Movielens
Movie Recommendation System
A Harvard Capstone Project

Manoj Bijoor

March 18, 2021

Abstract

. . . this is the abstract text. . .

Contents

List of tables

List of figures

List of Equations

1 Project Overview: MovieLens - A Harvard Capstone Project

A movie recommendation system using the MovieLens dataset.

For this project, I will be creating a movie recommendation system using the MovieLens dataset, provided by GroupLens Research[1], a research lab in the Department of Computer Science and Engineering at the University of Minnesota, Twin Cities specializing in recommender systems, online communities, mobile and ubiquitous technologies, digital libraries, and local geographic information systems.

GroupLens Research[2] has collected and made available rating data sets from the MovieLens web site[3]. The data sets were collected over various periods of time, depending on the size of the set.

I will use the 10M version of the MovieLens dataset[4] to make the computation a little easier.

First, I will download the MovieLens data and run code provided to generate my datasets.

Second, I will train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

1.1 Create Train and Final Hold-out Test Sets

I will develop my algorithm using the edx set. For a final test of my final algorithm, I predict movie ratings in the validation set (the final hold-out test set) as if they were unknown. RMSE[5] will be used to evaluate how close my predictions are to the true values in the validation set (the final hold-out test set). My target is RMSE $<$ 0.86490.

1.1.1 Important: Data sets usage

The validation data (the final hold-out test set) will NOT be used for training, developing, or selecting my algorithm and it will ONLY be used for evaluating the RMSE of my final algorithm. The final hold-out test set will only be used at the end of my project with my final model. It will not be used to test the RMSE of multiple models during model development. I will split the edx data into separate training and test sets to design and test my algorithm.

1.2 Final Product

1.2.1 My submission for this project is three files:

1. My report in Rmd format
2. My report in PDF format (knit from my Rmd file)
3. A script in R format that generates my predicted movie ratings and RMSE score (contains all code and comments for my project)

The report documents the analysis and presents the findings, along with supporting statistics and figures. The report assumes that the reader is not familiar with the project or the data. The report includes the RMSE generated and the following sections:

1. an introduction/overview/executive summary section that describes the dataset and summarizes the goal of the project and key steps that were performed

[1] https://grouplens.org/

[2] https://grouplens.org/datasets/movielens/

[3] https://movielens.org

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

[5] https://en.wikipedia.org/wiki/Root-mean-square_deviation

2. a methods/analysis section that explains the process and techniques used, including data cleaning, data exploration and visualization, insights gained, and my modeling approach
3. a results section that presents the modeling results and discusses the model performance
4. a conclusion section that gives a brief summary of the report, its limitations and future work

2 Exploratory Data Analysis

2.1 Data Wrangling

Data wrangling[6], sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.

The main steps could be described as follows:
1. Discovering
2. Structuring
3. Cleaning
4. Enriching
5. Validating
6. Publishing

Let's perform the steps or combinations thereof starting with Initial data Exploration & Visualization in the next few subsections.

2.1.1 Initial data Exploration & Visualization

The first ten rows out of 9000055 rows of the Movielens data can be found in Table 1.

Table 1: Movielens data

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical
1	370	5	838984596	Naked Gun 33 1/3: The Final Insult (1994)	Action|Comedy

Each row represents a rating given by one user to one movie.

We can see the number of unique users that provided ratings and how many unique movies were rated. The unique genres here is based on a column called "genres" that includes every genre that applies to the movie. Some movies fall under several genres. Defining a category as whatever combination appears in this column, we are going to refer to this category as "unique genres" or simply "genres" from now on.

The number of unique users, movies and genres can be found in Table 2.

Table 2: Unique Users, Movies and Genres

unique_users	unique_movies	unique_genres
69878	10677	797

If we multiply the number of unique users by number of unique movies, we get a very large number, actually 746087406, yet our data table has 9000055 rows. This implies that not every user rated every movie. So

[6] https://en.wikipedia.org/wiki/Data_wrangling

we can think of these data as a very large matrix, with users on the rows and movies on the columns, with many empty cells. The ‘gather’ function permits us to convert it to this format, but if we try it for the entire matrix, it will crash R.

Let’s show the matrix of seven users ie: userId’s 13-20 and four movies in Table 3.

Table 3: Matrix of seven users and four movies

userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Silence of the Lambs, The (1991)
13	NA	NA	4	NA
16	NA	3	NA	NA
17	NA	NA	NA	5
18	NA	3	5	5
19	4	1	NA	NA

2.1.1.1 A Very Sparse Matrix You can think of the task of a recommendation system as filling in the 'NAs' in the table above. To see how sparse the matrix is, here is the matrix in Figure 1 for a random sample of 100 movies and 100 users with yellow indicating a user/movie combination for which we have a rating.

