Movielens 10M Report [Harvardx Project]

Tyler Chang

7/3/2022

Contents

Overview of the Project

This reports documents the development and subsequent testing of six linear regression models and one matrix factorization-based model for predicting the ratings of movies given by viewers. The metric of success used for all models is *root mean square error* (**RMSE**), with a target goal of < 0.8649. As will be shown, while this goal can be met using a regularized linear regression model, the RMSE score can be reduced significantly beyond the goal using matrix factorization via the **recosystem** library.

The data set used is a subset of the movielens data set. The data set for this report, henceforth referred to as movielens, includes information about 10,000,054 movie ratings for over 10,000 movies provided by approximately 138,000 users.

The report is composed of four sections, the first of which is this overview. In the following section, **Analysis**, I begin by dividing the overall data set into two subsets, **edx** and **validation**. I then provide both visual and numeric descriptions of the data sets, including but not limited to, checking the data sets to ensure there are no missing values, determining the most and least reviewed films, and showing trends among users to rate above or below the overall average rating across all films.

In the next section, **Results**, the **edx** data set is further subdivided into two subsets, **train_edx** and **test_edx**. These new sets are then used to develop six linear regression models:

1. *Naive Mean*: Predicts based only the overall average rating.
2. *Mean + Movie Bias*: Adds in consideration for any bias associated with particular films' ratings.
3. *Mean + User Bias*: Excludes movie bias but includes potential bias coming from particular users.
4. *Mean + User and Movie Biases*: Includes both user and movie biases.
5. *Regularized Mean + User and Movie Biases*: Penalizes the impact of films that have very few reviews and the users who have reviewed very few films.
6. *Regularized Mean + User, Movie, and Time Biases*: This model considers potential biases from users, movies, and when a rating was given. It then penalizes outlier values before making a prediction.

The fourth, fifth, and sixth models are below the target goal but they do not maximally reduce the RMSE score. To further improve the RMSE score, I switch from a linear regression approach to matrix factorization via the recosystem library. In doing so, I develop a seventh model, which yields an RMSE of approximately 0.785, by far the lowest score among the models. This same model is then applied to the original edx and validation sets to develop the final predictive model.

The last section, **Conclusion**, provides a short summary of the report, acknowledges potential flaws and limitations, and provides some suggestions for future improvements. Additionally, a list of sources referenced in the production of this report is provided.

NOTE: All of the code included in this report was written on R version 4.1.2 using a Macintosh OS. While no issues should be expected knitting the associated Rmd file on a PC OS, this has not been tested.

Analysis

Libraries Several libraries are used throughout the report. Though some are not used until the latter sections, all libraries will be installed (if necessary) and called now.

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

library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(recosystem)
```

The Data Set The relevant data file, **Movielens 10M data**, can be accessed via the following links:

1. Website
2. Zip File

Before proceeding, the zip file containing the data must be downloaded, unzipped, and the data set's columns must be assigned names.

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

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

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

With the file now downloaded, the data set will be converted into a data frame. Note that movielens will contain all of the information used throughout the remainder of the report, though the object itself will not be used for model development.

```
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


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

It is now appropriate to divide the movielens data frame into a training set **edx** and a testing set **validation**. Note that while **validation** will be used at the end of the third section to find the RMSE score of the final model, it will not be used until then. 90% of the data will be partitioned into the **edx** set and the remaining 10% will be assigned to **validation**.

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
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")
```

Multiple rows have been removed from **validation** must now be added back into **edx**. Following that, several r objects–**dl, ratings, movies, test_index, temp, movielens, and removed**–will no longer be needed and can be deleted.

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

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

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

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

Quality check for edx and validation sets Before looking at the contents of either **edx** or **validation**, it is important to confirm that the dimensions of each set are correct. Note that edx should have approximately 9 million rows and 6 columns and validation should have approximately 1 million rows and 6 columns.

```
#Confirming the dimensions of the edx dataset
dim(edx)
```

```
## [1] 9000055       6
```

```
#Confirming the dimensions of the validation dataset
dim(validation)
```

```
## [1] 999999      6
```

It is also important to ensure that there are no NA values appearing in either data set.

```
#Checking for missing values in edx or validation
anyNA(edx)
```

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

```
anyNA(validation)
```

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

Fortunately, no NA values appear in either set and both sets have their respective expected dimensions.

Exploring the edx data set Now that the initial quality checks have been performed, it is appropriate to start exploring the **edx** data set. As a first step, let us take a look at the first six rows of **edx**.

```
head(edx)
```

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

The **edx** data set includes six columns–**userId, movieId, rating, timestamp, title, and genres**. Since movidId and title are interchangeable in regard to identifying which movie a given row is in reference to, I will only make use of the former.

Since being able to accurately predict the ratings is what we are most interested in, it makes sense to begin by exploring some details about the ratings themselves. To get a general idea of what range the ratings span and where the measures of center (mean and median) are, consider:

```
summary(edx$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.512   4.000   5.000
```

From this, we can see that while the **mean** is *3.512*, the **median** is non-trivially higher at *4*. This implies that there are more ratings above the mean than below it. This is further supported by the **1st quartile** being at *3*, which is at the 60th-percentile of possible values for a given rating. Additionally, we can see that no user gave any movie a rating of *0* but at least one gave a rating of *5*.

It is also helpful to know how many movies were actually rated and how many unique users provided at least one rating. These metrics can be computed via:

```
edx %>%
  summarize(number_of_raters = n_distinct(userId),
            number_of_movies = n_distinct(movieId))
```

```
##   number_of_raters number_of_movies
## 1            69878            10677
```

There are 69,878 unique users and only 10,677 movies. This, alongside the fact that there were over 10,000,000 rows in **movielens** (where each row represents a single rating), implies that a large number of users provided multiple reviews. Still, more can be learned by examining the frequency of each rating.

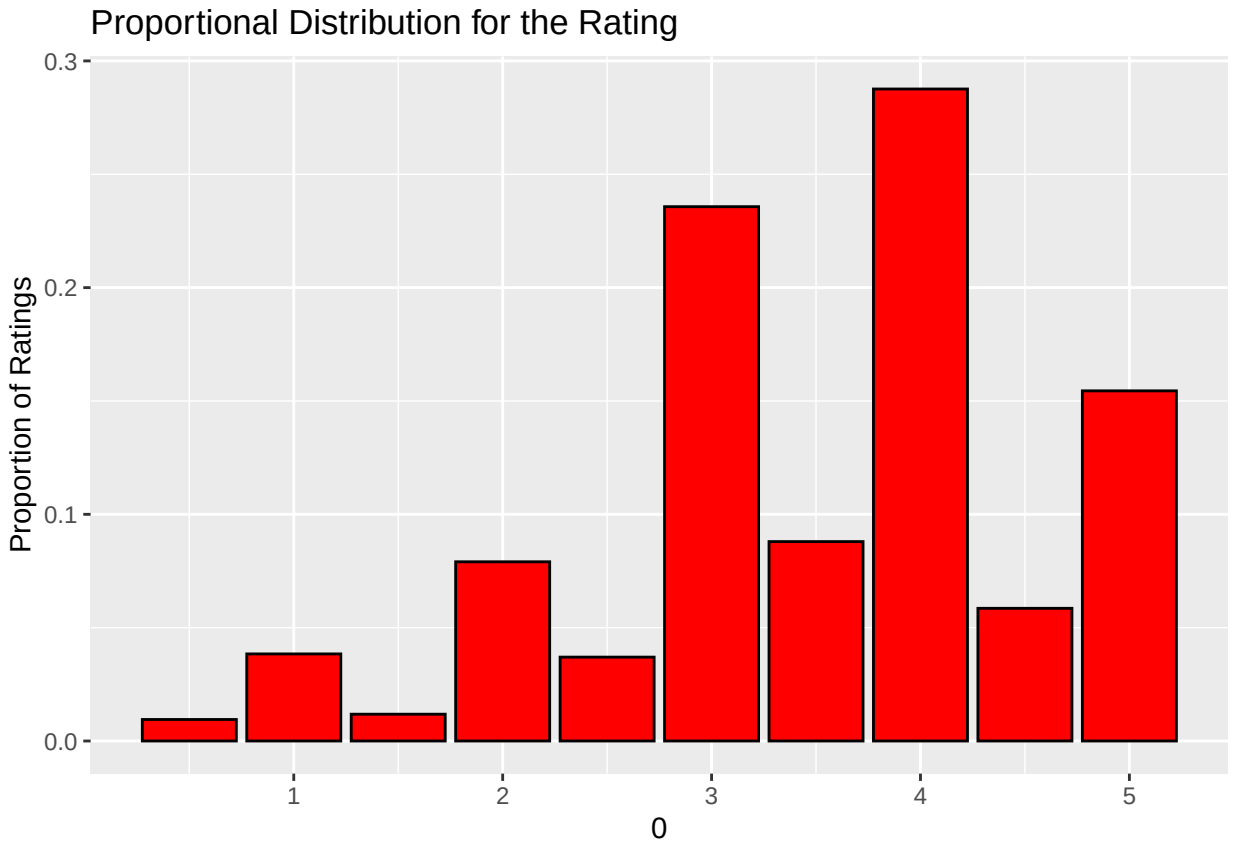
```
edx %>% group_by(rating) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) #this is ordered in terms of most common to least common rating
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
```

