# Capstone MovieLens

## Contents


**1. Introduction** **1**

1.1. The Project Goals . . . . . 1

1.2. The Dataset . . . . . 2

1.3. The Main Steps . . . . . 2

**2. Methodology and Analysis** **2**

2.1. Data import and preprocessing . . . . . 2

2.2. Data Exploration and Visualization . . . . . 5

2.3. Modeling Approach . . . . . 15

**3. Model Selection** **16**

3.1. Basic Model . . . . . 16

3.2. The movie Model . . . . . 17

3.3. The movie and user Model . . . . . 18

3.4. The movie, user and genres Model . . . . . 19

3.5. The movie, user and genres Model with regularization . . . . . 20

**4. Results** **23**

**5. Conclusion** **25**


# 1. Introduction

## 1.1. The Project Goals

Online streaming services have known a tremendous success in the last years, strengthened in 2020 by the covid crisis. With the multiplication of sources and contents, movie / serie recommender systems has become a key element both for provider and users. Netflix, one of the leading provider of online streaming has even launched in 2006 a a challenge to the data science community: improve their recommendation algorithm by 10% and win a million dollars.

For this project, which is one of the 2 projects of the HarvardX: PH125.9x Data Science: Capstone course, we will create a movie recommendation system using the MovieLens dataset.

MovieLens is a research site run by GroupLens Research at the University of Minnesota. Based on users movie ratings, MovieLens uses “collaborative filtering” technology to generate personalized predictions for movies. MovieLens provides datasets of several sizes of their user ratings.

### 1.2. The Dataset

As of January 2021, the latest version of the entire dataset generated in September 2018, contains 27,753,444 ratings and 1,108,997 tag applications across 58,098 movies. These data were created by 283,228 users between January 09, 1995 and September 26, 2018.

We will use the 10M version of the MovieLens dataset to make the computation easier. This data set contains 1,000,0054 ratings and 95,580 tags applied to 10,681 movies by 71,567 users. Users were selected at random for inclusion. All users selected had rated at least 20 movies. No demographic information is included. Each user is only represented by an id that has been anonymized. Movies are identifier by an id number, a title that also included the release year and one or several genre(s). Ratings are made on a 5-star scale, with half-star increments. The date and time of rating has also been added to this dataset.

Citation: F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

### 1.3. The Main Steps

We will first start by importing, checking and preparing the dataset provided by MovieLens. This step will notable include the split of the dataset in a training set used for modeling and a validation set that we will use to assess the performance of our model(s).

Then, we will analyse the content of the dataset and identify how we can use the different information or variables to correctly predict the rating of a movie. This step will rely on various data exploration and visualization tools.

We will then create different models based on the variables identified and selected during the exploration phase. We will train these models on the training set and test them on the validation set. We will eventually use the residual mean squared error (RMSE) to assess and select the best performing model in the results section.

## 2. Methodology and Analysis

### 2.1. Data import and preprocessing

Based on the code provided for this project, we are going to import the date and create a training set and a validation set with a split of 90/10. In order to ensure an appropriate evaluation of our algorithm(s), we ensure that movies and users in the validation set are aslo present in the training set.

First, we load the following libraries for our data analysis and modeling.

```
# Check and install the required libraries
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("data.table", repos = "http://cran.us.r-project.org")

# Load the required libraries
library(tidyverse)
library(caret)
library(data.table)
```

```
library(ggplot2)
library(lubridate)
library(gridExtra)
```

Based on the code and instructions provided in the Harvard Capstone course, we import the dataset, format the columns and create the training set ("edx") and validation set ("validation"). This code ensures that the movies and users in the validation set are also present in the training set.

```
# Import date and create training and test sets

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")


# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The training and validation sets are made respectively of 9,000,055 and 999,999 rows and 6 columns. Each row corresponds to one rating with the columns providing the detail of: - The user Id, - The movie Id, - The movie Title, - The movie genre(s), knowning one movie can have several genres. - The rating, which is the feature we want to predict, - The date / time of the rating provided with a timestamp.

```
# Display the dimensions of the training set (edx) and validation set (validation)
dim(edx)
```

```
## [1] 9000055       6
```

```
dim(validation)
```

```
## [1] 999999      6
```

We can see that the dataset is already in tidy format.

```
# Display the first rows of the training set
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

The data does not have any missing values.

```
# Check if the training and validation sets have any missing values.
any(is.na.data.frame(edx))
```

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

```
any(is.na.data.frame(validation))
```

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

In order to complete the data available for our analysis, we are going to add in a separate column the year of the movie which is provided in the title.

```
# Add the year of the movie to the training and validation sets
edx <- edx %>% mutate(movie_year = as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% mutate(movie_year = as.numeric(str_sub(title,-5,-2)))
# Display the first rows of the trainig set to check
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres movie_year
## 1:                Comedy|Romance       1992
## 2:         Action|Crime|Thriller       1995
## 3:  Action|Drama|Sci-Fi|Thriller       1995
## 4:       Action|Adventure|Sci-Fi       1994
## 5: Action|Adventure|Drama|Sci-Fi       1994
## 6:       Children|Comedy|Fantasy       1994
```

## 2.2. Data Exploration and Visualization

Most of the the data exploration and visualization will be performed on the training set which represents 90% of the whole dataset.

### 2.2.1. The rating

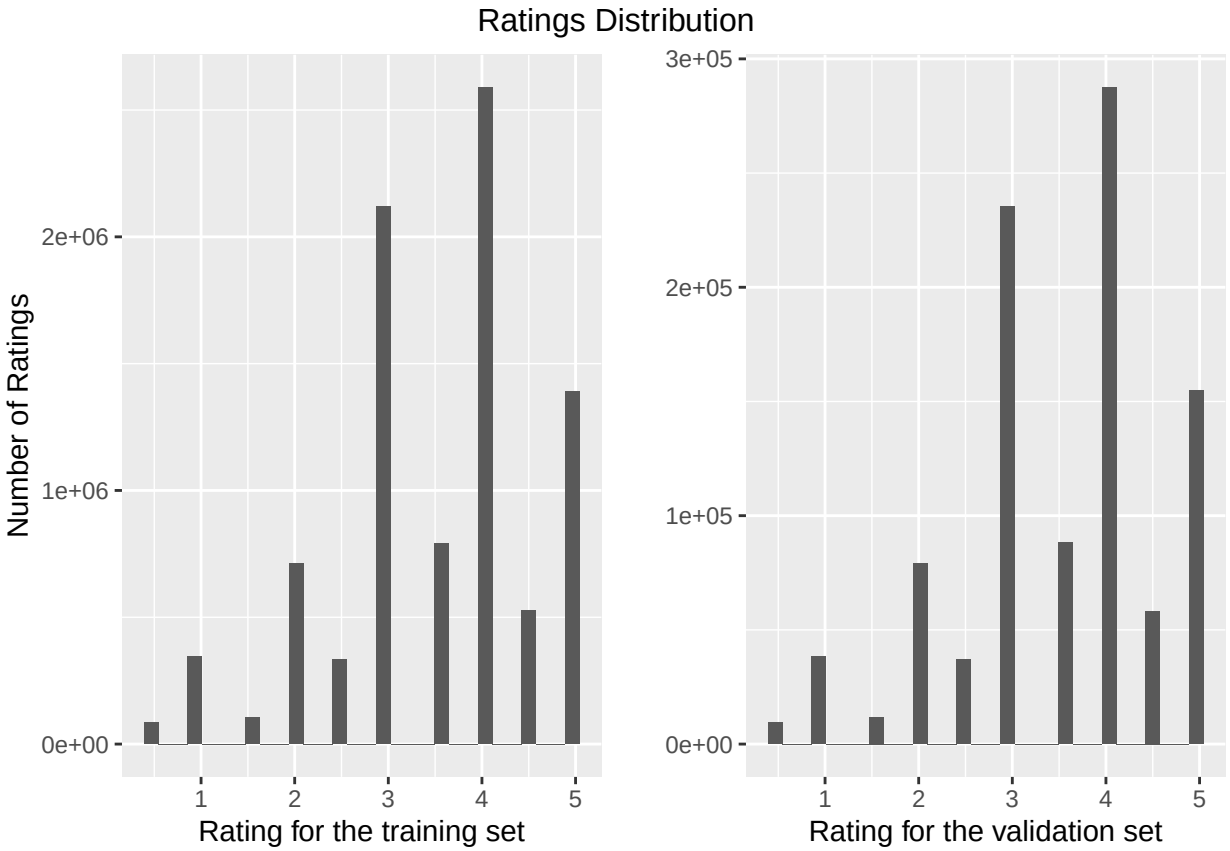
The rating distribution of the training set shows that: - 82% of the ratings are in the 3 to 5 range. The first 3 ones being 4, 3 and 5. - There are no 0 rating, - 80% of the ratings correspond to round number vs 20% for half numbers / stars.

```
# Compute the percentage of movie ratings per rating for the training set
prop.table(table(edx$rating))
```

```
##
##         0.5           1         1.5           2         2.5           3
## 0.009485942 0.038408543 0.011825039 0.079046406 0.037000885 0.235691893
##         3.5           4         4.5           5
## 0.087957685 0.287601576 0.058525865 0.154456167
```

