MovieLens Project

Jean Paul Chacón González

```r
##########################################################
# Create edx and final_holdout_test sets
##########################################################

# Note: this process could take a couple of minutes

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")

library(tidyverse)
library(caret)

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

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))
```

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

Introduction

Thanks to a 20-minute margin, a group called "The Ensemble" managed to earn 1 million dollars as the winning prize of a challenge created by the Netflix company. The challenge itself consisted in generating a movie recommendation system (model) that could beat their already-existing recommendation algorithm. The newly-proposed model needed to beat the aforementioned Netflix model by being 10% better in order to be eligible for the prize (Whiting (2009)).

This sets the ground as the motivation for this project, whose goal is to generate a movie recommendation model that beats the 0.8649 RMSE mark for the 10M MovieLens dataset. Now, how is this **mark** beat? What even is **RMSE**?

RMSE is a function that measures how well a statistical model predicts values. It does this by comparing the observed values (this is how the actual real-world values are called in the Machine Learning context) to the predicted values (values that our model said that will be observed). This means that we "improve" the model by changing it so that it yields a lower RMSE value. Because of its nature, this model manages to penalize the "bigger errors" (extraordinarily bad predictions) more than its MAE counterpart. The resulting output value of the RMSE function is in the same units as the output variable (Frost (2024)).

The MovieLens dataset is comprised of 10 million data points, with each data point symbolizing each rating given by a specific user to a specific movie.

In the next section (Analysis) this dataset will be explored more in-depth in order to learn about the different columns and data that it contains.

Analysis

Loss Function

In order to use an already built-in RMSE (loss) function, I'll use the Metrics' `rmse(actual, predicted)` function. For that, I need to import the library:

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

Data Exploration

In order to generate the models, I need to learn about how the dataset looks and which are the possible features for my model.

Glancing at the structure of the dataset:

```r
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

The following information from this 6-column and 9000055-observation `edx` dataset can be extracted:

Column Name	Data Type
`userId`	int
`movieId`	int
`rating`	num
`timestamp`	int
`title`	chr
`genres`	chr

Now taking a look at the first 6 rows of the dataset:

```r
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

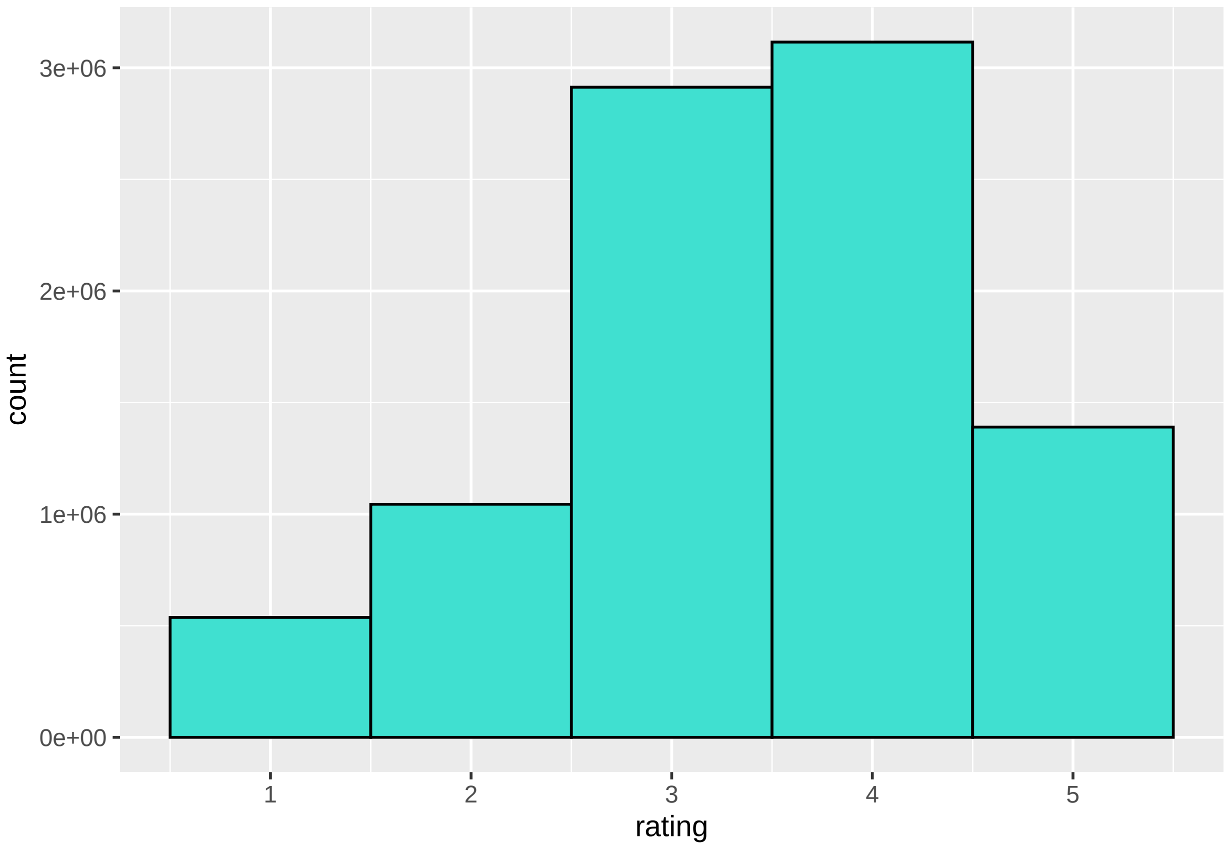
The dataset seems to contain the basic information about each movie's rating like the rating itself, the timestamp corresponding to when the movie was rated. the movie's name and genre. The genre itself could be an impactful predictor variable, but further analysis is required. An interesting observation is that the year of creation of the movie is embedded in the movie's title between parenthesis.

Rating Variable

The most important variable in the dataset is the `rating` column, since it will be my model's output variable.

First off, I will look at how this variable is distributed along the dataset:

```r
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
library(ggplot2)

ggplot(edx, aes(rating)) +
  geom_histogram(fill = "turquoise", color = "black", binwidth = 1)
```



Only the 3's and 4's ratings seem to surpass the 2 million rating count, with the other ratings having significantly lower counts. A rating of 4 is the most commonly-given rating to a movie.

This means that, according to the gathered dataset information, bad and extremely good movies (low and above 4 ratings respectively) are either not rated or they are mostly not present in the dataset. This could even be due to high-expectation viewers that end up being very disappointed after seeing a movie. On the other hand, the lower count of 5-rated movies could be justified by making the statement that "perfect" movies are hard to come across.

Average Rating In order to know what rating to classify as *average*, I need to calculate the dataset's average rating:

```r
mean(edx$rating)
```

```
## [1] 3.512465
```

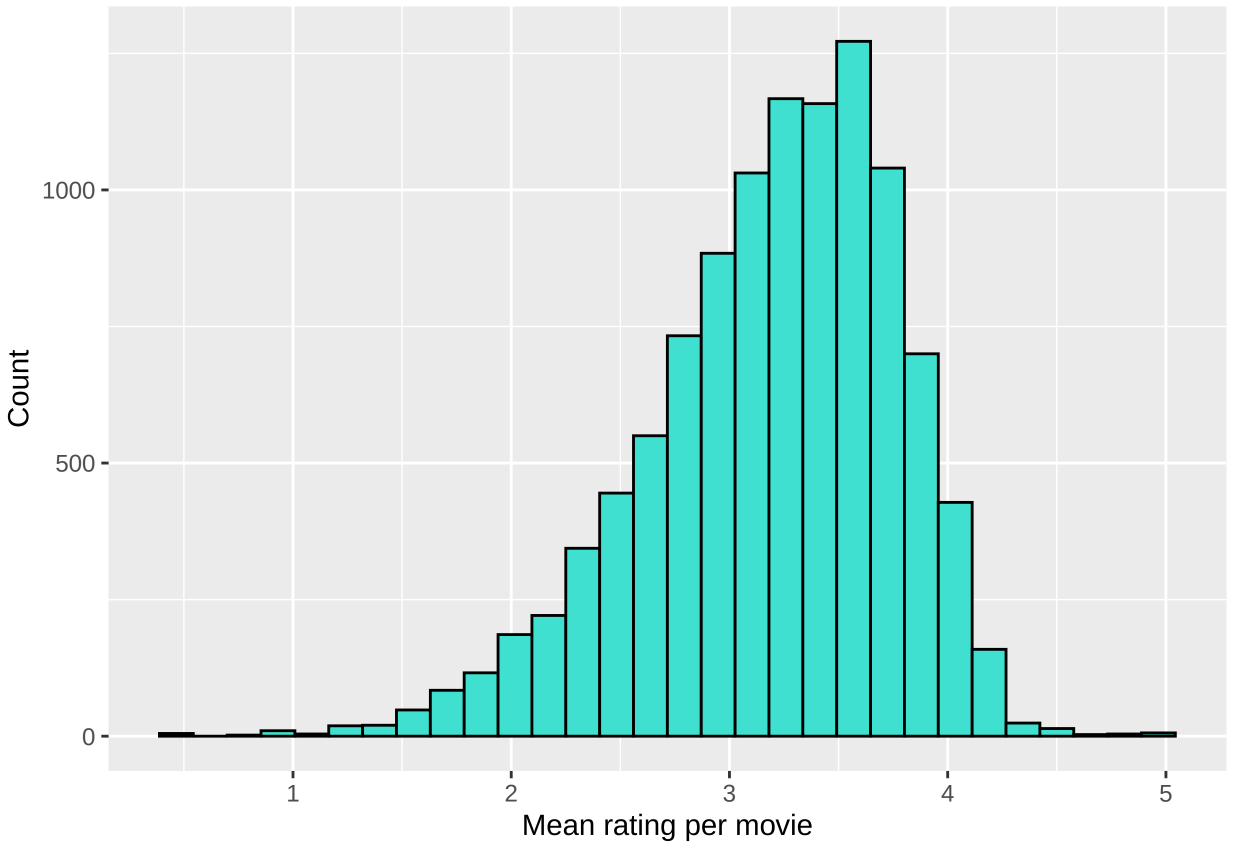
As per the output, all of the movies with ratings around the 3.5 mark can be considered as average movies.

