MovieLens Project

Fernando Jose Velasco Borea

April 24th 2019

Contents

1 Introduction and Overview

The following project has the objective to train a supervised machine learning movie recomendation algorithm based on the data set provided by HarvardX through the Data Science: Capstone course. The main goal is to train an algorithm that is able to yield an $RMSE < 0.87750$.

The general approach that will be taken is basically, first to analyze the edx and validation data set and its structure, then to start gathering useful insights through statistics and data visualization techniques that helps define a suitable course of action in the algorithm development process.

This project will be divided in the following steps:

1. Data Adquisition: The project will start by downloading the data set using the script provided by HarvardX.
2. Data Exploration: Once all the data is obtained, we will proceed with the exploratory data analysis.
3. Modeling: With the insights we gain on step 2, we will start to make the recommendation system model.
4. Model Testing (Results): Once we define our final model using our training data set, we will run it on our test data set.

Once we acomplish the steps previously described we will elaborate a "Conclusion" section with all the findings obtained during the exploratory data analysis and the final model implemented on the project as well as the results obtained with the model used.

This project will have a PDF and a Rmd version of this report as well as a script with all the codes used to make the project (with the exception of the R Markdown set-up lines included by default when opening a new file). The script will have detailed comments about each section to enhance readability and interpretation of the approaches used. The comments will include sectioned descriptions and line descriptions when needed.

1.1 Side Notes

To enhance code readability when viewing the Rmd version of this report and/or when viewing the MovieLens Script file to see just the coding part of the project, you can *fold* the all the sections from RStudio to then just *unfold* the section you are currently viewing, therefore, easing the interpretation of the code.

You can quickly do this from R studio going to *Edit > Folding > Collapse All* or simply with the shortcut *ALT + O* on windows. If you want to exapnd all the sections again, you can use the shortcut *ALT + SHIFT + O* on windows or from *Edit > Folding > Expand All*.

The code contained in this report can be found on the MovieLens Script file, as it follows the same structure and order as the report, therefore, making it easier to reproduce the results while maintaining code readability.

To render the Rmd version of this report you will need to have a LaTeX installation. If you don't have it, you can find more details on how to install it here

2 Data Adquisition

This step will be based almost entirely on the script provided by HarvardX for this course. The code contained on the script can be found below. Please note that depending on your internet conection and system characteristics, running this code and/or rendering the Rmd version of this report can take several minutes.

```
############################################################
# Create edx set, validation set, and submission file #
############################################################

# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------------------------------------------------
```

```
## v ggplot2 3.1.0       v purrr   0.3.1
## v tibble  2.0.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

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

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

ratings <- read.table(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")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data

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

# Make sure userId and movieId in validation set are also in edx set

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

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
```

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

```
edx <- rbind(edx, removed)

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

Once the data adquisition process has finished, we will quickly check the structure of the data sets we obtained to gain some preliminary insights. For this, we will simply use the `str()` function.

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  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 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
```

```
str(validation)
```

```
## 'data.frame':    999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Roman
```

Once we have check the data set structure, we are ready to proceed to the Data Exploration step to gather more information about the data.

3 Data Exploration

The first, yet very important insight that we gained on the preliminary data exploration we made on the previous step is that both, the `edx` and `validation` data sets follow the same structure, and we can also validate that looking at the first entries using the `head()` function.

```
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
```

```
head(validation)
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                  title
## 1                                  Dumb & Dumber (1994)
## 2                                  Jurassic Park (1993)
## 3                                     Home Alone (1990)
## 4                                        Rob Roy (1995)
## 5                                   Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                  genres
## 1                                Comedy
## 2        Action|Adventure|Sci-Fi|Thriller
## 3                          Children|Comedy
## 4                 Action|Drama|Romance|War
## 5                             Crime|Drama
## 6 Action|Adventure|Horror|Sci-Fi|Thriller
```

As we can see also from the Preliminary Data Exploration, the `edx` data set is very large, having over 9 million ratings. On the other hand, the `validation` data set has almost a million entries, more specifically, it has $999,999$. Taking this into account, we will initially try to use the same approach as on the Introduction to Data Science book provided by Professor Rafael A. Irizarry. This resource was mentioned on the Welcome to Data Science: Capstone section of the course. We will take this approach because fitting models with this

data set sizes would take too much time and might probably cause R to crash while trying to run the code, therefore, making it very dificult to reproduce the results.

We will start by checking out the number of unique users and the number of unique movies on both data sets:

```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

```
validation %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   68534     9809
```

We can multiply the number of users by the number of movies in both cases to then compare the result with the data set sizes, so we can know whether every user rated every movie or not.

```
edx_users_and_movies <- edx %>%
                          summarize(n_users = n_distinct(userId),
                          n_movies = n_distinct(movieId))

