# MovieLens Project: Prediction of Movie Ratings

Jonas Graf

11 - August - 2022

## Contents



---

# 1 Data Setup

```
# Libraries: install
if (!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if (!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if (!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if (!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if (!require(kableExtra)) install.packages("kableExtra", repos = "http://cran.us.r-project.org")
if (!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org")
if (!require(tidyr)) install.packages("tidyr", repos = "http://cran.us.r-project.org")
if (!require(forcats)) install.packages("forcats", repos = "http://cran.us.r-project.org")
if (!require(formatR)) install.packages("formatR", repos = "http://cran.us.r-project.org")

# Libraries: load
library(tidyverse)
library(caret)
library(data.table)
library(dplyr)
library(ggplot2)
library(kableExtra)
library(tidyr)
library(stringr)
library(forcats)
library(formatR)

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

# Setting options to allow download
options(timeout = max(10000, getOption("timeout")))

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

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

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

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

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

# Validation set will be 10% of MovieLens data
```

```r
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1,
    p = 0.1, list = FALSE)
edx <- movielens[-test_index, ]
temp <- movielens[test_index, ]

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

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

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

---

## 2 Introduction

The overarching aim of this project is to predict movie ratings (rating scores 0-5) using a subset of the MovieLens dataset.

This subset contains ~10,000,000 movie ratings. For this project, these ratings will be divided into 9,000,000 (90%) for training and 1,000,000 (10%) for validation purposes. The training dataset comprises ratings from ~70,000 users evaluating ~11,000 different movies. Of note, the rated movies are grouped in genres, e.g. Adventure, Action, Drama, Horror, and Thriller.

First, the data subset will be described. Second, a rating prediction model will be fitted on the training dataset. Third, the model deemed optimal will be applied to the validation dataset. The fitted model will be evaluated according to the Root Mean Squared Error (RMSE), which should be below 0.86490.

$$\boxed{\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i}(\hat{y}_i - y_i)^2}}$$

A regularized model including 'Movie', 'User', 'Genre', 'Date of Rating' as well as 'Year of Release' fulfills the overarching aim of the project by reaching an RMSE of 0.8620.

---

# 3 Methods & Analysis

## 3.1 Descriptive data exploration

First, an exploration of the raw training data is warranted.

### 3.1.1 First six data entries: overview

The head function in R provides us a first descriptive overview of the variables.

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

Movie title and year of release appear to be combined within the 'title' variable. Further, the time point of the rating has been captured as a time stamp.

### 3.1.2 Users and movies (counts) included in the training data subset: table.

| users | movies |
|---|---|
| 69878 | 10677 |

### 3.1.3 Classes of variables:table

| variable | class |
|---|---|
| userId | numeric |
| movieId | numeric |
| rating | integer |
| timestamp | character |
| title | character |
| genres | integer |

### 3.1.4 Formatting tasks

The descriptive data exploration revealed three formatting tasks:
1. Formatting 'timestamp' into readable data format.
2. Splitting the condensed 'genres' categorization.
3. Separating year of release from title.

Of note, 'timestamp' represents seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
*Source: https://files.grouplens.org/datasets/movielens/ml-10m-README.html*

## 3.2 Formatting training & validation datasets

Next, the formatting task need to be tackled.

### 3.2.1 Formatting 'timestamp' into readable data format: result.

```
# 1.) Format 'timestamp' into readable data format
edx$date <- as.POSIXct(edx$timestamp, origin = "1970-01-01")
validation$date <- as.POSIXct(validation$timestamp, origin = "1970-01-01")
## training
edx$year_rating <- format(edx$date, "%Y")
edx$month_rating <- format(edx$date, "%m")
edx$day_rating <- format(edx$date, "%d")
edx$date_rating <- paste(edx$year_rating, "-", edx$month_rating,
    "-", edx$day_rating)

## validation
validation$year_rating <- format(validation$date, "%Y")
validation$month_rating <- format(validation$date, "%m")
validation$day_rating <- format(validation$date, "%d")
validation$date_rating <- paste(validation$year_rating, "-",
    validation$month_rating, "-", validation$day_rating)

## Result after task 1
kable(head(edx), format = "latex") %>%
    kable_styling(latex_options = c("scale_down", "HOLD_position"))
```

| userId | movieId | rating | timestamp | title | genres | date | year_rating | month_rating | day_rating | date_rating |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 122 | 5 | 838985046 | Boomerang (1992) | Comedy\|Romance | 1996-08-02 13:24:06 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 185 | 5 | 838983525 | Net, The (1995) | Action\|Crime\|Thriller | 1996-08-02 12:58:45 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 292 | 5 | 838983421 | Outbreak (1995) | Action\|Drama\|Sci-Fi\|Thriller | 1996-08-02 12:57:01 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 316 | 5 | 838983392 | Stargate (1994) | Action\|Adventure\|Sci-Fi | 1996-08-02 12:56:32 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 329 | 5 | 838983392 | Star Trek: Generations (1994) | Action\|Adventure\|Drama\|Sci-Fi | 1996-08-02 12:56:32 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 355 | 5 | 838984474 | Flintstones, The (1994) | Children\|Comedy\|Fantasy | 1996-08-02 13:14:34 | 1996 | 08 | 02 | 1996 - 08 - 02 |

### 3.2.2 Splitting the 'genres' variable: result.

