MovieLens Project

Akash Kumar

June 5, 2019

Contents

Introduction **2**
- Background . . . 2
- Inspiration . . . 2
- Focus . . . 2
- Dataset . . . 2
- Goal . . . 2
- Setup . . . 2

Methods and Analysis **3**
- Data Preparation . . . 3
  - Here is the raw, uncleaned edx dataset: . . . 3
  - Cleaning the data: . . . 4
  - Here is the cleaned edx dataset: . . . 4
- Exploratory Data Analysis (EDA) . . . 5
  - Here is a brief summary of the cleaned edx dataset: . . . 7
- Plots and Visualizations . . . 7
  - 1. Distribution of movie ratings . . . 8
  - 2. Age of the movie . . . 8
  - 3. Movie genre . . . 9
  - 4. Popularity of the movie . . . 11
  - 5. Weighted rating of the movie . . . 12
  - 6. Average rating per user . . . 13
- Modelling and Training . . . 14
  - Evaluation with RMSE . . . 14
  - Models . . . 15
  - Model 1: Naive Baseline (mu) . . . 15
  - Model 2: Movie Effects (mu + b_i) . . . 15
  - Model 3: Movie Effects + User Effects (mu + b_i + b_u) . . . 17
  - Model 4: Movie Effects + User Effects + Regularization (mu + b_i + b_u + Regularization) . . . 18
- Validation . . . 20

Results **20**

Conclusion **21**

Introduction

Background

In the 21th century, online shopping and multimedia are used extensively by the general public. To generate the most profit from customers, online companies like Netflix, Facebook, and Amazon need to provide their users with what they most want and like. So, these companies use automated recommendation systems to provide good product recommendations to users based on past activity and the quality of their products. This leads to happier customers and higher profit margins.

Inspiration

This project is inspired by the 2006 Netflix Prize Challenge, where Netflix offered $1 million to the team that could improve their recommendation systems by 10%.

Focus

The focus of this project is to recreate our own version of Netflix's movie recommendation system.

Dataset

For this assignment, we will be using the 10M version of the MovieLens dataset.

Goal

The goal of the project is to develop and train models to predict users' movie ratings as accurately as possible. The RMSE of the predictions must be optimized such that it we have an **RMSE <= 0.87750**. For this project, we will be building these models using R (version 3.6.0).

Setup

To setup the dataset for analysis, we will be using the code provided by the edx staff.

The following code is used to create the edx set and validation set. Please note that this process can take a few minutes.

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

# 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")
```

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

Load all necessary libraries:

```r
library(tidyverse)
library(caret)
```

Methods and Analysis

Data Preparation

Here is the raw, uncleaned edx 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
```

```r
glimpse(edx)
```

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|D...
```

```r
summary(edx)
```

```
##      userId         movieId          rating        timestamp        
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08  
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09  
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09  
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
##     title              genres         
##  Length:9000055     Length:9000055    
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
##                                       
##                                       
##                                       
```

Cleaning the data:

After taking a brief glimpse at the edx dataset, we see that the title column in the contains two pieces of information: the name of the movie and the year that movie was released. To clean up the data, we can take all the release years and put them into a separate column called *release_year*. Then we can get the age of each movie by subtracting its release year from the present year, 2019. These ages will then be put into their own column called *movie_age*.

```r
# Each movie title contains the year in which that movie was released.
# To make the data cleaner, separate the release year from the movie title.
edx <- edx %>%
  extract(title, c("title", "release_year"),
          regex = "^(.*) \\((\\d*)\\)$", remove = TRUE)
edx$release_year <- as.numeric(edx$release_year)
# Add a column for the age (in years) of the movies.
edx <- edx %>% mutate(movie_age = 2019 - release_year)
edx <- edx[, c(1,2,3,4,6,8,5,7)] # Reorder the columns.
```

Here is the cleaned edx dataset:

```r
head(edx)    # Check out the new column.
```

```
##   userId movieId rating timestamp release_year movie_age
## 1      1     122      5 838985046         1992        27
## 2      1     185      5 838983525         1995        24
## 3      1     292      5 838983421         1995        24
## 4      1     316      5 838983392         1994        25
## 5      1     329      5 838983392         1994        25
## 6      1     355      5 838984474         1994        25
##                     title                        genres
## 1               Boomerang                Comedy|Romance
## 2                Net, The         Action|Crime|Thriller
## 3                Outbreak  Action|Drama|Sci-Fi|Thriller
## 4                Stargate       Action|Adventure|Sci-Fi
## 5 Star Trek: Generations Action|Adventure|Drama|Sci-Fi
## 6        Flintstones, The       Children|Comedy|Fantasy
```

```r
summary(edx) # Make sure data remains consistent.
```

```
##      userId         movieId          rating        timestamp        
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08  
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09  
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09  
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
##   release_year    movie_age         title              genres         
##  Min.   :1915   Min.   : 11.00   Length:9000055     Length:9000055    
##  1st Qu.:1987   1st Qu.: 21.00   Class :character   Class :character  
##  Median :1994   Median : 25.00   Mode  :character   Mode  :character  
##  Mean   :1990   Mean   : 28.78                                        
##  3rd Qu.:1998   3rd Qu.: 32.00                                        
##  Max.   :2008   Max.   :104.00                                        
```

Exploratory Data Analysis (EDA)

Let us start our EDA by first answering the quiz questions related to the MovieLens dataset.

```r
#================#
# Quiz Questions #
#================#
# Q1: How many rows and columns are there in the edx dataset?
dim(edx) # 9,000,055 rows and 8 columns
```

```
## [1] 9000055       8
```

```r
rating_count <- dim(edx)[1]
rating_count
```

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

```r
# Q2: How many zeros were given as ratings in the edx dataset?
edx %>% filter(rating == 0) %>% nrow() # No zeros were given as ratings.
```

```
## [1] 0
```

```r
# How many threes were given as ratings in the edx dataset?
edx %>% filter(rating == 3) %>% nrow() # 2,121,240 threes were given as ratings.
```

```
## [1] 2121240
```

```r
# Q3: How many different movies are in the edx dataset?
movie_count <- n_distinct(edx$movieId) # 10,677 movies.
movie_count
```

```
## [1] 10677
```

```r
# Q4: How many different users are in the edx dataset?
user_count <- n_distinct(edx$userId) # 69,878 users.
user_count
```

```
## [1] 69878
```

```r
# Q5: How many movie ratings are in each of the following genres in the edx dataset?
# Movies in Drama, Comedy, Thriller and Romance:
edx %>% filter(grepl("Drama", genres)) %>% nrow() # 3,910,127 movie ratings with Drama
```

```
## [1] 3910127
```

```r
edx %>% filter(grepl("Comedy", genres)) %>% nrow() # 3,540,930 movie ratings with Comedy
```

```
## [1] 3540930
```

```r
edx %>% filter(grepl("Thriller", genres)) %>% nrow() # 2,325,899 movie ratings with Thriller
```

