MovieLens Recomendation Project

HarvardX PH125.9x Capstone

David Vance

2024-04-30

Background and Introduction

The MovieLens database is a set of approximately 10 million movie ratings, composed of almost 70,000 users rating over 10,000 movies. Each record in the database contains 6 columns, which store the userid, movieid, rating, movie name, movie genre, and a timestamp of when the review was originally posted.

The purpose of this project is to design a machine learning algorithm to suggest movies to users that they might like. These suggestions can be based on many things, and as you will see, we will utilize the user's own past ratings, as well as the general sentiment of the movie and the movie's genre, and also the age of the movie. The database has initially been split into two sets. The first set is the "edx" data set, which contains 9,000,055 records. The second data set is the "final_holdout_test" data set, and will only be used to test the chosen machine learning model at the end of the project. In order to design a model to test on the final_holdout_test data set, we will first split the edx set into a training and testing set. Our various models will then be trained on the edx_train set, and then tested against the edx_test set. Each model will result in an RMSE value when tested versus the edx_test data set, and the model that results in the lowest RMSE will be chosen as the final model, and tested against the final_holdout_test data set.

In brief: Our goal in this project is to develop a machine learning algorithm that will achieve a final RMSE of <0.8649 on the final_holdout_test dataset.

```
# Our final goal
goal <- 0.8649
```

The code to create the edX and final_holdout_test datasets has been provided in the course material, and is as follows:

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

# Note: this process could take a couple of minutes

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

library(tidyverse)
library(caret)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
```

```r
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

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

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

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

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

Analysis

As mentioned, the first thing we need to do is to split the edX data set into training and testing data. We will use a seed here so that the analysis is repeatable. We also need to ensure that there are no users or movies in the testing set that are not in the training set. The following code will create our training and testing data sets from the original edx data set.

```
# Set seed for repeatability
set.seed(1, sample.kind="Rounding")

# Split up edx dataset into training and testing sets
test_index <- createDataPartition(edx$rating, times=1, p=0.1, list=FALSE)
edx_train <- edx[-test_index,]
edx_test_temp <- edx[test_index,]

# Ensure that there are no movies in the testing set that aren't in the training set
edx_test <- edx_test_temp %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
removed <- anti_join(edx_test_temp, edx_test)
edx_train <- rbind(edx_train, removed)

# Remove unneeded variables from memory
rm(edx_test_temp, removed, test_index)
```

Determining Model Accuracy

Before we start building models, we need to set up a function that we will use to determine how accurate our models are. We will use Root Mean Square Error, which is the square root of the average of all squared differences between predicted and actual results. The function is created as follows:

```
# Defining our error function that will determine model accuracy
RMSE <- function(ratings, y_hat) {
  sqrt(mean((ratings-y_hat)^2))
}
```

Model 1 - The Average Rating

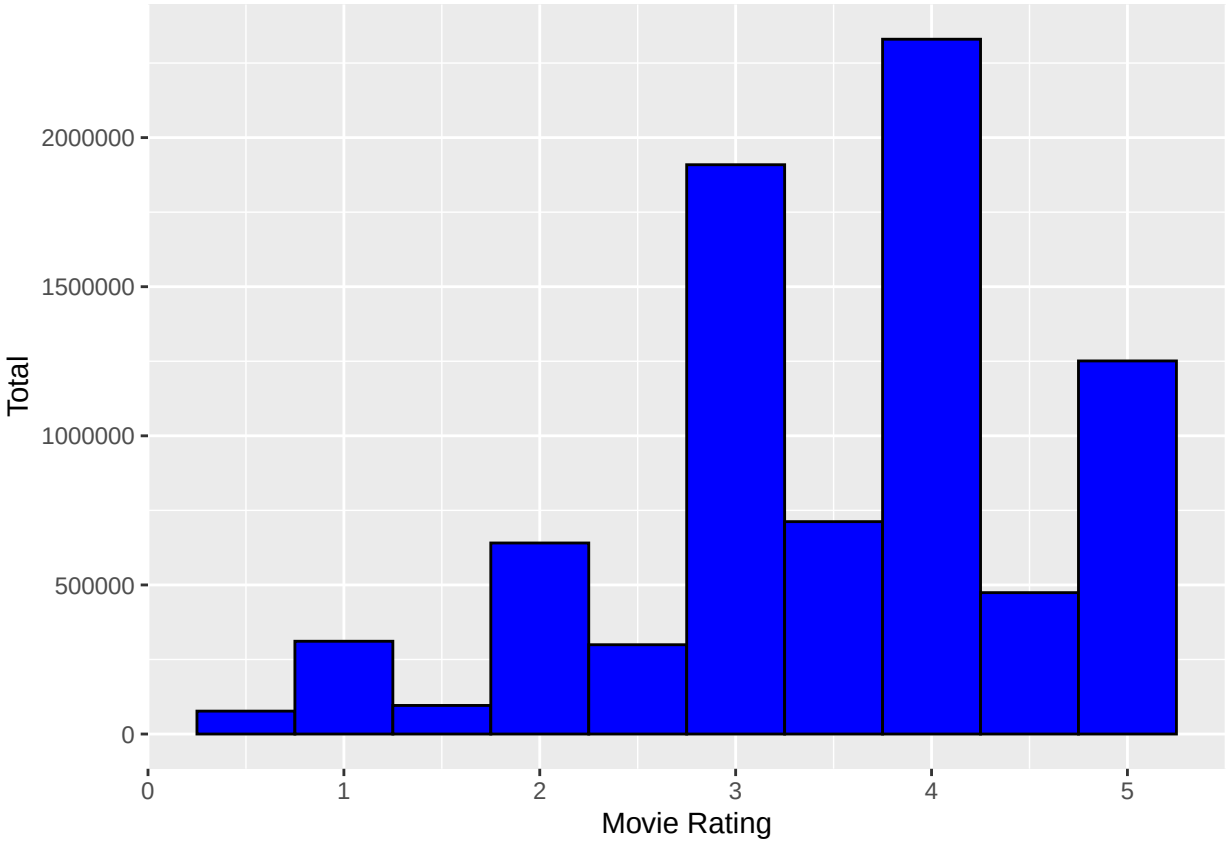
Now we will begin testing models to see if we can predict whether a particular user will enjoy a particular movie. The first model we will test is the very simplest: the average rating. We will determine the average rating of all of the movies in the edx_train dataset, and predict that every movie in the ex_test data set will have that average rating. This is obviously incorrect, but let's see just HOW incorrect it will be as a first step toward model building.

```
# Calculate the average rating across the entire edcx_train data set
average_rating <- mean(edx_train$rating)
average_rating
```

