Development of a Movie Rating Prediction Model

Alumni of the HarvardX Data Science Professional Certificate Program

2023-09-06

Executive summary

This project is an assignment of the "MovieLens Project", of "Data Science: Capstone" course. The main objective of this project is to develop a movie recommendation system using the 10M version of the MovieLens dataset. The project has consisted of: Cleaning and preprocessing of historical recommendation data; Selection of explanatory variables; Application of the regularization process for the design and generation of the model, and finally the evaluation of the model performance by means of the RMSE. For generating the model, five explanatory variables were used (movie name, user, year of the movie and rating, movie genre), followed by a regularization algorithm, in a random subset of 90% of the data. Finally, the model was evaluated on the remaining 10% of the data (not used in the model generation), resulting in a RMSE of **RMSE = 0.8638922**.

1. Introduction

1.1 Objective and context

This project is an assignment of the "MovieLens Project Submission" of "Data Science: Capstone" (HarvardX PH125.9x) course, and its objective is to develop a movie recommendation system designed on the basis of knowledge extracted from past behavior. These algorithms are typically used in both websites and video streaming platforms to provide personalized recommendations and finally increasing user retention by means of improving its experience. Consequently, these systems constitute a key part of the digital entertainment services, allowing the user to discover new movies based on their personal tastes.

1.2 Data

The movielens dataset is an initiative created by the University of Minnesota GroupLens lab to study recommendation system. The data set used in this project is the "MovieLens 10M Dataset", publicly available at https://files.grouplens.org/datasets/movielens/ml-10m.zip. This dataset contains 10000054 ratings and 95580 tags applied to 10681 movies, carried out by 71567 users of the online movie recommender service MovieLens. The database has been generated with manual entries, even if they should be entered identically to those found in IMDB, so errors and inconsistencies may exist and should be checked. Users, represented by an id, were selected at random for inclusion. Data used for this project are movies.dat, ratings.dat (i.e., no tags are used).

We use the code provided by the course to split the original dataset in the following ones:

- the *edx* set is used to develop our algorithm (train dataset)
- the *final_holdout_test*, not used to develop the algorithm, is used for a final test of our final algorithm in predicting movie ratings (test dataset)

1.3 Key steps

The project consists of the following steps:

- Cleaning and preprocessing of historical recommendation data
  - The database used has been generated with manual entries, so errors and inconsistencies may exist and should be checked and cleaned.
  - Data format definition: numbers, strings, and dates
  - Generation of new variables, as movie year
- Model design
  - Search and selection of relevant characteristics to be used as explanatory variables in the prediction of new ratings by our model.
  - The train dataset, *edx*, will be further subdivided into two datasets to evaluate the suitability of including such variables, based on their effect on the RMSE reduction.
- Final model generation
  - The selected variables will be used with all available data to generate the model, i.e. the *edx* dataset
  - Finally, a regularization process will be applied to minimize the loss function (RMSE) and prevent overfitting or underfitting.
- Model evaluation
  - The performance of the final model will be tested in the *final_holdout_test*, which has not been used for the generation of the prediction model.

2. Methods and Analysis

2.1 Data download and preparation

In this subsection We download the original database, and split it into the following subsets: - *edx* dataset: used to develop our algorithm - *final_holdout_test*: used for a final test of our final algorithm

Install and load packages

```
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(ggpubr)) install.packages("ggpubr", repos = "http://cran.us.r-project.org")

library(tidyverse); library(caret); library(ggpubr)
```

The following options sets as 120s the amount of time to wait for a response from the remote name server before retrying the query via a different one.

```
options(timeout = 120)
```

Download and unzip the original dataset, MovieLens 10M.

```
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Unzip ratings and movies files from the original dataset
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)
movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)
```

Generate and format dataframes with ratings and movies, and Use left_join fucntion to add columns from ratings to movies, keeping all observations in ratings

```
# Create a dataframe with ratings
ratings <- as.data.frame(
  str_split(read_lines(ratings_file),fixed("::"), simplify = TRUE),
  stringsAsFactors = FALSE)

# Define column names and format
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

# Create a dataframe with movies
movies <- as.data.frame(
  str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)

# Define column names and format
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

# Join both ratings and movies, keeping all observations in ratings
movielens <- left_join(ratings, movies, by = "movieId")
```

Split the *movielens* data frame into training (*edx*, 90% of data) and testing (*final_holdout_test*, 10% of data) datasets for model generation and testing, respectively.

```
set.seed(1, sample.kind="Rounding") #if using R 3.6 or later
#set.seed(1) #if using R 3.5 or earlier
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 final hold-out test set are also in *edx* set

```
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

Add rows removed from final hold-out test set back into *edx* set

```
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)
```

Finally, remove data that will not be used in the model generation

```
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

The year of the movie is not directly provided in the dataset, but we can extract it from the title column, as all the movie titles end with the year in parentheses.

Note that, in some few cases (175 out of 9000055, a 0.002%), the difference rating minus movie year is negative, which is an error probably due to manual data entry: it is not logical for a film to have been rated before its year of release.

```
edx %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(rating_year = year(date2),
         movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(year_diff = rating_year - movie_year) %>%
  mutate(year_diff_negative = ifelse(year_diff < 0,
                                     "Rating - movie year < 0",
                                     "Rating - movie year >= 0")) %>%
  {table(.$year_diff_negative)}
```

```
##
##  Rating - movie year < 0 Rating - movie year >= 0
##                      175                  8999880
```

These data, even if they are few, are cleaned by setting them to zero (ratings minus movie years ranges from -1 to -2) as follows

```
edx <- edx %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  mutate(diff_year_movie_rating = ifelse(diff_year_movie_rating < 0, 0, diff_year_movie_rating))
```

2.2 Data exploration and visualization

In this Section, we will explore the dataset and the relations between its parameters, especially ratings.

Our dataset is composed by 6 columns, with 9000055 rows

```
dim(edx)
```

```
## [1] 9000055      10
```

Let's have a look to the dataset

```
head(edx)
```

```
##   userId movieId rating timestamp                          title
## 1      1     122      5 838985046               Boomerang (1992)
## 2      1     185      5 838983525                Net, The (1995)
## 4      1     292      5 838983421                Outbreak (1995)
## 5      1     316      5 838983392                Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                         genres               date2 movie_year rating_year
## 1                Comedy|Romance 1996-08-02 11:24:06       1992        1996
## 2          Action|Crime|Thriller 1996-08-02 10:58:45       1995        1996
## 4  Action|Drama|Sci-Fi|Thriller 1996-08-02 10:57:01       1995        1996
## 5       Action|Adventure|Sci-Fi 1996-08-02 10:56:32       1994        1996
## 6 Action|Adventure|Drama|Sci-Fi 1996-08-02 10:56:32       1994        1996
## 7       Children|Comedy|Fantasy 1996-08-02 11:14:34       1994        1996
##   diff_year_movie_rating
## 1                      4
## 2                      1
## 4                      1
## 5                      2
## 6                      2
## 7                      2
```

The different users, movies, as well as time ranges of movies and ratings are calculated as follows, revealing 69878 unique users, 10677 unique movies, and 797 unique geners (and combinations of genres). In addition, movie years range from 1915 to 2008, and rating years from 1995 to 2009.

Note that movie year, not available in the dataset, has been extracted from the "title" column, as all the movie titles end with the year in parentheses.

```
edx %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)),
         rating_year = year(date2))  %>%
  summarize('Unique users' = n_distinct(userId),
            'Unique movies' = n_distinct(movieId),
            'Genres (and combinations)' = n_distinct(genres),
            'Movie years' = paste("From ", min(movie_year),
                                  " to ", max(movie_year), sep=""),
            'Rating years' = paste("From ", min(rating_year),
                                   " to ", max(rating_year), sep=""))
```

```
##   Unique users Unique movies Genres (and combinations)       Movie years
## 1        69878         10677                       797 From 1915 to 2008
##        Rating years
## 1 From 1995 to 2009
```

2.2.1 User id

In this Subsection, we will explore the number of ratings, and ratings averages, of the different Users available in the dataset.

Top and bottom users by number of ratings: 2 users have more then 6000 ratings, while 4 users have less than 14 ratings

```
num_rat_by_user_id <- edx %>% group_by(userId) %>% summarise(n = n()) %>% arrange(desc(n))
# Top 5
head(num_rat_by_user_id, 5)
```

```
## # A tibble: 5 x 2
##   userId     n
##    <int> <int>
## 1  59269  6616
## 2  67385  6360
## 3  14463  4648
## 4  68259  4036
## 5  27468  4023
```

```
# Bottom 5
tail(num_rat_by_user_id, 5)
```

```
## # A tibble: 5 x 2
##   userId     n
##    <int> <int>
## 1  71344    14
## 2  15719    13
## 3  50608    13
## 4  22170    12
## 5  62516    10
```

Let's explore the distribution of data by means of their quantiles. The mean (128.8) is is more than twice the median (62.0). Also, the 3rd quantile (141.0) is farther away from the median than the 1st quantile (32.0). All this indicates an asymmetric distribution to the right, with many data grouped at low values.

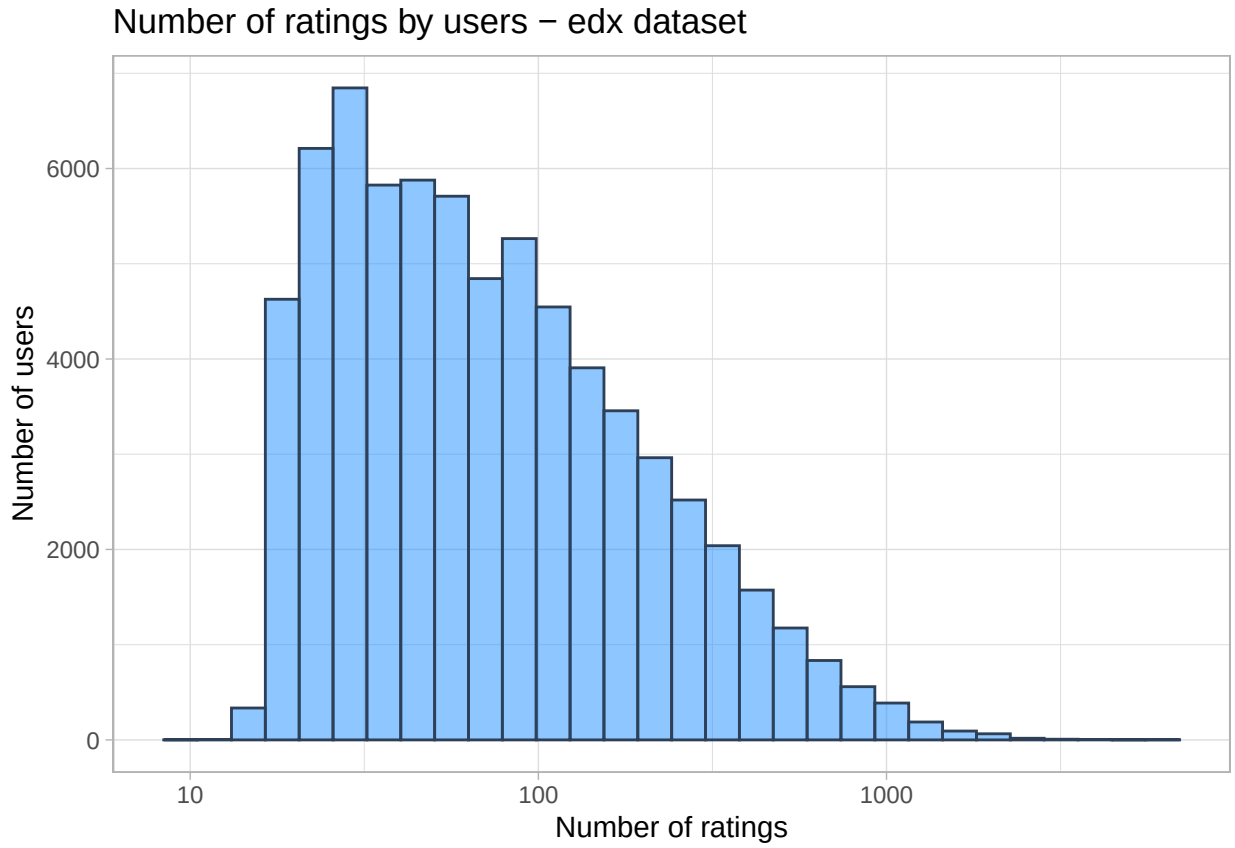
Filling gaps in the dataset is not a good strategy, as 75% of the users have no ratings in more than 98.6% of the movies (P75 = 141, number of different movies = 10677).