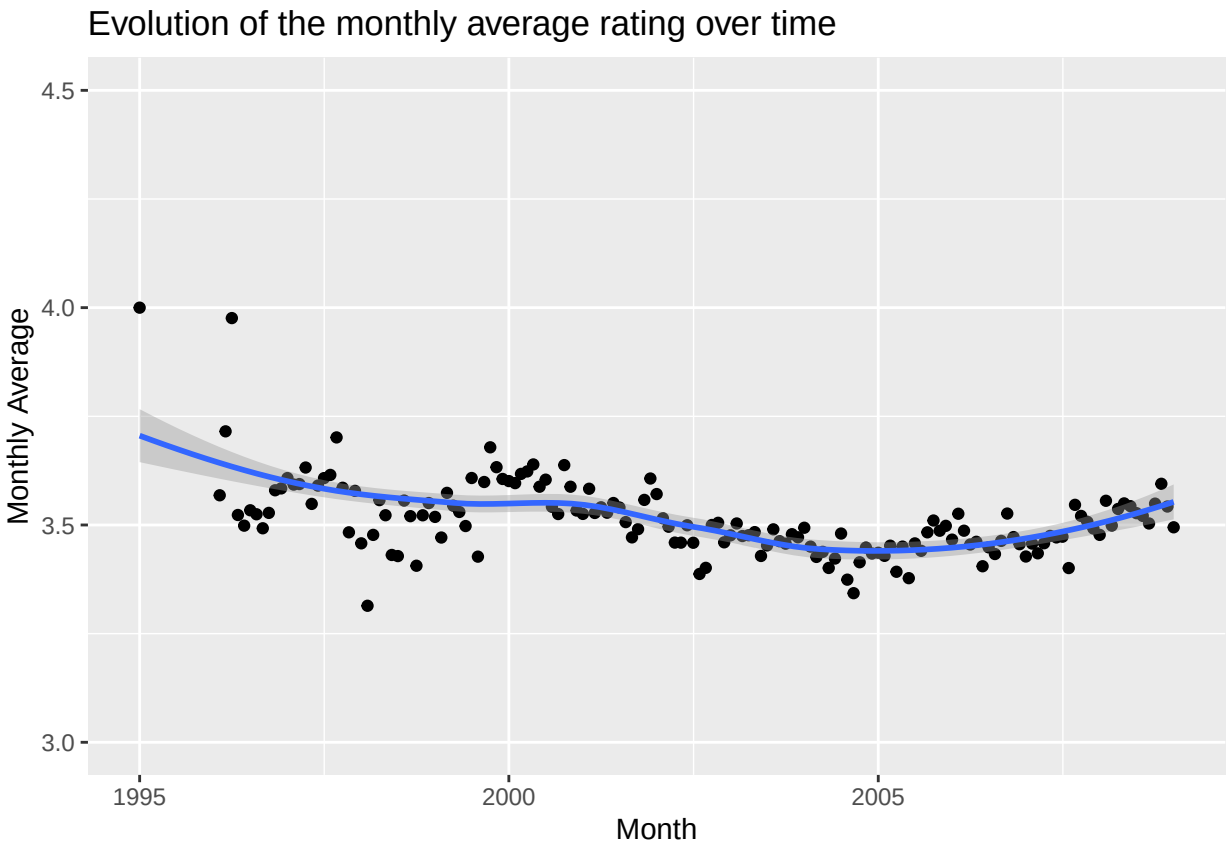
As expected, since the split train / validation was made based on the quality column, we can see that the distribution is similar for the validation set.

```
# Display side by side the distribution ratings for the training and validation sets
grid.arrange(qplot(edx$rating, xlab = "Rating for the training set", ylab = "Number of Ratings"),qplot(
```




We can now have a look at the evolution of the rating average over time.

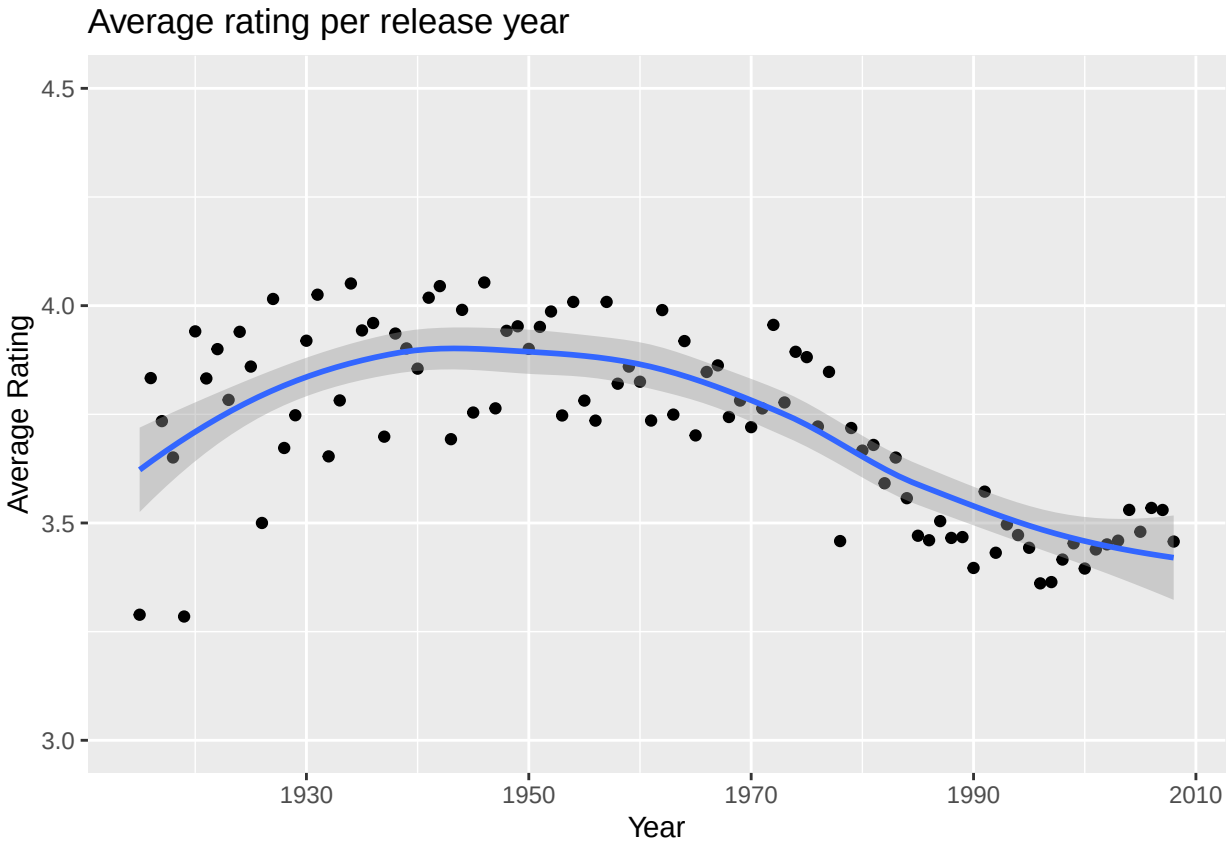
```
# Create a chart representing the evolution of the rating average over time
edx %>%
  mutate(date = as_datetime(timestamp))%>% # add a date column with a date format from the timestamp co
  group_by(month_date = round_date(date,unit="month")) %>% # group thee ratings by month of rating
  summarize(monthly_rating = mean(rating)) %>% # compute the average number of ratings per month
  ggplot(aes(y=monthly_rating,x=month_date)) + # Create a chart with the monthly averages and the corre
  ylim(c(3, 4.5)) +
  geom_point() +
  geom_smooth()  +
  labs(x="Month", y="Monthly Average") +
  ggtitle("Evolution of the monthly average rating over time")
```




We can see that the range of the variation in rating average is fairly limited over time. Thus, we can consider that the impact of the time on rating is quite limited and does not need to be included in our model(s).