```
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

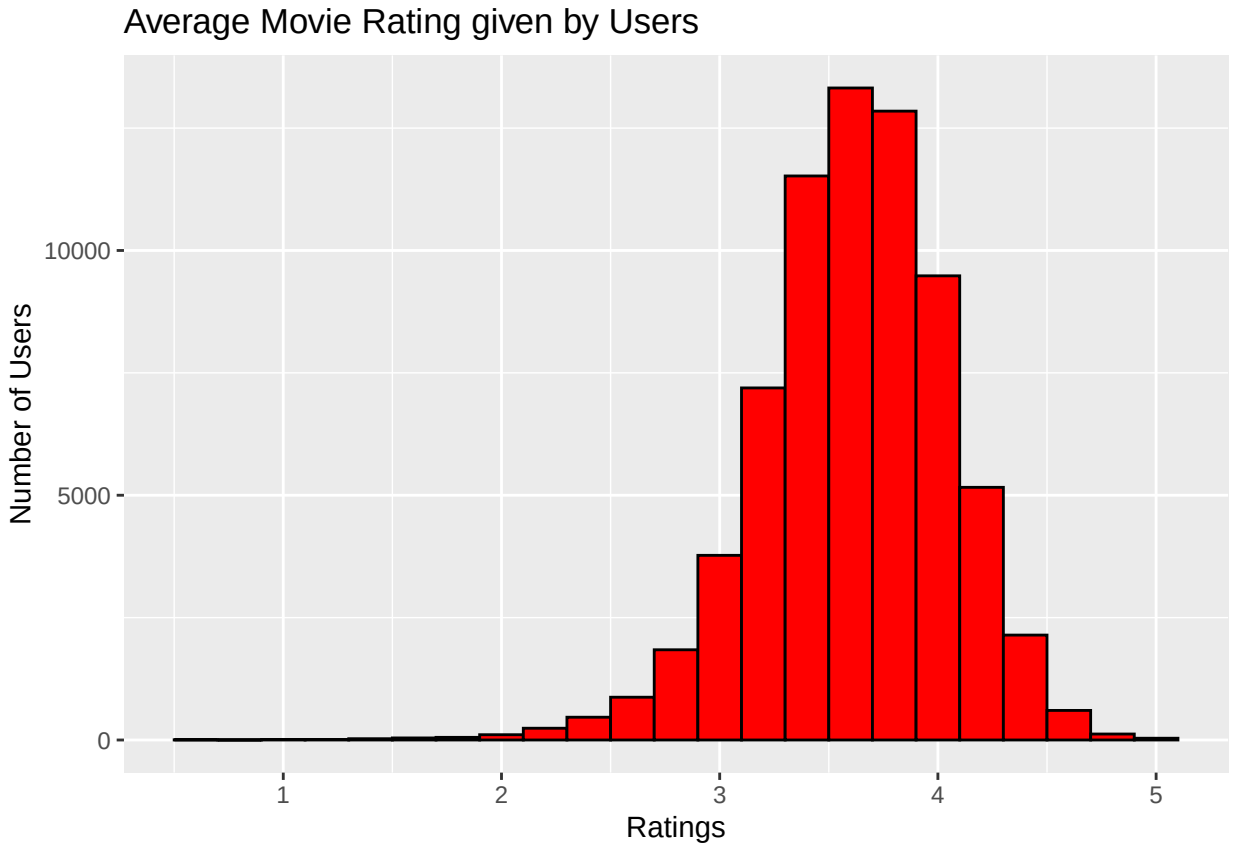
We can see from this table that *4*, *3*, and *5* are the most commonly given ratings and that non-natural number ratings are less common. It is helpful to visualize this distribution as a bar chart:

```
edx %>%
  ggplot(aes(rating, y = ..prop..)) +
  geom_bar(color = "black", fill = "red") +
  scale_x_continuous(seq(0, 5, by = 0.5)) +
  ggtitle("Proportional Distribution for the Rating") +
  labs(x = "Rating", y = "Proportion of Ratings")
```



The bar chart helps to reinforce the earlier suggestions that ratings tended to trend above the overall mean rating (*3.512*), as the plot is skewed to the left. Still, if we want to better understand the distribution of ratings, computing some summary statistics (median, mean, and quartiles) regarding the number of ratings given by users and visualizing the distribution of the ratings themselves is helpful.

```
#Plotting the distribution of the ratings
edx %>%
  group_by(userId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(binwidth = 0.2, color = "black", fill = "red", bins = 40) +
  labs(x = "Ratings", y = "Number of Users") +
  ggtitle("Average Movie Rating given by Users")
```



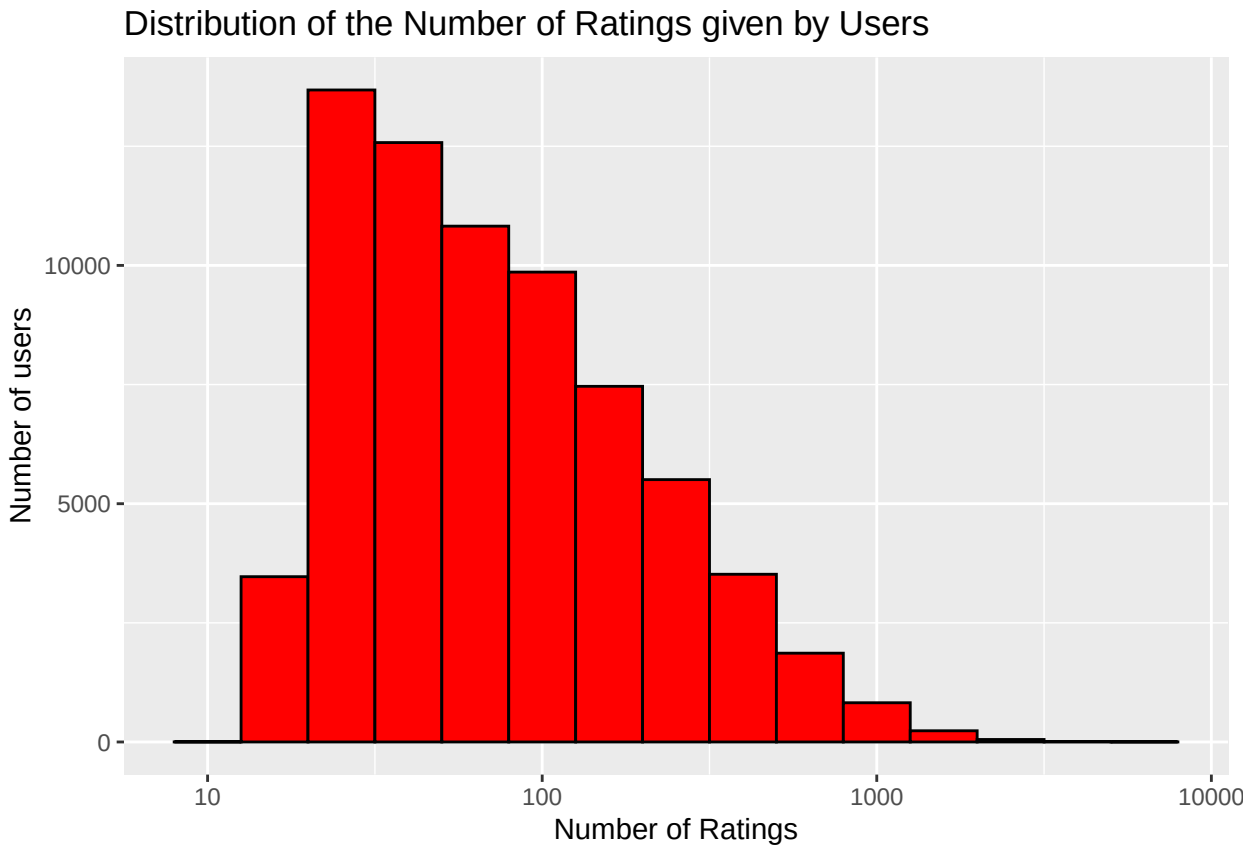
The plot is skewed to the left, meaning that users tend to give ratings above *2.5*, the middle rating. It is also notable that very few users gave a score lower than *2*, which suggests that models that take account of the trend to rate higher will see improvement over those that do not (assuming use of similar algorithms).

```
#Computing the summary statistics
user_rating_count <- edx %>%
  group_by(userId) %>%
  summarize(count = n())
summary(user_rating_count$count)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    10.0    32.0    62.0   128.8   141.0  6616.0
```

