Movie Recommendation System

HarvardX - Data Science : Capstone

Meryll Mercadier

2024-12-30

Preface

This project is an assessment, part of the online course HarvardX: Data Science: Capstone from Harvard University on Edx platform.
The HarvardX: Data Science: Capstone" is the last part of a global Professional Certificate Program in Data Science.
The purpose of this project is to autonomously put into practice all the knowledge learned during the past courses.
The following code will be highly commented in order to facilitate its readability and evaluation.

As English is not my native language, part of the comments meaning can be lost due to a wrong translation. Please take it in consideration by evaluating this project.

Summary

Introduction

In this project, the goal is to create a movie recommendation algorithm.
The algorithm will be tested on the "final_holdout_test" set and evaluated using the RMSE calculation method.
The target is an RMSE < 0.86490.
The data contains ratings from users of the online movie recommender service "MovieLens," and it is made available by GroupLens here.
You can find more information about the dataset here.
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.

Installing all packages needed

```
if (!require(tidyverse)) install.packages("tidyverse",
    repos = "http://cran.us.r-project.org")
library(tidyverse)

if (!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
library(caret)

if (!require(lubridate)) install.packages("lubridate",
    repos = "http://cran.us.r-project.org")

library(lubridate)

if (!require(tidytext)) install.packages("tidytext",
    repos = "http://cran.us.r-project.org")
library(tidytext)

if (!require(textdata)) install.packages("textdata",
    repos = "http://cran.us.r-project.org")
library(textdata)
```

Data collection and preparation

The code below for data collection and preparation is provided in the online course HarvardX: Data Science: Capstone

```
options(timeout = 120)

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

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)

ratings <- as.data.frame(str_split(read_lines(ratings_file),
    fixed("::"), simplify = TRUE), stringsAsFactors = FALSE)
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))

movies <- as.data.frame(str_split(read_lines(movies_file),
    fixed("::"), simplify = TRUE), stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
    mutate(movieId = as.integer(movieId))
```

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

# Final hold-out test set will be 10% of
# MovieLens data
set.seed(1)
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)

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

Data analysis

General understanding of the data

```
str(edx)
```

```
'data.frame':   9000061 obs. of  6 variables:
 $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
 $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
 $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 8389
 $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
 $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller" ...
```

```
summary(edx)
```

```
     userId         movieId          rating        timestamp
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
 1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
 Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
 Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
 3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
    title              genres
 Length:9000061     Length:9000061
 Class :character   Class :character
```

```
 Mode   :character   Mode   :character
```

```
class(edx)  #'data.frame'.
```

```
[1] "data.frame"
```

```
length(unique(edx$userId))  #there are 69878 unique raters.
```

```
[1] 69878
```

```
length(unique(edx$movieId))  #there are 10677 unique movies.
```

```
[1] 10677
```

```
length(unique(edx$genres))  #there are 797 unique genres combinations.
```

```
[1] 797
```

We will now explore the data following the order of the parameters in the dataframe: "userId", "movieId", "rating", "timestamp", "title", "genres".

Each line of code below includes:

A question above, which we try to answer with the line of code.

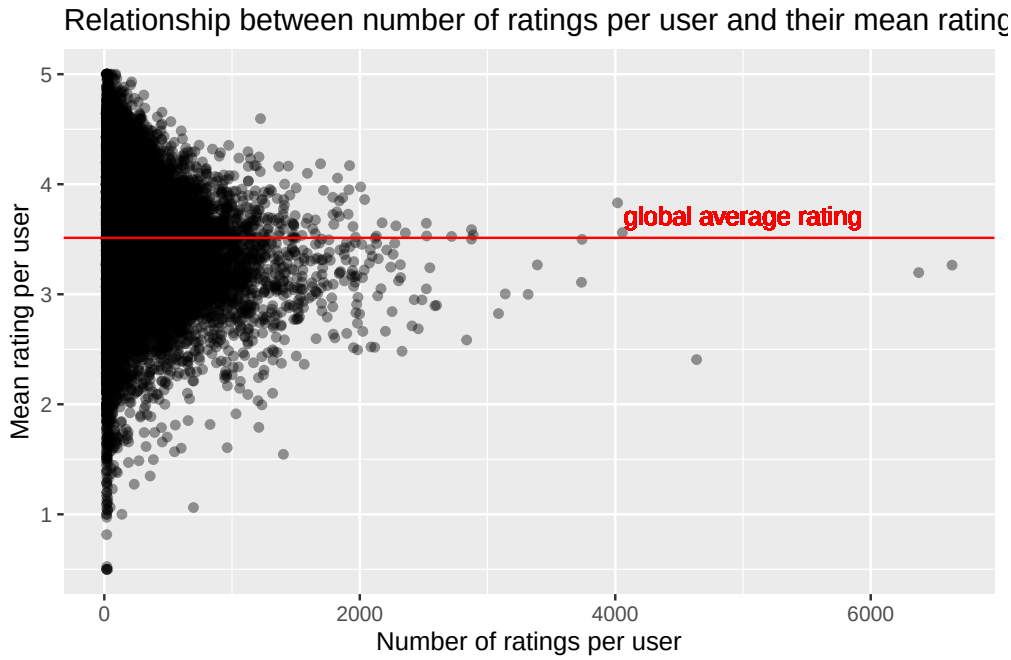
A comment to the right of the code explaining what the code does.

A comment below interpreting the output or the visual generated by the code.

Analyzing "userId" parameter

Does the number of ratings per user correlate with the mean rating?

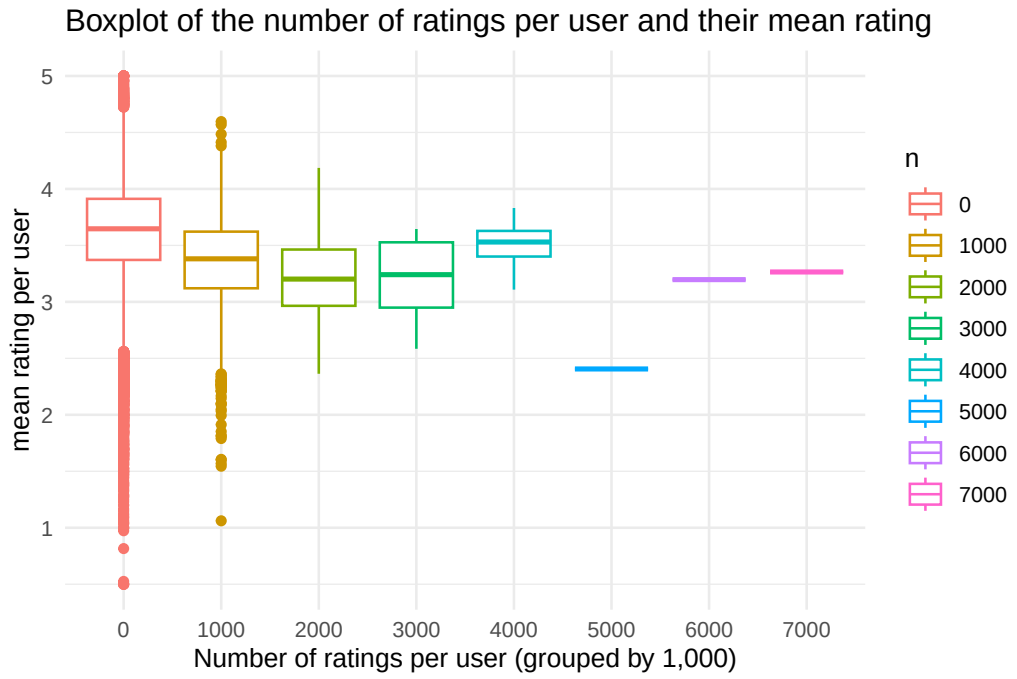
```
edx %>%
    group_by(userId) %>%
    summarize(n_rates_per_user = n(), mean_rating = mean(rating)) %>%
    arrange(-desc(n_rates_per_user)) %>%
    ggplot(aes(n_rates_per_user, mean_rating)) + geom_point(alpha = 0.4) +
    geom_hline(yintercept = mean(edx$rating), color = "red") +
    geom_text(x = 5000, y = mean(edx$rating) + 0.2,
        label = "global average rating", color = "red") +
    labs(title = "Relationship between number of ratings per user and their mean rating",
        x = "Number of ratings per user", y = "Mean rating per user")
```



It seems there is a correlation between the number of ratings per user and their mean rating. The more movies a user rates, the higher their average rating tends to be.

Can we better understand if the number of ratings per user correlates with the variability of ratings?

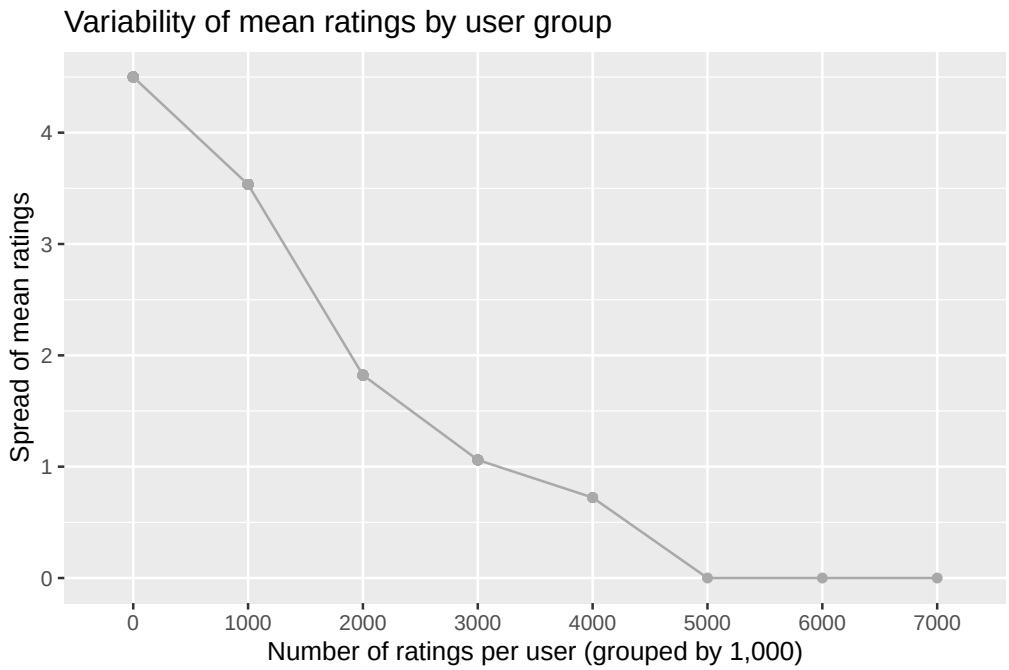
```
edx %>%
    group_by(userId) %>%
    summarize(n = round(n(), digits = -3), mean_rating = mean(rating)) %>%
    arrange(-desc(n)) %>%
    mutate(n = as.factor(n)) %>%
    ggplot(aes(n, mean_rating, color = n)) + geom_boxplot() +
    labs(title = "Boxplot of the number of ratings per user and their mean rating",
        x = "Number of ratings per user (grouped by 1,000)",
        y = "mean rating per user") + theme_minimal()
```



Contrary to the conclusion from the previous plot, we now see that there isn't a strong correlation between the mean rating and the number of ratings per user.
However, we can conclude that there is a correlation between the number of ratings per user and the variability of their mean ratings.

Can we better visualize the variability within each group?

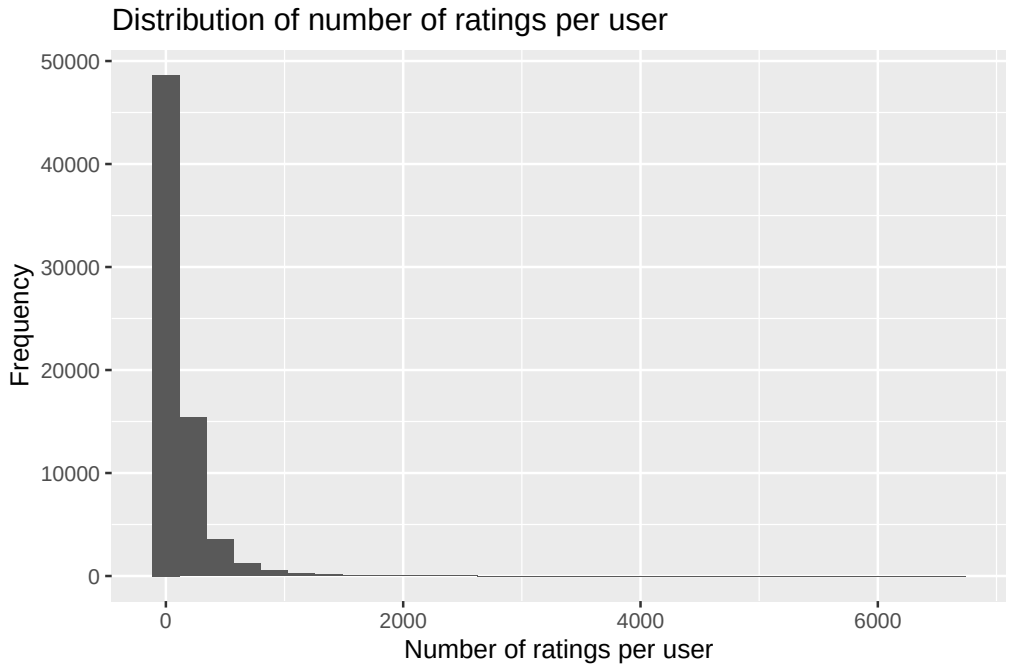
```
edx %>%
    group_by(userId) %>%
    summarize(n = round(n(), digits = -3), , mean_rating = mean(rating)) %>%
    arrange(desc(n)) %>%
    mutate(n = as.factor(n)) %>%
    group_by(n) %>%
    mutate(spread = max(mean_rating) - min(mean_rating)) %>%
    ggplot(aes(n, spread, group = 1)) + geom_line(color = "darkgrey") +
    geom_point(color = "darkgrey") + labs(title = "Variability of mean ratings by user group",
    x = "Number of ratings per user (grouped by 1,000)",
    y = "Spread of mean ratings")
```



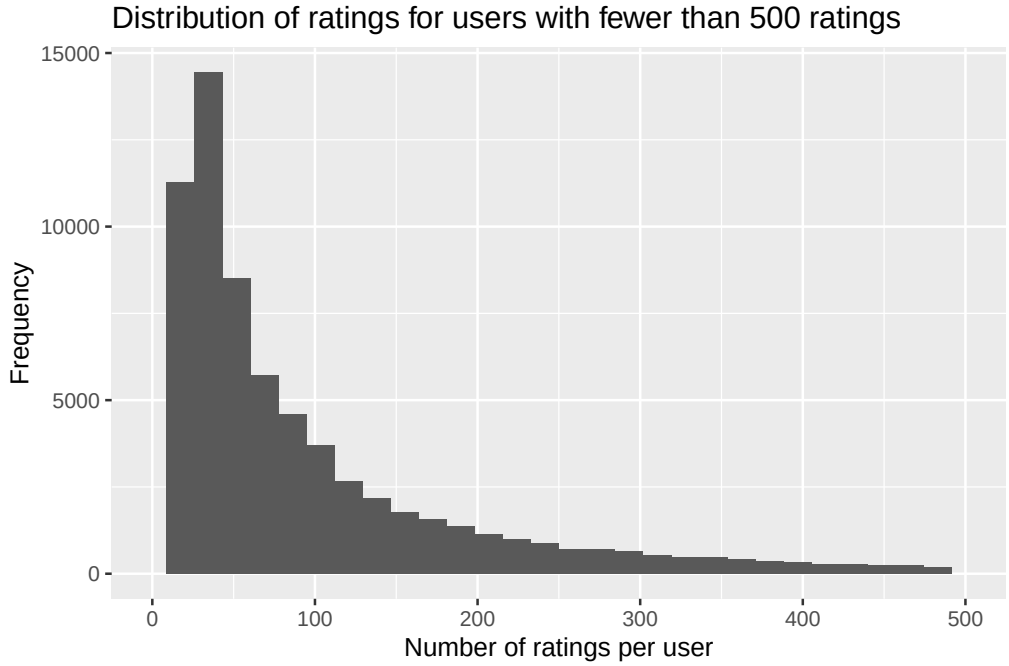
The plot highlights how the variability of mean ratings changes with the number of ratings per user.