(edx_users_and_movies$n_users * edx_users_and_movies$n_movies) == nrow(edx)
```

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

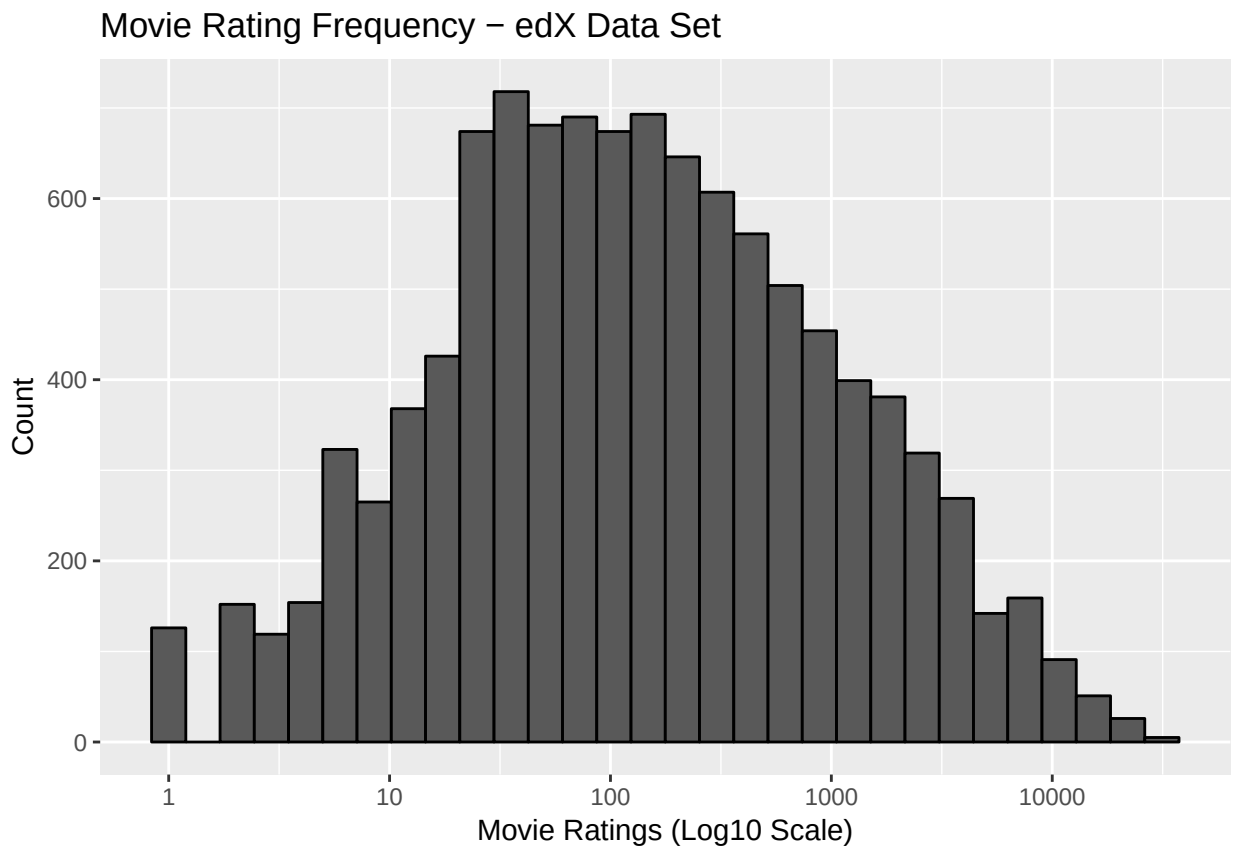
```
val_users_and_movies <- validation %>%
                          summarize(n_users = n_distinct(userId),
                          n_movies = n_distinct(movieId))

