HarvardX PH125.9x Data Science: Capstone MovieLens Project

Robert Walscheid

July 07, 2021

Contents

1. INTRODUCTION **3**

2. PREREQUISITES **4**

2.1 Required Packages 4

2.2 Required Data 5

3. MOVIELENS DATA FILES **6**

3.1 `ratings.dat` File 6

3.2 `movies.dat` File 7

4. DATA PREPARATION & ANALYSIS **9**

4.1 `movielens` Dataset 9

4.2 `edx` & `validation` Datasets 10

4.2.1 `edx_train` and `edx_test` Modeling Datasets 11

4.3 Dataset Analysis 12

5. MODELING **30**

MODEL 1: Predict by Overall Average Movie Rating Only (Naive Model) 30

MODEL 2: Account For The Individual Movie's Average Rating (Movie Effect) 31

MODEL 3: Account For The User's Rating (Movie + User Effect) 32

MODEL 4: Account For The Movie's Age When Rated (Movie + User + Age Effect) 34

MODEL 5: Account For The Movie's Genre (Movie + User + Genre Effect) 36

MODEL 6: Regularization of The Movie + User Effect 38

MODEL 7: Regularization of The Movie + User + Age + Genre Effect 42

MODEL 8: Using Matrix Factorization 45

6. RESULTS **49**

7. CONCLUSION **52**

1. INTRODUCTION

Recommendation systems are a highly successful application of machine learning algorithms, and are very prevalent among e-commerce businesses. This technology allows companies to create personalized recommendations of their products for their customers, based on the customer's preferences. For this project, a movie recommendation system using GroupLens's 10M version of the MovieLens dataset will be created to fulfill the "All Learners" portion of the **HarvardX: PH125.9x Data Science: Capstone** course.

GroupLens is a research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems.[1]

The MovieLens research website (run by GroupLens Research), uses "collaborative filtering" technology to generate personalized predictions for movies that registered users haven't seen yet.[2] The MovieLens 10M dataset was released 1/2009 and consists of users that were selected at random for inclusion. All users selected had rated at least 20 movies. Unlike previous MovieLens data sets, no demographic information is included. Each user is represented by an id, and no other information is provided.[3]

Developing an accurate machine learning algorithm is essential to a successful recommendation system. The goal of this project is to minimize the residual mean squared error (RMSE) between predicted ratings and actual ratings in the MovieLens 10M dataset, down to an RMSE result of less than 0.86490 using the R programming language[4]:

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

This project report will step through the process of programmatically obtaining and analyzing the MovieLens 10M data, creating and analyzing the necessary datasets, building and testing multiple machine learning algorithms, and will conclude with testing and commenting on the final model and its results. Two approaches will be used for model development: linear regression and matrix factorization. The resultant lowest RMSE for both approaches is 0.862 and 0.782, respectively.

[1] https://grouplens.org/about/what-is-grouplens/
[2] https://movielens.org/info/about
[3] https://files.grouplens.org/datasets/movielens/ml-10m-README.html
[4] https://www.r-project.org/

2. PREREQUISITES

This project and its code has minimum requirements in order to duplicate the environment necessary to successfully run in R Studio. It will be assumed that all required R packages and applications are current as of the date of this report (found on the cover page), and that the computer has Internet access.

Current RStudio Version Output:

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          4
## minor          1.0
## year           2021
## month          05
## day            18
## svn rev        80317
## language       R
## version.string R version 4.1.0 (2021-05-18)
## nickname       Camp Pontanezen
```

2.1 Required Packages

In order for the R code to run properly, the below packages are required. In the event that they are not available at runtime, the R code will download, install, and load them first.

Package Loading Code Snippet:

```
# Download and install the necessary packages for this project.
if(!require(tidyverse)) +
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) +
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) +
  install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(scales)) +
  install.packages("scales", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) +
  install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) +
  install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) +
  install.packages("gridExtra", repos = "http://cran.us.r-project.org")
if(!require(kableExtra)) +
  install.packages("kableExtra", repos = "http://cran.us.r-project.org")
if(!require(recosystem))
```

```
  install.packages("recosystem", repos = "http://cran.us.r-project.org")

# Load the necessary libraries for this project.
library(tidyverse)
library(caret)
library(data.table)
library(scales)
library(lubridate)
library(dplyr)
library(gridExtra)
library(kableExtra)
library(recosystem)
```

2.2 Required Data

The full MovieLens dataset, which consists of 27 million ratings and 1.1 million tag applications applied to 58,000 movies by 280,000 users, is considered too large for the purposes of this project. For simplicity, the "10M" dataset[5], which consists of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users, will be used.

Dataset Download Code Snippet:

```
# Download the MovieLens 10M dataset to the computer's temporary directory.
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

The .ZIP file will automatically unpack, leaving a `movies.dat` (512KB) and `ratings.dat` (252MB) file in a folder labeled `ml-10M100K` within the temporary directory on the computer, which contain the necessary records for this project.

[5]https://grouplens.org/datasets/movielens/10m

3. MOVIELENS DATA FILES

With the required R packages loaded and the MovieLens 10M data package downloaded and unpacked, the data must be ingested and analyzed. While the code provided by this course already reads the `movies.dat` and `ratings.dat` files into a tidy format, reviewing these files within a basic text editor (e.g. TextPad, Notepad, TextEdit, etc.) would be beneficial to understand their full contents for this project.

3.1 `ratings.dat` File

The `ratings.dat` file is 252MB in size and contains 10,000,054 rows of data, each row having its data delimited by two colons (::). The data is delimited into four distinct columns that will be assigned to the following names for this project:

1. *userId* (class=integer): The rater's unique identification number.
2. *movieId* (class=integer): The corresponding movie's unique record number being rated.
3. *rating* (class=numeric): The rater's rating, between 0-5 with the possibility of 0.5 intervals.
4. *timestamp* (class=integer): The rating's timestamp, in seconds, since January 1, 1970.

Sample Data (first 5 rows)
1::122::5::838985046
1::185::5::838983525
1::231::5::838983392
1::292::5::838983421
1::316::5::838983392

A basic analysis of this dataset once created is shown below.

ratings Dataset Creation Code Snippet:

```
# Create the data frame "ratings" and fill it with the 4 columns of data
# that were delimited by "::" in the ratings.dat file, labeling them
# userId, moveiId, rating, and timestamp, respectively.
#
# Note: this process could take a couple of minutes.
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
```

ratings Object Summary:

```
# Summary information of the "ratings" dataset
summary(ratings)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35741   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
```

ratings Data Structure:

```
# Structure of the "ratings" dataset
str(ratings)
```

```
## Classes 'data.table' and 'data.frame':   10000054 obs. of  4 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983
##  - attr(*, ".internal.selfref")=<externalptr>
```

3.2 `movies.dat` File

The `movies.dat` file is 512KB in size and contains 10,681 rows of data, each row having its data delimited by two colons ("::"). The data is delimited into three distinct columns that will be assigned to the following names for this project:

1. *movieId* (class=numeric): The movie's unique record, or identification, number.
2. *title* (class=character): The movie's title.
3. *genres* (class=character): The genre(s) of the movie.

Sample Data (first 5 rows)
1::Toy Story (1995)::Adventure|Animation|Children|Comedy|Fantasy
2::Jumanji (1995)::Adventure|Children|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama|Romance
5::Father of the Bride Part II (1995)::Comedy

This file will need to be parsed, and the data entered into a data frame called `movies`, containing the `movieId`, `title`, and `genres` columns as listed above. A basic analysis of this dataset once created is shown below.

movies Dataset Creation Code Snippet:

```
# Create the 'movies' data frame and fill it with the 3 columns of data
# that were delimited by '::' in the movies.dat file, labeling them moveiId,
# title, and genres, respectively:
#
# Note: this process could take a couple of minutes.
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# The movies matrix array needs to be converted to a usable dataframe,
# assigning the proper class types for this project to each column.
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
```

movies Object Summary:

```
# Summary information of the "movies" dataset
summary(movies)
```

```
##     movieId         title              genres
##  Min.   :    1   Length:10681       Length:10681
##  1st Qu.: 2755   Class :character   Class :character
##  Median : 5436   Mode  :character   Mode  :character
##  Mean   :13121
##  3rd Qu.: 8713
##  Max.   :65133
```

movies Data Structure:

```
# Structure of the "movies" dataset
str(movies)
```

```
## 'data.frame':    10681 obs. of  3 variables:
##  $ movieId: num  1 2 3 4 5 6 7 8 9 10 ...
##  $ title  : chr  "Toy Story (1995)" "Jumanji (1995)" "Grumpier Old Men (1995)" "Waiting to Exha
##  $ genres : chr  "Adventure|Animation|Children|Comedy|Fantasy" "Adventure|Children|Fantasy" "Co
```

As shown by the data structure output above, it can be seen that all 10,681 rows from the file were imported into the `movies` data frame. This will be important for when the `ratings` and `movies` data is merged together in the next section.

4. DATA PREPARATION & ANALYSIS

4.1 `movielens` Dataset

With the `ratings` and `movies` datasets now created, they will need to be joined into a single `movielens` dataset, by `movieId`, which will be used for the movie recommendation system. This new dataset is a data frame that consists of 10,000,054 rows and 6 columns (`userId`, `movieId`, `rating`, `timestamp`, `title`, and `genres`). A basic analysis of this dataset once created is shown below.

movielens Dataset Creation Code Snippet:

```
# Create the 'movielens' data frame by joining the 'ratings' and 'movies'
# datasets by 'movieId'.
movielens <- left_join(ratings, movies, by = "movieId")
```

movielens Object Summary:

```
# Summary information of the "movielens" dataset
summary(movielens)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35741   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:10000054    Length:10000054
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

movielens Data Structure:

```
# Structure of the "movielens" dataset
str(movielens)
```

```
## Classes 'data.table' and 'data.frame':   10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thril
##  - attr(*, ".internal.selfref")=<externalptr>
```

4.2 `edx` & `validation` Datasets

The `movielens` dataset will then be split into a training subset called `edx` and a test subset called `temp` that will contain a sample of 90% (9,000,047 rows) and 10% (1,000,007 rows) of the ratings data, respectively.

edx and *temp* Subset Creation Code Snippet:

```
# The training dataset, "edx", and validation test dataset, "temp", will be created
# from a 90%/10% split of the MovieLens data, respectively.
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,]
```

Because the data is split into two subsets, one that will validate the other, it is important to ensure that the unique identifiers (`movieId` and `userId` in this case) in the final hold-out validation data exists in the `edx` training data, otherwise the calculated RMSE could be skewed.

This final hold-out test set (which will aptly be be named `validation`), used to simulate the `edx` data, will be created by joining together the data from the corresponding `movieId` and `userId` records from the `edx` and `temp` data frames. The join results in the `validation` data frame consisting of 999,999 rows of data.

validation Dataset Creation Code Snippet:

```
# Create validation dataset, ensuring the userId and movieId combinations are
# also in the edx set.
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

validation Object Summary:

```
# Summary information for the "validation" dataset.
summary(validation)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:999999      Length:999999
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Once the `validation` dataset is created, any rows that were removed from `validation` need to be added back to `edx` to ensure 100% of the original `movielens` data (10,000,054 rows) is accounted for between the training and validation data frames. Looking at the difference in length in the summary output of `temp` (1,000,007) and `validation` (999,999), it can be observed that 8 rows will be added back to the `edx` data frame (totaling 9,000,055 rows). Once this data is added back, the `temp` dataset will be deleted during a cleanup process of all unnecessary variables.

Re-balance Data Code Snippet:

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

4.2.1 `edx_train` and `edx_test` Modeling Datasets

As a requirement for this project, the final hold-out test dataset (`validation`) is to only be used for evaluating and testing the RMSE of the final algorithm, and not during model development. Since 10% of the entire `movielens` dataset has already been allocated to the final hold-out (`validation`), the `edx` dataset itself will be split into a 90% training data subset (`edx_train`) and 10% test subset (`edx_test`), with 8,100,065 and 899,990 rows, respectively, to be used to build and test algorithms in the Modeling section of this report. This will bring the overall training/testing data split to approximately 80% (training)/20% (testing) for the project after both splits: 90% `edx` (with `edx` split into another 90% `edx_train`/10% `edx_test`)/10% `validation` (final hold-out), which is a commonly-used split percentage for larger datasets.

The analysis and charting in the next section (Section 4.3) will use the larger `edx` dataset, while the modeling section (Section 5) will use the smaller `edx_train`/`edx_test` datasets for model development.

edx train and *edx temp* Subset Creation Code Snippet:

```
# The training dataset, "edx_train", and validation test dataset, "edx_temp", will be
# created from a 90%/10% split of the "edx" dataset, respectively.
set.seed(1, sample.kind="Rounding")
edx_test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-edx_test_index,]
edx_temp <- edx[edx_test_index,]
```

edx test Dataset Creation Code Snippet:

```
# Create edx_test dataset, ensuring the userId and movieId combinations are also
# in edx_train set.
edx_test <- edx_temp %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
```

edx test Object Summary:

```
# Summary information for the "edx_test" dataset.
summary(edx_test)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18105   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35761   Median : 1834   Median :4.000   Median :1.036e+09
##  Mean   :35868   Mean   : 4133   Mean   :3.513   Mean   :1.033e+09
##  3rd Qu.:53598   3rd Qu.: 3638   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65130   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:899990      Length:899990
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Once the `edx_test` dataset is created, any rows that were removed from `edx_test` need to be added back to `edx_train` to ensure 100% of the original `edx` data (9,000,055 rows) is accounted for. Looking at the difference in length in the summary output of `edx_temp` (8,100,048) and `edx_test` (899,990), it can be observed that 17 rows will be added back to the `edx_train` data frame (totaling 8,100,065 rows). Once this data is added back, the `edx_temp` dataset will be deleted during a cleanup process of all remaining unnecessary variables.