Although showing a slightly more important variation range, we will also consider that the year of relase of the movies has a limited impact on the average rating.

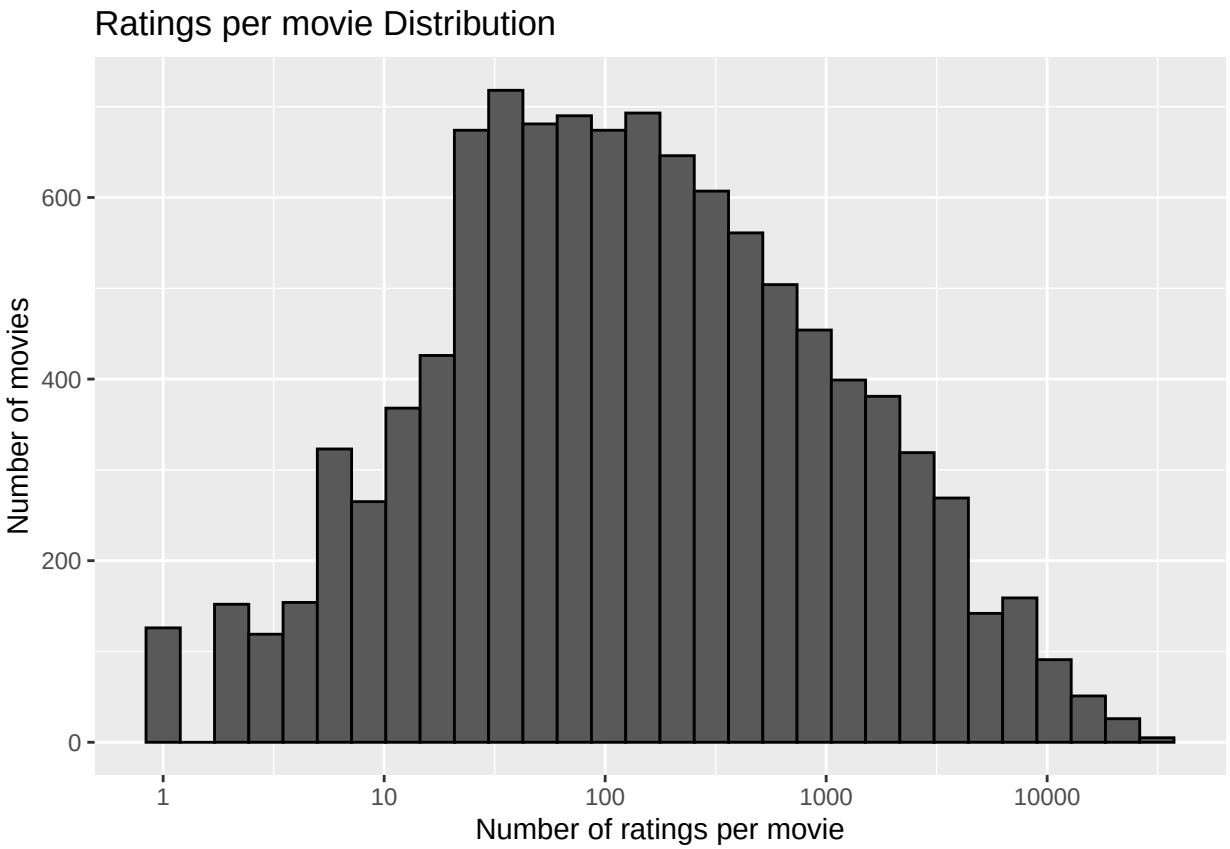
```
# Create a chart representing the evolution of the rating average depending on the year of release of t
edx %>%
  group_by(movie_year) %>% # Group the ratings my year of release
  summarize(average_rating = mean(rating)) %>% # Compute the corresponding yearly averages
  ggplot(aes(x=movie_year,y=average_rating)) + # Create a chart representing the yearly averages and th
  ylim(c(3, 4.5)) +
  geom_point() +
  geom_smooth()  +
  labs(x="Year", y="Average Rating") +
  ggtitle("Average rating per release year")
```




### 2.2.2. The movies

The number of ratings per movie distribution shows how some movies are more rated than others.

```
# Create an histogram of the number of ratings per movie
edx %>%
  count(movieId) %>%
  ggplot(aes(n))  +
  geom_histogram(bins=30,color='black') +
  scale_x_log10() +
  labs(x="Number of ratings per movie", y="Number of movies") +
  ggtitle("Ratings per movie Distribution")
```




We can notably see that more than 1,000 movies have less than 10 reviews.

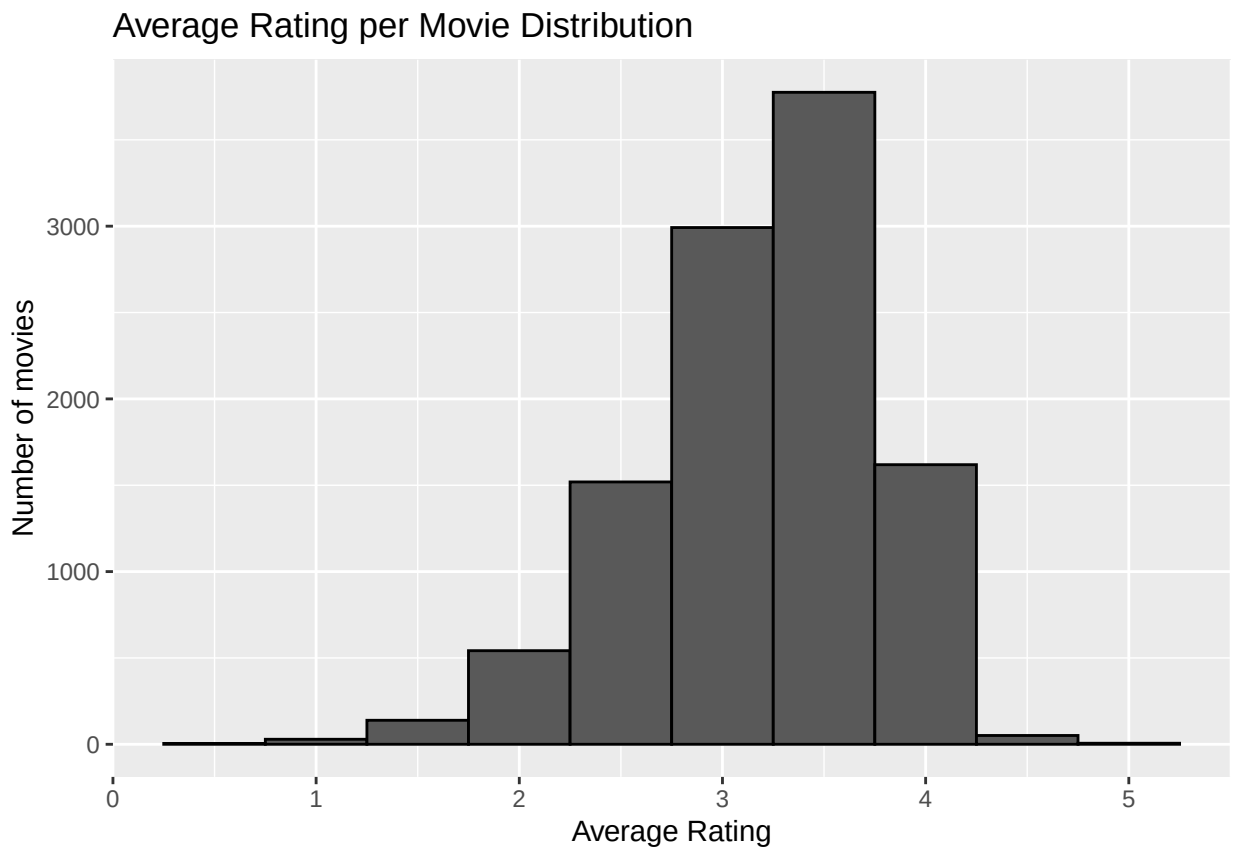
```
# Counts the number of movies, which have less than 10 reviews
edx %>%
  count(movieId) %>%
  filter(n<10) %>%
  nrow()
```

```
## [1] 1054
```

It implies that we will have to give less weight to movies that have been rated only a limited number of times.