Movie Identification Variable

In the context of movie predictions, its important to account for the fact that some movies have naturally higher quality than others (higher production, better cast, better story-line and more). This crucial factor needs to be present in the model.

As hinted before, the expected behavior of the dataset is to contain mostly-average rated movies, with the extremes (bad and incredible) having lower counts.

```
edx %>%
  group_by(movieId) %>%
  summarize(mean_rating_per_movie = mean(rating)) %>%
  ggplot(aes(mean_rating_per_movie)) +
  geom_histogram(fill = "turquoise", color = "black") +
  labs(x = "Mean rating per movie", y = "Count")
```



The expected behavior seems to comply with the actual behavior of the dataset, with it having higher counts around the mean rating (3.5) and lower counts around it. This generates a bell-like shaped curve (normal distribution curve).

The lowest-rated movies are:

```
edx %>%
  group_by(movieId) %>%
  summarize(
    movie = unique(title),
    mean_rating_per_movie = mean(rating),
    genre = unique(genres),
    rating_count = n()
  ) %>%
  arrange(mean_rating_per_movie) %>%
  head()
```

```
## # A tibble: 6 x 5
##   movieId movie                         mean_rating_per_movie genre rating_count
##     <int> <chr>                                         <dbl> <chr>        <int>
## 1    5805 Besotted (2001)                               0.5   Drama            2
## 2    8394 Hi-Line, The (1999)                           0.5   Drama            1
## 3   61768 Accused (Anklaget) (2005)                     0.5   Drama            1
## 4   63828 Confessions of a Superhero (~                 0.5   Docu~            1
## 5   64999 War of the Worlds 2: The Nex~                 0.5   Acti~            2
## 6    8859 SuperBabies: Baby Geniuses 2~                 0.795 Come~           56
```

As it can be seen through the output table, the lowest rated movies are barely rated, this introduces a huge bias because if a single user doesn't like the movie (like in Hi-Line, Accused and Confessions of a Superhero), then the movie is considered as being extremely bad, but in fact, the movie could be considered average or even good if rated by several people.

The aforementioned effect can be seen when the SuperBabies and the rest of the lowest-rated movies are compared. Since the SuperBabies movie is rated by more people, then it has a little higher mean rating.

Now looking at some of the highest-rated movies:

```
edx %>%
  group_by(movieId) %>%
  summarize(
    movie = unique(title),
    mean_rating_per_movie = mean(rating),
    genre = unique(genres),
    rating_count = n()
  ) %>%
  arrange(desc(mean_rating_per_movie)) %>%
  head()
```

```
## # A tibble: 6 x 5
##   movieId movie                         mean_rating_per_movie genre rating_count
##     <int> <chr>                                         <dbl> <chr>        <int>
## 1    3226 Hellhounds on My Trail (1999)                     5 Docu~            1
## 2   33264 Satan's Tango (Sátántangó) (~                     5 Drama            2
## 3   42783 Shadows of Forgotten Ancesto~                     5 Dram~            1
## 4   51209 Fighting Elegy (Kenka erejii~                     5 Acti~            1
## 5   53355 Sun Alley (Sonnenallee) (199~                     5 Come~            1
## 6   64275 Blue Light, The (Das Blaue L~                     5 Dram~            1
```

In this case, also few people rated the other end of the rating spectrum. Here a similar case happens as the one above, meaning that a single person thought that the Hellhounds movie was perfect, but if more people are set to be involved in the rating of this movie, it is almost certain that the mean rating of this movie would drop, since different people perceive a movie in a different way.

Rating Count

After mentioning the rating counts in the previous section, their possible impact on movie ratings will be studied in depth.

Starting with movies that weren't rated much:

Low rating counts Showing the movies with the lowest rating counts:

```
edx %>%
  group_by(movieId) %>%
  summarize(
    movie = unique(title),
    mean_rating_per_movie = mean(rating),
    genre = unique(genres),
    rating_count = n()
  ) %>%
  arrange(rating_count) %>%
  head(n=20)
```

```
## # A tibble: 20 x 5
##    movieId movie                         mean_rating_per_movie genre rating_count
##      <int> <chr>                                         <dbl> <chr>        <int>
##  1    3191 Quarry, The (1998)                              3.5 Drama            1
##  2    3226 Hellhounds on My Trail (199~                    5   Docu~            1
##  3    3234 Train Ride to Hollywood (19~                    3   Come~            1
##  4    3356 Condo Painting (2000)                           3   Docu~            1
##  5    3383 Big Fella (1937)                                3   Dram~            1
##  6    3561 Stacy's Knights (1982)                          1   Drama            1
##  7    3583 Black Tights (1-2-3-4 ou Le~                    3   Dram~            1
##  8    4071 Dog Run (1996)                                  1   Drama            1
##  9    4075 Monkey's Tale, A (Les Châte~                    1   Anim~            1
## 10    4820 Won't Anybody Listen? (2000)                    2   Docu~            1
## 11    5257 In the Winter Dark (1998)                       3.5 Drama            1
## 12    5565 Dogwalker, The (2002)                           2   Drama            1
## 13    5616 Mesmerist, The (2002)                           3.5 Come~            1
## 14    5676 Young Unknowns, The (2000)                      2.5 Drama            1
## 15    5702 When Time Ran Out... (a.k.a~                    1   Acti~            1
## 16    6085 Neil Young: Human Highway (~                    1.5 Come~            1
## 17    6189 Dischord (2001)                                 1   Dram~            1
## 18    6501 Strange Planet (1999)                           2   Come~            1
## 19    6758 Emerald Cowboy (2002)                           3   Docu~            1
## 20    6838 Once in the Life (2000)                         3   Crim~            1
```

As shown in this output table, movies with low rating_counts tend to have a low rating as well.

```
edx %>%
  group_by(movieId) %>%
  summarize(
    movie = unique(title),
    mean_rating_per_movie = mean(rating),
    genre = unique(genres),
    rating_count = n()
  ) %>%
  arrange(rating_count) %>%
  head(n=20) %>%
  summarize(mean = mean(mean_rating_per_movie))
```

```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  2.42
```

The mean rating of these low-count rated movies is 2.425, a very low value.

High rating counts Now, focusing on movies with the highest rating counts:

```
edx %>%
  group_by(movieId) %>%
  summarize(
    movie = unique(title),
    mean_rating_per_movie = mean(rating),
    genre = unique(genres),
    rating_count = n()
  ) %>%
  arrange(desc(rating_count)) %>%
  head(n=20)
```

```
## # A tibble: 20 x 5
##    movieId movie                        mean_rating_per_movie genre rating_count
##      <int> <chr>                                        <dbl> <chr>        <int>
##  1     296 Pulp Fiction (1994)                           4.15 Come~        31362
##  2     356 Forrest Gump (1994)                           4.01 Come~        31079
##  3     593 Silence of the Lambs, The (~                  4.20 Crim~        30382
##  4     480 Jurassic Park (1993)                          3.66 Acti~        29360
##  5     318 Shawshank Redemption, The (~                  4.46 Drama        28015
##  6     110 Braveheart (1995)                             4.08 Acti~        26212
##  7     457 Fugitive, The (1993)                          4.01 Thri~        25998
##  8     589 Terminator 2: Judgment Day ~                  3.93 Acti~        25984
##  9     260 Star Wars: Episode IV - A N~                  4.22 Acti~        25672
## 10     150 Apollo 13 (1995)                              3.89 Adve~        24284
## 11     592 Batman (1989)                                 3.39 Acti~        24277
## 12       1 Toy Story (1995)                              3.93 Adve~        23790
## 13     780 Independence Day (a.k.a. ID~                  3.38 Acti~        23449
## 14     590 Dances with Wolves (1990)                     3.74 Adve~        23367
## 15     527 Schindler's List (1993)                       4.36 Dram~        23193
## 16     380 True Lies (1994)                              3.50 Acti~        22823
## 17    1210 Star Wars: Episode VI - Ret~                  4.00 Acti~        22584
## 18      32 12 Monkeys (Twelve Monkeys)~                  3.87 Sci-~        21891
## 19      50 Usual Suspects, The (1995)                    4.37 Crim~        21648
## 20     608 Fargo (1996)                                  4.13 Come~        21395
```

It can be easily seen that these movies with higher rating counts have ratings closer to 5 compared to the ones with the lowest rating counts.

```
edx %>%
  group_by(movieId) %>%
  summarize(
    movie = unique(title),
    mean_rating_per_movie = mean(rating),
    genre = unique(genres),
    rating_count = n()
  ) %>%
  arrange(desc(rating_count)) %>%
  head(n=20) %>%
  summarize(mean = mean(mean_rating_per_movie))
```