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

Let's visualize all of the ratings in the training set to see if our average rating looks correct.

```
# Graph the distribution of movie ratings in the edx_train data set
edx_train %>% ggplot(aes(rating)) +
  geom_histogram(color="black", fill="blue", bins=10) +
  xlab("Movie Rating") + ylab("Total")
```



It's obvious from the graph that whole number ratings are given much more often. However, given that the two most numerous ratings are 3 and 4, an average rating of ~3.5 does make sense.

Finally, we can test how accurate our first model is using the RMSE function.

```
# If we predict that every movie in the test set has the average rating, how far off are we?
RMSE_1 <- RMSE(edx_test$rating, average_rating)
RMSE_1
```

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

We will store our results in a table to compare them all, and continue to add to the table as we go.

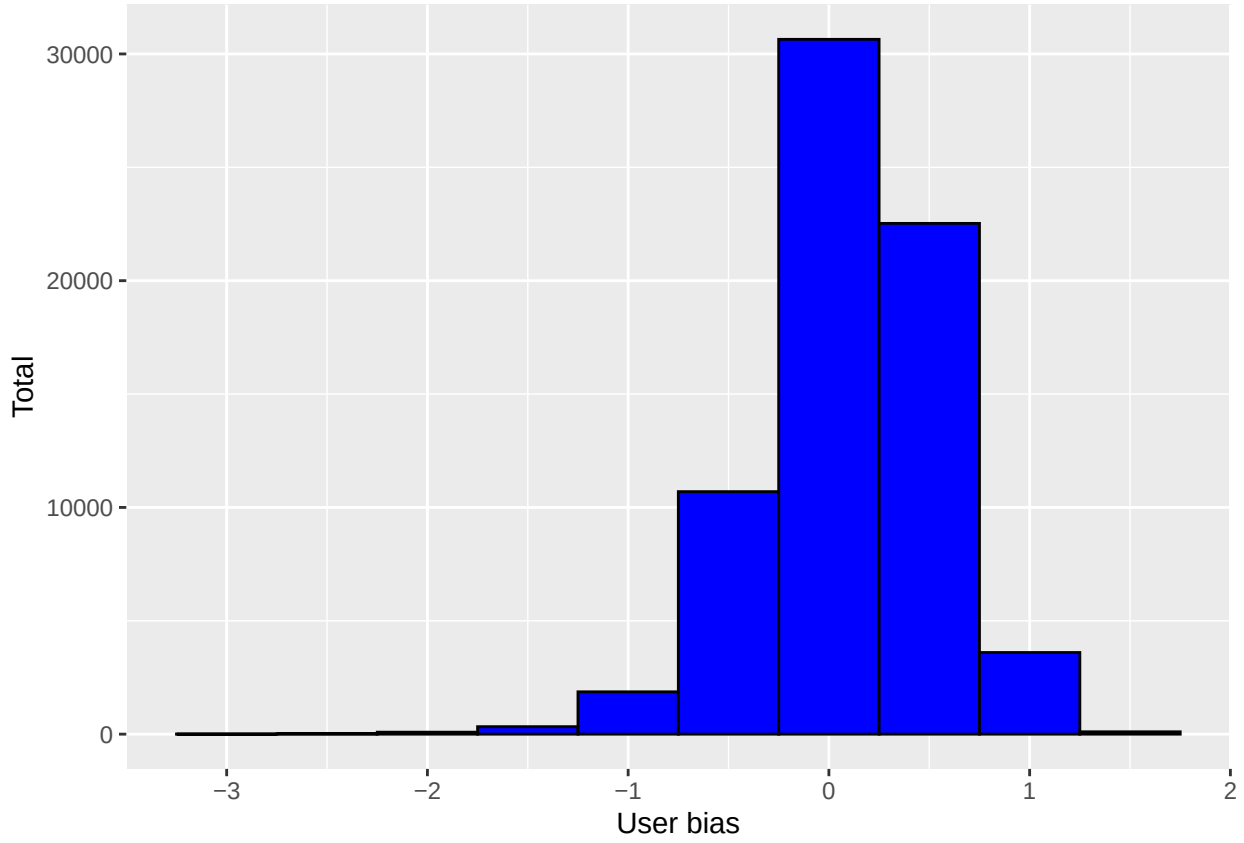
```
# Update results table
results_table <- tibble(Model = c("1. Average Rating","", "Goal"),
                        RMSE = c(RMSE_1, "", goal)) %>% knitr::kable()

results_table
```

Model	RMSE
1. Average Rating	1.06005370222409
Goal	0.8649

Our model that predicts ratings using just the average rating of the entire training set is not very accurate, off by more than 1 on average. This is of course not unexpected. Not every movie is of the same quality, and not every person rating the movies will have the same judgment. These biases can be corrected for to design new models, which take into consideration how tough or easy each individual person is at rating movies (aka, a user effect), and another that takes into consideration the quality of each individual movie (aka a movie effect). Let's look at these skews, in isolation, and see if one adds more variability than the other.

