448  -0.8803   0.7271   1.6901  12.7212 

Coefficients:
             Estimate Std. Error t value             Pr(>|t|)    
(Intercept)   2.84788    0.47202   6.033         0.0000000018 ***
train$budget  0.80118    0.02868  27.939 < 0.0000000000000002 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.727 on 2998 degrees of freedom
Multiple R-squared:  0.2066,    Adjusted R-squared:  0.2063 
F-statistic: 780.6 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

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

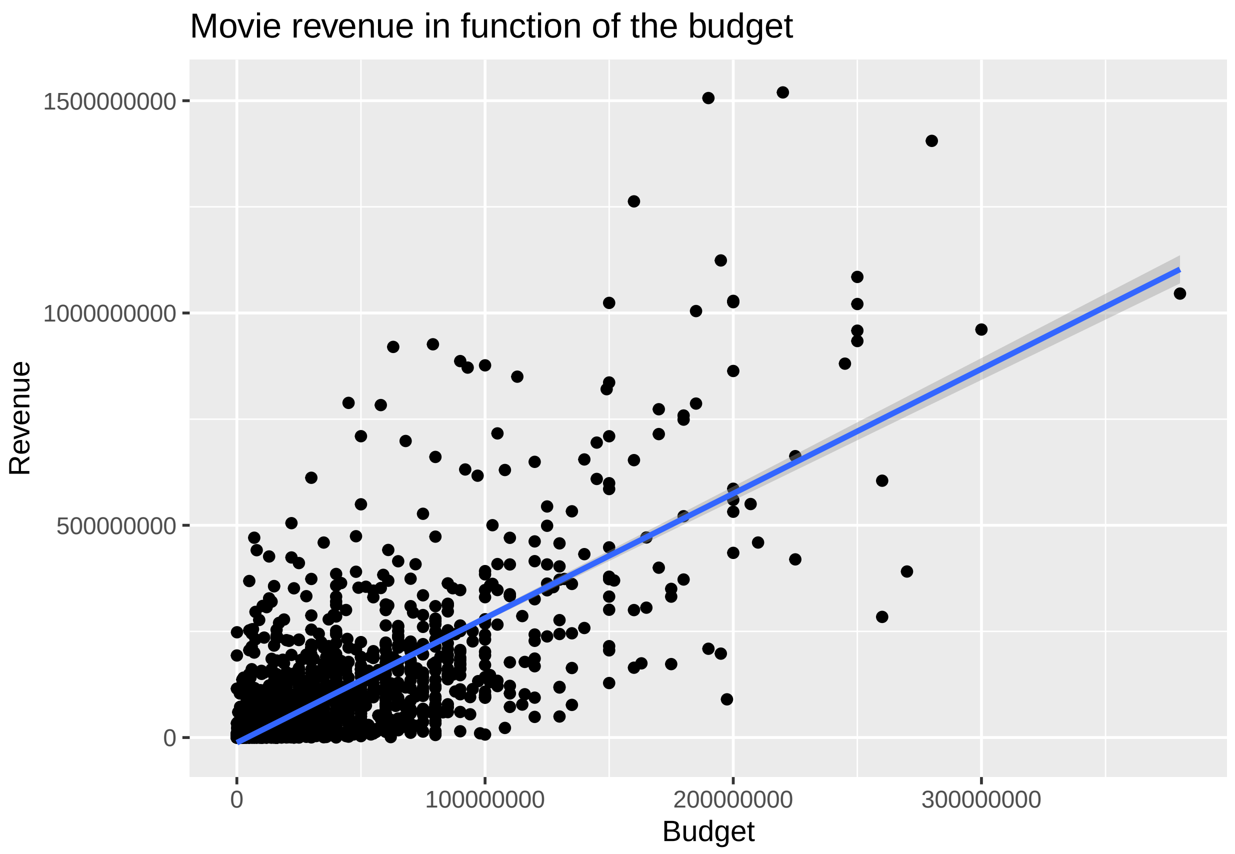
Then, we apply the RMSLE on our predicted revenues $f(X)$ and the revenues y given in the train dataset.

```
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 = 2.72638
```