```
# 2.) Split the condensed 'genres' categorization.
edx <- edx %>%
    separate_rows(genres, sep = "\\|")
validation <- validation %>%
    separate_rows(genres, sep = "\\|")
## Result after tasks 1 & 2
kable(head(edx), format = "latex") %>%
    kable_styling(latex_options = c("scale_down", "HOLD_position"))
```

| userId | movieId | rating | timestamp | title | genres | date | year_rating | month_rating | day_rating | date_rating |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 122 | 5 | 838985046 | Boomerang (1992) | Comedy | 1996-08-02 13:24:06 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 122 | 5 | 838985046 | Boomerang (1992) | Romance | 1996-08-02 13:24:06 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 185 | 5 | 838983525 | Net, The (1995) | Action | 1996-08-02 12:58:45 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 185 | 5 | 838983525 | Net, The (1995) | Crime | 1996-08-02 12:58:45 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 185 | 5 | 838983525 | Net, The (1995) | Thriller | 1996-08-02 12:58:45 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 292 | 5 | 838983421 | Outbreak (1995) | Action | 1996-08-02 12:57:01 | 1996 | 08 | 02 | 1996 - 08 - 02 |

### 3.2.3 Separating year of release from title variable.

```
edx <- edx %>%
    # Removing whitespace
mutate(title = str_trim(title)) %>%
    # Extracting year of release
extract(title, c("titleTemp", "year_release"), remove = FALSE,
    regex = "^(.*) \\(([0-9 \\-]*)\\)$") %>%
    # Handling years with '-'
mutate(year_release = if_else(str_length(year_release) > 4, as.integer(str_split(year_release,
    "-", simplify = TRUE)[1]), as.integer(year_release))) %>%
    # Redefining title variable
mutate(title = if_else(is.na(titleTemp), title, titleTemp)) %>%
    select(-titleTemp)

validation <- validation %>%
    mutate(title = str_trim(title)) %>%
    extract(title, c("titleTemp", "year_release"), remove = FALSE,
        regex = "^(.*) \\(([0-9 \\-]*)\\)$") %>%
    mutate(year_release = if_else(str_length(year_release) >
        4, as.integer(str_split(year_release, "-", simplify = TRUE)[1]),
        as.integer(year_release))) %>%
    mutate(title = if_else(is.na(titleTemp), title, titleTemp)) %>%
    select(-titleTemp)