How many ratings do people usually give?

```
edx %>%
    group_by(userId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    qplot() + labs(title = "Distribution of number of ratings per user",
    x = "Number of ratings per user", y = "Frequency")
```

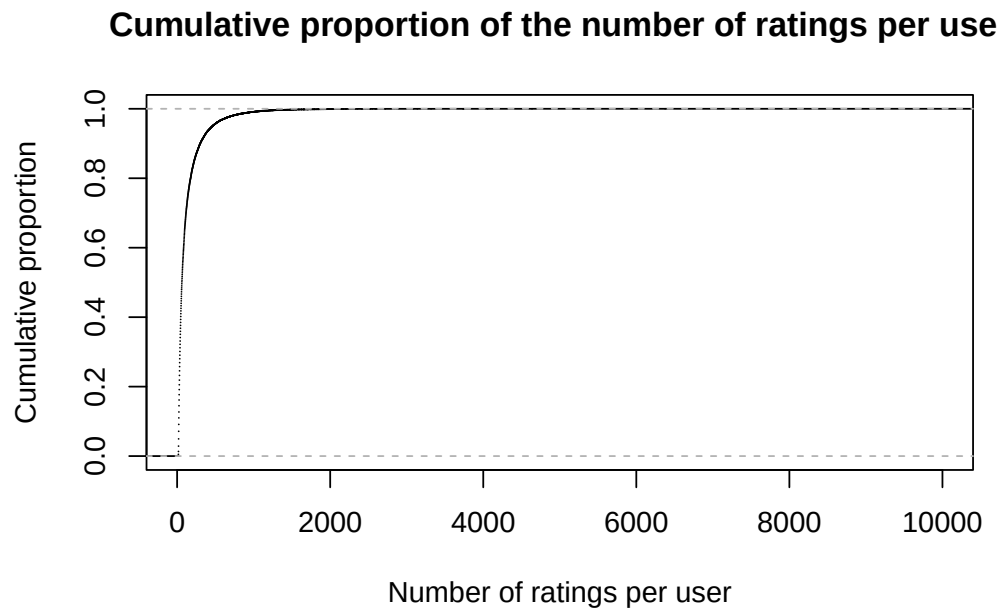



```
edx %>%
    group_by(userId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    qplot(xlim = c(0, 500)) + labs(title = "Distribution of ratings for users with fewer than 500 ratin
    x = "Number of ratings per user", y = "Frequency")
```




```
edx %>%
    group_by(userId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    ecdf() %>%
```

```
    plot(xlim = c(0, 10000), main = "Cumulative proportion of the number of ratings per user",
        xlab = "Number of ratings per user", ylab = "Cumulative proportion")
```




```
edx %>%
    group_by(userId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    quantile(, probs = 0.51)  # Finding the 51th percentile of ratings per user.
```

```
51%
 64
```

```
edx %>%
    group_by(userId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    quantile(, probs = 0.81)  # Finding the 81th percentile of ratings per user.
```

```
   81%
183.37
```

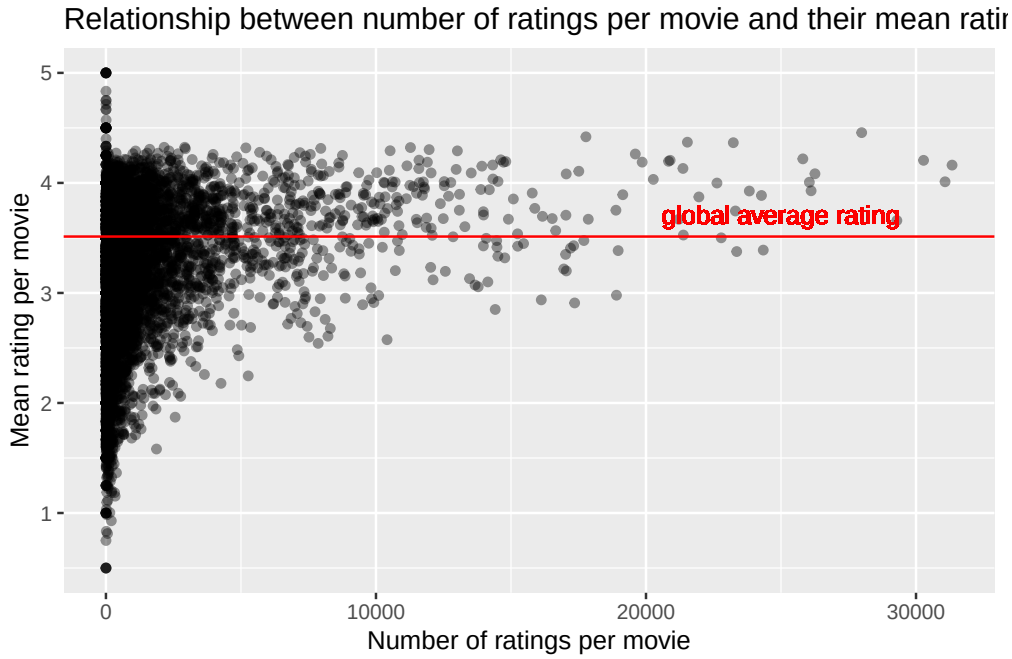
We see that most of the users gave only few ratings.
More than half of the users have given fewer than 64 ratings, and 80% of the users gave fewer than 183 ratings.

Analyzing "movieId" parameter

Does the number of ratings per movie correlate with the mean rating?

```
edx %>%
    group_by(movieId) %>%
```

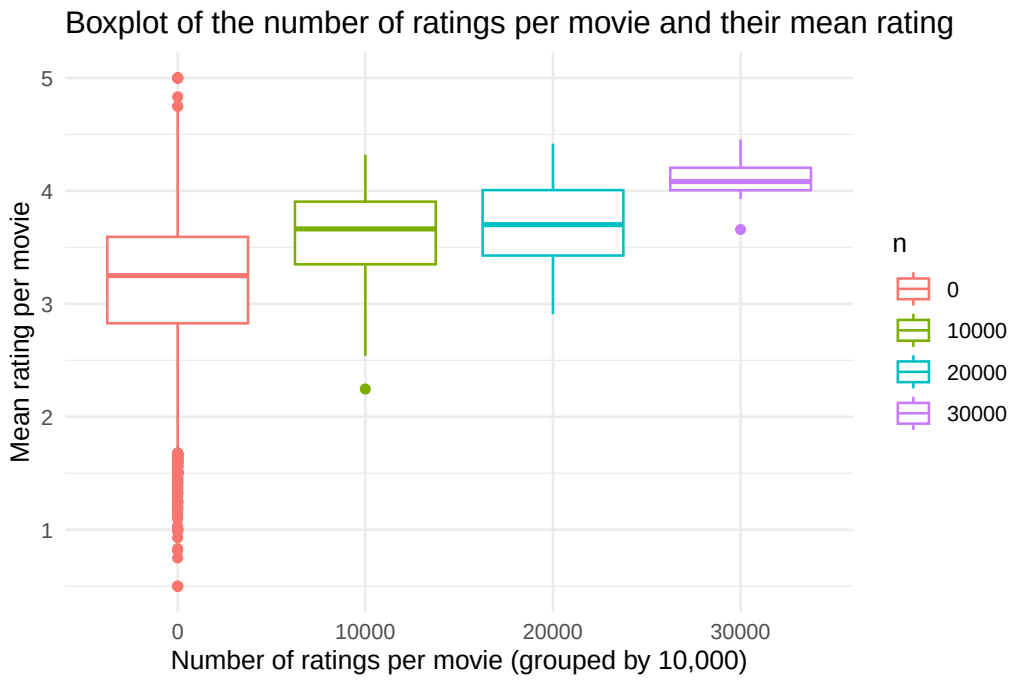
```
    summarize(n = n(), mean_rating = mean(rating)) %>%
    arrange(-desc(n)) %>%
    ggplot(aes(n, mean_rating)) + geom_point(alpha = 0.4) +
    geom_hline(yintercept = mean(edx$rating), color = "red") +
    geom_text(x = 25000, y = mean(edx$rating) + 0.2,
        label = "global average rating", color = "red") +
    labs(title = "Relationship between number of ratings per movie and their mean rating",
        x = "Number of ratings per movie", y = "Mean rating per movie")
```



We observe a correlation between the number of ratings per movie and their mean ratings. Generally, movies with more ratings tend to have higher average ratings.

Can we better understand if the number of ratings per movie correlates with the variability of ratings?

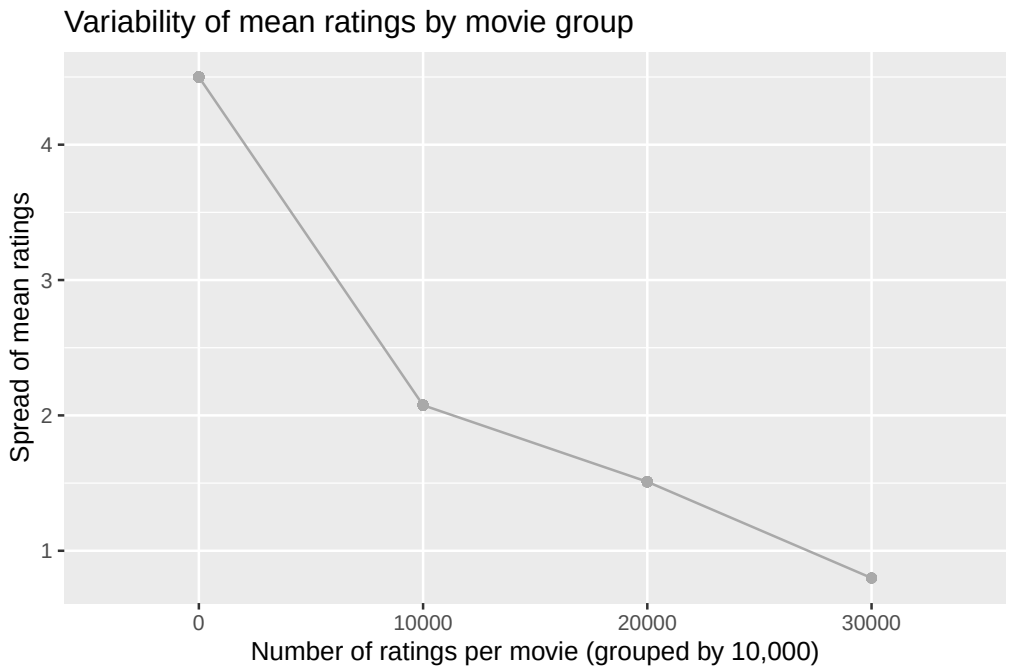
```
edx %>%
    group_by(movieId) %>%
    summarize(n = round(n(), digits = -4), mean_rating = mean(rating)) %>%
    arrange(-desc(n)) %>%
    mutate(n = as.factor(n)) %>%
    ggplot(aes(n, mean_rating, color = n)) + geom_boxplot() +
    labs(title = "Boxplot of the number of ratings per movie and their mean rating",
        x = "Number of ratings per movie (grouped by 10,000)",
        y = "Mean rating per movie") + theme_minimal()
```



These boxplots show that movies with more ratings tend to have higher average ratings.
Movies with fewer ratings show higher variability in their mean ratings.
As the number of ratings increases, the variability in ratings decreases.

Can we better visualize the variability within each group?

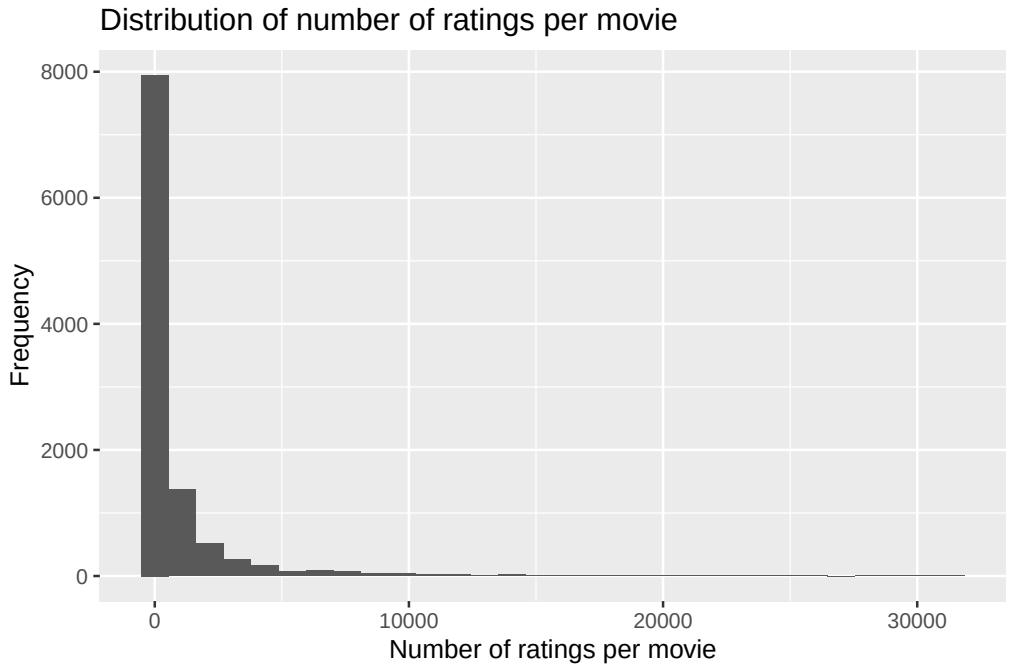
```
edx %>%
    group_by(movieId) %>%
    summarize(n = round(n(), digits = -4), , mean_rating = mean(rating)) %>%
    arrange(desc(n)) %>%
    mutate(n = as.factor(n)) %>%
    group_by(n) %>%
    mutate(spread = max(mean_rating) - min(mean_rating)) %>%
    ggplot(aes(n, spread, group = 1)) + geom_line(color = "darkgrey") +
    geom_point(color = "darkgrey") + labs(title = "Variability of mean ratings by movie group",
    x = "Number of ratings per movie (grouped by 10,000)",
    y = "Spread of mean ratings")
```



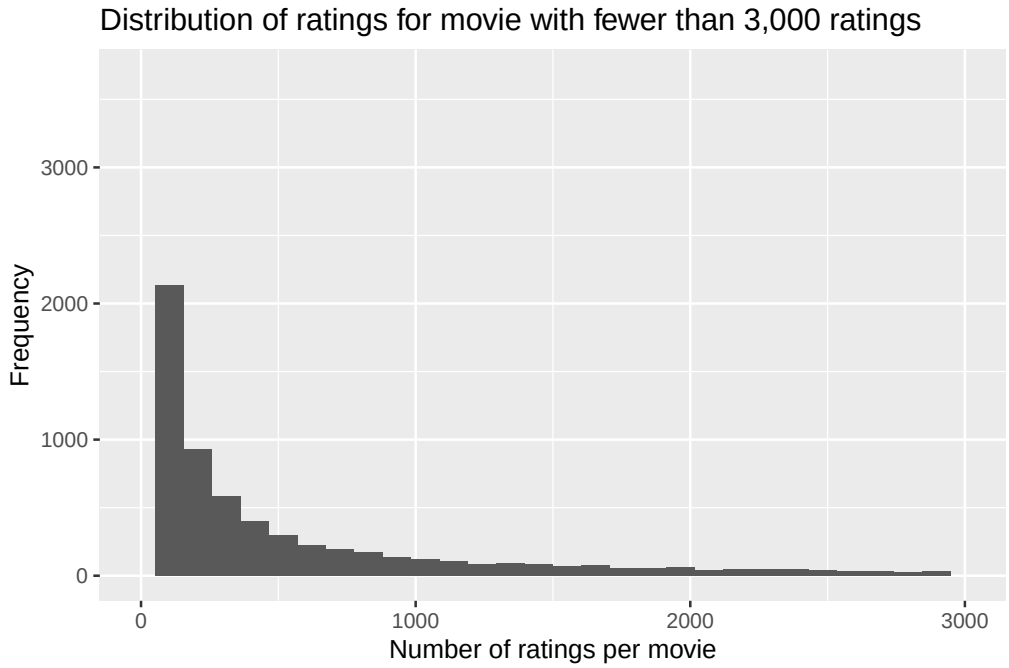
The plot highlights how the variability of mean ratings changes with the number of ratings per movie.