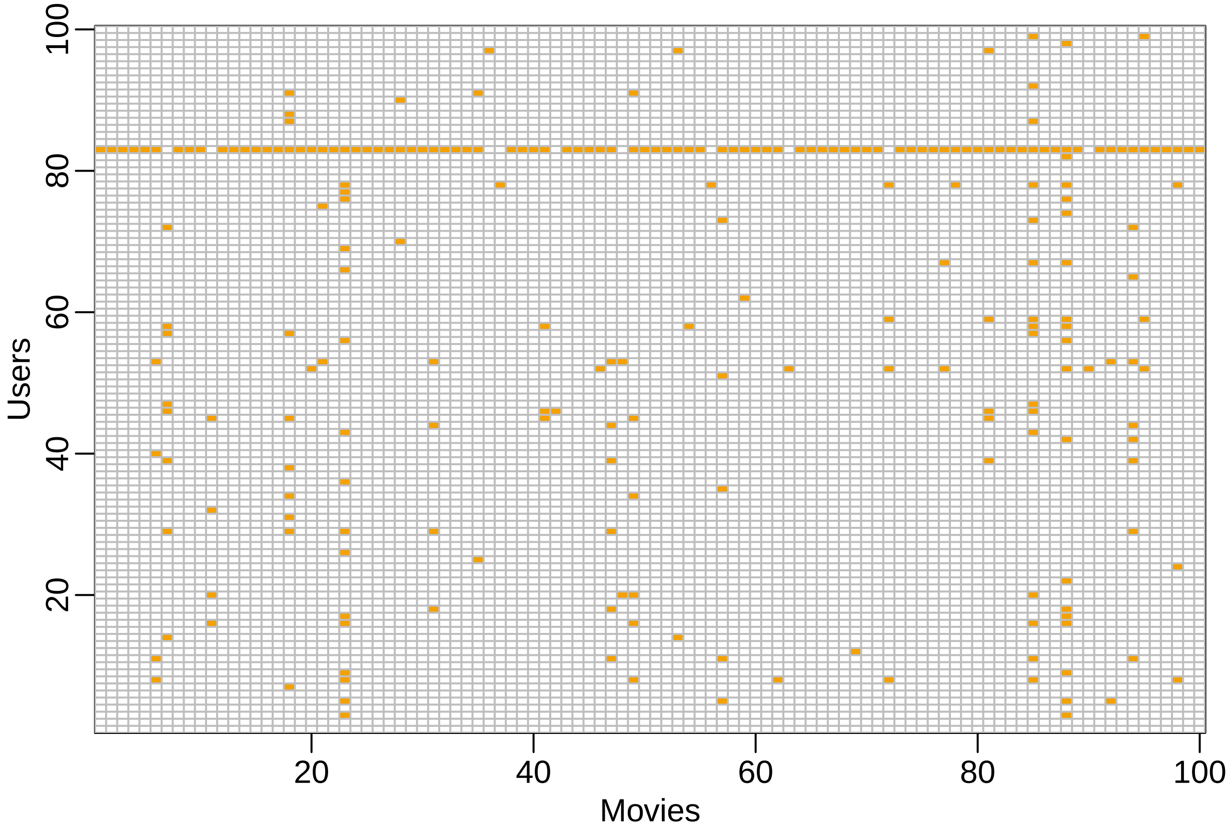


Figure 1: A Very Sparse Matrix

This machine learning challenge is quite complicated, because each outcome Y has a different set of predictors. To see this, note that if we are predicting the rating for movie i by user u, in principle, all other ratings related to movie i and by user u may be used as predictors, but different users rate different movies and a different number of movies. Furthermore, we may be able to use information from other movies that we have determined are similar to movie i or from users determined to be similar to user u. In essence, the entire matrix can be used as predictors for each cell.

2.1.1.2 General Properties of the data Let's look at some of the ***general properties*** of the data to better understand the challenges.

The ***first thing*** we notice is that some movies get rated more than others. Figure 2 shows the Movies getting rated distribution. This should not surprise us given that there are blockbuster movies watched by millions and artsy, independent movies watched by just a few:

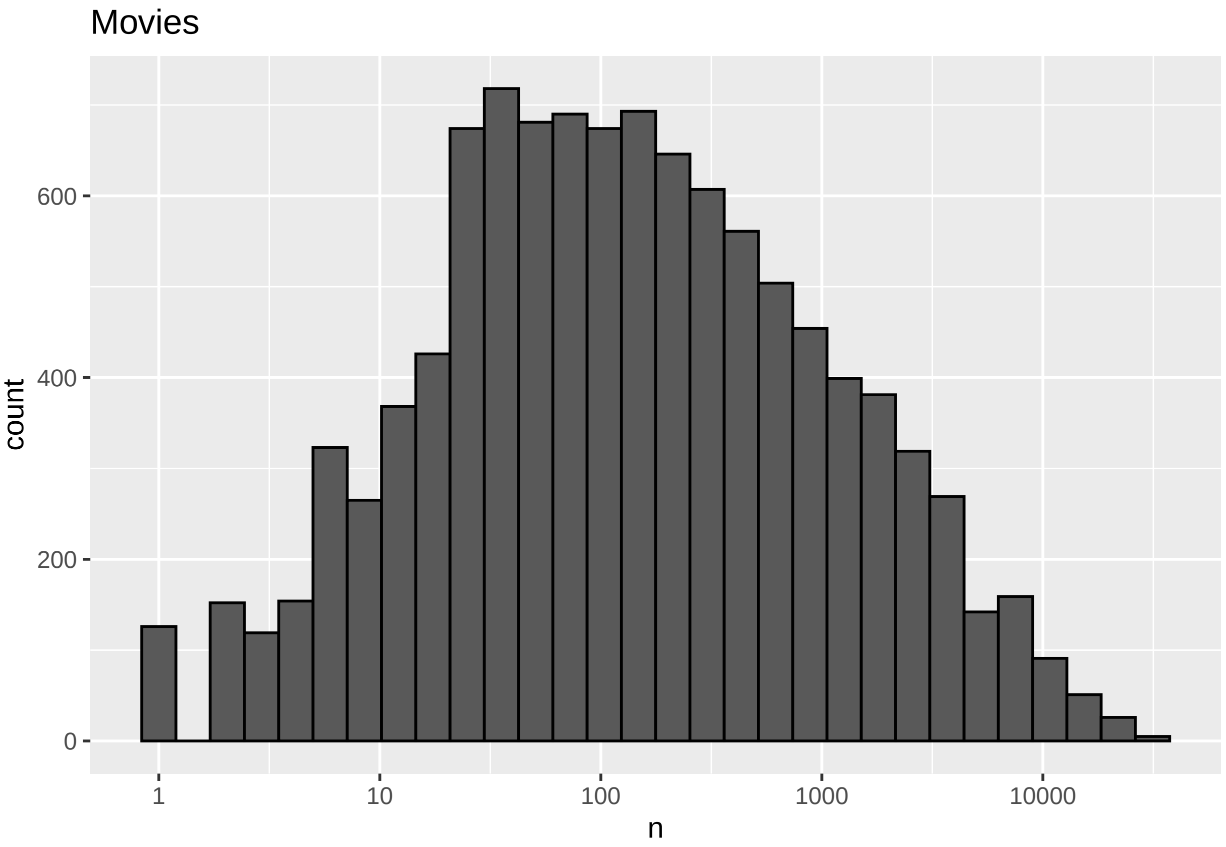


Figure 2: Movies getting rated distribution

Our ***second observation*** is that some users are more active than others at rating movies. Figure 3 shows Users rating movies distribution:

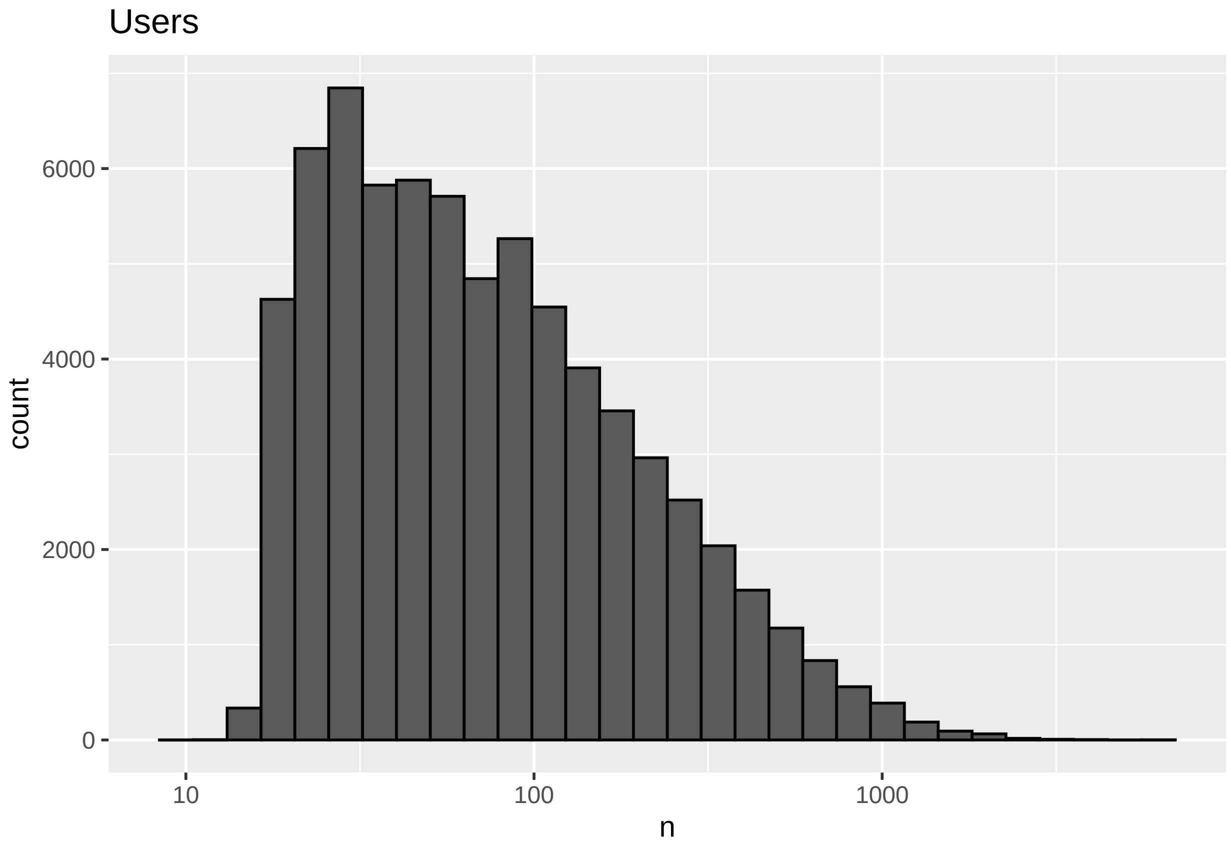


Figure 3: Users rating movies distribution

2.1.2 Further data Exploration, Visualization & Modification

2.1.2.1 Modify edx **Convert timestamp** in Movielens edx data Table 1 into date-time, a more readable and useful format named *rating_date* in Table 4 below.

Table 4: Movielens edx data with rating date-time

userId	movieId	rating	title	genres	rating_date
1	122	5	Boomerang (1992)	Comedy|Romance	1996-08-02 11:24:06
1	185	5	Net, The (1995)	Action|Crime|Thriller	1996-08-02 10:58:45
1	292	5	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1996-08-02 10:57:01
1	316	5	Stargate (1994)	Action|Adventure|Sci-Fi	1996-08-02 10:56:32
1	329	5	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1996-08-02 10:56:32

Split title in Movielens edx data Table 1 into title and year movie released, a more useful format named *movie_dt* in Table 5 below.

Table 5: Movielens edx data with movie release date

userId	movieId	rating	title	genres	rating_date	movie_dt
1	122	5	Boomerang	Comedy|Romance	1996-08-02 11:24:06	1992
1	185	5	Net, The	Action|Crime|Thriller	1996-08-02 10:58:45	1995
1	292	5	Outbreak	Action|Drama|Sci-Fi|Thriller	1996-08-02 10:57:01	1995
1	316	5	Stargate	Action|Adventure|Sci-Fi	1996-08-02 10:56:32	1994
1	329	5	Star Trek: Generations	Action|Adventure|Drama|Sci-Fi	1996-08-02 10:56:32	1994

2.1.2.2 Modify validation, repeat above steps

Convert timestamp in Movielens validation data Table 1 into date-time, a more readable and useful format named *rating_date* in Table 6 below.

Table 6: Movielens validation data with rating date-time

<table>
<tr><th>userId</th><th>movieId</th><th>rating</th><th>title</th><th>genres</th><th>rating_date</th></tr>
<tr><td>1</td><td>231</td><td>5</td><td>Dumb & Dumber (1994)</td><td>Comedy</td><td>1996-08-02 10:56:32</td></tr>
<tr><td>1</td><td>480</td><td>5</td><td>Jurassic Park (1993)</td><td>Action|Adventure|Sci-Fi|Thriller</td><td>1996-08-02 11:00:53</td></tr>
<tr><td>1</td><td>586</td><td>5</td><td>Home Alone (1990)</td><td>Children|Comedy</td><td>1996-08-02 11:07:48</td></tr>
<tr><td>2</td><td>151</td><td>3</td><td>Rob Roy (1995)</td><td>Action|Drama|Romance|War</td><td>1997-07-07 03:34:10</td></tr>
<tr><td>2</td><td>858</td><td>2</td><td>Godfather, The (1972)</td><td>Crime|Drama</td><td>1997-07-07 03:20:45</td></tr>
</table>

Split title in Movielens validation data Table 1 into title and year movie released, a more useful format named *movie_dt* in Table 7 below.