Re-balance Data Code Snippet:

```
# Add rows removed from edx_test set back into the edx_train set
edx_removed <- anti_join(edx_temp, edx_test)
edx_train <- rbind(edx_train, edx_removed)
```

4.3 Dataset Analysis

Reviewing the datasets in raw format, along with creating correlative visual aids is necessary when comparing and contrasting the data. While simple movie and user data analysis may be made with the dataset as-is, other variables will need to be created to dig deeper. To get a general overview of the `edx` data and its structure, the below output can be studied. In the below `edx` Object Summary, the average (mean) rating for all movies is shown as 3.512:

edx Object Summary:

```
# Summary information for the "edx" dataset.
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
##
##
##
```

```
# Get a quick glimpse of the data in edx.
glimpse(edx, width=80)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

```
# Number of unique users in the edx userId column:
n_distinct(edx$userId)  #69,878

# Number of unique movies in the edx movieId column:
n_distinct(edx$movieId)  #10,677

# Number of unique titles in the edx title column.
# (There might be movies with different movieIds that have the same title)
n_distinct(edx$title)  #10,676
```

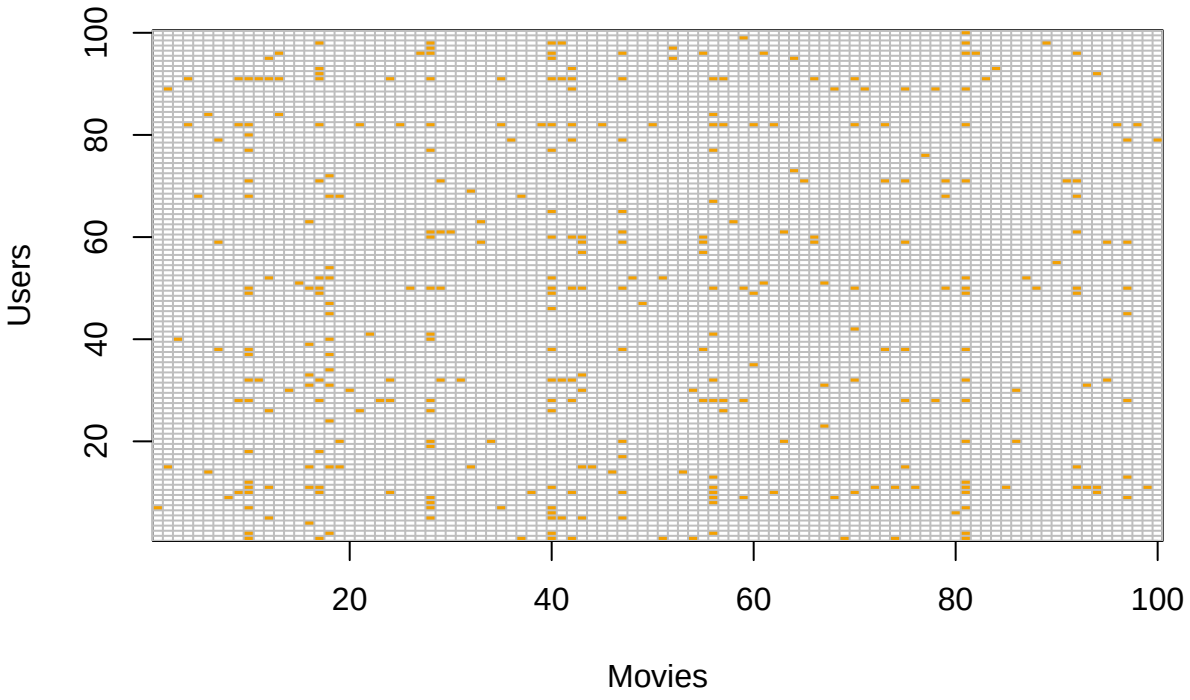
Seeing the results of the `movieId` count (10,677) as being greater than the `title` count (10,676) can be explained by there being two different `movieId`s that share the same title.

Given that there are 69,878 unique user IDs and 10,677 unique movie IDs, simple math shows: 69,878 users * 10,677 movies = a 746,087,406 total ratings potential if all users rated every movie. The summary output of the `edx` dataset above shows that there are only 9,000,055 ratings, which is approximately 12.1% of the total possible. The unknown variations of which users rated which movies provides insight not only into how sparse the dataset is, but also how challenging the creation of an accurate recommendation system will be.

The MovieLens data files downloaded in Section 2 do not provide any other information regarding the movie raters assigned to each `userId` (such as age, gender, demographic, etc.) to aid in building a better

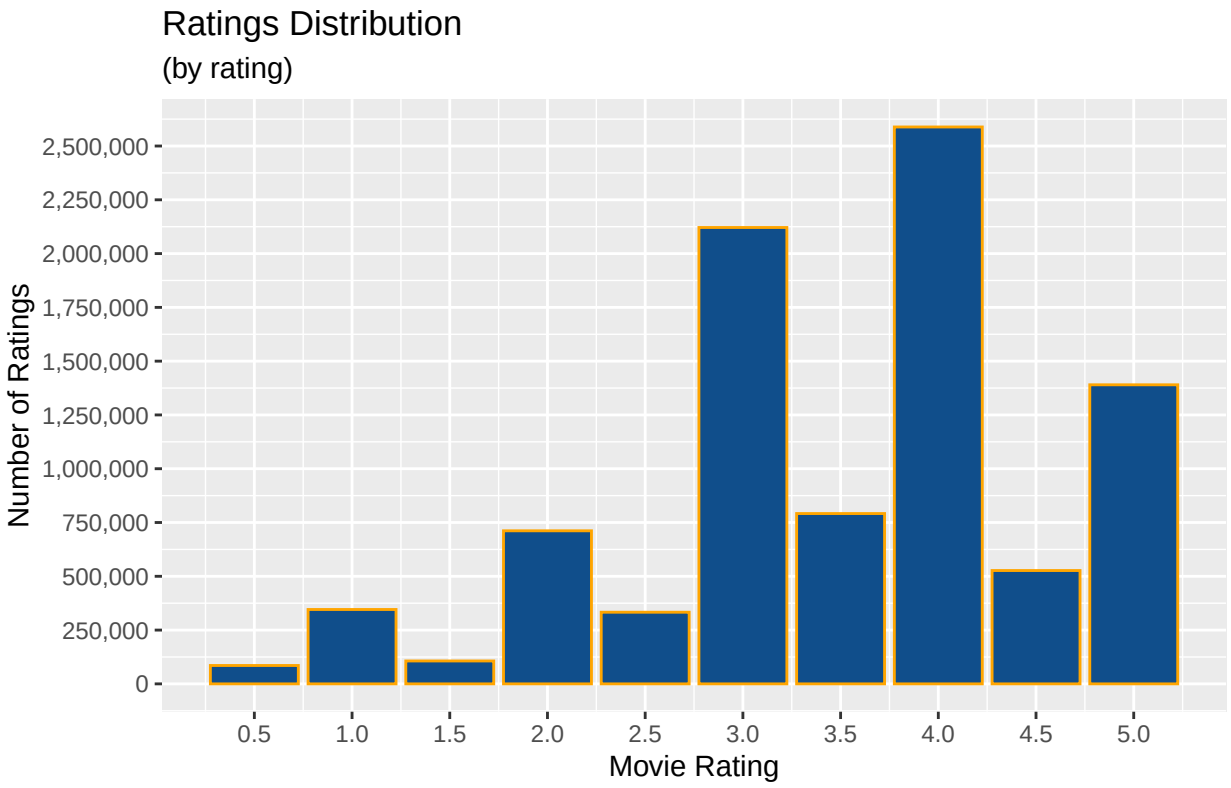
model. It also does not provide details as to whether the `userId` only has one owner, or whether multiple `userId`'s may have been created by a single person in the sample.

When comparing user ratings, a simple 100-user sample can be used to create a sparse matrix to show how random the data is. Each orange square shows a movie/user combination that has a rating in the training dataset:



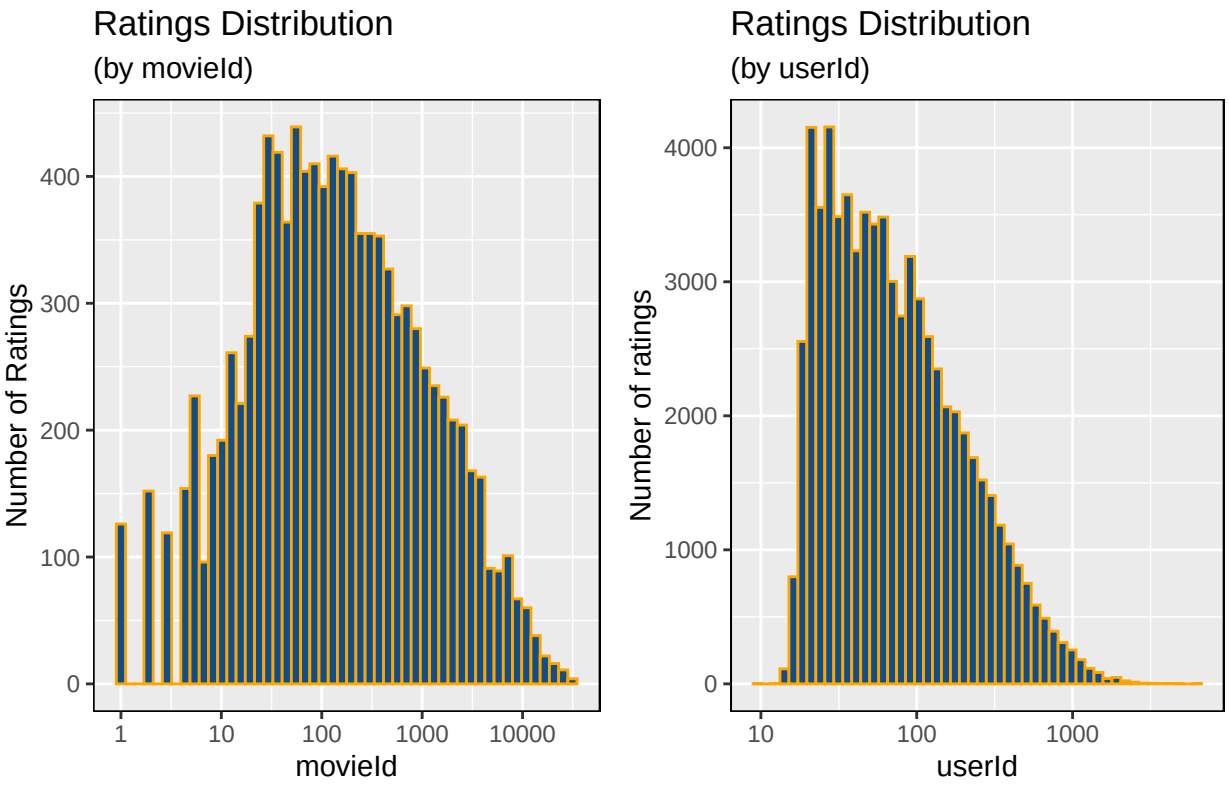