How many rates do movies have?

```
edx %>%
    group_by(movieId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    qplot() + labs(title = "Distribution of number of ratings per movie",
    x = "Number of ratings per movie", y = "Frequency")
```

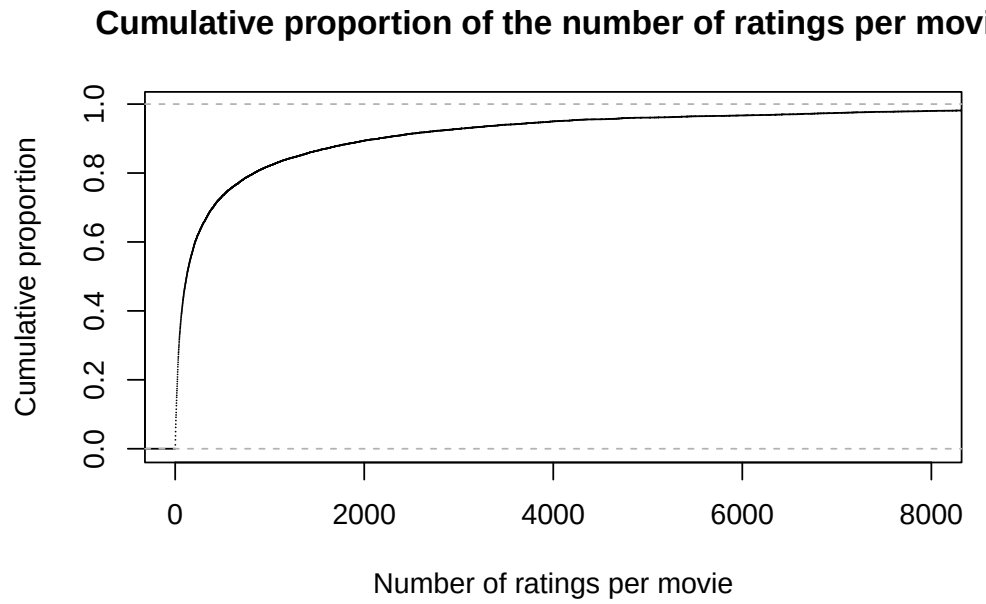



```
edx %>%
    group_by(movieId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    qplot(xlim = c(0, 3000)) + labs(title = "Distribution of ratings for movie with fewer than 3,000 ra
    x = "Number of ratings per movie", y = "Frequency")
```




```
edx %>%
    group_by(movieId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    ecdf() %>%
```

```
    plot(xlim = c(0, 8000), main = "Cumulative proportion of the number of ratings per movie",
        xlab = "Number of ratings per movie", ylab = "Cumulative proportion")
```




```
edx %>%
    group_by(movieId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    quantile(, probs = 0.51)  # Finding the 51th percentile of ratings per movie
```

```
51%
128
```

```
edx %>%
    group_by(movieId) %>%
    summarize(n = n()) %>%
    pull(n) %>%
    quantile(, probs = 0.81)  # Finding the 81th percentile of ratings per movie.
```

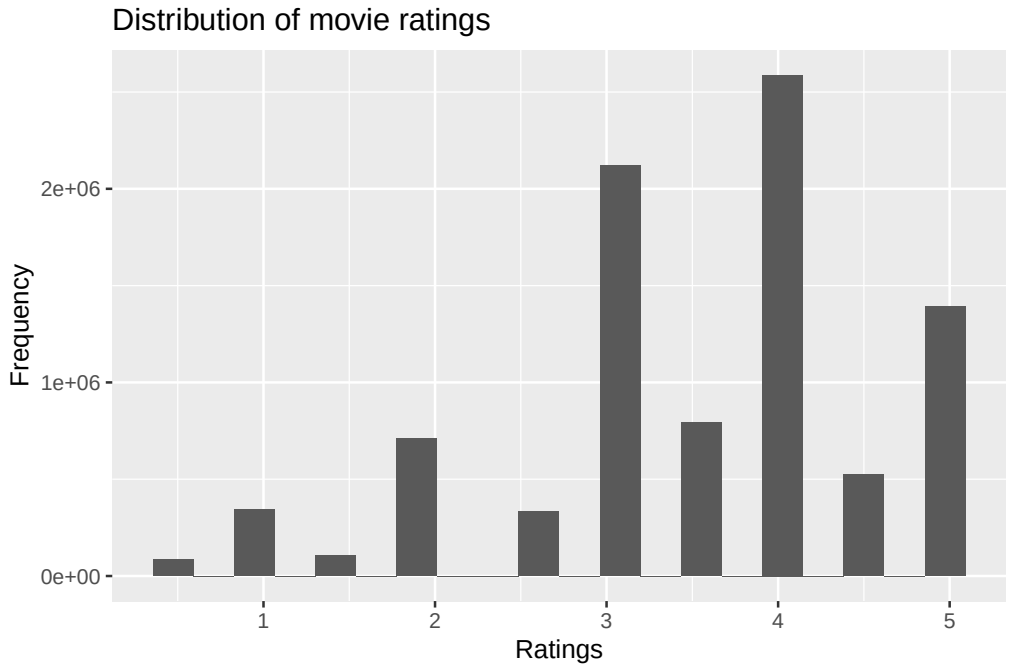
```
81%
905
```

We see that most of the movies have only few ratings.
More than half of the movies have fewer than 128 ratings, and 80% of the users have fewer than 903 ratings.

Analyzing "rating" parameter

What's the distribution of ratings?

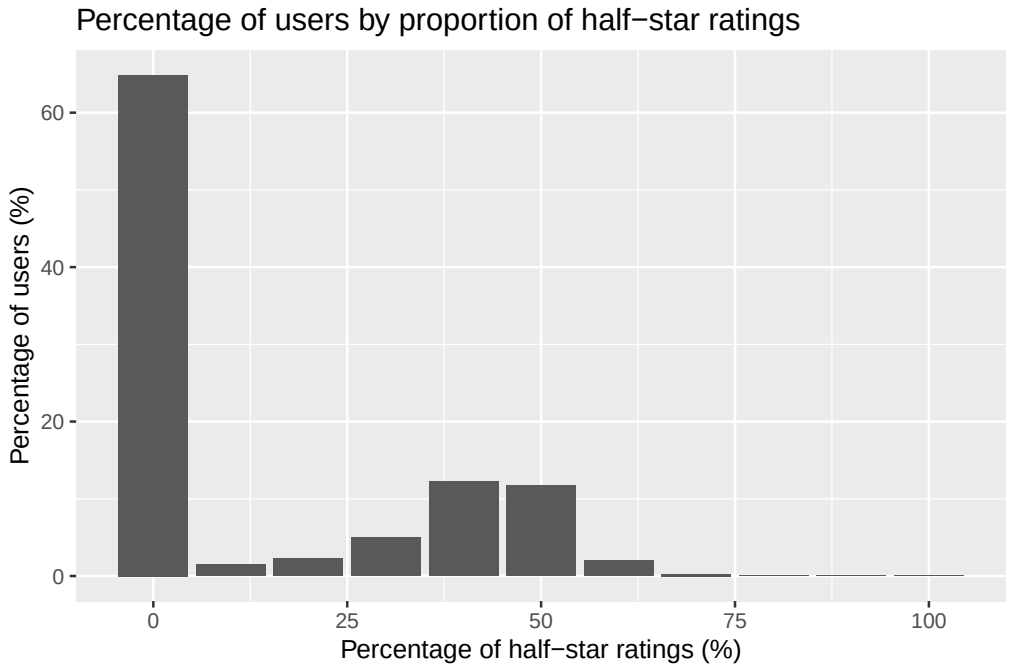
```
qplot(edx$rating, bins = 20) + labs(title = "Distribution of movie ratings",
    x = "Ratings", y = "Frequency")
```



We notice that whole-star ratings are more frequent than half-star ratings.
It seems there are two distinct distributions: one for whole-star ratings and another for half-star ratings.

Can we conclude that there are two types of raters: whole-star raters and half-star raters?

```
edx %>%
    group_by(userId) %>%
    summarize(dot = round(mean(str_detect(rating, "\\d\\.\\d")),
        digit = 1) * 100) %>%
    group_by(dot) %>%
    summarize(n = n(), part = (n/length(unique(edx$userId))) *
        100) %>%
    ggplot(aes(x = dot, y = part)) + geom_bar(stat = "identity") +
    labs(title = "Percentage of users by proportion of half-star ratings",
        x = "Percentage of half-star ratings (%)",
        y = "Percentage of users (%)")
```



This bar plot shows that most users don't use half-star ratings at all. Among users who do use half-star ratings, the majority still tend to favor whole-star ratings.

What's the percentage of users who use half-star ratings more than 50% of the time?

```
edx %>%
    group_by(userId) %>%
    summarize(dot = round(mean(str_detect(rating, "\\d\\.\\d")),
        digit = 1) * 100) %>%
    group_by(dot) %>%
    summarize(n = n(), part = (n/length(unique(edx$userId))) *
        100) %>%
    filter(dot > 50) %>%
    summarize(sum(part)) %>%
    pull()
```

```
[1] 2.335499
```

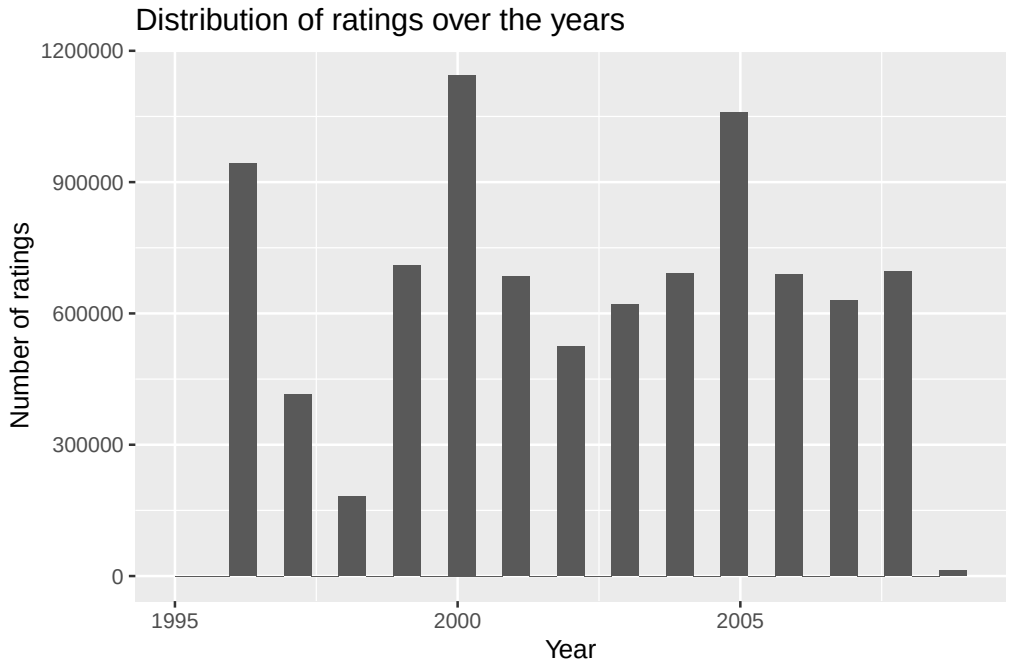
We observe that only a small fraction of users (2.3%) use half-star ratings more than 50% of the time. This shows us that there aren't really two distinct types of raters (whole-star & half-star raters).

Analyzing "timestamp" parameter

What's the distribution of ratings across years?
As stated in the GroupLens dataset README.txt:

Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970."

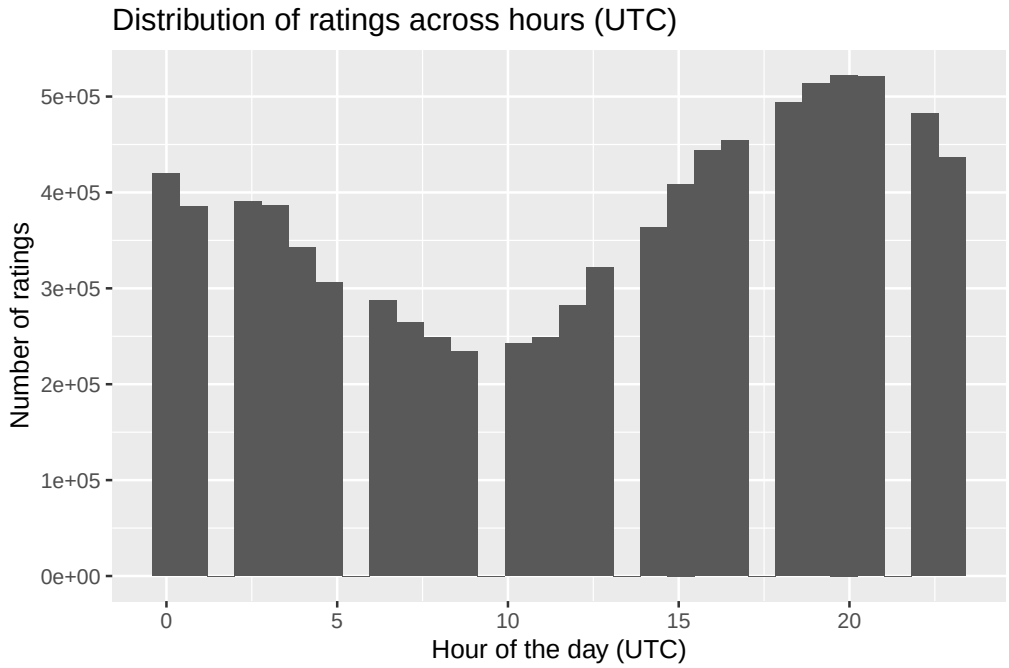
```
qplot(year(as.POSIXct(edx$timestamp, origin, tz = "UTC"))) +
    labs(title = "Distribution of ratings over the years",
        x = "Year", y = "Number of ratings")
```



The distribution doesn't provide much additional insight about the data.
The distribution doesn't appear to follow a normal pattern.
This irregularity might be due to the fact that this dataset is a random sample from a larger dataset.

What's the distribution of ratings across hours?

```
qplot(hour(as.POSIXct(edx$timestamp, origin, tz = "UTC"))) +
    labs(title = "Distribution of ratings across hours (UTC)",
        x = "Hour of the day (UTC)", y = "Number of ratings")
```

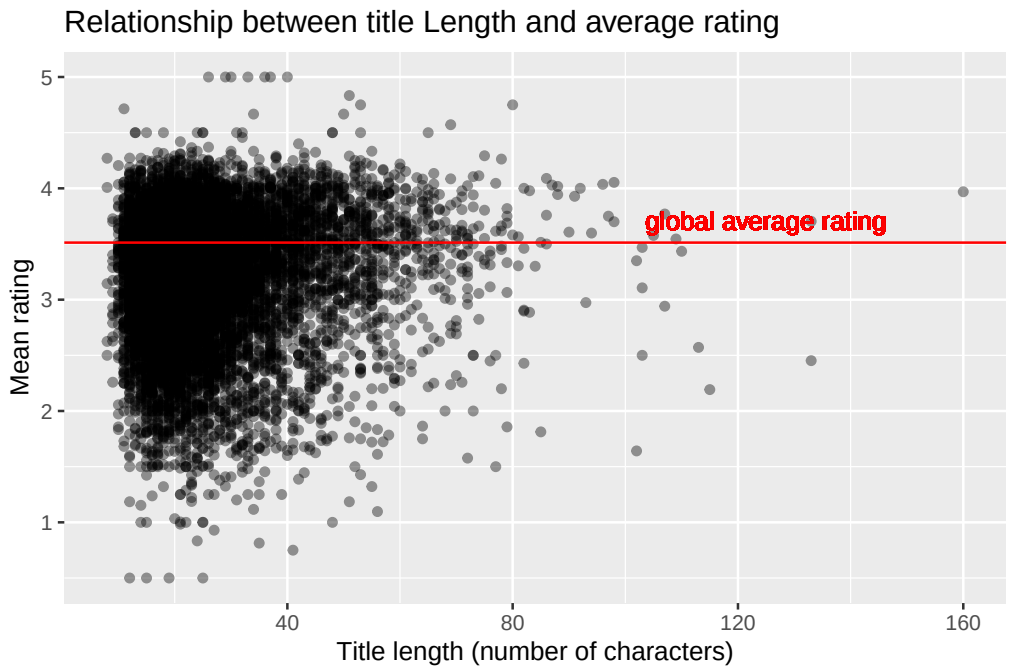


The distribution of ratings across hours appears normal.
However, there's a secondary smaller bump between 2 AM and 4 AM (UTC).
This could suggest different geographic profiles or time zones influencing rating behavior.
Despite this observation, the data isn't informative enough for further analysis.

Analyzing "title" parameter

Is the title length (number of characters) correlated with the average rating?