Table 7: Movielens validation data with movie release date

<table>
<tr><th>userId</th><th>movieId</th><th>rating</th><th>title</th><th>genres</th><th>rating_date</th><th>movie_dt</th></tr>
<tr><td>1</td><td>231</td><td>5</td><td>Dumb & Dumber</td><td>Comedy</td><td>1996-08-02 10:56:32</td><td>1994</td></tr>
<tr><td>1</td><td>480</td><td>5</td><td>Jurassic Park</td><td>Action|Adventure|Sci-Fi|Thriller</td><td>1996-08-02 11:00:53</td><td>1993</td></tr>
<tr><td>1</td><td>586</td><td>5</td><td>Home Alone</td><td>Children|Comedy</td><td>1996-08-02 11:07:48</td><td>1990</td></tr>
<tr><td>2</td><td>151</td><td>3</td><td>Rob Roy</td><td>Action|Drama|Romance|War</td><td>1997-07-07 03:34:10</td><td>1995</td></tr>
<tr><td>2</td><td>858</td><td>2</td><td>Godfather, The</td><td>Crime|Drama</td><td>1997-07-07 03:20:45</td><td>1972</td></tr>
</table>

2.1.2.3 Genres combinations per movie - a closer look The movielens data Table 8 also has a genres column. This column includes every genre that applies to the movie. Some movies fall under several genres. We define a category of genres as whatever combination of genres appears in this column, and refer to it as simply "genres".

Here we keep only categories with more than 1,000 ratings. Then compute the average and standard error for each category, and plot these as error bar plots. See Figure 4 :

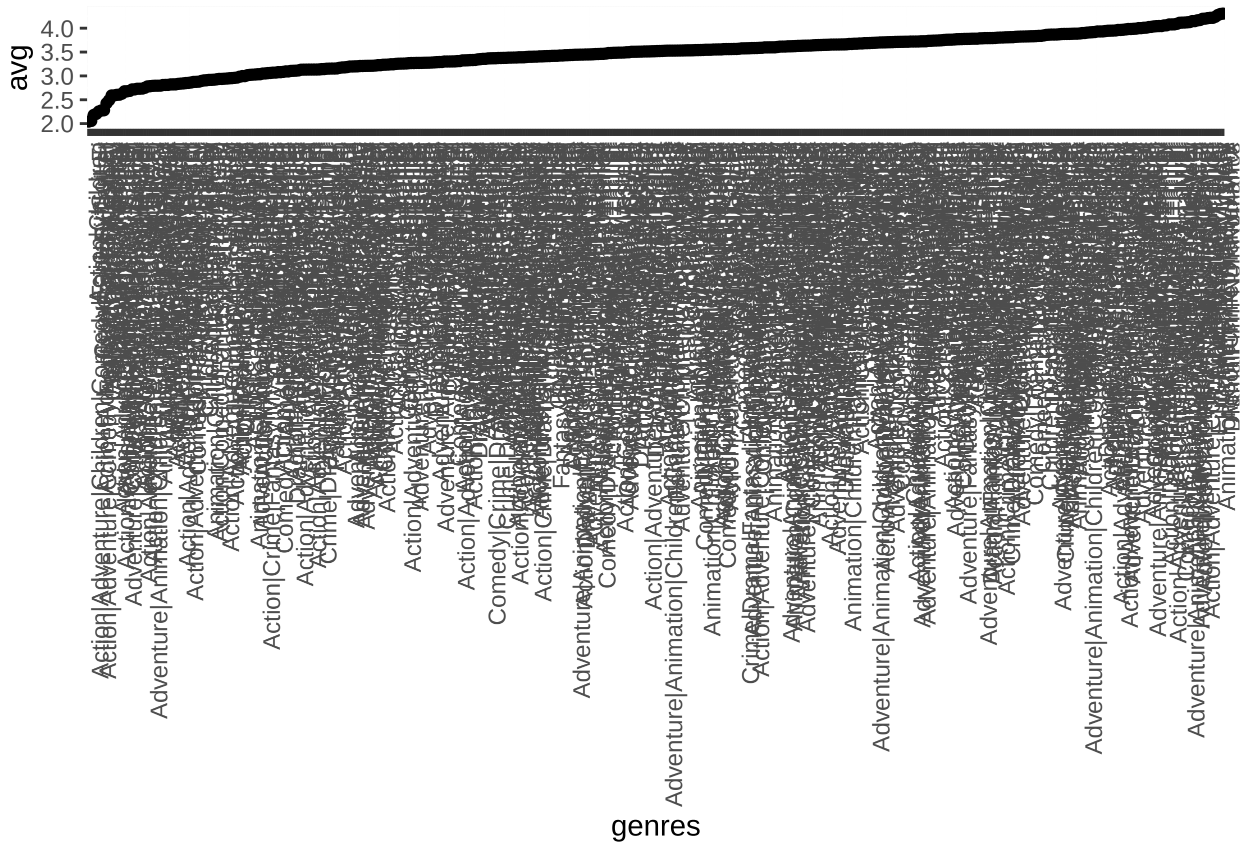


Figure 4: Movies genres error bar plots

The plot shows strong evidence of a genre effect.

2.1.2.4 Movie Rating Date-Time - a closer look The Movielens edx data Table 1 also includes a time stamp. This variable represents the time and date in which the rating was provided. The units are seconds since January 1, 1970. We create a new column date with the date named *rating_date* in subsection Modify edx to get Table 5 .