```
## [1] 2325899
```

```r
edx %>% filter(grepl("Romance", genres)) %>% nrow() # 1,712,100 movie ratings with Romance
```

```
## [1] 1712100
```

```r
# Q6: Which movie has the greatest number of ratings? - Pulp Fiction (1994)
top_10_movies <- edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  top_n(10) %>%
  arrange(desc(count))
top_10_movies
```

```
## # A tibble: 10 x 2
##    title                                                 count
##    <chr>                                                 <int>
##  1 Pulp Fiction                                          31362
##  2 Forrest Gump                                          31079
##  3 Silence of the Lambs, The                             30382
##  4 Jurassic Park                                         29360
##  5 Shawshank Redemption, The                             28015
##  6 Braveheart                                            26212
##  7 Fugitive, The                                         26020
##  8 Terminator 2: Judgment Day                            25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) 25672
## 10 Batman                                                24585
```

```r
# Q7: What are the five most given ratings in order from most to least?
# 4 > 3 > 5 > 3.5 > 2
edx %>% group_by(rating) %>%
  summarize(count = n()) %>%
  top_n(5) %>%
  arrange(desc(count))
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

```r
# Q8: True or False: In general, half star ratings are
# less common than whole star ratings. TRUE
rating_table <- edx %>% group_by(rating) %>% summarize(count = n())
rating_table <- as.data.frame(rating_table)
rating_table
```

```
##    rating   count
## 1     0.5   85374
## 2     1.0  345679
## 3     1.5  106426
## 4     2.0  711422
## 5     2.5  333010
## 6     3.0 2121240
## 7     3.5  791624
## 8     4.0 2588430
## 9     4.5  526736
## 10    5.0 1390114
```

```r
sum(rating_table[seq(1,9,2),2])  # 1,843,170 half star ratings.
```

```
## [1] 1843170
```

```r
sum(rating_table[seq(2,10,2),2]) # 7,156,885 whole star ratings.
```

```
## [1] 7156885
```

Here is a brief summary of the cleaned edx dataset:

This dataset contains 9,000,055 entries and 8 variables.

- *userId*: the ID number of the user.
- *movieId*: the ID number of the movie being rated.
- *rating*: the rating (ranging from 0 to 5) given to the movie.
- *timestamp*: the timestamp of the rating.
- *release_year*: the year the movie was released.
- *movie_age*: the age of the movie.
- *title*: the name of the movie.
- *genres*: the genres the movie fits.

There are 10,677 distinct movies and 69,878 distinct users. It is interesting to note that there are no ratings of 0. The mode of the movie ratings is 4.0, suggesting that there might be a rating bias. Later, we will incorporate this bias into our prediction models.

Plots and Visualizations

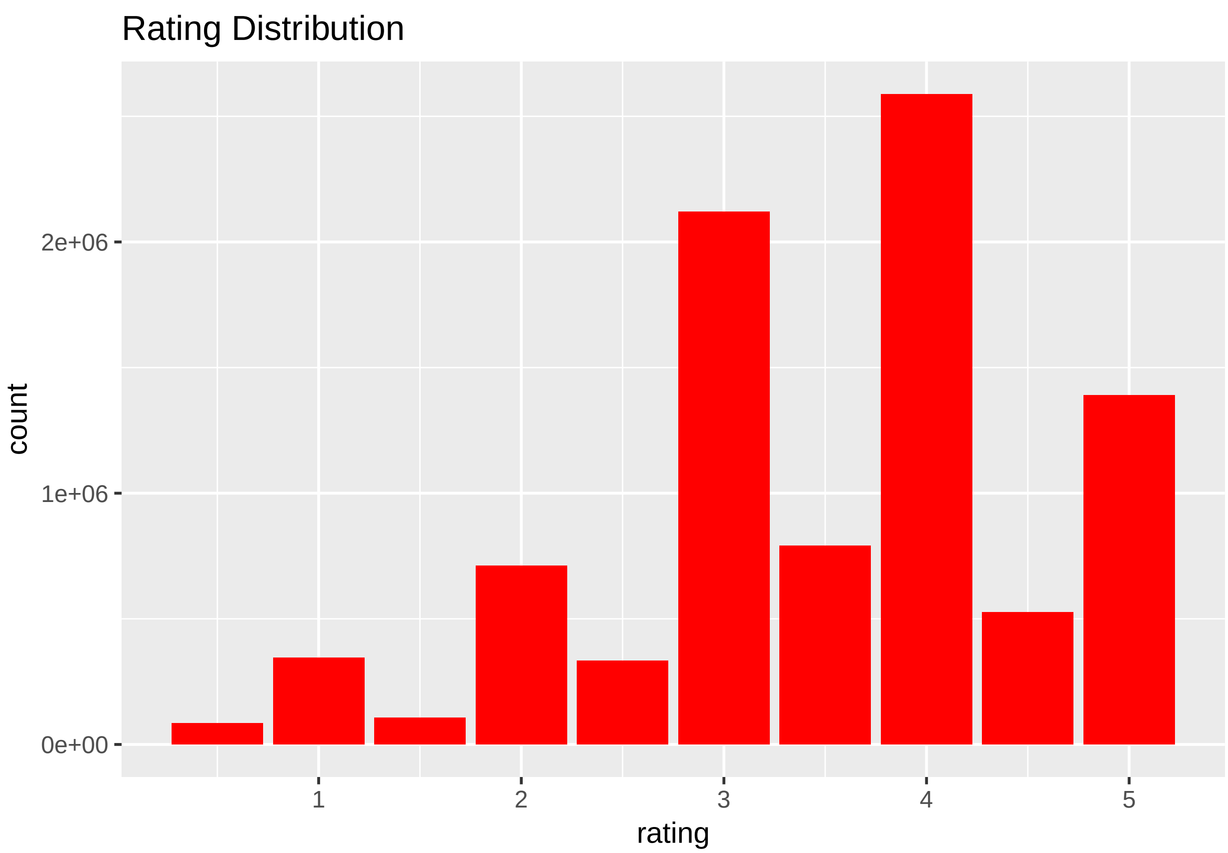
Before going forward, we need to consider some factors that might influence how a user rates a particular movie. Here are 6 such potential factors:

1. Distribution of movie ratings
2. Age of the movie
3. Movie genre
4. Popularity of the movie
5. Weighted rating of the movie
6. User's average rating

The following plots and visualizations will serve to highlight the 6 above mentioned factors and further aid in our EDA.

1. Distribution of movie ratings

First, we want to visualize the **Rating Distribution** using a histogram. To do this, we will be using the data from the *rating_table*.



From this histogram, we can easily see that most ratings are given 4.0 and there are relatively little ratings with 0.5. There is a clear bias towards ratings that are found between 3.0 and 4.0.

2. Age of the movie

The age of the movie may play a huge role in how often it gets viewed and how it gets rated. Looking at the age extremes, we see that the youngest movie is 11 years old and the oldest movie is 104 years old.