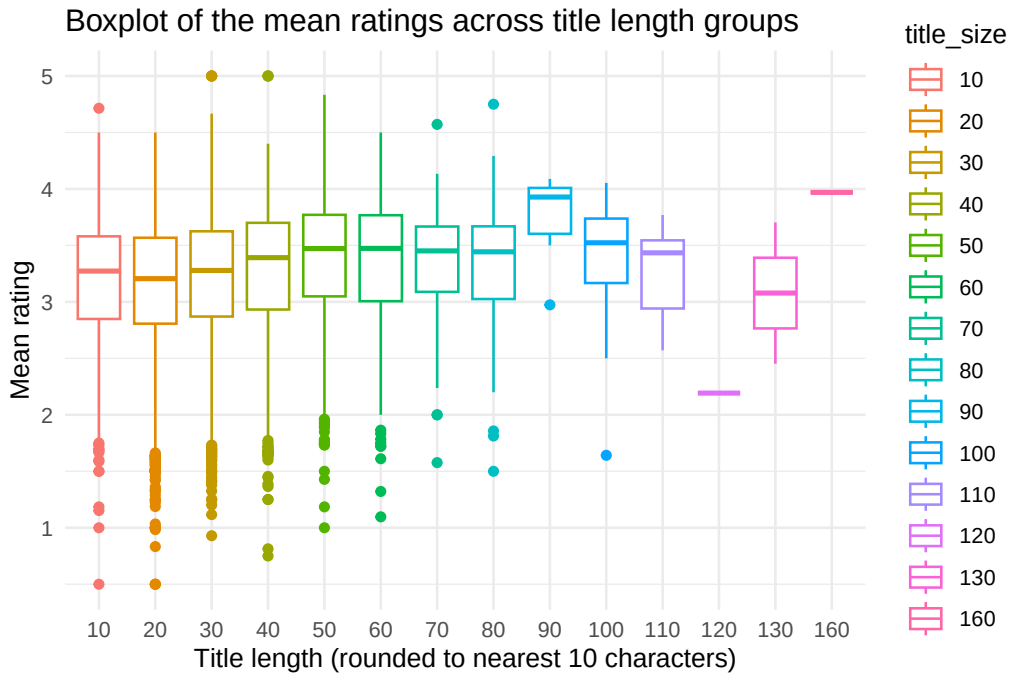
```
edx %>%
    mutate(title_size = nchar(title)) %>%
    group_by(movieId, title) %>%
    summarize(title_size = mean(title_size), mean_rate = mean(rating)) %>%
    ungroup() %>%
    select(title_size, mean_rate) %>%
    ggplot(aes(title_size, mean_rate)) + geom_point(alpha = 0.4) +
    geom_hline(yintercept = mean(edx$rating), color = "red") +
    geom_text(x = 125, y = mean(edx$rating) + 0.2,
        label = "global average rating", color = "red") +
    labs(title = "Relationship between title Length and average rating",
        x = "Title length (number of characters)",
        y = "Mean rating")
```



This plot does not allow us to make any clear conclusion.
There is significant variability in average ratings across all title lengths.

Can we better understand if title length is correlated with average rating?

```
edx %>%
    mutate(title_size = nchar(title)) %>%
    group_by(movieId, title) %>%
    summarize(title_size = round(mean(title_size),
        digit = -1), mean_rate = mean(rating)) %>%
    ungroup() %>%
    mutate(title_size = as.factor(title_size)) %>%
    select(title_size, mean_rate) %>%
    ggplot(aes(title_size, mean_rate, color = title_size)) +
    geom_boxplot() + labs(title = "Boxplot of the mean ratings across title length groups",
    x = "Title length (rounded to nearest 10 characters)",
    y = "Mean rating") + theme_minimal()
```

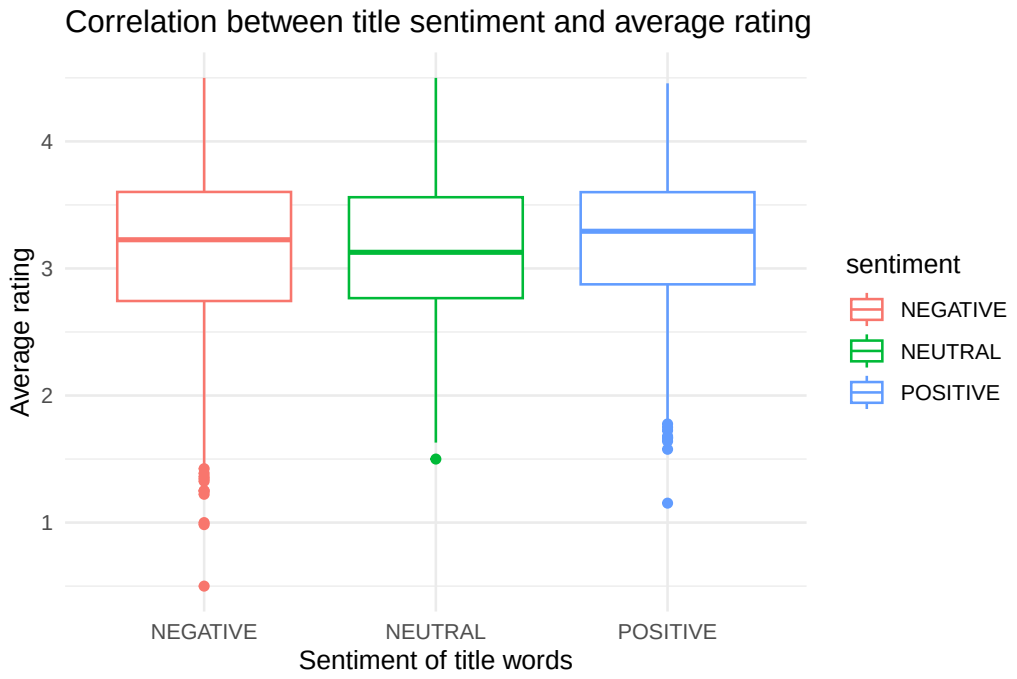


This result confirms our previous conclusion:
There is significant variability in average ratings across all title length groups.
We can't see clear correlation between title length and average rating.

Does the sentiment or emotion of words in the title correlate with the average rating?

```
title_words <- edx %>%
    select(movieId, title) %>%
    group_by(movieId) %>%
    unnest_tokens(word, title) %>%
    # Separating each word in movie titles,
    # removing numbers and common neutral words
filter(!word %in% stop_words$word & !str_detect(word,
    "^\\d+$"))
# Loading the Bing sentiment lexicon (dataframe
# with words and their sentiments ('positive' or
# 'negative'))
sentiments_bing <- get_sentiments("bing")
# Assigning sentiments to words in titles and
# aggregating sentiment per movie
# ('negative','neutral','positive')
title_bing <- title_words %>%
    inner_join(x = title_words, y = sentiments_bing,
        by = "word") %>%
    aggregate(model.matrix(~sentiment + 0) ~ movieId,
        , FUN = mean) %>%
    mutate(sentiment = ifelse(sentimentnegative > 0.5,
        "NEGATIVE", ifelse(sentimentnegative == 0.5,
            "NEUTRAL", "POSITIVE"))) %>%
```

```
    select(movieId, sentiment)
edx %>%
    group_by(movieId) %>%
    summarise(mean_rating = mean(rating)) %>%
    inner_join(, y = title_bing, by = "movieId") %>%
    ggplot(aes(sentiment, mean_rating, color = sentiment)) +
    geom_boxplot() + labs(title = "Correlation between title sentiment and average rating",
    x = "Sentiment of title words", y = "Average rating") +
    theme_minimal()
```



There is significant variability in average ratings across sentiments groups.
We can't see clear correlation between title sentiment and average rating.
A different sentiment analysis method (e.g., "afinn", "loughran", "nrc") might perhaps gives us more interesting results.

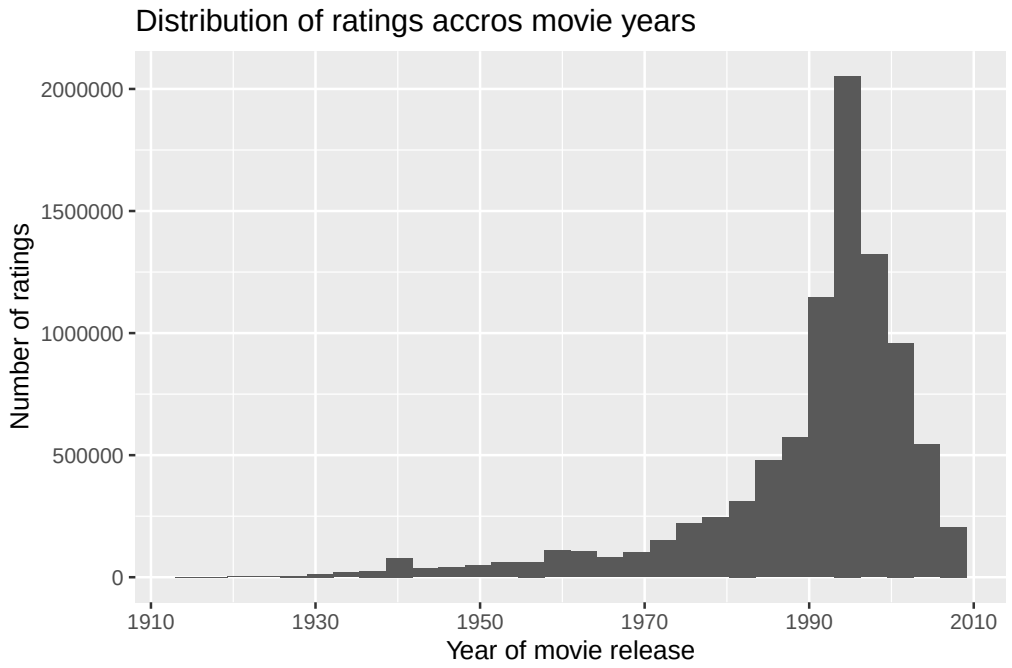
We can see that in the title there are the year of movies realizations.
Let's extract this information to see if it provides us additional insights.

```
year_movie <- str_extract_all(edx$title, "\\([0-9]{4}\\)$",
    simplify = TRUE)
year_movie <- as.numeric(str_extract_all(year_movie,
    "[0-9]{4}", simplify = TRUE))
edx <- edx %>%
    mutate(year_movie = year_movie, rating_year = year(as.POSIXct(edx$timestamp,
        origin, tz = "UTC")), rating_delay = rating_year -
        year_movie)
```

Analyzing "movie_year" parameter

what's the rating distribution across movie years?

```
qplot(edx$year_movie) + labs(title = "Distribution of ratings accros movie years",
    x = "Year of movie release", y = "Number of ratings")
```



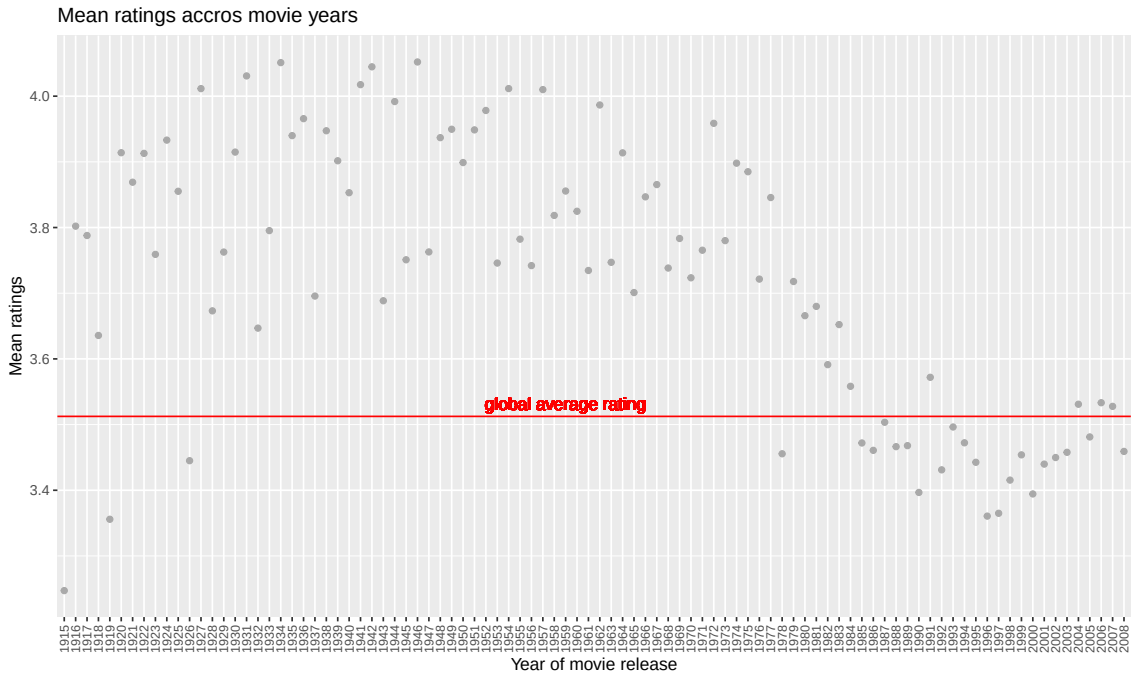

```
edx %>%
    group_by(year_movie) %>%
    summarize(n = n()) %>%
    arrange(desc(n)) %>%
    slice(1:10)  # Displays the top movie release years with the highest number of ratings.
```

```
# A tibble: 10 x 2
   year_movie      n
        <dbl>  <int>
 1       1995 787116
 2       1994 671676
 3       1996 593442
 4       1999 489899
 5       1993 481557
 6       1997 429928
 7       1998 401905
 8       2000 382510
 9       2001 305561
10       2002 272061
```

The number of ratings increased from the movie year 1910 to 1995.
There is a important number of ratings for movies released in 1995.
After 1995, the number of ratings gradually decreases up to 2010.

Is there a relationship between movie release year and mean ratings?

```
edx %>%
    group_by(year_movie) %>%
    summarize(mean_rating = mean(rating)) %>%
    mutate(year_movie = as.factor(year_movie)) %>%
    ggplot(aes(year_movie, mean_rating)) + geom_point(color = "darkgrey") +
    geom_hline(yintercept = mean(edx$rating), color = "red") +
    geom_text(x = 45, y = mean(edx$rating) + 0.02,
        label = "global average rating", color = "red") +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5,
        hjust = 1, size = 8)) + labs(title = "Mean ratings accros movie years",
    x = "Year of movie release", y = "Mean ratings")
```



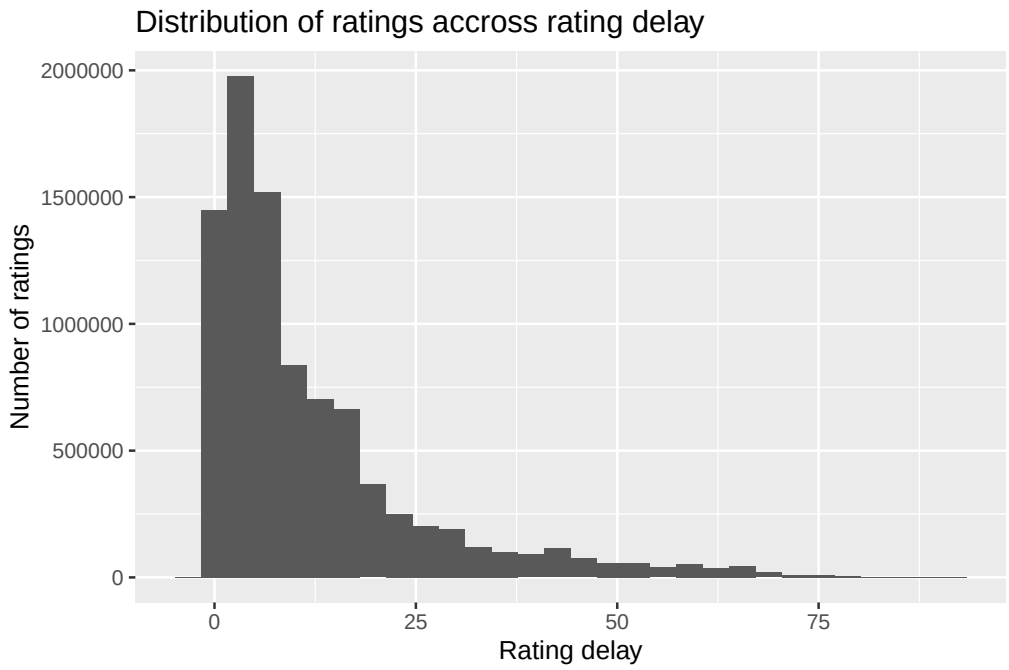
There seems to be a correlation between the movie release year and the mean ratings.
From 1915 to 1983, most mean ratings are above the global average rating.
After 1983, mean ratings tend to fall below the global average rating.
A possible explanation is that older movies are often watched based on recommendations, which might lead to higher average ratings over time.

According to this interpretation, it would be more relevant to analyze the "rating delay" than "movie year release"
("rating delay" = difference between the year of the rating and the year of the movie release).