Note that the **median** (*62*) number of ratings given by a user is considerably lower than the **mean** (*128.8*). This implies that there is a non-trivial number of users who rated a lot more movies than the median would suggest. The **maximum** number of ratings by any user being *6616* also supports this. Notably, the **3rd quartile** being only *141* suggests that the upper 25% of users (ordered based on number of ratings given) includes several users who rated far more movies than any within the bottom 75%. To better see this distribution, observe the following plot:

```
#Visualizing the distribution of the number of ratings given by each user.
user_rating_count %>%
  ggplot(aes(count)) +
  geom_histogram(binwidth = 0.2, color = "black", fill = "red", bins = 40) +
  scale_x_log10() +
  labs(x = "Number of Ratings", y = "Number of users") +
  ggtitle("Distribution of the Number of Ratings given by Users")
```



The plot shows that thousands of users reviewed over *100* movies each and at least a few hundred reviewed *1000* or more times. Still, the three largest groupings of rating-volumes all fall below *100*. Note that there are small groups of films that received over *1000* reviews or fewer than *11* reviews. To make these metrics more precise:

```
#Determining the movies with 1000+ ratings and counting how many there are.
most_rated <- edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  filter(count >= 1000)
nrow(most_rated)
```

```
## [1] 1902
```

```
#Determining the movies with 10 or fewer ratings and counting how many
least_rated <- edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  filter(count <= 10)
nrow(least_rated)
```

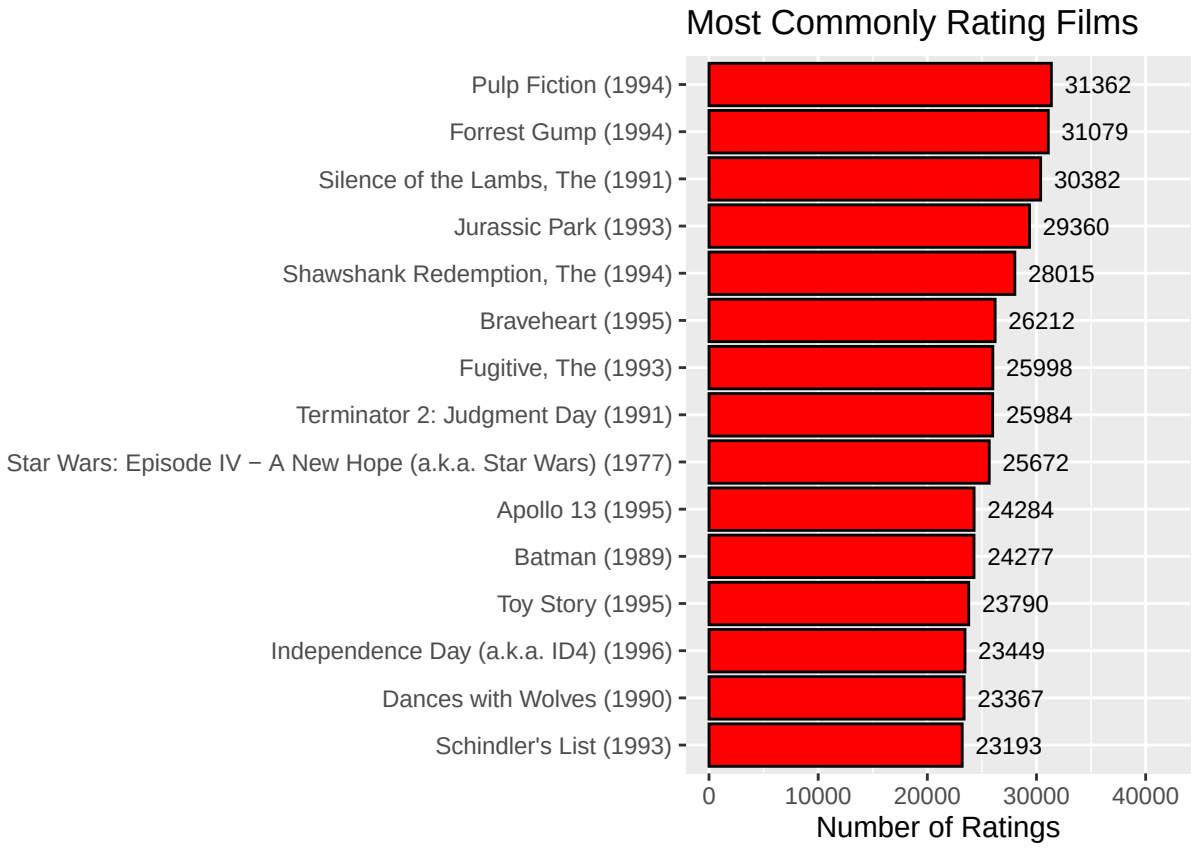
```
## [1] 1139
```

There are *1902* movies that received 1000 or more ratings and *1139* with 10 or fewer reviews. Examining all of the films at either extreme would likely be excessive but it may be helpful to see which films are really at each end of the spectrum. To that end, the following plot shows the 15 most often reviewed films.

```
#Determining which movies are the 15 most commonly rated.
most_rated_15 <- edx %>%
  group_by(title) %>%
```

```
  summarize(count = n()) %>%
  top_n(15, count) %>%
  arrange(desc(count))

#Visualizing the top 15 most rated movies.
most_rated_15 %>%
  top_n(15) %>%
  ggplot(aes(reorder(title, count), count)) +
  geom_bar(color = "black", fill = "red", stat = "identity") +
  ggtitle("Most Commonly Rating Films") +
  coord_flip(y = c(0, 42000)) +
  labs(y = "Number of Ratings", x = "") +
  geom_text(aes(label = count), hjust = -0.2, size = 3)
```

```
## Selecting by count
```



So far, I have not talked about **genres**. There are 797 difference genres listed, many of which are comprised of a combination of parts of one another. Though it is possible to separate each term and locate the base terms, doing so is technologically intensive (and in this case, restrive). Thus, unless shown to be necessary (which it is not for any of the forthcoming models), I will not be addressing them further.

Results

Now that we have a solid understanding of the **edx** data set, we can begin to build models for predicting movie ratings. As noted in the first section, six linear regression models and one matrix-factorization model will be constructed. In doing so, potential sources of bias originating from particular films, users, and time periods will be accounted for. Since, however, the **validation** set is being reserved for testing only the final

model, the first step in our model development is to divide **edx** into new testing and training sets.

Creating New Test and Train Sets As with the original partitioning of **movielens**, I will assign 90% of the data from **edx** to the train set, **train_edx**, and the remaining 10% to the test set, **test_edx**.

```
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
index <- createDataPartition(edx$movieId, times = 1, p = 0.9, list = FALSE)
train_edx <- edx %>%
  slice(index)
test_edx <- edx %>%
  slice(-index)
```

Before proceeding with the first model, a few quality checks must be performed on both **train_edx** and **test_edx**. First, the dimensions of both sets must be confirmed; **train_edx** should have 8,100,051 rows and 6 columns, and **test_edx** should have 900,004 rows and 6 columns.

```
dim(train_edx)
```

```
## [1] 8100051       6
```

```
dim(test_edx)
```

```
## [1] 900004      6
```

It is also important to make sure that the users and movies listed in **test_edx** are also listed in **train_edx**. If any user or movie appears in **test_edx** but not in **train_edx**, it will be removed.

```
test_edx <- test_edx %>%
  semi_join(train_edx, by = "movieId") %>%
  semi_join(train_edx, by = "userId")
```

It is worthwhile to look at the first few rows of both sets in order to ensure that they are formatted identically. Though this will not ensure that no issues appear anywhere in either set, should any such issue exist, this preliminary check can be helpful in identifying potential problems.

```
head(train_edx)
```

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

```
head(test_edx)
```

```
##    userId movieId rating  timestamp                             title
## 1:      1     588      5  838983339                     Aladdin (1992)
```

```
## 2:      2     780      3  868244698 Independence Day (a.k.a. ID4) (1996)
## 3:      2    1049      3  868245920   Ghost and the Darkness, The (1996)
## 4:      2    1356      3  868244603      Star Trek: First Contact (1996)
## 5:      3    1288      3 1133571035            This Is Spinal Tap (1984)
## 6:      4     208      3  844416866                    Waterworld (1995)
##                                        genres
## 1: Adventure|Animation|Children|Comedy|Musical
## 2:                 Action|Adventure|Sci-Fi|War
## 3:                            Action|Adventure
## 4:            Action|Adventure|Sci-Fi|Thriller
## 5:                              Comedy|Musical
## 6:                      Action|Adventure|Sci-Fi
```

Both sets appear to be formatted correctly and do not indicate any issues at this time.