```
num_rat_by_user_id %>% summary(count)
```

```
##      userId            n
##  Min.   :    1   Min.   :  10.0
##  1st Qu.:17943   1st Qu.:  32.0
##  Median :35799   Median :  62.0
##  Mean   :35782   Mean   : 128.8
##  3rd Qu.:53620   3rd Qu.: 141.0
##  Max.   :71567   Max.   :6616.0
```

This asymmetry can be seen in the histogram of values.

```
edx %>% group_by(userId) %>%
  summarize(count=n()) %>%
  ggplot(aes(x = count)) +
  geom_histogram(color = "#2e4057", fill="dodgerblue", alpha = 0.5) +
  scale_x_log10() + # Convenient for visual
  ggtitle("Number of ratings by users - edx dataset") +
  xlab("Number of ratings") +
  ylab("Number of users") +
  theme_light()
```



Let's sort Users by their rating average, exploring their top and bottom values. Top users have averaged ratings higher than 4.2, while bottom ones have less than 2.

```
avg_rat_by_user_id <- edx %>% group_by(userId) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000) %>%
  arrange(desc(avg))

# Top and bottom users by number rating average
# Top 5
head(avg_rat_by_user_id, 5)
```

```
## # A tibble: 5 x 4
##   userId     n   avg     se
##    <int> <int> <dbl>  <dbl>
```

```
## 1  32463  1247  4.60 0.0195
## 2  37450  1126  4.30 0.0219
## 3  28832  1232  4.24 0.0190
## 4  43015  1066  4.24 0.0229
## 5  19859  1156  4.23 0.0185
```

```
# Bottom 5
tail(avg_rat_by_user_id, 5)
```

```
## # A tibble: 5 x 4
##   userId     n   avg     se
##    <int> <int> <dbl>  <dbl>
## 1  18169  1195  2.03 0.0293
## 2  17196  1216  1.99 0.0308
## 3  59326  1026  1.93 0.0317
## 4  23159  1217  1.80 0.0321
## 5  30272  1393  1.55 0.0258
```

Note that only users with at least 1000 ratings have been considered

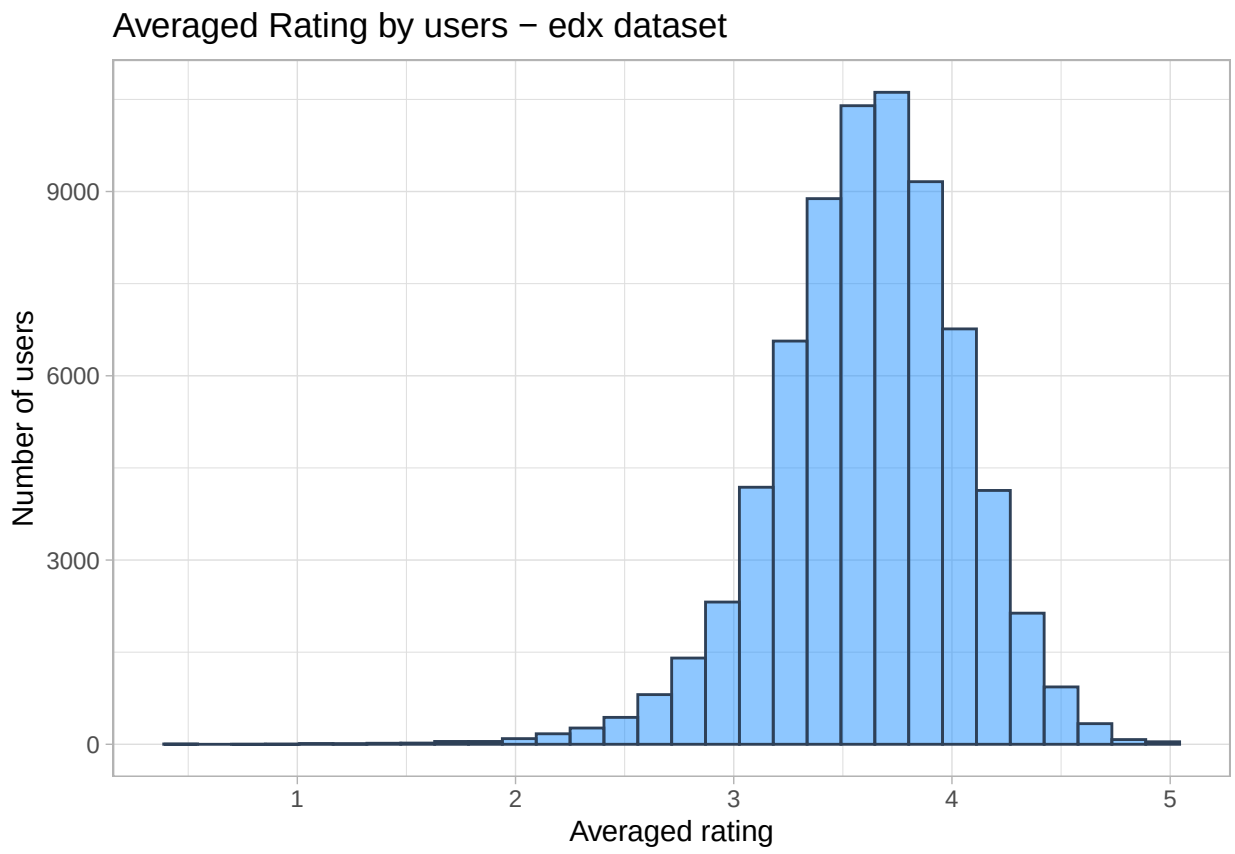
Let's explore the distribution of averaged ratings by means of their quantiles. The mean (3.24) is close to the median (3.26). Also, the 3rd quantile (3.51) is slightly closer to the median than the 1st quantile (2.99). All this indicates a quasi-symmetric distribution, centered at 3.5, with a few more data further to the left

```
avg_rat_by_user_id %>% summary(count)
```

```
##      userId            n             avg              se
##  Min.   :  182   Min.   :1000   Min.   :1.551   Min.   :0.007852
##  1st Qu.:18799   1st Qu.:1098   1st Qu.:2.987   1st Qu.:0.021807
##  Median :33905   Median :1247   Median :3.258   Median :0.025792
##  Mean   :34992   Mean   :1419   Mean   :3.237   Mean   :0.026429
##  3rd Qu.:51048   3rd Qu.:1526   3rd Qu.:3.506   3rd Qu.:0.031039
##  Max.   :71509   Max.   :6616   Max.   :4.598   Max.   :0.049396
```

Let's explore the distribution of averaged ratings among users.

