edX MovieLens Capstone Project

Jean Singer

5/6/2022

Contents

Introduction

The purpose of this project is to create a movie recommendation system using the MovieLens dataset. Recommendation systems are useful to companies like Netflix or Amazon because they enable the companies to suggest movies (or books or other products) that consumers are likely to enjoy based on their prior ratings or purchases.

The MovieLens dataset contains ratings from 69,878 users on 10,677 movies, and in total lists over 10 million ratings. In addition to user ID and movie ID, it contains the variables timestamp, title, and genres. Our goal is to create a model that will predict movie ratings based on a function of these variables, minimizing root mean square error (RMSE.)

The course demonstrated for us an approach to building a recommendation system based on the premise that each of the MovieLens variables will have effects or "biases" that to some degree impact a user's ratings of a given movie. For example, some movies are overall rated higher than others and would lend a positive bias. Or some users are generally more critical raters than others and would lend a negative bias. The model I created starts with the average rating across all movies and adds to that a series of biases based on selected variables. I also added a regularization step to the model, to minimize the outsize effect of ratings of movies with very few raters. In addition, I adjusted the model so that ratings below 0.5 were rounded up and ratings above 5 were rounded down.

The key steps taken were as follows:

- Generate the movielens dataset with code provided by the course. Add computed variables for ratings-per-movie and years-since-release.
- Set aside 10% of the Movielens dataset as a validation set. Take the remaining edx dataset and divide it into training and test sets.
- Create a series of models using the training set, adding one bias at a time and computing RMSE.
- Regularize the bias estimates, based on an optimized lambda.
- Round up all estimates less than 0.5 and round down all estimates greater than 5.
- Take the model with the lowest RMSE and run it on the validation set.

My final model contains the mean and biases for movie, user, number of ratings per movie, genres, and years since movie release, and is regularized and adjusted as described above. In the validation set, it generates an RMSE of 0.86389, which meets the target of less than 0.86490.

Methods and Analysis

Generate the Datasets

I ran the code provided by the course to generate the movielens dataset. (Code not shown.)

By examining the movielens structure, we see that it has 6 variables: user ID, movie ID, rating, timestamp, title, and genres.

```
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 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
##  - attr(*, ".internal.selfref")=<externalptr>
```

I added to the movielens dataset two variables that I would use later on: number of ratings received by each movie (num_ratings_movie) and years between movie release and rating (years_to_rating.) I added num_ratings_movie because movies that are very popular would tend to have large audiences, hence more ratings and potentially higher ratings. I added years between movie release and rating on the theory that people inclined to enjoy a movie will go out and see it earlier.

```
#Create the variable for number of ratings for each movie.
num_ratings_table <- movielens %>% group_by(movieId) %>%
  summarize(num_ratings_movie = n())
movielens <- movielens %>% left_join(num_ratings_table, by = "movieId")

#Create the variable for number of years between movie release and rating
#Extract release date from movie title and create a new column for release_year
movielens <- movielens %>% mutate(release_year = as.numeric(str_sub(title, -5, -2)))
#Convert the timestamp to a date
movielens <- movielens %>% mutate(date = as.Date(as_datetime(movielens$timestamp)))
#Add a column for the year of the rating
movielens<- movielens %>% mutate(rating_year = year(date))
#Compute number of years from release to rating and add as a variable
movielens <- movielens %>% mutate(years_to_rating = rating_year - release_year)
```

Next, I generated the edx and validation sets. The validation set is 10% of the movielens data. I set aside the validation set for use only at the end. I took the edx dataset and divided it into training and test sets.

```
#Create the edx and validation datasets.
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,]

# Remove from the validation set any observations with userIds,
#movieIds or years_to_rating that are not in the edx set.
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId") %>%
  semi_join(edx, by = "years_to_rating")

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

#Remove object names
rm(dl, ratings, movies, test_index, temp, movielens, removed)

#Now divide the edx dataset into a training and a temporary test set.
test_index <- createDataPartition(y = edx$rating, times = 1, p = .1, list = FALSE)
train <- edx[-test_index, ]
test_temp <- edx[test_index]

#Remove from the temporary test set any rows with userIds,
#movieIds or years_to_rating that are not in the training set
test <- test_temp %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")%>%
  semi_join(train, by = "years_to_rating")

#Add back to the training set any rows removed from the temporary
#test set.
removed <- anti_join(test_temp, test)
```

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

Data Exploration and Visualization

Dataset Dimensions

First, I examined the dimensions of the datasets. Below is a summary of the dimensions of the edx, training and test sets. (Code not shown.)

Dataset	Rows	Columns
edx	9000055	11
train	8100067	11
test	899988	11

We can see from the table above that the code has correctly taken approximately 10% of the edx dataset for the test set. We end up with a training set of approximately 8.1 million observations and a test set with just under 900,000 observations.

From the code below, we see that overall the edx dataset has 10,677 distinct movie titles, 69,878 distinct users, and 797 distinct genres (many of which are combinations of genres.)

```
#Identify number of distinct movies, users and genre categories
n_distinct(edx$movieId)
```

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

```
n_distinct(edx$userId)
```

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

```
n_distinct(edx$genres)
```

```
## [1] 797
```

We can see by running the dataset structure that the 11 variables in the dataset are now userID, movieID, rating, timestamp, title, genres, num_ratings_movie, release_year, date, rating_year, and years_to_rating.

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  11 variables:
##  $ userId           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId          : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating           : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp        : int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838
##  $ title            : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)"
##  $ genres           : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "A
##  $ num_ratings_movie: int  2412 14975 16075 18925 16167 5366 34457 4016 20972 8210 ...
##  $ release_year     : num  1992 1995 1995 1994 1994 ...
##  $ date             : Date, format: "1996-08-02" "1996-08-02" ...
##  $ rating_year      : num  1996 1996 1996 1996 1996 ...
##  $ years_to_rating  : num  4 1 1 2 2 2 2 2 2 2 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

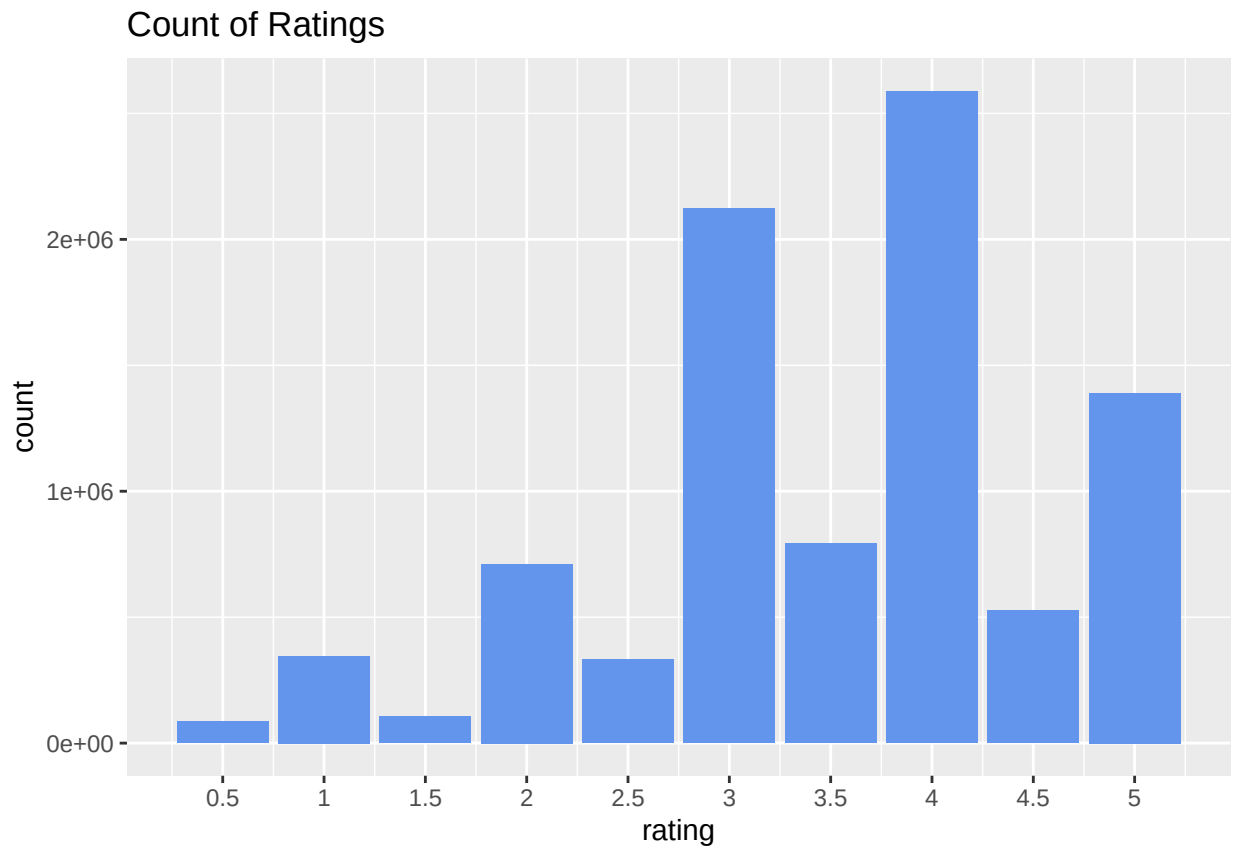
Variable Distributions and Relationships