Finally, a check for any NA values in **test_edx** or **train_edx** is very important. If any such values are present, they must either be removed or replaced with an appropriate substitue value.

```
any(is.na(train_edx))
```

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

```
any(is.na(test_edx))
```

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

Fortunately, there are no NA values in either set. So, both sets are now ready to be applied to model making. Before any models can be made, however, a metric for determining success is needed. For this report, **Root Mean Square Error** will be used to measure the quality of all the models.

Root Mean Square Error (RMSE) Given that RMSE provides a numeric measurement of how far a given prediction for an objects value is from the true value, it is an appropriate metric by which to judge the quality of a rating prediction model. For clarity sake, the RMSE function used here is based on:

$$\sqrt{(sum_{i=1}^{n}(((predicted_i - actual_i)^2)/n)}$$

Since the *mean* function in R will take account of the n in the denominator, the RMSE function will be defined as:

```
rmse <- function(true_rating, predicted_rating){
  sqrt(mean((true_rating - predicted_rating)^2))
}
```

With the RMSE function established, we are now prepared to create and test the first linear regression model.

Model #1: Naive Mean The first model is based on computing the overall average rating (**mu**) and does not make any adjustments for who provided each rating, when it was given, or to which film it is associated with. This is a *Naive Mean Model* and is likely to fall quite short of the target RMSE of 0.8649 or less.

First, **mu** must be computed from the **train_edx** data set before being compared to the actual ratings found in the *rating* column of the **test_edx** data set. Then, **mu** can be directly inputted into the RMSE function.

```
#Calculating the average rating for movies (ignores any possible biases)
mu <- mean(train_edx$rating)
mu #This is the average rating of the movies in the training set.
```

```
## [1] 3.512407
```

```
#Calculating the RMSE score based on mu alone.
model1_rmse <- rmse(test_edx$rating, mu)
model1_rmse #This should be approximately 1.059.
```

```
## [1] 1.05931
```

As predicted, the first model is considerably above the goal RMSE score.

Since it will be helpful to compare each subsequent model's RMSE score, I will create a table of each model's title and associated RMSE. As each model is created, its information will be added to the table.

```
model_comparison <- data.frame(method = "Naive Mean",
                               RMSE = model1_rmse)
```

The rating a given movie receives is predominantly based on two factors: how a movie is generally perceived and the preferences of a particular reviewer. A movie that is said to be truly excellent (or for that matter, really terrible) may have its ratings biased by viewers preconceptions about the film. For example, someone who does not enjoy a film that they had been told was quite good may be more disappointed (and thereby more inclined to rate more harshly) than someone who had no particular expectations about the movie's quality prior to seeing it. Similarly, a person who is squeamish may rate a film that depicts graphic injuries (such as war or medical films) lower than others regardless of the film's quality. Though these examples are speculative, the possibility of such cases makes it worthwhile to take account of potential movie and user biases.

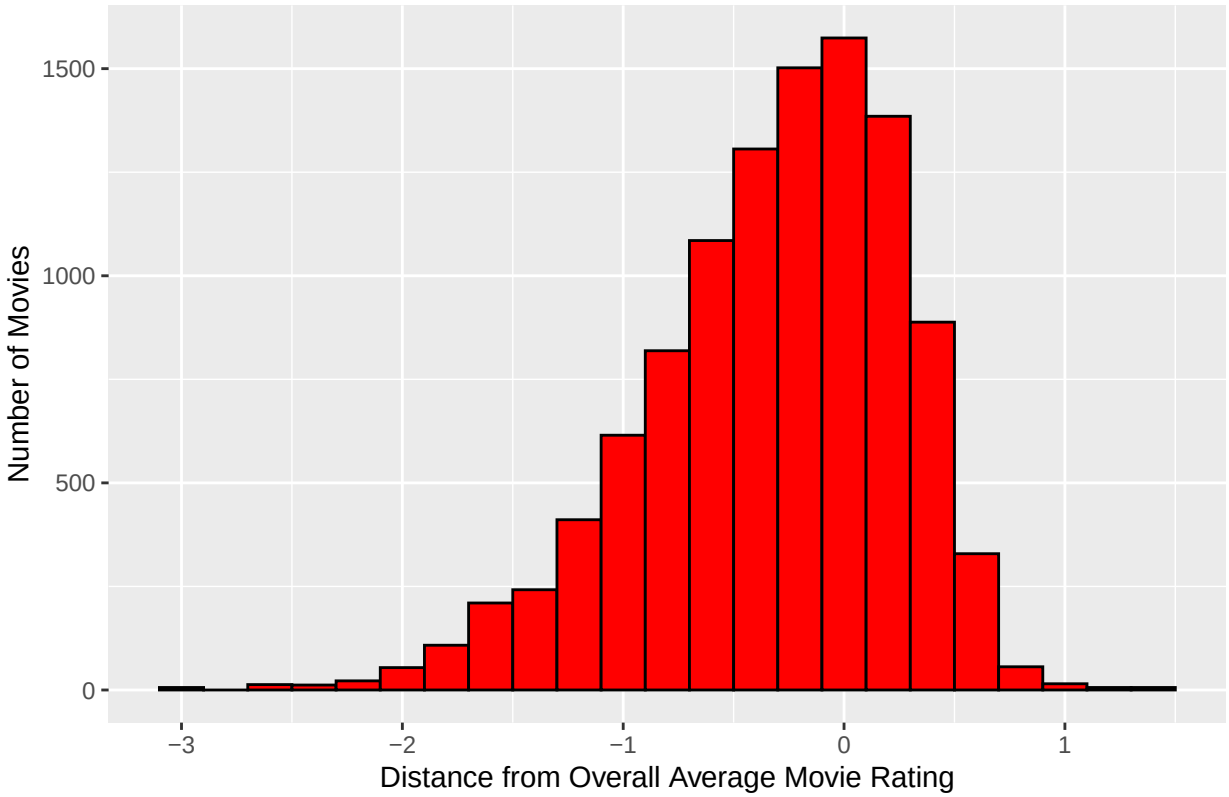
Model 2 addresses movie-specific bias but does not consider user-specific bias. Model 3 does the opposite, considering user bias but not adjusting for user bias. Thus, they shall be referred to as the **Mean + Movie Bias** and **Mean + User Bias** models, respectively.

Model 2: Mean + Movie Bias In order to create the model, a measurement of movie bias must first be made. Since we know that **mu** is the true average rating across all movies in **train_edx**, we can measure movie bias by taking the average distance between each rating in **train_edx** and **mu**.

```
movie_bias <- train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu)) #b_i is the average distance between the ratings and mu.
```

To get a better idea of the distribution of distances between the ratings and **mu**, consider the following plot:

```
#Plotting the movie_bias b_1 scores (distance each movie's rating was from the overall average rating)
movie_bias %>%
  ggplot(aes(b_i)) +
  geom_histogram(binwidth = 0.2, color = "black", fill = "red", bins = 40) +
  labs(x = "Distance from Overall Average Movie Rating",
       y = "Number of Movies") +
  ggtitle("")
```



We can see that while, as expected, the largest groups of films have ratings with plus/minus 1 of **mu**, there are non-trivially many which have a rating below mu-1. Thus, it is likely that the inclusion of movie bias adjustments will improve the model.