We compute the average rating for each week and plot this average against day. See Figure 5 :

```
‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```

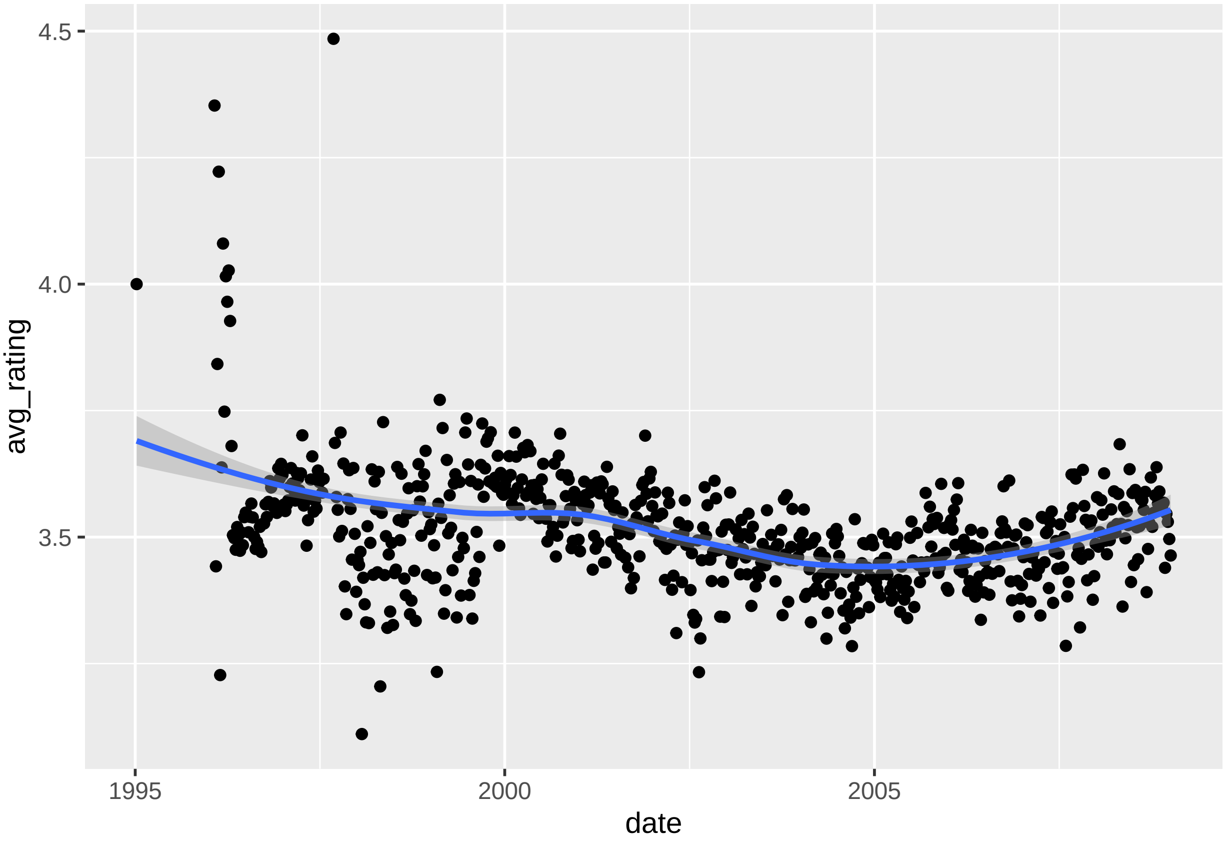


Figure 5: Movies average ratings for each week versus day

The plot shows some evidence of a time effect. If we define $d_{u,i}$ as the day for user’s u rating of movie i, then the following model given by Equation 1 is most appropriate:

$$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \epsilon_{u,i}, \text{ with f a smooth function of } d_{u,i} \quad (1)$$

Modify edx Let’s update the ***edx*** table with a new column for the average rating for each week and another column for the day rounded to the nearest value of the week to get Table 8 below:

Table 8: Movielens edx data with average rating due to rating time effect

userId	movieId	rating	title	genres	rating_date	movie_dt	date	avg_rating
1	122	5	Boomerang	Comedy|Romance	1996-08-02 11:24:06	1992	1996-08-04	3.538801
1	185	5	Net, The	Action|Crime|Thriller	1996-08-02 10:58:45	1995	1996-08-04	3.538801
1	292	5	Outbreak	Action|Drama|Sci-Fi|Thriller	1996-08-02 10:57:01	1995	1996-08-04	3.538801
1	316	5	Stargate	Action|Adventure|Sci-Fi	1996-08-02 10:56:32	1994	1996-08-04	3.538801
1	329	5	Star Trek: Generations	Action|Adventure|Drama|Sci-Fi	1996-08-02 10:56:32	1994	1996-08-04	3.538801

TODO: Repeat above for validation data as well and somehow add this to the modelling section

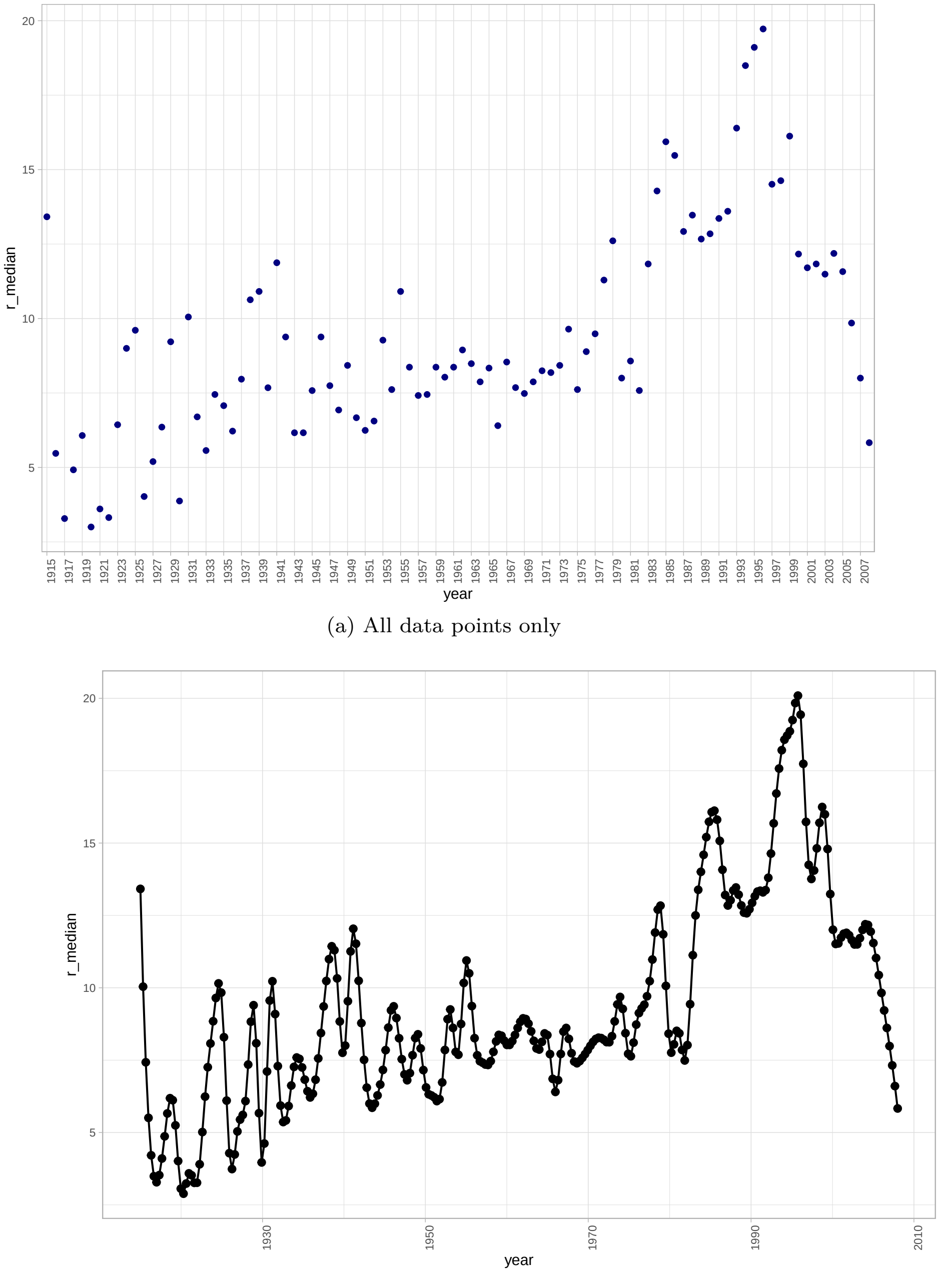
Modify validation We need to do the above ***avg_rating_time_effect*** update for the validation data as well. Let's update the ***validation*** table to get Table 9 below:

Table 9: Movielens validation data with average rating due to rating time effect

userId	movieId	rating	title	genres	rating_date	movie_dt	date	avg_rating
1	231	5	Dumb & Dumber	Comedy	1996-08-02 10:56:32	1994	1996-08-04	3.555820
1	480	5	Jurassic Park	Action|Adventure|Sci-Fi|Thriller	1996-08-02 11:00:53	1993	1996-08-04	3.555820
1	586	5	Home Alone	Children|Comedy	1996-08-02 11:07:48	1990	1996-08-04	3.555820
2	151	3	Rob Roy	Action|Drama|Romance|War	1997-07-07 03:34:10	1995	1997-07-06	3.606571
2	858	2	Godfather, The	Crime|Drama	1997-07-07 03:20:45	1972	1997-07-06	3.606571

2.1.2.5 Movie Release Date - a closer look Computing the number of ratings for each movie and then plotting it against the year the movie came out, that is the release date and using the square root transformation on the counts using Table 5 , we get see Figure 6 :

TODO: Align images



(a) All data points only

(b) Smooth line through all data points

Figure 6: Ratings Movie Release Date - All dates

we see that, on average, movies that came out after 1993 get more ratings. We also see that with newer movies, starting in 1993, the number of ratings decreases with year: the more recent a movie is, the less time users have had to rate it.

Among movies that came out in 1993 or later, we select the top 25 movies with the highest average number of ratings per year (n/year) and calculate the average rating of each of them. To calculate number of ratings per year, use 2018 as the end year. See Figure 7 :