I examined the following:

- Mean rating and distribution of ratings
- Relationship between number of movies and number of ratings
- Relationship between number of users and number of ratings
- Relationship between number of ratings a movie receives and average rating
- Relationship between the years that have elapsed between a movie's release and the movie's average rating

Code for mean rating and distribution of ratings:

```
#Examine the distribution of the ratings with a bar plot of count for each rating
edx %>% group_by(rating) %>%
  ggplot(aes(x = rating)) +
  geom_bar(fill = "cornflowerblue") +
  scale_x_continuous(breaks = seq(0.5,5,0.5),
                     labels = seq(0.5,5,0.5)) +
  ggtitle("Count of Ratings")
```



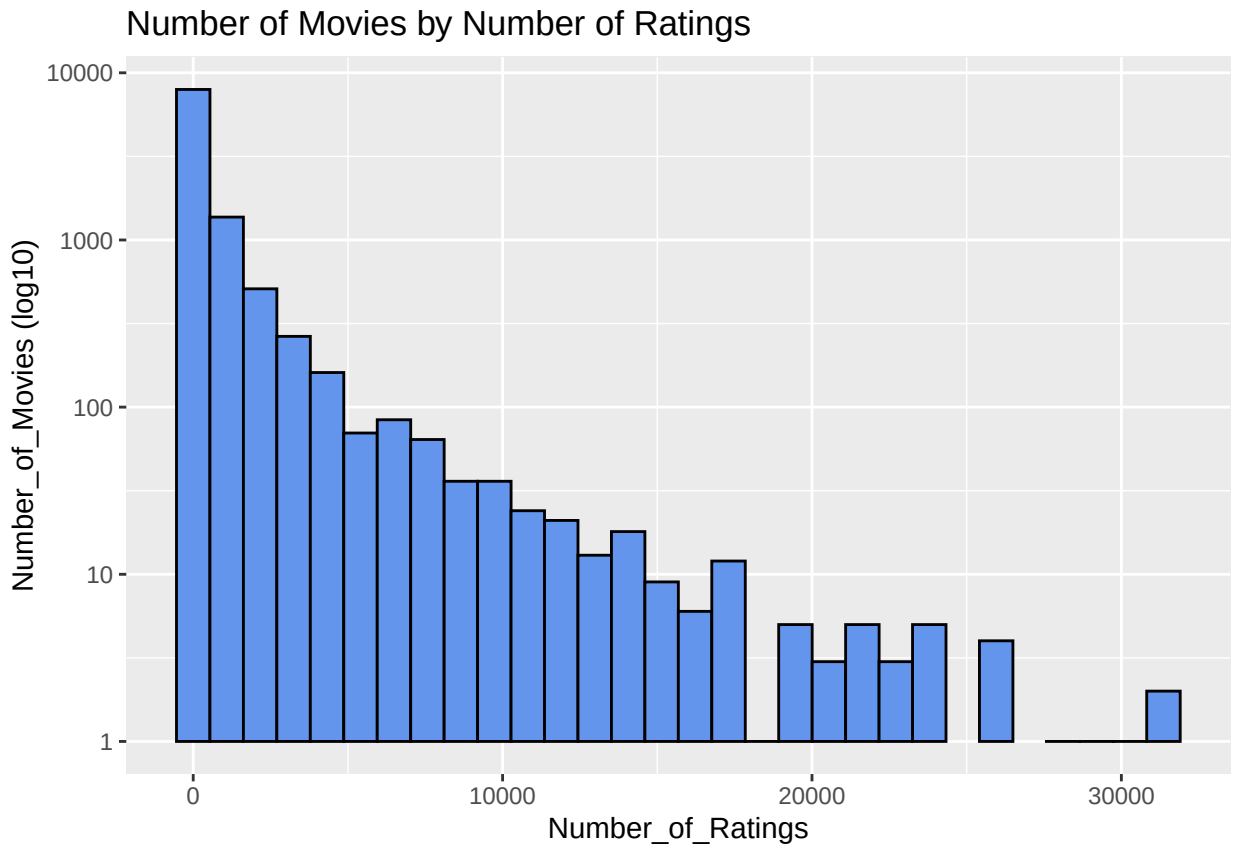
```
#Compute the mean rating
mean(edx$rating)
```

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

The bar plot above shows that the most common rating is a 4, and that "half" ratings are less common than full numbers. The mean rating is about 3.5.

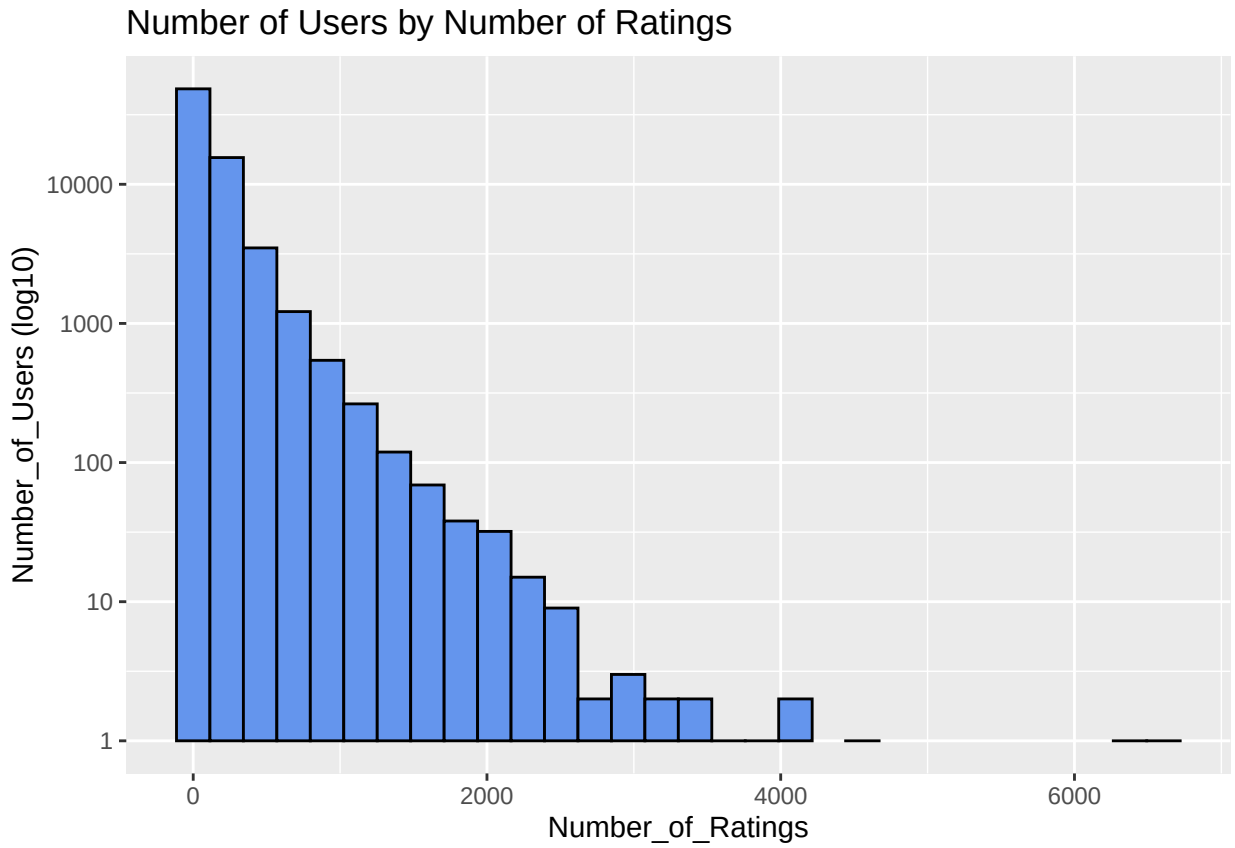
Code for relationships between number of movies and number of ratings, and number of users and number of ratings:

```
#Examine number of ratings by number of movies
edx %>% group_by(movieId)%>%
  summarize(Number_of_Ratings = n()) %>%
  ggplot(aes(Number_of_Ratings)) +
  geom_histogram(fill = "cornflowerblue", col = "black", bins = 30)+
  ylab("Number_of_Movies (log10)") +
  scale_y_continuous(trans = "log10") +
  ggtitle("Number of Movies by Number of Ratings")
```




```
#Examine number of ratings by number of users
  edx %>% group_by(userId)%>%
    summarize(Number_of_Ratings = n()) %>%
    ggplot(aes(Number_of_Ratings)) +
    geom_histogram(fill = "cornflowerblue", col = "black", bins = 30)+
```