User Bias

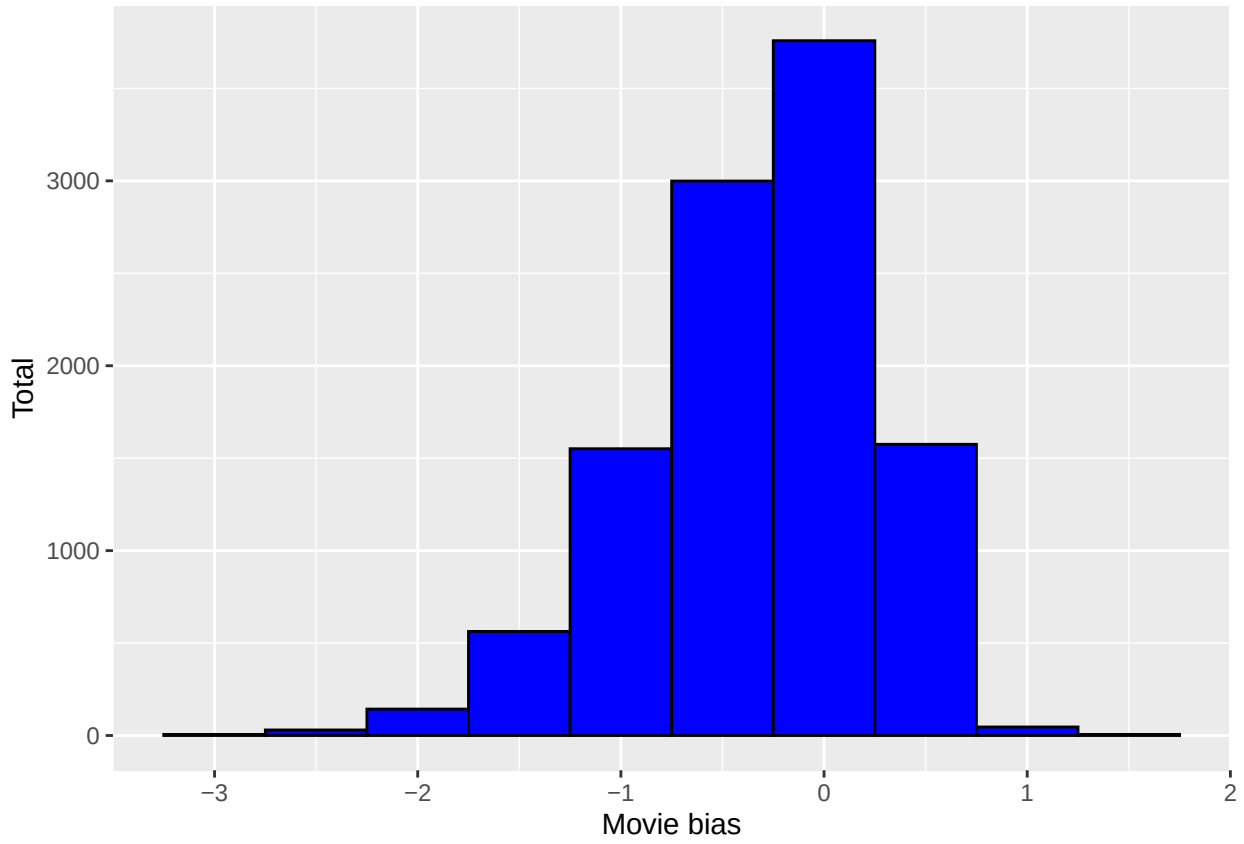
```
# Determine user bias in isolation
user_bias <- edx_train %>% group_by(userId) %>%
  summarize(u_bias = mean(rating-average_rating))

# Graph user bias
user_bias %>% ggplot(aes(u_bias)) +
  geom_histogram(color="black", fill="blue", bins=10) +
  xlab("User bias") + ylab("Total")
```



As can be seen, some users are generally tough graders, while others less stringent. As a result, an individual user's ratings might have an average well above or below the average for the entire set.

Movie Bias

```
# Determine movie bias in isolation
movie_bias <- edx_train %>% group_by(movieId) %>%
  summarize(m_bias = mean(rating-average_rating))

# Graph movie bias
movie_bias %>% ggplot(aes(m_bias)) +
  geom_histogram(color="black", fill="blue", bins=10) +
  xlab("Movie bias") + ylab("Total")
```



Interestingly, while user effects were skewed slightly toward higher ratings than average, movie effects seemed to skew towards lower ratings than average. This is likely due to the abundance of sparsely watched, badly rated movies, while frequently rated good movies help drag the overall average up. It's also noticeable that movie effects are greater than user effects, so we will try to correct for movie effects first.

Model 2 - Movie Effects Added

Using the bias for each individual movie as calculated just above, we will now add in those biaes to the average rating, and see if that produces a lower RMSE of ratings.

```
# Adding movie effects to average rating
y_hat2 <- edx_test %>% left_join(movie_bias, by = "movieId") %>%
  mutate(pred=(average_rating + m_bias)) %>% pull(pred)

RMSE_2 <- RMSE(edx_test$rating, y_hat2)
RMSE_2
```

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

This model, with an RMSE of 0.94296 is considerably better than model 1, which uses the average rating only. This is a good start to our model building. Let's add this model to our table.

```
# Update results table
results_table <- tibble(Model = c("1. Average Rating", "2. Movie Effects Added",
                                  "", "Goal"), RMSE = c(RMSE_1, RMSE_2, "",
                                  goal)) %>% knitr::kable()

results_table
```

Model	RMSE
1. Average Rating	1.06005370222409
2. Movie Effects Added	0.942961498004501
Goal	0.8649