```
edx %>% group_by(userId) %>%
  summarize(avg = mean(rating)) %>%
  ggplot(aes(avg)) +
  geom_histogram(color = "#2e4057", fill="dodgerblue", alpha = 0.5) +
  ggtitle("Averaged Rating by users - edx dataset") +
  xlab("Averaged rating") +
  ylab("Number of users") +
  theme_light()
```



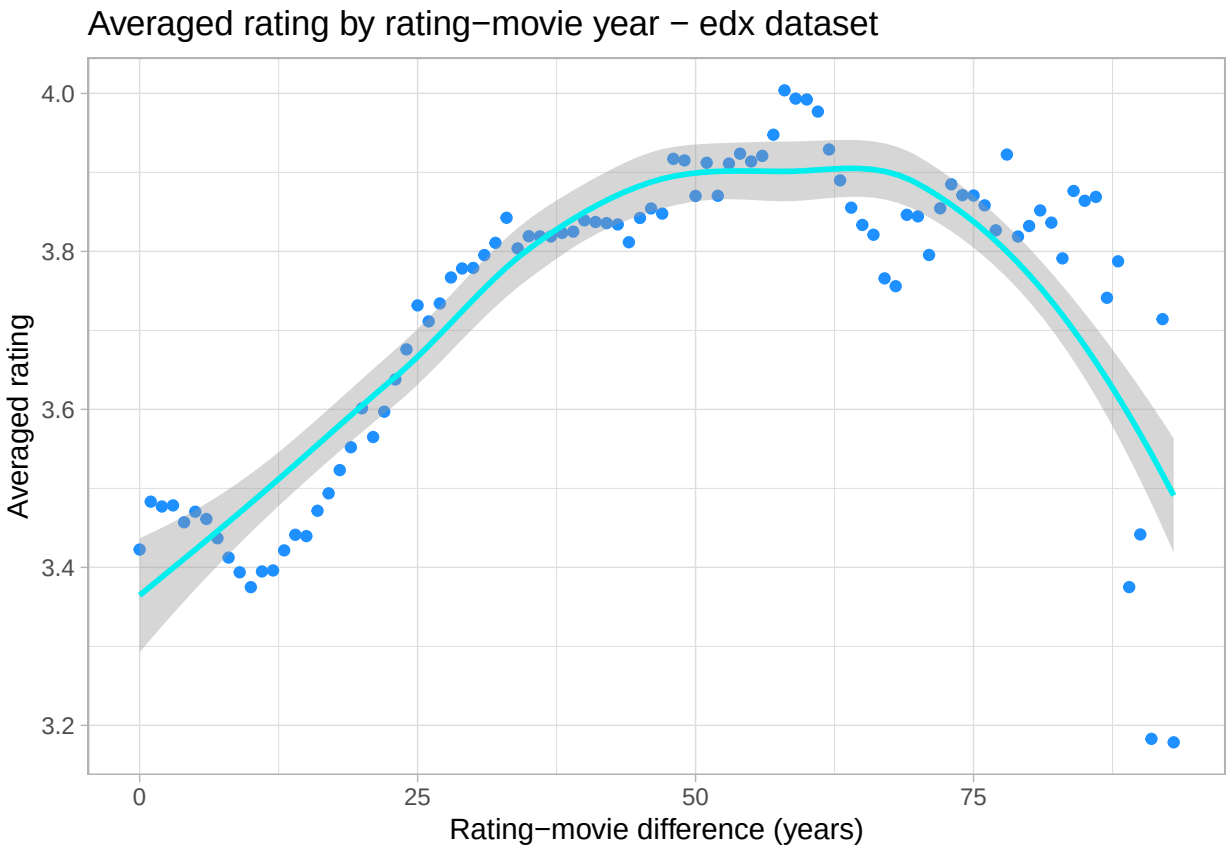
2.2.2 Movie and rating years

We will explore the averaged rating as a function of the difference between the rating and the movie year. This effect is evaluated due to the possible tendency to value older films more, either because we consider them to be good classics or because we only have access to those classic films of high quality.

The year of the movie is not directly provided in the dataset, but we can extract it from the title column, as all the movie titles end with the year in parentheses, as stated in previous subsection.

The following figure shows the rating average as a function of the difference between the rating and the movie year.

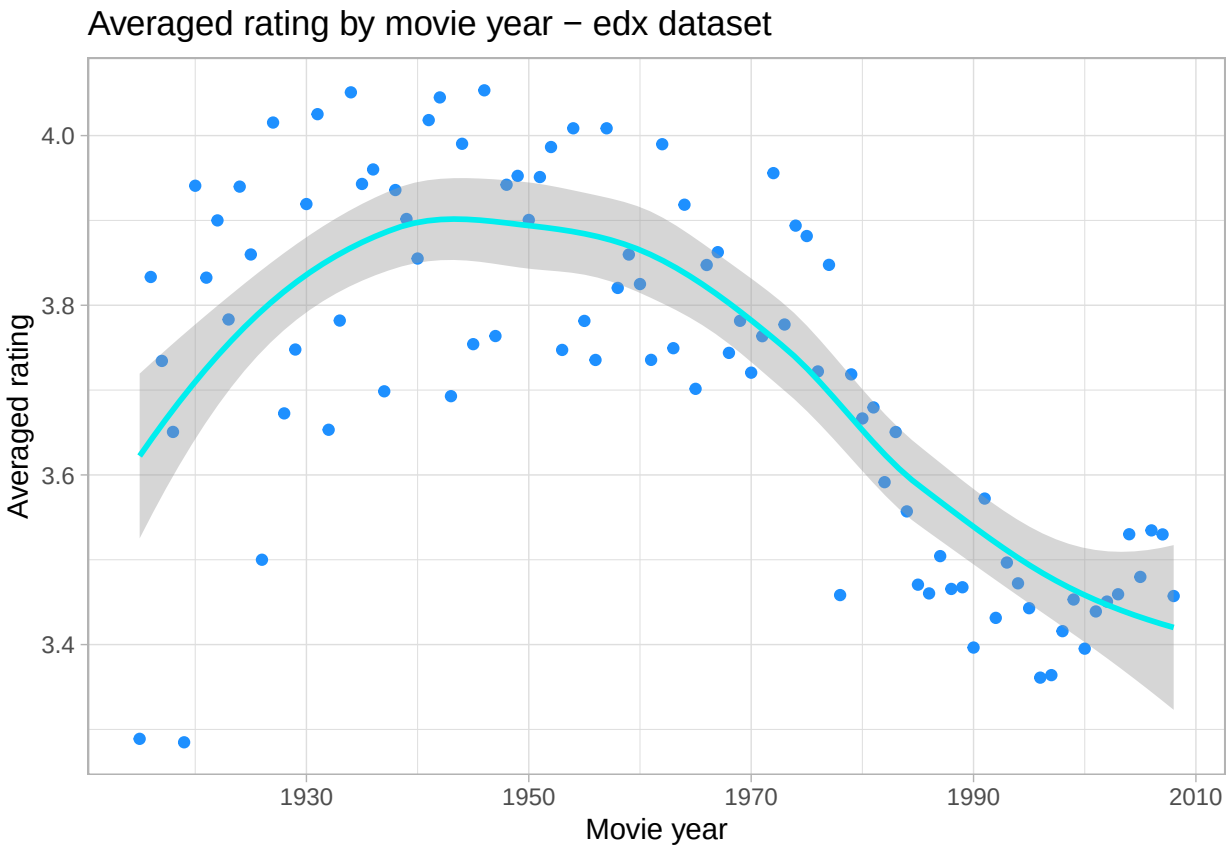
```
edx %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(rating_year = year(date2),
         movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(year_diff = rating_year - movie_year) %>%
  filter(year_diff >= 0) %>%
  group_by(year_diff) %>%
  summarise(ratings = mean(rating)) %>%
  ggplot(aes(x=year_diff, y = ratings)) +
  geom_point(color = "dodgerblue") +
  ggtitle("Averaged rating by rating-movie year - edx dataset") +
  xlab("Rating-movie difference (years)") +
  ylab("Averaged rating") +
  geom_smooth(color = "cyan2") + theme_light()
```



This graph reveals a clear influence of this parameter (difference between rating and movie year) in the averaged rating: years differences < 7 years lead to averaged ratings < 3.5, which increases up to 4 for years differences of 58 years. Also, the graph reveals a low dispersion

Let's explore the rating averages and number of ratings as a function of the year of the movie.

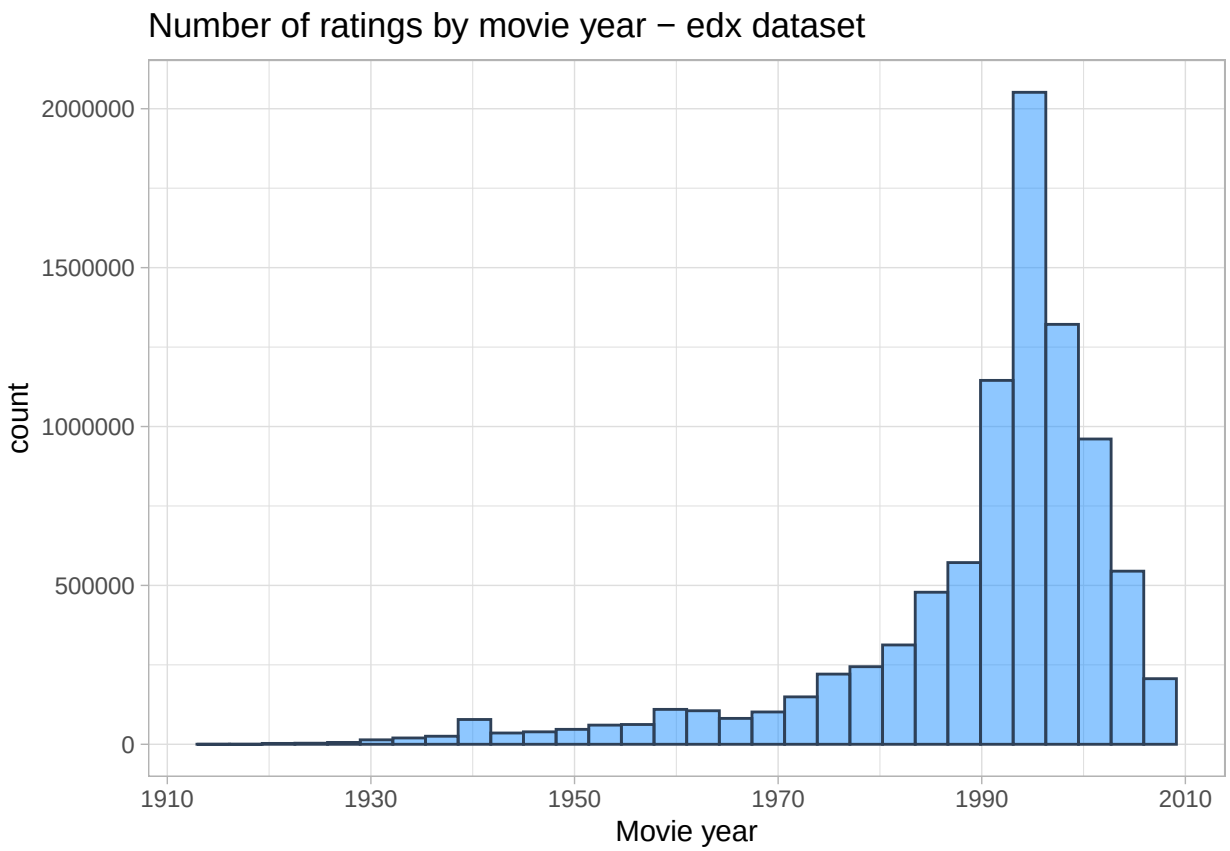
```
edx %>% mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(
    substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  group_by(movie_year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(movie_year, rating)) +
  geom_point(color = "dodgerblue") + geom_smooth(color = "cyan2") +
  ggtitle("Averaged rating by movie year - edx dataset") +
  xlab("Movie year") +
  ylab("Averaged rating") + theme_light()
```



The averaged rating increases from 3.61 before 1920, up to 3.9 in the 40's (with high dispersion). Then, it continuously decreases to 3.41 in 2008, with lower dispersion

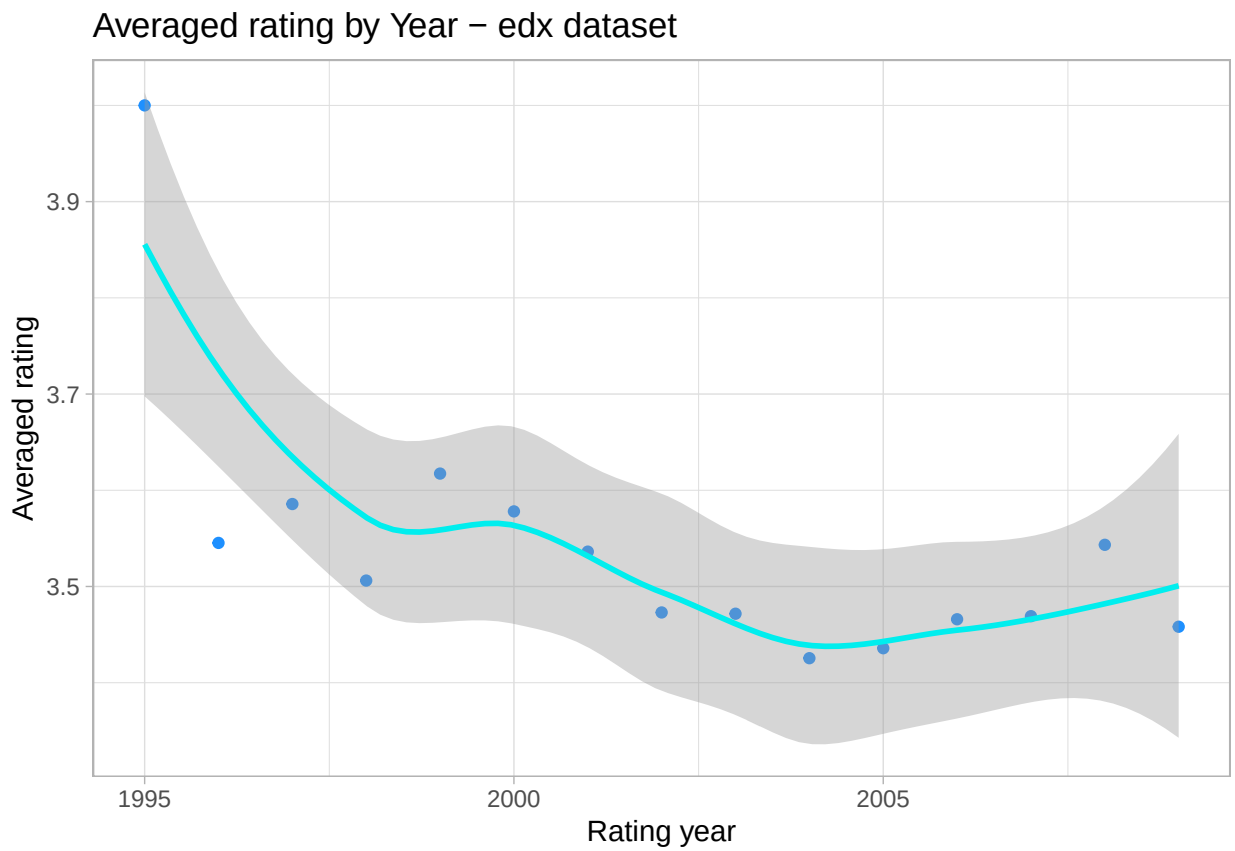
The number of ratings per movie year is shown in the following histogram. Movies from the 90's have received the highest number of ratings