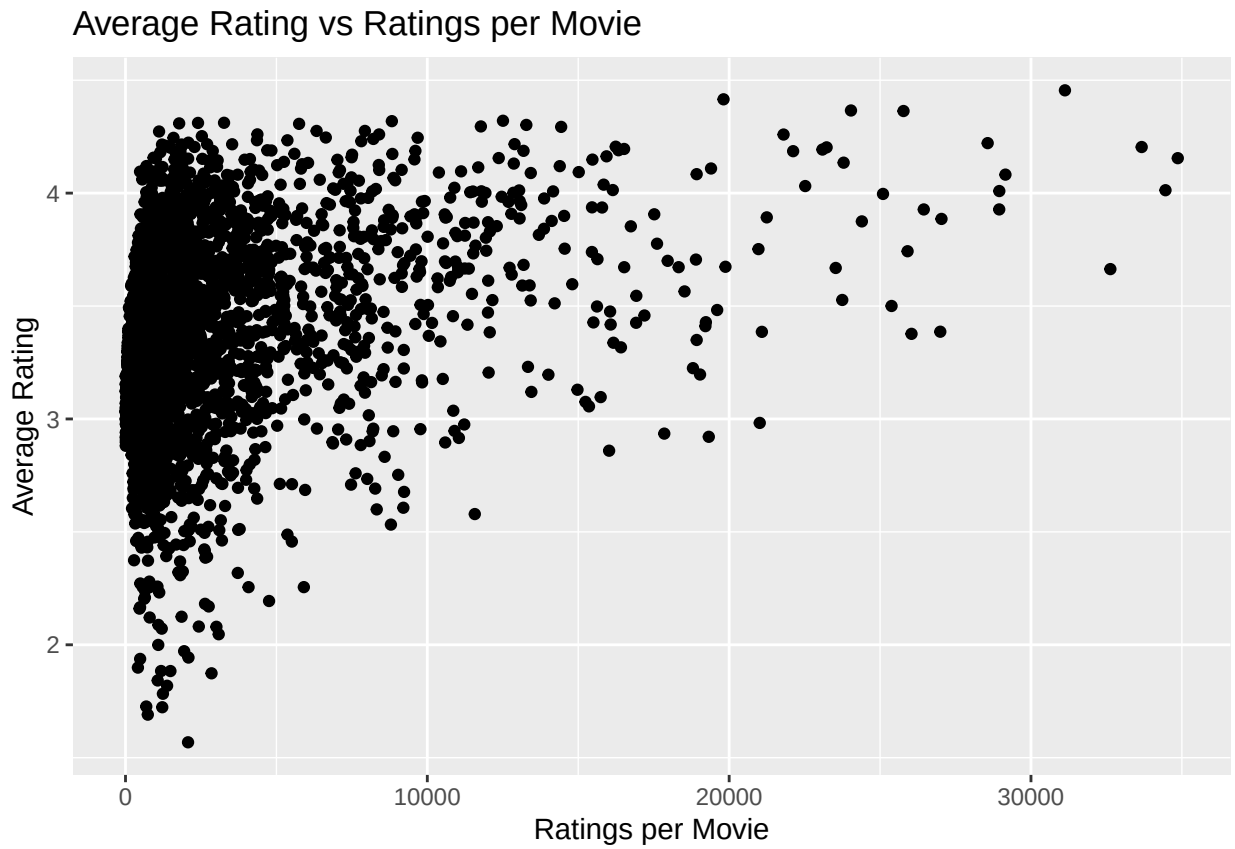
```
    ylab("Number_of_Users (log10)")+
    scale_y_continuous(trans = "log10")+
    ggtitle("Number of Users by Number of Ratings")
```



The resulting charts show that a relatively large number of movies get a small number of ratings and a relatively small number movies get a lot of ratings. Similarly, a relatively large number of users give a small number of ratings and a relatively small number of users give the most ratings.

Code for relationship between number of ratings a movie receives and its average rating:

```
#Examine whether movies with a lot of ratings tend to get
#higher ratings
num_mean_table <- edx %>% group_by(num_ratings_movie) %>%
    summarize(mean_rating = mean(rating))
num_mean_table %>% ggplot(aes(x=num_ratings_movie, y=mean_rating))+
    geom_point() +
  xlab("Ratings per Movie") +
  ylab("Average Rating") +
  ggtitle("Average Rating vs Ratings per Movie")
```



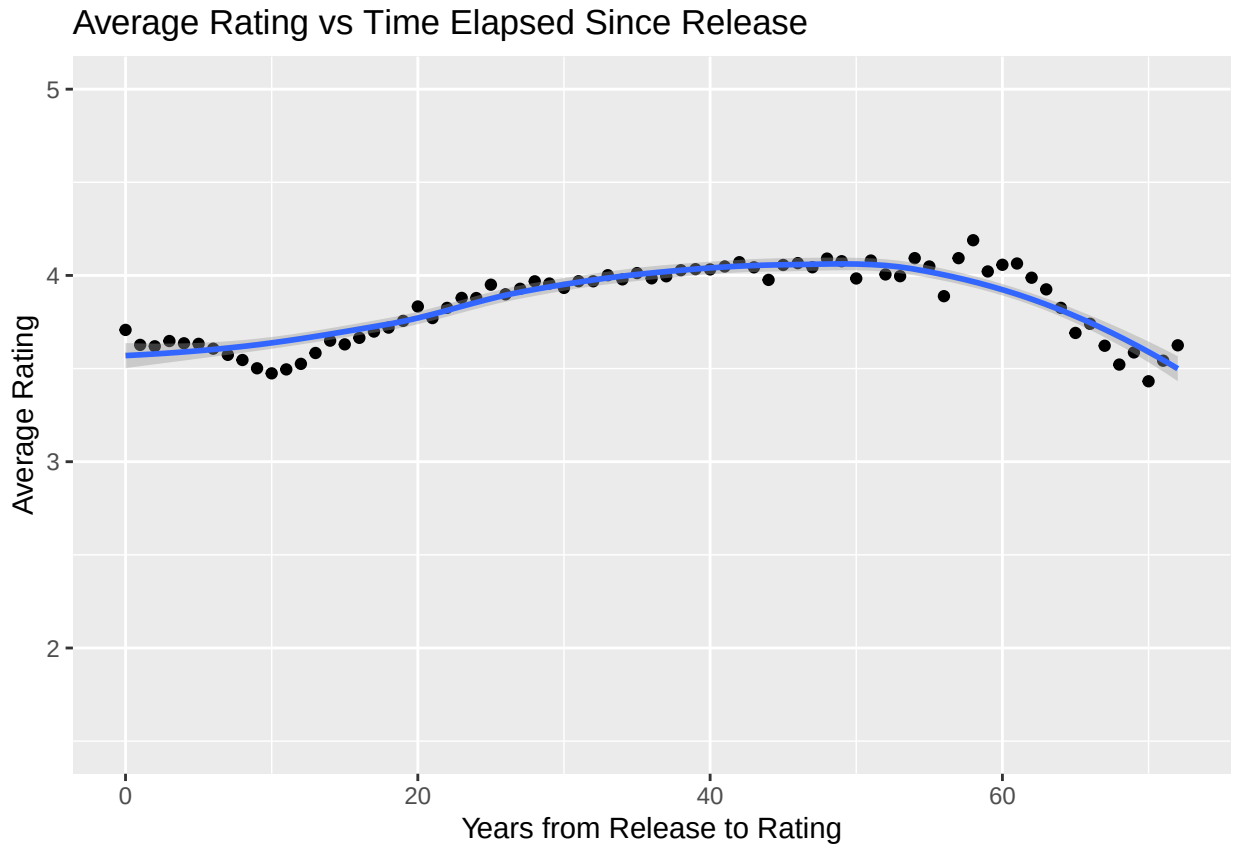

```
#run the test for pearson's correlation
cor.test(num_mean_table$num_ratings_movie, num_mean_table$mean_rating)
```

```
##
##  Pearson's product-moment correlation
##
## data:  num_mean_table$num_ratings_movie and num_mean_table$mean_rating
## t = 17.586, df = 2607, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.2909146 0.3595358
## sample estimates:
##      cor
## 0.325654
```