Model 3 - User Effects Added

Now let's determine how well our model does when we incorporate individual user effects on top of movie effects. For this, we will need to recalculate the user bias that we did above to back out the already accounted for movie effects.

```
# Recalculating user bias after accounting for movie bias
user_bias <- edx_train %>% left_join(movie_bias, by="movieId") %>%
  group_by(userId) %>%
  summarize(u_bias = mean(rating-average_rating-m_bias))
```

Now we can test this model's ability to predict ratings on the edx_test dataset.

```
# Accounting for movie and user bias, how does our model do?
y_hat3 <- edx_test %>% left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  mutate(pred=(average_rating + m_bias + u_bias)) %>%
  pull(pred)

RMSE_3 <- RMSE(edx_test$rating, y_hat3)
RMSE_3
```

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

As can be seen, adding in user effects results in another large increase in RMSE, bringing the error down to 0.86468, which actually already exceeds our goal of 0.8649. That is an excellent start to our model building. For now, let's add our newest result to the table.

```
# Update results table
results_table <- tibble(Model = c("1. Average Rating", "2. Movie Effects Added",
                                  "3. Movie and User Effects Added", "", "Goal"),
                        RMSE = c(RMSE_1, RMSE_2, RMSE_3, "", goal)) %>% knitr::kable()

results_table
```

Model	RMSE
1. Average Rating	1.06005370222409
2. Movie Effects Added	0.942961498004501
3. Movie and User Effects Added	0.86468429490229
Goal	0.8649

Although we have achieved an RMSE lower than our goal, there is no guarantee that this model will also work as well on the final_holdout_test set. As such, we will try to improve our model even further before testing it on the final test set.

Age Bias

In the same way that a movie's rating can be affected by its quality (movie effects) and the user doing the rating (user effects), it can also be affected by the age of the movie. Movies that are very new will have been seen, and rated, by fewer people. Similarly, very old movies will also be less frequently viewed and rated. As well, there may be a ratings effect for the age of a movie. Older movies (but not too old!) may be rated higher, while audiences tend to think less favorably of newer movies.

In order to determine the age of the movie, we will have to determine the release year of each movie. The title of each movie has the year it was released, in parentheses, at the end of the title. Because of this pattern, we know that the year a movie is released is the 5th last to 2nd last characters in the title column. We can thus extract the release year of the movie from the title of the movie, and then subtract it from the current year, to calculate the movie's age. We can then mutate our data sets (edx_test, edx_train, and final_holdout_test) to include a column entitled "age" that will store the number of years ago a movie was released, using the current year of 2024 as the baseline.

Note: we will not use the final_holdout_test set here for model creation, but are only modifying it for later use in the final model validation.

```
# Updating the data sets to add a column for release year
edx_train <- edx_train %>%
  mutate(age = 2024 - as.numeric(str_sub(title, start=-5, end=-2)))

edx_test <- edx_test %>%
  mutate(age = 2024 - as.numeric(str_sub(title, start=-5, end=-2)))

final_holdout_test <- final_holdout_test %>%
  mutate(age = 2024 - as.numeric(str_sub(title, start=-5, end=-2)))
```

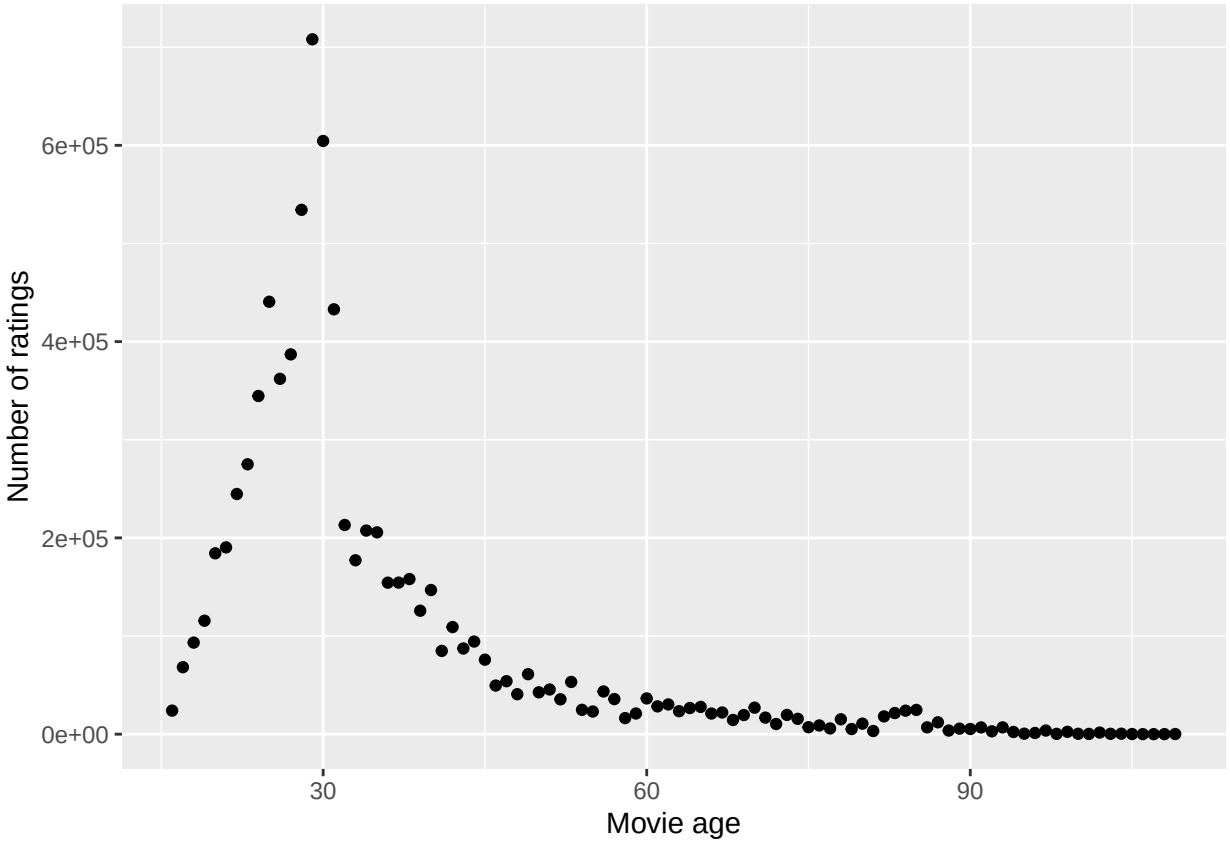
Let's look at part of the edx_train table to ensure that our mutation was correctly done.

```r
# Confirming that data mutation was successful
head(edx_train)
```