```
edx %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(
    substr(title, nchar(title)-5+1, nchar(title)-1))) %>%
  ggplot(aes(x=movie_year), ratings) +
  geom_histogram(color = "#2e4057", fill="dodgerblue", alpha = 0.5) +
  ggtitle("Number of ratings by movie year - edx dataset") +
  xlab("Movie year") +
  theme_light()
```



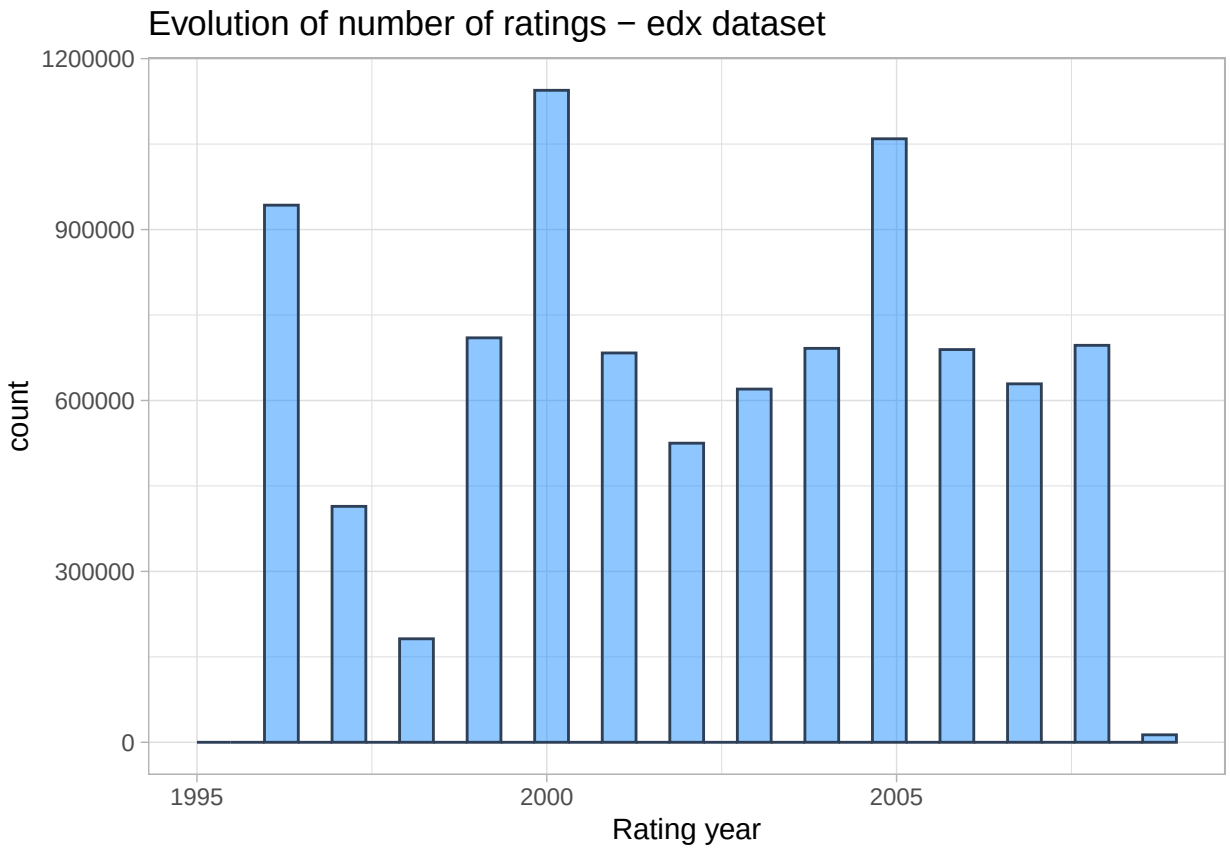
Let's explore how the averaged rating evolves year by year

```
edx %>% mutate(date2 = as_datetime(timestamp)) %>%
  mutate(rating_year = year(date2))  %>%
  group_by(rating_year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(rating_year, rating)) +
  geom_point(color = "dodgerblue") + geom_smooth(color = "cyan2") +
  ggtitle("Averaged rating by Year - edx dataset") +
  xlab("Rating year") +
  ylab("Averaged rating") + theme_light()
```



The following histogram shows the number of ratings by year, revealing an increase from the 2000's

```
edx %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(rating_year = year(date2))  %>%
  ggplot(aes(x=rating_year), ratings) +
  geom_histogram(color = "#2e4057", fill="dodgerblue", alpha = 0.5) +
  xlab("Rating year") +
  ggtitle("Evolution of number of ratings - edx dataset") +
  theme_light()
```



2.2.3 Genres

We have seen above that in our *edx* dataset there are 797 different genres, or combinations of genres. The following code shows the number of movies with different number of genres, where it can be seen that the highest number of movies (2721164 movies) have a combination of 3 genres, followed by 2 (2637775 movies) and 1 genre (1740569 movies). On the contrary, only 256 films present a combination of 8 genres.

```
edx %>% mutate(
  Number_of_genres = 1 + str_count(genres, fixed("|"))) %>%
  group_by(Number_of_genres) %>% summarise(Movies = n()) %>%
  arrange(desc(Movies))
```

```
## # A tibble: 8 x 2
##   Number_of_genres  Movies
##              <dbl>   <int>
## 1                3 2721164
## 2                2 2637775
## 3                1 1740569
## 4                4 1401379
## 5                5  421069
## 6                6   65998
## 7                7   11845
## 8                8     256
```

To check which are the genres, or combinations of genres, with the highest number of movies, we run the following code, revealing that Drama is the most common genre (733296 movies), followed by Comedy (700889 movies). The next three are different combinations of Comedy.

```
edx %>% mutate(
  Number_of_combinatios = str_count(genres, fixed("|"))) %>%
  group_by(Number_of_combinatios, genres) %>% summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 797 x 3
## # Groups:   Number_of_combinatios [8]
##    Number_of_combinatios genres                          n
##                    <int> <chr>                       <int>
##  1                     0 Drama                      733296
##  2                     0 Comedy                     700889
##  3                     1 Comedy|Romance             365468
##  4                     1 Comedy|Drama               323637
##  5                     2 Comedy|Drama|Romance       261425
##  6                     1 Drama|Romance              259355
##  7                     2 Action|Adventure|Sci-Fi    219938
##  8                     2 Action|Adventure|Thriller  149091
##  9                     1 Drama|Thriller             145373
## 10                     1 Crime|Drama                137387
## # i 787 more rows
```

There is a wide disparity between the averaged ratings of the different genres (or combinations of genres), as can be seen in the following left chart, where genres names have been replaced by numbers, to facilitate visualization due to the large number of different genres. The right chart shows the averaged rating on thise genres with at least 10000 ratings.

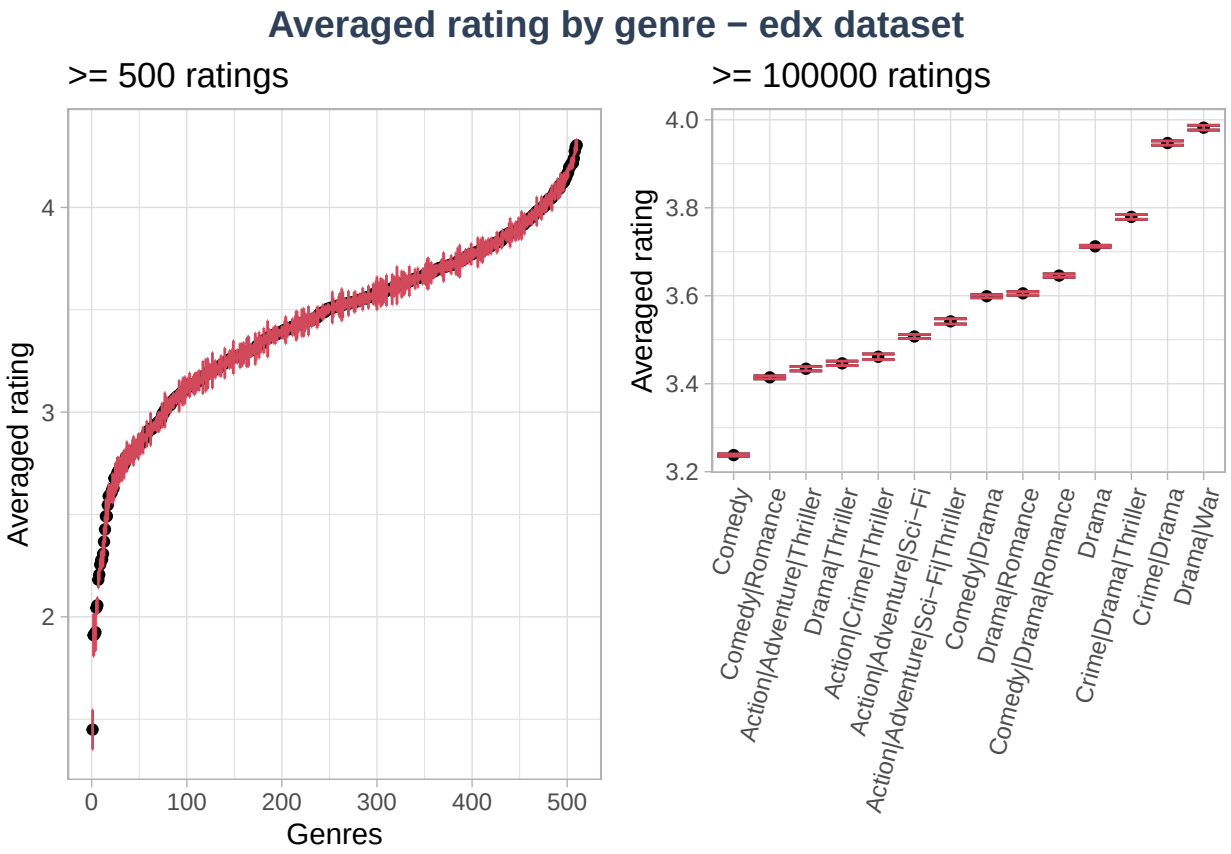
```
genres_graph_filter_500 <- edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 500) %>%
  mutate(genres = reorder(genres, avg)) %>%
  mutate(genres = as.numeric(genres)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar(color = "#d1495b") +
  ggtitle(">= 500 ratings") +
  xlab("Genres") +
  ylab("Averaged rating") +
  theme_light()