The argument for examining number of ratings a movie receives is that very popular movies would be seen and rated by a lot of people, and would be expected to receive higher ratings than those that are less popular and therefore seen and rated by fewer people. Using the code above, I found a weak but significant ($r = 0.33$, $p<.001$) correlation between the number of ratings a movie receives and the value of the rating. Based on the evidence that more widely seen and rated movies have at least a slightly higher likelihood of receiving a higher rating, I included this additional bias in my model.

Code for relationship between the years that have elapsed between a movie's release and the average rating:

```r
#create a table with years_to_rating and average ratings for movies
#with more than 4000 ratings
time_effect_table <- edx %>%
  filter(num_ratings_movie > 4000 & years_to_rating>=0) %>%
  group_by(years_to_rating) %>%
  summarise(average_rating = mean(rating))

#plot average rating vs years_to_rating for movies with more than 4000 ratings
time_effect_table %>%
  ggplot(aes(x=years_to_rating,y=average_rating)) + geom_point() +
  geom_smooth() +
  ylim(1.5, 5) +
  xlab("Years from Release to Rating") +
  ylab("Average Rating") +
  ggtitle("Average Rating vs Time Elapsed Since Release")
```




```r
#run the correlation between years_to_rating and average_rating
cor.test(time_effect_table$years_to_rating, time_effect_table$average_rating)
```

```
##
##  Pearson's product-moment correlation
##
## data:  time_effect_table$years_to_rating and time_effect_table$average_rating
## t = 3.6168, df = 71, p-value = 0.0005543
## alternative hypothesis: true correlation is not equal to 0
```

```
## 95 percent confidence interval:
##  0.1807750 0.5725476
## sample estimates:
##       cor
## 0.3944374
```

The plot shows over the shorter term (less than about 10 years), average ratings tend to decline with elapsed time. Potentially, people who are most likely to enjoy a movie see it earlier, and those who are less enthusiastic wait until it's been around for a while. But then the average ratings go up with time. Perhaps older movies that have stood the test of time get better ratings. However, after about 50 years, ratings then decline, perhaps as movies get very old and start to lose their relevance. The correlation between elapsed time and average rating is weak but significant (r = 0.39, p<.001) and therefore I included the time factor in my model.

Model Building

Average Rating

First I created the base model, using just the mean rating. I calculated the RMSE based on the formula below, where N is the number of observations, y_u,i is the rating for movie i by user u, and y_hat_u,i is the prediction:

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

```
#Build the base model, using just the overall mean rating as
#the predicted outcome
mu <- mean(train$rating)
mu
```

```
## [1] 3.512509
```

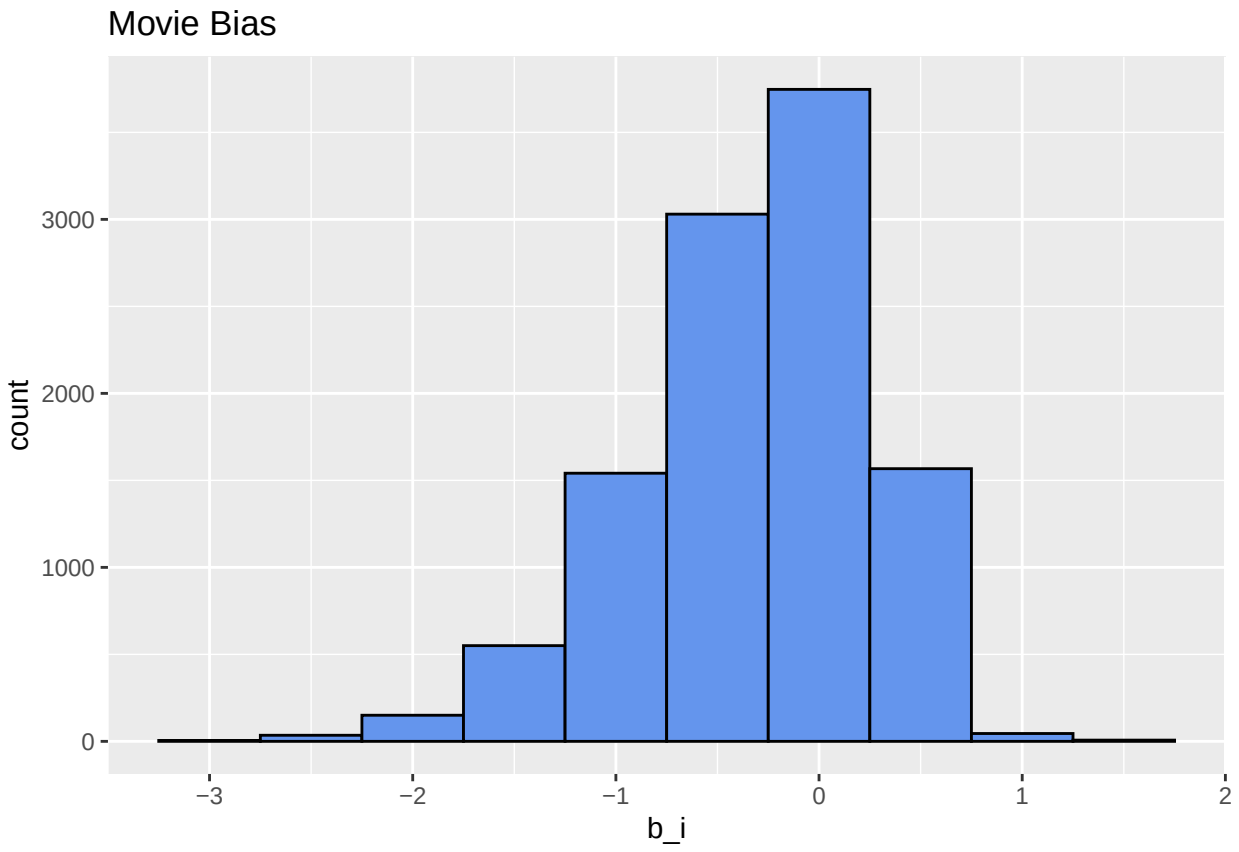
```
#Create a function to compute RMSE
RMSE <- function(actual_ratings, predicted_ratings){
  sqrt(mean((actual_ratings - predicted_ratings)^2))
}

#Compute the RMSE for the base model
RMSE_naive <- RMSE(test$rating, mu)
RMSE_naive <- format(round(RMSE_naive, 5), nsmall = 5)
RMSE_naive <- as.numeric(RMSE_naive)

#Create a table to track the models and their RMSEs.
rmse_results <- tibble(Model = c("Target", "Base Model"),
          Features = c("", "Average rating"), RMSE = c(0.86490, RMSE_naive))
rmse_results %>% knitr::kable()
```