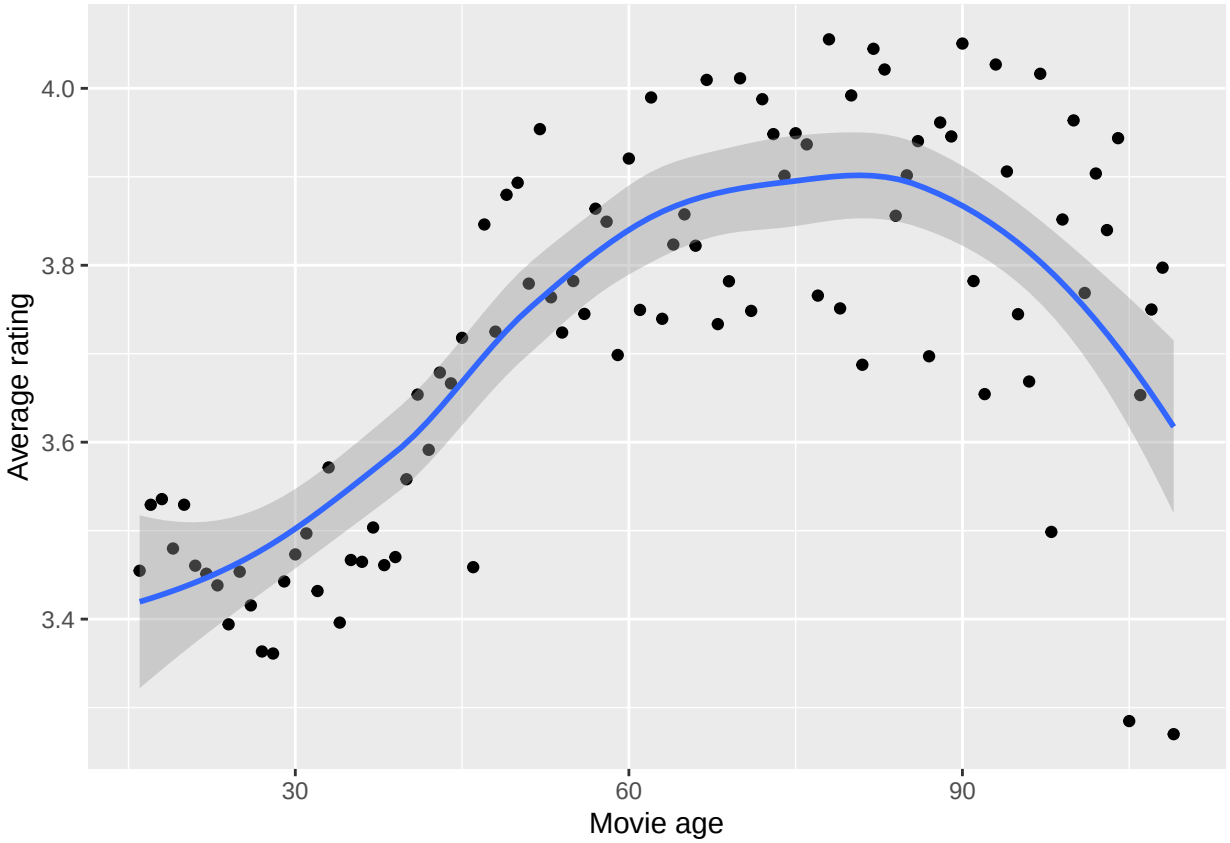
```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
## 8      1     356      5 838983653           Forrest Gump (1994)
##                         genres age
## 1                Comedy|Romance  32
## 4  Action|Drama|Sci-Fi|Thriller  29
## 5       Action|Adventure|Sci-Fi  30
## 6 Action|Adventure|Drama|Sci-Fi  30
## 7       Children|Comedy|Fantasy  30
## 8      Comedy|Drama|Romance|War  30
```

The mutation appears correct. Now that we have added the age of the movie to the data sets, we'll calculate the total number of ratings for movies of a given age, and also how the average rating of a movie is affected by its age.

```r
# Calculating age-related metrics in the edx_train data set
ratings_per_age <- edx_train %>% group_by(age) %>%
  summarize(ratings_count = n(), average_rating = mean(rating))

# Graphing number of ratings for movies of a certain age
ratings_per_age %>% ggplot(aes(age, ratings_count)) + geom_point() +
  xlab("Movie age") + ylab("Number of ratings")
```




```
# Graphing average rating of movies based on age
ratings_per_age %>% ggplot(aes(age, average_rating)) + geom_point() +
  geom_smooth() + xlab("Movie age") + ylab("Average rating")
```



It is quite evident from the data that movie age is an important factor. Movies that are around 25-30 years old (and thus released in the mid to late 90s) have a much larger number of ratings than movies of other ages. As well, it also appears to be true that relatively older movies are considered to be of higher quality, with average ratings higher than contemporary movies. In fact, the curve fit of average rating per year shows that movies that around 75-80 years old (and therefore released in the late 1940s) are rated approximately a half a star better than the most recently released movies. Truly, they don't make 'em like they used to! As such, we will add movie age effects into our model, along with movie effects and user effects.