genres_graph_filter_100000 <-edx %>% group_by(genres) %>%
  summarize(n = n(),
            avg = mean(rating),
            se = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg,
             ymin = avg - 2*se, ymax = avg + 2*se)) + geom_point() +
  geom_errorbar() +
  geom_errorbar(color = "#d1495b") +
```

```
  ggtitle(">= 100000 ratings") +
  xlab("") +
  ylab("Averaged rating") +
  theme_light() +
  theme(axis.text.x = element_text(angle = 75, hjust = 1))

# Let's combine both graphs
genres_plot <- ggarrange(genres_graph_filter_500, genres_graph_filter_100000)

# And include a title
annotate_figure(genres_plot,
                top = text_grob("Averaged rating by genre - edx dataset",
                                color = "#2e4057",
                                face = "bold", size = 14))
```



2.3 Modeling approach

In this subsection, we check the performance of different variables in the rating prediction, for which we will exclusively use the *edx* dataset.

For evaluating the effectiveness of that variables, we will split the *edx* dataset into subsequent Test and a Train datasets as follows:

```
# Data partition
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,list = FALSE)
# Define Train and Test datasets from data partition
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

We must make sure that we do not include users and movies in the evaluation set that do not appear in the training set: We remove these entries using the semi_join function:

```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

If we only consider the averaged movie rating,

```
mu <- mean(train_set$rating)
rmse_model_base_model <- RMSE(test_set$rating, mu)
rmse_model_base_model
```

```
## [1] 1.060704
```

We now evaluate the goodness of fit of the different variables to be included in the model.

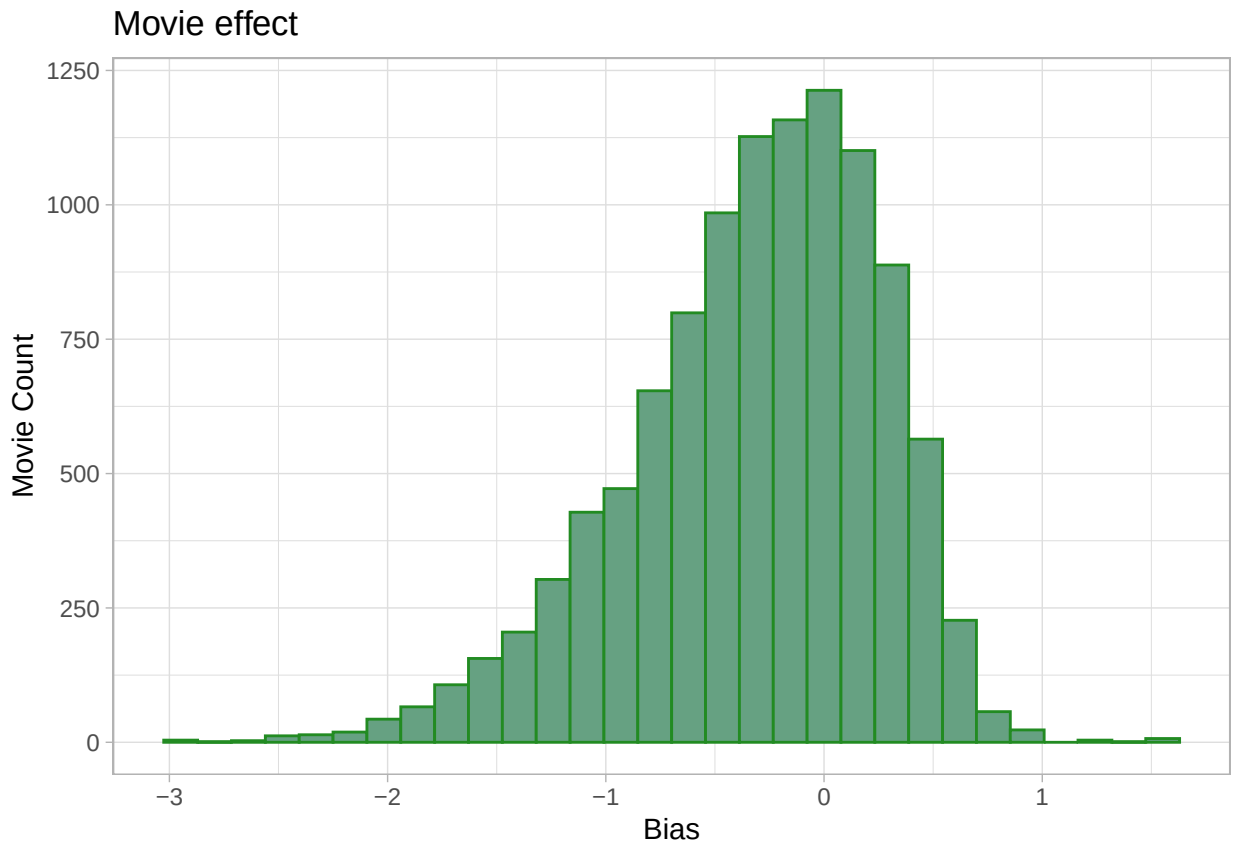
2.3.1 Movie effect

Given the number of different moviesd (10677) and useres (69878), we will not carry out a regression. Being mu the averaged value of all ratings, we will Calculate b_i as the the averaged difference between movies rating and the averaged value of all movies.

```
movie_diff_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

The bias distribution between rating and movie average is asymmetric and shows a great dispersion:

```
train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) %>%
  ggplot(aes(x = b_i)) +
  geom_histogram(color = "forestgreen", fill="#66a182") +
  ggtitle("Movie effect") +
  xlab("Bias") +
  ylab("Movie Count") +
  theme_light()
```



Predicted ratings in the test dataset only with Movie effect, and its RMSE, are calculated as follows:

```
predicted_ratings_bi <- mu + test_set %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  .$b_i # Extract only the predicted values

rmse_model_bi <- RMSE(test_set$rating,predicted_ratings_bi)
rmse_model_bi
```

```
## [1] 0.9437144
```

This initial model provides a marked reduction in RMSE, which is to be expected because the model only takes into account averages and is not capable of describing the full complexity of the phenomenon to be described.

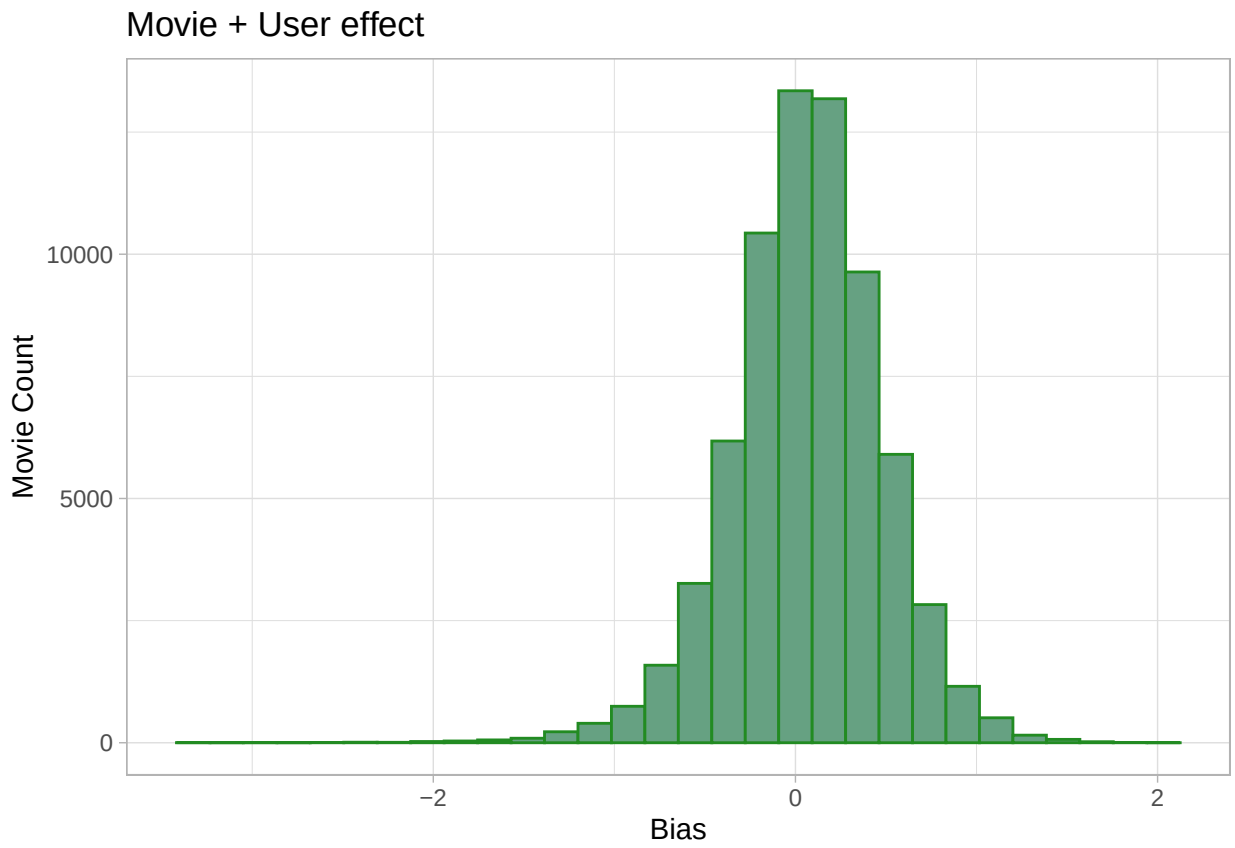
2.3.2 Movie + user effect

Calculate b_u, the user effect, using the training set

```
user_avgs <- train_set %>%
  # Include the previous movie averaged differences dataframe
  left_join(movie_diff_avgs, by='movieId') %>%
  # Consider different IDs
  group_by(userId) %>%
  # Calculate b_u = averaged difference between rating, averaged rating and b_i
  summarize(b_u = mean(rating - mu - b_i))
```