## Result after tasks 1, 2 & 3
kable(head(edx), format = "latex") %>%
    kable_styling(latex_options = c("scale_down", "HOLD_position"))
```

| userId | movieId | rating | timestamp | title | year_release | genres | date | year_rating | month_rating | day_rating | date_rating |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 122 | 5 | 838985046 | Boomerang | 1992 | Comedy | 1996-08-02 13:24:06 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 122 | 5 | 838985046 | Boomerang | 1992 | Romance | 1996-08-02 13:24:06 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 185 | 5 | 838983525 | Net, The | 1995 | Action | 1996-08-02 12:58:45 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 185 | 5 | 838983525 | Net, The | 1995 | Crime | 1996-08-02 12:58:45 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 185 | 5 | 838983525 | Net, The | 1995 | Thriller | 1996-08-02 12:58:45 | 1996 | 08 | 02 | 1996 - 08 - 02 |
| 1 | 292 | 5 | 838983421 | Outbreak | 1995 | Action | 1996-08-02 12:57:01 | 1996 | 08 | 02 | 1996 - 08 - 02 |

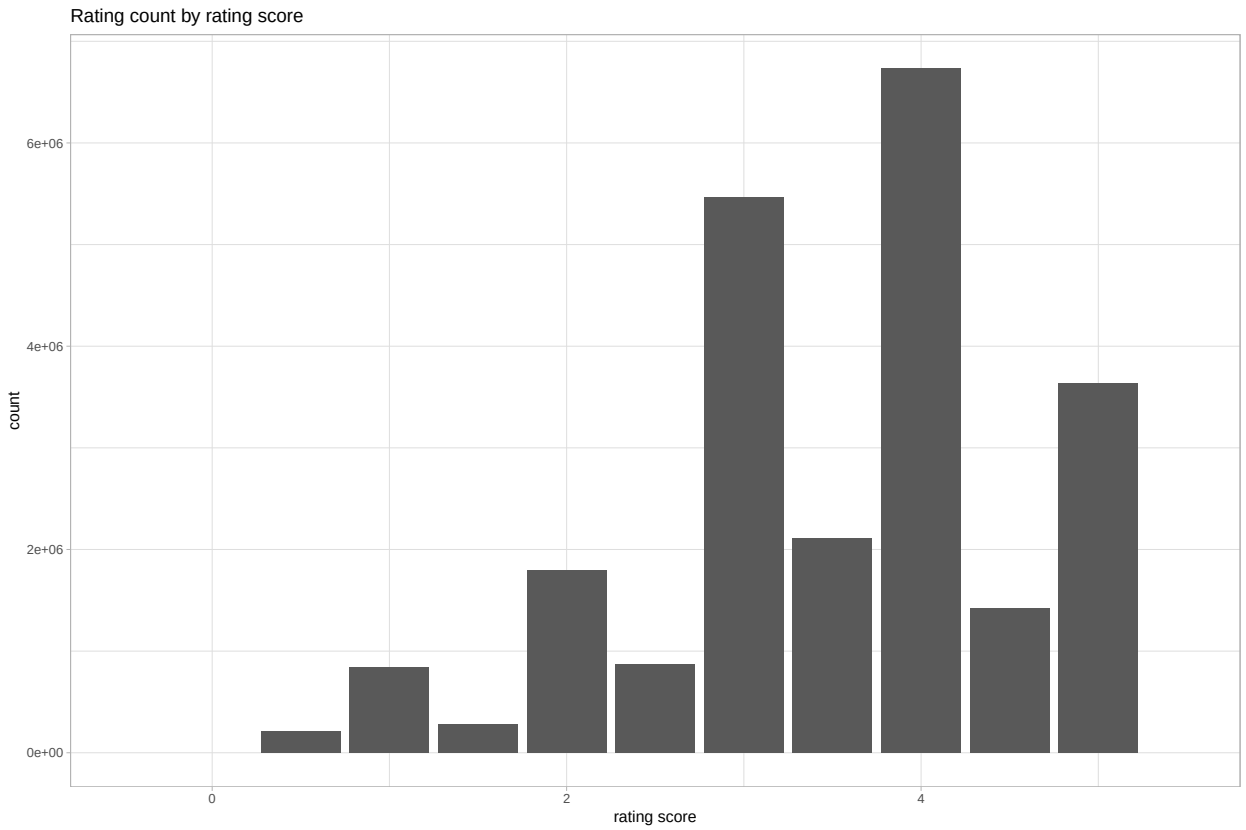
### 3.2.4 Converting & selecting columns of interest: result.

| userId | movieId | rating | title | genres | date_rating | year_rating | year_release |
|---|---|---|---|---|---|---|---|
| 1 | 122 | 5 | Boomerang | Comedy | 1996 - 08 - 02 | 1996 | 1992 |