```
## integer(0)
```

The simple distribution of movie ratings can be seen in **Figure 4.3.1** below. The movie ratings ranged between 0.5 to 5, at 0.5 increments. Most of the ratings were 4's and 3's, respectively, and whole number ratings were given more often by users than half-ratings (***X***.5 ratings):



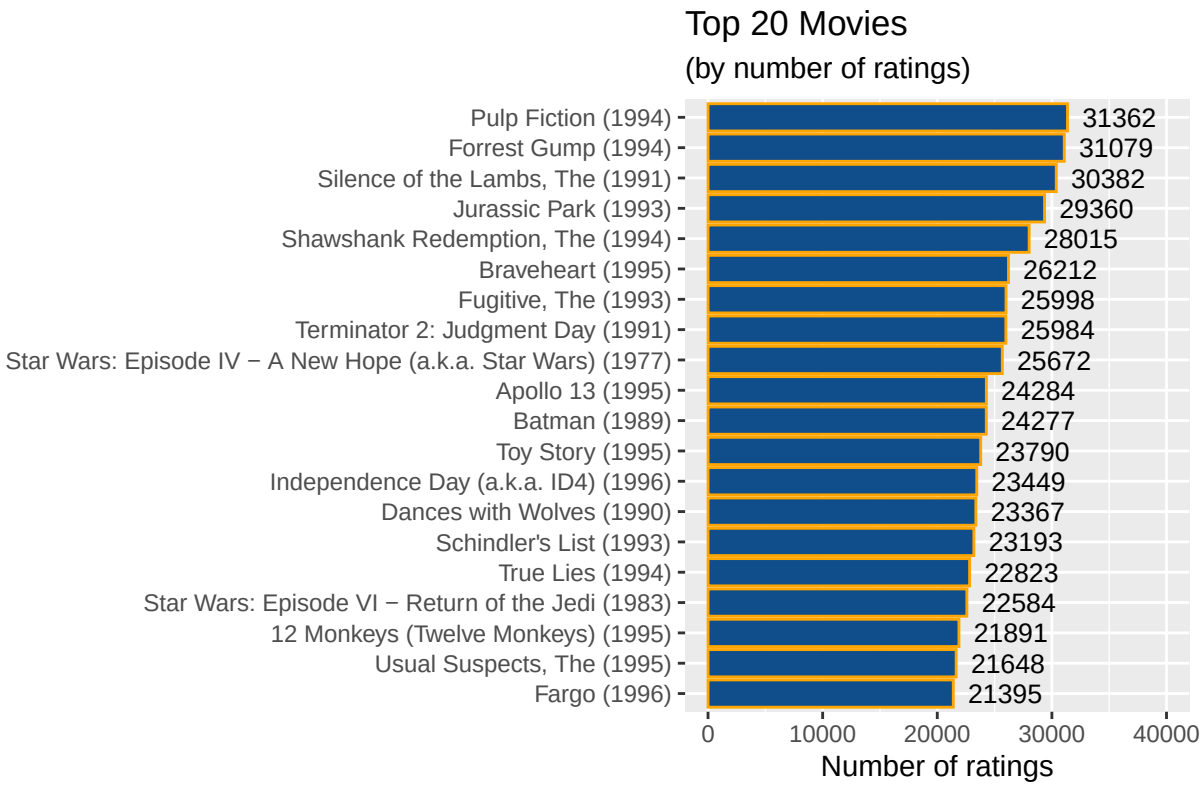
Source Data: edx
Figure 4.3.1

Viewing the ratings distribution by both `moveId` and `userId` in **Figure 4.3.2** below (using `scale_x_log10`) shows that some movies were rated more than others and some users rated more movies than others.



Source Data: edx
Figure 4.3.2

Figure 4.3.3 below shows the top 20 movies by the quantity of user ratings. While 17 out of 20 (85%) of the top 20 movies were released in the 1990's, the chart alone does not show whether these movies were actually rated around the year they were released, or any other relevant details.

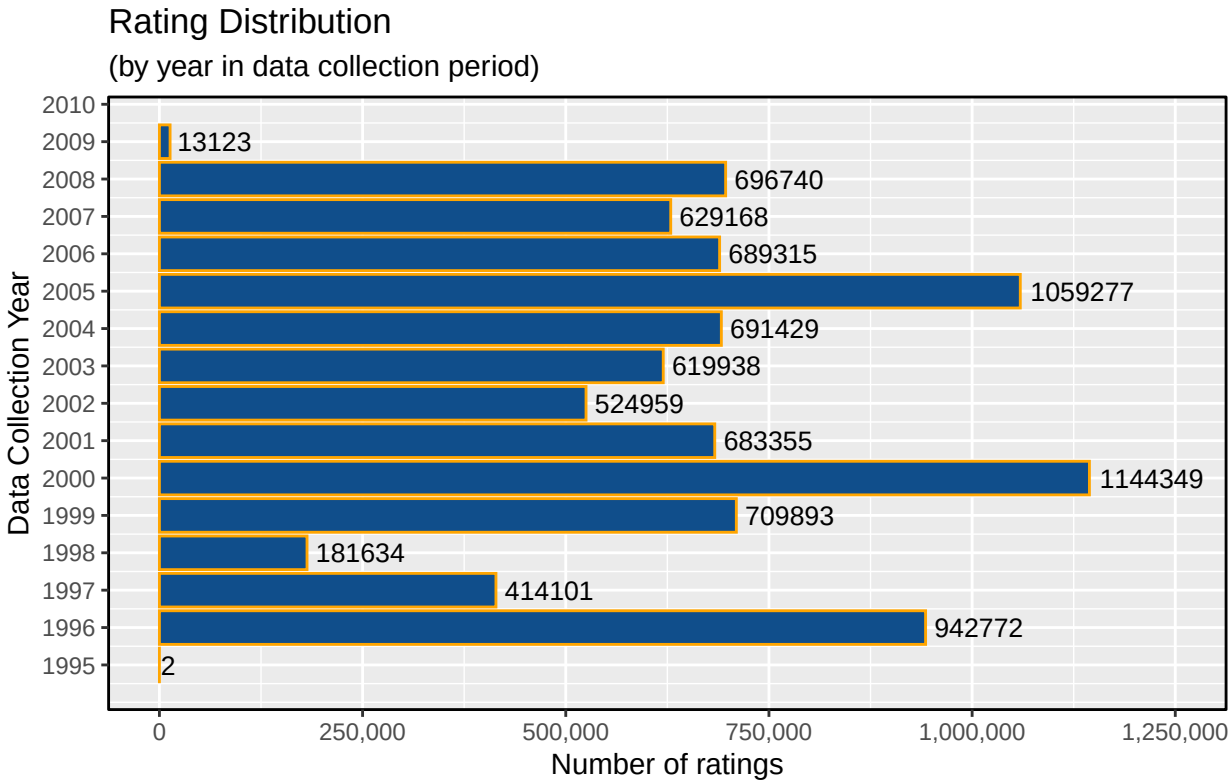


Source Data: edx
Figure 4.3.3

Taking the difference between the first and last rating's timestamp shows that the ratings were collected over a 14 year period:

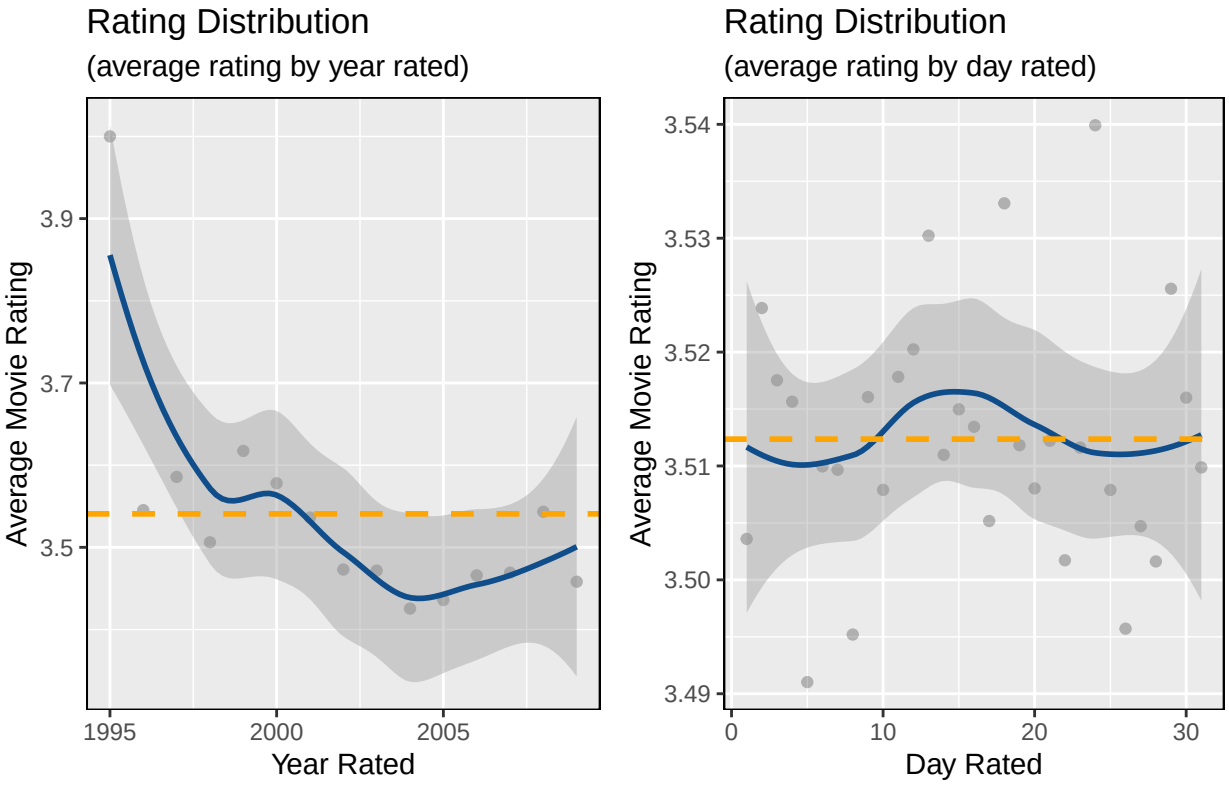
First Rating Date	Last Rating Date	Rating Period
1995-01-09	2009-01-05	**441479727s (~13.99 years)**

Viewing how the ratings were distributed each year over that 14 year period (**Figure 4.3.4** below) does not provide much useful information by itself:



Source Data: edx
Figure 4.3.4

The first (1995) and last (2008) years during the ratings collection period contain very few ratings, and will need to be taken into account when reviewing the average movie ratings given over that 14 year period in **Figure 4.3.5** below. Since there were only 2 ratings in 1995, it is not surprising that the average rating for that year is skewing the smooth line. Changing from year to day shows a much flatter smooth line, which means the date the movie is rated only has a small impact on the rating itself.



Source Data: edx
Figure 4.3.5

Comparing the average movie ratings for the top 20 most-rated movies to the overall `edx` dataset mean of 3.5124652 shows that the most-rated movies were also rated higher than average, which is not surprising that good movies are rated higher more often:

Movie Title	# of Ratings	Average Rating	+/- edx Mean
Pulp Fiction (1994)	31362	4.154789	**0.6423240**
Forrest Gump (1994)	31079	4.012822	**0.5003570**
Silence of the Lambs, The (1991)	30382	4.204101	**0.6916359**
Jurassic Park (1993)	29360	3.663522	**0.1510566**
Shawshank Redemption, The (1994)	28015	4.455131	**0.9426660**
Braveheart (1995)	26212	4.081852	**0.5693866**
Fugitive, The (1993)	25998	4.009155	**0.4966893**
Terminator 2: Judgment Day (1991)	25984	3.927859	**0.4153943**
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672	4.221311	**0.7088460**
Apollo 13 (1995)	24284	3.885789	**0.3733238**
Batman (1989)	24277	3.386292	**-0.1261736**
Toy Story (1995)	23790	3.927638	**0.4151725**
Independence Day (a.k.a. ID4) (1996)	23449	3.376903	**-0.1355621**
Dances with Wolves (1990)	23367	3.742628	**0.2301633**
Schindler's List (1993)	23193	4.363493	**0.8510281**
True Lies (1994)	22823	3.500285	**-0.0121804**
Star Wars: Episode VI - Return of the Jedi (1983)	22584	3.996436	**0.4839703**
12 Monkeys (Twelve Monkeys) (1995)	21891	3.874743	**0.3622778**
Usual Suspects, The (1995)	21648	4.365854	**0.8533885**
Fargo (1996)	21395	4.134821	**0.6223560**

For further age-based comparisons in this section, it would be more efficient from a computing-resource perspective to extract the following variables into a new data frame called `analysis_age_data` (a copy of the `edx` dataset), using the default `timestamp` column and splitting the name of the movie and the release year from the default `title` column:

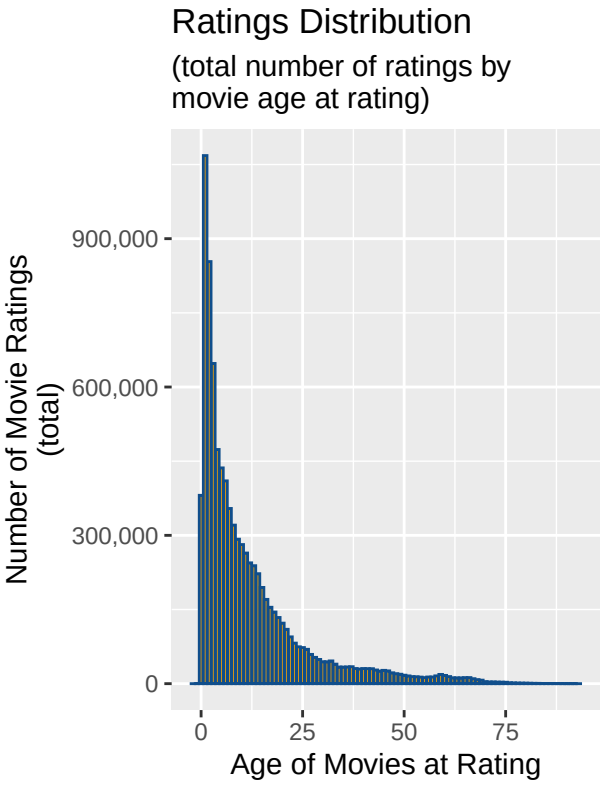
1. `age_at_rating` = The age of each movie when they were rated. (year rated - year released)
2. `ratings_per_age` = The number of ratings for each movie of a particular age.
3. `movies_per_age` = The number of movies that are a particular age.
4. `avg_ratings_per_age` = The average rating of all movies of a given age when rated.
5. `ratings_per_movie` = The number of ratings given for each movie.

Age-based Data Extraction Code Snippet:

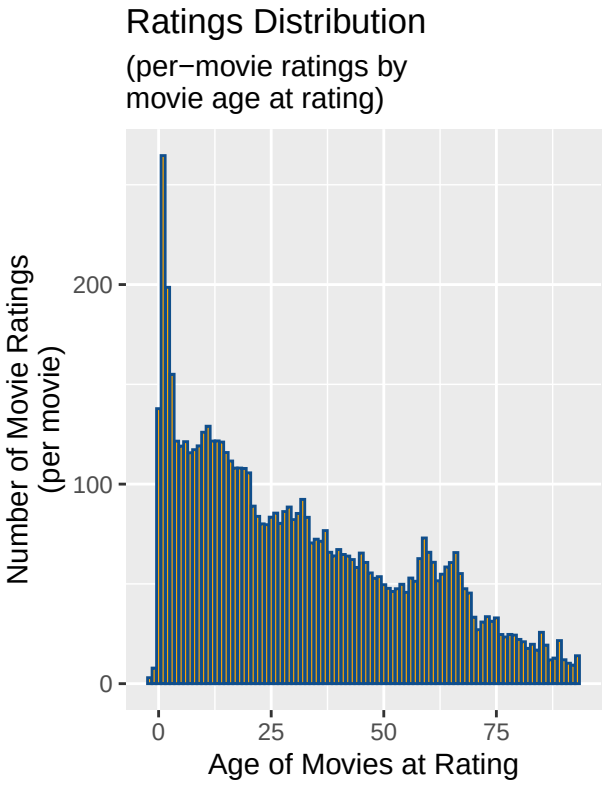
```
# Build time/age-based analysis dataset with year the movies were rated, movie title
# without the release year, the movie release year, age of movie at time of rating,
# number of ratings by movie age, number of movies by age, average ratings of a movie
# based on age, and number of ratings per movie.
analysis_age_data <- edx %>% mutate(year_rated = year(as_datetime(timestamp)),
                            title_without_year = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1_
  separate(title_without_year,c("title_without_year","year_released"),"__") %>%
  select(-timestamp) %>%
  mutate(age_at_rating = as.numeric(year_rated)-as.numeric(year_released)) %>%
  group_by(age_at_rating) %>%
  summarize(ratings_per_age=n(),
```