The bias distribution between rating and movie+user average is symmetric, and has lower dispersion

```
train_set %>%
  # Include the previous movie averaged differences dataframe
  left_join(movie_diff_avgs, by='movieId') %>%
  # Consider different IDs
  group_by(userId) %>%
  # Calculate b_u = averaged difference between rating, averaged rating and b_i
  summarize(b_u = mean(rating - mu - b_i)) %>%
  ggplot(aes(x = b_u)) +
  geom_histogram(color = "forestgreen", fill="#66a182") +
  ggtitle("Movie + User effect") +
  xlab("Bias") +
  ylab("Movie Count") +
  theme_light()
```



Check the model performance

```
predicted_ratings_bu <- test_set %>%
  # Include the previous movie averaged differences dataframe
  left_join(movie_diff_avgs, by='movieId') %>%
  # Include the previous userId averaged differences dataframe
  left_join(user_avgs, by='userId') %>%
  # calculate the predicted values
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

This new model, including both user and movie effects, provides a marked decrease in RMSE with respect to the model that only includes the movie effect

```
rmse_model_bi_bu <- RMSE(test_set$rating,predicted_ratings_bu)
rmse_model_bi_bu
```

```
## [1] 0.8661625
```

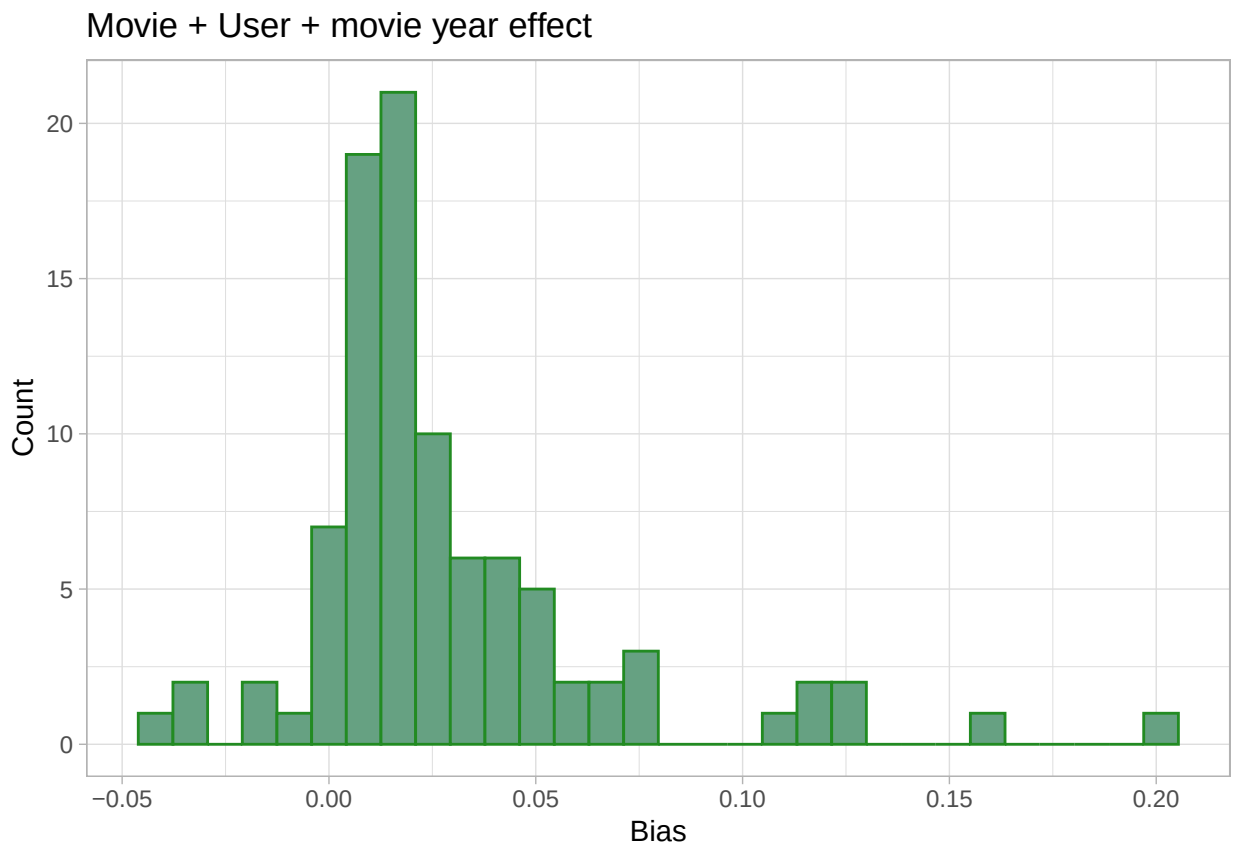
2.3.3 Movie + user effect + movie year

Calculate movie year effects (b_my), using the training set

```
movie_year_avgs <- train_set %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(
    substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(movie_year) %>%
  summarize(b_my = mean(rating - mu - b_i - b_u))
```

The bias distribution between rating and movie+user+movie year average is reduced with respect to the previous ones.

```
train_set %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(
    substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(movie_year) %>%
  summarize(m_y = mean(rating - mu - b_i - b_u)) %>%
  ggplot(aes(m_y)) +
    geom_histogram(color = "forestgreen", fill="#66a182") +
  ggtitle("Movie + User + movie year effect") +
  xlab("Bias") +
  ylab("Count") +
  theme_light()
```



Check the model performance

```
predicted_ratings_my <- test_set %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_year_avgs, by='movie_year') %>%
  mutate(pred = mu + b_i + b_u + b_my) %>%
  .$pred
```

This new model provides a slight reduction in RMSE

```
rmse_model_bi_bu_my <- RMSE(test_set$rating,predicted_ratings_my)
rmse_model_bi_bu_my
```

```
## [1] 0.8658322
```

2.3.4 Movie + user effect + movie year + difference movie vs rating year

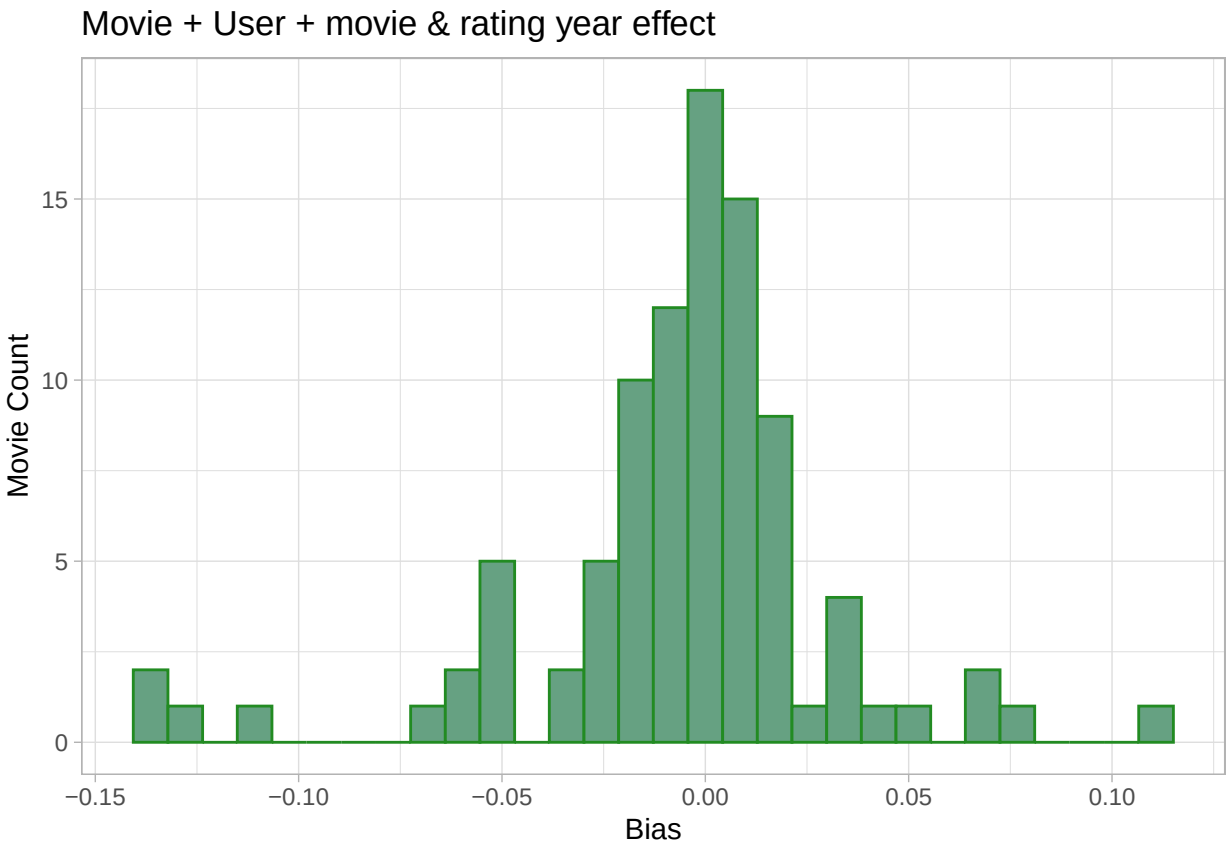
Calculate movie year effects (b_my), using the training set

```
movie_diff_y_avgs <- train_set %>%
  mutate(date2 = as_datetime(timestamp)) %>%
```

```
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  mutate(diff_year_movie_rating = ifelse(diff_year_movie_rating < 0,
                                         0, diff_year_movie_rating)) %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_year_avgs, by='movie_year') %>%
  group_by(diff_year_movie_rating) %>%
  summarize(b_diff_y = mean(rating - mu - b_i - b_u - b_my))
```

With few outliers, there is a symmetric distribution of bias with low dispersion