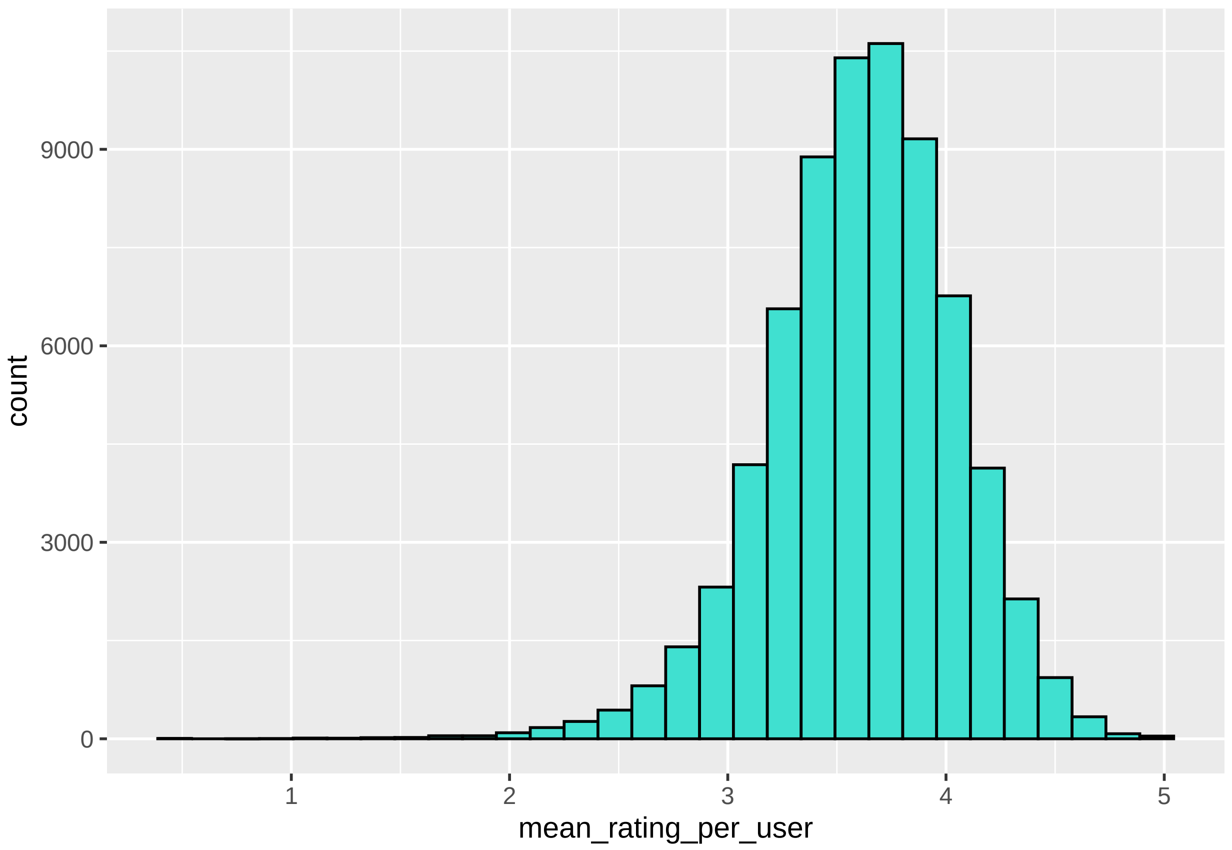
```
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  3.96
```

The mean of this higher-count rated movies is much higher (3.964) than the mean of the lowest-count rated movies (2.425), symbolizing a definitive effect that needs to be solved through regularization.

User Identification Variable

An identification of each user (`userId`) is given for every entry (observation/rating) in the dataset. In real life, some users (movie-watchers) are more inclined to like more some certain types of actors or movie genres. There's also some types of users that start to watch a movie with such high expectations that they leave a bad review of the movie, even if the movie isn't that bad, after they watch it.

```
edx %>%
  group_by(userId) %>%
  summarize(mean_rating_per_user = mean(rating)) %>%
  ggplot(aes(mean_rating_per_user)) +
  geom_histogram(fill = "turquoise", color = "black")
```



There doesn't seem to be high amounts of extreme-rating movie watchers.

Checking the amount of ratings that the users who gave low ratings made (this will help to see if these users just rated a bad movie or if in fact they are users who generally rate their movies low):

```
edx %>%
  group_by(userId) %>%
  summarize(mean_rating_per_user = mean(rating), rating_count = n()) %>%
  arrange(mean_rating_per_user) %>%
  head(n=10)
```

```
## # A tibble: 10 x 3
##    userId mean_rating_per_user rating_count
##     <int>                <dbl>        <int>
##  1  13496                0.5             17
##  2  48146                0.5             25
##  3  49862                0.5             17
##  4  62815                0.5             20
##  5  63381                0.5             18
##  6   6322                0.706           17
##  7   3457                1               19
##  8  24176                1              131
##  9  24490                1               17
## 10  28416                1.04            26
```

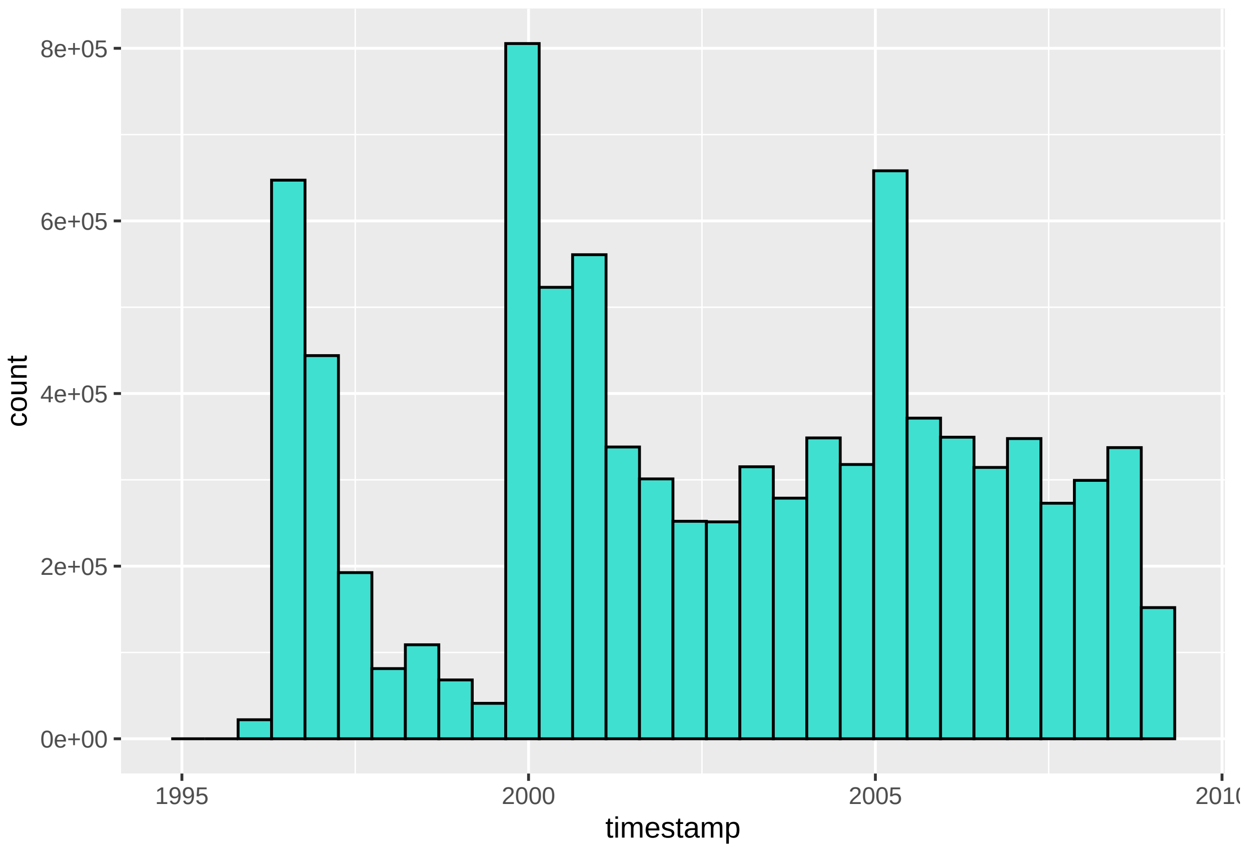
After thoughtful inspection of the above output, it can be said that the users who have a low average rating didn't necessarily rate a low amount of movies. So, this could be more due to the users having a bias towards giving low scores rather than the amount of ratings themselves that these specific users made. This will be accounted for in the model (user bias).

Timestamp Variable

The dataset comes with an additional variable called `timestamp` and it symbolizes the date when the actual rating was made (not the movie's release date). It is important to see if this would be a significant feature for the model.