We can now make the predictions for the ratings using:

```
model2_predicted <- mu + test_edx %>%
  left_join(movie_bias, by = "movieId") %>%
  .$b_i
```

Using these predictions, the RMSE can be calculated using the same function as with the first model.

```
model2_rmse <- rmse(model2_predicted, test_edx$rating) #this computes the RMSE score
model_comparison <- bind_rows(model_comparison,
                              data.frame(method = "Movie Bias Model",
                                         RMSE = model2_rmse)) #adds model 2 to the table.
model2_rmse #So that the RMSE score can be seen here.
```

```
## [1] 0.9420862
```

Model 2 has yielded an RMSE score of 0.9420862, an improvement of 0.1172237 over the first model. Still, this is significantly above the target goal of 0.8649 or lower.

The next model will not take account of any movie-specific bias. Instead, it will focus on addressing user-specific bias.

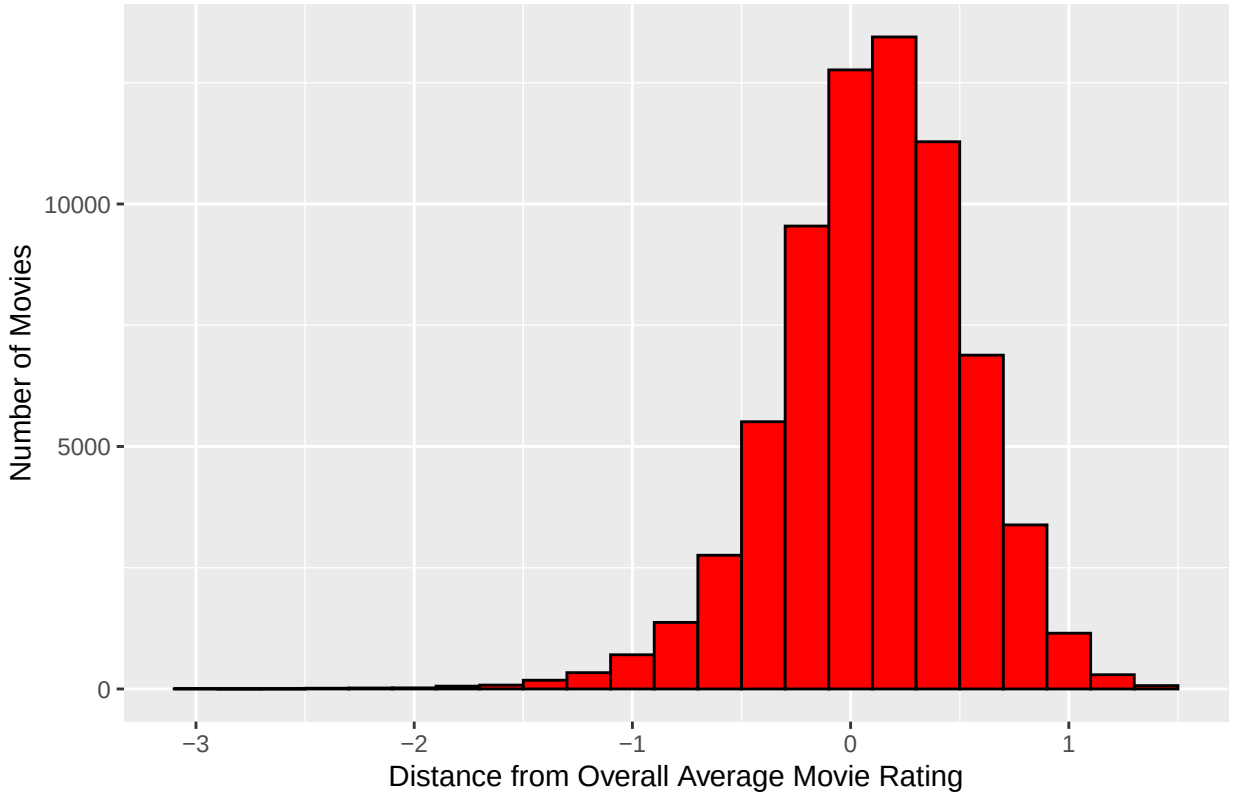
Model 3: Mean + User Bias A very similar approach to that used for model 2 can be applied to the making of model 3. A measure of distance between each rating and **mu** will again be used but instead of grouping using **movieId**, **userId** will be used.

```
user_bias <- train_edx %>%
  group_by(userId) %>%
```

```
  summarize(b_u = mean(rating - mu)) #b_u is the average distance between ratings and mu.
```

As before, a visualization of the distances will give insight into how user bias has affected the ratings.

```
user_bias %>%
  ggplot(aes(b_u)) +
  geom_histogram(binwidth = 0.2, color = "black", fill = "red", bins = 40) +
  labs(x = "Distance from Overall Average Movie Rating",
       y = "Number of Movies") +
  ggtitle("")
```



Notably, this plot appears to be more centered around 0 than the plot from model 2.There does seem to be a slightly larger number of movies above 0 than below, but this is a smaller spread than observed in the previous model. As such, it is unlikely that model 3 will be an improvement over model 2.

Following the earlier method, we can now predict the ratings and compute the RMSE score for model 3.

```
model3_predicted <- mu + test_edx %>%
  left_join(user_bias, by = "userId") %>%
  .$b_u #this is making the predictions for ratings.

model3_rmse <- rmse(model3_predicted, test_edx$rating) #computing the RMSE
model_comparison <- bind_rows(model_comparison,
                              data.frame(method = "User Bias Model",
                                         RMSE = model3_rmse)) #adding model 3 to the table.
```

If we examine the table comparing all of the models thus far, we can see that model 3 not only is not an improvement on model 2 but actually does worse, coming in at 0.03580404 higher.

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

method	RMSE
Naive Mean	1.0593098
Movie Bias Model	0.9420862
User Bias Model	0.9778902

Still, model 3 is a considerable improvement on the first model, as it provided a reduction of 0.08141961 in RMSE score. So, including user bias in the model alongside movie bias will help to further reduce the RMSE score.

****Model 4: Mean + User Bias + Movie Bias** Instead of handling movie and user biases in isolation of one another, model 4 will address them in tandem with one another. To do this, a new measurement, **b_w**, which will provide a measurement of distance based on both sources of potential bias, must be made. Beyond adopting the new measurement(s), the method for making model 4 is largely unchanged from the previous two models.

```
user_movie_biases <- train_edx %>%
  left_join(movie_bias, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_w = mean(rating - mu - b_i))
```

Note that **b_w** is computed by subtracting both the movie bias variable (from model 2) and **mu** from each rating, before taking the average of these differences. So, while it is possible to base model 4 on model 3, this process is better thought of as an iteration on model 2.

When making predictions for this model, we have to include both **b_i** (movie_bias) and **b_w** (user bias post-inclusion of movie bias). So, our prediction is now based on:

$$prediction = mu + b_i + b_w$$

Thus, the prediction process now looks like this:

```
model4_predicted <- test_edx %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_movie_biases, by = "userId") %>%
  mutate(prediction = mu + b_i + b_w) %>%
  .$prediction
```

With the predictions made, we can compute the RMSE score and add model 4 to the **model_comparison** table.

```
model4_rmse <- rmse(model4_predicted, test_edx$rating)
model_comparison <- bind_rows(model_comparison,
                              data.frame(method = "Movie + User Biases Model",
                                         RMSE = model4_rmse))
model_comparison %>% knitr::kable()
```

method	RMSE
Naive Mean	1.0593098
Movie Bias Model	0.9420862
User Bias Model	0.9778902
Movie + User Biases Model	0.8645122

Model 4 yields an RMSE of 0.8645122, which is 0.0003878177 below the target goal. So, model 4 would be an acceptable model but we can actually improve this further through regularization.