Analyzing "rating_delay" parameter

what's the rating distribution across rating delay?

```
qplot(edx$rating_delay) + labs(title = "Distribution of ratings accross rating delay",
    x = "Rating delay", y = "Number of ratings")
```



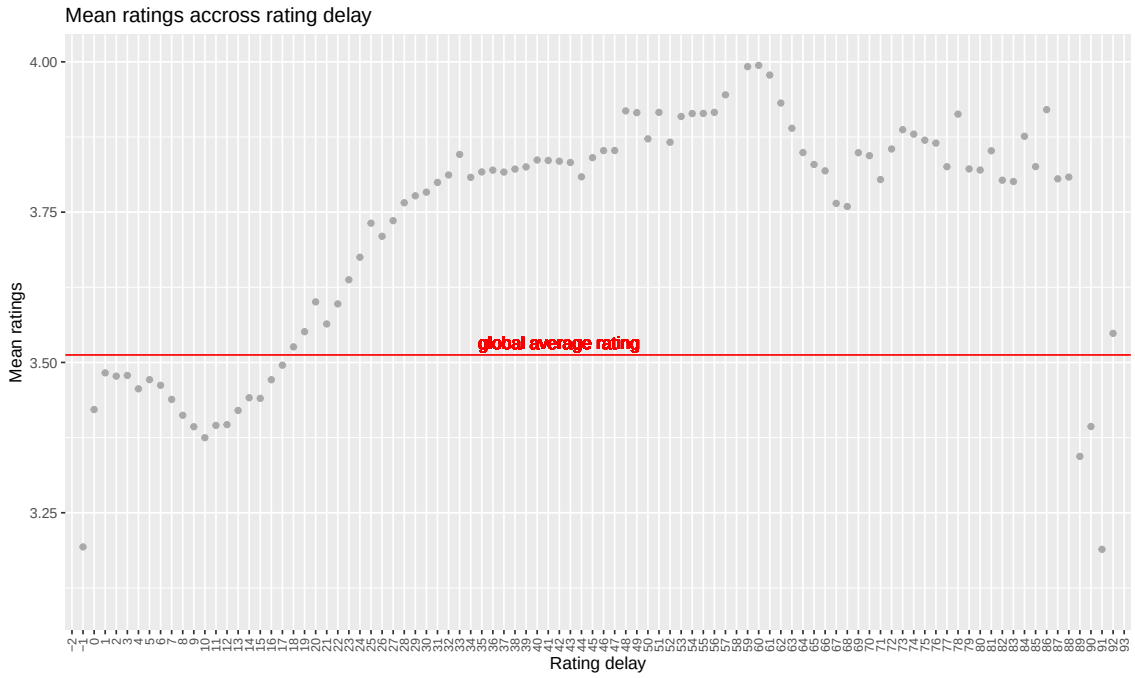

```
# Displays the top rating_delay with the highest
# number of ratings.
edx %>%
    group_by(rating_delay) %>%
    summarize(n = n()) %>%
    arrange(desc(n)) %>%
    slice(1:10)
```

```
# A tibble: 10 x 2
   rating_delay       n
          <dbl>   <int>
 1            1 1068301
 2            2  854027
 3            3  648254
 4            4  473786
 5            5  435680
 6            6  410133
 7            0  380526
 8            7  354169
 9            8  320514
10            9  292593
```

There is a important number of ratings on the first year after the movie relase.
The the number of ratings gradually decreases.

Is there a relationship between "rating delay" and mean ratings?

```r
edx %>%
    group_by(rating_delay) %>%
    summarize(mean_rating = mean(rating)) %>%
    mutate(rating_delay = as.factor(rating_delay)) %>%
    ggplot(aes(rating_delay, mean_rating)) + geom_line(color = "darkgrey") +
    geom_point(color = "darkgrey") + geom_hline(yintercept = mean(edx$rating),
    color = "red") + geom_text(x = 45, y = mean(edx$rating) +
    0.02, label = "global average rating", color = "red") +
    ylim(c(3.1, 4)) + theme(axis.text.x = element_text(angle = 90,
    vjust = 0.5, hjust = 1, size = 8)) + labs(title = "Mean ratings accross rating delay",
    x = "Rating delay", y = "Mean ratings")
```



We can see a correlation between the rating delay and the average rating.
This result permit us to validate the interpretation made with the movie year release :
Older movies are often watched based on recommendations, which might lead to higher average ratings over time.

Analyzing "genres" parameter

what are the top 10 rated genres?

```r
edx %>%
    group_by(genres) %>%
    summarize(n = n(), mean_rating = mean(rating)) %>%
    filter(n > 10) %>%
    arrange(desc(mean_rating)) %>%
    slice(1:10)
```

```
# A tibble: 10 x 3
   genres                                      n mean_rating
   <chr>                                   <int>       <dbl>
 1 Drama|Film-Noir|Romance                  2985        4.31
 2 Action|Crime|Drama|IMAX                  2333        4.29
 3 Animation|Children|Comedy|Crime          7183        4.28
 4 Film-Noir|Mystery                        5993        4.24
 5 Film-Noir|Romance|Thriller               2420        4.23
 6 Crime|Film-Noir|Mystery                  3989        4.23
 7 Crime|Film-Noir|Thriller                 4818        4.20
 8 Crime|Mystery|Thriller                  26774        4.20
 9 Action|Adventure|Comedy|Fantasy|Romance 14809        4.19
10 Crime|Thriller|War                       4611        4.16
```

what are the bottom 10 rated genres?

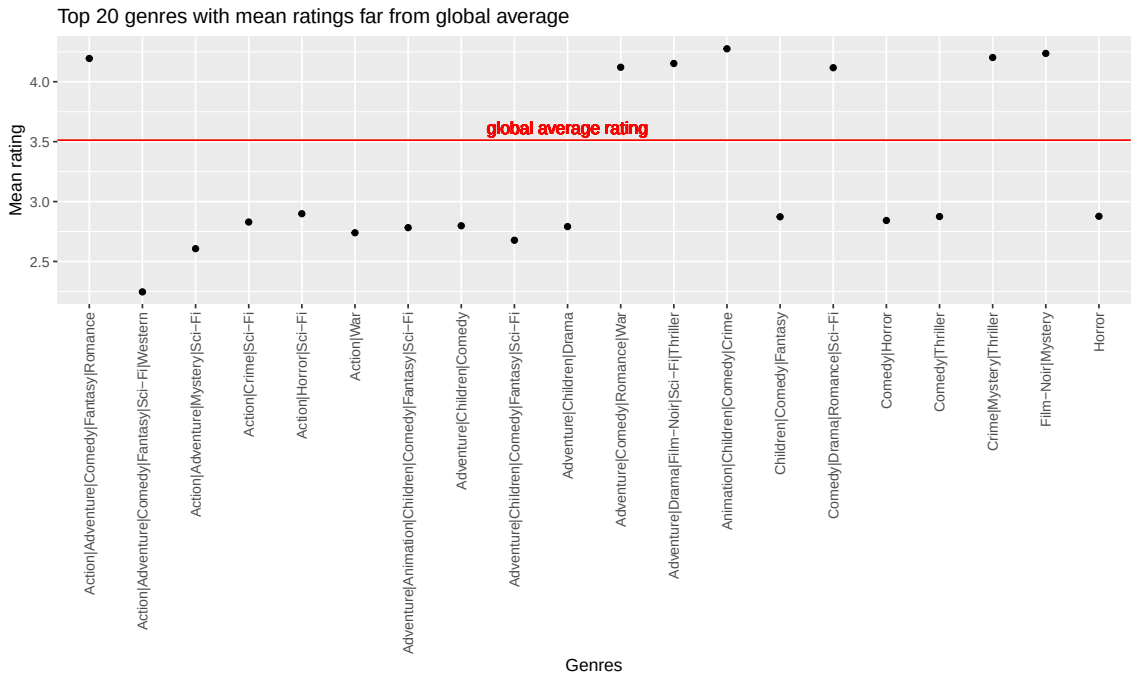
```
edx %>%
    group_by(genres) %>%
    summarize(n = n(), mean_rating = mean(rating)) %>%
    filter(n > 10) %>%
    arrange(-desc(mean_rating)) %>%
    slice(1:10)
```

```
# A tibble: 10 x 3
   genres                                        n mean_rating
   <chr>                                     <int>       <dbl>
 1 Documentary|Horror                          655        1.47
 2 Comedy|Film-Noir|Thriller                    23        1.59
 3 Action|Horror|Mystery|Thriller              325        1.59
 4 Adventure|Drama|Horror|Sci-Fi|Thriller      220        1.73
 5 Action|Children|Comedy                      523        1.88
 6 Action|Adventure|Drama|Fantasy|Sci-Fi        61        1.89
 7 Children|Fantasy|Sci-Fi                      57        1.89
 8 Action|Horror|Mystery|Sci-Fi                 23        1.91
 9 Action|Adventure|Children                   821        1.91
10 Adventure|Animation|Children|Fantasy|Sci-Fi  691        1.93
```

Are there any genres with a rate far from the average?

```
mu <- mean(edx$rating, na.rm = TRUE)
edx %>%
    group_by(genres) %>%
    mutate(n = n()) %>%
    filter(n > 5000) %>%
    summarize(mean_rating = mean(rating), mu_distance = abs(mu -
        mean_rating)) %>%
    arrange(desc(mu_distance)) %>%
    slice(1:20) %>%
```

```
    ggplot(aes(genres, mean_rating)) + geom_point() +
    geom_hline(yintercept = mean(edx$rating), color = "red") +
    geom_text(x = 10, y = mean(edx$rating) + 0.1, label = "global average rating",
        color = "red") + theme(axis.text.x = element_text(angle = 90,
    vjust = 0.5, hjust = 1)) + labs(title = "Top 20 genres with mean ratings far from global average",
    x = "Genres", y = "Mean rating")
```



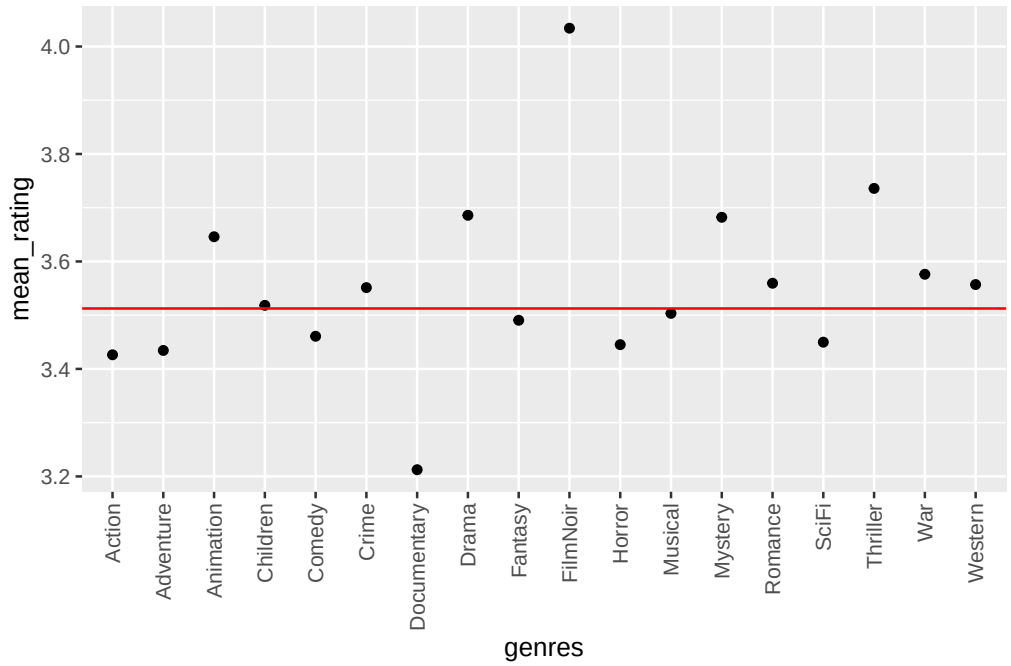
There is indeed a relationship between genres and mean ratings.
Some genres highly perform above or below average.

Does each genre individually have a correlation with the rating?

```
any_genres <- data.frame(genres = c("Action", "Adventure",
    "Animation", "Children", "Comedy", "Crime", "Documentary",
    "Drama", "Fantasy", "FilmNoir", "Horror", "Musical",
    "Mystery", "Romance", "SciFi", "Thriller", "War",
    "Western"), mean_rating = c(edx[str_which(edx$genres,
    "Action."), ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Adventure."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Animation."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Children."),
    ] %>%
    summarize(mean(rating)) %>%
```

```
    pull(), edx[str_which(edx$genres, ".Comedy."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Crime."), ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Documentary."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Drama."), ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Fantasy."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Film-Noir."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Horror."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Musical."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Mystery."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Romance."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Sci-Fi."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Thriller."),
    ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".War."), ] %>%
    summarize(mean(rating)) %>%
    pull(), edx[str_which(edx$genres, ".Western"),
    ] %>%
    summarize(mean(rating)) %>%
    pull()))


any_genres %>%
    ggplot(aes(genres, mean_rating)) + geom_point() +
    geom_hline(yintercept = mean(edx$rating), color = "red") +
    theme(axis.text.x = element_text(angle = 90, vjust = 0.5,
        hjust = 1))
```




```
any_genres %>%
    mutate(mu_distance = abs(mu - mean_rating)) %>%
    arrange(desc(mu_distance)) %>%
    slice(1:20)
```

```
        genres mean_rating mu_distance
1     FilmNoir    4.034070 0.521605561
2  Documentary    3.212471 0.299992839
3     Thriller    3.735938 0.223473787
4        Drama    3.685897 0.173432927
5      Mystery    3.682222 0.169757639
6    Animation    3.646002 0.133538288
7       Action    3.426305 0.086158587
8    Adventure    3.434406 0.078057471
9       Horror    3.445153 0.067310674
10         War    3.576170 0.063705681
11       SciFi    3.449790 0.062673755
12      Comedy    3.460799 0.051665228
13     Romance    3.559448 0.046984096
14     Western    3.557026 0.044562400
15       Crime    3.551316 0.038851859
16     Fantasy    3.490574 0.021890355
17     Musical    3.503401 0.009062916
18    Children    3.518288 0.005824441
```

Individual genres show clear variations in their average ratings, indicating a correlation with the mean rating.

Analyzing our database sparsity ratio

What's the sparsity raito of our database?

```
sparsity <- (1 - (nrow(edx)/(n_distinct(edx$movieId) *
    n_distinct(edx$userId)))) * 100
sparsity
```

```
[1] 98.7937
```

The database sparsity ratio is around 98%.

What would be the best filtering combinations (movieId & userId), to reduce the sparsity ratio to 90%?

```
movie_counts <- edx%>% #Calculate movie rating quantiles
  group_by(movieId)%>%
  summarise(n= n())
u<-seq(from=0,to=1,by=.1)
movie_percent<-as.data.frame(quantile(movie_counts$n,u))
colnames(movie_percent)<-"movie_filter"
movie_percent
```

```
     movie_filter
0%            1.0
10%          10.0
20%          23.0
30%          40.0
40%          70.0
50%         122.0
60%         210.0
70%         400.0
80%         834.0
90%        2155.4
100%      31336.0
```

```
user_counts <- edx%>% #Calculate user rating quantiles
  group_by(userId)%>%
  summarise(n= n())
u<-seq(from=0,to=1,by=.1)
user_percent<-as.data.frame(quantile(user_counts$n,u))
user_percent
```

```
     quantile(user_counts$n, u)
0%                           13
10%                          22
20%                          28
30%                          36
40%                          47
50%                          62
60%                          85
70%                         116
80%                         175
```

```
90%                          301
100%                        6637
```

```
colnames(user_percent)<-"user_filter"
colnames(user_percent)
```

```
[1] "user_filter"
```

```
# Create all possible combinations of user and movie filters
filters_combinations<-expand.grid(user_percent$user_filter,movie_percent$movie_filter)
colnames(filters_combinations)<-c("user_filter","movie_filter")
filters_combinations<-left_join(filters_combinations,movie_percent,by="movie_filter")
#Calculate sparsity and data usage for each filtering combination
sparseness_study<-mapply(function(user, movie) {
  edx_reduced<-edx%>%group_by(userId)%>%filter(n() >= user)%>%ungroup()%>%
    group_by(movieId)%>%filter(n() >= movie)%>%ungroup()
  n_movies<-n_distinct(edx_reduced$movieId)
  n_users<-n_distinct(edx_reduced$userId)
  data.frame(sparseness=c((1-(nrow(edx_reduced)/(n_movies*n_users)))*100),
             data_used=c((nrow(edx_reduced)/nrow(edx)*100)))
}, filters_combinations$user_filter, filters_combinations$movie_filter)

sparseness_study<-as.data.frame(t(sparseness_study))
sparseness_study<-sparseness_study%>%
  mutate(user_filter=filters_combinations$user_filter,movie_filter=filters_combinations$movie_filter)
#Filtering all the combination resulting to a sparsity =< 90%
best_filters<-sparseness_study%>%filter(!sparseness=="NaN")%>%
  mutate(sparseness=as.numeric(sparseness),data_used=as.numeric(data_used))%>%
  filter(sparseness<90)%>%
  arrange(desc(data_used))
best_filters
```

```
   sparseness   data_used user_filter movie_filter
1    89.27793 66.37329458          85        834.0
2    89.98702 65.87723128          28       2155.4
3    88.83760 63.45209216          36       2155.4
4    87.30196 60.09930377          47       2155.4
5    86.85246 58.73601301         116        834.0
6    85.30076 55.84019931          62       2155.4
7    87.59583 54.49076401         175        400.0
8    82.49527 50.11733809          85       2155.4
9    82.74388 47.81302038         175        834.0
10   78.35113 42.50611190         116       2155.4
11   88.33238 42.06957042         301        122.0
12   85.62312 40.22398293         301        210.0
13   80.90580 36.56770771         301        400.0
14   71.10842 31.83132870         175       2155.4
15   72.86091 29.77537597         301        834.0
16   53.77064 15.03217589         301       2155.4
17    0.00000  0.34817542          13      31336.0
18    0.00000  0.07374394        6637          1.0
```

To achieve a sparsity level of 90%, around 33.7% of the data must be filtered out.