Model 4 - Age Effects Added

```
# If we also account for age bias, on top of movie and user biases, how does our model do?
age_bias <- edx_train %>%
  left_join(movie_bias, by="movieId") %>%
  left_join(user_bias, by="userId") %>% group_by(age) %>%
  summarize(a_bias = mean(rating-average_rating-m_bias-u_bias))

y_hat4 <- edx_test %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  left_join(age_bias, by="age") %>%
  mutate(pred=(average_rating + m_bias + u_bias + a_bias)) %>%
  pull(pred)

RMSE_4 <- RMSE(edx_test$rating, y_hat4)
RMSE_4
```

```
## [1] 0.8643301
```

We did improve the model, ever so slightly. Let's add it to our results table.

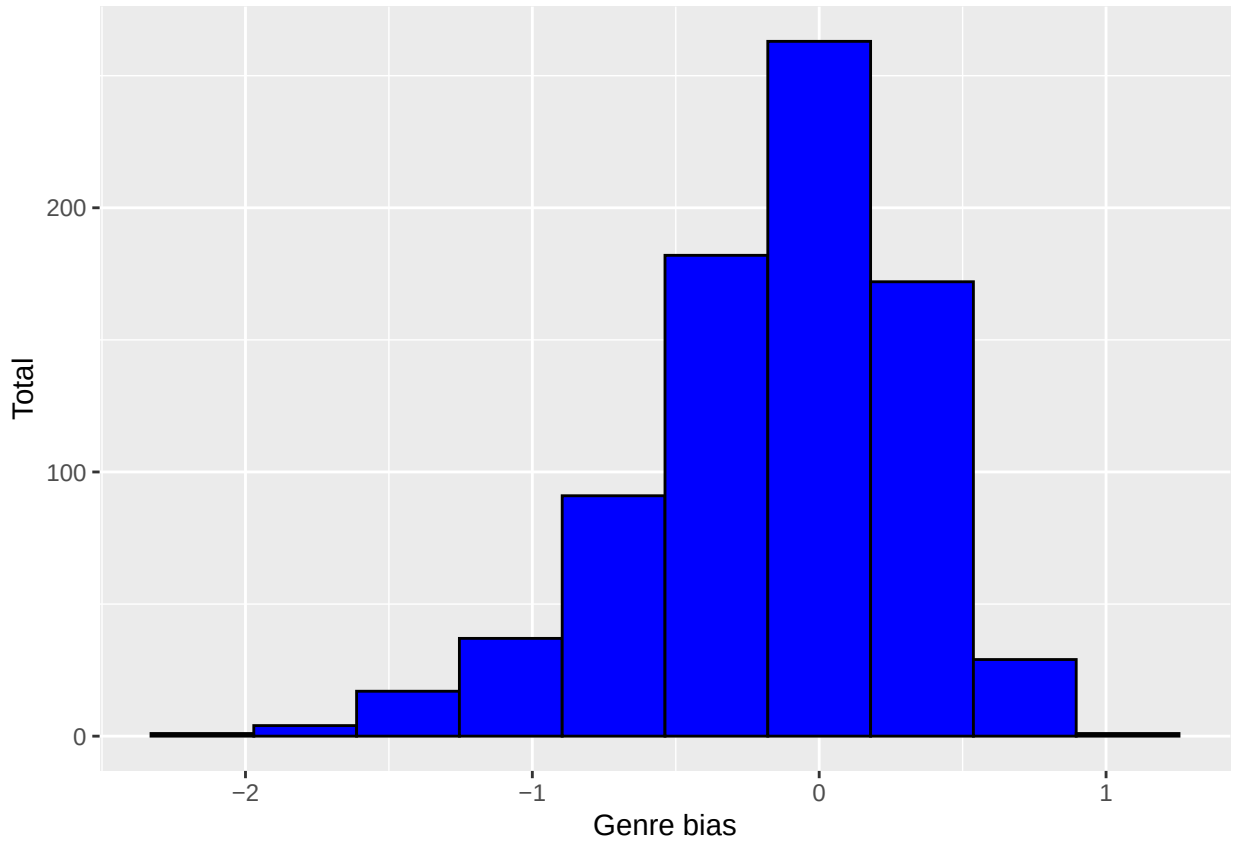
```
# Update results table
results_table <- tibble(Model = c("1. Average Rating", "2. Movie Effects Added",
                                  "3. Movie and User Effects Added",
                                  "4. Movie, User, and Age Effects Added",
                                  "", "Goal"),
                        RMSE = c(RMSE_1, RMSE_2, RMSE_3, RMSE_4, "", goal)) %>%
                        knitr::kable()

results_table
```

Model	RMSE
1. Average Rating	1.06005370222409
2. Movie Effects Added	0.942961498004501
3. Movie and User Effects Added	0.86468429490229
4. Movie, User, and Age Effects Added	0.864330070782133
Goal	0.8649

Genre Bias

Let's face it - not every type of movie is everyone's cup of tea. Some people may prefer horror movies, while others like to laugh, and still others like to be engaged in a mystery. It may well be the case that some movies, Oscar-bait type dramas, are rated more highly than others, for instance, slapstick comedies. Let's explore that, in isolation at first.

```
# Calculate genre bias in isolation
genre_bias <- edx_train %>% group_by(genres) %>%
  summarize(g_bias = mean(rating-average_rating))

# Graph genre bias
genre_bias %>% ggplot(aes(g_bias)) +
  geom_histogram(color="black", fill="blue", bins=10) +
  xlab("Genre bias") + ylab("Total")
```



Indeed, a large range of genre biases can be seen, with some genres averaging ratings of more than 2 stars lower than the average rating. Adding this effect into the model should help it be even more accurate.