```
train_set %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  # trend to provide higher ratings to "classic" movies
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  mutate(diff_year_movie_rating = ifelse(diff_year_movie_rating < 0, 0,
                                         diff_year_movie_rating)) %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_year_avgs, by='movie_year') %>%
  group_by(diff_year_movie_rating) %>%
  summarize(b_diff_y = mean(rating - mu - b_i - b_u - b_my)) %>%
  ggplot(aes(x = b_diff_y)) +
  geom_histogram(color = "forestgreen", fill="#66a182") +
  ggtitle("Movie + User + movie & rating year effect") +
  xlab("Bias") +
  ylab("Movie Count") +
  theme_light()
```



Check the model performance

```
predicted_ratings_my <- test_set %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  mutate(diff_year_movie_rating = ifelse(diff_year_movie_rating < 0, 0,
                                         diff_year_movie_rating)) %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_year_avgs, by='movie_year') %>%
  left_join(movie_diff_y_avgs, by='diff_year_movie_rating') %>%
  mutate(pred = mu + b_i + b_u + b_my + b_diff_y) %>%
  .$pred
```

This new model provides a slight reduction in RMSE

```
rmse_model_bi_bu_my_b_diff_y <- RMSE(test_set$rating,predicted_ratings_my)
rmse_model_bi_bu_my_b_diff_y
```

```
## [1] 0.8655143
```

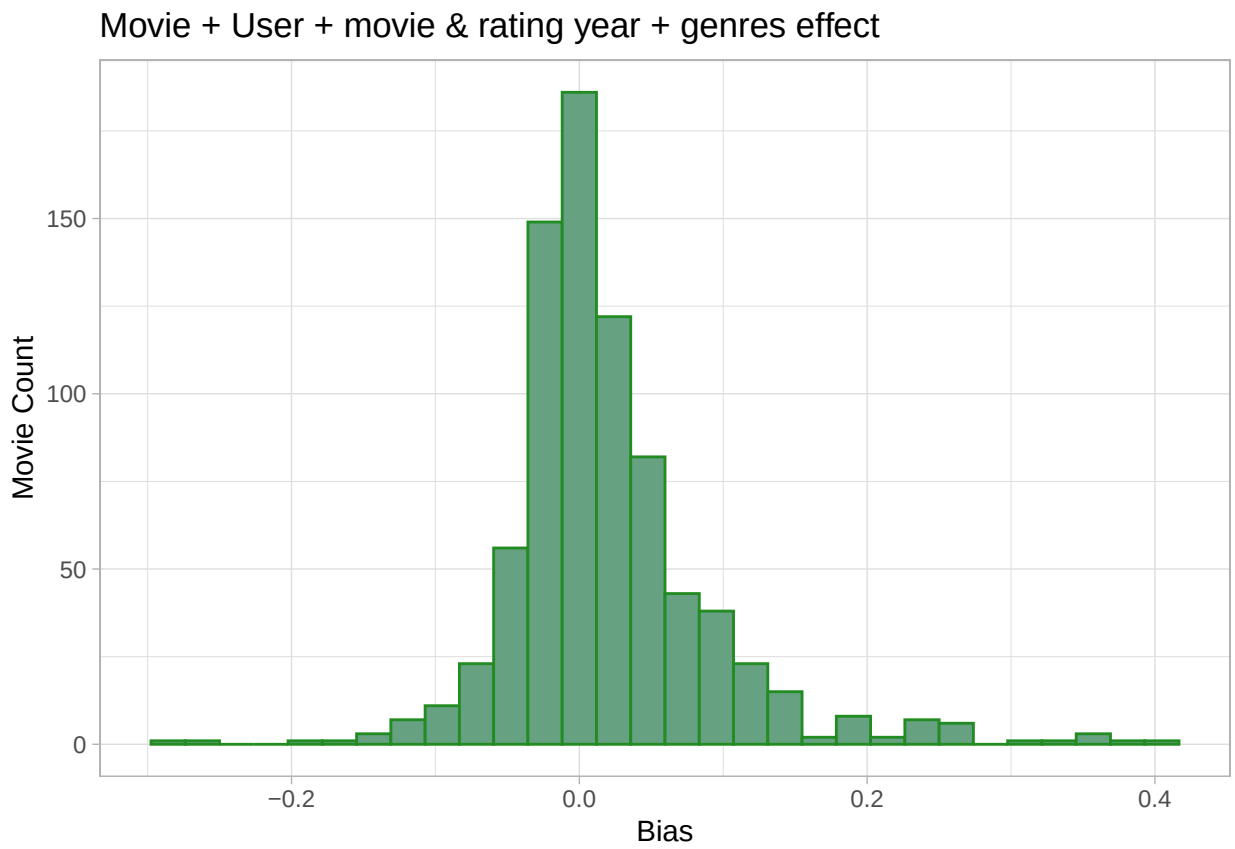
2.3.5 Movie + user effect + movie year + difference movie vs rating year + Genre

Calculate Genres effects (b_g), using the training set

```
movie_g_avgs <- train_set %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(
    substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  # trend to provide higher ratings to "classic" movies
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  mutate(diff_year_movie_rating = ifelse(diff_year_movie_rating < 0, 0,
                                         diff_year_movie_rating)) %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_year_avgs, by='movie_year') %>%
  left_join(movie_diff_y_avgs, by='diff_year_movie_rating') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_my - b_diff_y))
```

With few outliers, there is a slight right assymetric distribution of bias with dispersion

```
train_set %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  mutate(diff_year_movie_rating = ifelse(diff_year_movie_rating < 0, 0,
                                         diff_year_movie_rating)) %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_year_avgs, by='movie_year') %>%
  left_join(movie_diff_y_avgs, by='diff_year_movie_rating') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_my - b_diff_y)) %>%
  ggplot(aes(x = b_g)) +
  geom_histogram(color = "forestgreen", fill="#66a182") +
  ggtitle("Movie + User + movie & rating year + genres effect") +
  xlab("Bias") +
  ylab("Movie Count") +
  theme_light()
```



Check the model performance

```
predicted_ratings_g <- test_set %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  mutate(diff_year_movie_rating = ifelse(diff_year_movie_rating < 0, 0,
                                         diff_year_movie_rating)) %>%
  left_join(movie_diff_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(movie_year_avgs, by='movie_year') %>%
  left_join(movie_diff_y_avgs, by='diff_year_movie_rating') %>%
  left_join(movie_g_avgs, by='genres') %>%

  mutate(pred = mu + b_i + b_u + b_my + b_diff_y + b_g) %>%
  .$pred
```

This new model provides again a slight reduction in RMSE

```
rmse_model_bi_bu_my_b_diff_y_g <- RMSE(test_set$rating,predicted_ratings_g)
rmse_model_bi_bu_my_b_diff_y_g
```

```
## [1] 0.8652338
```

2.3.6 Regularization

Along with the parameters designed in previous subsections, we will use regularization to improve our model by calibrating it in order to minimize the adjusted loss function (RMSE) and prevent overfitting or underfitting.

```
lambdas <- seq(3.5, 6, 0.25)
lambdas
```

```
##  [1] 3.50 3.75 4.00 4.25 4.50 4.75 5.00 5.25 5.50 5.75 6.00
```

Apply the regularization procedure

```
rmses <- sapplyrmses <- sapply(lambdas, function(l){
  mu_reg <- mean(train_set$rating)

  # Just the mean rating
  b_i_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i_reg = sum(rating - mu_reg)/(n()+l))

  # User ID
  b_i_u_reg <- train_set %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg = sum(rating - mu_reg - b_i_reg)/(n()+l))

  # Movie year
  b_i_u_my_reg <- train_set %>%
    mutate(date2 = as_datetime(timestamp)) %>%
    mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
    left_join(b_i_reg, by='movieId') %>%
    left_join(b_i_u_reg, by='userId') %>%
    group_by(movie_year) %>%
    summarize(b_my = sum(rating - mu_reg - b_i_reg - b_u_reg)/(n()+l))

  # Difference between movie and rating year
  b_i_u_my_diff_reg <- train_set %>%
    mutate(date2 = as_datetime(timestamp)) %>%
    mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
    mutate(rating_year = year(date2)) %>%
    mutate(diff_year_movie_rating = rating_year - movie_year) %>%
    left_join(b_i_reg, by='movieId') %>%
    left_join(b_i_u_reg, by='userId') %>%
    left_join(b_i_u_my_reg, by='movie_year') %>%
    group_by(diff_year_movie_rating) %>%
    summarize(b_diff = sum(rating - mu_reg - b_i_reg - b_u_reg - b_my)/(n()+l))

  # Genres
  b_i_u_my_diff_g_reg <- train_set %>%
    mutate(date2 = as_datetime(timestamp)) %>%
    mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
    mutate(rating_year = year(date2)) %>%
```

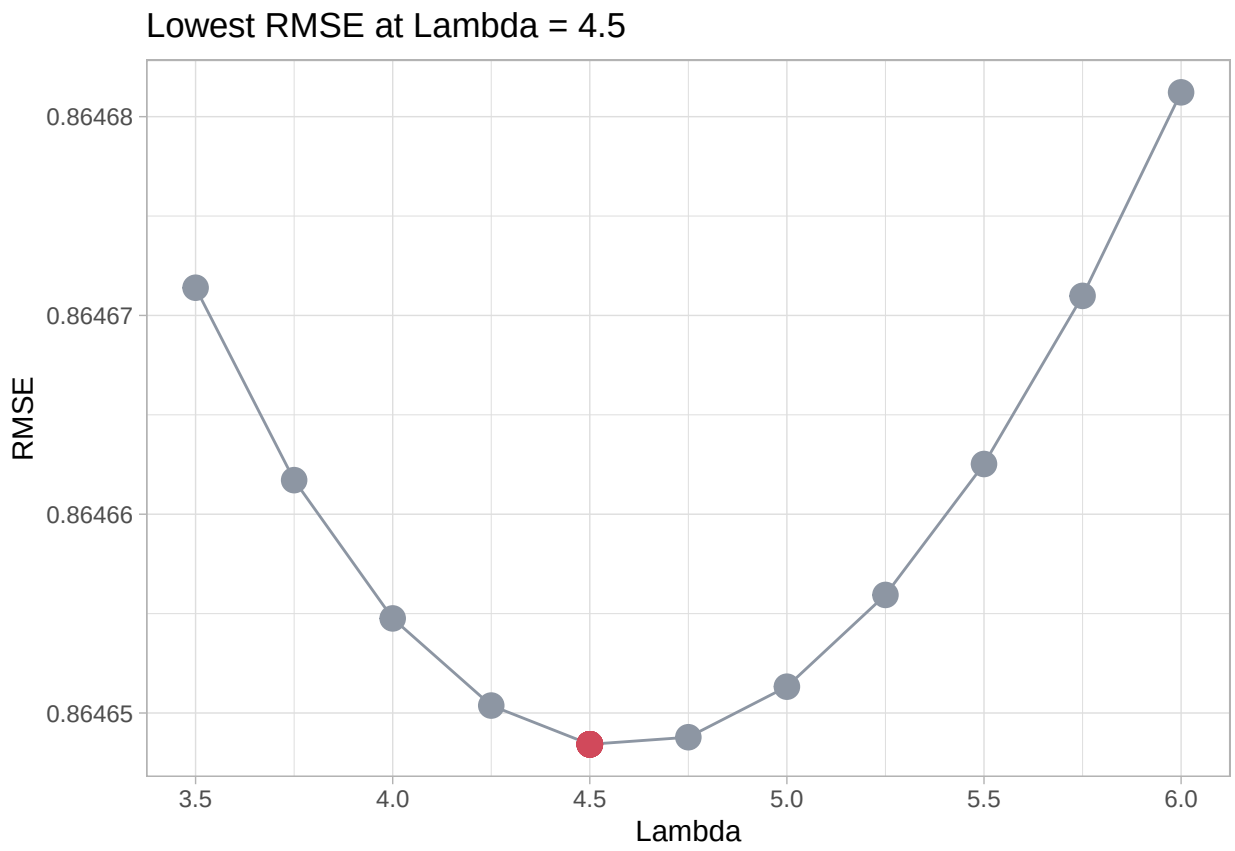
```
    mutate(diff_year_movie_rating = rating_year - movie_year) %>%
    left_join(b_i_reg, by='movieId') %>%
    left_join(b_i_u_reg, by='userId') %>%
    left_join(b_i_u_my_reg, by='movie_year') %>%
    left_join(b_i_u_my_diff_reg, by='diff_year_movie_rating') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu_reg - b_i_reg - b_u_reg - b_my - b_diff)/(n()+l))