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.

```
train %>%
    ggplot(aes(x = exp(budget) - 1, y = revenue)) +
           geom_point() +
           geom_smooth(method = lm) +
           ggtitle("Movie revenue in function of the budget") +
           labs(x = "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 (15), 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.

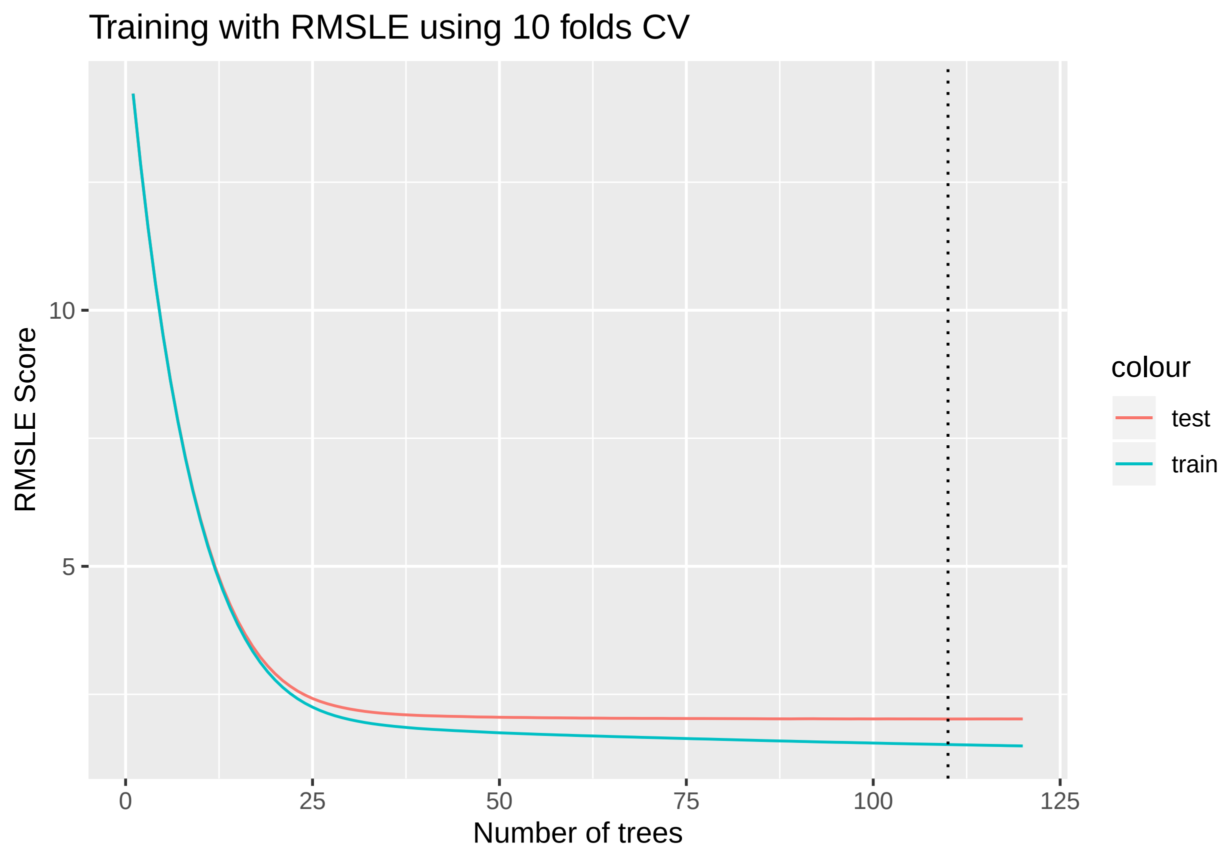
```
SquaredLogLossObjectiveFunction <- function(predicted, train.matrix)
{
    labels <- getinfo(train.matrix, "label")

    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.