```
            movies_per_age=n_distinct(movieId),
            avg_ratings_per_age=mean(rating),
            ratings_per_movie=n()/n_distinct(movieId))
```

Looking at the total number movie ratings by movie age when rated, **Figure 4.3.6** (below left) shows that newer movies were rated more frequently within the first 3 years of their release, with a sharp decline over the following 20 years. Graphing the number of ratings given only to movies within the movie's age group (**Figure 4.3.7** below right) continues to show that newer movies were still more frequently rated, but normalizing by the per-movie age at rating greatly slows the rate of decline.

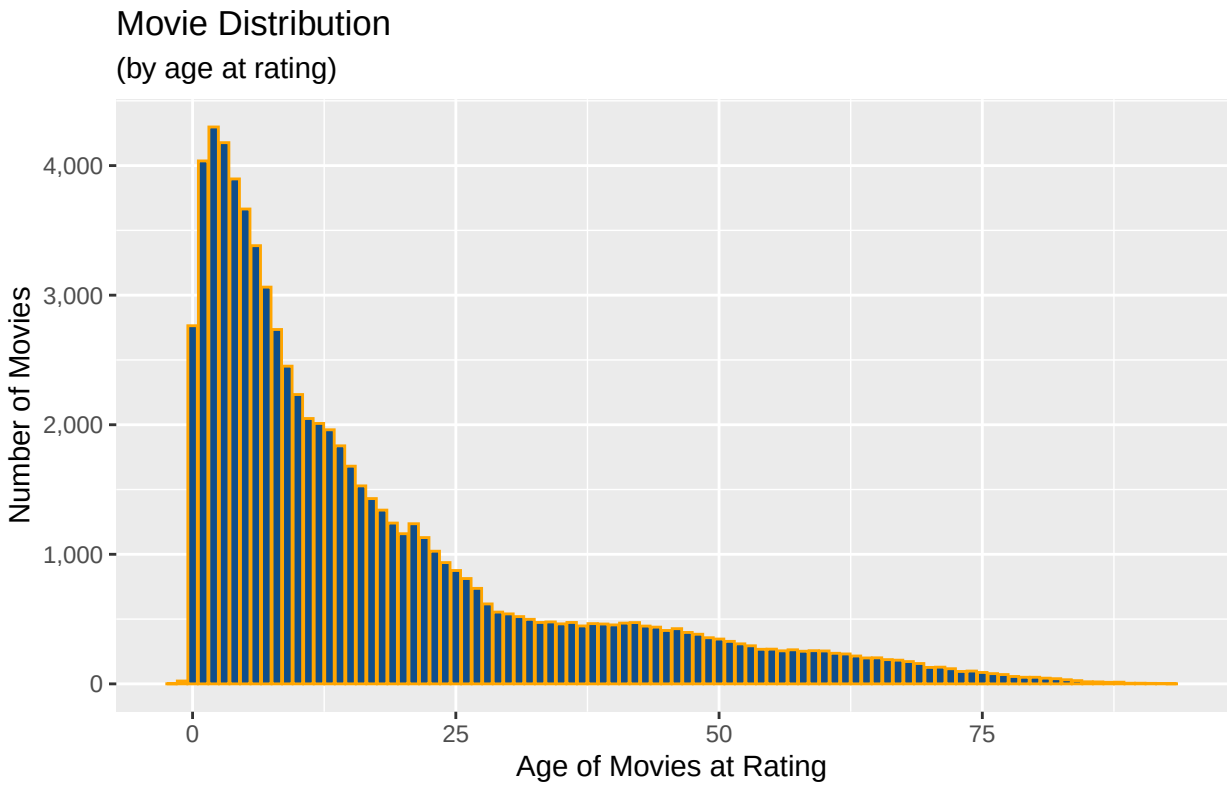


Source Data: edx
Figure 4.3.6



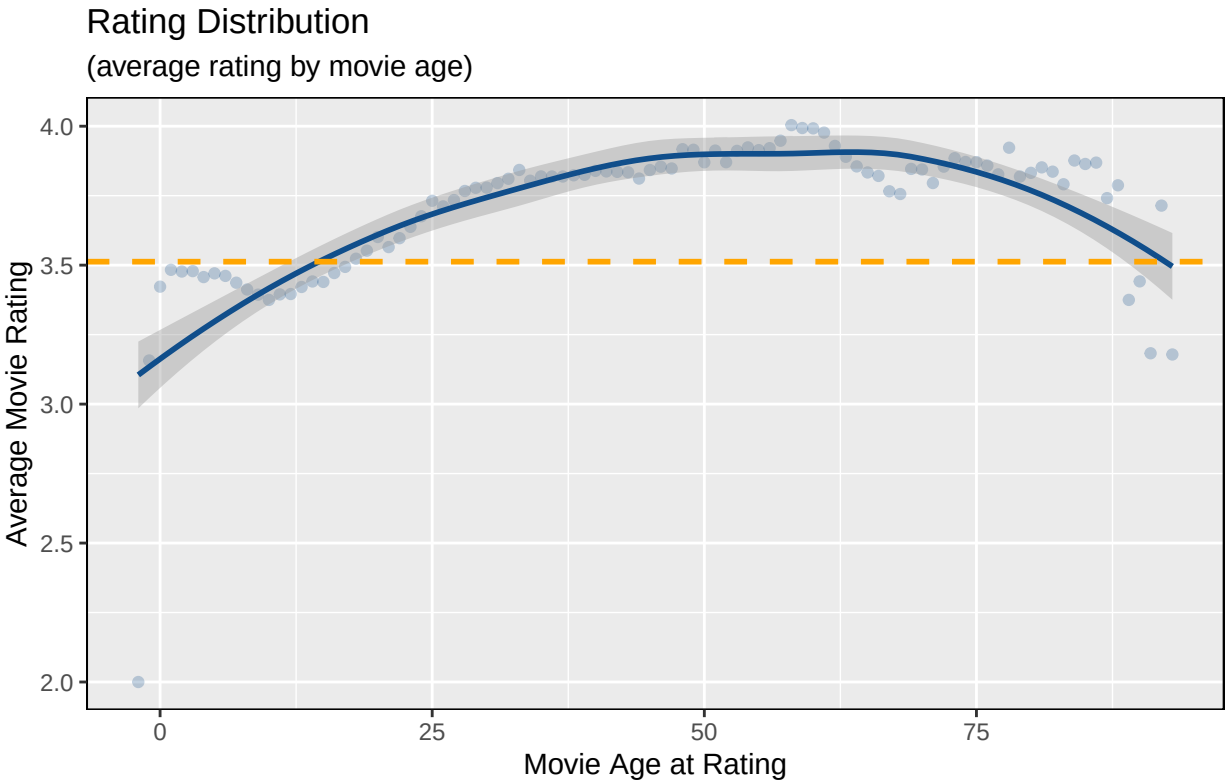
Source Data: edx
Figure 4.3.7

There could be several contributing factors to this, such as socioeconomic events allowing for greater recreational spending on movies, and/or that this MovieLens dataset was only collected over a 14 year period starting in 1995 (leading to why 85% of the top 20 movies were released in the 1990's), etc.



Source Data: edx
Figure 4.3.8

When comparing both **Figure 4.3.6** (total ratings by movie age) and **Figure 4.3.8** (number of movies by age) above, it appears that most ratings were for movies between 0-25 years old when they were rated. Plotting the average movie rating by movie age when rated (**Figure 4.3.9** below) shows that users gave a better rating to older movies vs. newer movies:



Source Data: edx
Figure 4.3.9

Plotting the average movie rating by movie release year (**Figure 4.3.10** below) shows a similar trend in user preference for older movies vs. newer movies.

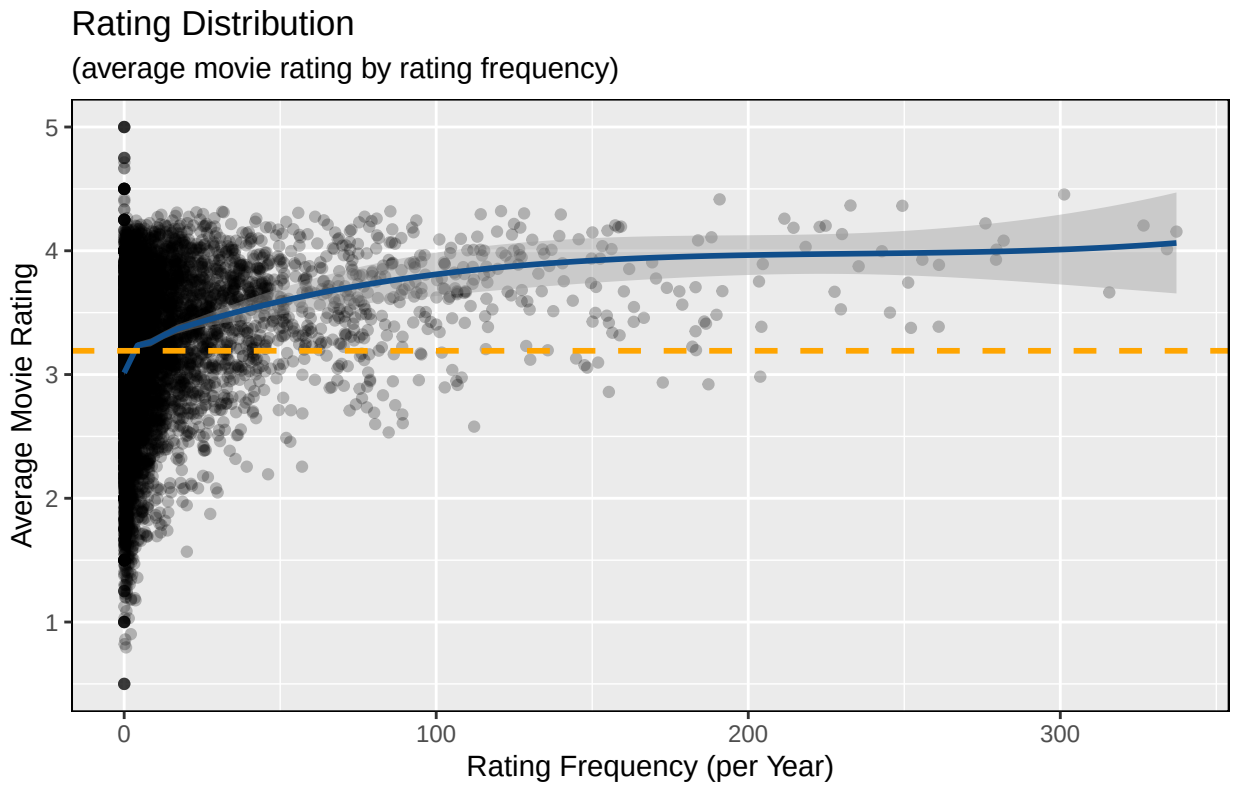
Source Data: edx
Figure 4.3.10

The significantly lower number of ratings given to older movies needs to be taken into account when viewing both **Figure 4.3.9** (average ratings based on movie age at rating) and **Figure 4.3.10** (average ratings based on the year the movie was released) above.

Gathering the movie release years shows 93-year period between the first and last movie in this dataset. Since the data collection period was over a 14 year period (starting in 1995), there is the possibility that not all users were able to rate all of the newer movies.

First Movie Release Year	Last Movie Release Year
1915	2008

When looking at the frequency in which each movie was rated over time, **Figure 4.3.11** below shows that movies rated more often each year were also often rated higher. This can easily be explained since more people watched popular movies.



Source Data: edx
Figure 4.3.11

Now, to begin analyzing more genre-related data, the `movielens` data structure output in section 4.1 showed that a movie being rated had the possibility of being assigned one or more genres. To begin charting genre-based comparisons, the `genres` data for each movie will be separated from their pipe (`|`) delimiter and stored in a column. The following variables will additionally need to be extracted into a new data frame (copied from `edx`) called `analysis_genre_data`, which will be referenced for genre-based data analysis later in this section.

1. `ratings_per_genre` = The number of ratings given to a particular genre.
2. `movies_per_genre` = The number of movies that are assigned to a particular genre.
3. `users_per_genre` = The number of users that rated a movie of a particular genre.
4. `avg_ratings_per_genre` = The average rating of movies assigned a particular genre.
5. `ratings_per_movie` = The number of ratings given for each movie.

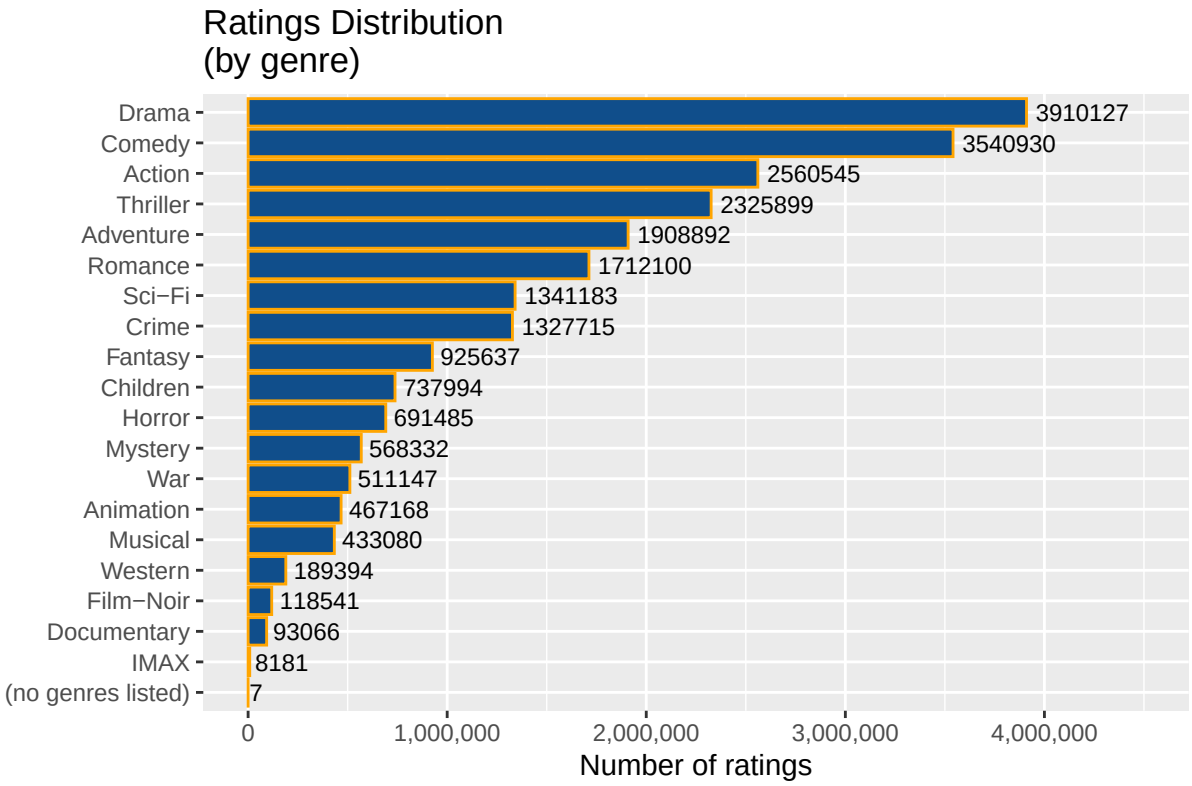
***Note: Preliminary testing shows the presence of a* `(no genres listed)` *entry, which this code will ignore (further details provided later in this section).*

Genre-based Data Extraction Code Snippet:

```
# Build Genre-based analysis dataset with number of ratings/genre, number of
# movies/genre, number of times users rated a particular genre, average rating
# of each genre, and the number of ratings/movie.
```