Model	Features	RMSE
Target		0.86490
Base Model	Average rating	1.06114

Movie Bias

Next, I created Model 1 with the mean and movie bias features.

```
#Compute movie bias from the training set.
movie_bias <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

#Plot the movie biases to see how they vary
movie_bias %>% ggplot(aes(b_i)) +
  geom_histogram(bins = 10, fill = "cornflowerblue", col = "black")+
  ggtitle("Movie Bias")
```




```
#Make predictions based on Model 1
prediction1 <- mu + (test %>% left_join(movie_bias,
                    by = "movieId") %>% pull(b_i))

#compute RMSE for Model 1
```

```
RMSE1 <- RMSE(test$rating, prediction1)
RMSE1
```

```
## [1] 0.9441568
```

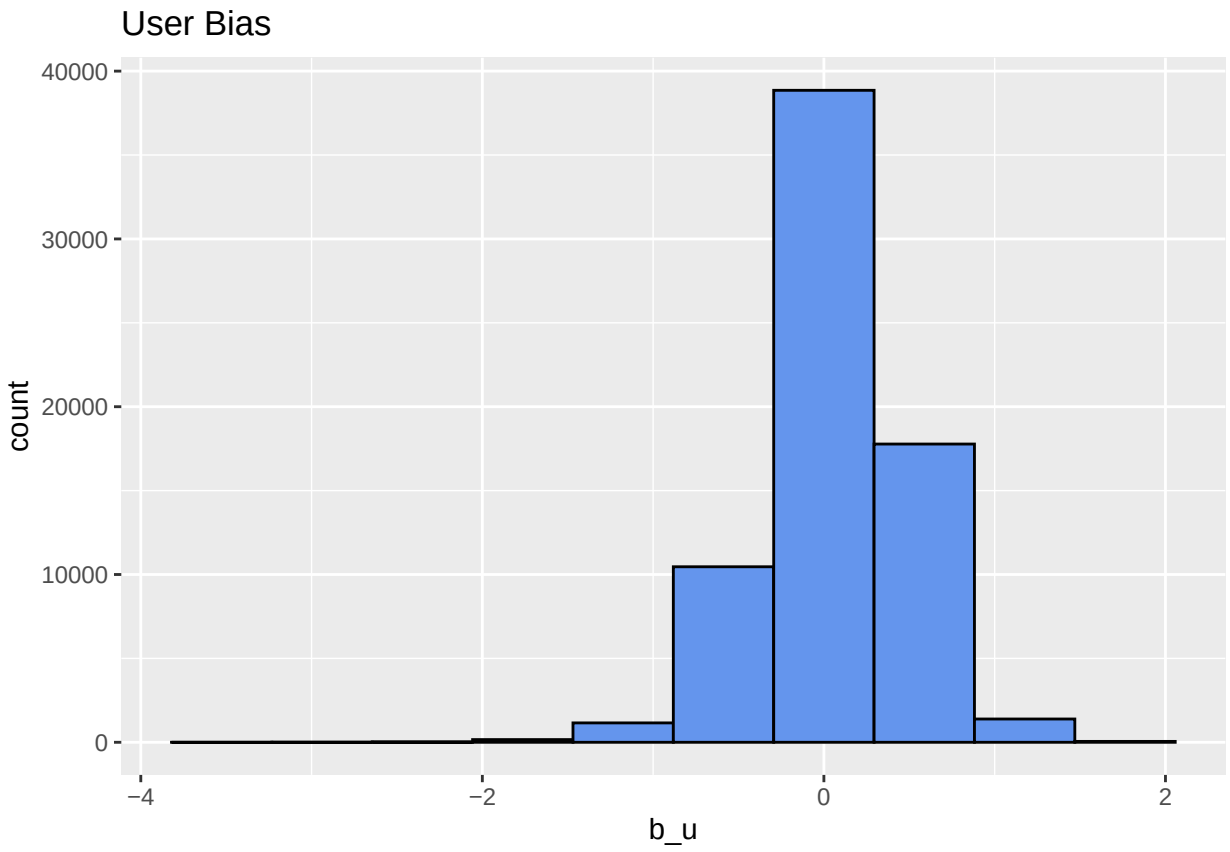
```
#Add Model 1 to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Model1",
                          Features = "Movie",
                          RMSE = RMSE1))
rmse_results %>% knitr::kable()
```

Model	Features	RMSE
Target		0.8649000
Base Model	Average rating	1.0611400
Model1	Movie	0.9441568

Movie Bias + User Bias

Then I created Model 2 with the mean and biases for movie and user.

```
#Create Model 2 by adding a feature for user bias
user_bias <- train %>%
  left_join(movie_bias, by = "movieId")%>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#Plot the user biases to see how they vary
user_bias %>% ggplot(aes(b_u)) +
  geom_histogram(bins = 10, fill = "cornflowerblue", col = "black")+
  ggtitle("User Bias")
```




```
#Make predictions based on Model 2
prediction2 <- test %>% left_join(movie_bias, by = "movieId") %>%
                        left_join(user_bias, by = "userId")%>%
                        mutate(pred = mu + b_i + b_u)%>%
                        pull(pred)

#compute RMSE for Model 2
RMSE2 <- RMSE(test$rating, prediction2)
RMSE2
```

```
## [1] 0.8659736
```

```
#Add Model 2 to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Model2",
                                 Features = "Movie, user",
                                 RMSE = RMSE2))
rmse_results %>% knitr::kable()
```

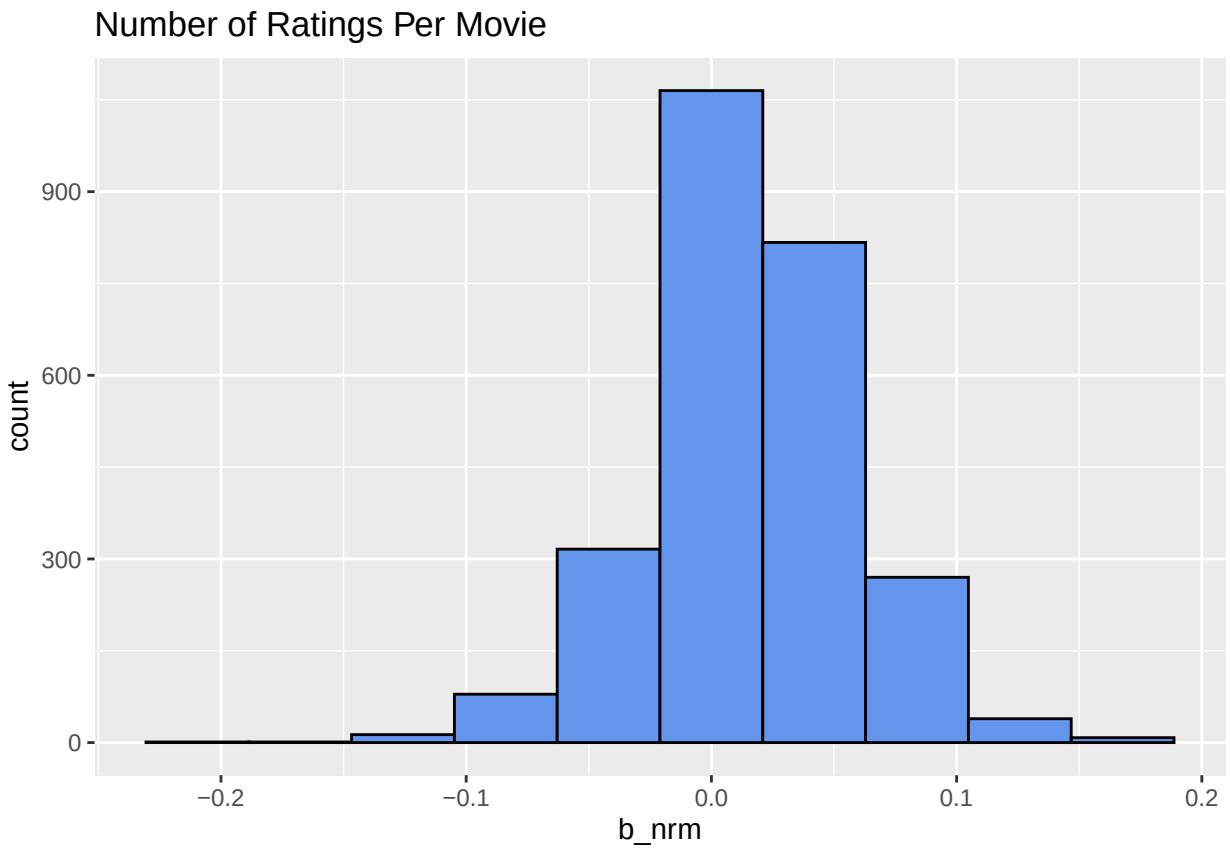
Model	Features	RMSE
Target		0.8649000
Base Model	Average rating	1.0611400
Model1	Movie	0.9441568