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

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       14.230030    0.009018050      14.231037    0.09066081
       2       12.854186    0.008323018      12.857017    0.09073187
       3       11.618391    0.007330202      11.623132    0.08954598
       4       10.509175    0.006563397      10.518346    0.08792858
       5        9.514251    0.006034525       9.526588    0.08199384
---
     116        1.502754    0.019821673       2.019170    0.12518564
     117        1.500384    0.019902560       2.019608    0.12539785
     118        1.497042    0.019349160       2.019086    0.12499591
     119        1.494619    0.019973512       2.019227    0.12563962
     120        1.491915    0.019894072       2.018973    0.12617990
Best iteration:
 iter train_rmse_mean train_rmse_std test_rmse_mean test_rmse_std
  110        1.519427     0.01898371       2.017816     0.1241728
```

```
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()
```

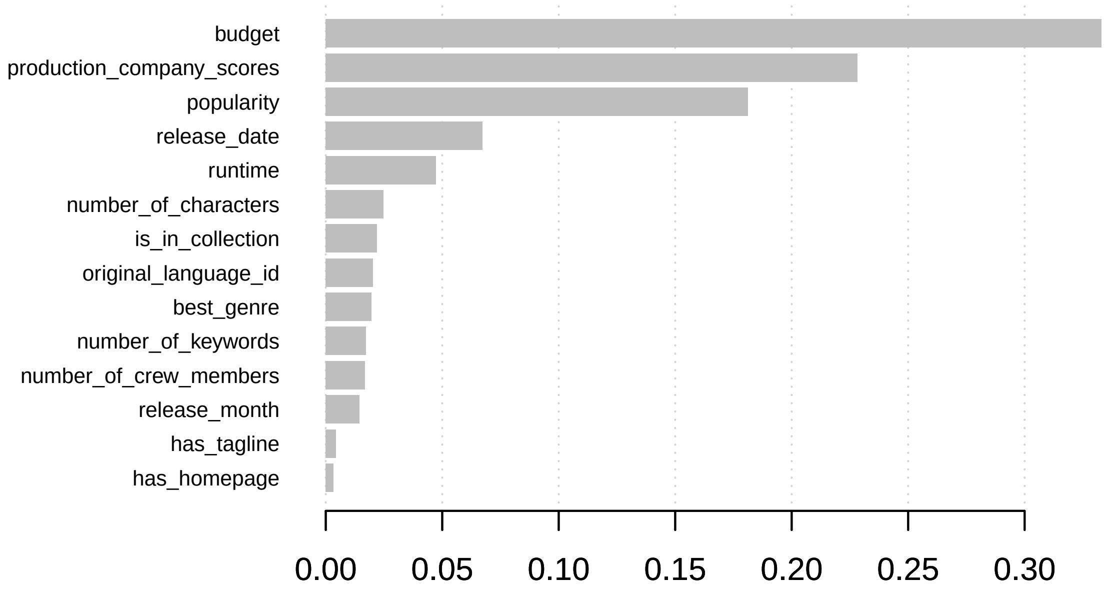
```
0.398 sec elapsed
```

```
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 0.332931552 0.155603892 0.164319249
 2: production_company_scores 0.228266989 0.100992058 0.107310530
 3:                popularity 0.181200712 0.103507953 0.138162307
 4:              release_date 0.067291800 0.162595122 0.146210597
 5:                   runtime 0.047320361 0.084874479 0.099932931
 6:      number_of_characters 0.024686638 0.059409112 0.058350101
 7:          is_in_collection 0.022075256 0.030383578 0.012743125
 8:      original_language_id 0.020214758 0.088736978 0.051643192
 9:                best_genre 0.019694788 0.057225783 0.053655265
10:        number_of_keywords 0.017253844 0.030815831 0.047619048
```

```
11:    number_of_crew_members 0.016912220 0.045580270 0.050972502
12:             release_month 0.014489699 0.056849093 0.049631120
13:               has_tagline 0.004342536 0.005584159 0.010060362
14:              has_homepage 0.003318846 0.017841691 0.009389671
```

```
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.

```
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)
```