(val_users_and_movies$n_users * val_users_and_movies$n_movies) == nrow(validation)
```

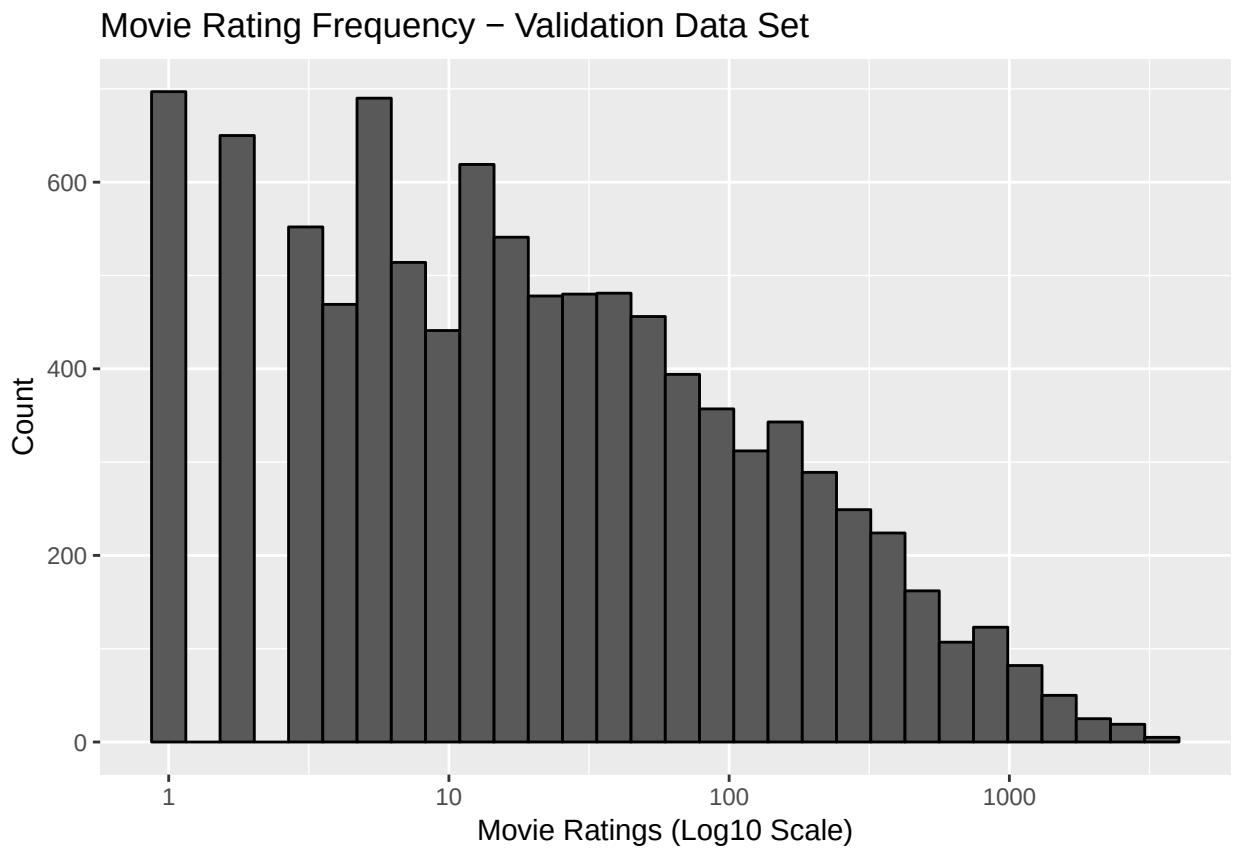
```
## [1] FALSE
```

As we can see, in both cases not every user rated every movie. Now, our final goal is to make a model that is able to predict the rating that a user would give to a movie, to then decide whether or not to recommend it to the user. Before we dive into the model development, we will check out the distribution of movie rating frequency as well as the user rating frequency on both data sets to make sure we see about the same pattern.

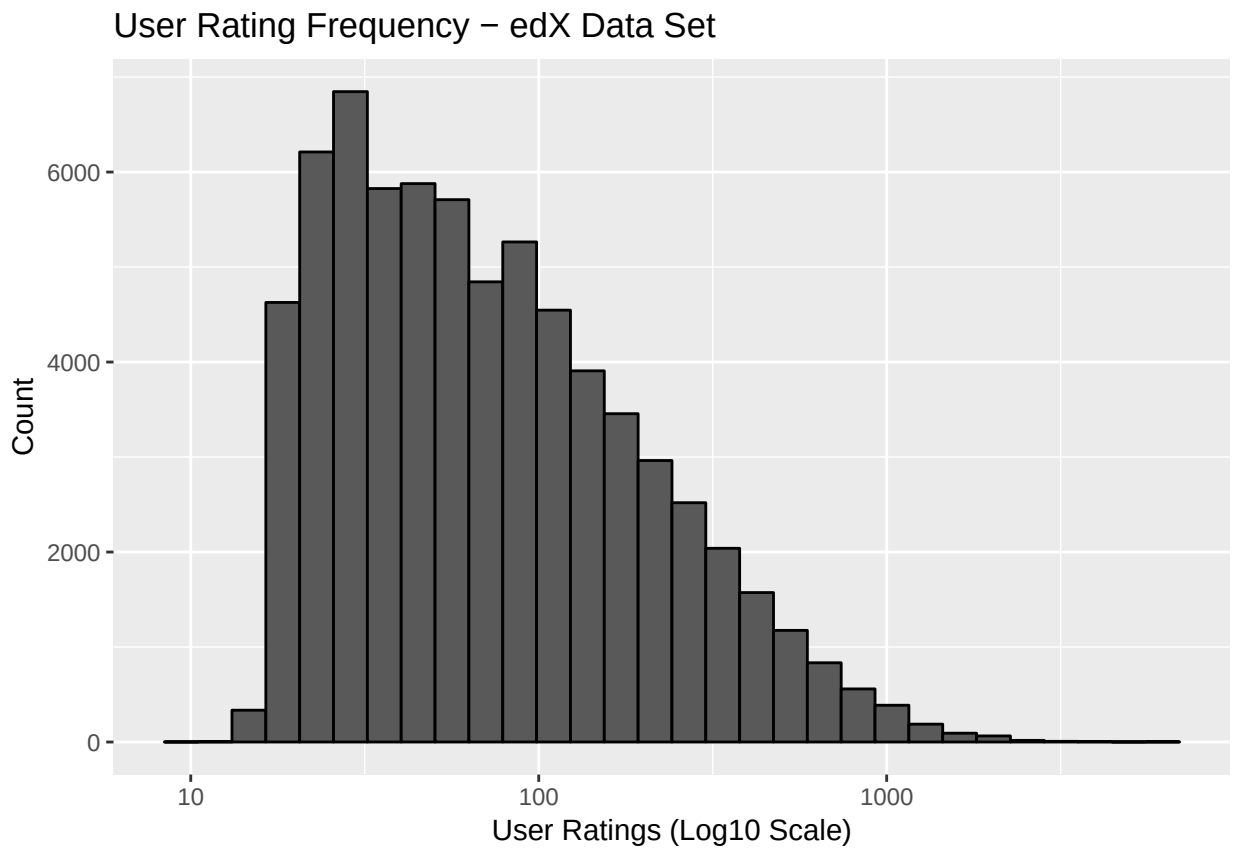
```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  labs(x = "Movie Ratings (Log10 Scale)", y = "Count", title = "Movie Rating Frequency - edX Data Set")
```



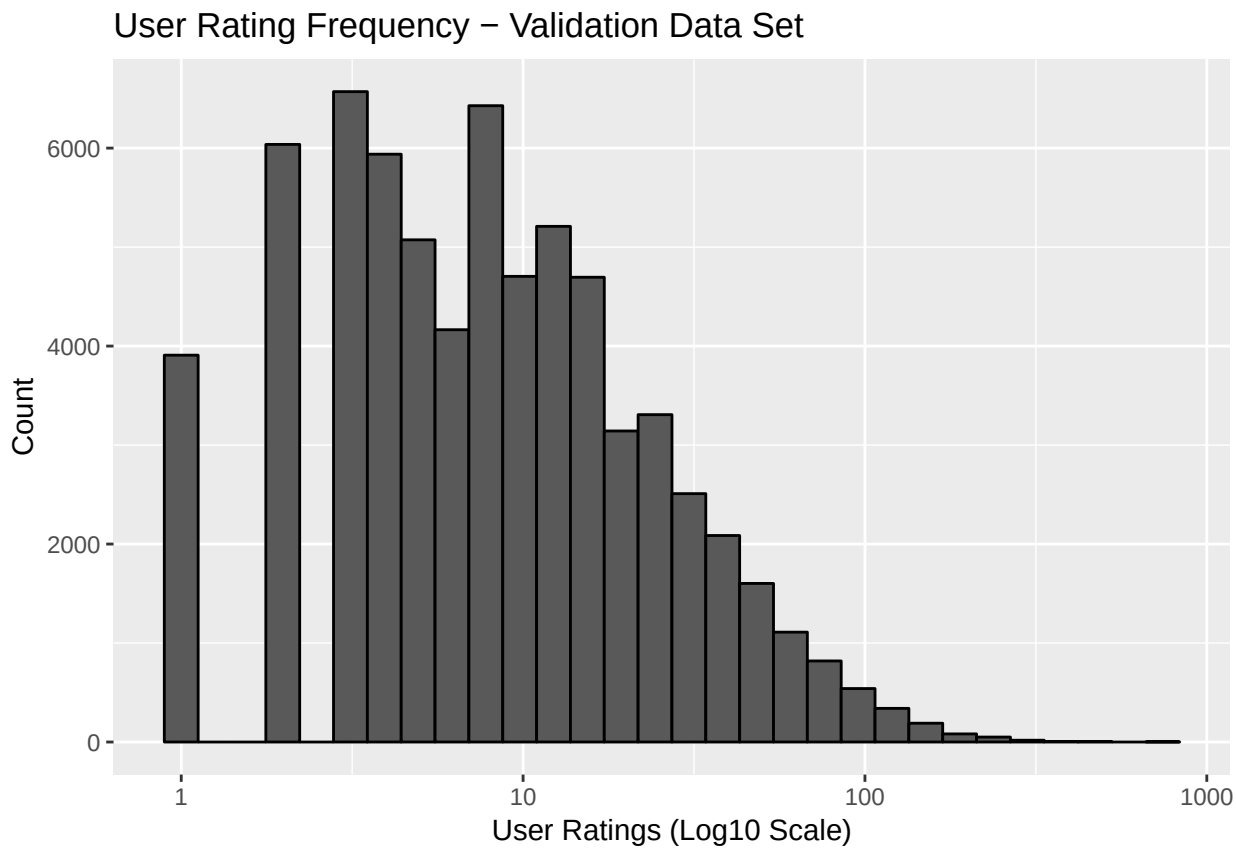