Model	Features	RMSE
Model2	Movie, user	0.8659736

Movie Bias + User Bias + Ratings per Movie

I created Model 3 by adding the bias for number of ratings a movie receives.

```
#Create Model 3 by adding a feature for the number of ratings
#each movie receives.
num_ratings_movie_bias <- train %>%
  left_join(movie_bias, by = "movieId")%>%
  left_join(user_bias, by = "userId")%>%
  group_by(num_ratings_movie) %>%
  summarize(b_nrm = mean(rating - mu - b_i - b_u))

#Plot the num_ratings_movie biases to see how they vary
num_ratings_movie_bias %>% ggplot(aes(b_nrm)) +
  geom_histogram(bins = 10, fill = "cornflowerblue", col = "black")+
  ggtitle("Number of Ratings Per Movie")
```




```
#Make predictions incorporating the num_ratings_movie biases
prediction3 <- test %>% left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId")%>%
  left_join(num_ratings_movie_bias, by = "num_ratings_movie")%>%
```

```
  mutate(pred = mu + b_i + b_u + b_nrm) %>%
  pull(pred)

#compute RMSE
RMSE3 <- RMSE(test$rating, prediction3)
RMSE3
```

```
## [1] 0.8650412
```

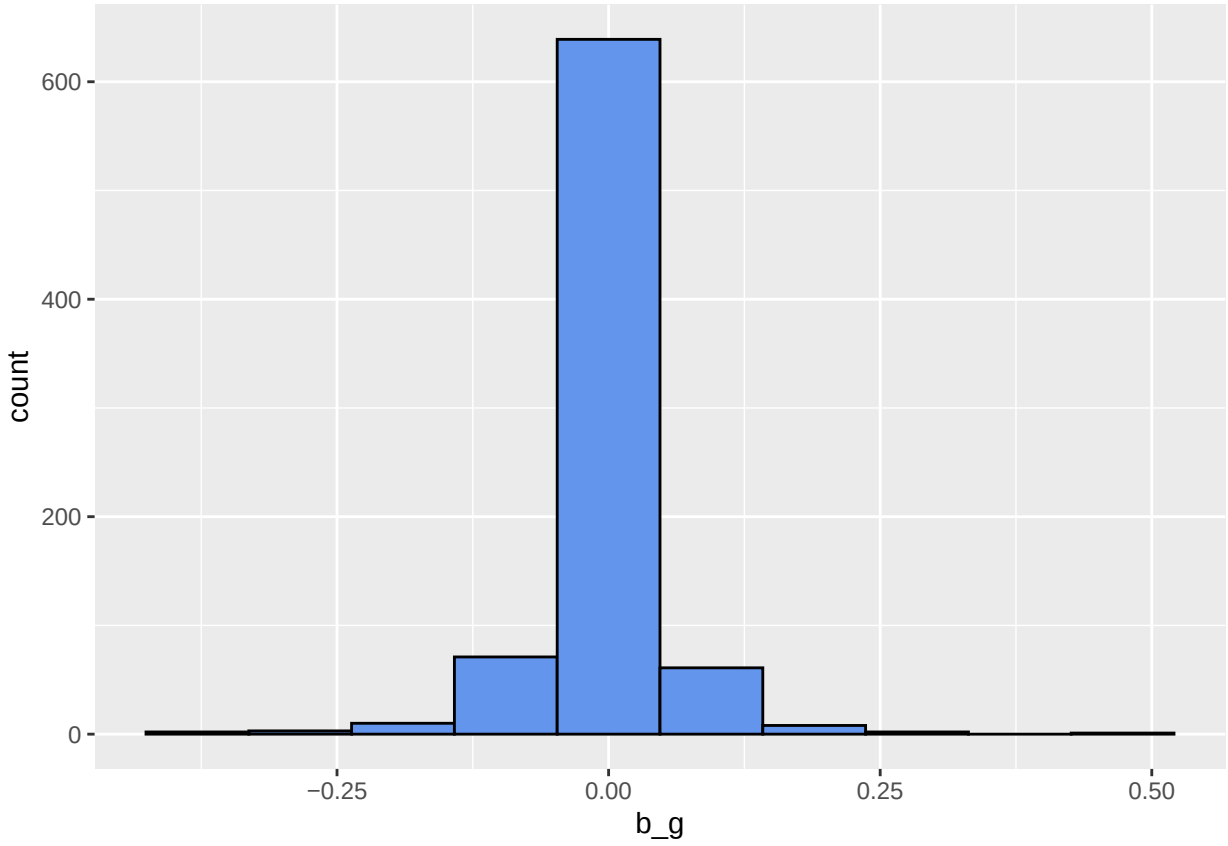
```
#Add Model 3 to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Model3",
                                 Features = "Movie, user, ratings per movie",
                                 RMSE = RMSE3))
rmse_results %>% knitr::kable()
```

Model	Features	RMSE
Target		0.8649000
Base Model	Average rating	1.0611400
Model1	Movie	0.9441568
Model2	Movie, user	0.8659736
Model3	Movie, user, ratings per movie	0.8650412

Movie Bias + User Bias + Ratings per Movie + Genres

I created Model 4 by adding the bias for genres. Genre is relevant because some genres are likely to be more enjoyed than others.

```
#Create Model 4 by adding a feature for the genre category
genre_bias <- train %>%
  left_join(movie_bias, by = "movieId")%>%
  left_join(user_bias, by = "userId")%>%
  left_join(num_ratings_movie_bias, by = "num_ratings_movie")%>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u - b_nrm))

#Plot the genre biases to see how they vary
genre_bias %>% ggplot(aes(b_g)) +
  geom_histogram(bins = 10, fill = "cornflowerblue", col = "black")
```




```
#Make predictions incorporating the genre biases
prediction4 <- test %>% left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId")%>%
  left_join(num_ratings_movie_bias, by = "num_ratings_movie")%>%
  left_join(genre_bias, by = "genres")%>%
  mutate(pred = mu + b_i + b_u + b_nrm + b_g)%>%
  pull(pred)

#compute RMSE
RMSE4 <- RMSE(test$rating, prediction4)
RMSE4
```

```
## [1] 0.864999
```

```
#Add the latest model to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Model4",
                                 Features = "Movie, user, ratings per movie, genres",
                                 RMSE = RMSE4))
rmse_results %>% knitr::kable()
```