```
‘geom_smooth()‘ using method = ’loess’ and formula ’y ~ x’
```

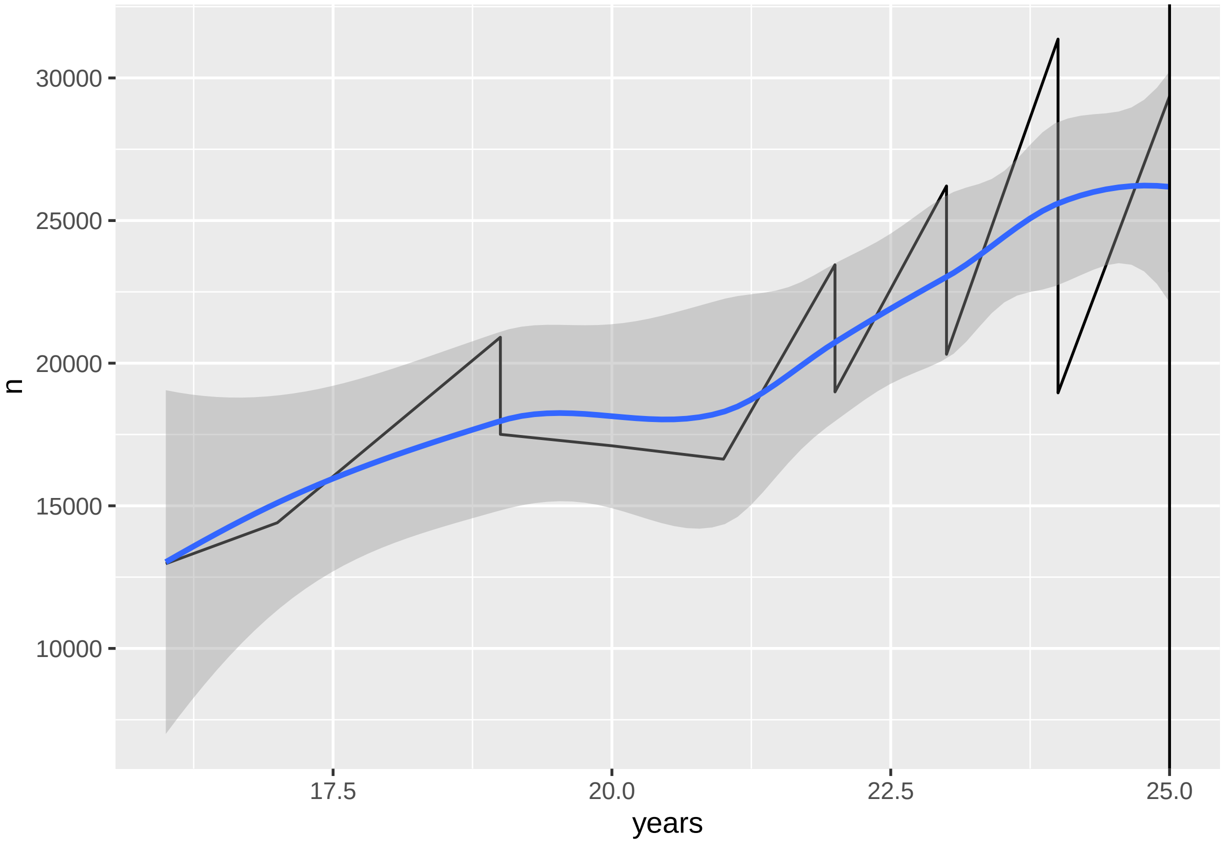


Figure 7: 25 Movies with the most ratings per year and their average rating post 1993

We see that the most rated movies tend to have above average ratings. This is not surprising: more people watch popular movies. To confirm this, we stratify the post 1993 movies by ratings per year and compute their average ratings. Figure 8 is a plot of average ratings versus ratings per year showing an estimate of the trend.

We see that the more a movie is rated, the higher the rating.

Post-1993 movies