Model 5 - Genre Effects Added

As before, we need to recalculate the genre bias from the in-isolation data above to account for the already calculated movie, user, and age biases. We will then apply our new model to the edx_test data set and see how accurate it is.

```
# Recalculate genre bias after accounting for previously determined biases
genre_bias <- edx_train %>%
  left_join(movie_bias, by="movieId") %>%
  left_join(user_bias, by="userId") %>%
  left_join(age_bias, by="age") %>%
  group_by(genres) %>%
  summarize(g_bias = mean(rating-average_rating-m_bias-u_bias-a_bias))

# How does adding genre effects affect the accuracy of the movie?
y_hat5 <- edx_test %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId") %>%
  left_join(age_bias, by="age") %>%
  left_join(genre_bias, by="genres") %>%
  mutate(pred=(average_rating + m_bias + u_bias + a_bias + g_bias)) %>%
  pull(pred)
```

```
RMSE_5 <- RMSE(edx_test$rating, y_hat5)
RMSE_5
```

```
## [1] 0.8640801
```

Once again, the model has been improved, even if just slightly. Every little bit counts! Let's add it to our table.

```
# Update results table
results_table <- tibble(Model = c("1. Average Rating", "2. Movie Effects Added",
                                  "3. Movie and User Effects Added",
                                  "4. Movie, User, and Age Effects Added",
                                  "5. Movie, User, Age, and Genre Effects Added",
                                  "", "Goal"),
                        RMSE = c(RMSE_1, RMSE_2, RMSE_3, RMSE_4, RMSE_5, "",
                                 goal)) %>% knitr::kable()

results_table
```

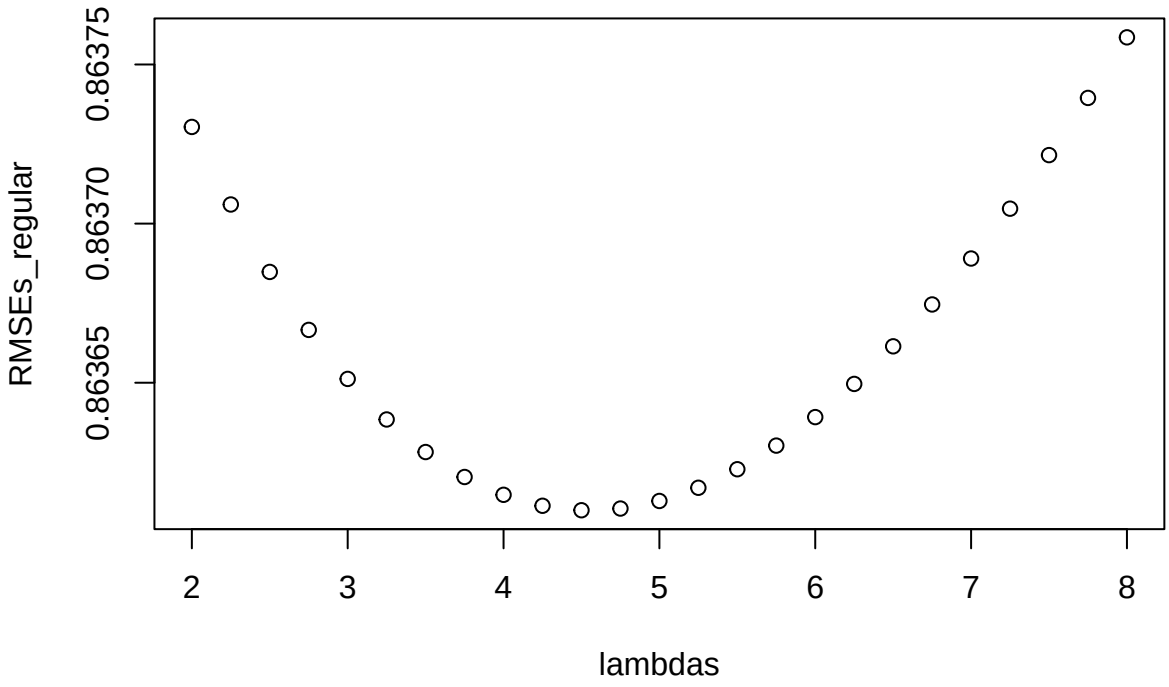
Model	RMSE
1. Average Rating	1.06005370222409
2. Movie Effects Added	0.942961498004501
3. Movie and User Effects Added	0.86468429490229
4. Movie, User, and Age Effects Added	0.864330070782133
5. Movie, User, Age, and Genre Effects Added	0.864080139682404
Goal	0.8649