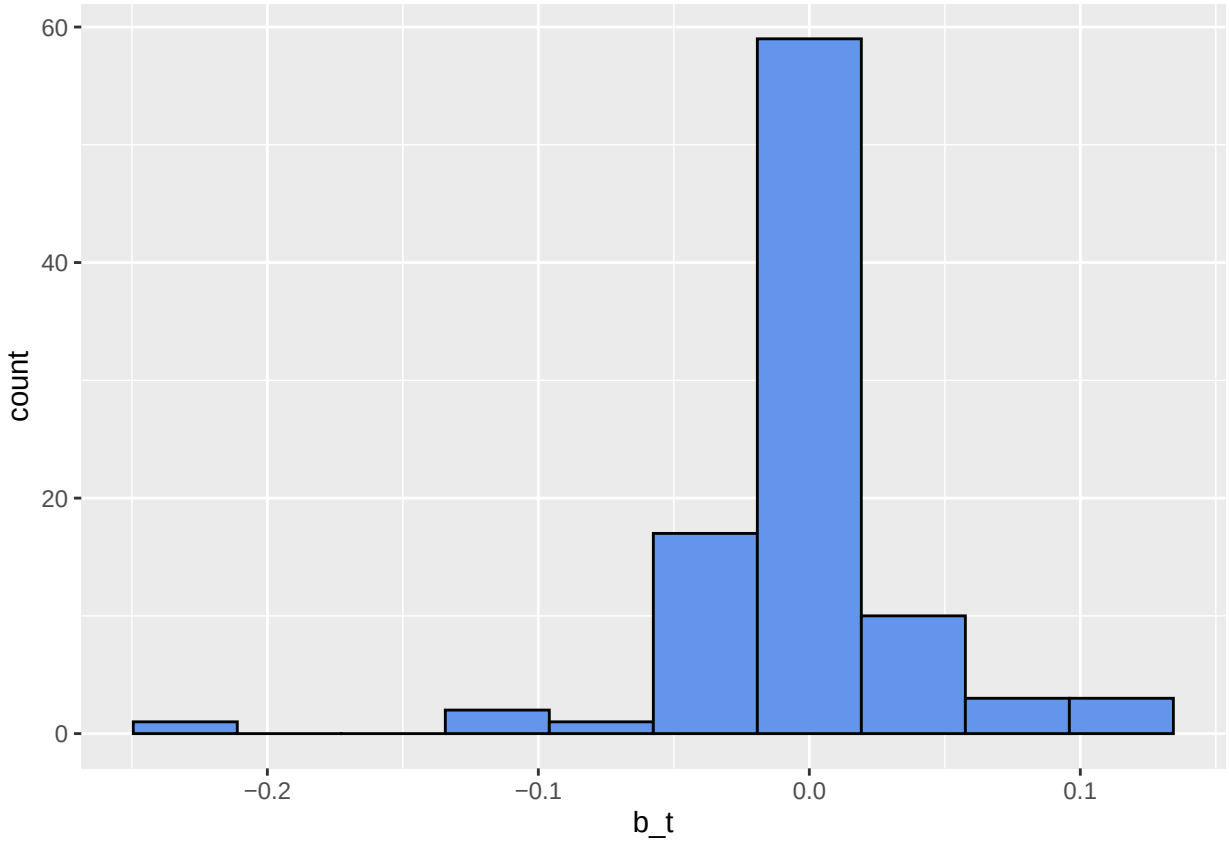
Model	Features	RMSE
Target		0.8649000
Base Model	Average rating	1.0611400

Model	Features	RMSE
Model1	Movie	0.9441568
Model2	Movie, user	0.8659736
Model3	Movie, user, ratings per movie	0.8650412
Model4	Movie, user, ratings per movie, genres	0.8649990

Movie Bias + User Bias + Ratings per Movie + Genres + Time

I created Model 5 by adding the bias for time elapsed between movie release and rating.

```
#Create Model 5 by adding a feature for time elapsed between movie release and rating
time_bias <- train %>%
  left_join(movie_bias, by = "movieId")%>%
  left_join(user_bias, by = "userId")%>%
  left_join(num_ratings_movie_bias, by = "num_ratings_movie")%>%
  left_join(genre_bias, by = "genres")%>%
  group_by(years_to_rating) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u - b_nrm - b_g))

#Plot the time biases to see how they vary
time_bias %>% ggplot(aes(b_t)) +
  geom_histogram(bins = 10, fill = "cornflowerblue", col = "black")
```



```
#Make predictions incorporating the time biases
prediction5 <- test %>% left_join(movie_bias, by = "movieId") %>%
  left_join(user_bias, by = "userId")%>%
  left_join(num_ratings_movie_bias, by = "num_ratings_movie")%>%
  left_join(genre_bias, by = "genres")%>%
  left_join(time_bias, by = "years_to_rating")%>%
  mutate(pred = mu + b_i + b_u + b_nrm + b_g + b_t)%>%
  pull(pred)

#compute RMSE
RMSE5 <- RMSE(test$rating, prediction5)
RMSE5
```

```
## [1] 0.8646313
```

```
#Add the latest model to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Model5",
                                 Features = "Movie, user, ratings per movie, genres, time",
                                 RMSE = RMSE5))
rmse_results %>% knitr::kable()
```

Model	Features	RMSE
Target		0.8649000
Base Model	Average rating	1.0611400
Model1	Movie	0.9441568
Model2	Movie, user	0.8659736
Model3	Movie, user, ratings per movie	0.8650412
Model4	Movie, user, ratings per movie, genres	0.8649990
Model5	Movie, user, ratings per movie, genres, time	0.8646313

Regularization

When some movies are rated by only a few users, we have greater uncertainty and the bias will tend to be larger and less trustworthy. We can adjust for this using regularization. Regularization enables us to penalize large estimates that result from small sample sizes, thereby constraining the total variability of the effect sizes. Instead of minimizing RMSE, we minimize the following equation, in which the second term is a penalty:

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

The values of b_i that minimize this equation are:

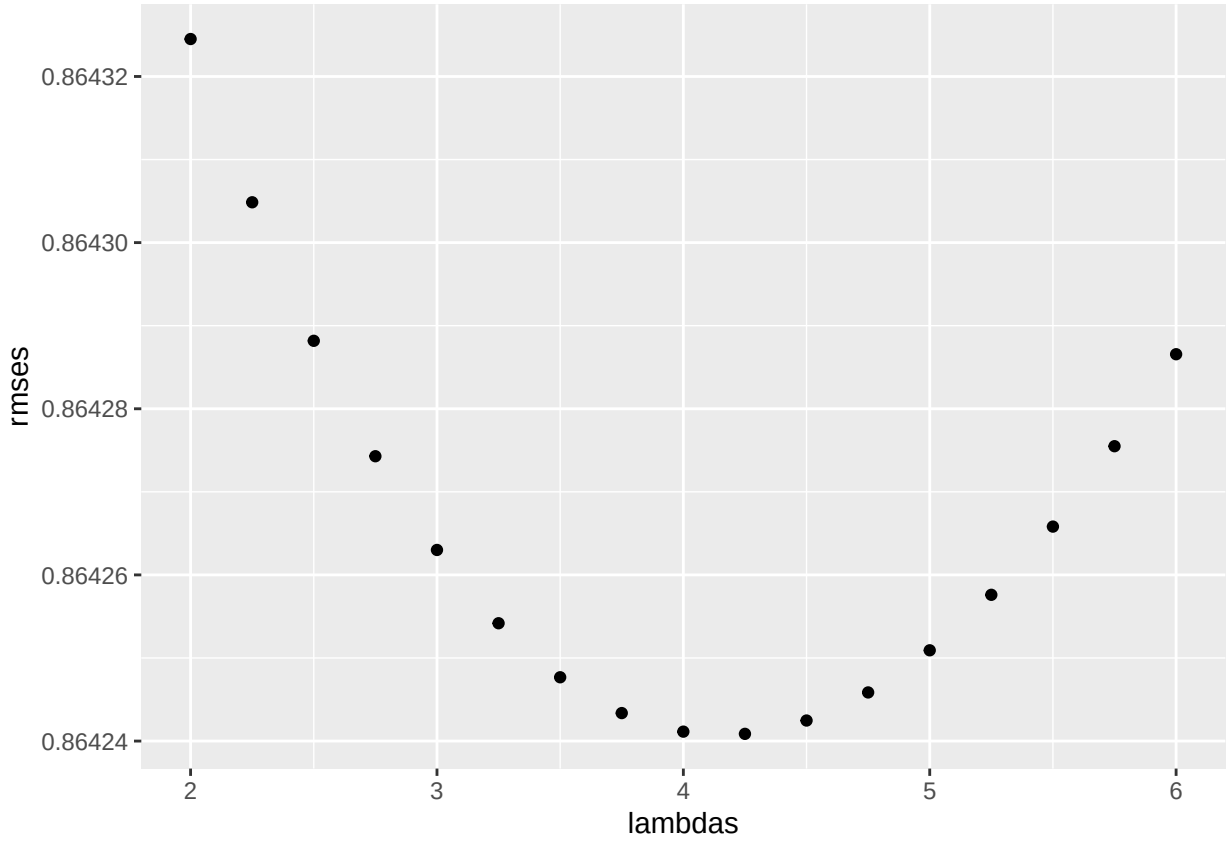
$$\hat{b}_i\left(\lambda\right)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}\left(Y_{u,i}-\hat{\mu}\right)$$

In the above equation, when sample size, n_i, is large, the penalty, lambda, is essentially ignored because n_i + lambda is approximately equal to n_i. But when n_i is small, lambda shrinks the size of the estimate. Lambda is a tuning parameter so we use cross-validation to choose its optimal value.