| 1 | 122 | 5 | Boomerang | Romance | 1996 - 08 - 02 | 1996 | 1992 |
| 1 | 185 | 5 | Net, The | Action | 1996 - 08 - 02 | 1996 | 1995 |
| 1 | 185 | 5 | Net, The | Crime | 1996 - 08 - 02 | 1996 | 1995 |
| 1 | 185 | 5 | Net, The | Thriller | 1996 - 08 - 02 | 1996 | 1995 |
| 1 | 292 | 5 | Outbreak | Action | 1996 - 08 - 02 | 1996 | 1995 |

## 3.3 Visualization

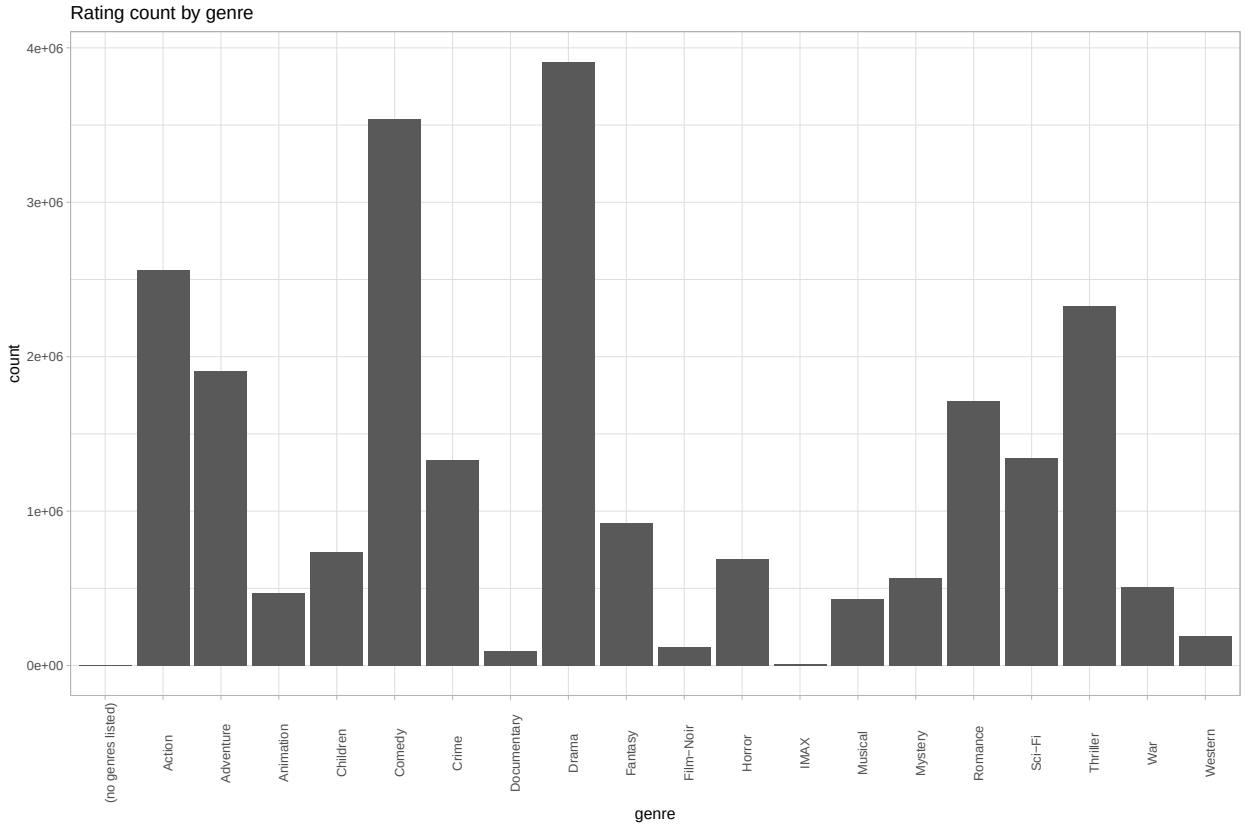
Next, providing a visualization of the formatted training dataset is warranted.

### 3.3.1 Ratings: count by rating score



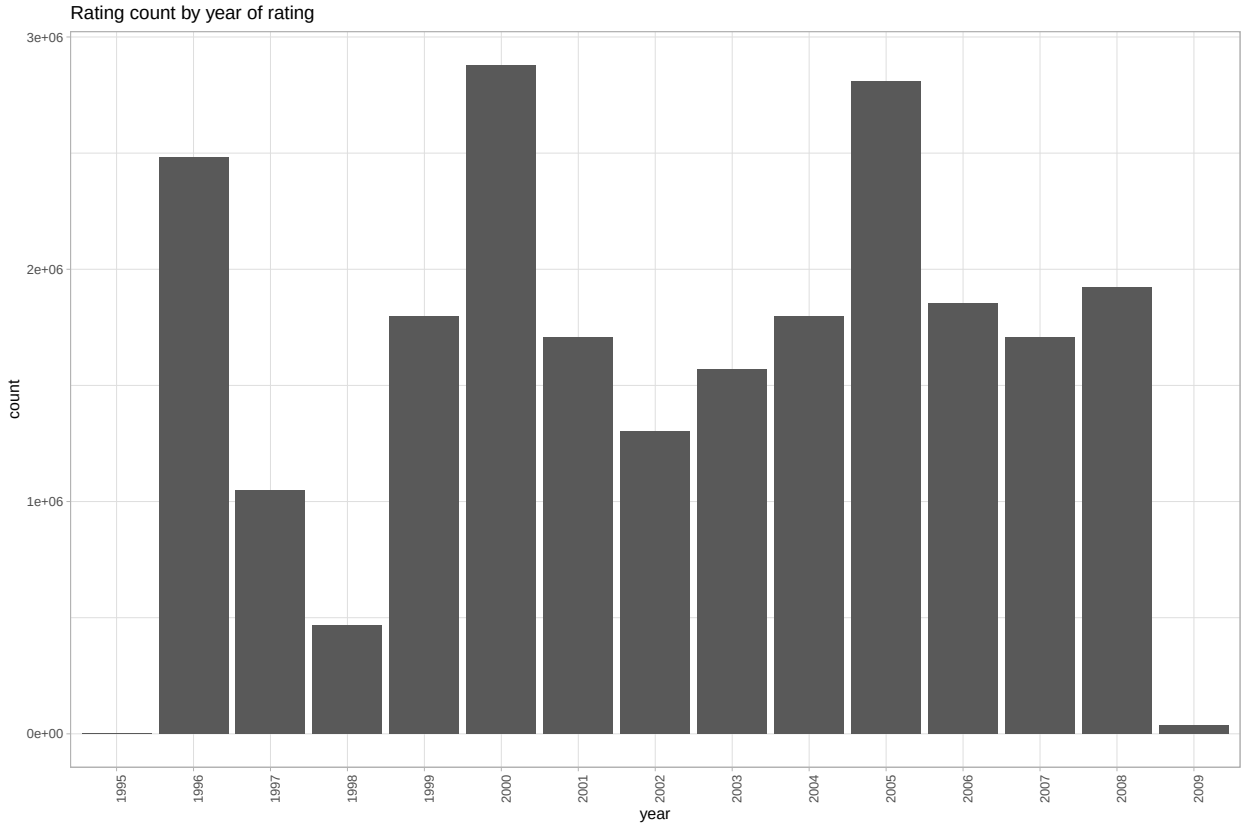

Visualization of the rating count by rating score revealed that negative ratings < 3 are less common than ratings ≥3 and above.

### 3.3.2 Ratings: count by genre



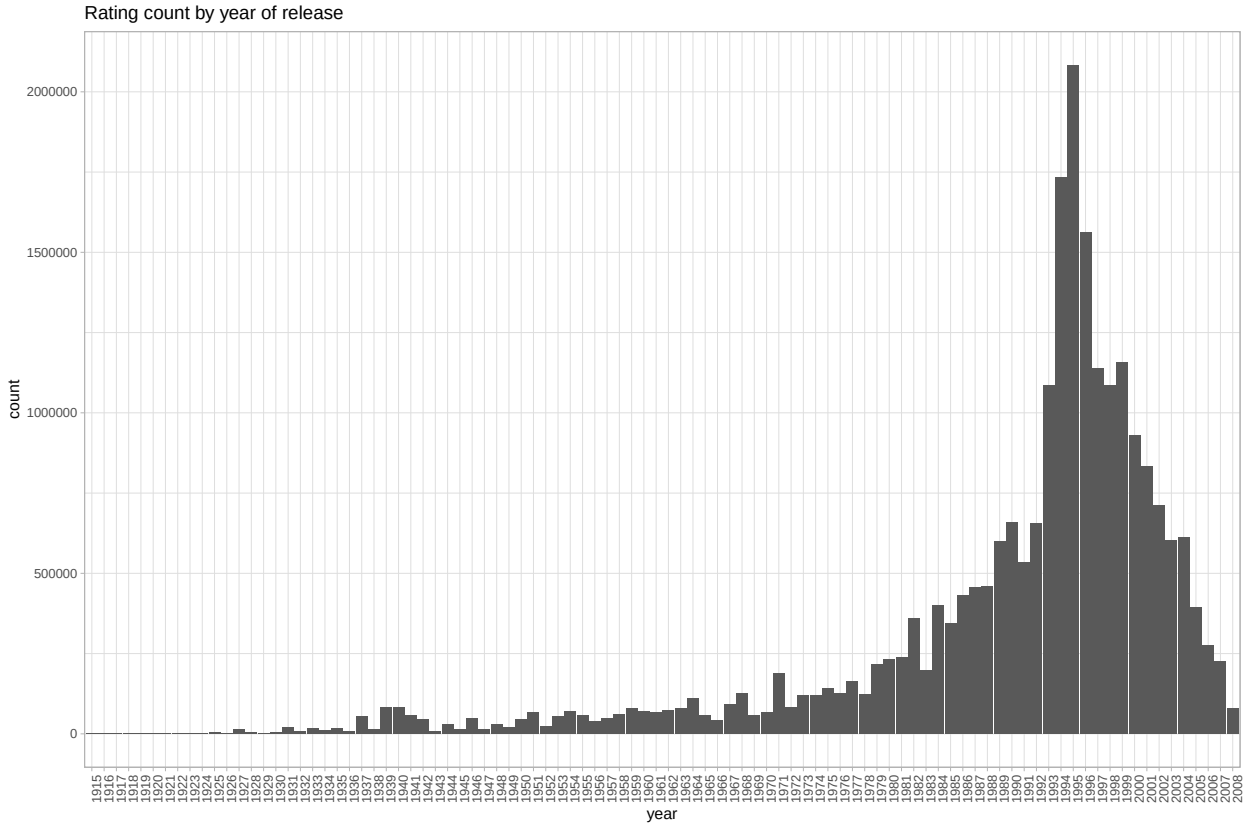

Visualization of the rating count by genre revealed that some genres may be more prone to ratings than others.

### 3.3.3 Ratings: count by year of rating




Visualization of the rating count by year of the rating revealed that some years are associated with more ratings than others.

### 3.3.4 Ratings: count by year of movie release




Visualization of the rating count by year of the movie release revealed that younger movies are associated with more ratings than older movies.

## 3.4 Models: building & evaluation

First, a partition of the edx set for training/testing purposes needs to be created.