Data analysis conclusion
Through our data analysis, we identified several parameters correlated with ratings:

“movieId”

“userId”

“movie_year” (movie release year)

“rating_delay” (difference between rating year and movie year release)

“genres”

We observed that our dataset contains a large number of rows, with over 9 million ratings.
With 69.8K users and 10.6K movies, this represents a total of 746 million possible ratings.
Our dataset is highly sparse, with an sparsity ratio around 98%.
Reducing sparsity to an optimal level (~90%) for methods like Singular Value Decomposition (SVD), would require filtering out a significant part of the data (around 33.7%).

Based on the above observations, we should :

Avoid using models heavily affected by sparsity, such as Singular Value Decomposition (SVD).

Avoid a model requiring a lot of computing resources, given the volume of data in our database, such as Linear Models (LM).

Instead, favor a manual decomposition approach, calculating each parameter bias independently (movieId, userId, movieYear, rating_delay, genres).

Model building

Buidling a train and a test set with a test set on 20% of the data

```
set.seed(1)
indexes <- split(1:nrow(edx), edx$userId)
test_ind <- sapply(indexes, function(ind) {
    sample(ind, ceiling(length(ind) * 0.2))
}) %>%
    unlist(use.names = TRUE) %>%
    sort()
test_set <- as.data.frame(edx[test_ind, ])
train_set <- as.data.frame(edx[-test_ind, ])
# deleting the movies that are not in the two
# sets.
test_set <- test_set %>%
    semi_join(train_set, by = "movieId")
train_set <- train_set %>%
    semi_join(test_set, by = "movieId")
```

Model 1 : Naive RMSE

```
# Calculate mean rating of the train_set
# (excluding NA values)
mu <- mean(train_set$rating, na.rm = TRUE)
# Calculating RMSE using global mu as prediction
# on our test_set
naive_rmse <- RMSE(test_set$rating, mu)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = "Just the average",
    RMSE = naive_rmse)
rmse_results
```

```
            method     RMSE
1 Just the average 1.060692
```

Model 2 : Adding the movie effect

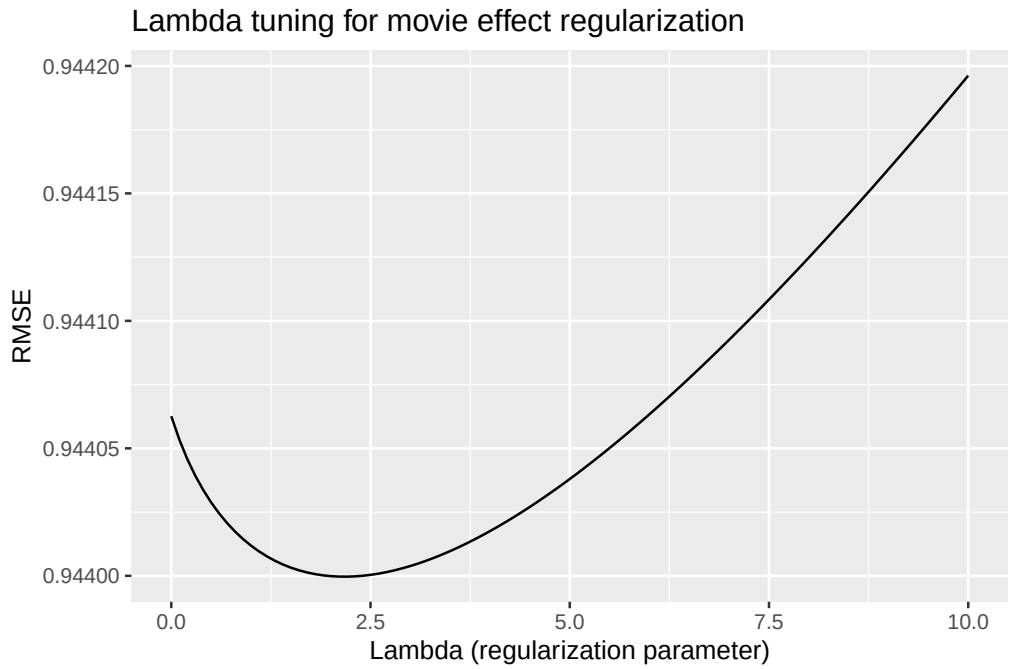
```
# Calculating the average deviation of each movie
# (movie_effect) from global average (mu).
movie_effect <- train_set %>%
    group_by(movieId) %>%
    summarize(m_ef = mean(rating) - mu)
# Creating our prediction by adding the movie
# effect (m_ef) to the global average (mu) for
# each movie of the test_set.
pred_movie <- left_join(test_set, movie_effect, by = "movieId") %>%
    mutate(pred = mu + m_ef) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_1 <- RMSE(test_set$rating, pred_movie)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Just the average",
    "Movie effect"), RMSE = c(naive_rmse, RMSE_1))
rmse_results
```

```
            method      RMSE
1 Just the average 1.0606916
2     Movie effect 0.9440626
```

Model 3 : Regularizing the movie effect

As seen in the data analysis, there is a lot of rating variability for movies with only few ratings.
Let's take in consideration this variability in our model by reducing the movie effect on the movies with few ratings.
Regularization can help us to do so.

```
# Calculating the sum of deviations of each movie
# from global average (mu), and the number of
# rating per movie.
movie_effect_sum <- train_set %>%
    group_by(movieId) %>%
    summarize(m_ef_sum = sum(rating - mu), n = n())
# Tuning the regularization parameter (lambda)
lambdas <- seq(0, 10, 0.1)  # Sequence of lambda values to test
# Applying each lambda values for regularization
# and calculating RMSE for each.
rmses <- sapply(lambdas, function(lambda) {
    movie_effect_reg <- movie_effect_sum %>%
        mutate(m_ef_reg = m_ef_sum/(n + lambda)) %>%
        select(movieId, m_ef_reg)
    left_join(test_set, movie_effect_reg, by = "movieId") %>%
        mutate(pred = mu + m_ef_reg) %>%
        summarize(rmse = RMSE(rating, pred)) %>%
        pull(rmse)
})
# Plot RMSEs across different lambda values to
# visualize the optimal choice
qplot(lambdas, rmses, geom = "line") + labs(title = "Lambda tuning for movie effect regularization",
    x = "Lambda (regularization parameter)", y = "RMSE")
```




```
# Selecting the optimal lambda
best_lambda_movie <- lambdas[which.min(rmses)]
```

```
# Updating regularized model with the optimal
# lambda
movie_effect_reg <- movie_effect_sum %>%
    mutate(m_ef_reg = m_ef_sum/(n + best_lambda_movie)) %>%
    select(movieId, m_ef_reg)
pred_movie_reg <- left_join(test_set, movie_effect_reg,
    by = "movieId") %>%
    mutate(pred = mu + m_ef_reg) %>%
    # Limiting the prediction to valid rating
    # range (0 to 5)
mutate(pred = ifelse(pred < 0, 0, ifelse(pred > 5,
    5, pred))) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_2 <- RMSE(test_set$rating, pred_movie_reg)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Just the average",
    "Movie effect", "Movie effect regularized"), RMSE = c(naive_rmse,
    RMSE_1, RMSE_2))
rmse_results
```

```
                    method      RMSE
1          Just the average 1.0606916
2              Movie effect 0.9440626
3 Movie effect regularized 0.9439997
```

Model 4 : Adding user effect

```
# Calculating the average deviation of each user
# (user_effect) from global average (mu).
user_effect <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    group_by(userId) %>%
    # Calculating the user_effect by removing mu
    # and the movie effect from each user mean
    # rating.
summarize(u_ef = mean(rating - m_ef_reg) - mu)
# Creating our prediction by adding the
# regularized movie effect (m_ef_reg) and the
# user effect (u_ef) to the global average (mu),
# respectively on each movieId, and userId of the
# test_set.
pred_user <- test_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect, by = "userId") %>%
    mutate(pred = mu + m_ef_reg + u_ef) %>%
    # Limiting the prediction to valid rating
    # range (0 to 5)
```

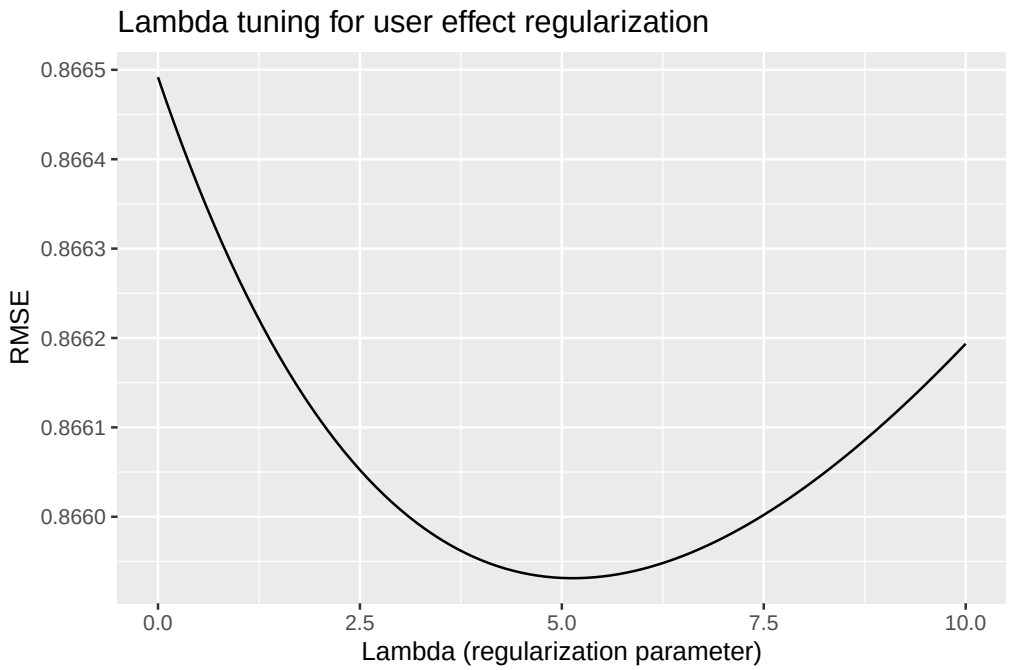
```r
mutate(pred = ifelse(pred < 0, 0, ifelse(pred > 5,
    5, pred))) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_3 <- RMSE(test_set$rating, pred_user)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Just the average",
    "Movie effect", "Movie effect regularized", "Movie (reg) + User effect"),
    RMSE = c(naive_rmse, RMSE_1, RMSE_2, RMSE_3))
rmse_results
```

```
                     method      RMSE
1          Just the average 1.0606916
2              Movie effect 0.9440626
3  Movie effect regularized 0.9439997
4 Movie (reg) + User effect 0.8663007
```

Model 5 : Regularizing the user effect

As seen in the data analysis, there is a lot of rating variability for the users with only few ratings
Let's take in consideration this variability, by regularizing our user effect.

```r
# Calculating the sum of deviations of each user
# from global average (mu), and the number of
# rating per user.
user_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(u_ef_sum = sum(rating - m_ef_reg - mu),
        n = n())
# Tuning the regularization parameter (lambda)
lambdas <- seq(0, 10, 0.1)  # Sequence of lambda values to test
# Applying each lambda values for regularization
# and calculating RMSE for each.
rmses <- sapply(lambdas, function(lambda) {
    user_effect_reg <- user_effect_sum %>%
        mutate(u_ef_reg = u_ef_sum/(n + lambda)) %>%
        select(userId, u_ef_reg)
    pred_movie_reg <- test_set %>%
        left_join(movie_effect_reg, by = "movieId") %>%
        left_join(user_effect_reg, by = "userId") %>%
        mutate(pred = mu + m_ef_reg + u_ef_reg) %>%
        summarize(rmse = RMSE(rating, pred)) %>%
        pull(rmse)
})
# Plot RMSEs across different lambda values to
# visualize the optimal choice
qplot(lambdas, rmses, geom = "line") + labs(title = "Lambda tuning for user effect regularization",
    x = "Lambda (regularization parameter)", y = "RMSE")
```




```
# Selecting the optimal lambda
best_lambda_user <- lambdas[which.min(rmses)]
# Updating regularized model with the optimal
# lambda
user_effect_reg <- user_effect_sum %>%
    mutate(u_ef_reg = u_ef_sum/(n + best_lambda_user)) %>%
    select(userId, u_ef_reg)
pred_user_reg <- test_set %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    mutate(pred = mu + m_ef_reg + u_ef_reg) %>%
    # Limiting the prediction to valid rating
    # range (0 to 5)
mutate(pred = ifelse(pred < 0, 0, ifelse(pred > 5,
    5, pred))) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_4 <- RMSE(test_set$rating, pred_user_reg)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Target: RMSE < ",
    "Just the average", "Movie effect", "Movie effect regularized",
    "Movie (reg) + User effect", "Movie+User effect (both regularized)"),
    RMSE = c(0.8649, naive_rmse, RMSE_1, RMSE_2, RMSE_3,
        RMSE_4))
rmse_results
```

```
                              method      RMSE
1                    Target: RMSE <  0.8649000
2                   Just the average 1.0606916
3                       Movie effect 0.9440626
```

```
4             Movie effect regularized 0.9439997
5            Movie (reg) + User effect 0.8663007
6 Movie+User effect (both regularized) 0.8658278
```

Model 6 : Add the movie's year effect

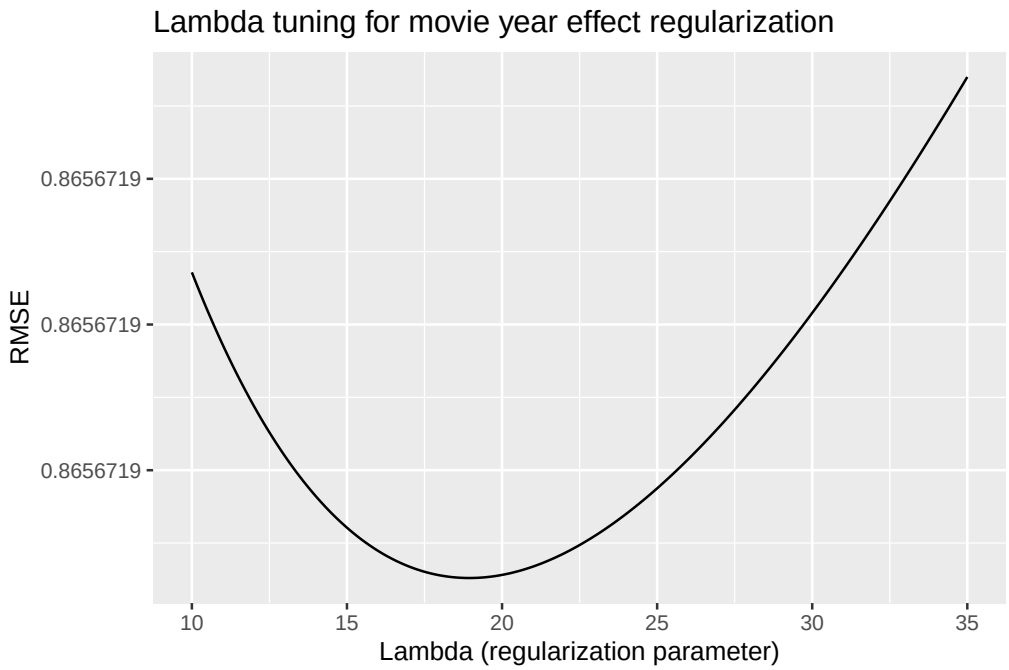
```
# Calculating the average deviation of each movie
# year (movie year effect) from global average
# (mu).
year_effect <- train_set %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    group_by(year_movie) %>%
    # Calculating the movie_year effect by
    # removing mu and previous effect from each
    # mean rating.
summarize(y_ef = mean(rating - u_ef_reg - m_ef_reg) -
    mu)
# Creating our prediction by adding all previous
# effect and the movie year effect (y_ef) to the
# global average (mu) on the test_set
pred_year <- left_join(test_set, year_effect, by = "year_movie") %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_5 <- RMSE(test_set$rating, pred_year)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Target: RMSE < ",
    "Just the average", "Movie effect", "Movie effect regularized",
    "Movie (reg) + User effect", "Movie+User effect (both regularized)",
    "Movie+User effect (both regularized) + year"),
    RMSE = c(0.8649, naive_rmse, RMSE_1, RMSE_2, RMSE_3,
        RMSE_4, RMSE_5))
rmse_results
```

```
                                      method      RMSE
1                           Target: RMSE <  0.8649000
2                           Just the average 1.0606916
3                               Movie effect 0.9440626
4                    Movie effect regularized 0.9439997
5                   Movie (reg) + User effect 0.8663007
6       Movie+User effect (both regularized) 0.8658278
7 Movie+User effect (both regularized) + year 0.8656719
```