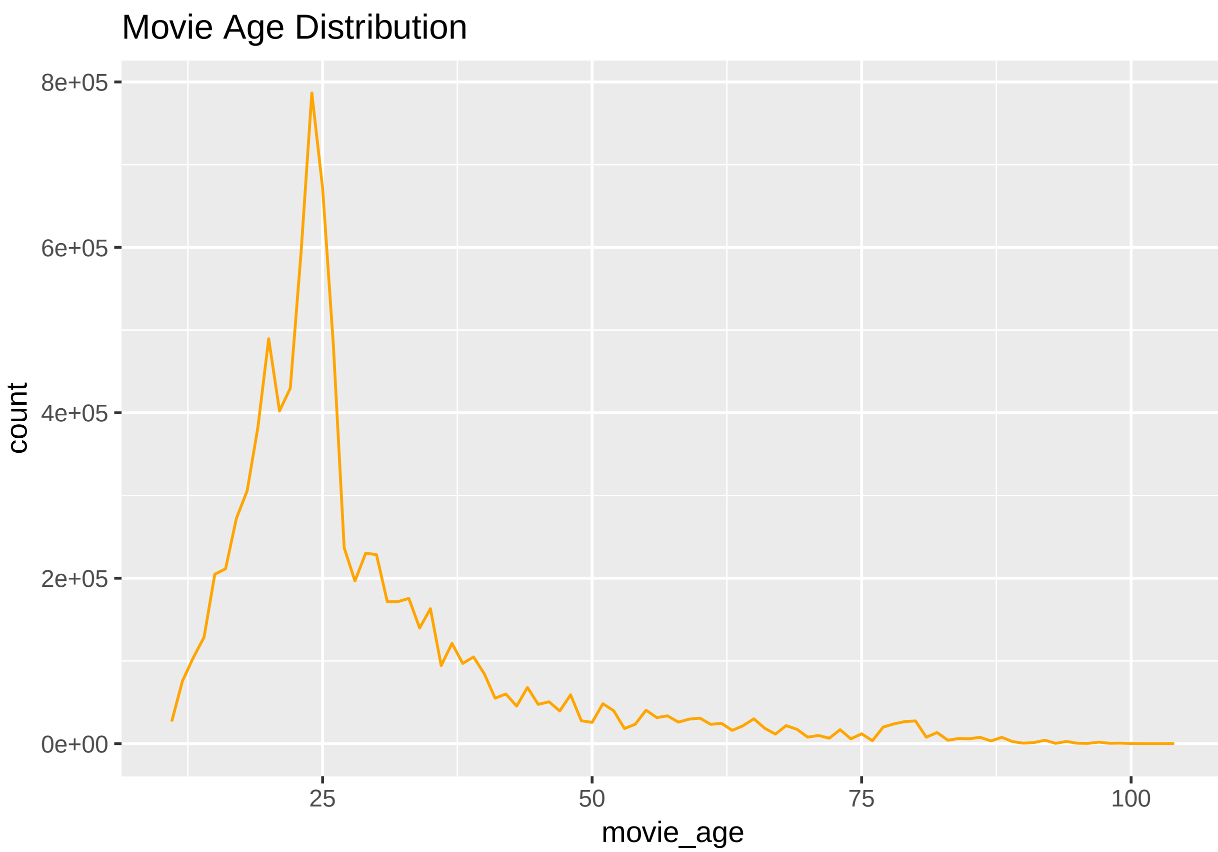
```
youngest_movie <- min(edx$movie_age)
youngest_movie # The most recent movie is 11 years old.
```

```
## [1] 11
```

```r
oldest_movie <- max(edx$movie_age)
oldest_movie   # The oldest movie is 104 years old.
```

```
## [1] 104
```

Plotting a **Movie Age Distribution** will allow us to see what the major age group for the movies is.



Notice the peak around age 25. Here, most of the movie ratings involve movies that are 13-37 years old. That means movies of this age group will have more ratings than movies in other age groups, making the user's rating of such movies more predictable based on the past ratings of other users.

3. Movie genre

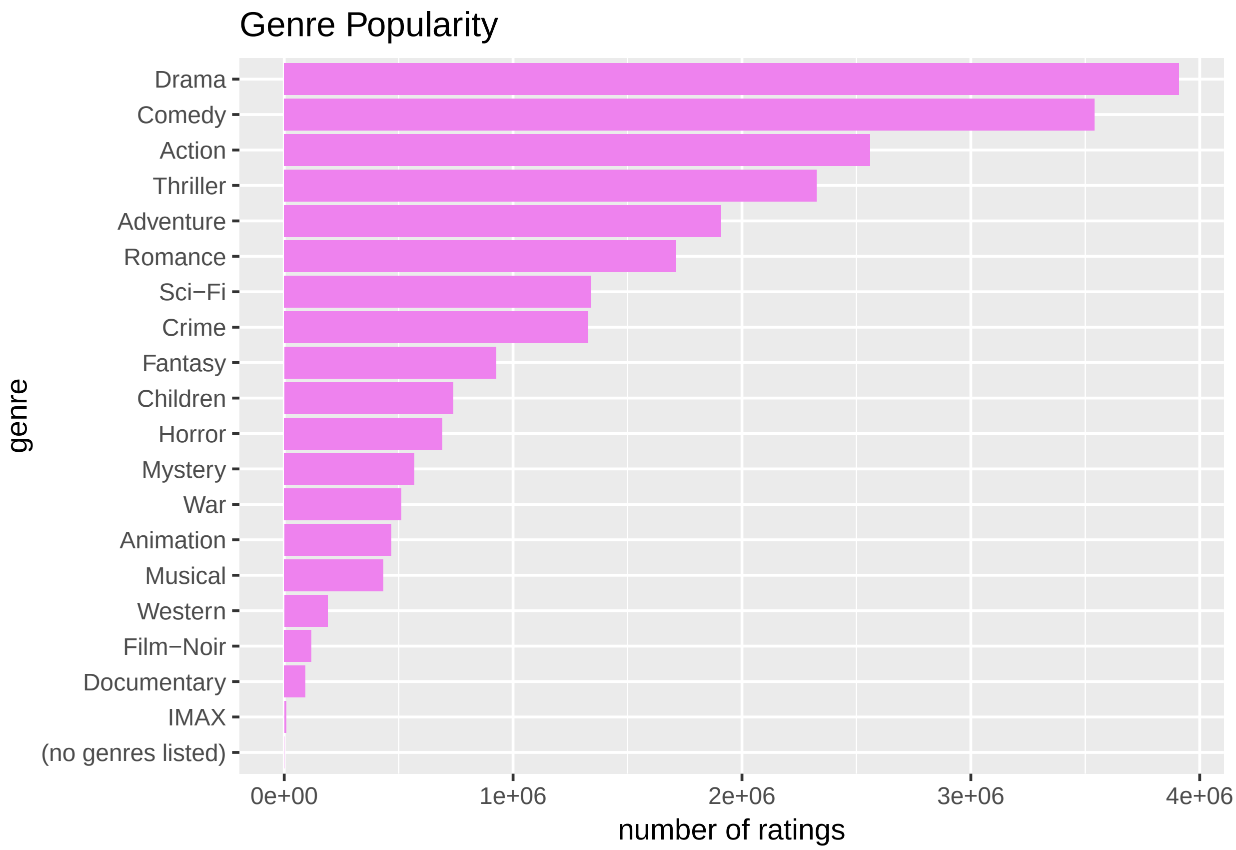
Among movie genres, some genres are more popular than others. Movies that fit a certain genre might get more views and/or better ratings.

```r
movie_genres <- edx %>% separate_rows(genres, sep = "\\|")
head(movie_genres)
```

```
##   userId movieId rating timestamp release_year movie_age     title
## 1      1     122      5 838985046         1992        27 Boomerang
## 2      1     122      5 838985046         1992        27 Boomerang
## 3      1     185      5 838983525         1995        24  Net, The
## 4      1     185      5 838983525         1995        24  Net, The
## 5      1     185      5 838983525         1995        24  Net, The
## 6      1     292      5 838983421         1995        24  Outbreak
##     genres
```

