MovieLens Capstone Project

Emi Bode

2024/05/06

Introduction

To develop a recommendation system for movies based on user ratings, we'll create an algorithm that predicts the preferences or ratings users would assign to different movies. This system will analyze ratings provided by users to generate personalized movie suggestions. Just like Netflix, we aim to forecast how users might rate specific movies. Inspired by successful strategies from previous Netflix challenges, we're embarking on a similar endeavor. In October 2006, Netflix initiated a challenge to enhance their recommendation algorithm, offering a million-dollar prize for a 10% improvement. By September 2009, the winners had been announced. For insights into how the winning algorithm was crafted, you can explore a summary and detailed explanation provided here. Now, we'll delve into the data analysis strategies utilized by the winning team to create our own movie recommendation system.

The aim of this assignment is to develop a recommendation system focused on suggesting movies, leveraging a rating scale as a key component.

Dataset

For this project, we'll be utilizing the MovieLens dataset compiled by GroupLens Research, which is available on the MovieLens website (http://movielens.org).

Data Loading

The dataset is loaded using the code provided by the course instructor, accessible through the following link: https://bit.ly/2Ng6tVW. This code splits the data into two parts: an initial dataset (edx set) and a 10% validation set. The edx set is further divided into training and test sets, while the validation set remains separate and is reserved for final evaluation purposes.

```
############################################################
# Create edx set, validation set, and submission file
############################################################

# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

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

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

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

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Final hold-out test set will be 10% of MovieLens data

set.seed(1)
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 set

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

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

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

#############################################################################
################################################
```

Verifying the data for any NA values

```
anyNA(edx)
```

```
## [1] TRUE
```

Data Overview and Initial Analysis

After loading the dataset, our initial step involves examining its structure and data types. We observe six variables: userId, movieID, rating, timestamp, title, and genres. It's noteworthy that for predictive purposes, we may need to separate the year from the title, and similarly, the genres might require separation for further analysis or prediction tasks.

```
str(edx)
```

```
## 'data.frame':    9000061 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  NA NA NA NA ...
##  $ genres   : chr  NA NA NA NA ...
```

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000061     Length:9000061
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

From the data summary, we observe that the ratings range from a minimum of 1 to a maximum of 5. The mean rating is calculated at 3.512, with a mode of 4.0, indicating a tendency towards higher ratings.