In Model 6, I applied regularization to minimize the outsize effect of ratings of movies with very few raters. (Warning: this can take a while to run.)

```
#Create Model 6 by adding regularization.
#First find the optimal lambda and then use it to run the model
#and compute RMSE.
lambdas <- seq(2, 6, .25)

rmses <- sapply(lambdas, function(lambda){

  mu <- mean(train$rating)

  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  b_nrm <- train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(num_ratings_movie) %>%
    summarize(b_nrm = sum(rating - b_u - b_i - mu)/(n()+lambda))

  b_g <- train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_nrm, by="num_ratings_movie") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_u - b_i - b_nrm - mu)/(n()+lambda))

  b_t <- train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_nrm, by="num_ratings_movie") %>%
    left_join(b_g, by="genres") %>%
    group_by(years_to_rating) %>%
    summarize(b_t = sum(rating - b_u - b_i - b_nrm - b_g - mu)/(n()+lambda))

  predicted_ratings <-
    test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_nrm, by="num_ratings_movie") %>%
    left_join(b_g, by="genres") %>%
    left_join(b_t, by="years_to_rating") %>%
    mutate(pred = mu + b_i + b_u + b_nrm + b_g + b_t) %>%
    pull(pred)
```

```
  return(RMSE(predicted_ratings, test$rating))
})

#Make a plot to see the optimal lambda
qplot(lambdas, rmses)
```




```
#Identify the optimal lambda
optimal_lambda <- lambdas[which.min(rmses)]
optimal_lambda
```

```
## [1] 4.25
```

```
#Identify the RMSE associated with the optimal lambda
optimal_rmse<- rmses[which.min(rmses)]
optimal_rmse
```

```
## [1] 0.8642409
```

```
#Add the latest model to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Model6",
                                 Features = "All of the above, regularized",
                                 RMSE = optimal_rmse))
rmse_results %>% knitr::kable()
```

Model	Features	RMSE
Target		0.8649000
Base Model	Average rating	1.0611400
Model1	Movie	0.9441568
Model2	Movie, user	0.8659736
Model3	Movie, user, ratings per movie	0.8650412
Model4	Movie, user, ratings per movie, genres	0.8649990
Model5	Movie, user, ratings per movie, genres, time	0.8646313
Model6	All of the above, regularized	0.8642409

The optimal lambda is 4.25 and the regularized model produces an RMSE of 0.86424.

Rounding Adjustment

The only possible values for ratings are whole numbers or numbers with 0.5. Therefore any predicted ratings less than 0.5 can be made more accurate by rounding up to 0.5 and any predicted ratings greater than 5 can be made more accurate by rounding down to 5. I found that approximately 1500 predicted values are below 0.5 or above 5 and adjusted them accordingly.

```
#Examine Model 6 to see how many predictions are below 0.5 or above 5.
lambda <- 4.25
mu <- mean(train$rating)

b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

b_nrm <- train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(num_ratings_movie) %>%
  summarize(b_nrm = sum(rating - b_u - b_i - mu)/(n()+lambda))

b_g <- train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_nrm, by="num_ratings_movie") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_u - b_i - b_nrm - mu)/(n()+lambda))

b_t <- train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_nrm, by="num_ratings_movie") %>%
  left_join(b_g, by="genres") %>%
  group_by(years_to_rating) %>%
  summarize(b_t = sum(rating - b_u - b_i - b_nrm - b_g - mu)/(n()+lambda))
```

```
predicted_ratings6 <-
  test %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_nrm, by="num_ratings_movie") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_t, by="years_to_rating") %>%
  mutate(pred = mu + b_i + b_u + b_nrm + b_g + b_t) %>%
  pull(pred)

#number of ratings <0.5
sum(predicted_ratings6 <0.5)
```

```
## [1] 66
```

```
#number of ratings >5
sum(predicted_ratings6 >5)
```

```
## [1] 1421
```

```
#Round up observations less than 0.5 and round down observations
#greater than 5
predicted_ratings6 <- ifelse(predicted_ratings6 <0.5, 0.5,
                             ifelse(predicted_ratings6 >5, 5,
                                    predicted_ratings6))

#Now calculate RMSE for the adjusted values.
FinalRMSE <- RMSE(test$rating, predicted_ratings6)
FinalRMSE
```

```
## [1] 0.8641262
```

The adjusted model brings the RMSE down to 0.86413, slightly lower than the unadjusted model.

Results

Model Building Results

Each consecutive step in the model-building process reduced RMSE. The final RMSE was 0.86413, which meets the goal of generating an RMSE below 0.86490.

```
#Add the final model to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(Model = "Final Model",
                                 Features = "Regularized movie, user, ratings per movie, genres, time +
                                 RMSE = FinalRMSE))
rmse_results %>% knitr::kable()
```

Model	Features	RMSE
Target		0.8649000
Base Model	Average rating	1.0611400
Model1	Movie	0.9441568
Model2	Movie, user	0.8659736
Model3	Movie, user, ratings per movie	0.8650412
Model4	Movie, user, ratings per movie, genres	0.8649990
Model5	Movie, user, ratings per movie, genres, time	0.8646313
Model6	All of the above, regularized	0.8642409
Final Model	Regularized movie, user, ratings per movie, genres, time + adjustment	0.8641262

Validation Results

I ran the final model on the validation set as follows:

```
#Run the model on the validation set
lambda <- 4.25
mu <- mean(train$rating)

b_i <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

b_u <- train %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

b_nrm <- train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(num_ratings_movie) %>%
  summarize(b_nrm = sum(rating - b_u - b_i - mu)/(n()+lambda))

b_g <- train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_nrm, by="num_ratings_movie") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - b_u - b_i - b_nrm - mu)/(n()+lambda))

b_t <- train %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_nrm, by="num_ratings_movie") %>%
  left_join(b_g, by="genres") %>%
  group_by(years_to_rating) %>%
  summarize(b_t = sum(rating - b_u - b_i - b_nrm - b_g - mu)/(n()+lambda))

predicted_ratings_val <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
```

```
  left_join(b_u, by = "userId") %>%
  left_join(b_nrm, by="num_ratings_movie") %>%
  left_join(b_g, by="genres") %>%
  left_join(b_t, by="years_to_rating") %>%
  mutate(pred = mu + b_i + b_u + b_nrm + b_g + b_t) %>%
  pull(pred)

#Compute adjusted ratings
predicted_ratings_val <- ifelse(predicted_ratings_val <0.5, 0.5,
                            ifelse(predicted_ratings_val >5, 5,
                                   predicted_ratings_val))

#Compute RMSE for the validation set
RMSE_val <- RMSE(validation$rating, predicted_ratings_val)
RMSE_val
```

```
## [1] 0.8638858
```

RMSE for the final model on the validation set is 0.86389, which meets the goal of generating an RMSE below 0.86490.

Conclusion

Summary In this report, I describe the development of a model to predict movie ratings based on the movielens dataset. The goal was to minimize RMSE to below 0.86490. My final model consists of the mean movie rating and biases for movie, user, number of ratings per movie, genre, and time elapsed from movie release to rating. The biases are regularized and I added a final adjustment for ratings below 0.5 or over 5. The final RMSE is 0.86389, which meets the goal.

Limitations and Future Work While an RMSE of 0.86389 meets the goal, this amount of error in general could create substantial differences between estimated and actual ratings. For example, an actual rating of 3 could be estimated as 3.86 or 2.14.; in other words, an actual 3-star rating could be estimated as nearly 4 or 2. It would therefore be advantageous to improve the model further.

Additional features could be added, such as the number of ratings per user (my model included number of ratings *per movie*) on the premise that people who rate more see a lot of movies and may be more critical.

When I created the model, each additional bias reduced the RMSE further but with diminishing returns. And, the order mattered. For example, I generated a slightly lower RMSE when I added number of ratings per movie before genres, as opposed to when I added genres before number of ratings per movie. It makes sense that order would matter, given how the biases are computed, with each bias relying on previously computed biases. To optimize the model as more features are added, order should be taken into consideration.

Different methods could also be used such as those considering implicit data (e.g., the fact that someone watches a lot of comedies and not a lot of science fiction would imply that they like comedies better and would rate it higher than science fiction), interactions between users and variables, or matrix factorization methods such as SVD.