```
analysis_genre_data <- edx %>% separate_rows(genres,sep = "\\|") %>%
  mutate(value=1) %>%
  group_by(genres) %>%
  summarize(ratings_per_genre=n(),
            movies_per_genre=n_distinct(movieId),
            users_per_genre=n_distinct(userId),
            avg_ratings_per_genre=mean(rating),
            ratings_per_movie=n()/n_distinct(movieId)) %>%
  filter(genres != "(no genres listed)") %>%
  arrange(desc(ratings_per_genre))
```

When separating the genres, a total of 20 different categories are found. One such genre category is labeled `(no genres listed)`, which accounts for 7 movie ratings (see **Figure 4.3.12** below), and is for movies that are not assigned to a specific genre.

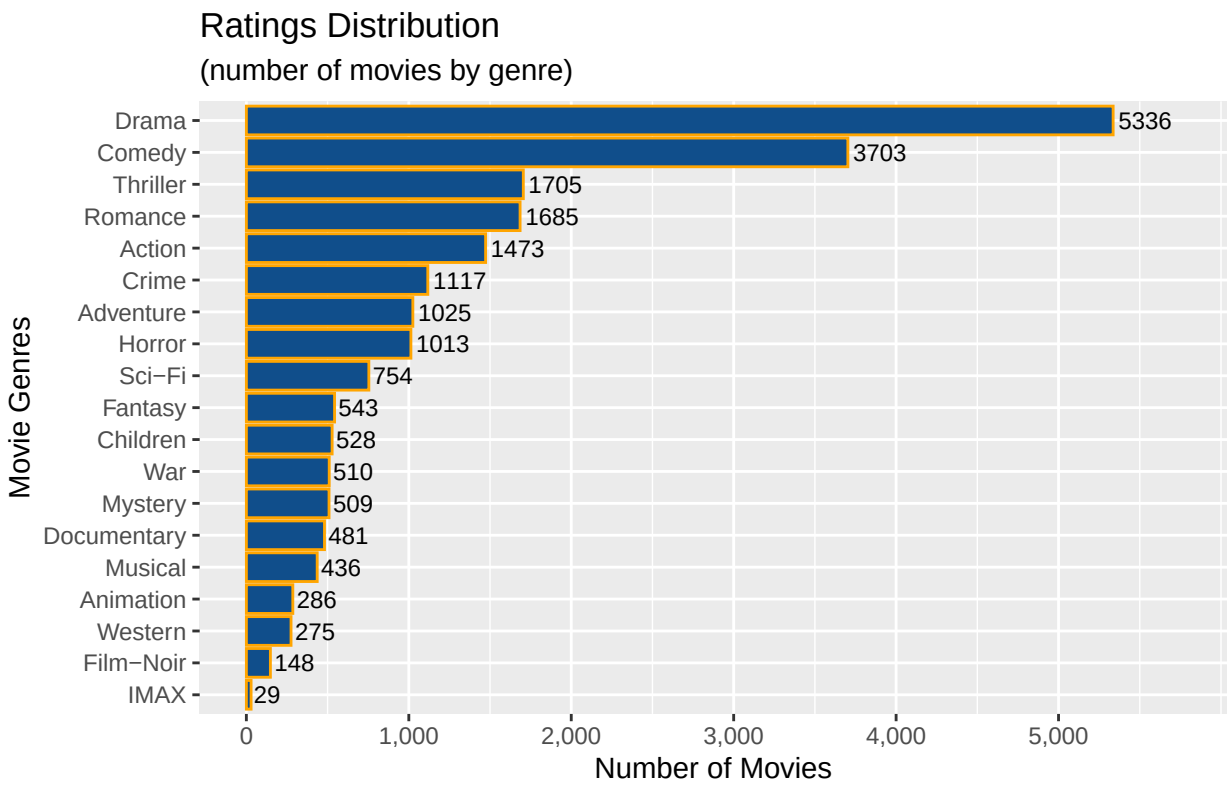


Source Data: edx
Figure 4.3.12

Extracting the 7 ratings that were for movies in the `(no genres listed)` category yields only a single movie, "Pull My Daisy (1958)", with `moveId` 8606.

Movie ID	Movie Title	Genre
8606	Pull My Daisy (1958)	(no genres listed)
8606	Pull My Daisy (1958)	(no genres listed)
8606	Pull My Daisy (1958)	(no genres listed)
8606	Pull My Daisy (1958)	(no genres listed)
8606	Pull My Daisy (1958)	(no genres listed)
8606	Pull My Daisy (1958)	(no genres listed)
8606	Pull My Daisy (1958)	(no genres listed)

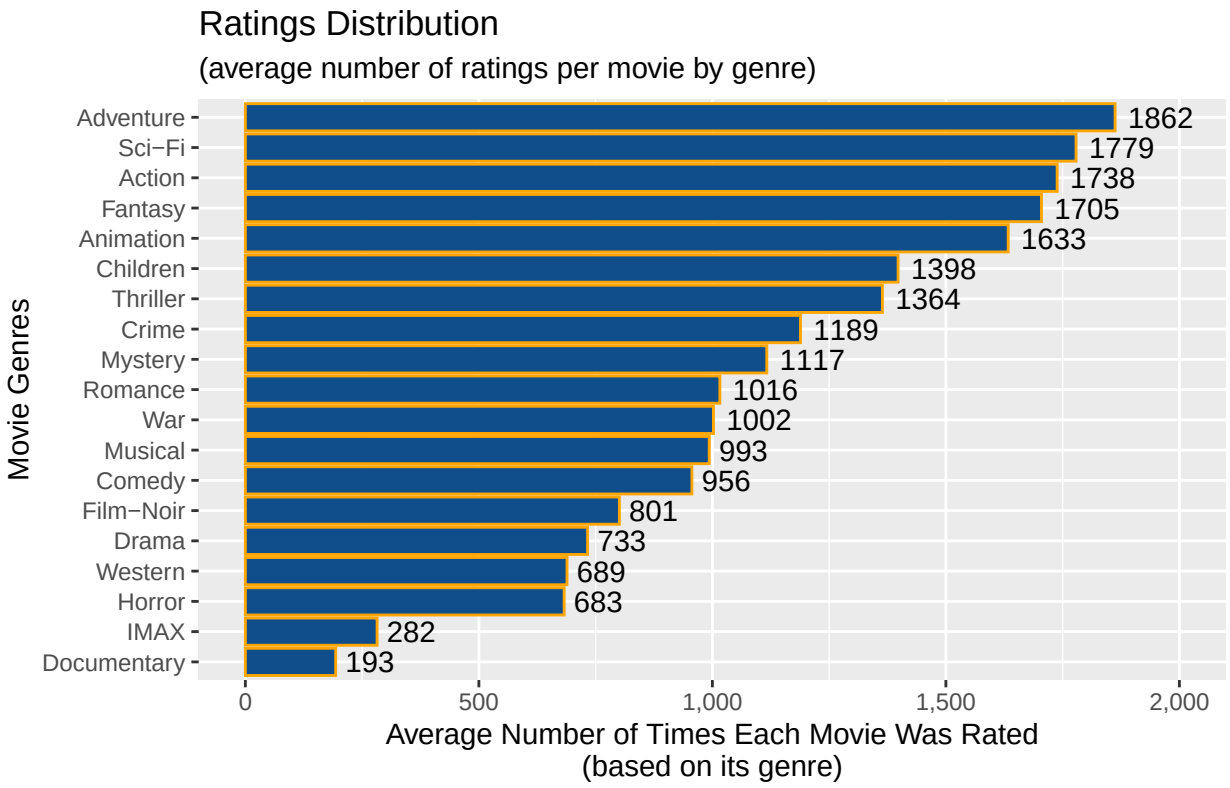
Given that there is only a single movie with the `(no genre listed)` label, any model considering genres as a data point in this project will ignore this category, leaving 19 total genre categories:



Source Data: edx
Figure 4.3.13

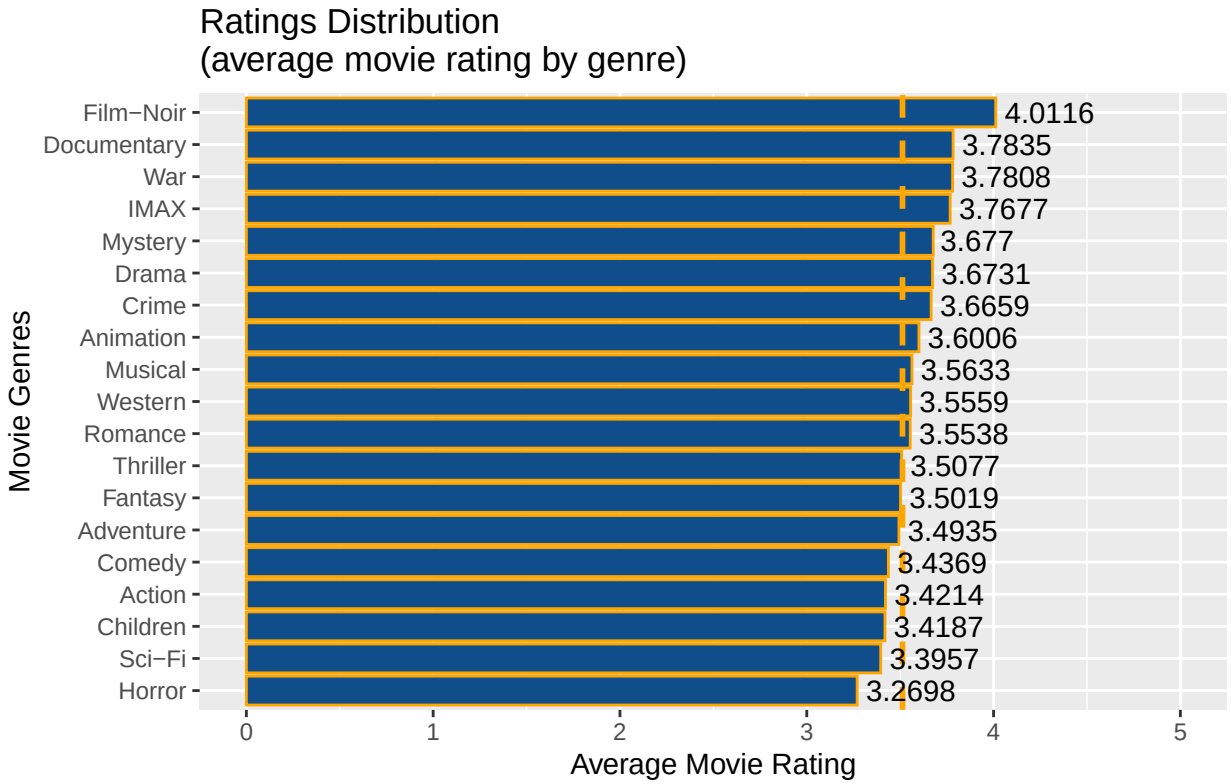
Looking at the distribution of movies by genre in **Figure 4.3.13** above, 49.98% of all movies were in the Drama genre (5,336 Drama/10,677 Total), and 34.68% were in the Comedy genre (3,703 Comedy/10,677 Total). Since each movie may be in more than one genre category, these most likely share several movies with the same `movieId`.

While there was a significant number of movies that were in the Drama and Comedy genre category, **Figure 4.3.14** below shows that movies in either of those categories were among the fewest to be rated on average.



Source Data: edx
Figure 4.3.14

When comparing the average ratings of movies by their genre (**Figure 4.3.15** below), the highest-rated (on average) genres were those that contained the fewest movies (see **Figure 4.3.13**) as well as were rated the fewest times on average (see **Figure 4.3.14**). While **Figure 4.3.15** definitely shows that genre has an effect on the rating, the amount of impact appears to be slight.



Source Data: edx
Figure 4.3.15

5. MODELING

Throughout the data analysis performed above, some data point comparisons show the possibility of higher correlations while others appear they would have little impact to the overall RMSE value, and thus the recommendation model. To determine the impact of each individual bias, various models will be tested, starting from the most basic baseline model through the addition of individual biases, bias combinations, regularization, and matrix factorization. Each resultant RMSE will be summarized in a list with previous results as to make it easy to compare all results and determine which model performs the best (having the lowest RMSE).

MODEL 1: Predict by Overall Average Movie Rating Only (Naive Model)

The first model is the simplest, as it assumes the same rating for all movies and all users, with all the differences explained by random variation. All variations in this model are viewed as equally weighted errors.

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

Where $\boldsymbol{Y}$ is the rating provided by user $\boldsymbol{u}$ for movie $\boldsymbol{i}$, μ is the overall average of all ratings, and ε represents independent errors sampled from the same distribution centered at zero.

The μ value in this case is determined as `mu_edx_train <- mean(edx_train$rating)`, which represents the true rating for all movies and users, and comes to 3.5124556:

```
# Model 1: mean (mu) calculation
mu_edx_train <- mean(edx_train$rating)
mu_edx_train
```

```
## [1] 3.512456
```

```
# Test the model by calculating the RMSE using the edx_test dataset.
model1_rmse <- RMSE(edx_test$rating, mu_edx_train)
model1_rmse
```

```
## [1] 1.060054
```

Using the `edx_test` data, the RMSE for this model is calculated to be 1.0600537.

Model	RMSE
Model 1: Average Rating Only (Naive Model)	**1.060054**

MODEL 2: Account For The Individual Movie's Average Rating (Movie Effect)

In order to improve upon Model 1, other variables need to be considered. As seen earlier in section 4.3, the average rating for each movie varies, adding a bias that we can factor into a model:

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

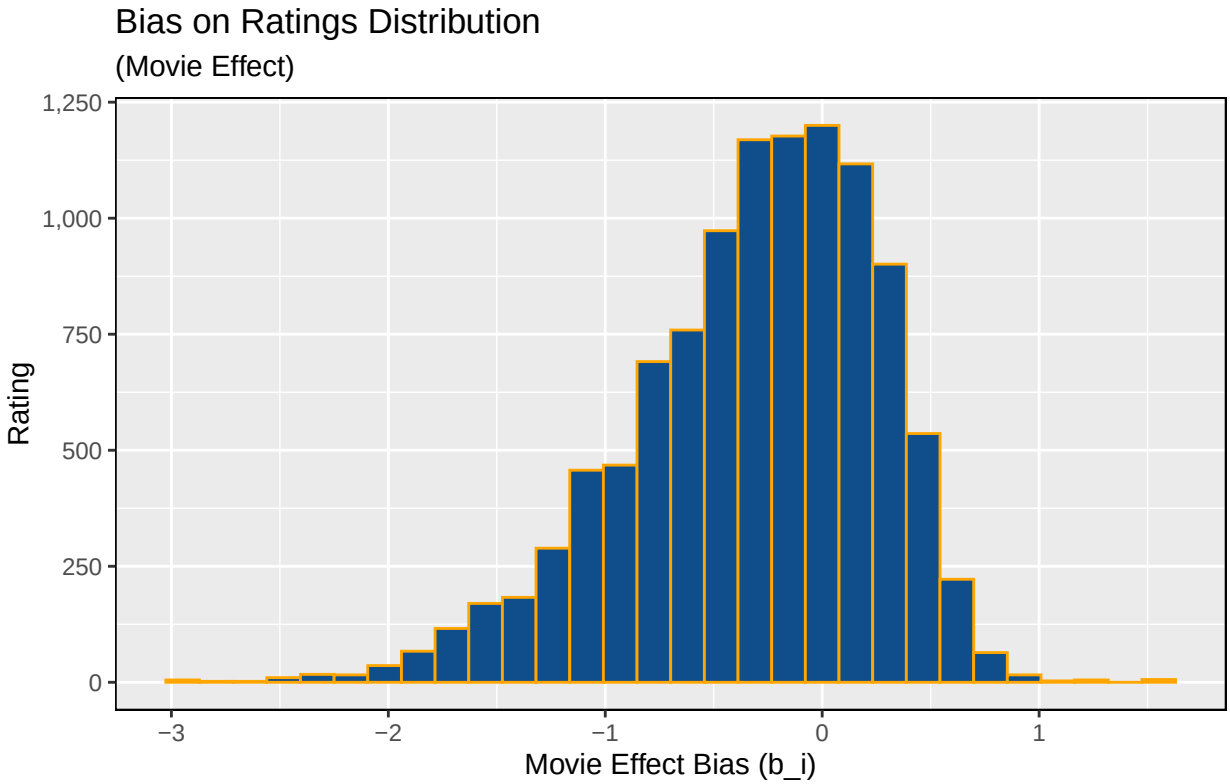
Where ***Y*** is the rating provided by user ***u*** for movie ***i***, μ remains the overall average of all ratings, b_i is the bias (***b***) added for each movie's (***i***) average rating, and ε represents independent errors sampled from the same distribution centered at zero.

```
# Mean (mu) calculation
mu_edx_train <- mean(edx_train$rating)

# Each individual movie's mean rating calculation
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_edx_train))

# Predict the ratings against the edx_test dataset.
predicted_ratings_m2 <- mu_edx_train + edx_test %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)

# Test the "Movie Effect" model by calculating the RMSE using the edx_test dataset.
model2_rmse <- RMSE(predicted_ratings_m2, edx_test$rating)
model2_rmse
```

```
## [1] 0.9429615
```

Model	RMSE
Model 1: Average Rating Only (Naive Model)	**1.0600537**
Model 2: Movie Effect	**0.9429615**

Adding the "Movie Effect" bias improved the prediction, as shown by the lower calculated RMSE of 0.9429615. Plotting the bias (**Figure 5.2.1** below) shows the slightly negative effect the movie's individual rating had on the ratings distribution. This takes into account the difference of the individual movie's average rating and the overall average rating of all movies, allowing for the prediction adjustment of μ by b_i.



Source Data: edx_train
Figure 5.2.1

MODEL 3: Account For The User's Rating (Movie + User Effect)

Model 2 takes into account the ratings for each individual movie, but that model can be improved upon by additionally taking into consideration the rating bias of the user (b_u) toward the movie. Some users rate box-office hit movies (statistically high-rated movies on average by a majority of the rating users) poorly, while others may rate all movies high (or low). This bias can have a great impact to those movies that have fewer ratings.

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

Where $\boldsymbol{Y}$ is the rating provided by user $\boldsymbol{u}$ for movie $\boldsymbol{i}$, μ is the overall average of all ratings, b_i is the bias ($\boldsymbol{b}$) added for each movie's ($\boldsymbol{i}$) average rating, b_u is the bias ($\boldsymbol{b}$) added for each user's ($\boldsymbol{u}$) average rating, and ε represents independent errors sampled from the same distribution centered at zero.

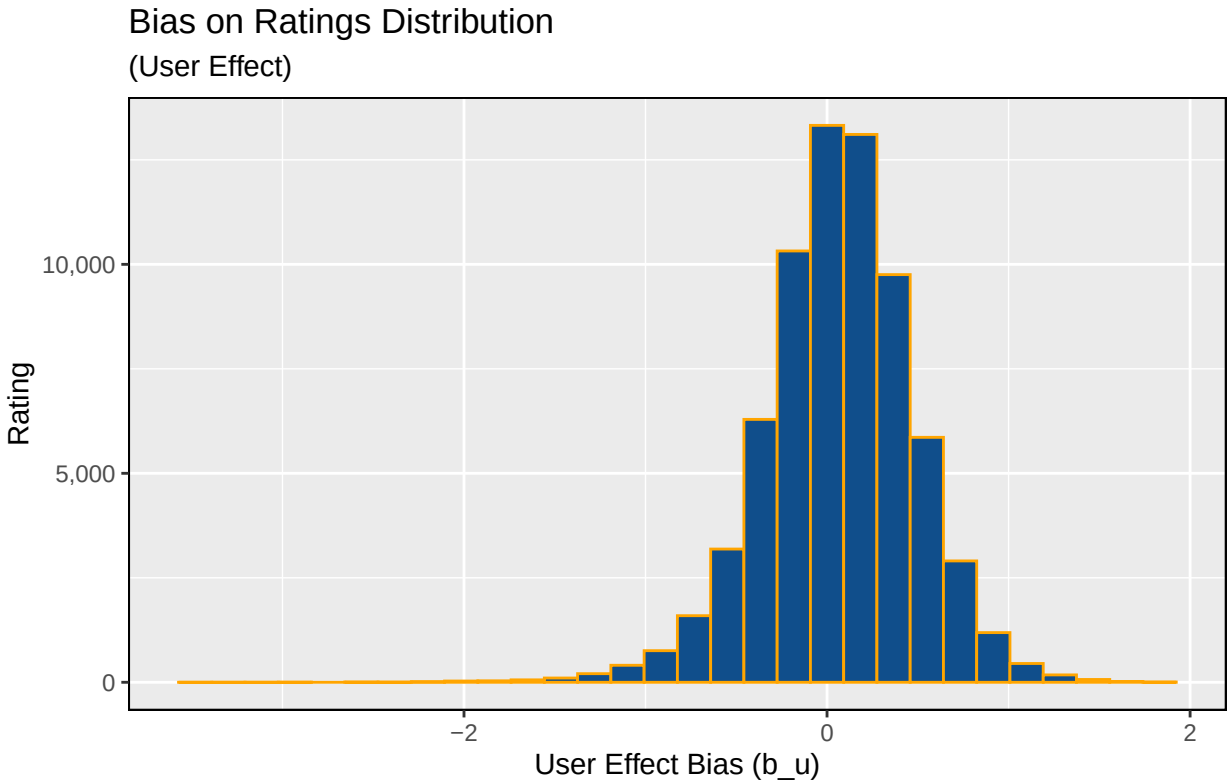
```
# Mean (mu) calculation
mu_edx_train <- mean(edx_train$rating)