```
`geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

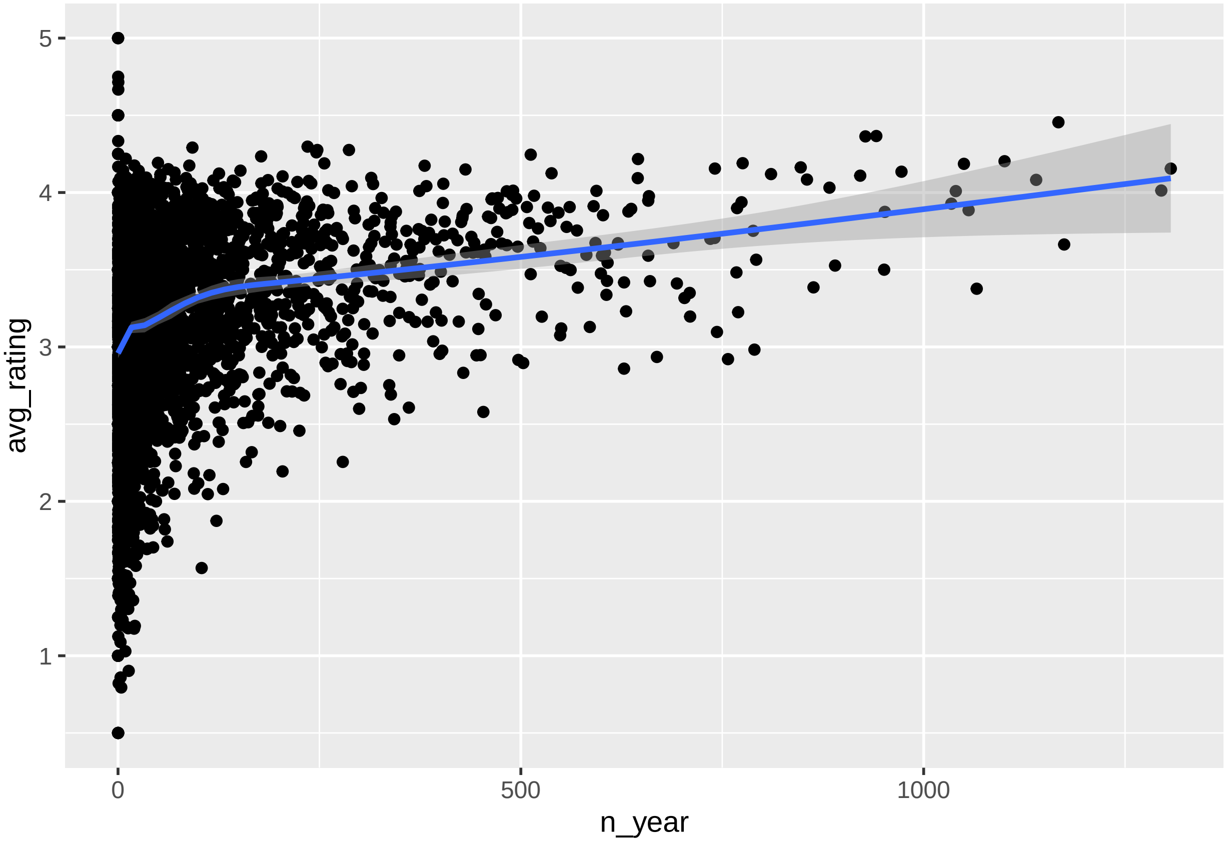


Figure 8: Movies average ratings versus ratings per year post 1993

Pre-1993 movies
Compare Pre-1993 movies trend shown here in Figure 9 Versus Post-1993 movies trend in Figure 8 above.

```
‘geom_smooth()‘ using method = ’gam’ and formula ’y ~ s(x, bs = "cs")’
```

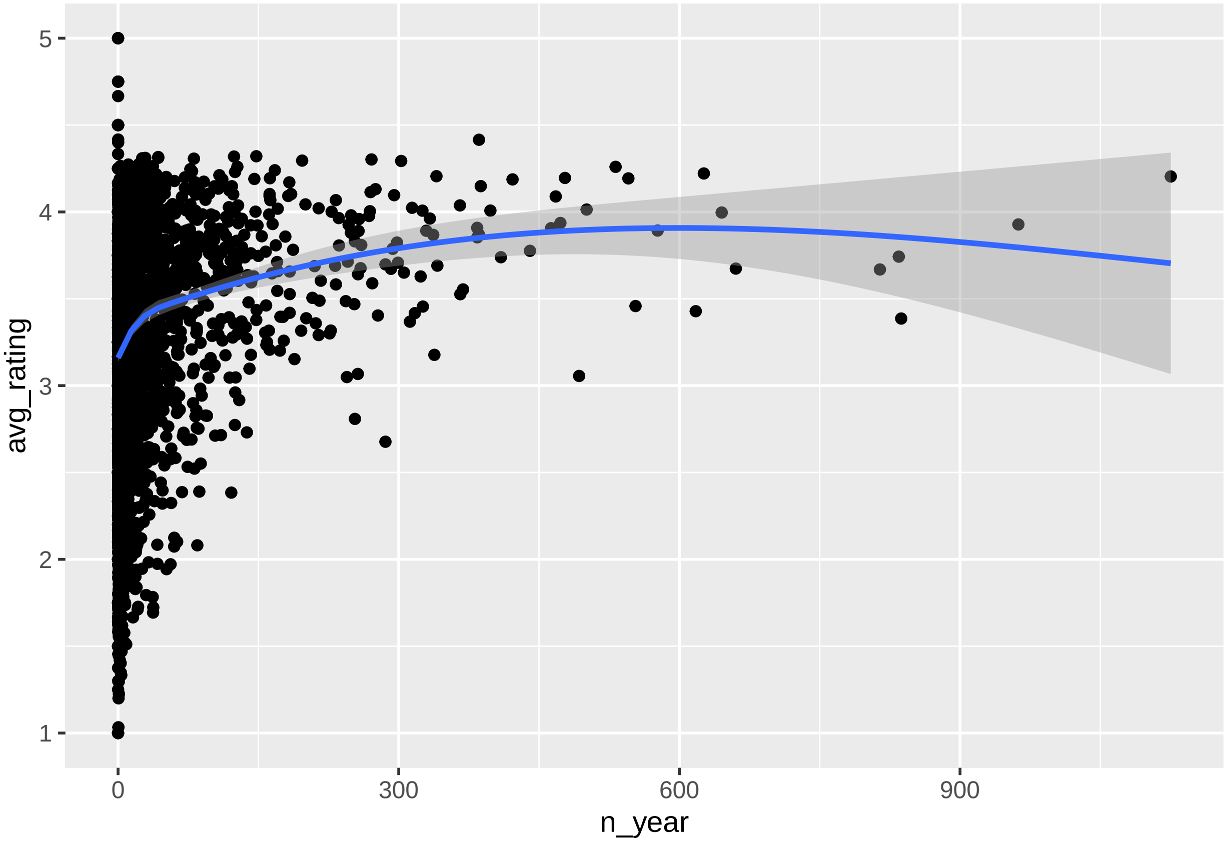


Figure 9: Movies average ratings versus ratings per year pre 1993

Modify edx data for Release Date Effect We stratify the movies by ratings per year and compute their average ratings based on what we learnt above, where we confirmed our intuition that more people watch popular movies. Finally edx data table looks as shown in Table 10 below. Figure 10"} is a plot of average ratings versus ratings per year showing an estimate of the trend.