```
validation %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  labs(x = "Movie Ratings (Log10 Scale)", y = "Count", title = "Movie Rating Frequency - Validation Data
```




```
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  labs(x = "User Ratings (Log10 Scale)", y = "Count", title = "User Rating Frequency - edX Data Set")
```



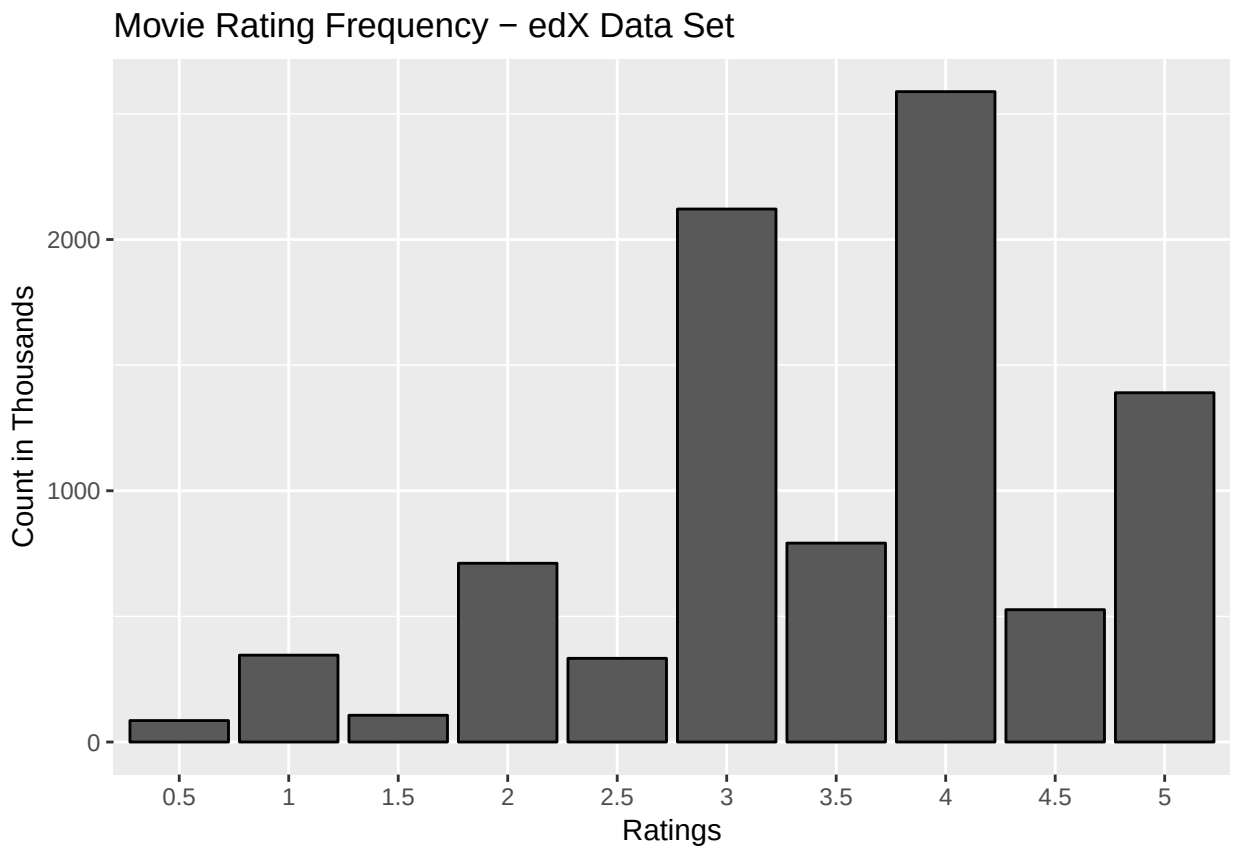