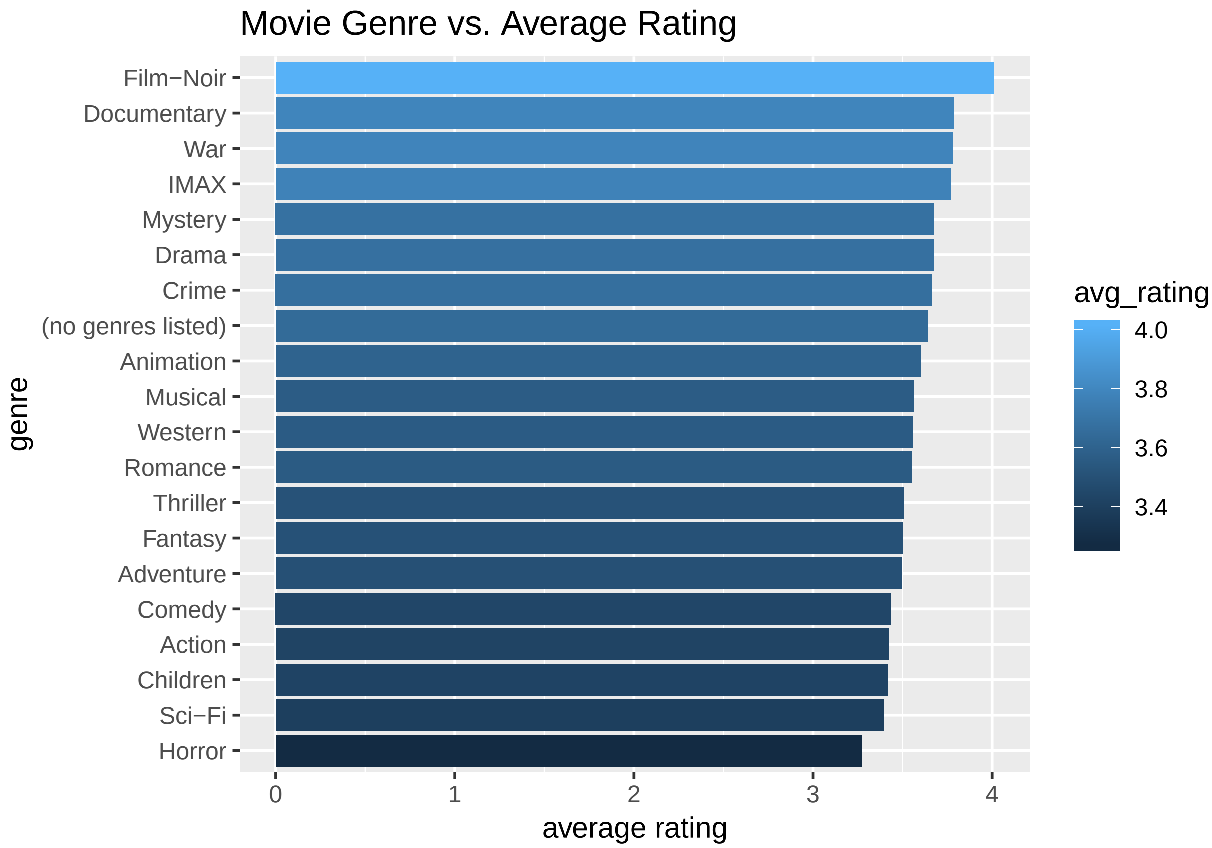
```
## 1   Comedy
## 2  Romance
## 3   Action
## 4    Crime
## 5 Thriller
## 6   Action
```

A histogram of the **Genre Popularity** will indicate the most popular genres.



Drama is the most popular movie genre.

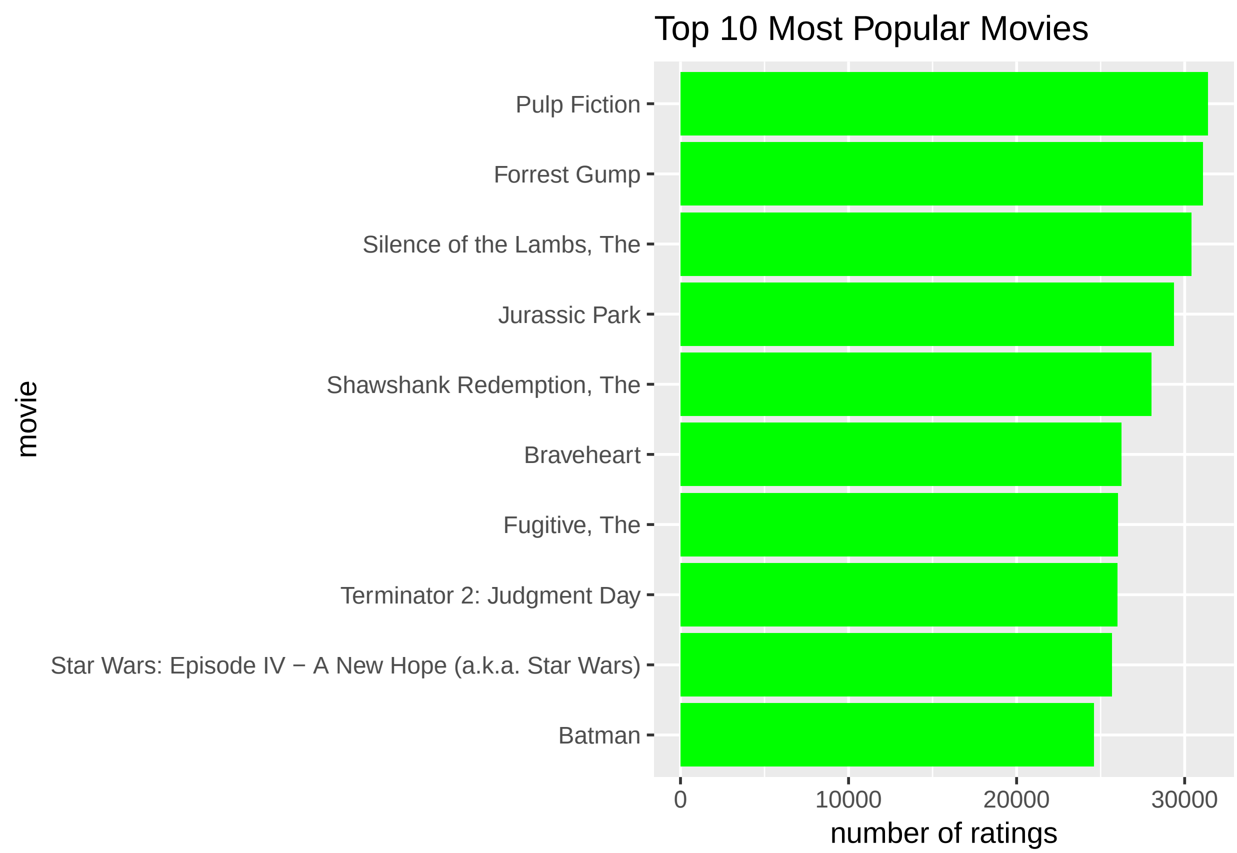
However, popularity alone does not determine how good a genre is. To do that, we need to look at the **Average Rating per Genre**.