Generating a graph of the amount of ratings made over time:

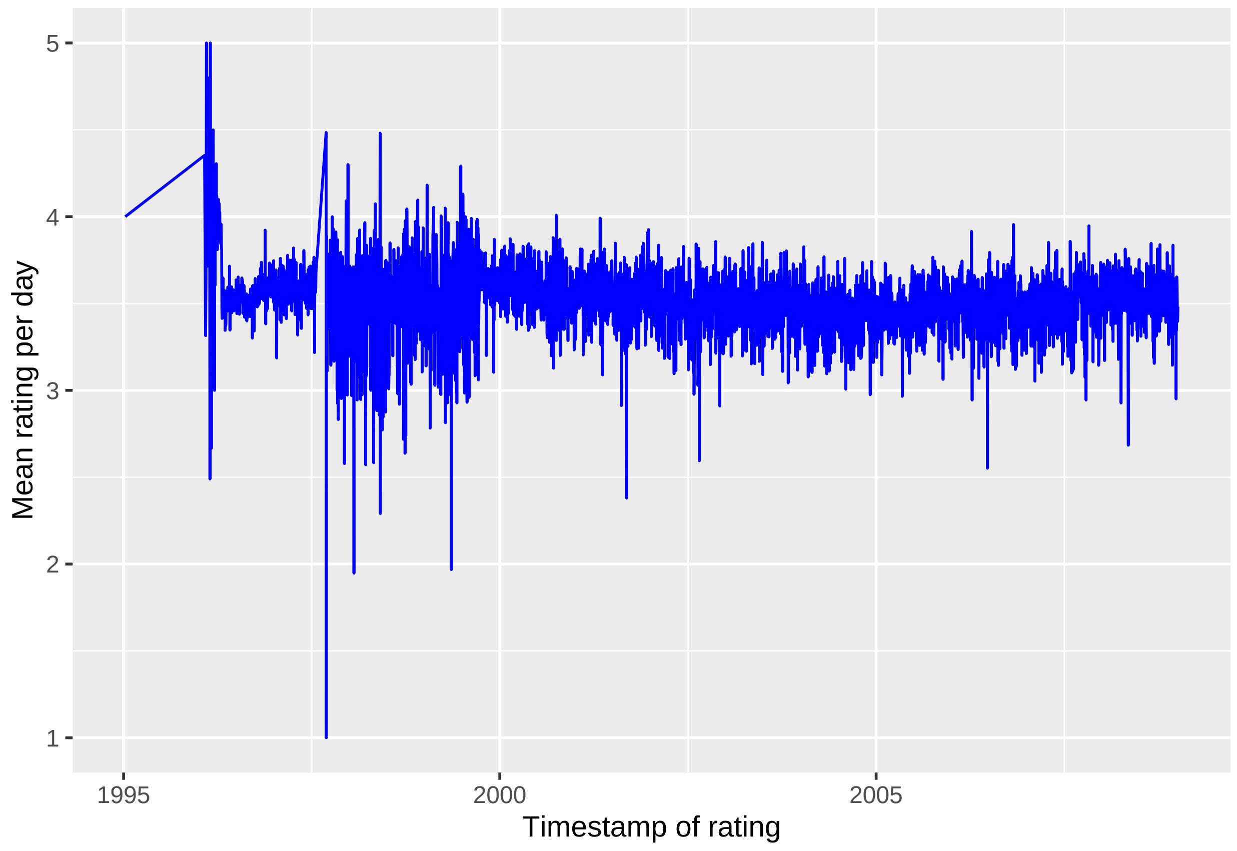
```r
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
library(lubridate)

edx %>%
  mutate(timestamp = as_datetime(timestamp)) %>%
  mutate(timestamp = round_date(timestamp, unit = "day")) %>%
  ggplot(aes(timestamp)) +
  geom_histogram(fill = "turquoise", color = "black")
```



There seems to be a strange pattern here, movies rated before the year 2000 seem to have a descendent pattern, while movies rated in 2000 and beyond seem to have a more stable (although with some unusual high counts) pattern.

Generating a graph to see if the timestamp itself has something to do with the score that movies receive:

```r
edx %>%
  mutate(timestamp = as_datetime(timestamp)) %>%
  mutate(timestamp = round_date(timestamp, unit = "day")) %>%
  group_by(timestamp) %>%
  summarize(mean_rating_per_week = mean(rating)) %>%
  ggplot(aes(timestamp, mean_rating_per_week)) +
  geom_line(color = "blue") +
  labs(x = "Timestamp of rating", y = "Mean rating per day")
```



```r
edx <- edx %>%
  mutate(timestamp = as_datetime(timestamp)) %>%
  mutate(timestamp = round_date(timestamp, unit = "day"))

final_holdout_test <- final_holdout_test %>%
  mutate(timestamp = as_datetime(timestamp)) %>%
  mutate(timestamp = round_date(timestamp, unit = "day"))
```

Although there isn't a clear pattern of the timestamp affecting the ratings themselves, there seems to be a great amount of outliers (very high scores being given) on the left side of the graph (earlier years).

Most values (ignoring the outliers) seem to be between the 2.75 and the 3.75 score.

Due to the timestamp affecting some movie ratings in some dates, this will be an added bias for the model.

Title Variable

The title variable is a character vector (made up of strings) that contain the actual names of the movies. From this, the year of publication can be extracted and it is enclosed by parenthesis.

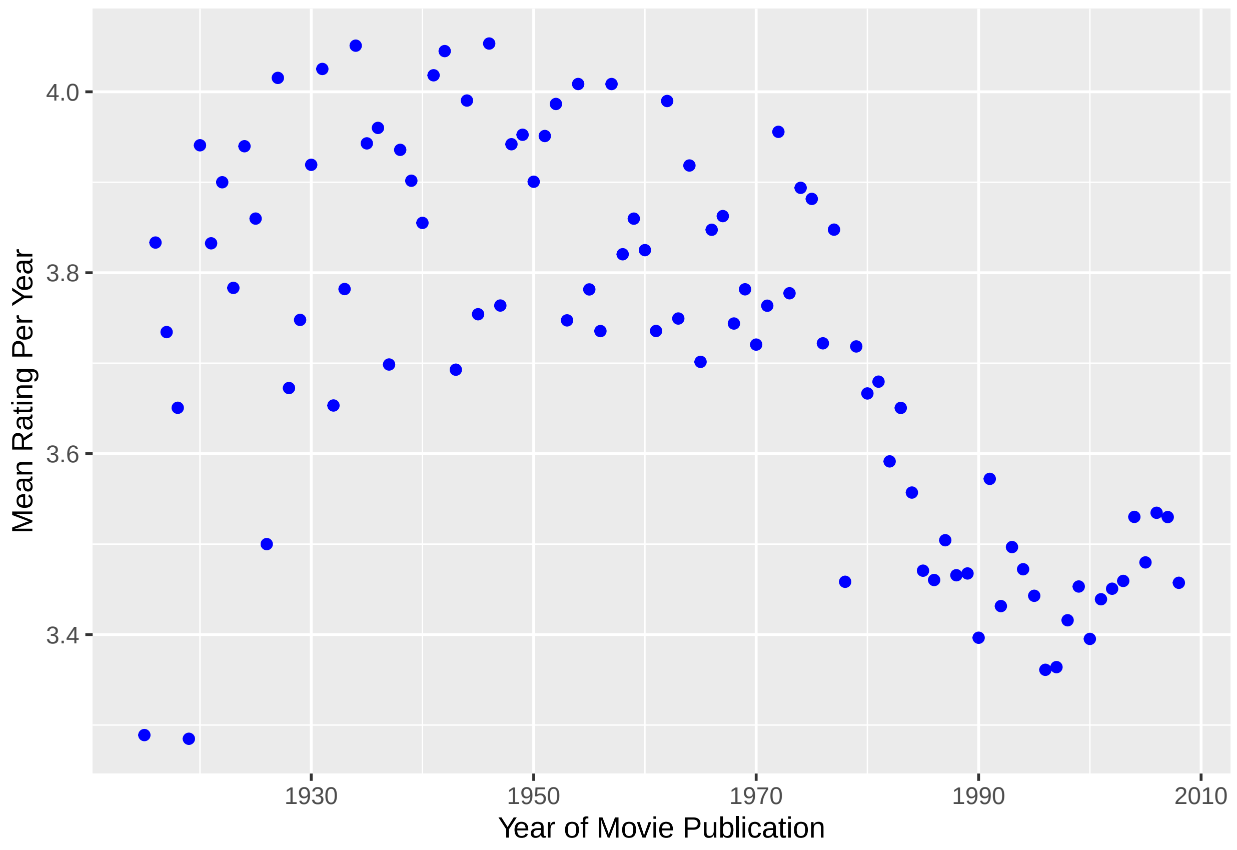
For this, regular expressions will be used to extract the 4 digits enclosed inside of the parenthesis:

```
edx <- edx %>%
  mutate(year_publication = str_extract(title, "\\(\\d{4}\\)") %>%
           str_remove_all("[\\(\\)]")) %>%
  mutate(year_publication = as.integer(year_publication))

final_holdout_test <- final_holdout_test %>%
  mutate(year_publication = str_extract(title, "\\(\\d{4}\\)") %>%
           str_remove_all("[\\(\\)]")) %>%
  mutate(year_publication = as.integer(year_publication))


head(edx)
```

```
##   userId movieId rating  timestamp                         title
## 1      1     122      5 1996-08-02              Boomerang (1992)
## 2      1     185      5 1996-08-02               Net, The (1995)
## 4      1     292      5 1996-08-02               Outbreak (1995)
## 5      1     316      5 1996-08-02               Stargate (1994)
## 6      1     329      5 1996-08-02 Star Trek: Generations (1994)
## 7      1     355      5 1996-08-02       Flintstones, The (1994)
##                          genres year_publication
## 1                Comedy|Romance             1992
## 2         Action|Crime|Thriller             1995
## 4  Action|Drama|Sci-Fi|Thriller             1995
## 5       Action|Adventure|Sci-Fi             1994
## 6 Action|Adventure|Drama|Sci-Fi             1994
## 7       Children|Comedy|Fantasy             1994
```

Year Variable Now, having generated a new variable from the title, it's pertinent to see if this new variable could be used as a feature for the model.

```
edx %>%
  group_by(year_publication) %>%
  summarize(mean_rating_per_year_publication = mean(rating)) %>%
  ggplot(aes(year_publication, mean_rating_per_year_publication)) +
  geom_point(color = "blue") +
  labs(x = "Year of Movie Publication", y = "Mean Rating Per Year")
```



There is a clear descendent pattern in the mean ratings per year as time went on, with earlier years having higher average ratings than more recent years, the downwards pattern starts around the year 1980, symbolizing that people are more critical of newer movies.

This is yet another clear effect that needs to be accounted for in the model (year of publication bias).