  # Predictions
  predicted_ratings_reg <- test_set %>%
    mutate(date2 = as_datetime(timestamp)) %>%
    mutate(movie_year = as.numeric(
      substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
    mutate(rating_year = year(date2)) %>%
    mutate(diff_year_movie_rating = rating_year - movie_year) %>%
    left_join(b_i_reg, by='movieId') %>%
    left_join(b_i_u_reg, by='userId') %>%
    left_join(b_i_u_my_reg, by='movie_year') %>%
    left_join(b_i_u_my_diff_reg, by='diff_year_movie_rating') %>%
    left_join(b_i_u_my_diff_g_reg, by='genres') %>%
    mutate(pred = mu_reg + b_i_reg + b_u_reg + b_my + b_diff + b_g) %>%
    .$pred

  return(RMSE(test_set$rating,predicted_ratings_reg))
})
```

Let's have a look to the RMSE as a function of lambdas

```
data.frame(Lambda = lambdas, RMSE = rmses) %>%
  ggplot(aes(x = Lambda, y = RMSE)) +
  geom_point(color="#8d96a3", size=4) +
  geom_line(color="#8d96a3") +
  geom_point(aes(y = min(rmses), x = lambdas[which.min(rmses)]),
             color = '#d1495b', size=4) + #,shape = 10, size = 7, stroke = 1.2) +
  ggtitle(paste("Lowest RMSE at Lambda = ", lambdas[which.min(rmses)] , sep = "")) +
  xlab("Lambda") +
  ylab("RMSE") +
  theme_light()
```



The optimal lambda is, then

```
lambdas[which.min(rmses)]
```

```
## [1] 4.5
```

Summarizing, the following results are obtained using the different variables

```
models_performance <- data.frame(
Model = c("Base model", # Averaged rating of all movies
          "M", # Movie effect
          "M+U", # Movie + user effect
          "M+U+Y", # Movie + user effect + Movie year
          "M+U+Y+D", # Movie + user effect + Movie year + Ratig-Movie difference
          "M+U+Y+D+G" , #Movie + user effect + Movie year + Ratig-Movie difference + Gender
          "M+U+Y+D+G+R" # Movie + user effect + Movie year + Ratig-Movie difference + Gender + Regulari
          ),
RMSE = c(rmse_model_base_model, rmse_model_bi, rmse_model_bi_bu, rmse_model_bi_bu_my,
rmse_model_bi_bu_my_b_diff_y, rmse_model_bi_bu_my_b_diff_y_g, min(rmses)))

# Include an improvement column, as well as the % of improvement
models_performance <-
  models_performance %>% mutate(
    Improvement = RMSE[1] - RMSE,
    'Improvement (%)' = round(100*(RMSE[1] - RMSE)/RMSE[1], 3))
```

```
models_performance
```

```
##          Model      RMSE Improvement Improvement (%)
## 1   Base model 1.0607045   0.0000000           0.000
## 2            M 0.9437144   0.1169901          11.029
## 3          M+U 0.8661625   0.1945420          18.341
## 4        M+U+Y 0.8658322   0.1948723          18.372
## 5      M+U+Y+D 0.8655143   0.1951902          18.402
## 6    M+U+Y+D+G 0.8652338   0.1954707          18.428
## 7 M+U+Y+D+G+R 0.8646484   0.1960561          18.484
```

In the previous table, M = Movie effect M+U = Movie + user effect M+U+Y = Movie + user effect + Movie year M+U+Y+D = Movie + user effect + Movie year + Ratig-Movie difference M+U+Y+D+G = Movie + user effect + Movie year + Ratig-Movie difference + Gender M+U+Y+D+G+R = Movie + user effect + Movie year + Ratig-Movie difference + Gender + Regularization

The model that only considers the rating average shows a markedly higher RMSE than the rest of the models generated. This is reasonable, since we have found that there is a great variability in the ratings depending on many parameters, so that an average value does not adequately predict the ratings.

The inclusion of one explanatory variable (movie effect) reduces the RMSE in a ~11% with respect to the base model (only movie rating average), while the inclusion of the second explanatory variable reduces the RMSE in a ~18%. The consideration of other variables, as well as the regularization procedure, leads to subsequent slight reductions of RMSE.

3. Results

3.1 Final model generation

In this subsection, the final model will be generated from the insights obtained in last section, and using the whole dataset reserved for generating the model, the *edx* dataset.

Choose the optimal lambda that minimizes the RMSE

```
optimal_lambda <- lambdas[which.min(rmses)]
optimal_lambda
```

```
## [1] 4.5
```

Include the mean rating in the model

```
mu <- mean(edx$rating)
b_i_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_reg = sum(rating - mu)/(n() + optimal_lambda))
```

Include User ID

```
b_i_u_reg <- edx %>%
  left_join(b_i_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_reg = sum(rating - mu - b_i_reg)/(n() + optimal_lambda))
```

Include Movie year

```
b_i_u_my_reg <- edx %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  left_join(b_i_reg, by='movieId') %>%
  left_join(b_i_u_reg, by='userId') %>%
  group_by(movie_year) %>%
  summarize(b_my = sum(rating - mu - b_i_reg - b_u_reg)/(n() + optimal_lambda))
```

Include the difference between movie and rating year

```
b_i_u_my_diff_reg <- edx %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  left_join(b_i_reg, by='movieId') %>%
  left_join(b_i_u_reg, by='userId') %>%
  left_join(b_i_u_my_reg, by='movie_year') %>%
  group_by(diff_year_movie_rating) %>%
  summarize(b_diff = sum(rating - mu - b_i_reg - b_u_reg - b_my)/(n() + optimal_lambda))
```

Include Genres

```
b_i_u_my_diff_g_reg <- edx %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  left_join(b_i_reg, by='movieId') %>%
  left_join(b_i_u_reg, by='userId') %>%
  left_join(b_i_u_my_reg, by='movie_year') %>%
  left_join(b_i_u_my_diff_reg, by='diff_year_movie_rating') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i_reg - b_u_reg - b_my - b_diff)/(n() + optimal_lambda))
```

3.2 Testing of the recommendation system

In this subsection, the model is tested with the *final_holdout_test* dataset, not used in the model generation. To achieve this, we will first generate the predictions made by our model with the input variables of the testing dataset, and then we will compare these predictions with the actual values.

```
predicted_ratings_final <- final_holdout_test %>%
  mutate(date2 = as_datetime(timestamp)) %>%
  mutate(movie_year = as.numeric(
    substr(title, nchar(title)-5+1, nchar(title)-1)))  %>%
  mutate(rating_year = year(date2)) %>%
  # trend to provide higher ratings to "classic" movies
  mutate(diff_year_movie_rating = rating_year - movie_year) %>%
  # Set to zero negative differences
  mutate(diff_year_movie_rating = ifelse(diff_year_movie_rating < 0, 0, diff_year_movie_rating)) %>%
```

```
  left_join(b_i_reg, by='movieId') %>%
  left_join(b_i_u_reg, by='userId') %>%
  left_join(b_i_u_my_reg, by='movie_year') %>%
  left_join(b_i_u_my_diff_reg, by='diff_year_movie_rating') %>%
  left_join(b_i_u_my_diff_g_reg, by='genres') %>%
  mutate(pred = mu + b_i_reg + b_u_reg + b_my + b_diff + b_g) %>%
  .$pred
```

And the final RMSE of the Recommendation system is

```
rmse_model_final <- RMSE(final_holdout_test$rating,predicted_ratings_final)
rmse_model_final
```

```
## [1] 0.8638922
```

4. Conclusion

A movie recommendation system has been generated from a public dataset, "MovieLens 10M Dataset". We have started from the exploration of the dataset, which contains both identifiers for users and movies, as well as the movie title (including the year of the movie), and the rating date, and how these parameters relate to the rating. Based on this knowledge, a part of the original dataset, called *edx* and consisting of 90% of its data, has been used to evaluate the suitability of considering different explanatory variables in the generation of the model, as well as to add the regularization process. As a result, it has been found that the inclusion of these explanatory variables in the model leads to small improvements in the model, translated into reductions in the RMSE, or that it ends up leading to a significant improvement in the base model (consisting of simply using the mean of the ratings). Finally, the RMSE of the movie recommendation model, generated with 90% of the original dataset data, and evaluated on the remaining 10% of the data (not used for the model) is **0.8638922**. The generated model, therefore, meets the performance requirements for this project.

There are limitations in the generated model, including the type of regularization applied, or the consideration of additional variables that take into account both gender and user (i.e., considering gender preferences for each user), as well as dates in the context of each user. Additional variables about the movies, such as director, protagonists, etc. If we overcome these limitations, the model could increase its performance. Future works include the consideration of more variables, other databases to search for more insights and evaluate possible improvements. Also, the system could be presented on a web page for users to evaluate films and thus obtain recommendations, and thus periodically update the model.

5. References

Irizarry, R. A. (2019). Introduction to data science: Data analysis and prediction algorithms with R. CRC Press.

Maxwell Harper, F., Konstan, J.A. (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

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D. A., François, R., ... & Yutani, H. (2019). Welcome to the Tidyverse. Journal of open source software, 4(43), 1686.

Kuhn, M. (2015). Caret: classification and regression training. Astrophysics Source Code Library, ascl-1505.