There is not much change in the ratings among genres. Notice that obscure genres like "Film-Noir" are rated relatively high while mainstream genres like "Horror" and "Action" are rated relatively low. The reason for this is that films with obscure genres are viewed by less people and so, are not critiqued as harshly as the popular genres. Overall, however, there is not much change in the ratings among genres.

4. Popularity of the movie

Movie Popularity is based on the number of ratings. The more ratings a certain movie has, the more popular it is.



The most popular (most watched) movie is *Pulp Fiction*.

5. Weighted rating of the movie

Popular movies are not necessarily good movies. Likewise, obscure movies are not necessarily bad movies. So we need to use weighted rating to identity the best rated movies.

To calculate the weighted rating, we will be using IMDb's **weighted rating** function:

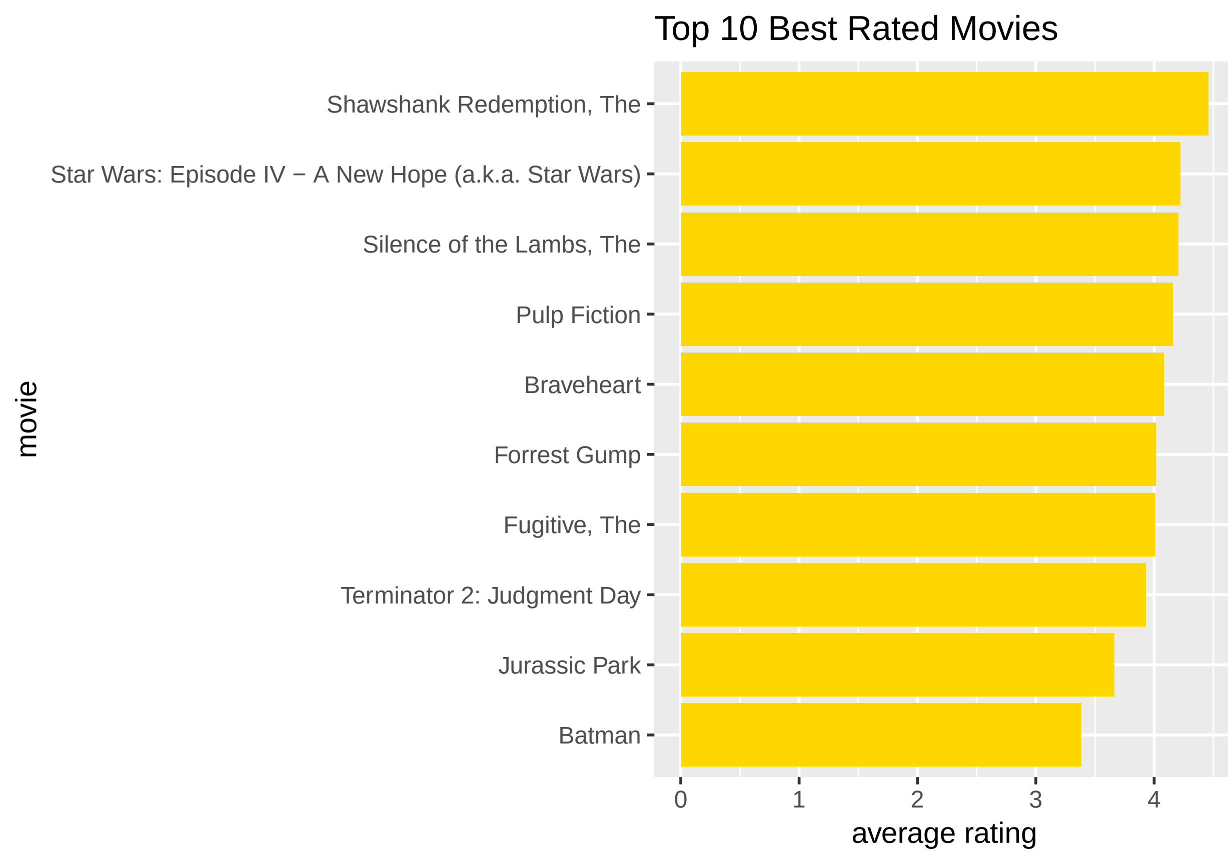
$$WR = \left(\frac{v}{v+m}\right) * R + \left(\frac{m}{v+m}\right) * C$$

R = average for the movie (mean) = (Rating)
v = number of votes for the movie = (votes)
m = minimum votes required to be listed in the Top 50 (currently 1000)
C = the mean vote across the whole report (average of average ratings)

The WR function in R:

```
wr <- function(R, v, m, C) {
  return (v/(v+m))*R + (m/(v+m))*C
}
```