Genre Variable

The final possible variable is the `genre` variable, that describes whether the movie is an action movie, comedy, thriller movie, etc.

The amount of unique genre combinations needs to be calculated to see if it can be plotted:

```
n_distinct(edx$genres)
```

```
## [1] 797
```

There are 797 different (unique) genre combinations, this makes it impossible to plot each and every one in a boxplot. Instead I will grab 10 random genres and plot them against their ratings.

First, I will get the 10 random genres and I will put them in a list:

```
set.seed(2024) # for sample reproducibility

genres_to_show <- edx %>%
  pull(genres) %>%
  unique()
```

```
genres_to_show <- genres_to_show %>% sample(10)
```

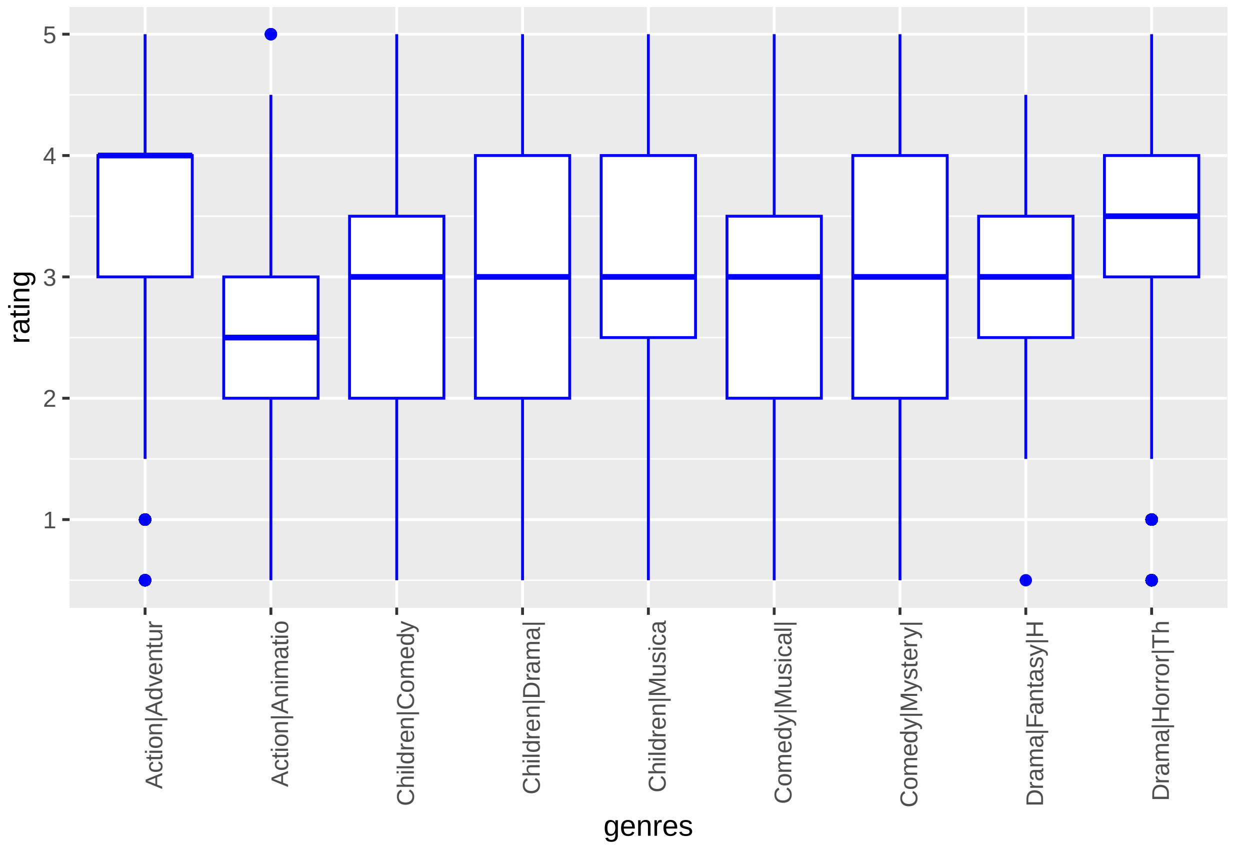
The randomly-selected genre combinations are:

```
genres_to_show
```

```
##  [1] "Drama|Fantasy|Horror|Sci-Fi"
##  [2] "Action|Animation|Fantasy"
##  [3] "Children|Comedy|Western"
##  [4] "Children|Musical"
##  [5] "Children|Drama|Fantasy"
##  [6] "Comedy|Mystery|Romance"
##  [7] "Action|Adventure|Comedy|Crime|Drama|Romance"
##  [8] "Comedy|Musical|Sci-Fi"
##  [9] "Action|Adventure|War|Western"
## [10] "Drama|Horror|Thriller"
```

Now, having a more manageable genre list, the plotting can proceed:

```
edx %>%
  filter(genres %in% genres_to_show) %>%
  mutate(genres = substr(genres, 1, 15)) %>%
  ggplot(aes(genres, rating)) +
  geom_boxplot(col = "blue") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



The genre text is too long to be shown correctly in the boxplot, so it's been truncated to the first 15 characters. For the 10 sampled genres, there seems to be a difference of rating values between the genres, since for example the "Action|Animatio" boxplot has lower overall values than the ones present in the "Action|Adventur" plot, symbolizing that genre effect (genre bias) needs to be accounted for in the model.

Data Partition

The code that obtains the dataset and divides it into a working set (`edx`) and a testing set (`test_holdout_set`) has already been run and it is at the top of the `.Rmd` code. However, since there's no correct way of adjusting the model's hyperparameters (these affect how machine learning models are configured) without executing it in the `test_holdout_set`, I will create a further separation of the `edx` set.

This separation results in the following datasets:

- `edx`
  - `train_data`
  - `test_data`
- `test_holdout_set`

```
set.seed(2024)

# 80% for testing, it stratifies by default
train_indices <- createDataPartition(edx$rating, p = 0.8, list = FALSE)

# Create the actual dataframes
train_data <- edx[train_indices, ]
test_data <- edx[-train_indices, ]

#### Exclude users and movies not present in training_data
test_data <- test_data %>%
     semi_join(train_data, by = "movieId") %>%
     semi_join(train_data, by = "userId")



# rm(edx) # removing the original dataset since it won't be used anymore.
```

Model Implementation

The dimensions of the `train_data` dataset are:

```
dim(train_data)
```

```
## [1] 7200045       7
```

7200045 rows and 7 columns. In view of the fact that the dimensions of the training dataset are so big, the usual `lm()` model utilization can't be used because it causes R to crash. Instead, a "manual" approach to the problem will be used.

Predicting with the average

In order to set a baseline model, the mean of the ratings themselves will be used as the output of the model for every movie (regardless of its characteristics and features).

So, the value to be predicted for each movie's rating is:

```
mu <- mean(train_data$rating)
mu
```

```
## [1] 3.512449
```

Initially, each movie will be predicted to have a value of 3.512449

Calculating the loss function for this simple model:

```
initial_rmse <- rmse(test_data$rating, mu)
initial_rmse
```

```
## [1] 1.060406
```

The RMSE loss has a value of 1.06, far from the objective (0.8649).

In order to keep track of the metrics along the project, a tibble will be created.

```
metric_results <- tibble(
  Model_Name = c("Mean"),
  RMSE = c(1.060406)
)

metric_results
```

```
## # A tibble: 1 x 2
##   Model_Name  RMSE
##   <chr>      <dbl>
## 1 Mean        1.06
```

Adding movie bias

It is generally known that independently of the genre of a movie, there are movies that are significantly better than others, so accounting for this will be introduced to improve the model:

```
per_movie_bias <- train_data %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

head(per_movie_bias)
```

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <int>  <dbl>
## 1       1  0.413
## 2       2 -0.307
## 3       3 -0.362
## 4       4 -0.659
## 5       5 -0.449
## 6       6  0.298
```

Now that each movie has its bias (basically a measure of how good the movie is), these biases need to be assigned to the testing set to predict.

```
test_data_with_movie_bias <- test_data %>%
  left_join(per_movie_bias, by = "movieId")

head(test_data_with_movie_bias)
```

```
##   userId movieId rating  timestamp                               title
## 1      1     364      5 1996-08-02                Lion King, The (1994)
## 2      1     588      5 1996-08-02                       Aladdin (1992)
## 3      1     589      5 1996-08-02    Terminator 2: Judgment Day (1991)
## 4      2     648      2 1997-07-07           Mission: Impossible (1996)
## 5      2     780      3 1997-07-07 Independence Day (a.k.a. ID4) (1996)
## 6      2     802      2 1997-07-07                    Phenomenon (1996)
##                                         genres year_publication         b_i
## 1  Adventure|Animation|Children|Drama|Musical             1994  0.244079654
## 2 Adventure|Animation|Children|Comedy|Musical             1992  0.152073202
## 3                               Action|Sci-Fi             1991  0.416297319
## 4           Action|Adventure|Mystery|Thriller             1996 -0.123468376
## 5                 Action|Adventure|Sci-Fi|War             1996 -0.132201055
## 6                               Drama|Romance             1996  0.005665583
```

Having the biases with the test movies, a prediction can be made by applying a sum of the mean (mu) to the movie's specific bias (b_i, with i representing each movie index):

```
movie_bias_predictions <- mu + test_data_with_movie_bias$b_i
```

Calculating this model's loss:

```
movie_bias_rmse <- rmse(test_data$rating, movie_bias_predictions)
movie_bias_rmse
```

```
## [1] 0.9440403
```

So, accounting for the fact that some movies are better-regarded than others, the model was improved to an RMSE of 0.944.