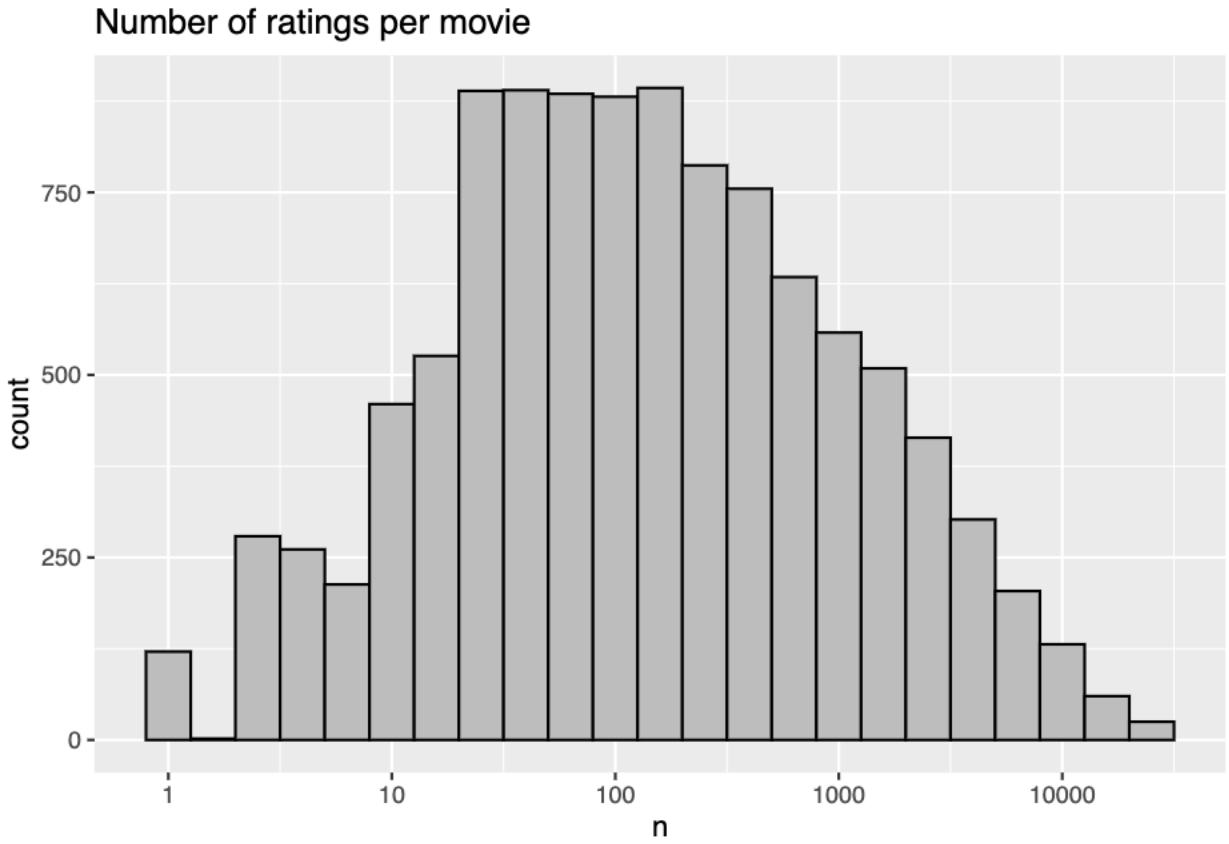
```
## Selecting by count
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588021
## 2    3   2121638
## 3    5   1390541
## 4    3.5  792037
## 5    2    710998
```

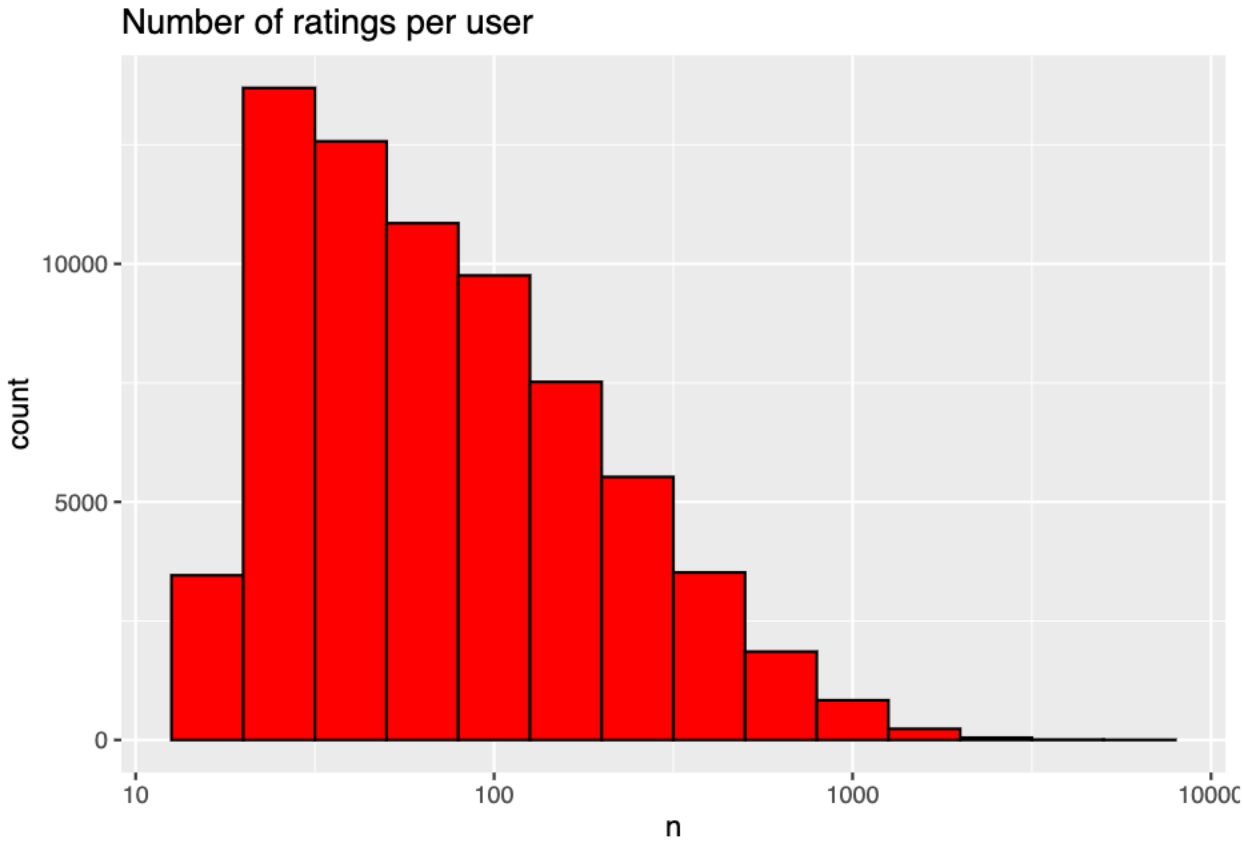
Obtaining the count of movies and users in the dataset:

```
##   n_users n_movies
## 1   69878    10677
```