```
# Creating data partition of edx for cross-validation
# Validation set will be 10% of edx data
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1,
    p = 0.1, list = FALSE)
train_edx <- edx[-test_index, ]
temp <- edx[test_index, ]
# Make sure userId and movieId in validation set are also
# in edx set
test_edx <- temp %>%
    semi_join(train_edx, by = "movieId") %>%
    semi_join(train_edx, by = "userId")
# Add the Rows removed from the test_edx back into
# train_edx
removed <- anti_join(temp, test_edx)
```

```
## Joining, by = c("userId", "movieId", "rating", "title", "genres",
## "date_rating", "year_rating", "year_release")
```

```
train_edx <- rbind(train_edx, removed)
rm(temp, test_index, removed)
```

### 3.4.1 Predicting the mean rating

Always predicting the mean rating is perhaps the simplest model. In the training dataset, the mean is ~3.5.

```
mean(train_edx$rating)
```

```
## [1] 3.527007
```

```
# Mean rating of all movies
mu_hat <- mean(train_edx$rating)
# Predict the RMSE on the test_edx set
mean_model_result <- RMSE(test_edx$rating, mu_hat)
# Gathering RMSE results in a dataframe
results <- data.frame(model = "_Mean_ **test**", RMSE = mean_model_result)
results %>%
    kable(format = "simple", align = "c")
```

| model | RMSE |
|:---:|:---:|
| *Mean* **test** | 1.051741 |

An RMSE of >1 underscores a poor model performance.

### 3.4.2 Considering the movie

The goal is to improve the 'predicting the mean' model by considering a possible movie effect.

The following equation depicts this approach:

$$\boxed{Y_{u,i} = \hat{\mu} + b_i + \epsilon_{u,i}}$$

In short, $\hat{\mu}$ represents the mean rating and $\varepsilon_{i,u}$ the independent errors. The $b_i$ represents the magnitude of the movie effect $i$.

```
# Mean rating of all movies
mu_hat <- mean(train_edx$rating)
# Calculating the mean by user
movie_avgs <- train_edx %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu_hat))
# Computing predicted ratings on test_edx dataset
movie_model <- test_edx %>%
    left_join(movie_avgs, by = "movieId") %>%
    mutate(pred = mu_hat + b_i) %>%
    pull(pred)
mean_movie_result <- RMSE(movie_model, test_edx$rating)
# Expanding the results dataframe
results <- results %>%
    add_row(model = "_Mean_+_Movie_ **test**", RMSE = mean_movie_result)
results %>%
    kable(format = "simple", align = "c")
```

| model | RMSE |
|---|---|
| *Mean* **test** | 1.051741 |
| *Mean+Movie* **test** | 0.940685 |

When considering the movie effect, the RMSE is 0.9662. Given that this value is still above the target RMSE, the performance of this model does not suffice.

### 3.4.3 Considering movie & user

The RMSE is still too high. Hence, further considerations are warranted. Here, a user effect is being accounted for.

The following equation depicts this more complex approach:

$$\boxed{Y_{u,i} = \hat{\mu} + b_u + b_i + \epsilon_{u,i}}$$

In addition to the movie equation above, $b_u$ represents the magnitude of a potential user effect $u$.

```
# Mean rating of all movies
mu_hat <- mean(train_edx$rating)
# Calculating mean by movie
movie_avgs <- train_edx %>%
```

```
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu_hat))
# Calculating mean by user
user_avgs <- train_edx %>%
    left_join(movie_avgs, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat - b_i))
# Computing predicted ratings on test_edx dataset
mean_user_movie_model <- test_edx %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>%
    mutate(pred = mu_hat + b_u + b_i) %>%
    pull(pred)
mean_user_movie_model_result <- RMSE(mean_user_movie_model, test_edx$rating)
# Expanding results dataframe
results <- results %>%
    add_row(model = "_Mean_+_Movie_+_User_ **test**", RMSE = mean_user_movie_model_result)
results %>%
    kable(format = "simple", align = "c")
```

| model | RMSE |
|---|---|
| *Mean* **test** | 1.0517408 |
| *Mean+Movie* **test** | 0.9406850 |
| *Mean+Movie+User* **test** | 0.8568071 |

When considering both user and movie, the RMSE falls close to our target value with 0.8568.

#### 3.4.4 Considering user, movie & genre

The following equation represents this model:

$$\boxed{Y_{u,i} = \hat{\mu} + b_i + b_u + b_g + \epsilon_{u,i}}$$

In addition to the previous model, the $b_{u,g}$ represents the magnitude of a given genre effect $u, g$ on a given user.