The average rating per movie distribution also shows how movie are rated. We can see there are significant differences especially for movies having an average rating of less than 2.5 / more than 4 and movies having an average rating of 3 and 3.5.

```
# Create an histogram of the average ratings per movie
edx %>%
  group_by(movieId) %>%
  summarize(average_rating=mean(rating)) %>%
  ggplot(aes(average_rating))  +
  geom_histogram(bins=10,color='black') +
  labs(x="Average Rating", y="Number of movies") +
  ggtitle("Average Rating per Movie Distribution")
```

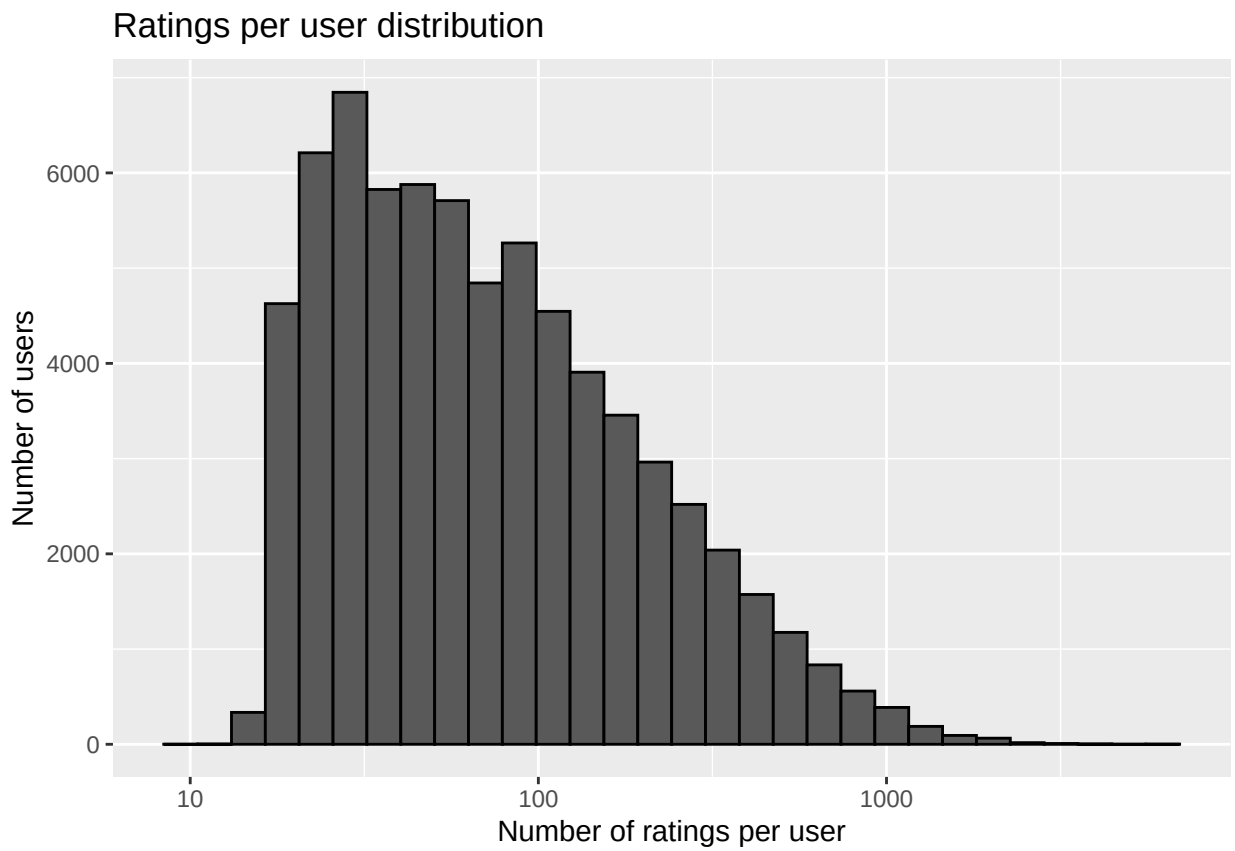



These 2 charts emphasizes how movies can be rated differently in number of rating and value / rate given, which we can qualify as movie effect.

### 2.2.3. The users

Likewise, we can show how some users are rating movies more often than others with a full range from about 10 to more than 1,000:

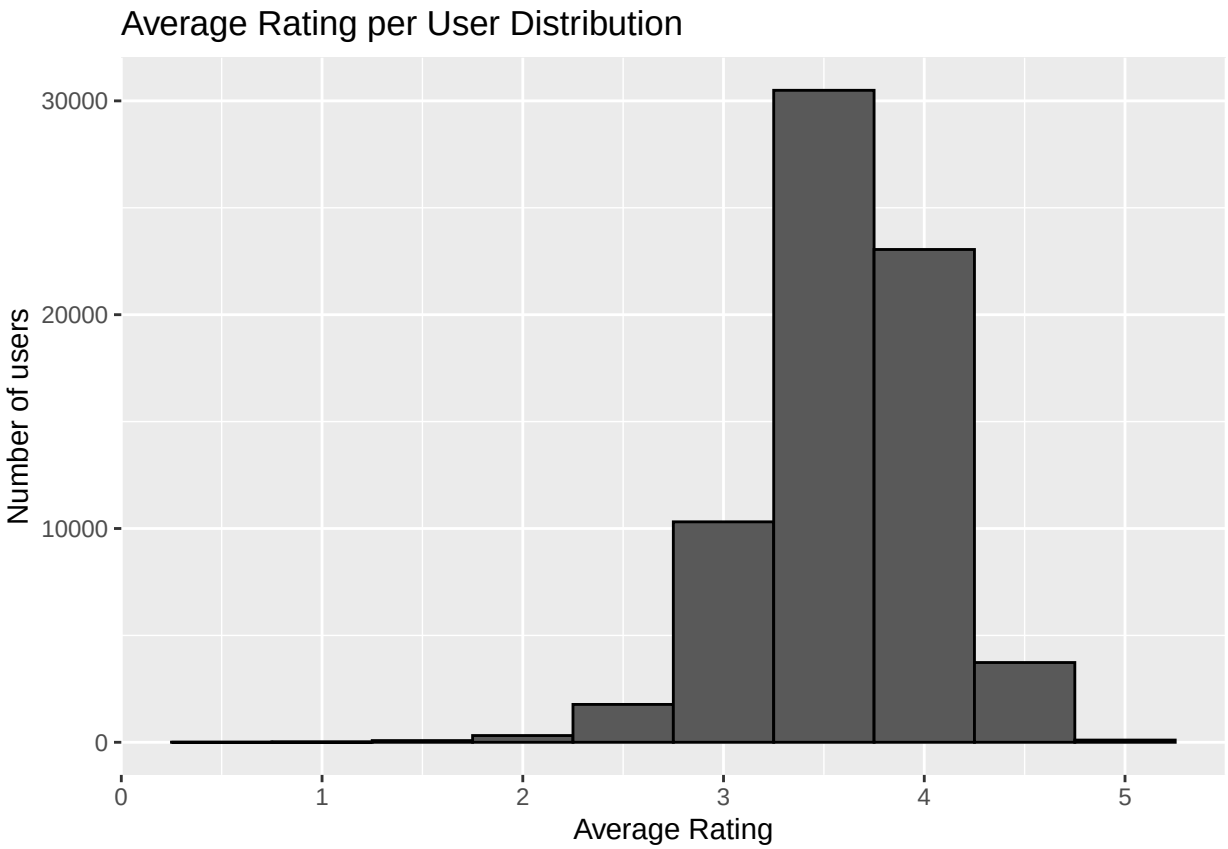
```
# Create an histogram of the number of ratings per user
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  labs(x="Number of ratings per user", y="Number of users") +
  ggtitle("Ratings per user distribution")
```




We can also see how users rate differently the movies in terms of rating average:

```
# Create an histogram of the average rating given per user
edx %>%
  group_by(userId) %>%
  summarize(average_rating=mean(rating)) %>%
  ggplot(aes(average_rating))  +
  geom_histogram(bins=10,color='black') +
  labs(x="Average Rating", y="Number of users") +
  ggtitle("Average Rating per User Distribution")
```