Model 7 : Regularizing the movie year effect

As seen in the data analysis, there are many more ratings for the recent movies.
Let's take in consideration the variability due to the movie year.

```
# Calculating the sum of deviations of each
# movie_year from global average (mu), and the
# number of rating per movie_year.
year_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect_reg, by = "userId") %>%
    group_by(year_movie) %>%
    summarize(y_ef_sum = sum(rating - m_ef_reg - u_ef_reg -
        mu), n = n())
# Tuning the regularization parameter (lambda)
lambdas <- seq(10, 35, 0.1)  # Sequence of lambda values to test
# Applying each lambda values for regularization
# and calculating RMSE for each.
rmses <- sapply(lambdas, function(lambda) {
    year_effect_reg <- year_effect_sum %>%
        mutate(y_ef_reg = y_ef_sum/(n + lambda)) %>%
        select(year_movie, y_ef_reg)
    pred_year_reg <- test_set %>%
        left_join(movie_effect_reg, by = "movieId") %>%
        left_join(user_effect_reg, by = "userId") %>%
        left_join(year_effect_reg, by = "year_movie") %>%
        mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg) %>%
        summarize(rmse = RMSE(rating, pred)) %>%
        pull(rmse)
})
# Plot RMSEs across different lambda values to
# visualize the optimal choice
qplot(lambdas, rmses, geom = "line") + labs(title = "Lambda tuning for movie year effect regularization
    x = "Lambda (regularization parameter)", y = "RMSE")
```




```
# Selecting the optimal lambda
best_lambda_year <- lambdas[which.min(rmses)]
# Updating regularized model with the optimal
# lambda
year_effect_reg <- year_effect_sum %>%
    mutate(y_ef_reg = y_ef_sum/(n + best_lambda_year)) %>%
    select(year_movie, y_ef_reg)
pred_year_reg <- test_set %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg) %>%
    # Limiting the prediction to valid rating
    # range (0 to 5)
mutate(pred = ifelse(pred < 0, 0, ifelse(pred > 5,
    5, pred))) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_6 <- RMSE(test_set$rating, pred_year_reg)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Target: RMSE < ",
    "Just the average", "Movie effect", "Movie effect regularized",
    "Movie (reg) + User effect", "Movie+User effect (both regularized)",
    "Movie+User effect (both regularized) + year",
    "Movie+User+year effect (all regularized)"), RMSE = c(0.8649,
    naive_rmse, RMSE_1, RMSE_2, RMSE_3, RMSE_4, RMSE_5,
    RMSE_6))
rmse_results
```

```
                                    method      RMSE
1                          Target: RMSE <  0.8649000
```

```
2                            Just the average 1.0606916
3                                Movie effect 0.9440626
4                     Movie effect regularized 0.9439997
5                    Movie (reg) + User effect 0.8663007
6          Movie+User effect (both regularized) 0.8658278
7 Movie+User effect (both regularized) + year 0.8656719
8    Movie+User+year effect (all regularized) 0.8655682
```

Model 8 : Add the rating delay effect

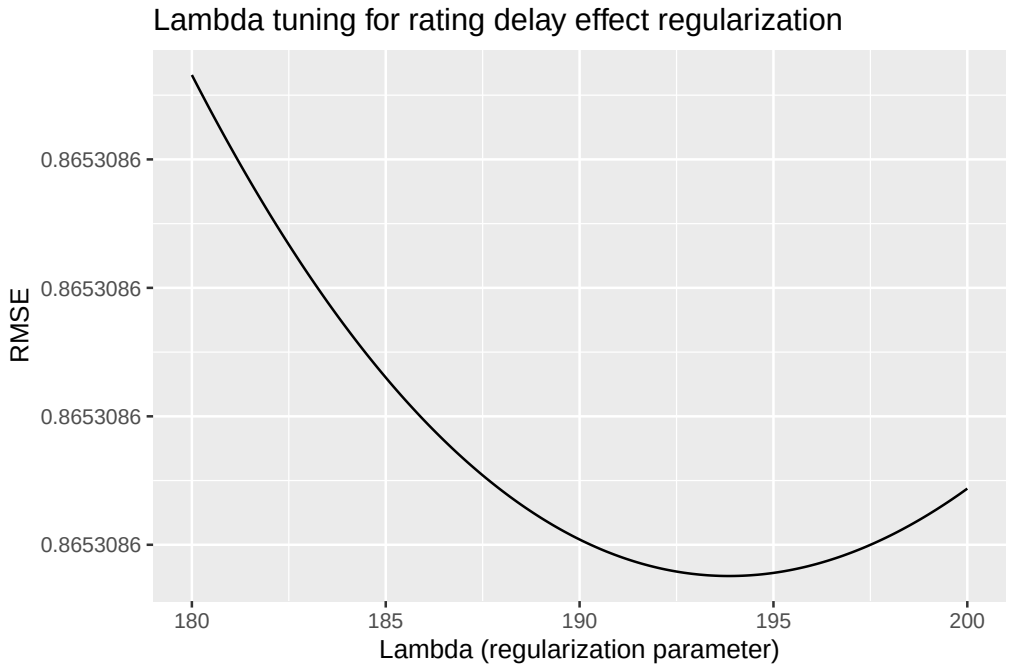
```
# Calculating the average deviation of each
# rating delay (rating_delay effect) from global
# average (mu).
delay_effect <- train_set %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    group_by(rating_delay) %>%
    # Calculating the rating delay effect by
    # removing mu and previous effect from each
    # mean rating.
summarize(d_ef = mean(rating - u_ef_reg - m_ef_reg -
    y_ef_reg) - mu)
# Creating our prediction by adding all previous
# effect and the rating effect (d_ef) to the
# global average (mu), on the test_set
pred_delay <- test_set %>%
    left_join(delay_effect, by = "rating_delay") %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg +
        d_ef) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_7 <- RMSE(test_set$rating, pred_delay)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Target: RMSE < ",
    "Just the average", "Movie effect", "Movie effect regularized",
    "Movie (reg) + User effect", "Movie+User effect (both regularized)",
    "Movie+User effect (both regularized) + year",
    "Movie+User+year effect (all regularized)", "Movie+User+year effect (all regularized) + comment dela
    RMSE = c(0.8649, naive_rmse, RMSE_1, RMSE_2, RMSE_3,
        RMSE_4, RMSE_5, RMSE_6, RMSE_7))
rmse_results
```

```
                                          method      RMSE
1                                Target: RMSE <  0.8649000
```

```
2                                        Just the average 1.0606916
3                                            Movie effect 0.9440626
4                                 Movie effect regularized 0.9439997
5                                Movie (reg) + User effect 0.8663007
6                      Movie+User effect (both regularized) 0.8658278
7              Movie+User effect (both regularized) + year 0.8656719
8                 Movie+User+year effect (all regularized) 0.8655682
9 Movie+User+year effect (all regularized) + comment delay 0.8653094
```

Model 9 : Regularizing the rating delay effect

```
# Calculating the sum of deviations of each
# rating_delay from global average (mu), and the
# number of rating per rating_delay
delay_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    group_by(rating_delay) %>%
    summarize(d_ef_sum = sum(rating - m_ef_reg - u_ef_reg -
        y_ef_reg - mu), n = n())
# Tuning the regularization parameter (lambda)
lambdas <- seq(180, 200, 0.1)  # Sequence of lambda values to test
# Applying each lambda values for regularization
# and calculating RMSE for each.
rmses <- sapply(lambdas, function(lambda) {
    delay_effect_reg <- delay_effect_sum %>%
        mutate(d_ef_reg = d_ef_sum/(n + lambda)) %>%
        select(rating_delay, d_ef_reg)
    pred_delay_reg <- test_set %>%
        left_join(movie_effect_reg, by = "movieId") %>%
        left_join(user_effect_reg, by = "userId") %>%
        left_join(year_effect_reg, by = "year_movie") %>%
        left_join(delay_effect_reg, by = "rating_delay") %>%
        mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg +
            d_ef_reg) %>%
        summarize(rmse = RMSE(rating, pred)) %>%
        pull(rmse)
})
# Plot RMSEs across different lambda values to
# visualize the optimal choice
qplot(lambdas, rmses, geom = "line") + labs(title = "Lambda tuning for rating delay effect regularizati
    x = "Lambda (regularization parameter)", y = "RMSE")
```




```
# Selecting the optimal lambda
best_lambda_delay <- lambdas[which.min(rmses)]
# Updating regularized model with the optimal
# lambda
delay_effect_reg <- delay_effect_sum %>%
    mutate(d_ef_reg = d_ef_sum/(n + best_lambda_delay)) %>%
    select(rating_delay, d_ef_reg)
pred_delay_reg <- test_set %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    left_join(delay_effect_reg, by = "rating_delay") %>%
    mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg +
        d_ef_reg) %>%
    # Limiting the prediction to valid rating
    # range (0 to 5)
mutate(pred = ifelse(pred < 0, 0, ifelse(pred > 5,
    5, pred))) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_8 <- RMSE(test_set$rating, pred_delay_reg)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Target: RMSE < ",
    "Just the average", "Movie effect", "Movie effect regularized",
    "Movie (reg) + User effect", "Movie+User effect (both regularized)",
    "Movie+User effect (both regularized) + year",
    "Movie+User+year effect (all regularized)", "Movie+User+year effect (all regularized) + comment dela
    "Movie+User+year+comment delay effect (all regularized)"),
    RMSE = c(0.8649, naive_rmse, RMSE_1, RMSE_2, RMSE_3,
        RMSE_4, RMSE_5, RMSE_6, RMSE_7, RMSE_8))
rmse_results
```

```
                                                          method      RMSE
1                                              Target: RMSE <  0.8649000
2                                             Just the average 1.0606916
3                                                Movie effect 0.9440626
4                                  Movie effect regularized 0.9439997
5                                 Movie (reg) + User effect 0.8663007
6                       Movie+User effect (both regularized) 0.8658278
7                  Movie+User effect (both regularized) + year 0.8656719
8                     Movie+User+year effect (all regularized) 0.8655682
9  Movie+User+year effect (all regularized) + comment delay 0.8653094
10   Movie+User+year+comment delay effect (all regularized) 0.8652041
```

Model 10 : adding the genres effect

```
# Calculating the average deviation of each genre
# (genres_effect) from global average (mu).
genre_effect <- train_set %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    left_join(delay_effect_reg, by = "rating_delay") %>%
    group_by(genres) %>%
    # Calculating the genres effect by removing
    # mu and previous effect from each mean
    # rating.
summarize(g_ef = mean(rating - u_ef_reg - m_ef_reg -
    y_ef_reg - d_ef_reg) - mu)
# Creating our prediction by adding all previous
# effect and the rating effect (g_ef) to the
# global average (mu), on the test_set
pred_genre <- test_set %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    left_join(delay_effect_reg, by = "rating_delay") %>%
    left_join(genre_effect, by = "genres") %>%
    mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg +
        d_ef_reg + g_ef) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_9 <- RMSE(test_set$rating, pred_genre)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Target: RMSE < ",
    "Just the average", "Movie effect", "Movie effect regularized",
    "Movie (reg) + User effect", "Movie+User effect (both regularized)",
    "Movie+User effect (both regularized) + year",
```

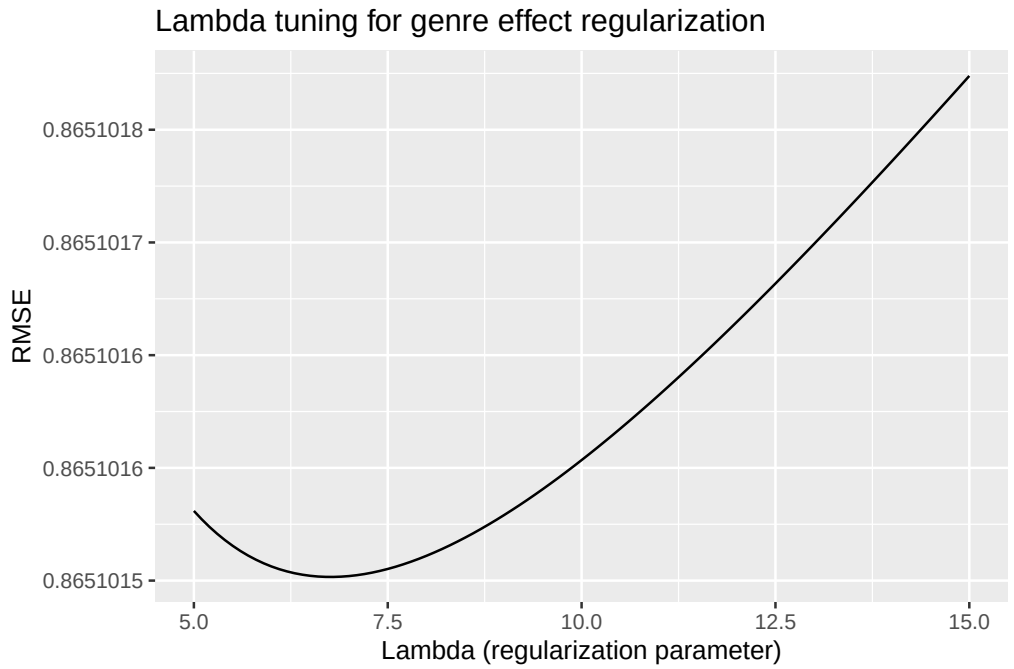
```
    "Movie+User+year effect (all regularized)", "Movie+User+year effect (all regularized) + comment dela
    "Movie+User+year+comment delay effect (all regularized)",
    "Movie+User+year+comment delay effect (all regularized) + genres"),
    RMSE = c(0.8649, naive_rmse, RMSE_1, RMSE_2, RMSE_3,
        RMSE_4, RMSE_5, RMSE_6, RMSE_7, RMSE_8, RMSE_9))
rmse_results
```

```
                                                                 method      RMSE
1                                                  Target: RMSE <  0.8649000
2                                                 Just the average 1.0606916
3                                                    Movie effect 0.9440626
4                                             Movie effect regularized 0.9439997
5                                            Movie (reg) + User effect 0.8663007
6                                  Movie+User effect (both regularized) 0.8658278
7                           Movie+User effect (both regularized) + year 0.8656719
8                              Movie+User+year effect (all regularized) 0.8655682
9             Movie+User+year effect (all regularized) + comment delay 0.8653094
10              Movie+User+year+comment delay effect (all regularized) 0.8652041
11 Movie+User+year+comment delay effect (all regularized) + genres 0.8651047
```