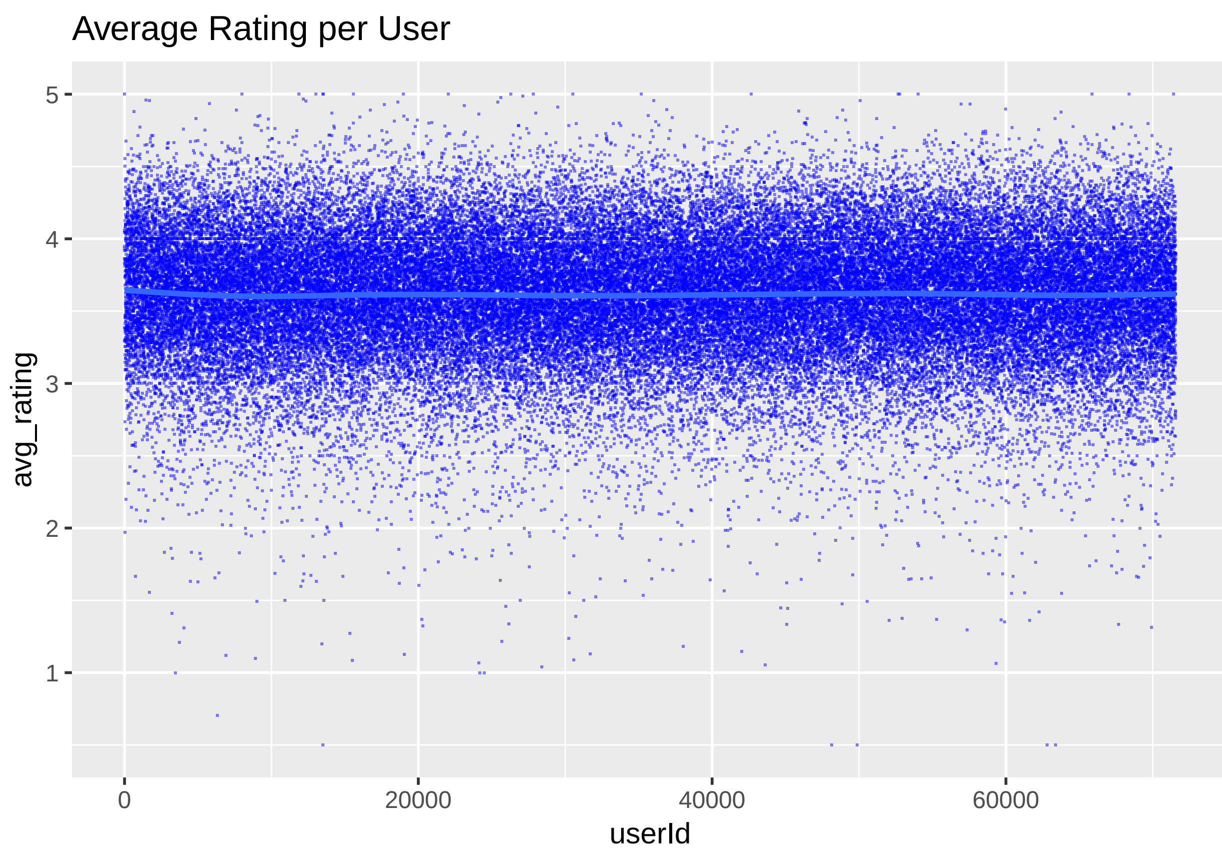
Having defined a weighted rating function, we can use this function to find the best rated movies.



The best rated movie is *The Shawshank Redemption*. Notice that this plot and the previous one contain the exact same movies.

6. Average rating per user

The **average rating** of an individual user will greatly help to predict their rating on any ambiguous movie.




```r
mean(user_avg_rating$avg_rating) # 3.614 (Mean average rating)
```

```
## [1] 3.613602
```

```r
mean(edx$rating)                  # 3.512 (Mean rating overall)
```

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

```r
summary(user_avg_rating)
```

```
##      userId        avg_rating
##  Min.   :    1   Min.   :0.500
##  1st Qu.:17943   1st Qu.:3.357
##  Median :35799   Median :3.635
##  Mean   :35782   Mean   :3.614
##  3rd Qu.:53620   3rd Qu.:3.903
##  Max.   :71567   Max.   :5.000
```

From this plot, we see that most of the average ratings range from 3.0 to 4.5. The mean average rating is 3.614, which is higher than the mean rating (3.512) of the edx dataset. This means that the individual is more likely to rate a movie higher than the overall population.

Modelling and Training

Evaluation with RMSE

In order to evaluate the performance of the prediction models, we will be using the Root Mean Square Error (RMSE) function. The RMSE is defined as:

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

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

Models

Based on what we learned from our EDA, we decided to build and test 4 models:

- Model 1: Naive Baseline (mu)
- Model 2: Movie Effects (mu + b_i)
- Model 3: Movie Effects + User Effects (mu + b_i + b_u)
- Model 4: Movie Effects + User Effects + Regularization (mu + b_i + b_u + Regularization)

These models will be trained on the edx dataset and the best model (Model 4) will be used to test the predictions on the validation set.

Model 1: Naive Baseline (mu)

Model 1 is the simplest of the 4 models. It basically uses the average rating (mu) of all the ratings as the prediction for all the movies. It ignores movie and user bias influence on the outcome of the prediction.

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

```
# Mean rating of the edx set:
mu_rating <- mean(edx$rating)
mu_rating # 3.512
```

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

```
# Calculate RMSE:
rmse_result_1 <- RMSE(validation$rating, mu_rating)
rmse_result_1
```

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

```
# Add RSME result to the results table.
rmse_results <- tibble(Method = "Naive Baseline", RMSE = rmse_result_1)
rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Baseline	1.061202

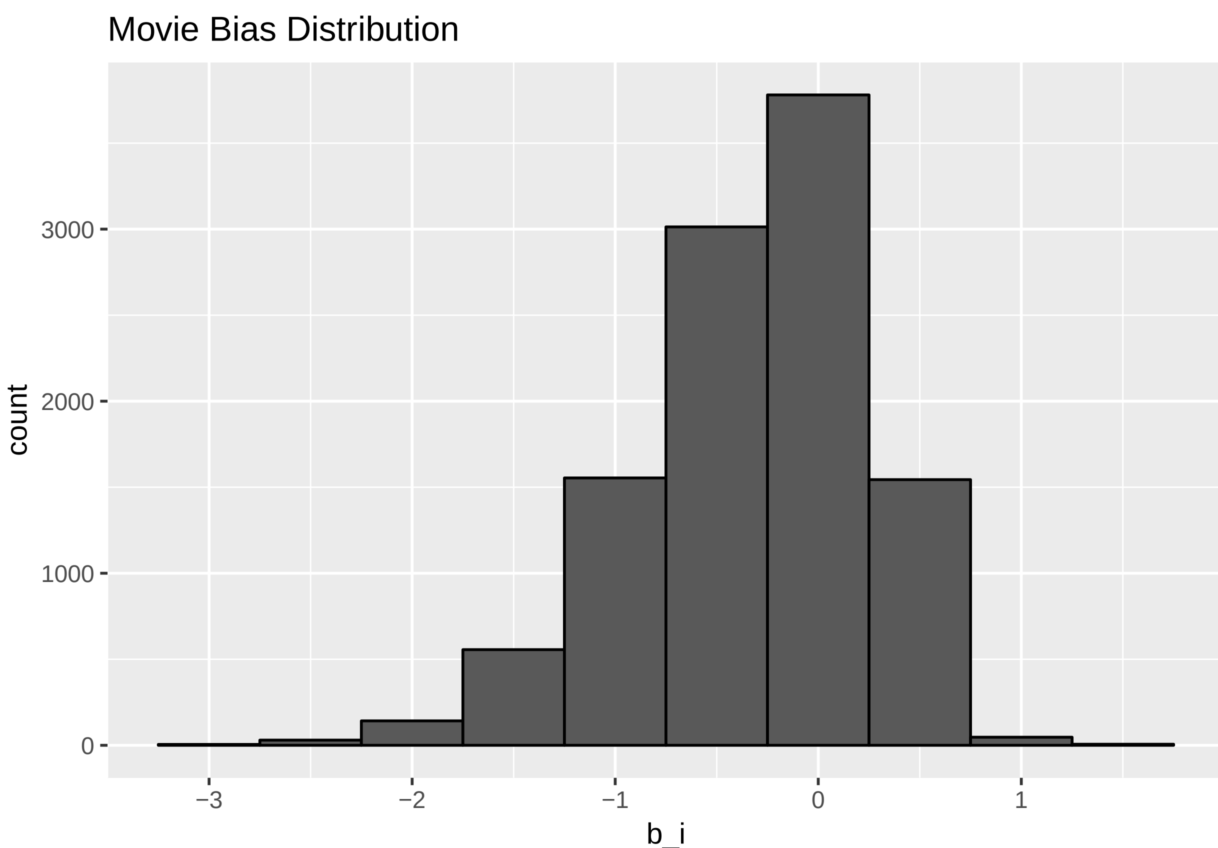
The predicted RMSE for Model 1 is about **1.06033**. An RMSE > 1 is typically not considered to be a good prediction.

Model 2: Movie Effects (mu + b_i)

From our EDA, we learned that some movies are rated higher than others. So we can add movie bias (b_i) to our previous model to improve our predictions.

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

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <dbl>  <dbl>
## 1       1  0.415
## 2       2 -0.307
## 3       3 -0.365
## 4       4 -0.648
## 5       5 -0.444
## 6       6  0.303
```




```
# b_i:
b_i <- edx %>%
  left_join(movie_avgs, by = 'movieId') %>%
  pull(b_i)
# Prediction:
predicted_ratings <- mu_rating + b_i
# Calculate RMSE:
rmse_result_2 <- RMSE(predicted_ratings, edx$rating)
rmse_result_2
```