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

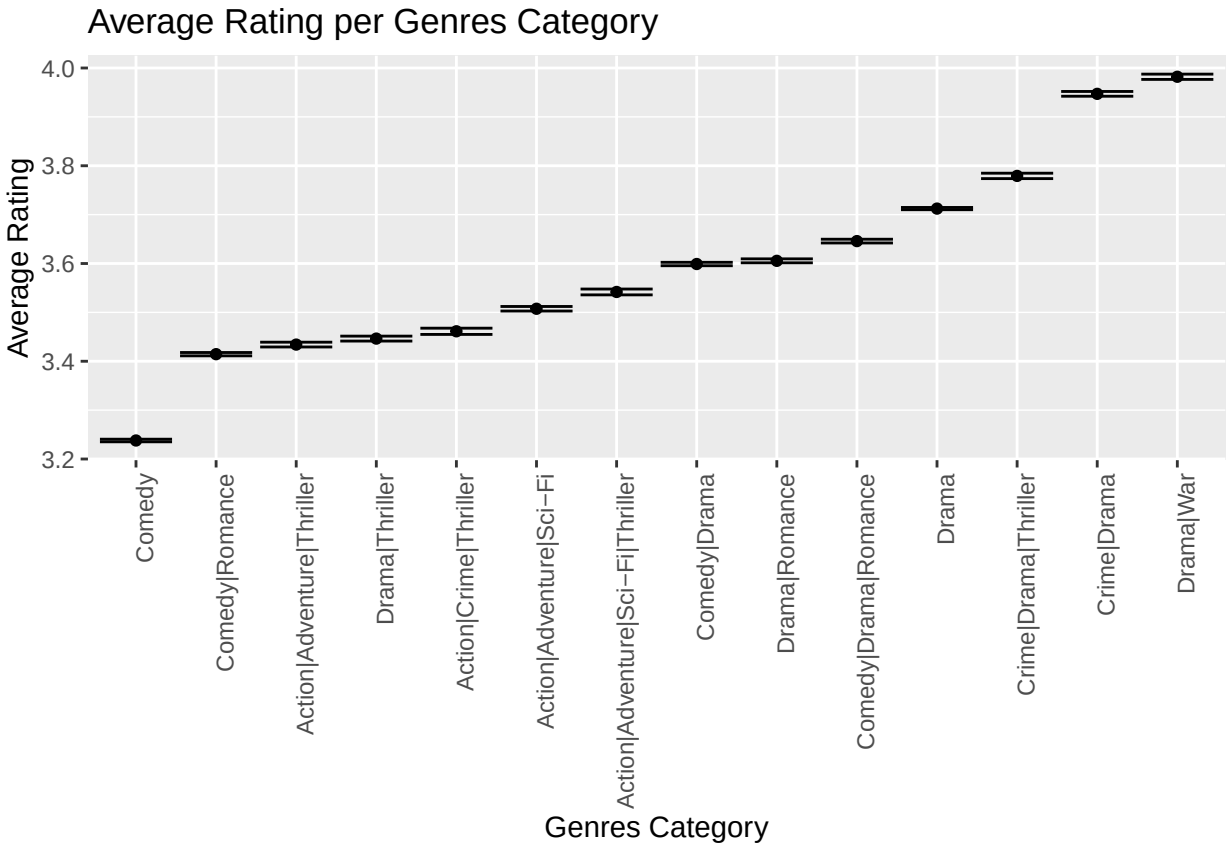



These 2 charts emphasizes how users can rate differently both in terms of number of rating and value / rate given, which we can qualify as user effect.

#### 2.2.4. The genres

We can also notice the impact of the genres on the average rating and corresponding standard error. Below chart shows the average rating per combination of genres provided by the dataset for the combinations having more than 100,000 ratings and sorted by highest average.

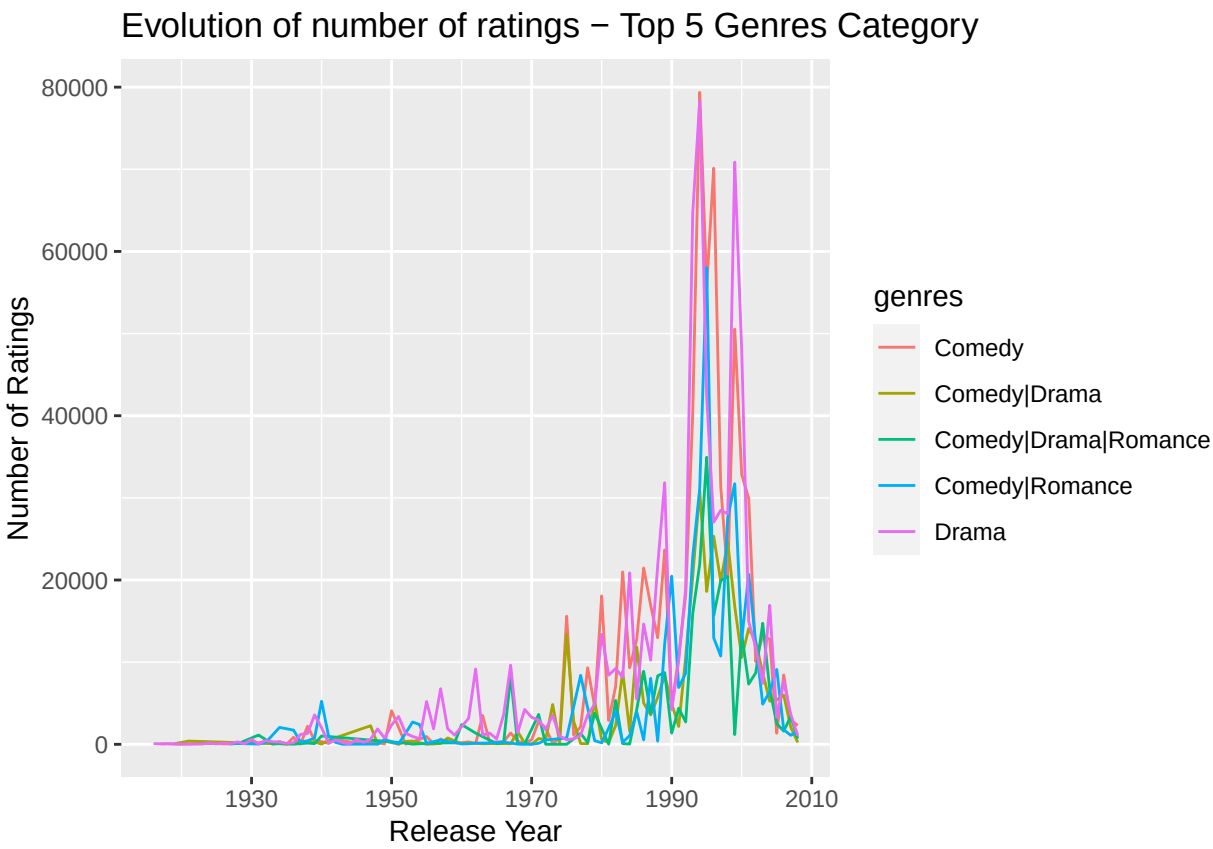
```
# Create a chart representing the average rating per combination of genres for the combinations having
edx %>%
  group_by(genres) %>% # Group the ratings by genre
  summarize(number = n(), average = mean(rating), se=sd(rating)/sqrt(number)) %>% # Compute the number,
  filter(number > 100000) %>% # Filter on the genres having more than 50,000 ratings
  arrange(desc(average)) %>% # Sort by descending order of average number of ratings per genre
  mutate(genres=reorder(genres,average)) %>% # Reorger the genres by ascending average for the chart
  ggplot(aes(x=genres, y=average, ymin=average-2*se,ymax=average+2*se)) + # Create the chart of average
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  labs(x="Genres Category", y="Average Rating") +
  ggtitle("Average Rating per Genres Category")
```




We can also see how the genres rating evolves over time. Below chart presents the evolution of the top 5 genres combination per year of release.