Model 11 : Regularizing the genres effect

```
# Calculating the sum of deviations of each
# genres from global average (mu), and the number
# of rating per genres
genre_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    left_join(delay_effect_reg, by = "rating_delay") %>%
    group_by(genres) %>%
    summarize(g_ef_sum = sum(rating - m_ef_reg - u_ef_reg -
        y_ef_reg - d_ef_reg - mu), n = n())
# Tuning the regularization parameter (lambda)
lambdas <- seq(5, 15, 0.1)  # Sequence of lambda values to test
# Applying each lambda values for regularization
# and calculating RMSE for each.
rmses <- sapply(lambdas, function(lambda) {
    genre_effect_reg <- genre_effect_sum %>%
        mutate(g_ef_reg = g_ef_sum/(n + lambda)) %>%
        select(genres, g_ef_reg)
    pred_genre_reg <- test_set %>%
        left_join(movie_effect_reg, by = "movieId") %>%
        left_join(user_effect_reg, by = "userId") %>%
        left_join(year_effect_reg, by = "year_movie") %>%
        left_join(delay_effect_reg, by = "rating_delay") %>%
        left_join(genre_effect_reg, by = "genres") %>%
        mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg +
            d_ef_reg + g_ef_reg) %>%
        summarize(rmse = RMSE(rating, pred)) %>%
```

```
        pull(rmse)
})
# Plot RMSEs across different lambda values to
# visualize the optimal choice
qplot(lambdas, rmses, geom = "line") + labs(title = "Lambda tuning for genre effect regularization",
    x = "Lambda (regularization parameter)", y = "RMSE")
```




```
# Selecting the optimal lambda
best_lambda_genre <- lambdas[which.min(rmses)]
# Updating regularized model with the optimal
# lambda
genre_effect_reg <- genre_effect_sum %>%
    mutate(g_ef_reg = g_ef_sum/(n + best_lambda_genre)) %>%
    select(genres, g_ef_reg)
pred_genre_reg <- test_set %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    left_join(delay_effect_reg, by = "rating_delay") %>%
    left_join(genre_effect_reg, by = "genres") %>%
    mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg +
        d_ef_reg + g_ef_reg) %>%
    # Limiting the prediction to valid rating
    # range (0 to 5)
mutate(pred = ifelse(pred < 0, 0, ifelse(pred > 5,
    5, pred))) %>%
    pull(pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_10 <- RMSE(test_set$rating, pred_genre_reg)
# Saving the RMSE result into a data frame for
# future comparison
```

```r
rmse_results <- data.frame(method = c("Target: RMSE < ",
    "Just the average", "Movie effect", "Movie effect regularized",
    "Movie (reg) + User effect", "Movie+User effect (both regularized)",
    "Movie+User effect (both regularized) + year",
    "Movie+User+year effect (all regularized)", "Movie+User+year effect (all regularized) + comment dela
    "Movie+User+year+comment delay effect (all regularized)",
    "Movie+User+year+comment delay effect (all regularized) + genres",
    "Movie+User+year+comment delay+genres effect (all regularized)"),
    RMSE = c(0.8649, naive_rmse, RMSE_1, RMSE_2, RMSE_3,
        RMSE_4, RMSE_5, RMSE_6, RMSE_7, RMSE_8, RMSE_9,
        RMSE_10))
rmse_results
```

```
                                                                 method      RMSE
1                                                      Target: RMSE <  0.8649000
2                                                     Just the average 1.0606916
3                                                        Movie effect 0.9440626
4                                                Movie effect regularized 0.9439997
5                                               Movie (reg) + User effect 0.8663007
6                                    Movie+User effect (both regularized) 0.8658278
7                           Movie+User effect (both regularized) + year 0.8656719
8                              Movie+User+year effect (all regularized) 0.8655682
9            Movie+User+year effect (all regularized) + comment delay 0.8653094
10             Movie+User+year+comment delay effect (all regularized) 0.8652041
11 Movie+User+year+comment delay effect (all regularized) + genres 0.8651047
12   Movie+User+year+comment delay+genres effect (all regularized) 0.8649914
```

Model 12 : Cross validation on the final model

```r
# Making a cross validation training on our model
# applying all regularized effect.  Creating
# different random seeds, to generate different
# train and test set.
seeds <- c(1, 220, 3330, 44440, 555550)
# Making a cross-validation loop across all
# different seeds
all_preds <- sapply(seeds, function(seed) {
    set.seed(seed)
    # Buidling a train and a test set with a test
    # set on 20% of the data
    indexes <- split(1:nrow(edx), edx$userId)
    test_ind <- sapply(indexes, function(ind) {
        sample(ind, ceiling(length(ind) * 0.2))
    }) %>%
        unlist(use.names = TRUE) %>%
        sort()
    test_set <- as.data.frame(edx[test_ind, ])
    train_set <- as.data.frame(edx[-test_ind, ])
    # deleting the movies that aren't in the two
    # sets.
```

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

###### Calculating the regularized movie
###### effect
movie_effect_sum <- train_set %>%
    group_by(movieId) %>%
    summarize(m_ef_sum = sum(rating - mu), n = n())

movie_effect_reg <- movie_effect_sum %>%
    mutate(m_ef_reg = m_ef_sum/(n + best_lambda_movie)) %>%
    select(movieId, m_ef_reg)

###### Calculating the regularized user
###### effect
user_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(u_ef_sum = sum(rating - m_ef_reg -
        mu), n = n())

user_effect_reg <- user_effect_sum %>%
    mutate(u_ef_reg = u_ef_sum/(n + best_lambda_user)) %>%
    select(userId, u_ef_reg)

###### Calculating the regularized movie year
###### effect
year_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect_reg, by = "userId") %>%
    group_by(year_movie) %>%
    summarize(y_ef_sum = sum(rating - m_ef_reg -
        u_ef_reg - mu), n = n())

year_effect_reg <- year_effect_sum %>%
    mutate(y_ef_reg = y_ef_sum/(n + best_lambda_year)) %>%
    select(year_movie, y_ef_reg)

###### Calculating the regularized delay
###### effect
delay_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    group_by(rating_delay) %>%
    summarize(d_ef_sum = sum(rating - m_ef_reg -
        u_ef_reg - y_ef_reg - mu), n = n())

delay_effect_reg <- delay_effect_sum %>%
    mutate(d_ef_reg = d_ef_sum/(n + best_lambda_delay)) %>%
    select(rating_delay, d_ef_reg)
```

```
    ###### Calculating the regularized genre
    ###### effect
    genre_effect_sum <- train_set %>%
        left_join(movie_effect_reg, by = "movieId") %>%
        left_join(user_effect_reg, by = "userId") %>%
        left_join(year_effect_reg, by = "year_movie") %>%
        left_join(delay_effect_reg, by = "rating_delay") %>%
        group_by(genres) %>%
        summarize(g_ef_sum = sum(rating - m_ef_reg -
            u_ef_reg - y_ef_reg - d_ef_reg - mu), n = n())

    genre_effect_reg <- genre_effect_sum %>%
        mutate(g_ef_reg = g_ef_sum/(n + best_lambda_genre)) %>%
        select(genres, g_ef_reg)

    #### Making the predictions
    prediction <- test_set %>%
        left_join(user_effect_reg, by = "userId") %>%
        left_join(movie_effect_reg, by = "movieId") %>%
        left_join(year_effect_reg, by = "year_movie") %>%
        left_join(delay_effect_reg, by = "rating_delay") %>%
        left_join(genre_effect_reg, by = "genres") %>%
        mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg +
            d_ef_reg + g_ef_reg) %>%
        # Limiting the prediction to valid rating
        # range (0 to 5)
    mutate(pred = ifelse(pred < 0, 0, ifelse(pred >
        5, 5, pred))) %>%
        select(userId, movieId, pred)
})
# Creating dataframes for each prediction of the
# loop
pred1 <- data.frame(userId = all_preds[, 1]$userId,
    movieId = all_preds[, 1]$movieId, pred = all_preds[,
        1]$pred)
pred2 <- data.frame(userId = all_preds[, 2]$userId,
    movieId = all_preds[, 2]$movieId, pred = all_preds[,
        2]$pred)
pred3 <- data.frame(userId = all_preds[, 3]$userId,
    movieId = all_preds[, 3]$movieId, pred = all_preds[,
        3]$pred)
pred4 <- data.frame(userId = all_preds[, 4]$userId,
    movieId = all_preds[, 4]$movieId, pred = all_preds[,
        4]$pred)
pred5 <- data.frame(userId = all_preds[, 5]$userId,
    movieId = all_preds[, 5]$movieId, pred = all_preds[,
        5]$pred)
# Adding all predictions to the test_set
all_preds <- test_set %>%
    select(movieId, userId) %>%
    left_join(pred1, by = c("movieId", "userId")) %>%
    left_join(pred2, by = c("movieId", "userId")) %>%
    left_join(pred3, by = c("movieId", "userId")) %>%
```

```
    left_join(pred4, by = c("movieId", "userId")) %>%
    left_join(pred5, by = c("movieId", "userId"))
# Adding column names to the dataframe
colnames(all_preds) <- c("movieId", "userId", "pred1",
    "pred2", "pred3", "pred4", "pred5")
# Creating the average prediction from the 5
# predictions
pred <- all_preds %>%
    mutate(mean_pred = rowMeans(all_preds[, 3:7], na.rm = TRUE)) %>%
    pull(mean_pred)
# Calculating RMSE using our prediction on our
# test_set
RMSE_11 <- RMSE(test_set$rating, pred)
# Saving the RMSE result into a data frame for
# future comparison
rmse_results <- data.frame(method = c("Target: RMSE < ",
    "Just the average", "Movie effect", "Movie effect regularized",
    "Movie (reg) + User effect", "Movie+User effect (both regularized)",
    "Movie+User effect (both regularized) + year",
    "Movie+User+year effect (all regularized)", "Movie+User+year effect (all regularized) + comment dela
    "Movie+User+year+comment delay effect (all regularized)",
    "Movie+User+year+comment delay effect (all regularized) + genres",
    "Movie+User+year+comment delay+genres effect (all regularized)",
    "Cross validation on the last Model"), RMSE = c(0.8649,
    naive_rmse, RMSE_1, RMSE_2, RMSE_3, RMSE_4, RMSE_5,
    RMSE_6, RMSE_7, RMSE_8, RMSE_9, RMSE_10, RMSE_11))

rmse_results
```

```
                                                          method      RMSE
1                                                 Target: RMSE <  0.8649000
2                                                Just the average 1.0606916
3                                                   Movie effect 0.9440626
4                                          Movie effect regularized 0.9439997
5                                         Movie (reg) + User effect 0.8663007
6                                Movie+User effect (both regularized) 0.8658278
7                           Movie+User effect (both regularized) + year 0.8656719
8                              Movie+User+year effect (all regularized) 0.8655682
9            Movie+User+year effect (all regularized) + comment delay 0.8653094
10             Movie+User+year+comment delay effect (all regularized) 0.8652041
11 Movie+User+year+comment delay effect (all regularized) + genres 0.8651047
12   Movie+User+year+comment delay+genres effect (all regularized) 0.8649914
13                               Cross validation on the last Model 0.8647048
```

Model evaluation

Evaluating our cross validation model on “final_holdout_test”.

```r
# Preparing the data (creating year_movie,
# rating_year and rating_delay)
year_movie <- str_extract_all(final_holdout_test$title,
    "\\([0-9]{4}\\)$", simplify = TRUE)
year_movie <- as.numeric(str_extract_all(year_movie,
    "[0-9]{4}", simplify = TRUE))

final_holdout_test <- final_holdout_test %>%
    mutate(year_movie = year_movie, rating_year = year(as.POSIXct(final_holdout_test$timestamp,
        origin, tz = "UTC")), rating_delay = rating_year -
        year_movie)
# Creating different random seeds, to generate
# different train and test set.
seeds <- c(1, 220, 3330, 44440, 555550)
# Making a cross-validation loop across all
# different seeds
all_preds <- sapply(seeds, function(seed) {
    set.seed(seed)
    # Buidling a train and a test set with a test
    # set on 20% of the data
    indexes <- split(1:nrow(edx), edx$userId)
    test_ind <- sapply(indexes, function(ind) {
        sample(ind, ceiling(length(ind) * 0.2))
    }) %>%
        unlist(use.names = TRUE) %>%
        sort()
    test_set <- as.data.frame(edx[test_ind, ])
    train_set <- as.data.frame(edx[-test_ind, ])
    # deleting the movies that aren't in the two
    # sets.
    test_set <- test_set %>%
        semi_join(train_set, by = "movieId")
    train_set <- train_set %>%
        semi_join(test_set, by = "movieId")

    ###### Calculating the regularized movie
    ###### effect
    movie_effect_sum <- train_set %>%
        group_by(movieId) %>%
        summarize(m_ef_sum = sum(rating - mu), n = n())

    movie_effect_reg <- movie_effect_sum %>%
        mutate(m_ef_reg = m_ef_sum/(n + best_lambda_movie)) %>%
        select(movieId, m_ef_reg)

    ###### Calculating the regularized user
    ###### effect
    user_effect_sum <- train_set %>%
        left_join(movie_effect_reg, by = "movieId") %>%
        group_by(userId) %>%
        summarize(u_ef_sum = sum(rating - m_ef_reg -
            mu), n = n())
```

```
user_effect_reg <- user_effect_sum %>%
    mutate(u_ef_reg = u_ef_sum/(n + best_lambda_user)) %>%
    select(userId, u_ef_reg)

###### Calculating the regularized movie year
###### effect
year_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect_reg, by = "userId") %>%
    group_by(year_movie) %>%
    summarize(y_ef_sum = sum(rating - m_ef_reg -
        u_ef_reg - mu), n = n())

year_effect_reg <- year_effect_sum %>%
    mutate(y_ef_reg = y_ef_sum/(n + best_lambda_year)) %>%
    select(year_movie, y_ef_reg)

###### Calculating the regularized delay
###### effect
delay_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    group_by(rating_delay) %>%
    summarize(d_ef_sum = sum(rating - m_ef_reg -
        u_ef_reg - y_ef_reg - mu), n = n())

delay_effect_reg <- delay_effect_sum %>%
    mutate(d_ef_reg = d_ef_sum/(n + best_lambda_delay)) %>%
    select(rating_delay, d_ef_reg)

###### Calculating the regularized genre
###### effect
genre_effect_sum <- train_set %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    left_join(delay_effect_reg, by = "rating_delay") %>%
    group_by(genres) %>%
    summarize(g_ef_sum = sum(rating - m_ef_reg -
        u_ef_reg - y_ef_reg - d_ef_reg - mu), n = n())

genre_effect_reg <- genre_effect_sum %>%
    mutate(g_ef_reg = g_ef_sum/(n + best_lambda_genre)) %>%
    select(genres, g_ef_reg)

#### Making the predictions
prediction <- final_holdout_test %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(year_effect_reg, by = "year_movie") %>%
    left_join(delay_effect_reg, by = "rating_delay") %>%
    left_join(genre_effect_reg, by = "genres") %>%
```

```
        mutate(pred = mu + m_ef_reg + u_ef_reg + y_ef_reg +
            d_ef_reg + g_ef_reg) %>%
        # Limiting the prediction to valid rating
        # range (0 to 5)
    mutate(pred = ifelse(pred < 0, 0, ifelse(pred >
        5, 5, pred))) %>%
        select(userId, movieId, pred)
})
# Creating dataframes for each prediction of the
# loop
pred1 <- data.frame(userId = all_preds[, 1]$userId,
    movieId = all_preds[, 1]$movieId, pred = all_preds[,
        1]$pred)
pred2 <- data.frame(userId = all_preds[, 2]$userId,
    movieId = all_preds[, 2]$movieId, pred = all_preds[,
        2]$pred)
pred3 <- data.frame(userId = all_preds[, 3]$userId,
    movieId = all_preds[, 3]$movieId, pred = all_preds[,
        3]$pred)
pred4 <- data.frame(userId = all_preds[, 4]$userId,
    movieId = all_preds[, 4]$movieId, pred = all_preds[,
        4]$pred)
pred5 <- data.frame(userId = all_preds[, 5]$userId,
    movieId = all_preds[, 5]$movieId, pred = all_preds[,
        5]$pred)
# Adding all predictions to the
# final_holdout_test'
all_preds <- final_holdout_test %>%
    select(movieId, userId) %>%
    left_join(pred1, by = c("movieId", "userId")) %>%
    left_join(pred2, by = c("movieId", "userId")) %>%
    left_join(pred3, by = c("movieId", "userId")) %>%
    left_join(pred4, by = c("movieId", "userId")) %>%
    left_join(pred5, by = c("movieId", "userId"))
# Adding column names to the dataframe
colnames(all_preds) <- c("movieId", "userId", "pred1",
    "pred2", "pred3", "pred4", "pred5")
# Creating the average prediction from the 5
# predictions
pred <- all_preds %>%
    mutate(mean_pred = rowMeans(all_preds[, 3:7], na.rm = TRUE)) %>%
    pull(mean_pred)
# Identifying NAs positions in our pred data
# frame
NAs <- which(is.na(pred), arr.ind = TRUE)
# Defining 'mu' as the train_set global rating
# average
mu <- mean(train_set$rating, na.rm = TRUE)
pred[NAs] <- mu  #replacing pred NAs by 'mu'
# Calculating RMSE using our prediction on 'our
# test_set'final_holdout_test'
final_RMSE <- RMSE(final_holdout_test$rating, pred)
# Comparing our RMSE result with our RMSE target
```

```r
rmse_results <- data.frame(method = c("Target RMSE < ",
    "Final model ="), RMSE = c(0.8649, final_RMSE))
rmse_results
```

```
          method      RMSE
1 Target RMSE <  0.8649000
2  Final model = 0.8641906
```

Conclusion

By analyzing all dataset parameters, we identified several correlations with movie ratings.
This allowed us to construct a handmade Linear Model by calculating the bias associated with each correlated parameter.
By applying cross-validation to this model, we achieved an RMSE below our target (around 8.855 $\times$ 10^-4)

To further improve our model, several approaches can be considered:

Incorporating title size effect:
We could analyze the impact of the number of characters in movie titles on ratings and include this effect in our model.

Optimizing cross-validation batches:
Currently, our model uses a 5-batch cross-validation strategy.
Testing different numbers of batches (e.g., 10, 20, or more) might help optimize the model between underfitting and overfitting.

Adding additional parameters:
Integrating additional features from the original dataset (e.g., user profiles, location) could refine the bias estimation and improve predictive performance.

Reducing sparsity with more data:
Increasing the volume of data in our dataset would reduce sparsity, potentially permit us the use of advanced methods like matrix factorization techniques (eg. Singular Value Decomposition (SVD)).

All the improvement above could lead to a more accurate and robust recommendation model.