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

```
# Add RSME result to the results table.
rmse_results <- bind_rows(rmse_results, tibble(Method = "Movie Effects",
                                               RMSE = rmse_result_2))
rmse_results %>% knitr::kable()
```

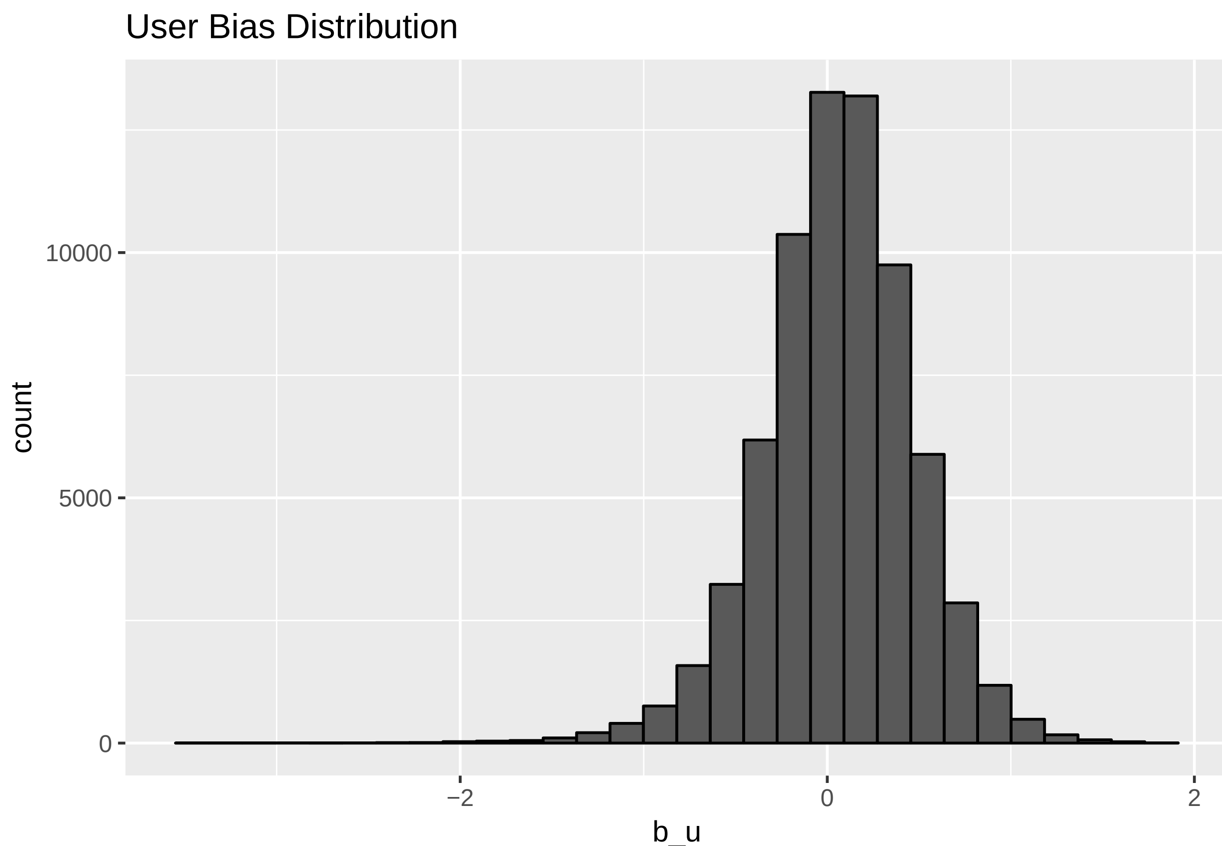
Method	RMSE
Naive Baseline	1.0612018
Movie Effects	0.9423475

The predicted RMSE for Model 2 is about **0.94235**. This is better than Model 1's RMSE, but still not enough.

Model 3: Movie Effects + User Effects (mu + b_i + b_u)

For our third model, we can consider adding user bias (b_u) into the previous model, since different users are known to rate movies differently. Incorporating user bias into our model will further help to reduce the RMSE.