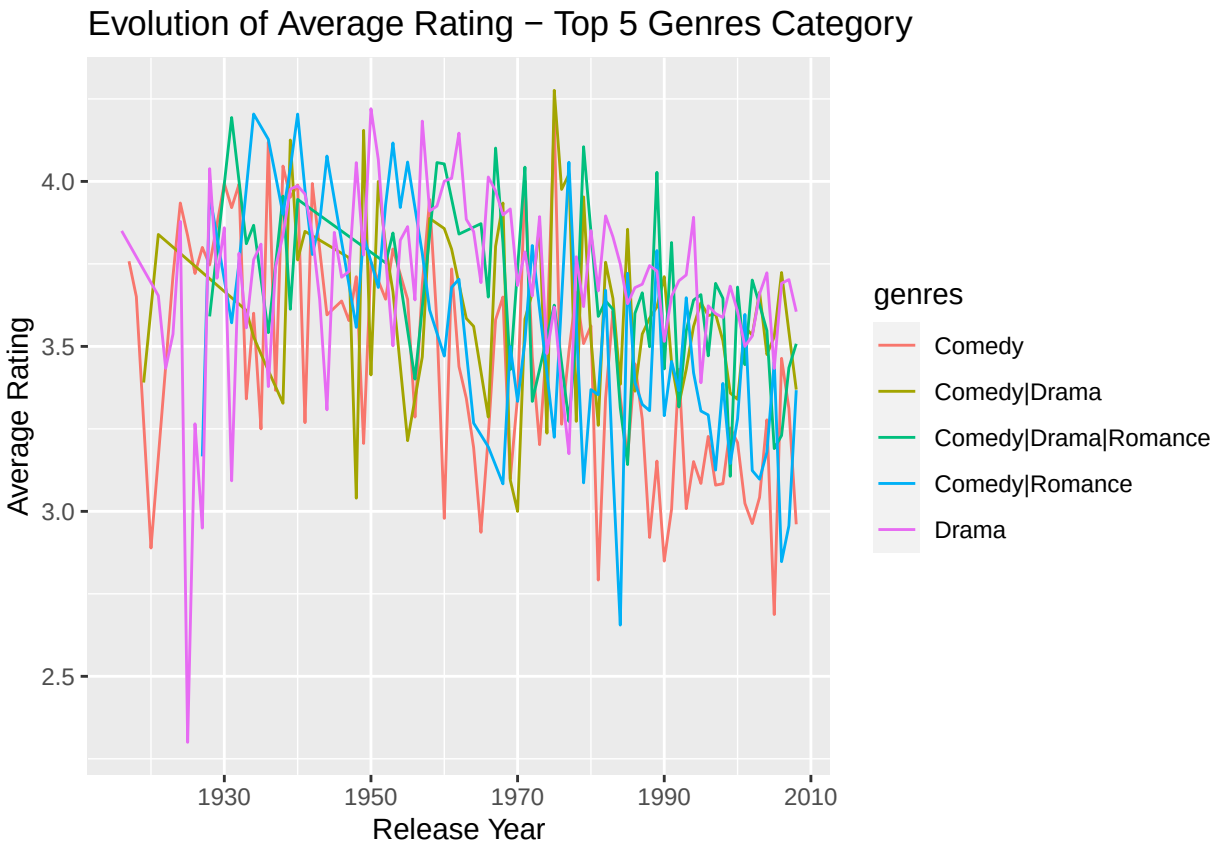
```
# Get the top 5 genres per number of ratings
top_5_genres <- edx %>%
  group_by(genres) %>% # Group the ratings by genre
  summarize(number = n()) %>% # Compute the number of ratinges per genre
  arrange(desc(number)) %>% # Sort by descending number of ratings per genre
  head(5) # Get the top 5 genres with the corresponding number of rating

# Create line chart representing the evolution of the number of ratings per genre over time
edx %>%
  filter(genres %in% top_5_genres$genres) %>% # Filter the training set on the top 5 genres per number
  group_by(movie_year,genres) %>% # Group by year of release and genres
  summarize(rating_number = n()) %>% # Compute the corresponding number of ratings
  ggplot(aes(x = movie_year, y = rating_number)) + # Create the chart representing the evolution of num
  geom_line(aes(color=genres)) +
  labs(x="Release Year", y="Number of Ratings") +
  ggtitle("Evolution of number of ratings - Top 5 Genres Category")
```




Likeweise, we can have a look at the evolution of averate rating per genre for the the top 5 genres over time:

```
# Create line chart representing the evolution of the average rating per genre over time
edx %>%
  filter(genres %in% top_5_genres$genres) %>% # Filter the training set on the top 5 genres per number
  group_by(movie_year,genres) %>% # Group by year of release and genres
  summarize(average_rating = mean(rating)) %>% # Compute the corresponding average rating
  ggplot(aes(x = movie_year, y = average_rating)) + # Create the chart representing the evolution of th
  geom_line(aes(color=genres)) +
  labs(x="Release Year", y="Average Rating") +
  ggtitle("Evolution of Average Rating - Top 5 Genres Category")
```




Altough showing some significant variations over time, the number of ratings seems to follow the same trends for the main genres combination. Hence, and as for the movie and user effects, we will not take into account the time factor for the impact of genre.

## 2.3. Modeling Approach

### 2.3.1. Select the models

Based on our data exploration, we will implement and evaluate different models including the effects of movie, users and genres.

We can start as a comparison basis with the simpliest of model that would only be based only on the averate rating calculated based on the training set.

Then, our goal will be to add the different effects identified in the data exploration phase and to assess the impact in terms of performance.

### 2.3.2. The models evaluation

The evaluation will be based on the based on the residual mean squared error (RMSE) on a test set. We define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with $\hat{y}_{u,i}$. The RMSE is then defined as:

$\sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$ with N being the number of user/movie combinations and the sum occurring over all these combinations.

We will use the following function to evaluate our models.

```r
# Define the RMSE function
RMSE <- function(true_ratings, predicted_ratings){
     sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

#### 2.3.3. Create Training and Test sets

In order to train, test and select our models, we are going to split our edx dataset in a training set and a test set.

```r
# Test set will be 10% of edx data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure that the userId, movieId and genres in test set are also present in the training set
test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by = "genres")

# Add rows removed from the test set back into training set
removed <- anti_join(temp, test_set)
train_set<- rbind(train_set, removed)
rm(test_index, temp, removed)
```

# 3. Model Selection

## 3.1. Basic Model

So, if we start with the basic model that we will use as comparison basis, we calculate first the average of all ratings for the training set.

```r
# Define the function to create, train the basic model and return the prediction
basic_model <- function(training_dataset){
  # Compute the average rating
  mu <- mean(training_dataset$rating)
  # Return the average rating
  return(mu)
}
# Compute the basic model with the training set
basic_model(train_set)
```

```
## [1] 3.512456
```

Then, we compute the RMSE with our fonction:

```r
# Compute the predicted rating for the test set (= average)
basic_model_prediction <- basic_model(train_set)

# Compute the corresponding RSME
basic_model_rmse <- RMSE(test_set$rating, basic_model_prediction)
basic_model_rmse
```

```
## [1] 1.060054
```

To be able to compare the performance of our different models, we create the following table:

```r
# Create the table and store the RMSE corresponding to our first basic model
rmse_results <- tibble(Method = "Average Rating Model", RMSE = basic_model_rmse)
head(rmse_results)
```

```
## # A tibble: 1 x 2
##   Method                  RMSE
##   <chr>                  <dbl>
## 1 Average Rating Model 1.060054
```

### 3.2. The movie Model

Our goal here is to add to the basic model the movie effect that we can consider as the difference between the movie rating and the average rating.

We can build this models with the following function:

```r
# Create a function to build, train the movie model based on a training dataset and return the predicti
movie_model <- function(training_dataset, test_dataset){
  # Compute the average rating
  mu <- mean(training_dataset$rating)
  # Compute the movie effect
  movie_effect <- training_dataset %>%
    group_by(movieId) %>% # Group the ratings by movie
    summarize(b_i = mean(rating - mu)) # Compute and store the movie effect the difference between the
  # Compute the predictions for the movies in the test set
  model_prediction <- mu + test_dataset %>%
  left_join(movie_effect, by='movieId') %>%
  .$b_i
  # Return the predictions
  return(model_prediction)
}
```

Thanks to this function, we can then train our model based on the train_set and return the predictions based on the test_set:

```r
# Compute the predicted ratings based on the test set for this model
model_1_prediction <- movie_model(train_set,test_set)

# Compute the corresponding RMSE
model_1_rmse <- RMSE(test_set$rating, model_1_prediction)
model_1_rmse
```

```
## [1] 0.9429615
```

Comparing to our basic model, we can see that the movie effect decreases the RMSE thus improving our model.