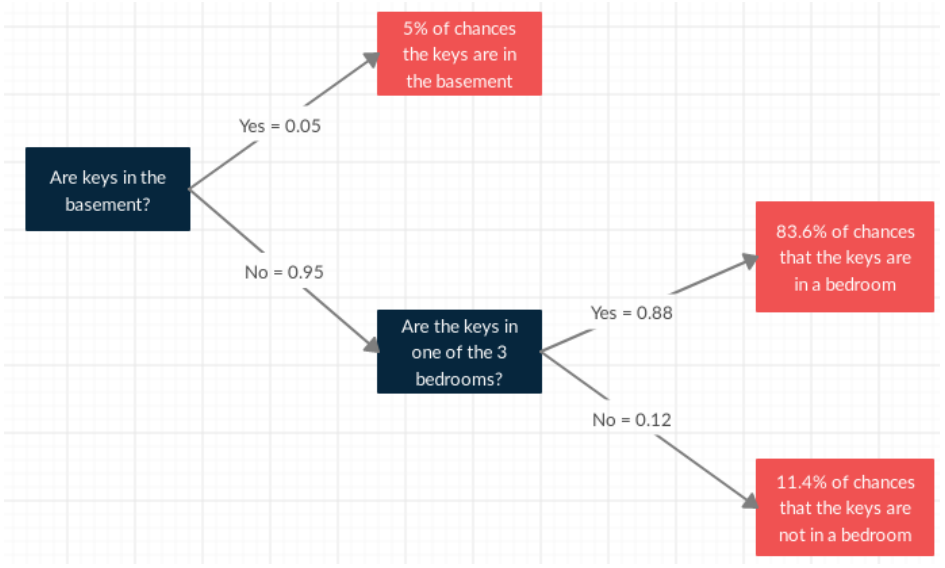
5.3 Random Forest Model

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.

```
library(randomForest)

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

```
0.615 sec elapsed
```

```
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 2.368761
2                 2 2.249164
3                 3 2.301336
4                 4 2.288876
5                 5 2.285833
6                 6 2.266551
7                 7 2.244070
8                 8 2.251997
9                 9 2.258866
10               10 2.262359
11               11 2.244073
12               12 2.222780
13               13 2.214335
14               14 2.215829
15               15 2.215989
16               16 2.212921
17               17 2.211649
18               18 2.213452
19               19 2.200049
20               20 2.200226
21               21 2.204085
22               22 2.199638
23               23 2.199273
24               24 2.197093
25               25 2.199657
```

```
26                 26 2.201309
27                 27 2.199083
28                 28 2.199679
29                 29 2.196818
30                 30 2.191968
31                 31 2.186190
32                 32 2.180583
33                 33 2.181283
34                 34 2.182534
35                 35 2.184219
36                 36 2.184766
37                 37 2.183346
38                 38 2.182366
39                 39 2.181025
40                 40 2.178899
41                 41 2.182150
42                 42 2.182987
43                 43 2.181592
44                 44 2.180803
45                 45 2.178406
46                 46 2.177188
47                 47 2.177589
48                 48 2.177090
49                 49 2.176590
50                 50 2.176289
51                 51 2.175554
52                 52 2.174310
53                 53 2.173001
54                 54 2.171928
55                 55 2.170984
56                 56 2.169410
57                 57 2.167981
58                 58 2.166647
59                 59 2.167708
60                 60 2.164764
61                 61 2.165181
62                 62 2.166248
63                 63 2.164364
64                 64 2.163822
65                 65 2.165469
66                 66 2.165136
67                 67 2.166337
68                 68 2.166902
69                 69 2.166487
70                 70 2.165164
71                 71 2.165362
72                 72 2.164087
73                 73 2.162866
74                 74 2.163068
75                 75 2.161131
76                 76 2.159686
77                 77 2.159991
78                 78 2.158494
79                 79 2.157662
```