# Each individual movie's mean rating calculation
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_edx_train))

# User's mean rating calculation, using the movie_avgs b_i vector.
user_avgs <- edx_train %>%
```

```
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_edx_train - b_i))

# Predict the ratings against the edx_test dataset.
predicted_ratings_m3 <- edx_test %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(predict_m3 = mu_edx_train + b_i + b_u) %>%
  pull(predict_m3)

# Test the "Movie + User Effect" model by calculating the RMSE using the edx_test
# dataset.
model3_rmse <- RMSE(predicted_ratings_m3, edx_test$rating)
model3_rmse
```

```
## [1] 0.8646843
```

Model	RMSE
Model 1: Average Rating Only (Naive Model)	**1.0600537**
Model 2: Movie Effect	**0.9429615**
Model 3: Movie + User Effect	**0.8646843**

Adding the "User Effect" bias with the "Movie Effect" bias improved the prediction, lowering the calculated RMSE to 0.8646843. While this RMSE already meets the requirement of this project, additional modeling will be done to see what the impact of other biases, regularization, and matrix factorization will have in lowering the RMSE further for the recommendation system. Plotting the bias (**Figure 5.3.1** below) shows the slightly positive effect the user's average rating (b_u) has on the mean (μ) rating distribution.



Source Data: edx_train
Figure 5.3.1

MODEL 4: Account For The Movie's Age When Rated (Movie + User + Age Effect)

An additional bias to consider would be the age of the movie when it was rated. In the initial data analysis of the `edx` data set in section 4.3, it was found that newer movies had more ratings than older movies, but older movies received higher average ratings. Several factors or biases can contribute to a curve like this, such as the user rating the movie right after seeing it online or in the theater, or some users may prefer older, more classic movies, while others may only want to watch the latest movies. This bias (b_a) will be added to the "Movie + User Effect" model to account for an additional dimension of data.

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

Where ***Y*** is the rating provided by user ***u*** for movie ***i***, μ is the overall average of all ratings, b_i is the bias (***b***) added for each movie's (***i***) average rating, b_u is the bias (***b***) added for each user's (***u***) average rating, b_a is the bias (***b***) added for each movie's (***a***) age at rating, and ε represents independent errors sampled from the same distribution centered at zero.

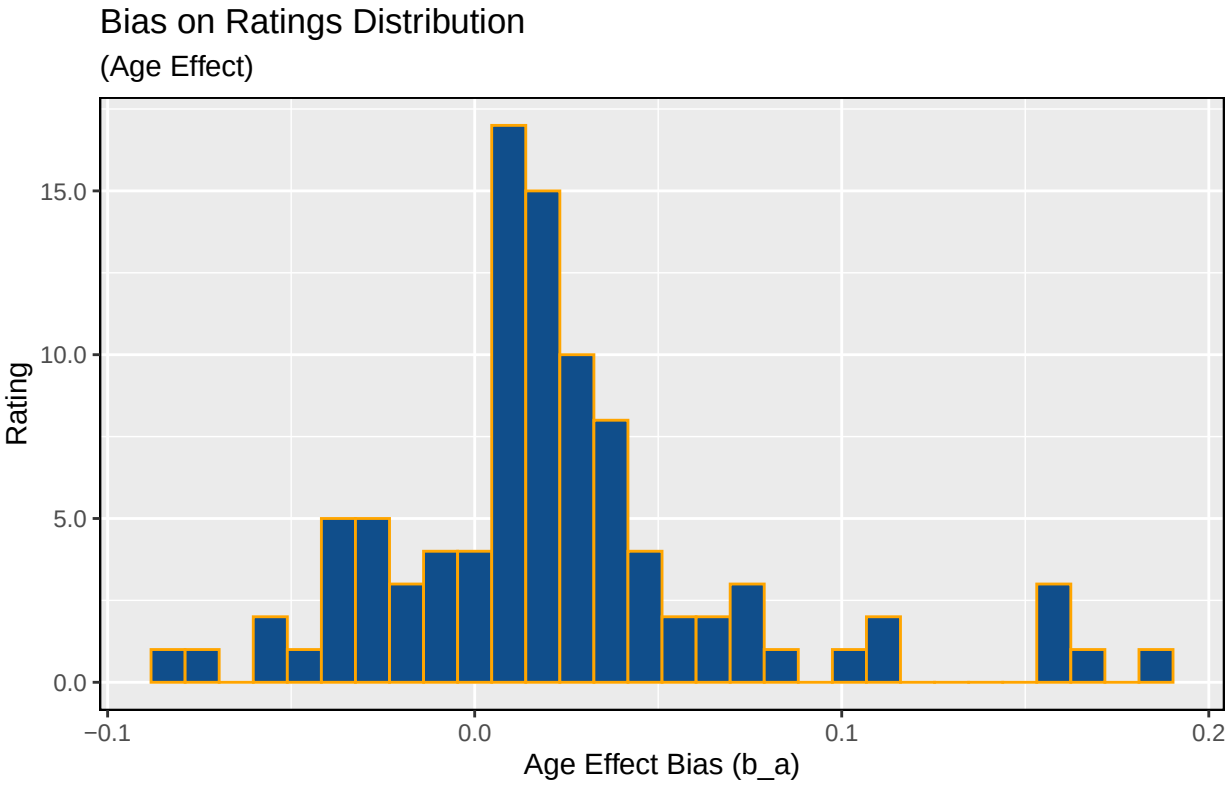
```
# Mean (mu) calculation
mu_edx_train <- mean(edx_train$rating)

# Each individual movie's mean rating calculation
movie_age_avgs <- edx_train %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
```

```
  mutate(year_rated = year(as_datetime(timestamp)),
         title_without_year = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\2" )) %>%
  separate(title_without_year,c("title_without_year","year_released"),"__") %>%
  mutate(age_at_rating = as.numeric(year_rated) - as.numeric(year_released)) %>%
  group_by(age_at_rating) %>%
  summarize(b_a = mean(rating - mu_edx_train - b_i - b_u))

# Predict the ratings against the edx_test dataset.
predicted_ratings_m4 <- edx_test %>%
  mutate(year_rated = year(as_datetime(timestamp)),
         title_without_year = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\2" )) %>%
  separate(title_without_year,c("title_without_year","year_released"),"__") %>%
  mutate(age_at_rating = as.numeric(year_rated) - as.numeric(year_released)) %>%
  group_by(age_at_rating) %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(movie_age_avgs, by = "age_at_rating") %>%
  mutate(predict_m4 = mu_edx_train + b_i + b_u + b_a) %>%
  pull(predict_m4)

# Test the "Movie + User + Age Effect" model by calculating the RMSE using the
# edx_test dataset.
model4_rmse <- RMSE(predicted_ratings_m4, edx_test$rating)
model4_rmse
```

```
## [1] 0.8642597
```

Model	RMSE
Model 1: Average Rating Only (Naive Model)	**1.0600537**
Model 2: Movie Effect	**0.9429615**
Model 3: Movie + User Effect	**0.8646843**
Model 4: Movie + User + Age Effect	**0.8642597**

Adding the "Age Effect" bias with the "Movie + User Effect" model only very slightly improved the prediction, lowering the calculated RMSE to 0.8642597. Plotting the bias (**Figure 5.4.1** below) shows the slightly positive effect the movie's age when the user rated it (b_a) has on the mean (μ) rating distribution.



Source Data: edx_train
Figure 5.4.1

MODEL 5: Account For The Movie's Genre (Movie + User + Genre Effect)

Toward the end of section 4.3, the analysis of whether genre had an impact on the movie's ratings showed a slight, but definitive effect when the genres were split to individual categories. Adding this bias will show whether the imbalance of the number of ratings for each genre to the above-average ratings for fewest-rated genres (especially when each movie can belong to more than one), will have a positive or negative effect on the recommender system (based on its calculated RMSE).

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}^{k} b_k + \varepsilon_{u,i}$$

with $x_{u,i}^{k} = 1$ if $g_{u,i}$ (the genre for user $\boldsymbol{u}$'s rating of movie $\boldsymbol{i}$) is genre k.

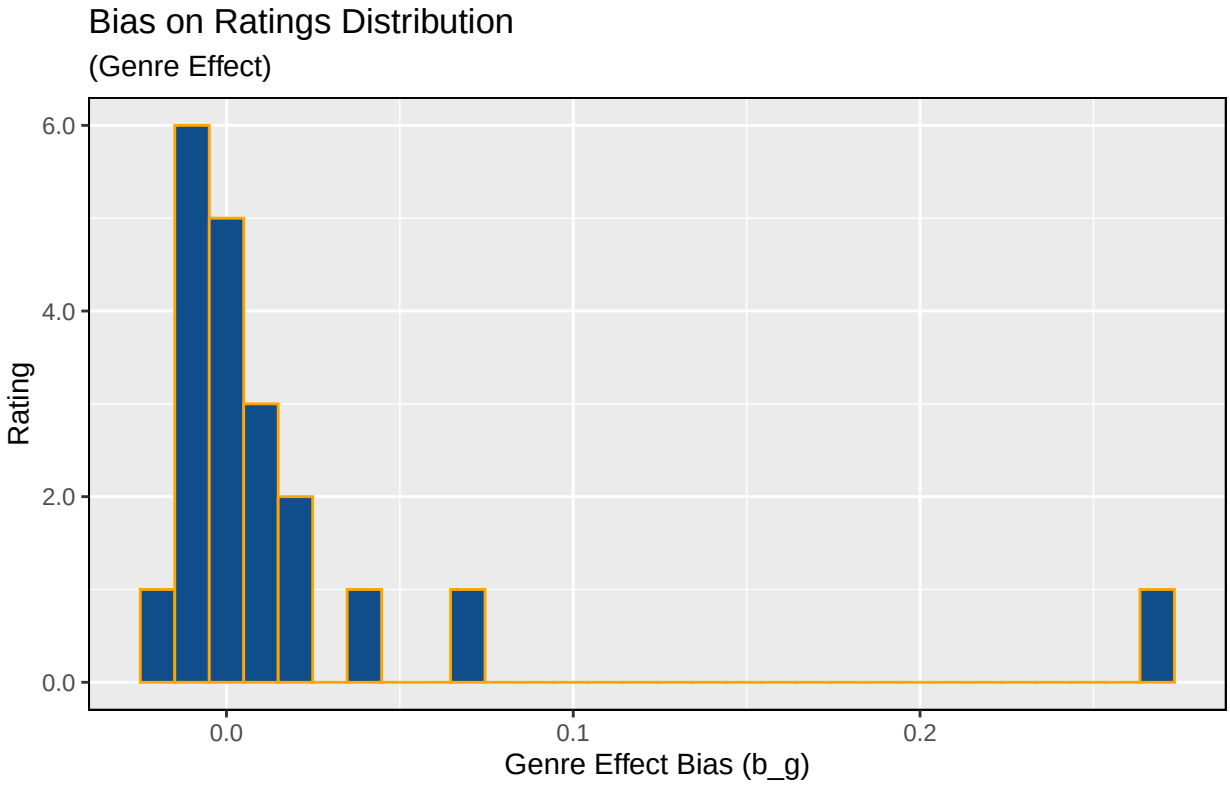
```
# Split the genres for each movie into their individual form for both the edx_train
# and edx_test datasets, copying all data into a new data frame for each.
edx_train_split_genres <- edx_train %>% separate_rows(genres, sep = "\\|")
edx_test_split_genres <- edx_test %>% separate_rows(genres, sep = "\\|")

# Mean (mu) calculation
mu_edx_train <- mean(edx_train_split_genres$rating)

# Calculate each movie's mean rating
movie_avgs <- edx_train_split_genres %>%
```

```
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_edx_train))

# Calculate each user's mean rating for each movie
user_avgs <- edx_train_split_genres %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_edx_train - b_i))

# Calculate the mean for each movie rated by each user that was in a
# particular genre category.
genre_avgs <- edx_train_split_genres %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_edx_train - b_i - b_u))

# Predict the ratings against the edx_test_split_genres dataset.
predicted_ratings_m5 <- edx_test_split_genres %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  mutate(predict_m5 = mu_edx_train + b_i + b_u + b_g) %>%
  pull(predict_m5)

# Test the "Movie + User + Genre Effect" model by calculating the RMSE using the
# edx_test dataset.
model5_rmse <- RMSE(predicted_ratings_m5, edx_test_split_genres$rating)
model5_rmse
```

```
## [1] 0.8628863
```

Model	RMSE
Model 1: Average Rating Only (Naive Model)	**1.0600537**
Model 2: Movie Effect	**0.9429615**
Model 3: Movie + User Effect	**0.8646843**
Model 4: Movie + User + Age Effect	**0.8642597**
Model 5: Movie + User + Genre Effect	**0.8628863**

Adding the "Genre Effect" bias with the "Movie + User Effect" model only very slightly improved the prediction, lowering the calculated RMSE to 0.8628863. Plotting the bias (b_g) in **Figure 5.5.1** below shows the slightly positive effect the movie's genre has on the mean (μ) rating distribution.



Source Data: edx_train
Figure 5.5.1

MODEL 6: Regularization of The Movie + User Effect

Remembering **Figure 4.3.2** (Rating Distribution by `movieId` and `userId`), some movies were rated a greater number of times more than others, and some users rated a greater number of movies than others. Regularization is a method that can limit the effects of either or both of these two variables by penalizing large estimates that come from small sample sizes in order to improve the RMSE results. It assumes smaller weights generate simpler models, which helps avoid overfitting. This model will be regularizing both movies and users:

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

Where λ is a tuning parameter (chosen by using cross-validation on the `edx_train` training set *only*, with the assessment using the `edx_test` test set) that will determine how much to penalize any large estimates from small sample sizes.

The parameter estimates (the values of b that minimize this equation) for `movieId` are:

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

Where n_i is the number or ratings for movie ***i***. When n_i is small (fewer ratings for movie ***i***), the λ value has a greater impact on the equation, whereas, if n_i is large (many people rated movie ***i***), the λ value has little effect, since it assumes more skepticism by shrinking.

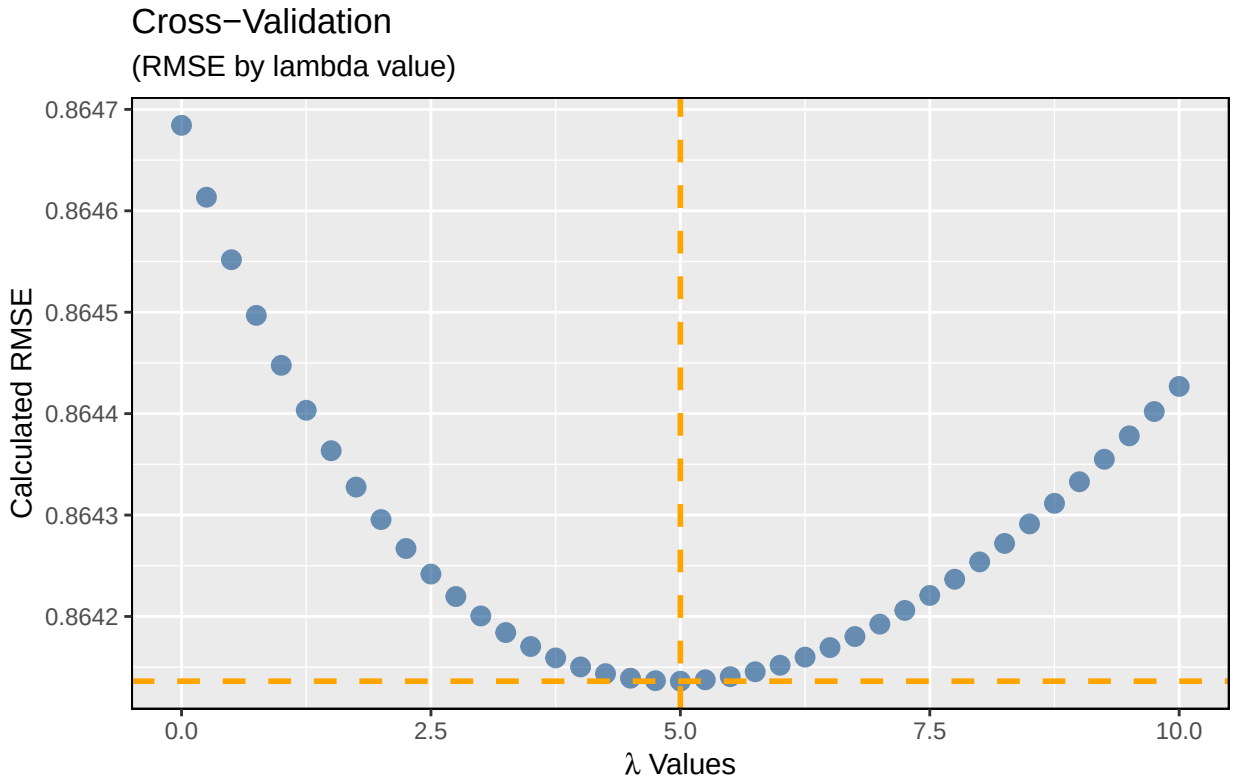
The parameter estimates for `userId` in this model already assume the regularized movie parameter estimates (b_i):

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

Where n_u is the number of ratings provided by user ***u***.