```
validation %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  labs(x = "User Ratings (Log10 Scale)", y = "Count", title = "User Rating Frequency - Validation Data S
```



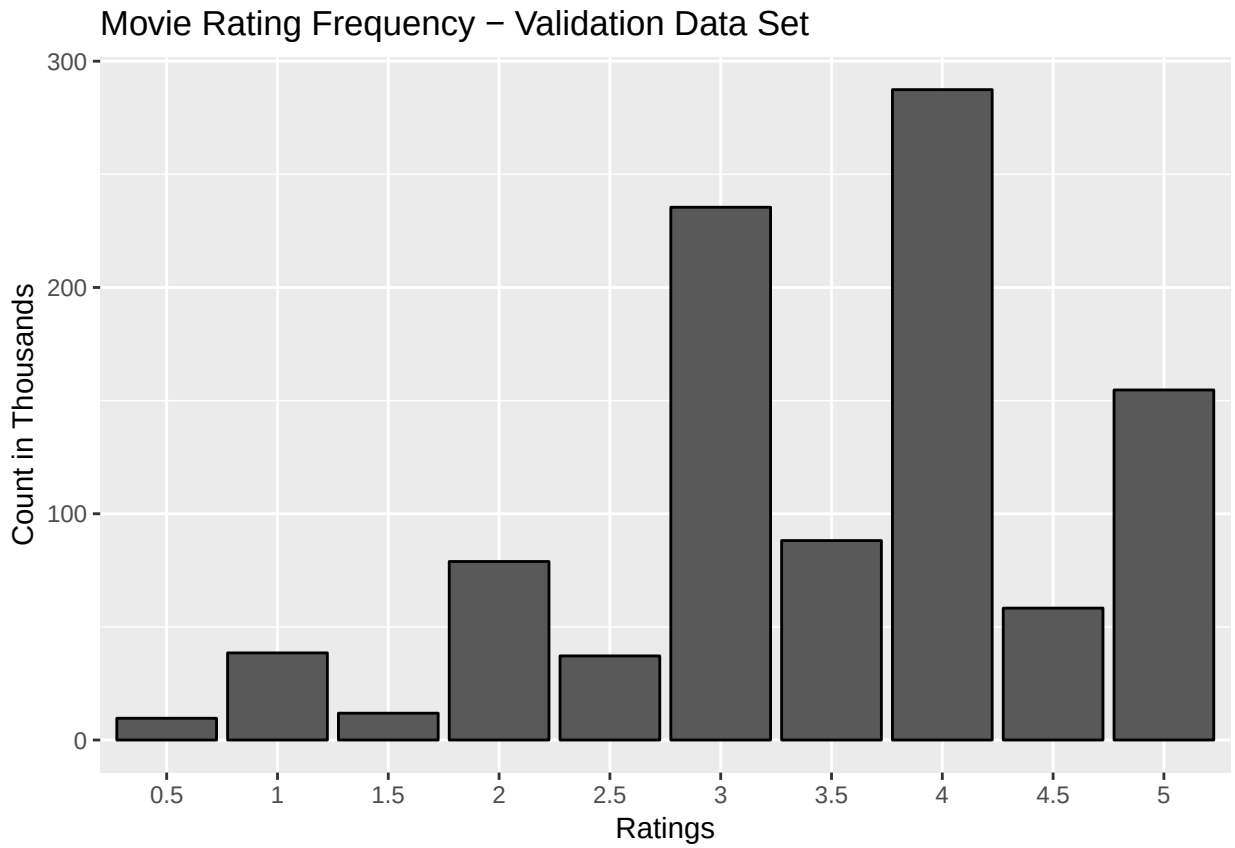
As we can see, both data sets follow a very similar distribution, telling us practically that some movies are rated more frecuently than others and some users are more active rating movies than others, which is expected as the validation data set is a partition from the 10M MovieLens data set. Still, it is useful to make sure both data sets follow a similar structure on their distribution. When we go back to the user and movie count, we can actually see that the number of users and movies is very similar on both data set, and the difference is very likely to be caused due to the removal of ratings we made on the Data Adquisition step when we removed users and movies that were present on the validation data set but not on the edx data set.

Lastly, we will check the rating distribution on both data sets to see what rating value is the most common accross the users. This will serve us as a guideline to evaluate the viavility of using the average rating as an initial prediction. If we see that both data sets follow a similar rating distribution, we will use that fact to start our modeling approach.