Model 5: Regularized Mean + User Bias + Movie Bias As shown in the analysis section, there are over 1000 movies that received fewer than 11 ratings and a non-trivial number of users who reviewed very few films. All previous models did not make any adjustments for this, making them and their RMSE scores susceptible to cases such as a film only having a single rating of 5 or a user only reviewing a few films. To correct for this, I will make use of the tuning parameter **lambdas** in order to penalize the relative weight of these outlier films and users. Since it is presently unknown what the optimal value for **lambdas** is, i.e. which value will minimize the RMSE score, multiple values must be considered.

```
lambdas <- seq(0, 10, 0.25)
```

The inclusion of the lambda values does require that **b_i** and **b_w** be redefined to incorporate a lambda term. This, in effect, means that both **b_i** and **b_w** (renamed as **b_u**) must be divided by:

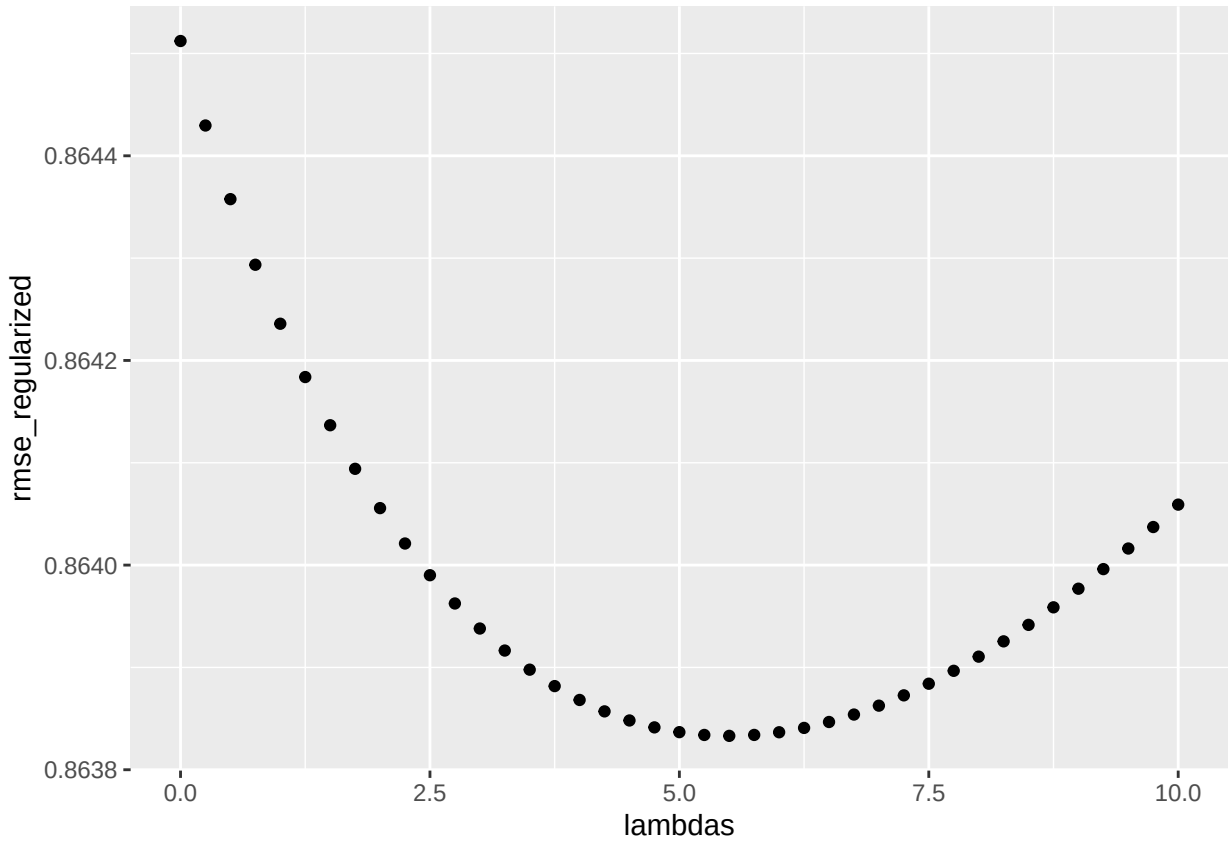
$$n + lambda$$

Thus, the new method for model making goes as follows:

```
rmse_regularized <- sapply(lambdas, function(l){
  mu <- mean(train_edx$rating) #this is the overall mean rating.

  b_i <- train_edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + l)) #this is movie bias.

  b_u <- train_edx %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + l)) #this is movie + user bias.

  predicted_rating <- test_edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(prediction = mu + b_i + b_u) %>%
    .$prediction #this makes the prediction

  return(rmse(predicted_rating, test_edx$rating))
})
```

Note that **rmse_regularized** is a list of RMSE scores; it is not yet clear which lambda provides the lowest RMSE score nor is it known what that score is. To see this, consider the following plot comparing lambda values to RMSE scores.

```
qplot(lambdas, rmse_regularized)
```



The plot suggests that the optimal lambda is slightly above 5 but this is not sufficiently precise. To gain greater precision, consider:

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

```
## [1] 5.5
```

We now know that the optimal lambda is 5.5, which yields an RMSE value of 0.8638331.

Adding this new model to the table, we can see a small improvement over model 4.

```
model5_rmse <- min(rmse_regularized)
#Adding model 5 to the table
model_comparison <- bind_rows(model_comparison,
                              data.frame(method = "Regularized User + Movie Biases Model",
                                         RMSE = model5_rmse))
model_comparison %>% knitr:: kable()
```

method	RMSE
Naive Mean	1.0593098
Movie Bias Model	0.9420862
User Bias Model	0.9778902
Movie + User Biases Model	0.8645122
Regularized User + Movie Biases Model	0.8638331

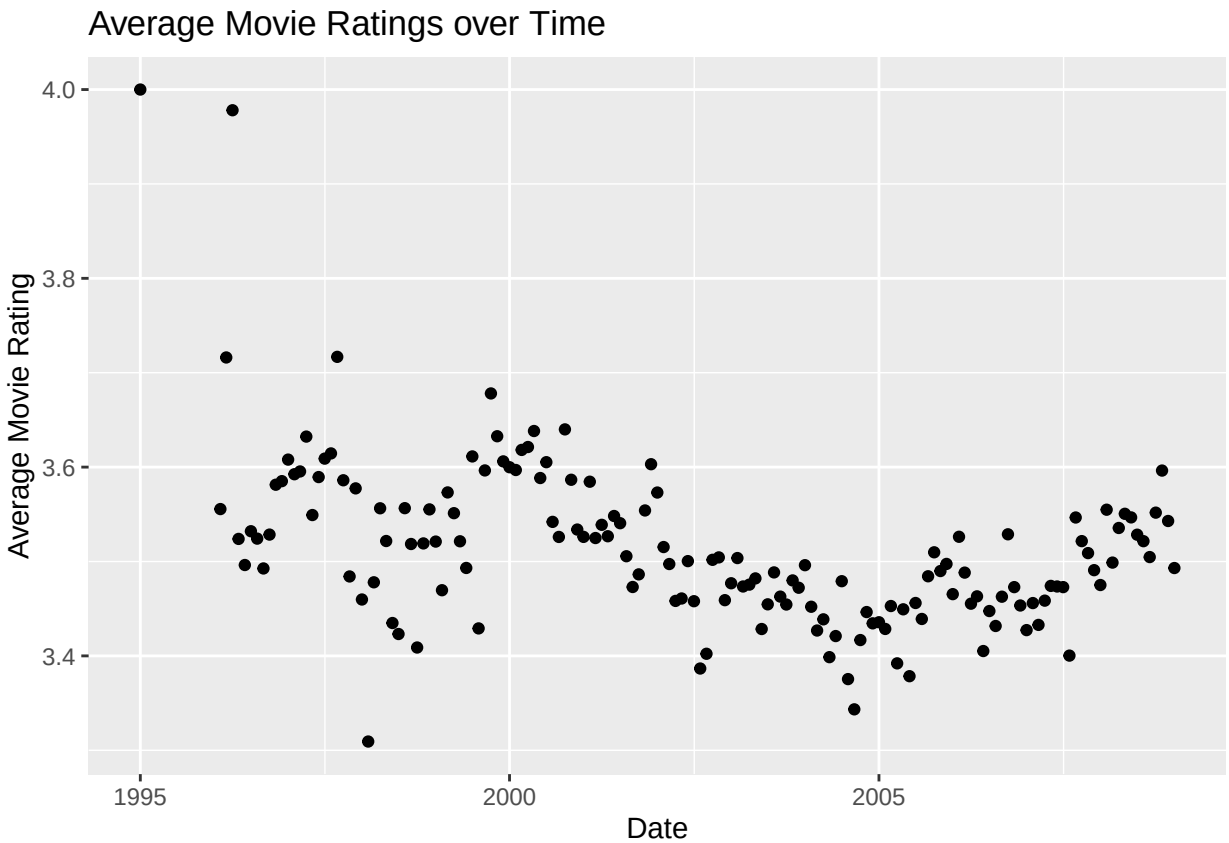
With an RMSE score of 0.8638331, we are solidly below the target score. Before moving on from a linear regression approach, however, one more potential source of bias should be considered: *time*.