```
# Set the lambda tuning parameter sequence to be 1-10 at 0.25 intervals.
lambdas <- seq(0, 10, 0.25)

# Calculate the individual rmse values using each lambda parameter interval.
rmses_m6 <- sapply(lambdas, function(l){
  mu_edx_train <- mean(edx_train$rating)
  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i=sum(rating - mu_edx_train)/(n() + l))
  b_u <- edx_train %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u=sum(rating - mu_edx_train -b_i)/(n() + l))
  predicted_ratings_m6 <- edx_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(predict_m6 = mu_edx_train + b_i + b_u) %>%
    pull(predict_m6)
  return(RMSE(predicted_ratings_m6, edx_test$rating))
})

# Find the optimal lambda value that had the best RMSE
model6_opt_lambda <- lambdas[which.min(rmses_m6)]
model6_opt_lambda
```

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



Source Data: edx_train
Figure 5.6.1

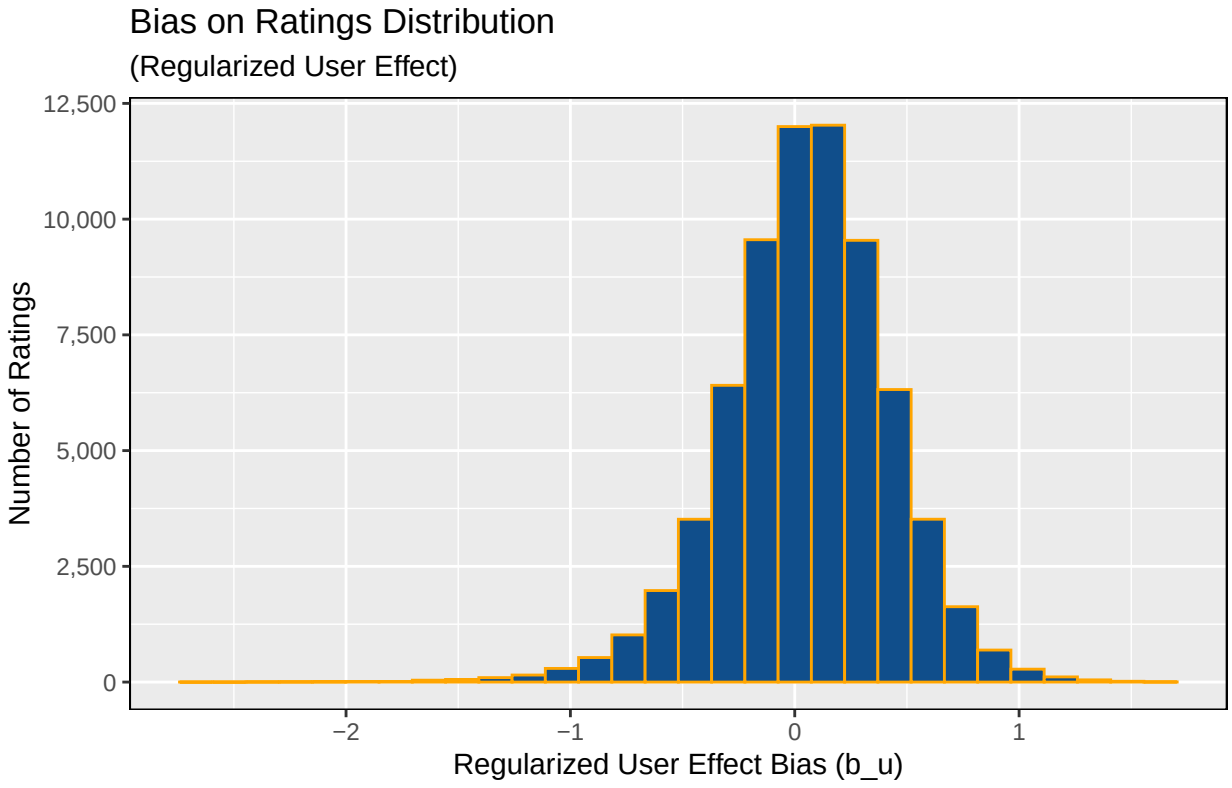
The cross-validation plot in **Figure 5.6.1** above shows the intersection points between the lowest RMSE and the λ value used when calculating it. The remaining portion of the regularization algorithm can now be run with the optimal lambda value of 5:

```
# Use the optimal lambda value to calculate the regularized movie bias
b_i <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i=sum(rating - mu_edx_train)/(n() + model6_opt_lambda))
# Use the optimal lambda value to calculate the regularized user bias
b_u <- edx_train %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u=sum(rating - mu_edx_train - b_i)/(n() + model6_opt_lambda))
# Predict the ratings against the edx_test dataset.
predicted_ratings_m6 <- edx_test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(predict_m6 = mu_edx_train + b_i + b_u) %>%
  pull(predict_m6)

# Test the Regularized "Movie + User Effect" model by calculating the RMSE using
# the edx_test dataset.
model6_rmse <- RMSE(predicted_ratings_m6, edx_test$rating)
model6_rmse
```

```
## [1] 0.8641341
```

Model	RMSE
Model 1: Average Rating Only (Naive Model)	**1.0600537**
Model 2: Movie Effect	**0.9429615**
Model 3: Movie + User Effect	**0.8646843**
Model 4: Movie + User + Age Effect	**0.8642597**
Model 5: Movie + User + Genre Effect	**0.8628863**
Model 6: Regularized Movie + User Effect	**0.8641341**

Adding Regularization to the "Movie + User Effect" model improved the prediction a little, lowering the originally calculated RMSE from 0.8646843 to 0.8641341. Plotting the bias (**Figure 5.6.2** below) shows the slightly positive effect the regularized user's average rating has on the rating distribution.



Source Data: edx_train
Figure 5.6.2

MODEL 7: Regularization of The Movie + User + Age + Genre Effect

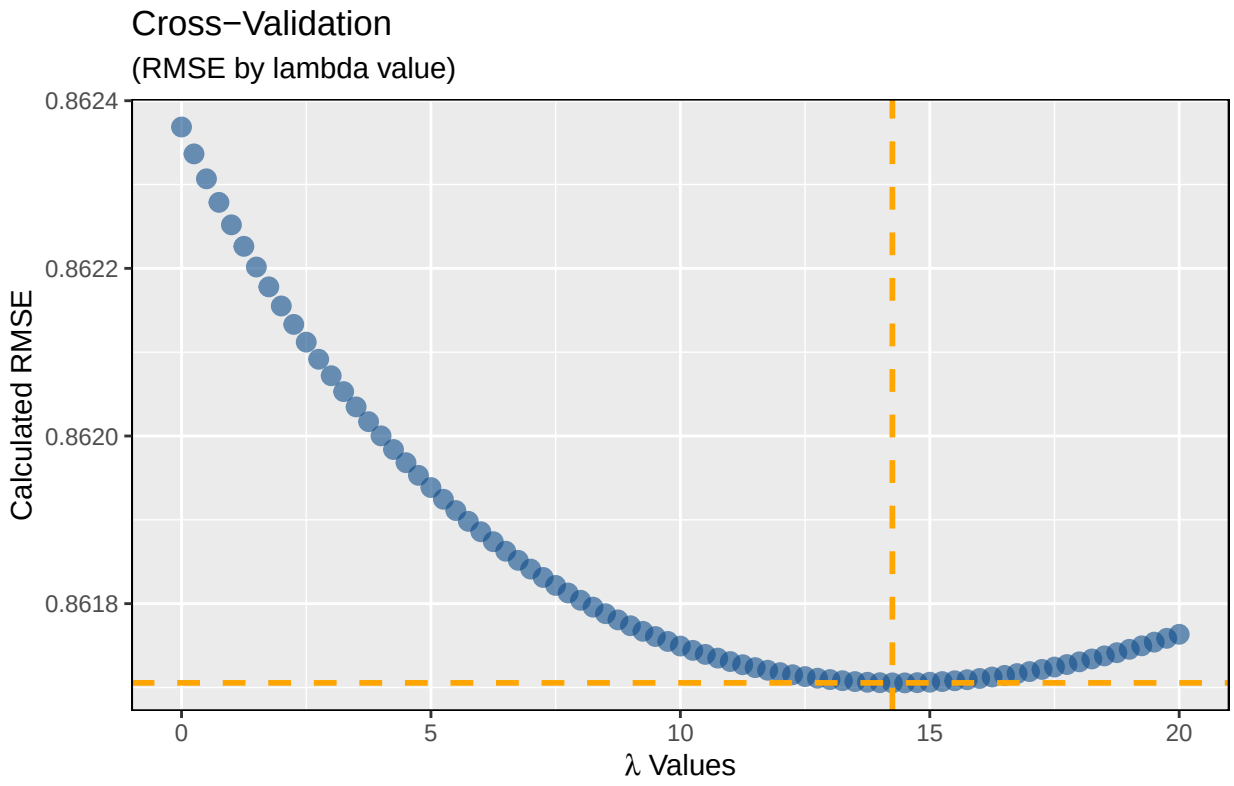
Since the regularized "Movie + User Effect" model was able to lower the RMSE, this model will show what would happen if more biases were added to the equation. The non-regularized biases were plotted for all four factors (movie, user, age, genre) individually, which showed varying impacts to the ratings. This model will place all four together and regularization will be used (using the same λ value for each) to penalize large estimates.

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

```
# Convert the timestamp to a date/time format and extract the year, separate
# the movie release year from the movie's title, store the difference between
# the two as the movie's age when rated, then split the genres for each movie
# into their individual form for both the edx_train and edx_test datasets,
# copying all data into a new data frame for each.
edx_train_m7 <- edx_train %>% mutate(year_rated = year(as_datetime(timestamp)),
                           title_without_year = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\
  separate(title_without_year,c("title_without_year","year_released"),"__") %>%
  mutate(age_at_rating = as.numeric(year_rated) - as.numeric(year_released)) %>%
  separate_rows(genres, sep = "\\|")

edx_test_m7 <- edx_test %>% mutate(year_rated = year(as_datetime(timestamp)),
                                   title_without_year = str_replace(title,"^(.+)\\s\\((\\d{4})
  separate(title_without_year,c("title_without_year","year_released"),"__") %>%
  mutate(age_at_rating = as.numeric(year_rated) - as.numeric(year_released)) %>%
  separate_rows(genres, sep = "\\|")

# Set the lambda tuning parameter sequence to be 1-20 at 0.25 intervals.
lambdas <- seq(0, 20, 0.25)

# Calculate the individual rmse values using each lambda parameter interval.
rmses_m7 <- sapply(lambdas, function(l){
  mu_edx_train <- mean(edx_train_m7$rating)
  b_i <- edx_train_m7 %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_edx_train)/(n() + l))
  b_u <- edx_train_m7 %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu_edx_train - b_i)/(n() + l))
  b_a <- edx_train_m7 %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(age_at_rating) %>%
    summarize(b_a = sum(rating - mu_edx_train - b_i - b_u)/(n() + l))
  b_g <- edx_train_m7 %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
```

```
    left_join(b_a, by = "age_at_rating") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu_edx_train - b_i - b_u - b_a)/(n() + l))
  predicted_ratings_m7 <- edx_test_m7 %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_a, by = "age_at_rating") %>%
    left_join(b_g, by = "genres") %>%
    mutate(predict_m7 = mu_edx_train + b_i + b_u + b_a + b_g) %>%
    pull(predict_m7)
  return(RMSE(predicted_ratings_m7, edx_test_m7$rating))
})

# Find the optimal lambda value that had the best (lowest) RMSE
model7_opt_lambda <- lambdas[which.min(rmses_m7)]
model7_opt_lambda
```

```
## [1] 14.25
```



Source Data: edx_train
Figure 5.7.1

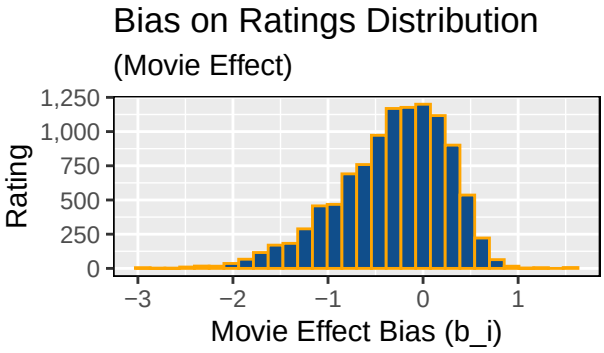
The cross-validation plot in **Figure 5.7.1** above shows the intersection points between the lowest RMSE and the λ value used when calculating it. The remaining portion of the regularization algorithm can now be run with the optimal lambda value of 14.25:

```
# Use the optimal lambda value to calculate the "Movie + User + Age + Genre
# Effect" predicted ratings.