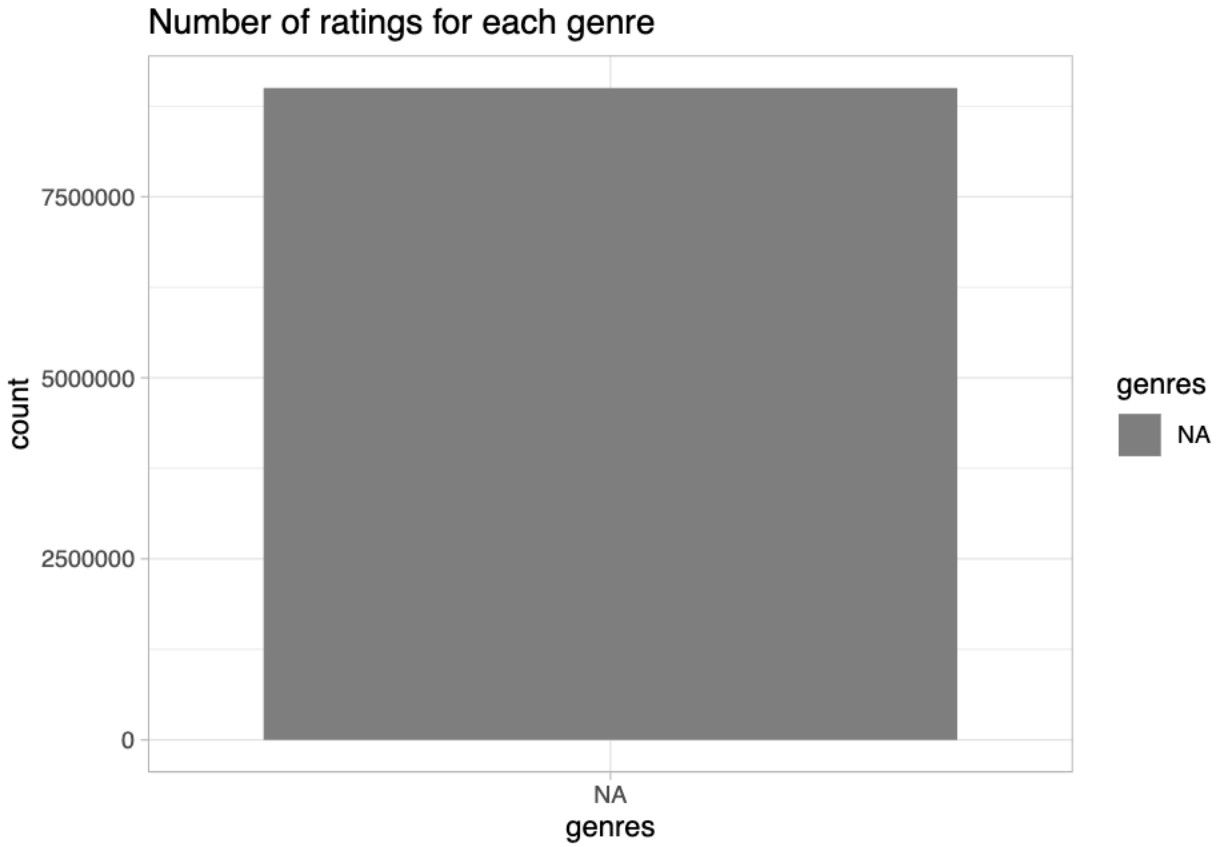
Determining the number of ratings received by each movie:



We observe variations in the number of ratings received by different movies, possibly indicating differences in popularity. To further explore this, we visualize the distribution of ratings for each user.



We notice discrepancies in user activity levels, with certain users rating movies more frequently than others. Calculating the number of ratings for each movie genre:



Let's explore the top 10 most popular genres.

```
## # A tibble: 1 x 2
##   genres   count
##   <chr>    <int>
## 1 <NA>   9000061
```

Partitioning the data:

Before constructing the model, we partition the edx dataset, allocating 20% for the test set and reserving 80% for the training set.

```
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

RMSE calculation Function

In the Netflix challenge, a standard error loss metric was employed. The winner was determined based on the residual mean squared error (RMSE) calculated on a designated test set. RMSE serves as the measure of accuracy for the evaluation.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = TRUE))
}
```

The first model:

In our initial model, we adopt a simplistic approach where the same rating is predicted for all movies, irrespective of the user. This model assumes a uniform rating across all movies and users, without considering

any biases. The method operates on the assumption of the following linear equation:

$Y\ u, i\ =?? + ??\ u, i$

```
Mu_1 <- mean(train_set$rating)
Mu_1
```

```
## [1] 3.51238
```

```
naive_rmse <- RMSE(test_set$rating,Mu_1)
naive_rmse
```

```
## [1] 1.059648
```