Model 6: Regularized Mean + User Bias + Movie Bias + Time Bias In this final linear regression model, consideration of when a rating was given will be factored into the model. To do this, I will first create duplicates of the testing and training subsets of **edx**. Using this new pair of sets, I will convert the *timestamp* column data in each set to a more readable format.

```
train_edx_norm_dates <- train_edx %>%
  mutate(round_dates = round_date(as_datetime(timestamp), unit = "month"))
test_edx_norm_dates <- test_edx %>%
  mutate(round_dates = round_date(as_datetime(timestamp), unit = "month"))
```

Note that the dates have been rounded to the nearest month. This has been done for two reasons: (1) a day-by-day basis would have been too technologically intensive and (2) as will be seen by the following plot, time is not a major contributing factor to rating trends.

```
#Plotting the average rating over time (based on monthly average rating.)
train_edx_norm_dates %>%
  group_by(round_dates) %>%
  summarize(average_rating = mean(rating)) %>%
  ggplot(aes(round_dates, average_rating)) +
  geom_point() +
  xlab("Date") +
  ylab("Average Movie Rating") +
  ggtitle("Average Movie Ratings over Time")
```



The vast majority of points indicate a rating between 3.4 and 3.6, with the most extreme of exceptions coming prior to 2000. Bearing in mind that **mu** is approximately 3.5, this is to be expected. There is some variation in the average rating over time but it does appear to be relatively small. So, while some improvement in the model should be expected, it is likely to be fairly small.

As before, the model is to be regularized using the **lambdas** tuning parameter.

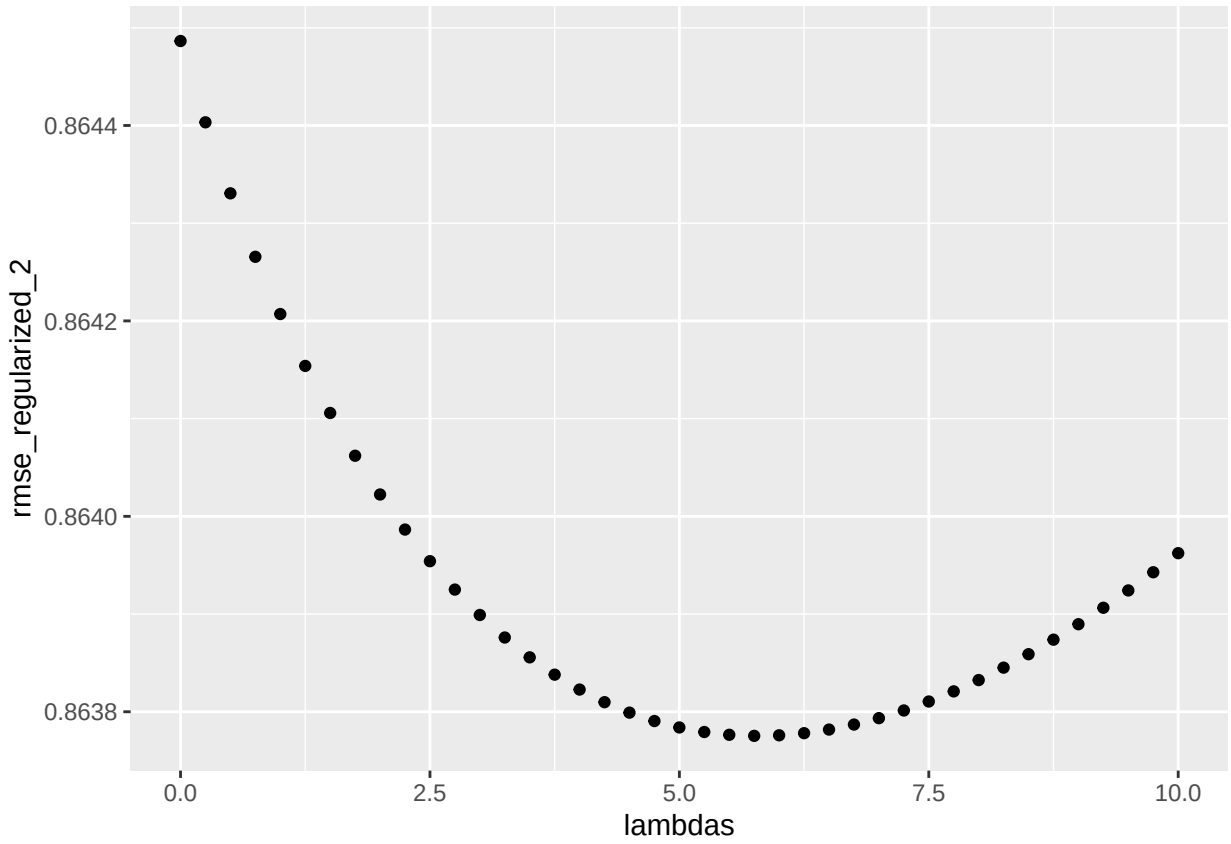
```
lambdas <- seq(0, 10, 0.25)
```

Note that the process for making the predictions follows the same pattern as with model 6, with the only alterations being those made to include time.

```
rmse_regularized_2 <- sapply(lambdas, function(l){
  mu <- mean(train_edx$rating)

  b_i <- train_edx_norm_dates %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu) / (n() + l)) #this is movie bias.

  b_u <- train_edx_norm_dates %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + l)) #this is movie + user bias.

  b_t <- train_edx_norm_dates %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(round_dates) %>%
    summarize(b_t = sum(rating - b_i - b_u - mu) / (n() + l)) #this is the time term.

  predicted_rating <- test_edx_norm_dates %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_t, by = "round_dates") %>%
    mutate(prediction = mu + b_i + b_u + b_t) %>%
    .$prediction

  return(rmse(predicted_rating, test_edx$rating))
})
```

Again, this leaves us with a list of RMSE values associated with each of the lambda values but does not tell us what the optimal lamdba is. To determine this, consider the following plot.

```
#Showing the lambdas compared to their rmse scores.
qplot(lambdas, rmse_regularized_2)
```




```
#Determining the lambda with the smallest RMSE
lambda <- lambdas[which.min(rmse_regularized_2)]
lambda
```

```
## [1] 5.75
```

This yields an optimal lambda value of 5.75 with a corresponding RMSE score of 0.8637754. Looking at the table of models, it can be seen that this is an improvement but only very slightly.

```
#Computing the lowest RMSE
model6_rmse <- min(rmse_regularized_2)
#Adding model 6 to the table
model_comparison <- bind_rows(model_comparison,
                              data.frame(method = "Regularized User + Movie + Time Biases Model",
                                         RMSE = model6_rmse))
#Showing the table
model_comparison %>% knitr::kable()
```

method	RMSE
Naive Mean	1.0593098
Movie Bias Model	0.9420862
User Bias Model	0.9778902
Movie + User Biases Model	0.8645122
Regularized User + Movie Biases Model	0.8638331
Regularized User + Movie + Time Biases Model	0.8637754

Model 7: Recosystem and Matrix Factorization The **recosystem** library in R is designed to make recommendation systems using matrix factorization. It requires that both the training and testing sets be

formatted as *sparse matrix triplets*. So, the first step is to convert **train_edx** and **test_edx** into the correct form (*Note*: Since time data is not relevant here, I have gone back to using the original testing and train sets instead of those used in model 6.).

```
train_edx_sparMatTrip <- with(train_edx,
                              data_memory(user = userId, item_index = movieId, rating = rating))
test_edx_sparMatTrip <- with(test_edx,
                             data_memory(user = userId, item_index = movieId, rating = rating))
```

Note that the movieIds are being used to index the elements in each matrix. As will be relevant in the coming steps, p is used to refer to the user and$q$ refers to the item_index.

From here, an R object of class *Recosys* must be created.

```
rec_obj <- Reco()
class(rec_obj)
```

```
## [1] "RecoSys"
## attr(,"package")
## [1] "recosystem"
```

All of the remaining steps until it comes time to compute the RMSE score are based around three argument functions: *tune*, *train*, and *predict*.

In order to tune **rec_obj**, the *opts* arguments (described below) must be specified:

1. *dim*: This is the number of latent factors that are expected to be found in the data.
2. *costp_12*: This is the regularization cost for the user factors.
3. *costq_12*: This is the regularization cost for the item_index factors.
4. *lrate*: This controls the rate at which the gradient will decline.
5. *niter*: This is controlling how many iterations there are.