```
# Update the RMSE results table with the RMSE of this model
rmse_results <- bind_rows(rmse_results,tibble(Method="Movie Effect Model", RMSE = model_1_rmse ))
rmse_results
```

```
## # A tibble: 2 x 2
##   Method                    RMSE
##   <chr>                    <dbl>
## 1 Average Rating Model 1.060054
## 2 Movie Effect Model   0.9429615
```

### 3.3. The movie and user Model

We are going now to add the user effect as the difference between the user rating of a movie and the average + movie effect of the corresponding movie.

We can build this models with the following function:

```
# Create a function to build, train the movie + user model based on a training dataset and return the p
movie_user_model <- function(training_dataset, test_dataset){
  # Compute the average rating
  mu <- mean(training_dataset$rating)
  # Compute the movie effect
  movie_effect <- training_dataset %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))
  # Compute the user effect
  user_effect <- training_dataset %>%
  left_join(movie_effect, by='movieId') %>% # Add the movie effect to the training set
  group_by(userId) %>% # Group the rating by user id
  summarize(b_u = mean(rating - mu - b_i)) # Compute and store the user effect
  # Compute the predictions based on the test_dataset
  model_prediction <- test_dataset %>%
     left_join(movie_effect, by='movieId') %>%
     left_join(user_effect, by='userId') %>%
     mutate(prediction = mu + b_i + b_u) %>%
     .$prediction
  # Return the predictions
  return(model_prediction)
}
```

We can then train our model based on the train_set and return the predictions based on the test_set:

```
# Compute the predicted ratings based on the test set for this model
model_2_prediction <- movie_user_model(train_set,test_set)

# Compute the corresponding RMSE
model_2_rmse <- RMSE(test_set$rating, model_2_prediction)
model_2_rmse
```

```
## [1] 0.8646843
```

The user effect improves further our model as we can see by comparing with the previous models.

```
# Update the RMSE results table with the RMSE of this model
rmse_results <- bind_rows(rmse_results, tibble(Method="Movie + User Effects Model",  RMSE = model_2_rmse
rmse_results
```

```
## # A tibble: 3 x 2
##   Method                          RMSE
##   <chr>                          <dbl>
## 1 Average Rating Model       1.060054
## 2 Movie Effect Model         0.9429615
## 3 Movie + User Effects Model 0.8646843
```

### 3.4. The movie, user and genres Model

After the movie and user effects, we now compute the gender effect as the difference between the average rating for a genre and the sum of average rating + movie effect + user effect.

We can build this models with the following function:

```
# Create a function to build, train the movie + user + genre model based on a training dataset and retu
movie_user_model_genres <- function(training_dataset, test_dataset){
  # Compute the average rating
  mu <- mean(training_dataset$rating)
  # Compute the movie effect
  movie_effect <- training_dataset %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))
  # Compute the user effect
  user_effect <- training_dataset %>%
  left_join(movie_effect, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
  # Compute the genres effect
  genres_effect <- training_dataset %>%
    left_join(movie_effect, by='movieId') %>% # Add the movie effect to the training set
    left_join(user_effect, by='userId') %>%  # Add the user effect to the training set
    group_by(genres) %>% # Group the ratings per genre
    summarize(b_g = mean(rating - mu - b_i - b_u)) # Compute and store the genre effect
  # Compute the predictions based on the test_dataset
  model_prediction <- test_dataset %>%
    left_join(movie_effect, by ='movieId') %>%
    left_join(user_effect, by ='userId') %>%
    left_join(genres_effect, by ='genres') %>%
    mutate(prediction = mu + b_i + b_u + b_g) %>%
    .$prediction
  # Return the predictions
  return(model_prediction)
}
```

We can then train our model based on the train_set and return the predictions based on the test_set:

```
# Compute the predicted ratings based on the test set for this model
model_3_prediction <- movie_user_model_genres(train_set,test_set)

# Compute the corresponding RMSE
model_3_rmse <- RMSE(test_set$rating, model_3_prediction)
model_3_rmse
```

```
## [1] 0.8643241
```

The user effect slightly improves further our model as we can see by comparing with the previous models.

```
# Update the RMSE results table with the RMSE of this model
rmse_results <- bind_rows(rmse_results, data_frame(Method="Movie + User + Genres Effects Model",  RMSE =
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
rmse_results
```

```
## # A tibble: 4 x 2
##   Method                                RMSE
##   <chr>                                <dbl>
## 1 Average Rating Model                1.060054
## 2 Movie Effect Model                  0.9429615
## 3 Movie + User Effects Model          0.8646843
## 4 Movie + User + Genres Effects Model 0.8643241
```

During the data exploration phase, we noticed that: - some movies were more rated than others, with significant differences in average rating per movie, - somes users were rating more than others, with significant differences in average rating per user, - and some genres were more rated than others, with significant differences in average rating per genre.

These differences can actually impact the performance of our models when there are actually a limited number of ratings per movie. In that situation, the potential bias a single user may have will have more impact on the movie and / or the genre comparing to movies with significantly more ratings.

In order to take into account these differences, we will apply to our best model a regularization, which aims at penalizing the RMSE when the number of ratings for the different effects is limited.

## 3.5. The movie, user and genres Model with regularization

Regularization constrains the total variability of the effect sizes by penalizing large estimates that come from small sample sizes. This regularisation is applied thanks to penalty term  that is applied to the different effects.

For instance for the movie effect, the goal is to minimize a RMSE equation that adds a penalty that gets larger when many $b_i$ are large:

$$\sum_{u,i}(y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$$

Using calculus we can actually show that the values of $b_i$ that minimize this equation are:

$\hat{b}_i(\lambda) = \frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i} - \hat{\mu})$ where $n_i$ is the number of ratings made for movie i We can build this model with the following function:

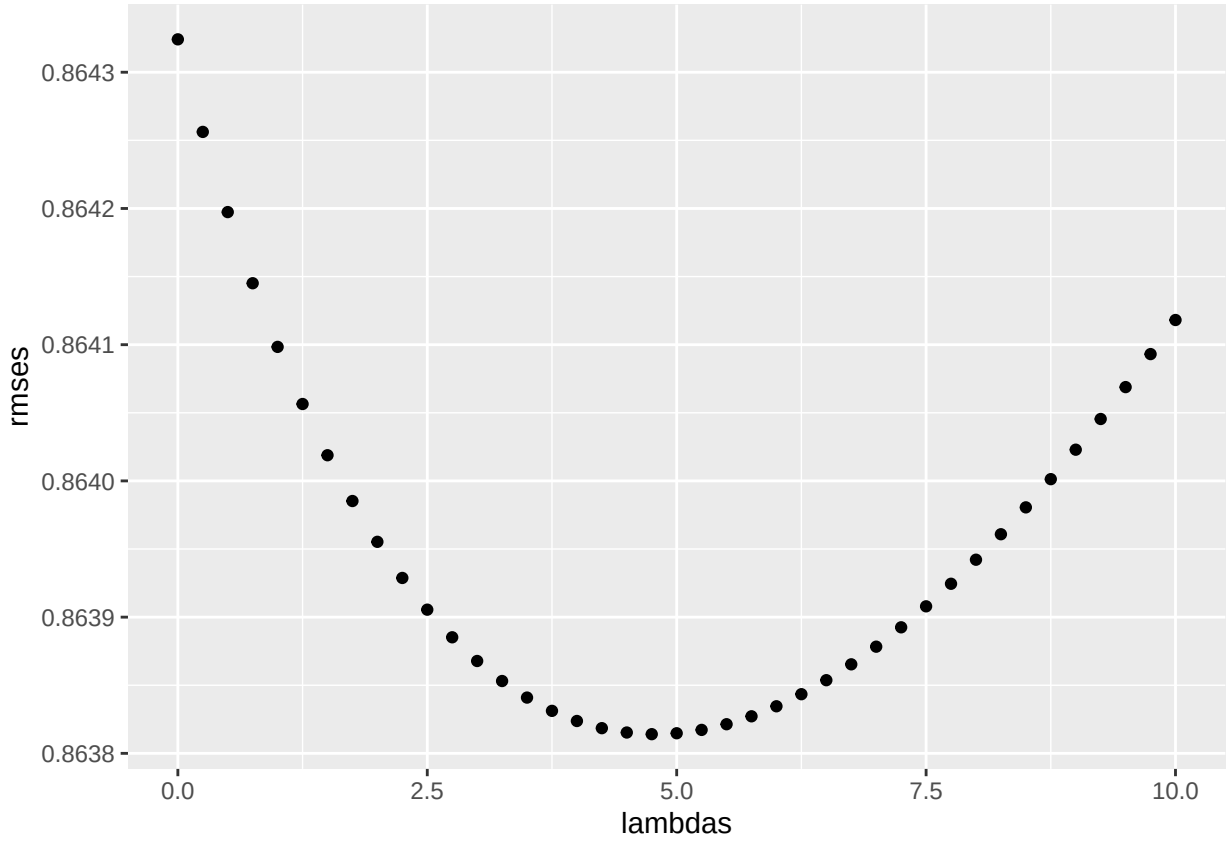
```
# Create a function to build, train the movie + user + genre model including regularization based on a
movie_user_model_genres_reg <- function(training_dataset, test_dataset,l){
  # Compute the average rating
  mu <- mean(training_dataset$rating)
  # Compute the movie effect
  movie_effect <- training_dataset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  # Compute the user effect
  user_effect <- training_dataset %>%
  left_join(movie_effect, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+l))
  # Compute the genres effect
  genres_effect <- training_dataset %>%
    left_join(movie_effect, by='movieId') %>%
    left_join(user_effect, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))
  # Compute the predictions based on the test_dataset
  model_prediction <- test_dataset %>%
    left_join(movie_effect, by ='movieId') %>%
    left_join(user_effect, by ='userId') %>%
    left_join(genres_effect, by ='genres') %>%
    mutate(prediction = mu + b_i + b_u + b_g) %>%
    .$prediction
  # Return the predictions
  return(model_prediction)
}
```

First, we need to find the penalty  that fits best our last model. We will use cross validation on the test set to find the best  .