Adding this metric to the result tibble:

```
metric_results <- add_row(
  metric_results,
    Model_Name = "Mean + Movie Bias",
    RMSE = 0.9440403
  )

head(metric_results)
```

```
## # A tibble: 2 x 2
##   Model_Name          RMSE
##   <chr>              <dbl>
## 1 Mean               1.06
## 2 Mean + Movie Bias 0.944
```

Adding user bias

As mentioned in the Analysis section, there exist users that generally report low ratings for the movies that they watch (when the data is grouped by user, the user's average ratings are low). Therefore this also needs to be accounted for in the model.

Calculating user bias:

```
per_user_bias <- train_data %>%
  left_join(per_movie_bias, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

head(per_user_bias)
```

```
## # A tibble: 6 x 2
##   userId      b_u
##    <int>    <dbl>
## 1      1  1.77
## 2      2 -0.00577
## 3      3  0.296
## 4      4  0.625
## 5      5  0.0923
## 6      6  0.237
```

With u being each unique user. Now these values need to be assigned with the actual users present in the test set in order to predict:

```
user_movie_bias_predictions <- test_data %>%
  left_join(per_movie_bias, by = "movieId") %>% # adding movie biases
  left_join(per_user_bias, by = "userId") %>%
  mutate(prediction = mu + b_i + b_u) %>%
  pull(prediction)

head(user_movie_bias_predictions)
```

```
## [1] 5.521867 5.429860 5.694084 3.383212 3.374479 3.512346
```

With these predictions being made, the RMSE can now be calculated:

```
rmse(test_data$rating, user_movie_bias_predictions)
```

```
## [1] 0.8665758
```

The model improved to an RMSE of 0.8665758, which is close but not lower than 0.8649.

```r
metric_results <- add_row(
  metric_results,
    Model_Name = "Mean + (Movie + User) Bias",
    RMSE = 0.8665758
  )

head(metric_results)
```

```
## # A tibble: 3 x 2
##   Model_Name                  RMSE
##   <chr>                      <dbl>
## 1 Mean                       1.06
## 2 Mean + Movie Bias          0.944
## 3 Mean + (Movie + User) Bias 0.867
```

Adding Regularization

In the analysis section was described the effect that low rating count has on the movie's rating. In order to account for this effect, regularization will be introduced into the model. The formula of the regularization introduced is:

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

With lambda being the regularization hyperparameter. What this does is that it introduces a penalty depending on the amount of samples (n_i) present. If there are a lot of samples then λ is almost insignificant compared to n_i, but if the sample count is low, then λ is significant and adds a penalty.

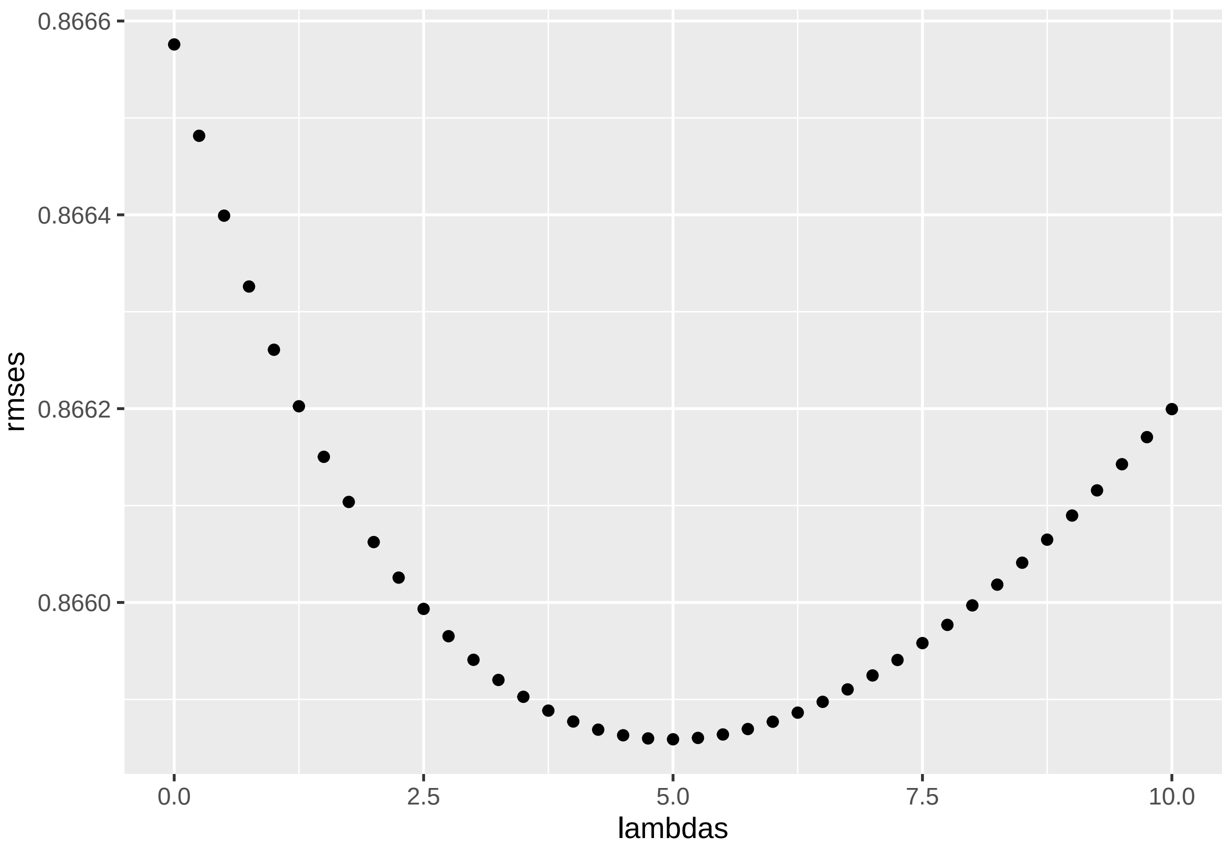
This means that a larger value of λ introduces more regularization that a low one.

This is a hyperparameter, meaning that it is a "setting" that can be changed to alter the model's behavior and performance.

There is no way to determine exactly which value of lambda will yield the best model, so I will try out different lambda values with my training and test sets (*this does not include the test_holdout_set*).

```r
#TODO: CHANGE THIS
lambdas <-seq(0, 10, 0.25)
rmses<-sapply(lambdas, function(lambd){

  mu<-mean(train_data$rating)

  b_i<-train_data %>%
    group_by(movieId) %>%
    summarize(b_i= sum(rating-mu)/(n()+lambd))

  b_u<-train_data %>%
    left_join(b_i,by="movieId") %>%
    group_by(userId)%>%
    summarize(b_u= sum(rating-b_i-mu)/(n()+lambd))

  predicted_ratings <- test_data %>%
    left_join(b_i,by= "movieId") %>%
    left_join(b_u,by= "userId") %>%
    mutate(pred= mu+ b_i+ b_u) %>%
    pull(pred)
  return(rmse(test_data$rating, predicted_ratings))
})
qplot(lambdas,rmses)
```

```
## Warning: 'qplot()' was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```



After thoroughly trying several λ hyperparameters on a range from 0 to 10, the best resulting λ value is:

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

```
## [1] 5
```

Additionally, the lowest RMSE can be found out from the RMSE list generated by the above function:

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

```
## [1] 0.8658588
```

The RMSE obtained with this regularization approach is better, indicating that it is beneficial to the model.

The lowest RMSE produced by this updated model is 0.8658588.

This new value will be added to the results tibble:

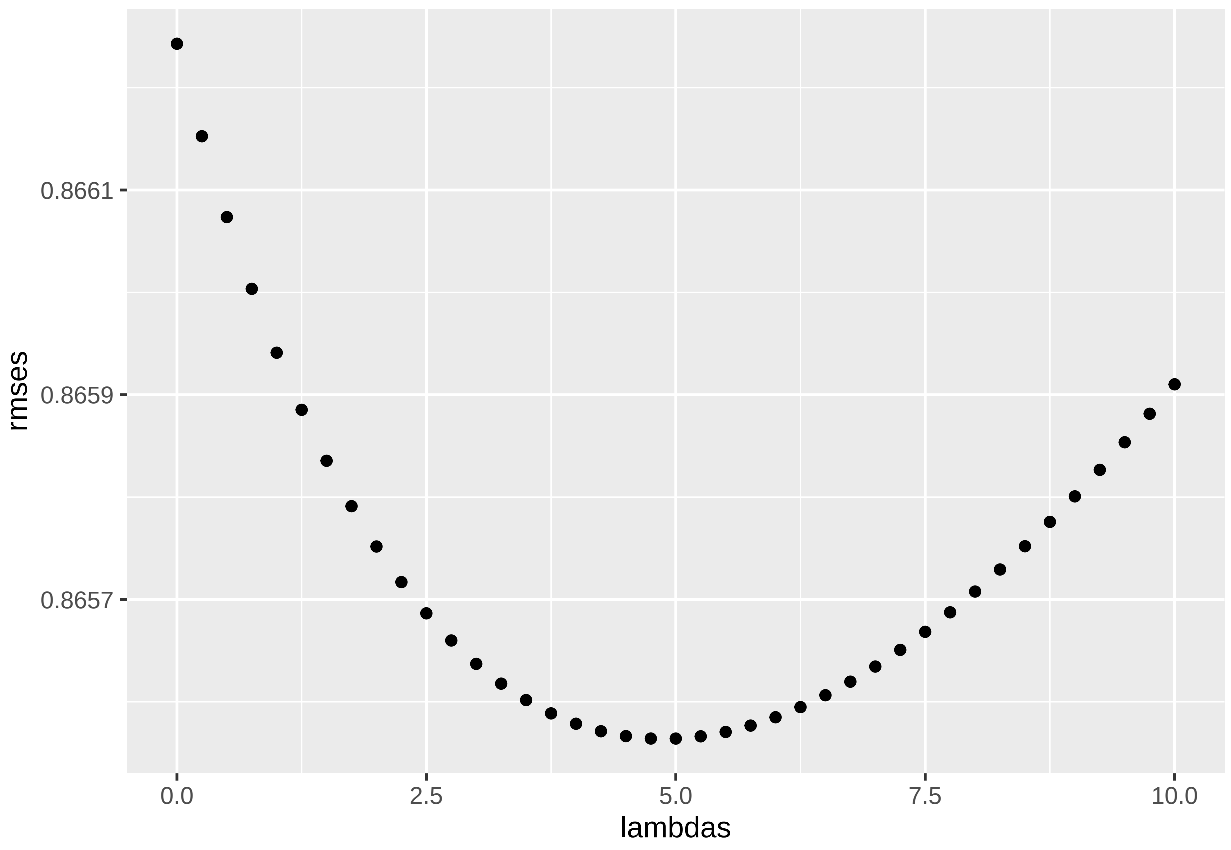
```r
metric_results <- add_row(
  metric_results,
    Model_Name = "Mean + (Movie + User) Bias + Regularization",
    RMSE = 0.8658588
  )