In particular, matrix factorization is not necessarily linear, hence the need to control the rate at which the gradient changes (this looks like a quarter of an ellipse). Thus, the tuning looks like:

```
rec_tuner <- rec_obj$tune(train_edx_sparMatTrip,
                          opts = list(dim = c(10, 30),
                                      costp_12 = c(0.01, 0.1),
                                      costq_12 = c(0.01, 0.1),
                                      lrate = c(0.01, 0.1),
                                     niter = 15))
```

One advantage of the **Recosystem** library is that the same tuning parameters can be carried over to the training of the model and the making of predictions.

```
#Training the model
rec_obj$train(train_edx_sparMatTrip,
              opts = c(rec_tuner$min, nthread = 4, niter = 30))
```

```
## iter      tr_rmse          obj
##    0       0.9830   1.1053e+07
##    1       0.8755   8.9904e+06
##    2       0.8429   8.3423e+06
##    3       0.8208   7.9592e+06
##    4       0.8044   7.6885e+06
##    5       0.7919   7.4977e+06
##    6       0.7819   7.3535e+06
##    7       0.7733   7.2371e+06
##    8       0.7661   7.1453e+06
##    9       0.7598   7.0685e+06
```

```
##    10       0.7542   7.0016e+06
##    11       0.7492   6.9422e+06
##    12       0.7448   6.8953e+06
##    13       0.7407   6.8508e+06
##    14       0.7370   6.8132e+06
##    15       0.7337   6.7796e+06
##    16       0.7307   6.7497e+06
##    17       0.7277   6.7203e+06
##    18       0.7251   6.6967e+06
##    19       0.7226   6.6745e+06
##    20       0.7203   6.6522e+06
##    21       0.7181   6.6329e+06
##    22       0.7161   6.6151e+06
##    23       0.7143   6.5991e+06
##    24       0.7125   6.5850e+06
##    25       0.7108   6.5699e+06
##    26       0.7093   6.5577e+06
##    27       0.7078   6.5448e+06
##    28       0.7064   6.5328e+06
##    29       0.7052   6.5226e+06
```

```
#Making the predictions
rec_predictions <- rec_obj$predict(test_edx_sparMatTrip, out_memory())
```

Now, just as it was with previous models, the RMSE score can be directly computed.

```
#Computing the RMSE score
model7_rmse <- rmse(rec_predictions, test_edx$rating)

#Adding model 7 to list of models
model_comparison <- bind_rows(model_comparison,
                              data.frame(method = "Recosystem Matrix Factorization",
                                         RMSE = model7_rmse))
#Showing the table
model_comparison %>% knitr::kable()
```

method	RMSE
Naive Mean	1.0593098
Movie Bias Model	0.9420862
User Bias Model	0.9778902
Movie + User Biases Model	0.8645122
Regularized User + Movie Biases Model	0.8638331
Regularized User + Movie + Time Biases Model	0.8637754
Recosystem Matrix Factorization	0.7843913

With an RMSE under 0.8, the Recosystem model is by far the best model. Accordingly, it will now be applied to the **edx** and **validation** sets to get a final model.

Model 8: The Final Model (Recosystem) Outside of changing the sources of the data from **train_edx** and **test_edx** to **edx** and **validation**, the process for making this final model is identical to model 7. Thus, I have presented it as a single block of code without much elaboration.

```
#Making Sparse Matrix Triplet versions of Edx and Validation.
edx_sparMatTrip <- with(edx, data_memory(user = userId, item_index = movieId,
```

```
                                          rating = rating))
valid_sparMatTrip <- with(validation, data_memory(user = userId,
                                                  item_index = movieId, rating = rating))

#Making an r object of class "Recosys" (to be used subsequently)
rec_obj_valid <- Reco()

#Tuning the recosys object rec_obj_valid
rec_tuner <- rec_obj_valid$tune(edx_sparMatTrip,
                          opts = list(dim = c(10, 30),
                                      costp_12 = c(0.01, 0.1),
                                      costq_12 = c(0.01, 0.1),
                                      lrate = c(0.01, 0.1),
                                      niter = 15))
#Training the model
rec_obj_valid$train(edx_sparMatTrip,
              opts = c(rec_tuner$min, nthread = 4, niter = 30))
```

```
## iter      tr_rmse          obj
##    0       0.9739   1.2044e+07
##    1       0.8718   9.8736e+06
##    2       0.8374   9.1593e+06
##    3       0.8156   8.7401e+06
##    4       0.8004   8.4658e+06
##    5       0.7886   8.2680e+06
##    6       0.7789   8.1195e+06
##    7       0.7709   8.0003e+06
##    8       0.7641   7.9008e+06
##    9       0.7584   7.8235e+06
##   10       0.7533   7.7560e+06
##   11       0.7488   7.6988e+06
##   12       0.7447   7.6506e+06
##   13       0.7410   7.6056e+06
##   14       0.7376   7.5675e+06
##   15       0.7345   7.5353e+06
##   16       0.7317   7.5025e+06
##   17       0.7290   7.4742e+06
##   18       0.7266   7.4491e+06
##   19       0.7243   7.4264e+06
##   20       0.7222   7.4060e+06
##   21       0.7202   7.3870e+06
##   22       0.7184   7.3695e+06
##   23       0.7166   7.3529e+06
##   24       0.7151   7.3385e+06
##   25       0.7135   7.3240e+06
##   26       0.7121   7.3117e+06
##   27       0.7108   7.2990e+06
##   28       0.7094   7.2874e+06
##   29       0.7083   7.2776e+06
```

```
#Making the predictions
rec_predictions <- rec_obj_valid$predict(valid_sparMatTrip, out_memory())

#Calculating the rmse
```

```r
model8_rmse <- rmse(rec_predictions, validation$rating)

#Adding the final model to the table of model comparisons
model_comparison <- bind_rows(model_comparison,
                              data.frame(method = "Final Model (Recosystem)",
                                         RMSE = model8_rmse))
#Reviewing all of the models
model_comparison %>% knitr::kable()
```

method	RMSE
Naive Mean	1.0593098
Movie Bias Model	0.9420862
User Bias Model	0.9778902
Movie + User Biases Model	0.8645122
Regularized User + Movie Biases Model	0.8638331
Regularized User + Movie + Time Biases Model	0.8637754
Recosystem Matrix Factorization	0.7843913
Final Model (Recosystem)	0.7810403

By using the Recosystem library, we have obtained a final RMSE score of 0.7809, which is well below the target goal.

Conclusion

The goal of this work was to develop at least one machine learning model that could predict movie ratings and obtain an RMSE score less than 0.8649. Besides the final model to which **validation** was applied, four models reached this benchmark (showing in order of worst to best):

1. Movie + User Bias
2. Regularized User + Movie Bias
3. Regularized User + Movie + Time Bias
4. Recosystem Matrix Factorization

By using linear regression models, an RMSE of 0.8637754 was obtained. This was further improved to approximately 0.781 by applying matrix factorization to the **edx** and **validation** sets. Still, this is not to say that further improvements could not be made.

Perhaps the most notable source of possible improvement comes from the **genres** data that was provided by the **movielens** data set. As noted earlier, the impact that particular genres had on rating trends was not explored here and its inclusion into model 6 might have further shrunk the RMSE. That said, it seems doubtful that its addition would have resulted in a similarly sizable reduction as was achieved via matrix factorization. Also, though a sizable number of tuning parameters were explicitly controlled when using the **Recosystem** library, I did not adjust all of the permitted parameters. What effect further specification might have had is at this point purely speculative but might warrant further investigation were this report to be revisited later.

A final note is that while this report does include multiple models capable of achieving the desired benchmark RMSE, it does not presently allow for a user to enter their own data (preferred genres, time of release, etc.) in order to obtain an actual recommendation. Were this feature to be added, I would transition the document from being formatted for R Markdown to a Shiny document.

References In the course of developing this report, I consulted the following sources.

- https://cran.r-project.org/web/packages/recosystem/recosystem.pdf
- https://github.com/yixuan/recosystem
- https://rdrr.io/cran/recosystem/man/tune.html
- Book: R for Everyone: Advanced Analytics and Graphics (Jared P. Lander)

Notes This report is presented as partial completion of Harvard University's Professional Certificate in Data Science. Portions of the code included here were provided by the staff of the program as a part of the eight and ninth courses (*Machine Learning* and *Capstone*). For more information about the program, please see: Harvardx Data Science Certificate.