```
# Set a list of different values of  to test
lambdas <- seq(0, 10, 0.25)

# Compute the RMSE with the new functions with the different values of
rmses <- sapply(lambdas, function(l){
  model_4_prediction <- movie_user_model_genres_reg(train_set,test_set,l)
  return(RMSE(test_set$rating,model_4_prediction))
})
```

```
# Plot the RMSE versus the corresponding values of
qplot(lambdas, rmses)
```




And the best is:

```
# Store the best  that minimize the RMSE
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 4.75
```

We can then train our model based on the train_set and this value of and return the predictions based on the test_set:

```
# Compute the predicted ratings based on the test set for this model
model_4_prediction <- movie_user_model_genres_reg(train_set,test_set, lambda)

# Compute the corresponding RMSE
model_4_rmse <- RMSE(test_set$rating, model_4_prediction)
model_4_rmse
```

```
## [1] 0.8638141
```

The regularization slightly improves further our model as we can see by comparing with the previous models.

```
# Update the RMSE results table with the RMSE of this model
rmse_results <- bind_rows(rmse_results, data_frame(Method="Movie + User + Genres Effects regularized Mo
rmse_results
```

```
## # A tibble: 5 x 2
##   Method                                    RMSE
##   <chr>                                    <dbl>
## 1 Average Rating Model                  1.060054
## 2 Movie Effect Model                    0.9429615
## 3 Movie + User Effects Model            0.8646843
## 4 Movie + User + Genres Effects Model   0.8643241
## 5 Movie + User + Genres Effects regularized Model 0.8638141
```

# 4. Results

We are going to train our different models on the full training set (edx) to confirm the differences in performance and get the smallest RMSE based on the predictions computed on the validation set.

The first basic model computes the average rating for all the movies in the training set:

```
# Compute the predicted rating for the test set (= average)
basic_model_prediction_final <- basic_model(edx)

# Compute the corresponding RSME
basic_model_rmse_final <- RMSE(validation$rating, basic_model_prediction_final)

# Create the table and store the RMSE corresponding to our first basic model
rmse_results_final <- tibble(Method = "Average Rating Model", RMSE = basic_model_rmse_final)
rmse_results_final
```

```
## # A tibble: 1 x 2
##   Method                  RMSE
##   <chr>                  <dbl>
## 1 Average Rating Model 1.061202
```

The model with the movie effect provides a first improvement over the simple rating average.

```
# Compute the predicted ratings based on the test set for this model
model_1_prediction_final <- movie_model(edx,validation)

# Compute the corresponding RMSE
model_1_rmse_final <- RMSE(validation$rating, model_1_prediction_final)

# Update the RMSE results table with the RMSE of this model
rmse_results_final <- bind_rows(rmse_results_final,tibble(Method="Movie Effect Model", RMSE = model_1_rm
rmse_results_final
```

```
## # A tibble: 2 x 2
##   Method                   RMSE
##   <chr>                   <dbl>
## 1 Average Rating Model 1.061202
## 2 Movie Effect Model   0.9439087
```

The model with the movie and user effects further improves the performance of our predictions:

```
# Compute the predicted ratings based on the test set for this model
model_2_prediction_final <- movie_user_model(edx, validation)

# Compute the corresponding RMSE
model_2_rmse_final <- RMSE(validation$rating, model_2_prediction_final)

# Update the RMSE results table with the RMSE of this model
rmse_results_final <- bind_rows(rmse_results_final, tibble(Method="Movie + User Effects Model",  RMSE =
rmse_results_final
```

```
## # A tibble: 3 x 2
##   Method                          RMSE
##   <chr>                          <dbl>
## 1 Average Rating Model       1.061202
## 2 Movie Effect Model         0.9439087
## 3 Movie + User Effects Model 0.8653488
```

The addition of the genre effect provides a slight improvement:

```
# Compute the predicted ratings based on the validation set for this model
model_3_prediction_final <- movie_user_model_genres(edx, validation)

# Compute the corresponding RMSE
model_3_rmse_final <- RMSE(validation$rating, model_3_prediction_final)

# Update the RMSE results table with the RMSE of this model
rmse_results_final <- bind_rows(rmse_results_final, tibble(Method="Movie + User + Genres Effects Model"
rmse_results_final
```

```
## # A tibble: 4 x 2
##   Method                                    RMSE
##   <chr>                                    <dbl>
## 1 Average Rating Model                1.061202
## 2 Movie Effect Model                  0.9439087
## 3 Movie + User Effects Model          0.8653488
## 4 Movie + User + Genres Effects Model 0.8649469
```

Last, the regularization provides a final improvement over the other models:

```
# Compute the predicted ratings based on the validation set for this model
model_4_prediction_final <- movie_user_model_genres_reg(edx,validation, lambda)

# Compute the corresponding RMSE
model_4_rmse_final <- RMSE(validation$rating, model_4_prediction_final)

# Update the RMSE results table with the RMSE of this model
rmse_results_final <- bind_rows(rmse_results_final, tibble(Method="Movie + User + Genres Effects regula
rmse_results_final
```

```
## # A tibble: 5 x 2
##   Method                                              RMSE
```

```
##   <chr>                                              <dbl>
## 1 Average Rating Model                              1.061202
## 2 Movie Effect Model                                0.9439087
## 3 Movie + User Effects Model                        0.8653488
## 4 Movie + User + Genres Effects Model               0.8649469
## 5 Movie + User + Genres Effects regularized Model 0.8644514
```

Eventually, our last model that includes all the 3 effects and the regularization finally provides the best result with a RMSE of 0.8644514.

```
# Compute and display the gain of the last model vs the basic model in value
rmse_gain_value <- rmse_results_final[[1,"RMSE"]] - rmse_results_final[[5,"RMSE"]]
rmse_gain_perc <- rmse_gain_value/rmse_results_final[[1,"RMSE"]]*100
tibble("RSME gain in value"= rmse_gain_value,"RSME gain in percentage" = rmse_gain_perc)
```

```
## # A tibble: 1 x 2
##   `RSME gain in value` `RSME gain in percentage`
##                  <dbl>                     <dbl>
## 1            0.1967504                  18.54034
```

Which represents a gain of 0.1967 in value and 18.5% vs the "basic model" or the overall rating average of the training set.

# 5. Conclusion

Based on the 10M version of the MovieLens dataset, which contains 1,000,0054 ratings of 10,681 movies by 71,567 users, we have decided to implement a system that would predict the ratings of movie. The main information in that dataset were, movie id, title, release year and genre(s), user id, rating and date of rating.

During the data exploration phase, we have identifed the effects of movie, user and genre on the ratings provided. In order to test the performance of our model(s), we have split our dataset in a training and validation sets and we have set up a function to calculate the residual mean squared error (RMSE) applied to the difference between our model(s) predictions and the actual ratings of our validation set. We have then split our training set in a smaller training set and a test set that we used to build, fine-tune and compare several models including these different effects. Last, we have applied regularization to our best performing model to limit the impact of large rating estimates that come from small sample sizes.

Finally, we have tested our models on the validation set. The best model that included all the effects and the regularization managed to reach a RMSE of 0.8644514, which represented a gain of 18,5% comparing to the performance of the simpliest model (using the average rating of the training set).

Although it represents a significant gain in performance, one can consider that the margin for error in rating is still important for a 0-5 rating system. One way to improve this performance could be to work on the largest dataset availble, which would require more important computing power.

Another signifiance limit is that this system will not work on predicting new movies that have never been rated before. To handle them, we would cocould conher methods or systems such as content-based or collaborative filtering models.