```
edx %>%
mutate(rating = as.factor(rating)) %>%
  count(rating) %>%
  ggplot(aes(x = rating, y = n/1000)) +
  geom_col(color = "black") +
  labs(x = "Ratings", y = "Count in Thousands", title = "Movie Rating Frequency - edX Data Set")
```




```
validation %>%
  mutate(rating = as.factor(rating)) %>%
  count(rating) %>%
  ggplot(aes(x = rating, y = n/1000)) +
  geom_col(color = "black") +
  labs(x = "Ratings", y = "Count in Thousands", title = "Movie Rating Frequency - Validation Data Set")
```



As we can see, we are seeing almost the same ratings distribution on both data sets, clearly seeing that the 4-Stars rating is the most common and the Half-Star ratings are less common than whole stars ratings. The facts we are seeing about the same distribution on both data sets and we see a clear most-common rating on both data sets, we can start creating our modeling approach.

4 Modeling

Now that we will start building our model, we will only work on the edX data set, as the validation data set will work as a new and unseen data for our model. As we saw previously, the most common rating accross the edX data set was 4, so we will start with a very simple model, predicting that rating regardless of everything else. We will use the $RMSE$ to evaluate our system, so we will start by defining a function that computes that value for us, as using any algorithm included on a library would take way too long to train because of the data set size. In case we are unable to achieve our $RMSE$ goal, we will evaluate further approaches.

The $RMSE$ formula is defined as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

The $RMSE$ can be interpreted like a standard deviation, meaning that the value we get for this it is basically the typical error we do a prediction. So, if we get $RMSE > 1$ we are failing by more than one star in our rating prediction, which is not good at all.

First we will define the function to calculate the $RMSE$, which will be as follows:

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

We will start with a very simple approach, predicting the most-common rating of the `edX` data set (4-Star rating) regardless of the rest of the data and then evaluate the $RMSE$ we get from the prediction.

4.1 Most-Common Rating Model

```
mc_rmse <- RMSE(edx$rating, 4)
mc_rmse
```

```
## [1] 1.167044
```

As expected, our $RMSE$ is quite bad, we are predicting with an error of approximately 1.17 stars. Still, this sets us a baseline for our next modeling approaches. To keep our reported results tidy, we will be storing our results on a chart.

```
rmse_results <- data_frame(Model = "Most-Common Rating Model", RMSE = mc_rmse)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.167044

4.2 Average Rating Model

Taking into account our previous result, we will base our next model on the average rating of the data set, which we can write as:

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

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the "true" rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimizes the $RMSE$ is the least square estimate of $Y_{u,i}$, in this case, it's the average of all ratings:

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

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

As we can see, we got a somewhat close value to our initial approach of predicting the 4-Star rating as it is the most common one in the data. This should bring down the $RMSE$ because the average should minimize the $RMSE$ as explained above. We will compute now the $RMSE$ with the prediction being the average rating:

```
avg_rating_rmse <- RMSE(edx$rating, mu)
avg_rating_rmse
```

```
## [1] 1.060331
```

We will now save the result of our new model into the chart we built earlier:

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model ="Average Rating Model",
                                     RMSE = avg_rating_rmse))
rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.167044
Average Rating Model	1.060331

We can see an improvement when compared to our previous $RMSE$. This suggest we can do even better, we will build the next model based upon our previous Average Rating Model.

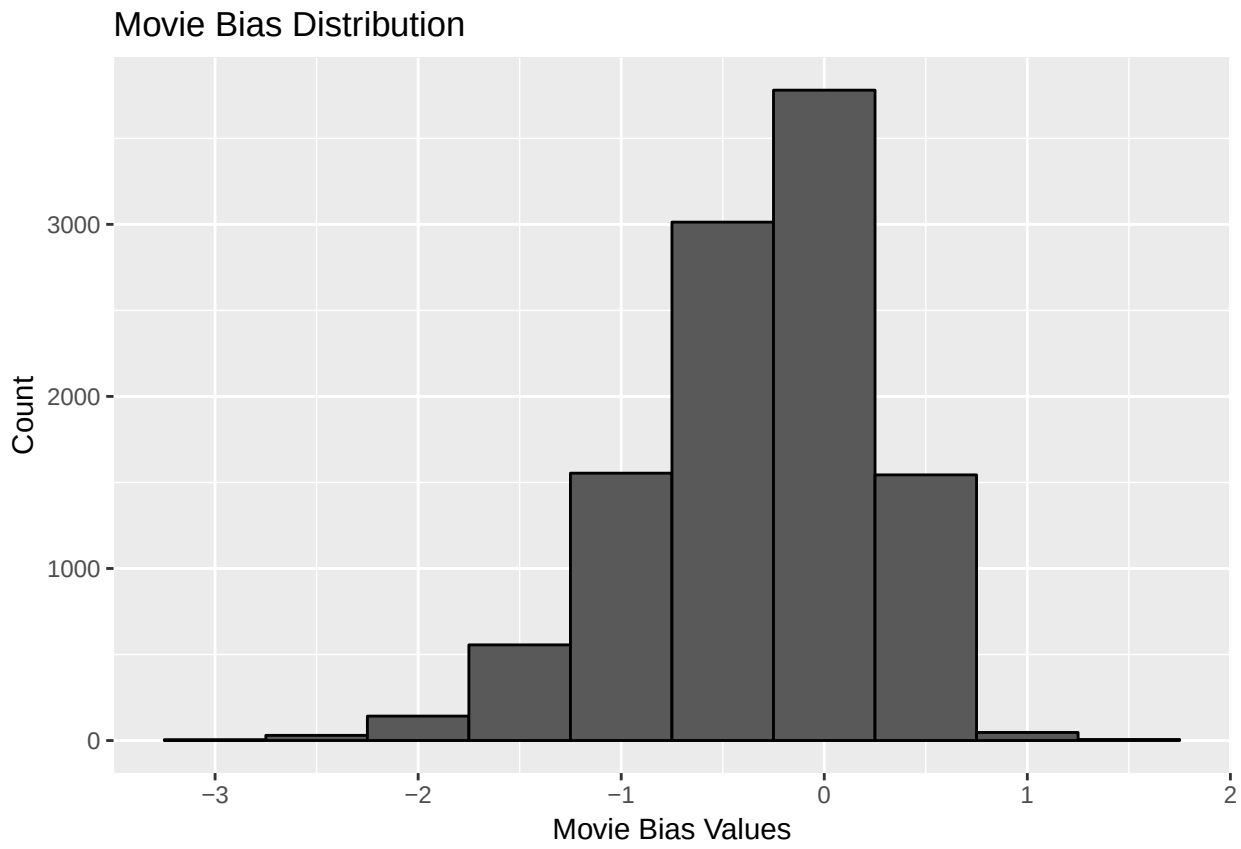
4.3 Movie Effect Model

By experience, we know some movies are rated higher than others, so we will take this fact into consideration, adding a Movie Effect term into our model. We will call this term b_i because basically we will be taking into account the "Bias" (b) for each movie (i). We can write it as follows:

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

First, we will look at the distribution of the b_i term, to do so, we will first compute it and then plot the results as follows:

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% ggplot(aes(b_i)) +
  geom_histogram(bins = 10, color = "black") +
  labs(x = "Movie Bias Values", y = "Count", title = "Movie Bias Distribution")
```



As we can see, the histogram is skewed to the left, meaning that it is usual to see movies that have bad ratings. Now we have our movie penalty taken into account, so given that we have a $\hat{\mu} = 3.5$ on our average, a value of $b_i = 1.5$ would imply a perfect 5-Star rating. Let's see how this new term affects our results:

```
predicted_ratings <- mu + edx %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

movie_bias_rmse <- RMSE(predicted_ratings, edx$rating)
movie_bias_rmse
```

```
## [1] 0.9423475
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model ="Movie Bias Model",
                                     RMSE = movie_bias_rmse))
rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.1670444
Average Rating Model	1.0603313
Movie Bias Model	0.9423475

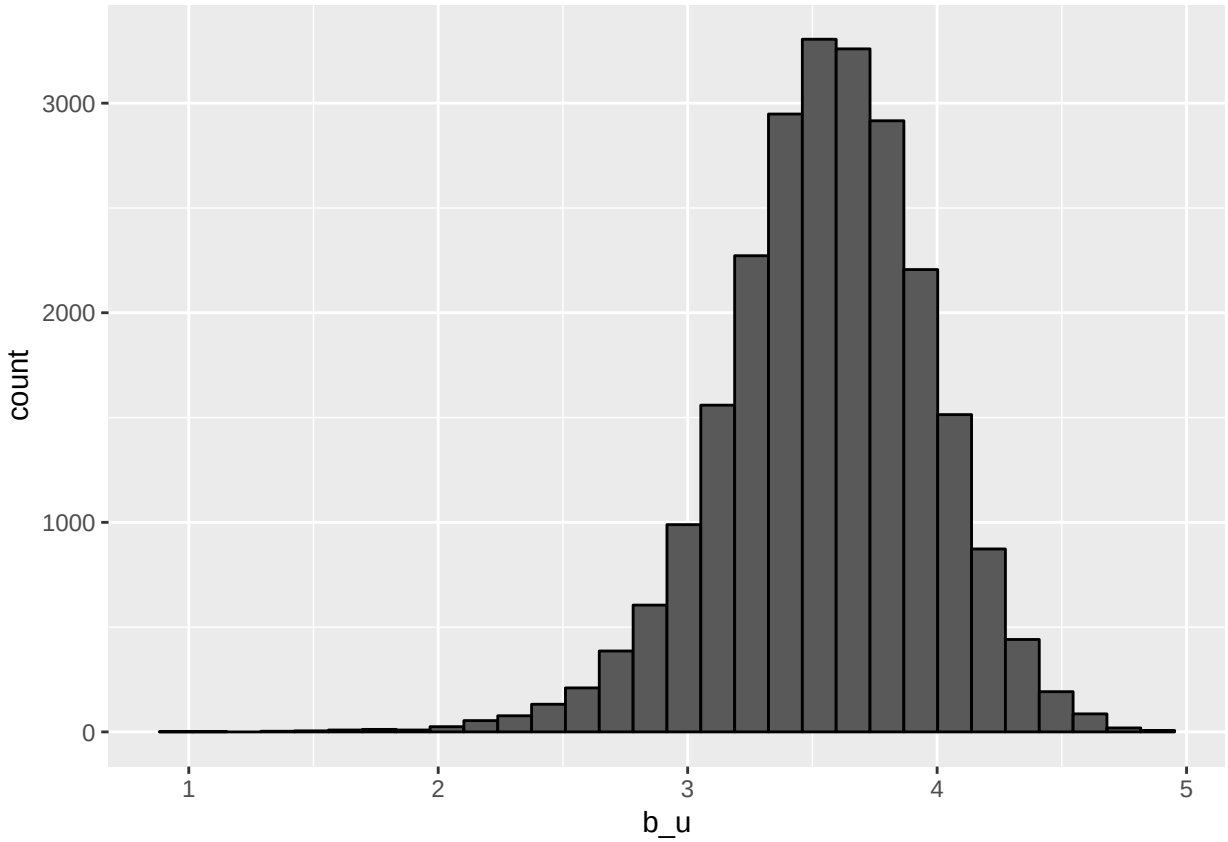
As we can see, we are now below 1, meaning that we managed to reduce our average error without prediction to less than 1 star. Still, we need to improve our model to be able to achieve our $RMSE < 0.8775$ goal.

4.4 Movie Bias and User Bias Model

We know this because of the insights we gained on the Data Exploration phase that there are some users that tend to rate the movies lower than the average while there are some others that do the opposite. We will try to add this into consideration to our existing model since we definitely improved the results of our previous model when we included the Movie Bias. Let's see if we see an improvement when we also include a User Bias into the model.