Model 6 - Adding Regularization

For a further attempt at improvement of the model, we will now look to regularization. Regularization is a way to deal with the fact that not every movie, and not every user, in the database has the same number of ratings. For instance, movies that very few people have seen (and rated) may have a very high or very low rating that doesn't represent the "true quality" of that movie. Similarly, if a particular user only rates bad movies, or only rates good movies, that will skew their user effect. Regularization accounts for that by not using the mean, but rather dividing the sum of ratings by the number of ratings plus some tuneable parameter lambda. We can run simulations with many different lambdas to see what value of lambda minimizes the RMSE. Basically, we will rerun the prior model 5 analysis (which uses movie, user, age and genre effects), but testing various values of lambda to see where the RMSE is minimized. Once the analysis is run, we will plot the RMSE value obtained vs the lambda value, and determine which value of lambda minimizes RMSE.

```
# Using regularization to determine the lambda value that minimizes RMSE in our model
lambdas <- seq(2, 8, 0.25)

regularization <- function(x) {

  movie_bias <- edx_train %>%
    group_by(movieId) %>%
    summarize(m_bias = sum(rating-average_rating)/(n()+x))
```

```
  user_bias <- edx_train %>%
    left_join(movie_bias, by="movieId") %>%
    group_by(userId) %>%
    summarize(u_bias = sum(rating-average_rating-m_bias)/(n()+x))

  age_bias <- edx_train %>%
    left_join(movie_bias, by="movieId") %>%
    left_join(user_bias, by="userId") %>%
    group_by(age) %>%
    summarize(a_bias = sum(rating-average_rating-m_bias-u_bias)/(n()+x))

  genre_bias <- edx_train %>%
    left_join(movie_bias, by="movieId") %>%
    left_join(user_bias, by="userId") %>%
    left_join(age_bias, by="age") %>%
    group_by(genres) %>%
    summarize(g_bias = sum(rating-average_rating-m_bias-u_bias-a_bias)/(n()+x))

  y_hat <- edx_test %>%
    left_join(movie_bias, by="movieId") %>%
    left_join(user_bias, by="userId") %>%
    left_join(age_bias, by="age") %>%
    left_join(genre_bias, by="genres") %>%
    mutate(pred=(average_rating + m_bias + u_bias + a_bias + g_bias)) %>%
    pull(pred)

  RMSE(edx_test$rating, y_hat)

}

RMSEs_regular <- sapply(lambdas, regularization)

plot(lambdas, RMSEs_regular)
```



The plot of potential lambdas vs RMSE shows that there is a value of lambda that minimizes RMSE. What is that value?

```
# Using the best value of lambda, how does our regularized model do?
best_lambda <- lambdas[which.min(RMSEs_regular)]
best_lambda
```

```
## [1] 4.5
```

```
RMSE_6 <- RMSEs_regular[best_lambda]
RMSE_6
```

```
## [1] 0.8636666
```

Regularization didn't make a huge improvement to the model but it was still an improvement nonetheless. Let's add it to the table.

```
# Update results table
results_table <- tibble(Model = c("1. Average Rating", "2. Movie Effects Added",
                                  "3. Movie and User Effects Added",
                                  "4. Movie, User, and Age Effects Added",
                                  "5. Movie, User, Age, and Genre Effects Added",
                                  "6. Adding Regularization", "", "Goal"),
                        RMSE = c(RMSE_1, RMSE_2, RMSE_3, RMSE_4, RMSE_5, RMSE_6, "",
                                 goal)) %>% knitr::kable()
```

```
results_table
```

Model	RMSE
1. Average Rating	1.06005370222409
2. Movie Effects Added	0.942961498004501
3. Movie and User Effects Added	0.86468429490229
4. Movie, User, and Age Effects Added	0.864330070782133
5. Movie, User, Age, and Genre Effects Added	0.864080139682404
6. Adding Regularization	0.863666603239279
Goal	0.8649

Results

Our current model combines user effects, movie effects, age effects, genre effects, and a regularization step using a lambda value of 4.5. This model achieves an RMSE value of 0.86367 on the edx_test data set, which is quite a bit better than our goal of 0.8649. We will now declare this model to be our final mode, and test it against the final_holdout_test set to see if we can achieve an RMSE below our goal.

```
# Using our final model on the final_holdout_test data set
movie_bias <- edx_train %>%
  group_by(movieId) %>%
  summarize(m_bias = sum(rating-average_rating)/(n()+best_lambda))

user_bias <- edx_train %>%
  left_join(movie_bias, by="movieId") %>%
  group_by(userId) %>%
  summarize(u_bias = sum(rating-average_rating-m_bias)/(n()+best_lambda))

age_bias <- edx_train %>%
  left_join(movie_bias, by="movieId") %>%
  left_join(user_bias, by="userId") %>%
  group_by(age) %>%
  summarize(a_bias = sum(rating-average_rating-m_bias-u_bias)/(n()+best_lambda))

genre_bias <- edx_train %>%
  left_join(movie_bias, by="movieId") %>%
  left_join(user_bias, by="userId") %>%
  left_join(age_bias, by="age") %>%
  group_by(genres) %>%
  summarize(g_bias = sum(rating-average_rating-m_bias-u_bias-a_bias)/(n()+best_lambda))

y_hat_final <- final_holdout_test %>%
  left_join(user_bias, by = "userId") %>%
  left_join(movie_bias, by = "movieId") %>%
  left_join(age_bias, by="age") %>%
  left_join(genre_bias, by="genres") %>%
  mutate(pred=(average_rating + m_bias + u_bias + a_bias + g_bias)) %>%
  pull(pred)

# Final model RMSE on final_holdout_test data set
```

```
RMSE_final <- RMSE(final_holdout_test$rating, y_hat_final)
RMSE_final
```

```
## [1] 0.8646597
```

Conclusion

Our final model, when tested on the final_holdout_test set, achieved an RMSE of 0.86466, besting our goal of 0.8649. We have therefore successfully designed a machine learning model that uses data from millions of movie ratings to quite accurately predict whether a given user will like a given movie.

Even so, there is likely more that could be done to improve the model in the future. For instance, the data sets contain a "timestamp" column that records when the rating was entered by the user. It's possible that using this review date and time, in conjunction with the movie's release year, may further improve the model's accuracy. For instance, a user in 2024 reviewing an old classic that they haven't seen in years, like perhaps Ghostbusters, which was released 40 years ago in 1984, might have fond memories of it and rate it more favorably then if they are rating a newer movie that they literally just finished watching. On the flip side, this data may already be accurately captured by the age effects that we added into the model, as we saw that older movies are generally rated higher, regardless of review date. Thus, while there's no guarantee that this would improve the model significantly, it is certainly worth exploring.

Finally, we note that even our best model, which achieves an RMSE of 0.86466 and bests our goal, still provides an estimated rating that is off by more than four fifths of a star per prediction. No machine learning model can be perfect, and ours certainly isn't, but we have designed a model that is much better than simply throwing darts at the Netflix homepage, and for that we can be satisfied.