head(metric_results)
```

```
## # A tibble: 4 x 2
##   Model_Name                                   RMSE
##   <chr>                                       <dbl>
## 1 Mean                                        1.06
## 2 Mean + Movie Bias                           0.944
## 3 Mean + (Movie + User) Bias                  0.867
## 4 Mean + (Movie + User) Bias + Regularization 0.866
```

Adding genre bias

In the Analysis section I went over the `genre` variable and showed how the genre introduces a bias on the movie's rating (boxplot). That bias will be accounted for in this section.

```r
lambdas <-seq(0, 10, 0.25)
rmses<-sapply(lambdas, function(lambd){

  mu<-mean(train_data$rating)

  b_i<-train_data %>%
    group_by(movieId) %>%
    summarize(b_i= sum(rating-mu)/(n()+lambd))

  b_u<-train_data %>%
    left_join(b_i,by="movieId") %>%
    group_by(userId)%>%
    summarize(b_u= sum(rating-b_i-mu)/(n()+lambd))

  b_g <- train_data %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n() + lambd))

  predicted_ratings <- test_data %>%
    left_join(b_i,by= "movieId") %>%
    left_join(b_u,by= "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred= mu + b_i+ b_u + b_g) %>%
    pull(pred)
  return(rmse(test_data$rating, predicted_ratings))
})
qplot(lambdas,rmses)
```



Now, extracting the best-performing lambda:

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

```
## [1] 5
```

With this lambda (5), the RMSE obtained is:

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

```
## [1] 0.8655641
```

Once again, the model was improved (0.8655641) compared to the last one (0.8658588), but this is still far from the objective of being lower than 0.8649.

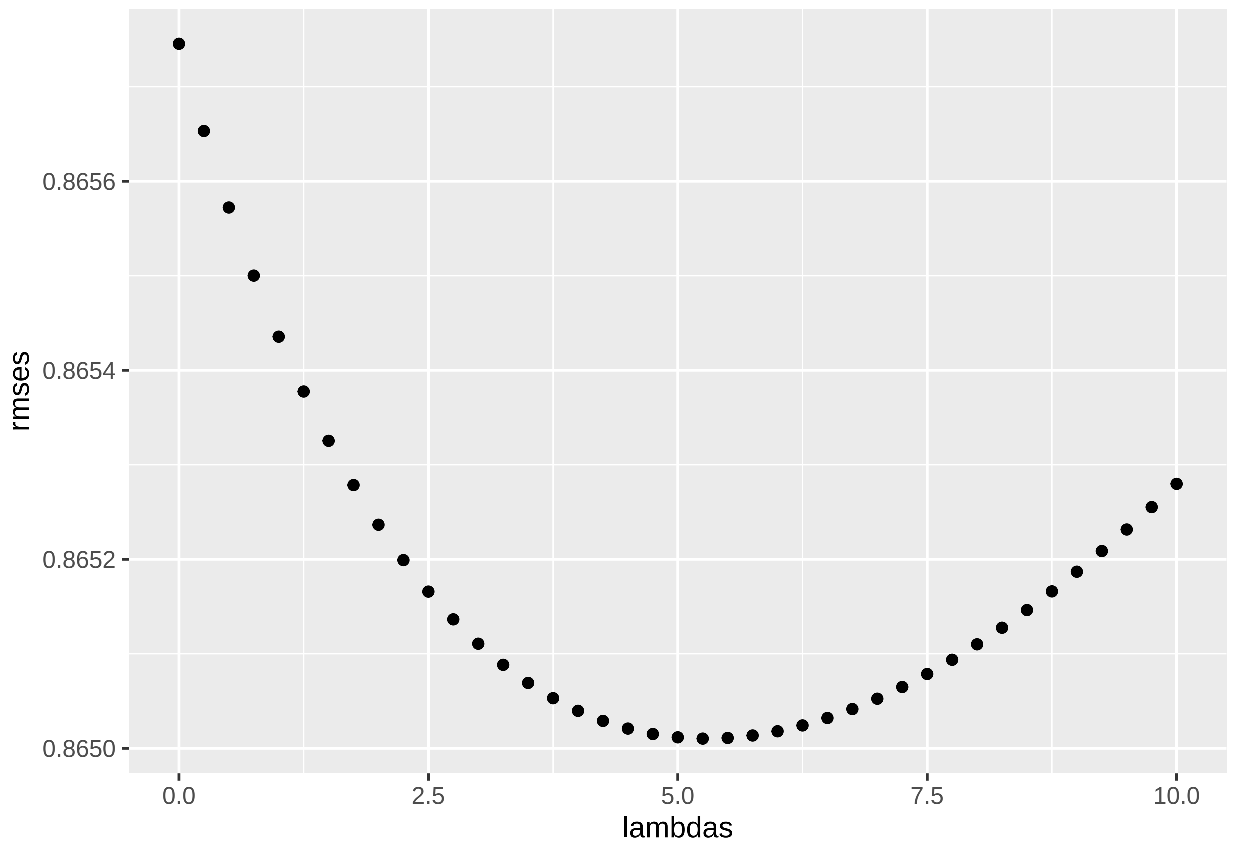
```
metric_results <- add_row(
  metric_results,
    Model_Name = "Mean + (Movie + User + Genre) Bias + Regularization",
    RMSE = 0.8655641
  )

head(metric_results)
```

```
## # A tibble: 5 x 2
##   Model_Name                                       RMSE
##   <chr>                                           <dbl>
## 1 Mean                                            1.06
## 2 Mean + Movie Bias                               0.944
## 3 Mean + (Movie + User) Bias                      0.867
## 4 Mean + (Movie + User) Bias + Regularization     0.866
## 5 Mean + (Movie + User + Genre) Bias + Regularization 0.866
```

Adding timestamp bias.

Thanks to the line plot and histogram made when analyzing the timestamp variable, it was concluded that the timestamp of each movie rating affects the rating value itself. Due to this, the timestamp effect will be introduced into the model.

```r
lambdas <-seq(0, 10, 0.25)
rmses<-sapply(lambdas, function(lambd){

  mu<-mean(train_data$rating)

  b_i<-train_data %>%
    group_by(movieId) %>%
    summarize(b_i= sum(rating-mu)/(n()+lambd))

  b_u<-train_data %>%
    left_join(b_i,by="movieId") %>%
    group_by(userId)%>%
    summarize(b_u= sum(rating-b_i-mu)/(n()+lambd))

  b_g <- train_data %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n() + lambd))

  b_d <- train_data %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(timestamp) %>%
    summarize(b_d = sum(rating - mu - b_i - b_u - b_g)/(n() + lambd))

  predicted_ratings <- test_data %>%
    left_join(b_i,by= "movieId") %>%
    left_join(b_u,by= "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_d, by="timestamp") %>%
    mutate(pred= mu + b_i+ b_u + b_g + b_d) %>%
    pull(pred)
  return(rmse(test_data$rating, predicted_ratings))
})
qplot(lambdas,rmses)
```



Extracting the best lambda obtained:

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

```
## [1] 5.25
```

With a lambda of 5.25, the lowest RMSE is:

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