```
# Mean rating of all movies
mu_hat <- mean(train_edx$rating)
# Calculating mean by movie
movie_avgs <- train_edx %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu_hat))
# Calculating mean by user
user_avgs <- train_edx %>%
    left_join(movie_avgs, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat - b_i))
# Calculating mean by genre
genre_avgs <- train_edx %>%
    left_join(user_avgs, by = "userId") %>%
```

```
    left_join(movie_avgs, by = "movieId") %>%
    group_by(genres) %>%
    summarize(b_g = mean(rating - mu_hat - b_u - b_i))
# Computing predicted ratings on test_edx dataset
mean_user_movie_genre_model <- test_edx %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(genre_avgs, by = "genres") %>%
    mutate(pred = mu_hat + b_u + b_i + b_g) %>%
    pull(pred)
mean_user_movie_genre_model_result <- RMSE(mean_user_movie_genre_model,
    test_edx$rating)
# Expanding results dataset
results <- results %>%
    add_row(model = "_Mean_+_Movie_+_User_+_Genre_ **test**",
        RMSE = mean_user_movie_genre_model_result)
results %>%
    kable(format = "simple", align = "c")
```

| model | RMSE |
|---|---|
| *Mean* **test** | 1.0517408 |
| *Mean+Movie* **test** | 0.9406850 |
| *Mean+Movie+User* **test** | 0.8568071 |
| *Mean+Movie+User+Genre* **test** | 0.8566977 |

Genre decreases the RMSE. Perhaps including the date of rating into our model may lead to further improvement.

### 3.4.5 Considering movie, user, genre & date of rating

The following equation represents this model:

$$\boxed{Y_{u,i} = \hat{\mu} + b_i + b_u + b_g + b_{dra} + \epsilon_{u,i}}$$

```
# Mean rating of all movies
mu_hat <- mean(train_edx$rating)
# Calculate the mean by movie
movie_avgs <- train_edx %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu_hat))
# Calculate the mean by user
user_avgs <- train_edx %>%
    left_join(movie_avgs, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat - b_i))
# Calculating mean by genre
genre_avgs <- train_edx %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(movie_avgs, by = "movieId") %>%
    group_by(genres) %>%
```

```
    summarize(b_g = mean(rating - mu_hat - b_u - b_i))
# Calculating mean by date of rating
date_ra_avgs <- train_edx %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(genre_avgs, by = "genres") %>%
    group_by(date_rating) %>%
    summarize(b_d_ra = mean(rating - mu_hat - b_u - b_i - b_g))
# Compute the predicted ratings on test_edx dataset
mean_user_movie_date_ra_model <- test_edx %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(genre_avgs, by = "genres") %>%
    left_join(date_ra_avgs, by = "date_rating") %>%
    mutate(pred = mu_hat + b_u + b_i + b_d_ra + b_g) %>%
    pull(pred)
mean_user_movie_date_ra_model_result <- RMSE(mean_user_movie_date_ra_model,
    test_edx$rating)
# Expanding results dataframe
results <- results %>%
    add_row(model = "_Mean_+_Movie_+_User_+_Genre_+_DateRa_ **test**",
        RMSE = mean_user_movie_date_ra_model_result)
results %>%
    kable(format = "simple", align = "c")
```

| model | RMSE |
|:---:|:---:|
| *Mean* **test** | 1.0517408 |
| *Mean+Movie* **test** | 0.9406850 |
| *Mean+Movie+User* **test** | 0.8568071 |
| *Mean+Movie+User+Genre* **test** | 0.8566977 |
| *Mean+Movie+User+Genre+DateRa* **test** | 0.8560564 |

Adding the date of the rating to the equation slightly improved (decreased) the RMSE.

#### 3.4.6 Considering user, movie, genre, date of rating & year of release

The following equation represents this expanded model:

$$\boxed{Y_{u,i} = \hat{\mu} + b_i + b_u + b_g + b_{dra} + b_{yre} + \epsilon_{u,i}}$$

```
# Mean rating of all movies
mu_hat <- mean(train_edx$rating)
# Calculating mean by movie
movie_avgs <- train_edx %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu_hat))
# Calculating mean by user
user_avgs <- train_edx %>%
    left_join(movie_avgs, by = "movieId") %>%
```