```
80                 80 2.158089
81                 81 2.157387
82                 82 2.157584
83                 83 2.157232
84                 84 2.157915
85                 85 2.158035
86                 86 2.157415
87                 87 2.155942
88                 88 2.155772
89                 89 2.155696
90                 90 2.156234
91                 91 2.155780
92                 92 2.156400
93                 93 2.157072
94                 94 2.156709
95                 95 2.156495
96                 96 2.156653
97                 97 2.155042
98                 98 2.155990
99                 99 2.156618
100               100 2.157045
```

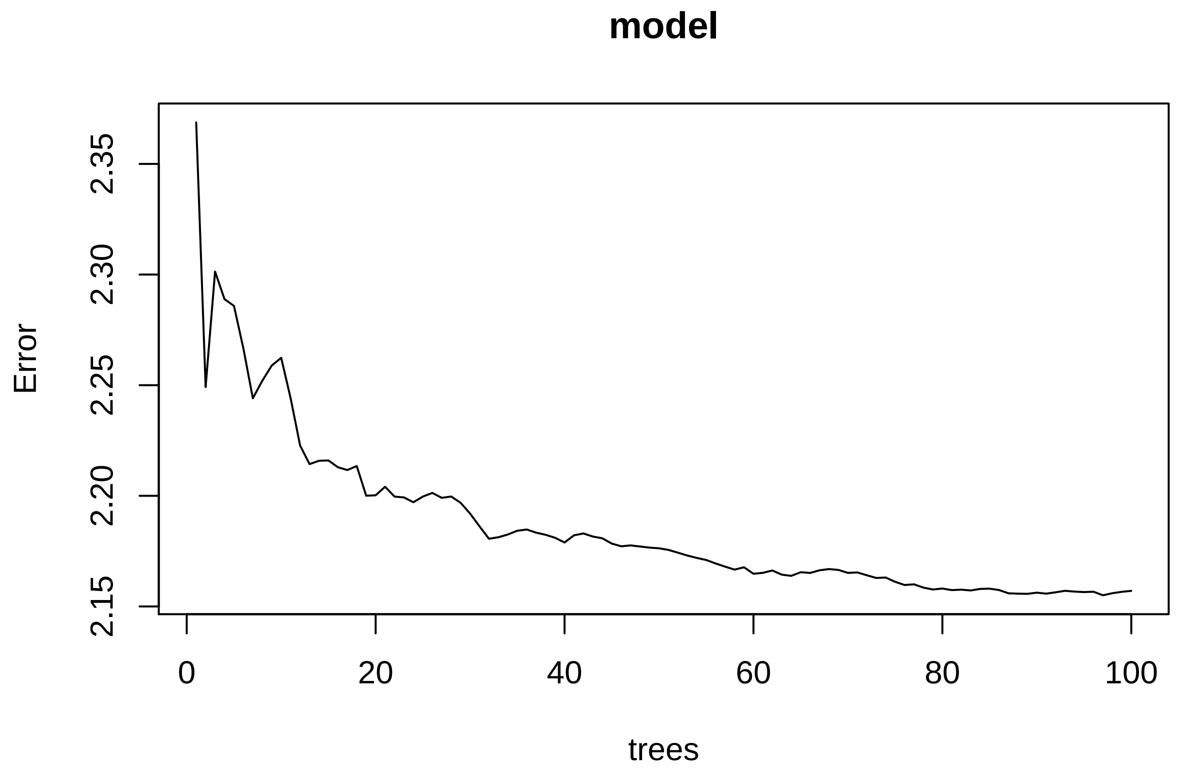
```
random_forest.score <- results$RMSLE[which.min(results$RMSLE)]
cat("The number of trees minimizing the RMSLE is:", results$number_of_trees[which.min(results$RMSLE)],
```