This code generates a table to store the RMSE results obtained from each method, facilitating comparison between different approaches.

```
rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## i Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

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

method	RMSE
Just the average	1.059648

Creating the second model:

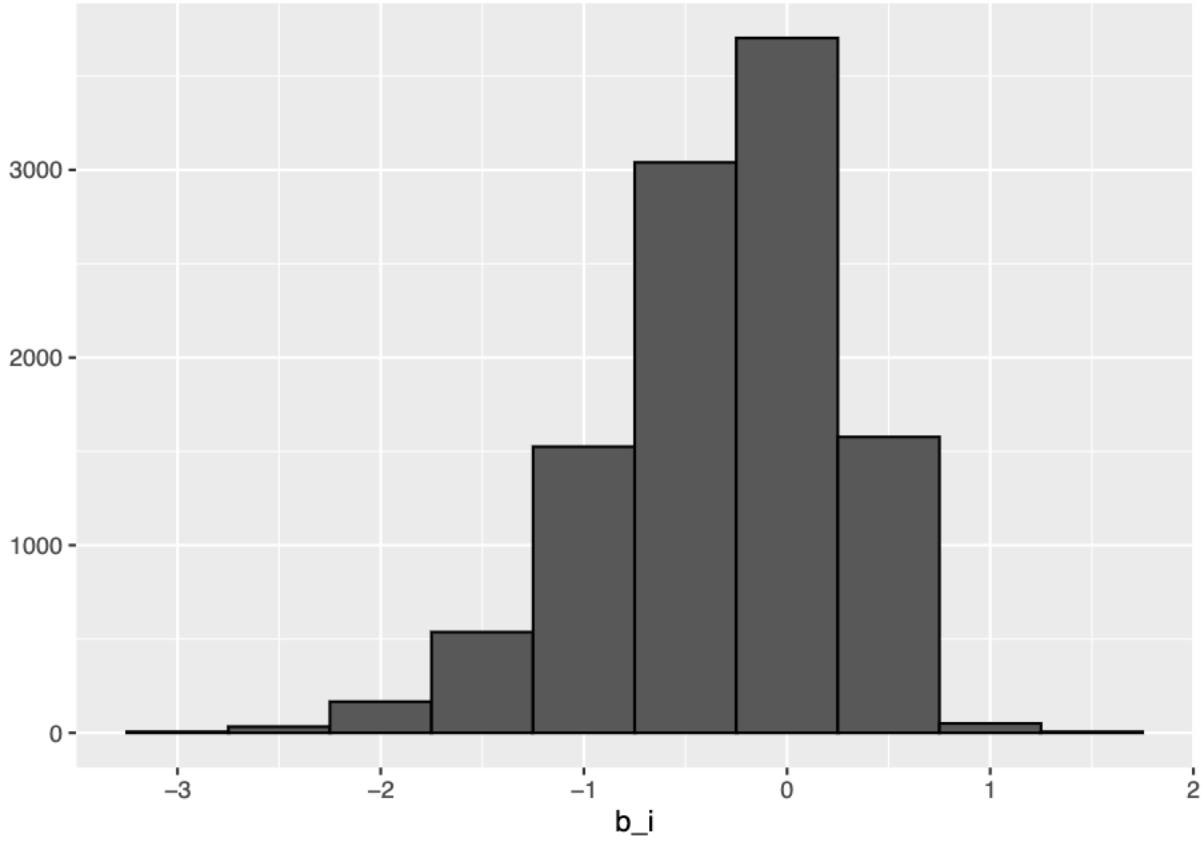
As observed during the exploratory analysis certain movies received more ratings compared to others. We can enhance our previous model by incorporating the term b_i to denote the average rating for movie i. We can once again utilize least squares to estimate the movie effect

$Y\ u, i\ =?? + b\ i\ + ??\ u, i$ Due to the large number of parameters $b\ i$ corresponding to each movie, employing the lm() function directly can lead to significant computational slowdown. Therefore, we opt for a more efficient approach by computing it using the average, as follows:

```
Mu_2 <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - Mu_2))
```

We observe variability in the estimate, as depicted in the plot presented here:

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Let's examine how the prediction accuracy improves after modifying the equation $Y\ u, i\ = ?? + b\ i$

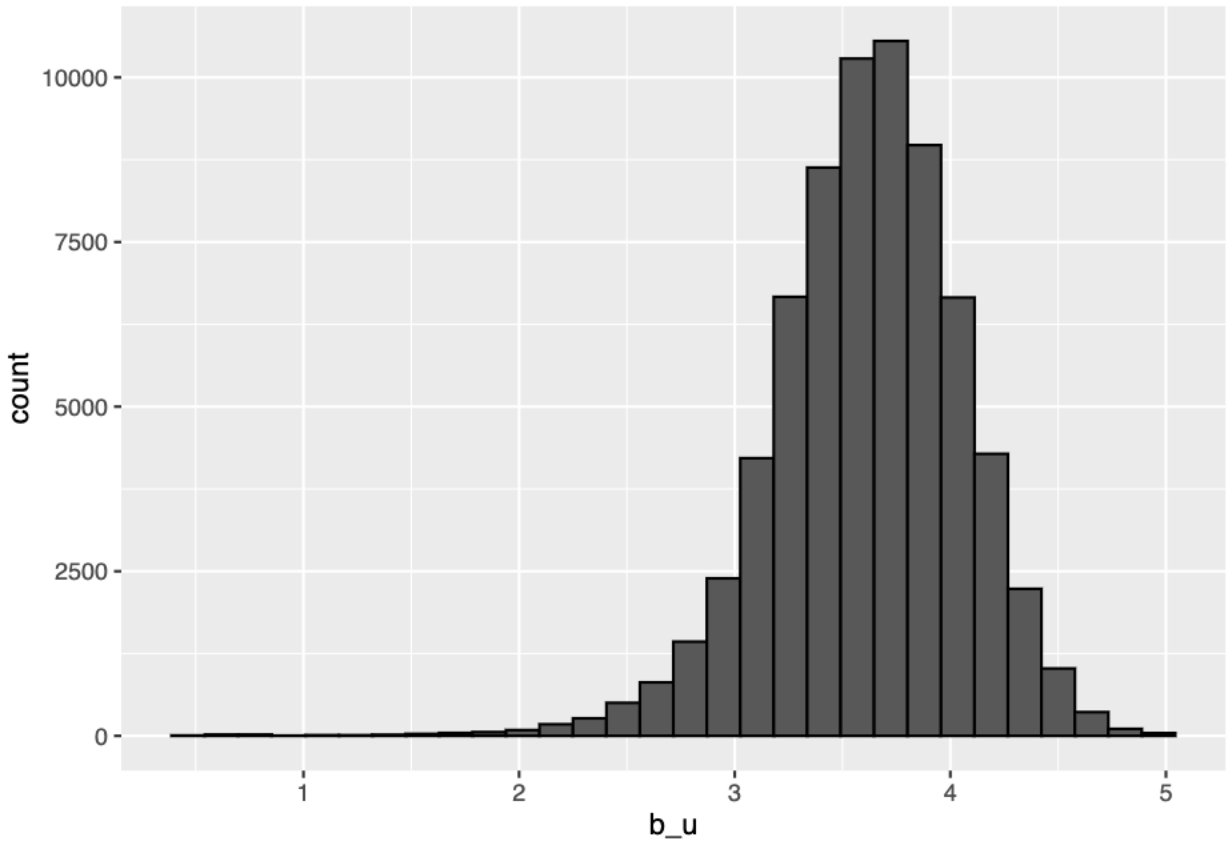
```
predicted_ratings <- Mu_2 + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0596481
Movie Effect Model	0.9431724

Creating the third model:

Comparing users who have rated more than 100 movies:



There is significant variability observed across user ratings as well. This suggests that further improvement to our model may be nessesary such as: $Y\ u, i\ = ?? + b\ i + ??\ u, i$ We could fit this model by using use the lm() function but as mentioned earlier, it would be very slow due to large dataset lm(rating ~ as.factor(movieId) + as.factor(userId))

Here is the code:

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - Mu_2 - b_i))
```

Now, let's examine how the RMSE has improved this time:

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = Mu_2 + b_i + b_u) %>%
  pull(pred)


model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_3_rmse))
rmse_results%>% knitr::kable()
```

method	RMSE
Just the average	1.0596481
Movie Effect Model	0.9431724
Movie + User Effects Model	0.8655154

The RMSE of the validation set is:

```
valid_pred_rating <- validation %>%
  left_join(movie_avgs , by = "movieId" ) %>%
  left_join(user_avgs , by = "userId") %>%
  mutate(pred = Mu_2 + b_i + b_u ) %>%
  pull(pred)

model_3_valid <- RMSE(validation$rating, valid_pred_rating)
rmse_results <-  bind_rows( rmse_results, data_frame(Method = "Validation Results" , RMSE = model_3_vali
rmse_results%>% knitr::kable()
```

method	RMSE	Method
Just the average	1.0596481	NA
Movie Effect Model	0.9431724	NA
Movie + User Effects Model	0.8655154	NA
NA	0.8666188	Validation Results

Conclusion

We've developed three strategies: a naive approach, a model focusing on movie-specific effects, and a more complex model integrating both user and movie effects. Among these, the third model yielded the most promising RMSE results. To delve deeper into analysis, I propose a more intricate prediction strategy leveraging the release year of each movie as a bias. This approach involves categorizing older movies, such as those from the 60s or 80s, as distinct genres, thereby refining our predictive model. For optimal precision, I recommend employing a linear model to accommodate these additional factors.