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




```r
# b_i: Same as Model 2.
# b_u:
b_u <- edx %>%
  left_join(user_avgs, by = 'userId') %>%
  pull(b_u)
# Prediction:
predicted_ratings <- mu_rating + b_i + b_u
# Calculate RMSE:
rmse_result_3 <- RMSE(predicted_ratings, edx$rating)
rmse_result_3
```

```
## [1] 0.8567039
```

```r
# Add RSME result to the results table.
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie Effects + User Effects",
                                 RMSE = rmse_result_3))
rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Baseline	1.0612018
Movie Effects	0.9423475
Movie Effects + User Effects	0.8567039

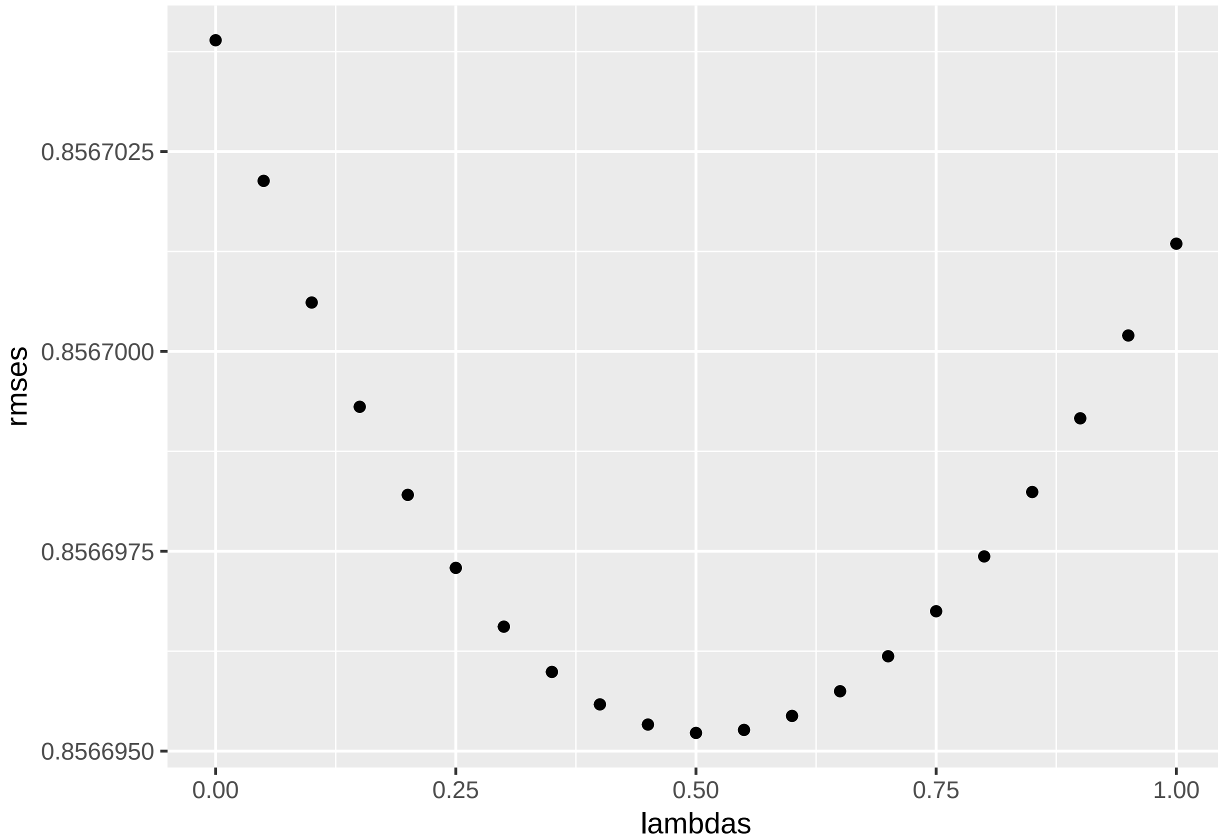
The predicted RMSE for Model 3 is about **0.85670**. This is better than Model 2's RMSE and is less than our **target RMSE of 0.87750**. However, we can still improve our model.

Model 4: Movie Effects + User Effects + Regularization (mu + b_i + b_u + Regularization)

While our previous model did great, the movie and user effects were not regularized. Those noisy estimates resulting from these unregulated effects cause large errors, which in turn increase our RMSE. So it is necessary to penalize large estimates formed using small sample sizes. To do this, we will be using the penalized least squares method.

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

```r
# Penalized least squares method:
# Use cross-validation to pick the best lambda.
lambdas <- seq(0, 1, 0.05) # Sequence of lambdas.
# Compute the predictions on the validation set using
# different lambda values.
rmses <- sapply(lambdas, function(lambda) {
  # Movie bias:
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_rating) / (n() + lambda))
  # User bias:
  b_u <- edx %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_rating) / (n() + lambda))
  # Prediction:
  predicted_ratings <- edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(prediction = mu_rating + b_i + b_u) %>% .$prediction
  return(RMSE(edx$rating, predicted_ratings))
})
# Plot:
qplot(lambdas, rmses)
```




```r
# The lambda that results in the lowest RMSE:
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 0.5
```

```r
# RMSE:
rmse_result_4 <- min(rmse_results$RMSE)
rmse_result_4
```

```
## [1] 0.8567039
```

```r
# Add RSME result to the results table.
rmse_results <- bind_rows(rmse_results,
                          tibble(Method="Movie Effects + User Effect + Regularization",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

Method	RMSE
Naive Baseline	1.0612018
Movie Effects	0.9423475
Movie Effects + User Effects	0.8567039
Movie Effects + User Effect + Regularization	0.8566952

The predicted RMSE for Model 4 is about **0.85670**. This RMSE is obtained when $\lambda = 0.5$. While this model does slightly better than Model 3, there is not much of a difference, even after regulation.

Validation

Testing Model 4 on the validation set:

```
mu_rating <- mean(validation$rating)
lambda <- 0.5
# Movie bias:
b_i <- validation %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_rating) / (n() + lambda))
# User bias:
b_u <- validation %>%
  left_join(b_i, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_rating) / (n() + lambda))
# Prediction:
predicted_ratings <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(prediction = mu_rating + b_i + b_u) %>% .$prediction
# RMSE:
validation_rmse <- RMSE(validation$rating, predicted_ratings)
validation_rmse
```

```
## [1] 0.8258487
```

The RMSE on the validation set is about **0.82585**, which is well below the **target RMSE of 0.87750**.

Results

Model 1 RMSE:

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

Model 2 RMSE:

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

Model 3 RMSE:

```
## [1] 0.8567039
```

Model 4 RMSE:

```
## [1] 0.8567039
```

Validation RMSE:

```
## [1] 0.8258487
```

Models 3 and 4 both have an **RMSE <= 0.87750**. The test on the validation set also resulted in an **RMSE <= 0.87750**. Therefore, our goal RMSE has been achieved.

Conclusion

For this project, we took a simple, iterative modelling approach. We used machine learning to create a movie recommendation system to predict how a user will rate a particular movie. We started our models with a naive baseline which made predictions based on the average rating (mu) alone. Later, we incorporated movie and user effects to make our predictions more accurate. Then, we used regularization on the movie and user effects to further decrease our RMSE. Finally, we tested Model 4 on the validation set and achieved an **RMSE of 0.82585**, which is well below our **target RMSE of 0.87750**.

Based on what we have observed from these models, it can be concluded that the *movieId* and *userId* alone have enough predictive power to determine how a user will rate a movie.

While we could have built and implemented other machine learning models such as KNN, Kmeans, random forest, and ensemble, we simply did not have sufficient hardware or computational time to build such models. Most of these algorithms could not be used due to the size of the dataset and limited time. That is why we decided to use a simpler modelling approach for this project.