# Use the optimal lambda value to calculate the regularized movie bias
b_i <- edx_train_m7 %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_edx_train)/(n() + model7_opt_lambda))
# Use the optimal lambda value to calculate the regularized user bias
b_u <- edx_train_m7 %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_edx_train - b_i)/(n() + model7_opt_lambda))
# Use the optimal lambda value to calculate the regularized movie age at rating bias
b_a <- edx_train_m7 %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(age_at_rating) %>%
  summarize(b_a = sum(rating - mu_edx_train - b_i - b_u)/(n() + model7_opt_lambda))
# Use the optimal lambda value to calculate the regularized movie genre bias
b_g <- edx_train_m7 %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_a, by = "age_at_rating") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu_edx_train - b_i - b_u - b_a)/(n() + model7_opt_lambda))
# Predict the ratings against the edx_test_m7 dataset.
predicted_ratings_m7 <- edx_test_m7 %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_a, by = "age_at_rating") %>%
  left_join(b_g, by = "genres") %>%
  mutate(predict_m7 = mu_edx_train + b_i + b_u + b_a + b_g) %>%
  pull(predict_m7)

# Test the Regularized "Movie + User + Age + Genre Effect" model by calculating
# the RMSE using the edx_test_m7 dataset.
model7_rmse <- RMSE(predicted_ratings_m7, edx_test_m7$rating)
model7_rmse
```

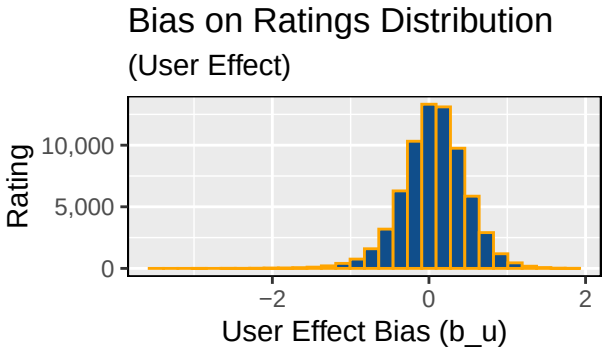
```
## [1] 0.8617055
```

Model	RMSE
Model 1: Average Rating Only (Naive Model)	**1.0600537**
Model 2: Movie Effect	**0.9429615**
Model 3: Movie + User Effect	**0.8646843**
Model 4: Movie + User + Age Effect	**0.8642597**
Model 5: Movie + User + Genre Effect	**0.8628863**
Model 6: Regularized Movie + User Effect	**0.8641341**
Model 7: Regularized Movie + User + Age + Genre Effect	**0.8617055**

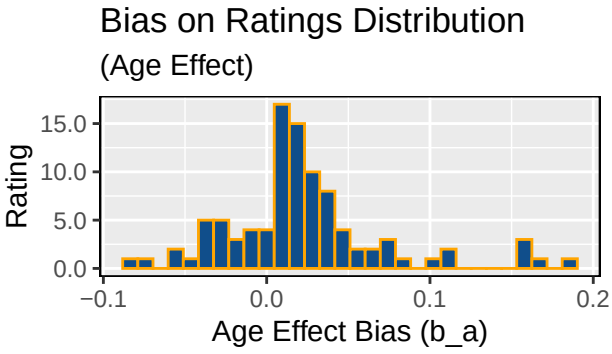
Taking this approach of regularizing all of the tested biases together and lumping them into one model only has a small incremental decrease in the calculated RMSE for each bias added. Continually adding to the complexity of the model can potentially introduce the issue of over-fitting, though these results could also have been anticipated based on **Figures 5.2.1, 5.3.1, 5.4.1, and 5.5.1** (grouped together below) that all chart the effect their respective bias (though, not regularized) has on the ratings distribution. Adding both the movie's age effect and genre effect to the originally regularized "Movie + User Effect" only lowered the RMSE from 0.8646843 to 0.8617055 (an improvement of only 0.29788%).



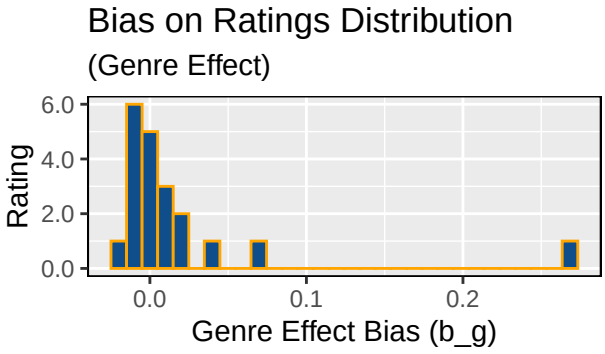
Source Data: edx_train
Figure 5.2.1



Source Data: edx_train
Figure 5.3.1



Source Data: edx_train
Figure 5.4.1



Source Data: edx_train
Figure 5.5.1

MODEL 8: Using Matrix Factorization

One popular technique to use for recommender systems is matrix factorization. The idea is to approximate the whole rating matrix by the product of two matrices– in this case, each user gets a row and each movie gets a column:

$$r_{u,i} = y_{u,i} - \hat{b_i} + \hat{b_u}$$

where $y_{u,i}$ is the entry in row $\boldsymbol{u}$ and column $\boldsymbol{i}$.

The previous models used did not take into account any similar ratings patterns of the users or movies. Matrix factorization observes these patterns by factorizing the residuals (r) into vectors p (unrelated user effects) and q (the principal component: movies), where $r_{u,i} \approx p_u q_i$. The resulting equation used in this model is:

$$Y_{u,i} = \mu + b_i + b_i + p_{u,1} q_{1,i} + p_{u,2} q_{2,i} + \epsilon_{i,j}$$

This model will use the "recosystem" package[6] (Recommender System using Matrix Factorization), which is simply an R wrapper of the LIBMF library that uses a fast parallel stochastic gradient method for matrix factorization.[7]

The use of the `recosystem` package in this model consists of the following steps:

1. Create a `recosystem`-formatted copy of both the `edx_train` and `edx_test` datasets, passing in only the `userId`, `movieId` and `rating` columns.
2. Create a model object (a Reference Class object in R) by calling `Reco()`.
3. Call the `$tune()` method to select best tuning parameters along a set of candidate values. *Note: If tune() is not specified, the default parameters will be used, and the calculation period will be exponentially longer. Cross validation will be used to tune these parameters, and will be chosen by minimizing the RMSE as a loss function.*
4. Train the model against `edx_train` using the `$train()` method, calling in a number of parameters that will come from the result of calling `$tune()`.
5. Use the `$predict()` method against the `edx_test` dataset to compute the predicted values.
6. Search the resultant data frame for any predictions that are higher than 5, and lower than 0.5 and set those rows to 5 and 0.5, respectively.
7. Test the model, calculating the RMSE value.

```
# Create a Recosystem-formatted training dataset from the in-memory edx_train dataset.
edx_train_mf <- with(edx_train, data_memory(user_index = userId,
                                    item_index = movieId,
                                    rating = rating))

# Create a Recosystem-formatted testing dataset from the in-memory edx_test dataset.
edx_test_mf <- with(edx_test, data_memory(user_index = userId,
                                  item_index = movieId,
                                  rating = rating))
# Create the model object
r <- Reco()

# Cross validation (k=5 by default) of Recosystem-formatted edx_train training dataset
# to compute the residuals from a set of tuning parameters.  If the "tune_opts" list is
# left blank, all defaults will be used, which can take 8 or more hours to run.  The
# first time this was run, all defaults were used and the below values were determined
# to be optimal parameters.
```

[6] https://github.com/yixuan/recosystem
[7] https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

```
tune_opts <- r$tune(edx_train_mf,
              opts = list(dim = 30, # Number of latent factors
                          lrate = 0.1, # Learning rate, or step size in gradient descent
                          costp_l1 = 0, # L1 regularization parameter for user factors
                          costp_l2 = 0.01, # L2 regularization parameter for user factors
                          costq_l1 = 0, # L1 regularization parameter for item factors
                          costq_l2 = 0.1, # L2 regularization parameter for item factors
                          nthread  = 6, # Number of parallel computing threads
                          niter = 20)) # Number of iterations

# Train the algorithm with the recosystem-formatted edx_train dataset, using the optimal
# tuning parameters.
r$train(edx_train_mf, opts = c(tune_opts$min, nthread = 6, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9800   1.0994e+07
##    1       0.8764   9.0023e+06
##    2       0.8429   8.3557e+06
##    3       0.8204   7.9561e+06
##    4       0.8039   7.6872e+06
##    5       0.7914   7.4971e+06
##    6       0.7814   7.3511e+06
##    7       0.7731   7.2378e+06
##    8       0.7659   7.1443e+06
##    9       0.7595   7.0665e+06
##   10       0.7539   7.0008e+06
##   11       0.7490   6.9431e+06
##   12       0.7445   6.8947e+06
##   13       0.7404   6.8517e+06
##   14       0.7366   6.8104e+06
##   15       0.7332   6.7777e+06
##   16       0.7301   6.7460e+06
##   17       0.7272   6.7175e+06
##   18       0.7245   6.6913e+06
##   19       0.7219   6.6695e+06
```

```
# Run the recosystem predict function against the edx_test dataset.
y_hat_ratings <-  r$predict(edx_test_mf, out_memory())

# Find which rows have rating predictions over the max value of 5 (the highest rating).
over5 <- which(y_hat_ratings > 5)
# Set all discovered rows with ratings above 5 to max rating value of 5.
y_hat_ratings[over5] <- 5

# Find which rows have rating predictions under the min value of 0.5 (the lowest rating).
under0_5 <- which(y_hat_ratings < 0.5)
# Set all discovered rows with ratings under 0.5 to minimum rating value of 0.5.
y_hat_ratings[under0_5] <- 0.5
```

```
# Test the Matrix Factorization Model's RMSE value
model8_rmse <- RMSE(edx_test$rating, y_hat_ratings)
model8_rmse
```

```
## [1] 0.7857226
```

To aid the model, knowing there cannot be any rating above 5 or below 0.5, any predictions that fit that profile are adjusted to the highest possible value of 5 or lowest of 0.5, respectively. This run found 3302 rows with predictions above the highest whole number rating of 5, and 433 rows with predictions under the lowest rating value, 0.5.

Model	RMSE
Model 1: Average Rating Only (Naive Model)	**1.0600537**
Model 2: Movie Effect	**0.9429615**
Model 3: Movie + User Effect	**0.8646843**
Model 4: Movie + User + Age Effect	**0.8642597**
Model 5: Movie + User + Genre Effect	**0.8628863**
Model 6: Regularized Movie + User Effect	**0.8641341**
Model 7: Regularized Movie + User + Age + Genre Effect	**0.8617055**
Model 8: Matrix Factorization	**0.7857226**

Based on the substantial drop in the calculated RMSE to 0.7857226, observing similarities in the rating pattern of users to movies has a significant impact on the predictability of a user's movie preference (by their rating).

The `niter` property (number of training iterations on the `edx_train_mf` dataset) in the recosystem tuning parameters was additionally tested with the value of 40 to see what would happen, and the overall RMSE lowered even further to 0.78429. Though, simply adding additional iterations can quickly add the issue of "over-smoothing".

6. RESULTS

Having done all of the initial data analysis in section 4 and pinpointing which factors may contribute to (or at least warrant a model to test) an effective movie recommendation system, stepping through eight (8) different models yielded the best results with the use of matrix factorization.

While testing the various performance tuning options of the recosystem package did result in a few extended compute-intensive timeframes (up to 12 hours on an 8-core hyperthreaded processor with 64GB RAM), the final tests with the optimal parameters used in this report only took around 3 minutes to complete from start to finish. In contrast, Model 7 (Regularization of The Movie + User + Age + Genre Effect) consistently takes between 30-45 minutes to complete from start to finish because the number of computations compound for each additional effect.

The final RMSE results table below shows the summary list of all prediction models tested for a movie recommendation system using the edx_train and edx_test datasets.

Model	RMSE
Model 1: Average Rating Only (Naive Model)	**1.0600537**
Model 2: Movie Effect	**0.9429615**
Model 3: Movie + User Effect	**0.8646843**
Model 4: Movie + User + Age Effect	**0.8642597**
Model 5: Movie + User + Genre Effect	**0.8628863**
Model 6: Regularized Movie + User Effect	**0.8641341**
Model 7: Regularized Movie + User + Age + Genre Effect	**0.8617055**
Model 8: Matrix Factorization	**0.7857226**

The final hold-out testing of this model using the initial `edx` and `validation` datasets yielded the following final RMSE:

```
# Create a Recosystem-formatted training dataset from the in-memory edx dataset.
edx_mf <- with(edx, data_memory(user_index = userId,
                                item_index = movieId,
                                rating = rating))

# Create a Recosystem-formatted testing dataset from the in-memory validation dataset.
validation_mf <- with(validation, data_memory(user_index = userId,
                                              item_index = movieId,
                                              rating = rating))
# Create the model object
r <- Reco()

# Cross validation (k=5 by default) of Recosystem-formatted edx training dataset to
# compute the residuals from a set of tuning parameters.  If the "tune_opts" list is
# left blank, all defaults will be used, which can take 8 or more hours to run.  The
# first time this was run, all defaults were used and the below values were determined
# to be optimal parameters.
```

```
tune_opts <- r$tune(edx_mf,
              opts = list(dim = 30, # Number of latent factors
                          lrate = 0.1, # Learning rate, or step size in gradient descent
                          costp_l1 = 0, # L1 regularization parameter for user factors
                          costp_l2 = 0.01, # L2 regularization parameter for user factors
                          costq_l1 = 0, # L1 regularization parameter for item factors
                          costq_l2 = 0.1, # L2 regularization parameter for item factors
                          nthread  = 6, # Number of parallel computing threads
                          niter = 20)) # Number of iterations

# Train the algorithm with the recosystem-formatted edx dataset, using the optimal
# tuning parameters.
r$train(edx_mf, opts = c(tune_opts$min, nthread = 6, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9726   1.1987e+07
##    1       0.8728   9.8768e+06
##    2       0.8393   9.1832e+06
##    3       0.8174   8.7642e+06
##    4       0.8014   8.4732e+06
##    5       0.7894   8.2758e+06
##    6       0.7796   8.1213e+06
##    7       0.7714   8.0034e+06
##    8       0.7644   7.9037e+06
##    9       0.7584   7.8250e+06
##   10       0.7532   7.7554e+06
##   11       0.7485   7.6987e+06
##   12       0.7443   7.6473e+06
##   13       0.7405   7.6027e+06
##   14       0.7371   7.5627e+06
##   15       0.7339   7.5289e+06
##   16       0.7310   7.4985e+06
##   17       0.7284   7.4701e+06
##   18       0.7259   7.4444e+06
##   19       0.7236   7.4201e+06
```

```
# Run the recosystem predict function against the validation test dataset.
y_hat_ratings <-  r$predict(validation_mf, out_memory())

# Find which rows have rating predictions over the max value of 5 (the highest rating).
over5 <- which(y_hat_ratings > 5)
# Set all discovered rows with ratings above 5 to max rating value of 5.
y_hat_ratings[over5] <- 5

# Find which rows have rating predictions under the min value of 0.5 (the lowest rating).
under0_5 <- which(y_hat_ratings < 0.5)
# Set all discovered rows with ratings under 0.5 to minimum rating value of 0.5.
y_hat_ratings[under0_5] <- 0.5
```

```
# Calculate the Matrix Factorization Model's RMSE value
final_model_rmse <- RMSE(validation$rating, y_hat_ratings)
final_model_rmse
```

```
## [1] 0.7823329
```

Model	RMSE
FINAL MODEL: Matrix Factorization	**0.7823329**

7. CONCLUSION

The purpose of this project was to create a movie recommendation system using the MovieLens 10M dataset, with the goal of obtaining a Root Mean Square Error (RMSE) value that is less than 0.86490.

Interestingly, the model with the lowest RMSE value only used the `movieId` and `userId` features. While adding considerations for movie, user, movie age at rating, and movie genre biases did improve the ratings prediction (having achieved the target RMSE in earlier models), matrix factorization was most effective in discovering the patterns that exist between the users and movies in the sparse dataset. Regularization did not provide a significant improvement (only a 0.05502% performance increase in the Regularized Movie + User model).

After testing 8 different predictive models, using their respective RMSE values as the success indicator for each, the Matrix Factorization model (using the `recosystem` package) yielded the lowest value of 0.7823329. Comparing this RMSE to the initial "Naive Model" (1.0600537), where it was assumed that all users rated all movies the same, there was an improvement of approximately 28%.

The sparse nature of the data impacts the prediction accuracy, especially for both users and movies that have few ratings. Future work would include re-training these models over time as additional data is entered (for better accuracy), investigate impact of the user rating time period to determine if they were influenced by any historical or socioeconomic events, and investigate per-bias lambda values when computing the residuals to see if accuracy can be improved.