```
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu_hat - b_i))
# Calculating mean by genre
genre_avgs <- train_edx %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(movie_avgs, by = "movieId") %>%
    group_by(genres) %>%
    summarize(b_g = mean(rating - mu_hat - b_u - b_i))
# Calculating mean by date of rating
date_ra_avgs <- train_edx %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(genre_avgs, by = "genres") %>%
    group_by(date_rating) %>%
    summarize(b_d_ra = mean(rating - mu_hat - b_u - b_i - b_g))
# Calculating mean by year of release
year_re_avgs <- train_edx %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(genre_avgs, by = "genres") %>%
    left_join(date_ra_avgs, by = "date_rating") %>%
    group_by(year_release) %>%
    summarize(b_y_re = mean(rating - b_u - b_i - b_d_ra - b_g -
        mu_hat))
# Computing predicted ratings on test_edx dataset
mean_user_movie_date_ra_model <- test_edx %>%
    left_join(user_avgs, by = "userId") %>%
    left_join(movie_avgs, by = "movieId") %>%
    left_join(genre_avgs, by = "genres") %>%
    left_join(date_ra_avgs, by = "date_rating") %>%
    left_join(year_re_avgs, by = "year_release") %>%
    mutate(pred = mu_hat + b_u + b_i + b_d_ra + b_g + b_y_re) %>%
    pull(pred)
mean_user_movie_date_ra_model_result <- RMSE(mean_user_movie_date_ra_model,
    test_edx$rating)
# Expanding results dataframe
results <- results %>%
    add_row(model = "_Mean_+_Movie_+_User_+_Genre_+_DateRa_+_YearRe_ **test**",
        RMSE = mean_user_movie_date_ra_model_result)
results %>%
    kable(format = "simple", align = "c")
```

| model | RMSE |
|:---:|:---:|
| *Mean* **test** | 1.0517408 |
| *Mean+Movie* **test** | 0.9406850 |
| *Mean+Movie+User* **test** | 0.8568071 |
| *Mean+Movie+User+Genre* **test** | 0.8566977 |
| *Mean+Movie+User+Genre+DateRa* **test** | 0.8560564 |
| *Mean+Movie+User+Genre+DateRa+YearRe* **test** | 0.8557129 |

Given that the last model in the table above showed the best performance, it will be considered for further optimization.

### 3.4.7 Regularization

The regularization approach allows us to adjust for variables (e.g., users and movies) with large estimates which were formed from small sample sizes by adding the term $\lambda$ (lambda). For example, in order to optimize $b_u$ and $b_i$, it is necessary to use this equation:

$$\boxed{\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i)^2+\lambda\sum_i b_i^2}$$

which may be reduced to:

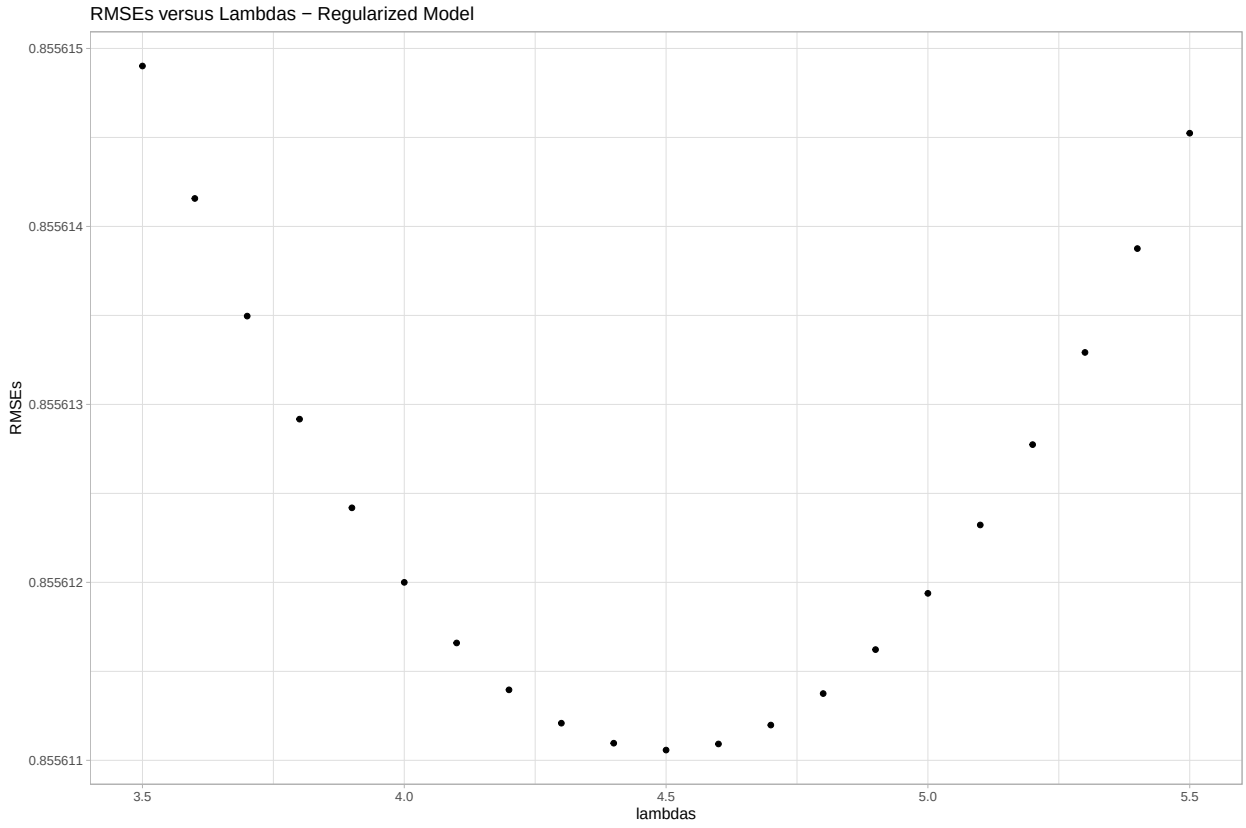
$$\boxed{\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu})}$$

### 3.4.8 Considering user, movie, date of rating, genre & year of release with regularization

```
mu_hat <- mean(train_edx$rating) # Mean of all movies
lambdas <- seq(3.5, 5.5, 0.1) # Defining table of lambdas
# Cross-validation of lambdas
rmses <- sapply(lambdas, function(lambda) {
    b_i <- train_edx %>% # Including movies
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu_hat) / (n() + lambda))

    b_u <- train_edx %>% # Including users
      left_join(b_i, by='movieId') %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - b_i - mu_hat) / (n() + lambda))
    b_g <- train_edx %>% # Including genres
      left_join(b_u, by='userId') %>%
      left_join(b_i, by='movieId') %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating-b_u-b_i-mu_hat) / (n()+lambda))
    b_d_ra <- train_edx %>% # Including date of rating
      left_join(b_u, by='userId') %>%
      left_join(b_i, by='movieId') %>%
      left_join(b_g, by='genres') %>%
      group_by(date_rating) %>%
      summarize(b_d_ra = sum(rating - b_u -b_i - b_g-mu_hat) / (n() + lambda))
    b_y_re <- train_edx %>% # Including year of release
      left_join(b_u, by='userId') %>%
      left_join(b_i, by='movieId') %>%
      left_join(b_g, by='genres') %>%
      left_join(b_d_ra, by='date_rating') %>%
      group_by(year_release) %>%
      summarize(b_y_re = sum(rating-b_u-b_i-b_d_ra-b_g-mu_hat) / (n()+lambda))
    predicted_ratings <- test_edx %>% # Predicting ratings
      left_join(b_u, by='userId') %>%
      left_join(b_i, by='movieId') %>%
```

```
      left_join(b_d_ra, by="date_rating") %>%
      left_join(b_g, by="genres") %>%
      left_join(b_y_re, by="year_release") %>%
      mutate(pred = mu_hat + b_u + b_i + b_g + b_d_ra  + b_y_re) %>%
      pull(pred)
   return(RMSE(predicted_ratings, test_edx$rating))
})
# Plot: RMSEs versus lambdas
df <- data.frame(RMSE = rmses, lambdas = lambdas)
ggplot(df, aes(lambdas, rmses)) +
  geom_point()+
  theme_light()+
  labs(title = "RMSEs versus Lambdas - Regularized Model")+
  labs(x = "lambdas", y = "RMSEs")
```