First of all, we will only take into consideration users that have rated 100 movies or more, so our model is based on active users. Let's begin with computing the average rating for user u that have rated 100 movies or more and then we will plot the distribution:

```
edx %>%
  group_by(userId) %>%
  filter(n()>=100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



As we expected, we see the vast majority of the data around the computed average, still, we can see that some users tend to give a worse rating while some others tend to give a better rating, so taking this fact into consideration should improve our $RMSE$ values. Now that we have seen this new term can be handy for our model, we can write it as follows:

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

The term b_u is a user-specific effect, so if a fussy user gives a bad rating (we would have a negative value of b_u for this user) to a great movie (which should have a positive b_i or in other words, a positive movie bias), both values should counter each other so we might be able to predict that this user gave to a great movie a 3-Star rating rather than a 5-Star rating, and by doing so, we should have a predicted rating much closer to the actual rating, therefore reducing our $RMSE$ when we predict for all users.

Now, we will compute our user bias and then use it as a new predictor to our model to see if our $RMSE$ improves:

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

```
movie_user_bias_rmse <- RMSE(predicted_ratings, edx$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Movie + User Bias Model",
                                     RMSE = movie_user_bias_rmse))
rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.1670444
Average Rating Model	1.0603313
Movie Bias Model	0.9423475
Movie + User Bias Model	0.8567039

As we can see, we achieved our $RMSE$ goal with this model. Now that we have defined several approaches and gradually improved them, we will now test them on our validation data set and see if we get similar results.

5 Model Testing (Results)

We will now run each model we've built on the validation data set to make sure our final model holds up with our goal and to evaluate the results obtained on the other models compared to the ones we get with the validation data set

5.1 Most-Common Rating Model

We will start by running our first model on the validation set and see how it performed, to then store the results into a new chart.

```
mc_rmse <- RMSE(validation$rating, 4)
mc_rmse
```

```
## [1] 1.168016
```

```
val_rmse_results <- data_frame(Model = "Most-Common Rating Model", RMSE = mc_rmse)
val_rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.168016

As we can see, we got a pretty similar result as the first one. Now we will run the Average Rating Model with the validation data set and see how it performed.

5.2 Average Rating Model

```
val_avg_rating_rmse <- RMSE(validation$rating, mu)
val_avg_rating_rmse
```

```
## [1] 1.061202
```

```
val_rmse_results <- bind_rows(val_rmse_results,
                          data_frame(Model ="Average Rating Model",
                                     RMSE = val_avg_rating_rmse))
val_rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.168016
Average Rating Model	1.061202

As the `validation` data set is a smaller data set compared to the `edx` data set, it is expected that the $RMSE$ increased a bit when we used the Average Rating Model, still, we got about the same result as with the one obtained on the `edx` data set. Now, we will check the Movie Bias Model.

5.3 Movie Bias Model

```
val_movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

val_predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

val_movie_bias_rmse <- RMSE(val_predicted_ratings, validation$rating)
val_movie_bias_rmse
```

```
## [1] 0.9439087
```

```
val_rmse_results <- bind_rows(val_rmse_results,
                          data_frame(Model ="Movie Bias Model",
                                     RMSE = val_movie_bias_rmse))
val_rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.1680160
Average Rating Model	1.0612018
Movie Bias Model	0.9439087

As in this case our model is still based on the `edx` data set, therefore, using the average calculated which was based on that data set, it was expected to get again a slighty greater $RMSE$, still, yielding a very similar result. Lastly, we will check for our final model performance with the validation data set.

5.4 Movie and User Bias Model

```
val_user_avgs <- edx %>%
  left_join(val_movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

val_predicted_ratings <- validation %>%
  left_join(val_movie_avgs, by='movieId') %>%
  left_join(val_user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

val_movie_user_bias_rmse <- RMSE(val_predicted_ratings, validation$rating)
val_movie_user_bias_rmse
```

```
## [1] 0.8653488
```

```
val_rmse_results <- bind_rows(val_rmse_results,
                          data_frame(Model="Movie + User Bias Model",
                                     RMSE = val_movie_user_bias_rmse))
val_rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.1680160
Average Rating Model	1.0612018
Movie Bias Model	0.9439087
Movie + User Bias Model	0.8653488

This time we got a larger increase on the $RMSE$ than we got on the `edx` data set, mainly because we are still relying on values originally computed on that data set to then test the model on a different and smaller data set. Still, our model held to accomplish our goal of achieving an $RMSE < 0.87750$.

6 Conclusions

As we saw on the Model Testing phase, we achieved our goal of an $RMSE < 0.87750$ that held through both `edx` and `validation` data sets. In this case our final model is:

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

Which gave us the following results from the training set (`edx` data set) and test set (`validation` data set):

```
rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.1670444
Average Rating Model	1.0603313
Movie Bias Model	0.9423475
Movie + User Bias Model	0.8567039

```
val_rmse_results %>% knitr::kable()
```

Model	RMSE
Most-Common Rating Model	1.1680160
Average Rating Model	1.0612018
Movie Bias Model	0.9439087
Movie + User Bias Model	0.8653488

In this type of machine learning challenges, is very important to take into account biases that might be affecting our data. By doing so, we will be able to create more accurate and adaptative models depending on the case study we are facing. Another potential improvement we could add to our model is a Genre Bias as by experience we can tell that there are some movie genres that tend to be rated higher than others. Even though we will not cover it on this project, that observation can be taken into consideration if we would like to improve further our final model.

As a final conclusion for this project, we can say that on machine learning challenges that involves human bevaiour trying to replicate or quantify psycological behavior even when it can be very complex, it can also help a lot to improve the results of the model that is being developed as we clearly saw within this project when we went down from a $RMSE = 1.0612$ to an $RMSE = 0.8653$ just by including a Movie Bias and a User Bias into our final model.