```
The number of trees minimizing the RMSLE is: 97
```

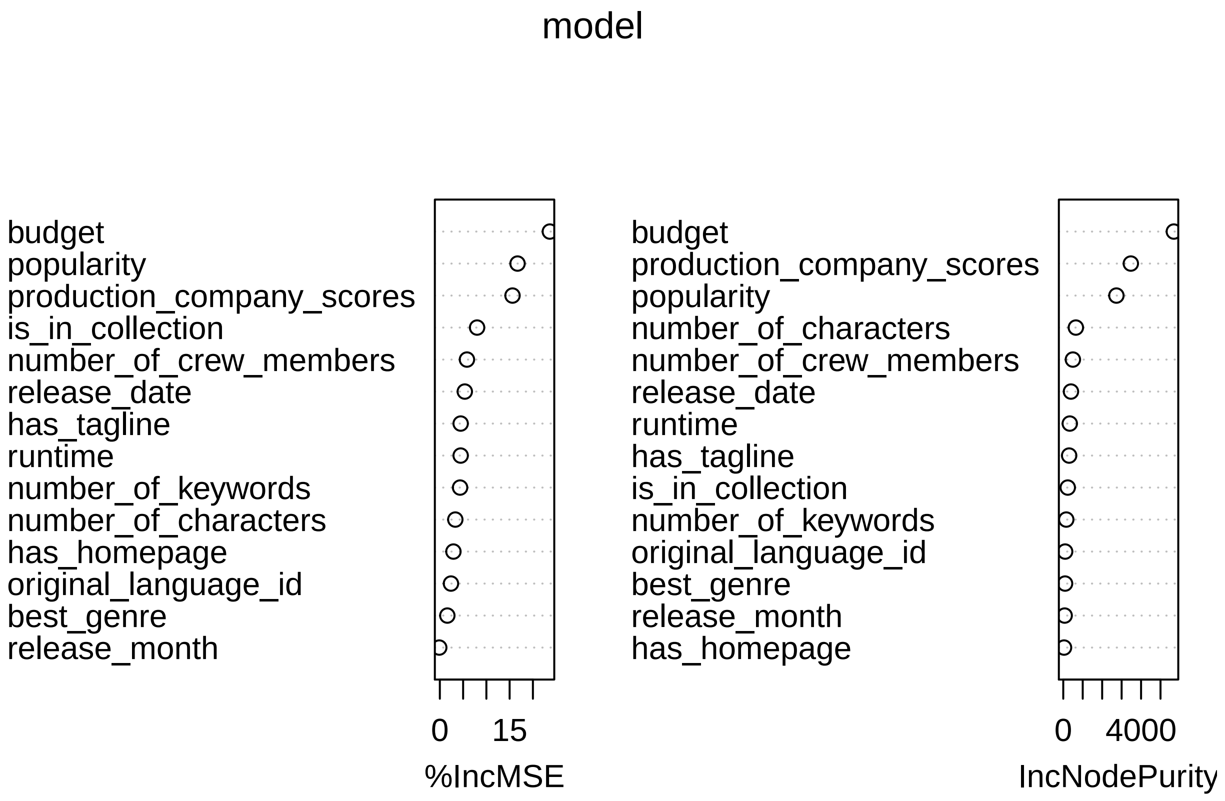
```
cat("The minimum RMSLE is:", random_forest.score, "\n")
```

```
The minimum RMSLE is: 2.155042
```

```
plot(model)
```



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



```
                              %IncMSE IncNodePurity
budget                     23.6419927    5685.40079
popularity                 16.6974573    2732.42505
release_date                5.3620066     394.96814
runtime                     4.4720667     335.05207
is_in_collection            8.0027530     230.95018
number_of_keywords          4.3472620     153.81393
number_of_crew_members      5.8226255     494.08641
number_of_characters        3.3196635     647.20124
best_genre                  1.6059892      92.62662
has_homepage                2.9171374      39.57024
original_language_id        2.3888348      96.65482
has_tagline                 4.4749976     301.83284
release_month              -0.1261375      71.80266
production_company_scores 15.6155381    3473.59842
```

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	2.726380	0.006
Extreme Gradient Boosting Trees	2.017816	0.398
Random Forest	2.155041	0.615

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  4935 1231108665
2  3964 1135221749
3  5463 1132658809
4  3986 1105121596
5  4051 1073400948
6  3097 1070740648
7  4797 1054928808
8  3210 1054256975
9  7393 1037945890
10 3045 1032295355
```

6.3 Retrospective on Predictions