```
# Identifying lambda value associated with lowest RMSE
lambda_min <- lambdas[which.min(rmses)]
# Predicting RMSE on the test_edx set
reg_model <- min(rmses)
# Expanding results dataframe
results <- results %>% add_row(model="_Mean_+_Movie_+_User_+_Genre_+_DateRa_+_YearRe_+_Reg_ **test**", I
results %>% kable(format = "simple", align = 'c')
```

| model | RMSE |
|---|---|
| *Mean* **test** | 1.0517408 |
| *Mean+Movie* **test** | 0.9406850 |
| *Mean+Movie+User* **test** | 0.8568071 |
| *Mean+Movie+User+Genre* **test** | 0.8566977 |
| *Mean+Movie+User+Genre+DateRa* **test** | 0.8560564 |
| *Mean+Movie+User+Genre+DateRa+YearRe* **test** | 0.8557129 |
| *Mean+Movie+User+Genre+DateRa+YearRe+Reg* **test** | 0.8556111 |

The optimal lambda generated using the training dataset only is:

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

#### 3.4.9 Final model applied to validation dataset

```
# Mean of all movies
mu_hat <- mean(edx$rating)
# Applying lambda min
rmse <- sapply(lambda_min, function(lambda) {
    # Including movies
    b_i <- edx %>%
        group_by(movieId) %>%
        summarize(b_i = sum(rating - mu_hat)/(n() + lambda))
    # Including users
    b_u <- edx %>%
        left_join(b_i, by = "movieId") %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating - b_i - mu_hat)/(n() + lambda))
    # Including genres
    b_g <- train_edx %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_i, by = "movieId") %>%
        group_by(genres) %>%
        summarize(b_g = sum(rating - b_u - b_i - mu_hat)/(n() +
            lambda))
    # Including date of rating
    b_d_ra <- train_edx %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_g, by = "genres") %>%
        group_by(date_rating) %>%
        summarize(b_d_ra = sum(rating - b_u - b_i - b_g - mu_hat)/(n() +
            lambda))
    # Including year of release
    b_y_re <- train_edx %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_g, by = "genres") %>%
        left_join(b_d_ra, by = "date_rating") %>%
        group_by(year_release) %>%
```

```
        summarize(b_y_re = sum(rating - b_u - b_i - b_g - b_d_ra -
            mu_hat)/(n() + lambda))
    # Predicting ratings
    predicted_ratings <- validation %>%
        left_join(b_u, by = "userId") %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_d_ra, by = "date_rating") %>%
        left_join(b_g, by = "genres") %>%
        left_join(b_y_re, by = "year_release") %>%
        mutate(pred = mu_hat + b_u + b_i + b_g + b_d_ra + b_y_re) %>%
        pull(pred)
    return(RMSE(predicted_ratings, validation$rating))
})
# Expanding results dataframe
results <- results %>%
    add_row(model = "_Mean_+_Movie_+_User_+_Genre_+_DateRa_+_YearRe_+_Reg_ **validation**",
        RMSE = rmse)
results %>%
    kable(format = "simple", align = "c")
```

| model | RMSE |
|---|---|
| *Mean* **test** | 1.0517408 |
| *Mean+Movie* **test** | 0.9406850 |
| *Mean+Movie+User* **test** | 0.8568071 |
| *Mean+Movie+User+Genre* **test** | 0.8566977 |
| *Mean+Movie+User+Genre+DateRa* **test** | 0.8560564 |
| *Mean+Movie+User+Genre+DateRa+YearRe* **test** | 0.8557129 |
| *Mean+Movie+User+Genre+DateRa+YearRe+Reg* **test** | 0.8556111 |
| *Mean+Movie+User+Genre+DateRa+YearRe+Reg* **validation** | 0.8620975 |

The regularized User+Movie+Date_Rating+Genre+Year_release model offers a prediction with an RMSE below the desired cutoff.

---

# 4 Results

## 4.1 Table summary

The table below summarizes the results of this project.

| model | RMSE |
|---|---|
| *Mean* **test** | 1.0517408 |
| *Mean+Movie* **test** | 0.9406850 |
| *Mean+Movie+User* **test** | 0.8568071 |
| *Mean+Movie+User+Genre* **test** | 0.8566977 |
| *Mean+Movie+User+Genre+DateRa* **test** | 0.8560564 |
| *Mean+Movie+User+Genre+DateRa+YearRe* **test** | 0.8557129 |
| *Mean+Movie+User+Genre+DateRa+YearRe+Reg* **test** | 0.8556111 |
| *Mean+Movie+User+Genre+DateRa+YearRe+Reg* **validation** | 0.8620975 |

## 4.2 Final RMSE value

```
rmse
```

```
## [1] 0.8620975
```

---

# 5 Conclusion

Including the variables 'movie ID' and 'user ID' is key when optimizing movie rating predictions. Other variables such as 'date of rating', and regularization did not affect the predictions as much. Excluding some of the variables from the final validation model may have led to a similar final result.

---

# 6 References


https://rafalab.github.io/dsbook/
https://grouplens.org/datasets/movielens/10m/
http://files.grouplens.org/datasets/movielens/ml-10m.zip
https://files.grouplens.org/datasets/movielens/ml-10m-README.html


---

# 7 Appendix

## 7.1 Transforming RMarkdown to RScript

```
# Example code
# knitr::purl('grafj_movielens_r_markdown.Rmd',
# documentation = 2)
```

## 7.2 R Version

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## crt            ucrt
## system         x86_64, mingw32
## status
## major          4
## minor          2.1
## year           2022
## month          06
## day            23
## svn rev        82513
## language       R
## version.string R version 4.2.1 (2022-06-23 ucrt)
## nickname       Funny-Looking Kid
```