```
## [1] 0.8650102
```

As expected, the performance of the model improved to a 0.8650102 RMSE, though more work needs to be done to reach the 0.8649 mark.

Again, this will be added to the result metrics tibble:

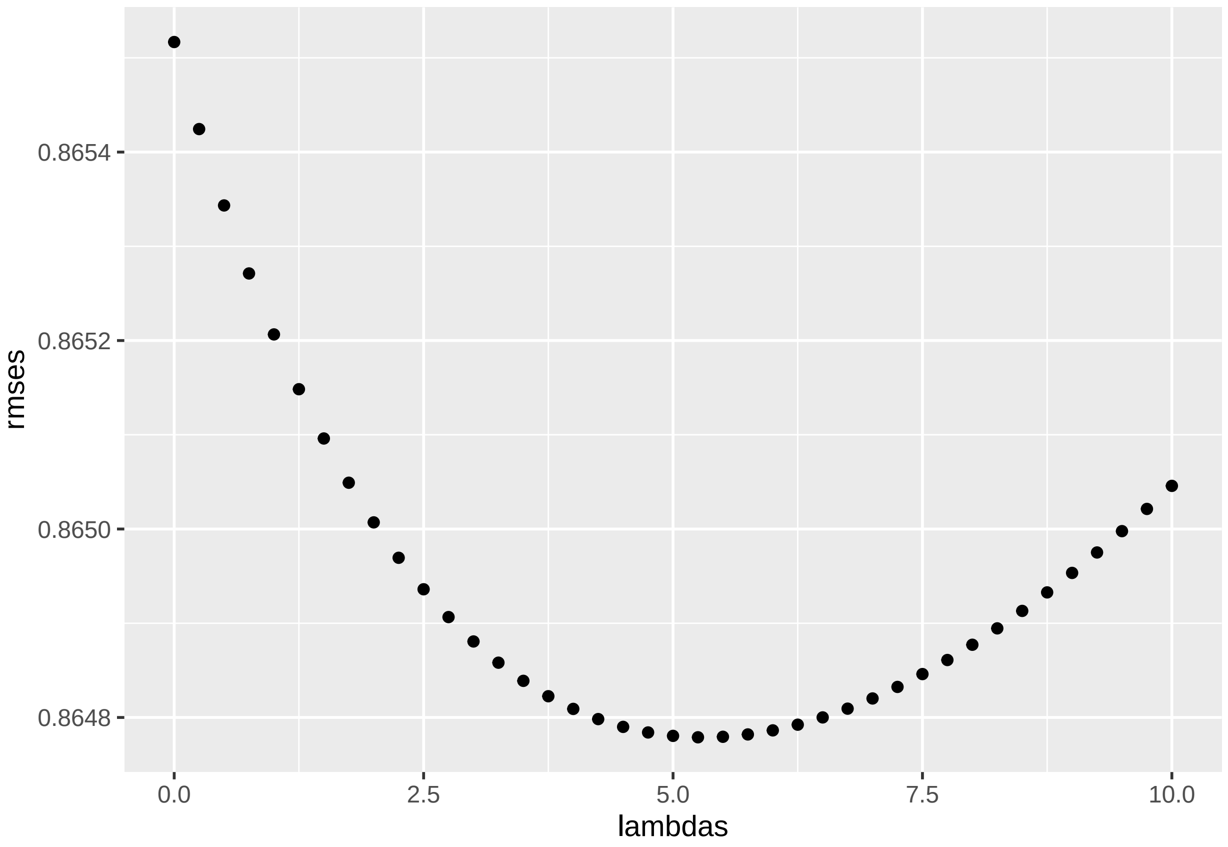
```
metric_results <- add_row(
  metric_results,
    Model_Name = "Mean + (Movie + User + Genre + Timestamp) Bias + Regularization",
    RMSE = 0.8650102
  )

head(metric_results)
```

```
## # A tibble: 6 x 2
##   Model_Name                                                RMSE
##   <chr>                                                    <dbl>
## 1 Mean                                                     1.06
## 2 Mean + Movie Bias                                        0.944
## 3 Mean + (Movie + User) Bias                               0.867
## 4 Mean + (Movie + User) Bias + Regularization              0.866
## 5 Mean + (Movie + User + Genre) Bias + Regularization      0.866
## 6 Mean + (Movie + User + Genre + Timestamp) Bias + Regularization 0.865
```

Adding year of publication bias

During the Analysis, it was discovered, thanks to the scatterplot generated, that the year of the movie's publication had a significant effect over the rating that it is given. In this section, this effect will be accounted for.

```r
#TODO: CHANGE THIS
lambdas <-seq(0, 10, 0.25)
rmses<-sapply(lambdas, function(lambd){

  mu<-mean(train_data$rating)

  b_i<-train_data %>%
    group_by(movieId) %>%
    summarize(b_i= sum(rating-mu)/(n()+lambd))

  b_u<-train_data %>%
    left_join(b_i,by="movieId") %>%
    group_by(userId)%>%
    summarize(b_u= sum(rating-b_i-mu)/(n()+lambd))

  b_g <- train_data %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n() + lambd))

  b_d <- train_data %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(timestamp) %>%
    summarize(b_d = sum(rating - mu - b_i - b_u - b_g)/(n() + lambd))

  b_yp <- train_data %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_d, by="timestamp") %>%
    group_by(year_publication) %>%
    summarize(b_yp = sum(rating - mu - b_i - b_u - b_g - b_d)/(n() + lambd))

  predicted_ratings <- test_data %>%
    left_join(b_i,by= "movieId") %>%
    left_join(b_u,by= "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_d, by="timestamp") %>%
    left_join(b_yp, by="year_publication") %>%
    mutate(pred= mu + b_i+ b_u + b_g + b_d + b_yp) %>%
    pull(pred)
  return(rmse(test_data$rating, predicted_ratings))
})
qplot(lambdas,rmses)
```



The lambda that yields the lowest RMSE seems to be around the 5 value.

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

```
## [1] 5.25
```

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

```
## [1] 0.864779
```

With a lambda of 5.25, yielding an RMSE of 0.864779, this is the best performing model that surpasses the objective of having an RMSE lower than 0.8649.

Finally, adding this model's RMSE to the tibble:

```
metric_results <- add_row(
  metric_results,
    Model_Name = "Mean + (Movie + User + Genre + Timestamp + Year) Bias + Regularization",
    RMSE = 0.864779
  )

head(metric_results, n=7)
```

```
## # A tibble: 7 x 2
##   Model_Name                                                              RMSE
##   <chr>                                                                  <dbl>
## 1 Mean                                                                   1.06
## 2 Mean + Movie Bias                                                      0.944
## 3 Mean + (Movie + User) Bias                                             0.867
## 4 Mean + (Movie + User) Bias + Regularization                            0.866
## 5 Mean + (Movie + User + Genre) Bias + Regularization                    0.866
## 6 Mean + (Movie + User + Genre + Timestamp) Bias + Regularization        0.865
## 7 Mean + (Movie + User + Genre + Timestamp + Year) Bias + Regularization 0.865
```

Applying the model to the final_holdout_test

Now that the best hyperparameter of lambda was found for the model, this model can now be applied to the `final_holdout_test` set in order to see how well the model performs in the real-world (with unseen data):

```
mu<-mean(edx$rating)

b_i<-edx %>%
  group_by(movieId) %>%
  summarize(b_i= sum(rating-mu)/(n()+lambda))

b_u<-edx %>%
  left_join(b_i,by="movieId") %>%
  group_by(userId)%>%
  summarize(b_u= sum(rating-b_i-mu)/(n()+lambda))

b_g <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n() + lambda))

b_d <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(timestamp) %>%
  summarize(b_d = sum(rating - mu - b_i - b_u - b_g)/(n() + lambda))

b_yp <- edx %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by="timestamp") %>%
  group_by(year_publication) %>%
  summarize(b_yp = sum(rating - mu - b_i - b_u - b_g - b_d)/(n() + lambda))

predicted_ratings <- final_holdout_test %>%
  left_join(b_i,by= "movieId") %>%
  left_join(b_u,by= "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by="timestamp") %>%
  left_join(b_yp, by="year_publication") %>%
  mutate(pred= mu + b_i+ b_u + b_g + b_d + b_yp) %>%
  pull(pred)
rmse(final_holdout_test$rating, predicted_ratings)
```

```
## [1] 0.863641
```

Notice that when assigning `predicted_ratings`, now the dataset used before the pipe symbol is the `final_holdout_test` and not the `test_data` set.

So, by using this fine-tuned model to predict movie ratings, the RMSE obtained is 0.863641, a value lower than the benchmark-to-beat 0.8649.

Results

Through the application of regularization to the movie, user, movie genre, time (day) of rating and the year of publication biases added up to the mean of all of the movie's ratings, the model achieved an RMSE of *0.863641*. This model's RMSE beats the benchmark value of 0.8649 that needed to be beaten.

The reasoning behind the good performance of the model can be explained through the findings and examinations made in the Analysis section. In which, I discussed (and showed through graphs) why some variables were good candidate features for the model that was going to be created early on. For example, the rating count introduced a big bias on whether a movie could be said to be good or not (if only one person rated a movie, then a general consensus couldn't be obtained for it, therefore introducing that aforementioned bias) that was solved through the usage of regularization.

Conclusion

Throughout this study, I analyzed each variable (column) that was present in the original `edx` dataframe and discovered the possibility of extracting a new variable (year) from an already existing one (title). I also analyzed different patterns that are present in the given data and explained how they could be of possible great use as features for the model later on.

Then, in the Model Implementation section, the model was built from the ground up, making improvements by adding features and observing the behavior of the model as it began to be made more complex in order to achieve a good performance. This section was finalized through the application of the fine-tuned model for predicting ratings in the `test_holdout_set` dataset, which wasn't utilized until that point.

Limitations and Future Work

Although this model performs well will the present data in this study, I think that this model could be further improved if more knowledge could be present about each movie. For example, a variable like the amount of actors or the actors' name themselves could be added and studied in order to see if the model reduces its RMSE value.

Another important aspect that could be an improving feature is the addition of the charge that the users had to pay to watch a movie (there are some users who don't think that spending a lot of money in order to watch a movie is worth it) money-wise has an effect on the ratings themselves over that specific movie.

Finally, another possible feature could be usage of the budget that each movie had to be made (usually higher-budget movies perform better than lower-budget ones).

References

Frost, Jim. 2024. "Root Mean Square Error (RMSE)." *Statistics By Jim.* https://statisticsbyjim.com/regression/root-mean-square-error-rmse/.

Whiting, Rick. 2009. "Researchers Solve Netflix Challenge, Win $1 Million Prize." *CRN*. https://www.crn.com/news/applications-os/220100498/researchers-solve-netflix-challenge-win-1-million-prize.