All Years

```
‘geom_smooth()‘ using method = ’gam’ and formula ’y ~ s(x, bs = "cs")’
```

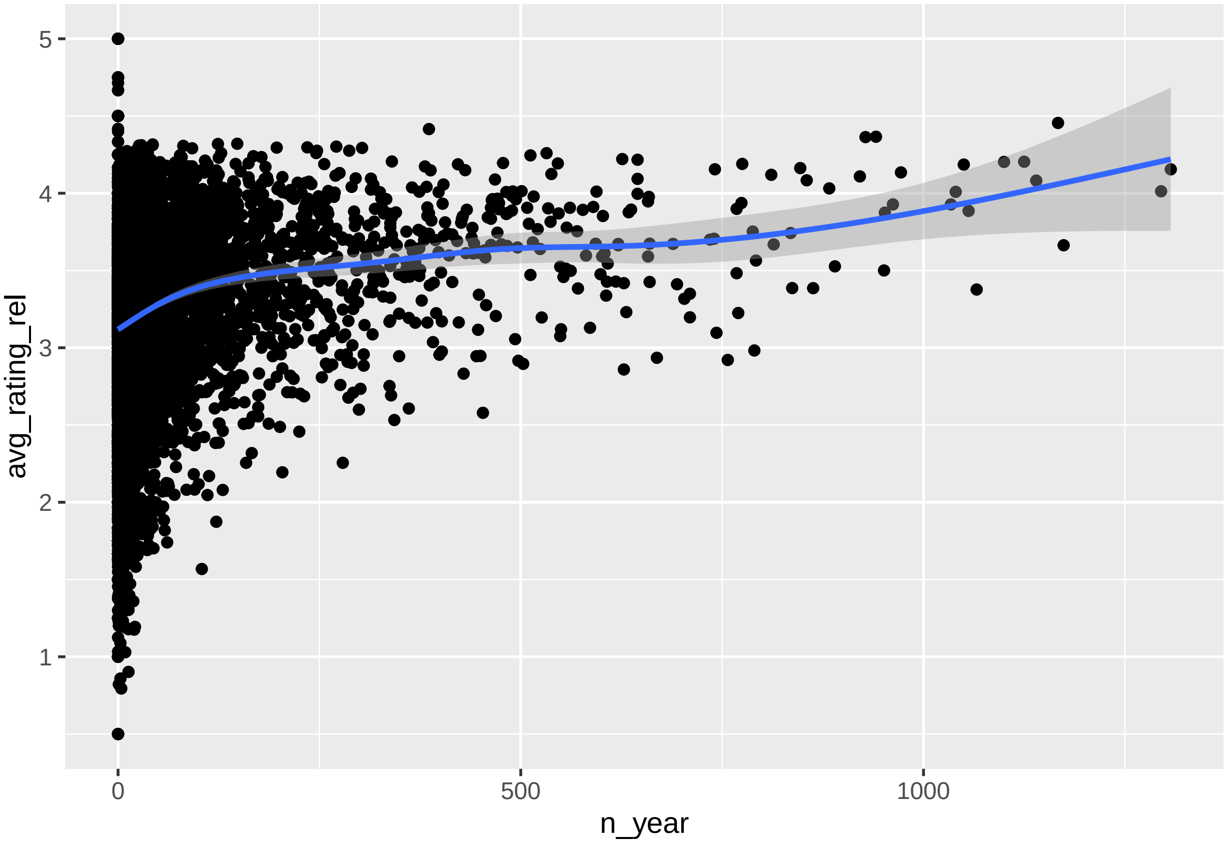


Figure 10: Movies average ratings versus ratings per year for all years for edx

Table 10: Movielens edx data with average rating due to release date effect

userId	movieId	rating	title	genres	rating_date	movie_dt	date	avg_rating	avg_rating_rel	n	years	n_year
1	122	5	Boomerang	Comedy|Romance	1996-08-02 11:24:06	1992	1996-08-04	3.538801	2.858586	2178	26	84
1	185	5	Net, The	Action|Crime|Thriller	1996-08-02 10:58:45	1995	1996-08-04	3.538801	3.129334	13469	23	586
1	292	5	Outbreak	Action|Drama|Sci-Fi|Thriller	1996-08-02 10:57:01	1995	1996-08-04	3.538801	3.418011	14447	23	628
1	316	5	Stargate	Action|Adventure|Sci-Fi	1996-08-02 10:56:32	1994	1996-08-04	3.538801	3.349677	17030	24	710
1	329	5	Star Trek: Generations	Action|Adventure|Drama|Sci-Fi	1996-08-02 10:56:32	1994	1996-08-04	3.538801	3.337457	14550	24	606

```
tibble [9,000,055 x 13] (S3: tbl_df/tbl/data.frame)
 $ userId        : int [1:9000055] 1 1 1 1 1 1 1 1 1 1 ...
 $ movieId       : num [1:9000055] 122 185 292 316 329 355 356 362 364 370 ...
 $ rating        : num [1:9000055] 5 5 5 5 5 5 5 5 5 5 ...
 $ title         : chr [1:9000055] "Boomerang " "Net, The " "Outbreak " "Stargate " ...
 $ genres        : chr [1:9000055] "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thrill
 $ rating_date   : POSIXct[1:9000055], format: "1996-08-02 11:24:06" "1996-08-02 10:58:45" ...
 $ movie_dt      : num [1:9000055] 1992 1995 1995 1994 1994 ...
 $ date          : POSIXct[1:9000055], format: "1996-08-04" "1996-08-04" ...
 $ avg_rating    : num [1:9000055] 3.54 3.54 3.54 3.54 3.54 ...
 $ avg_rating_rel: num [1:9000055] 2.86 3.13 3.42 3.35 3.34 ...
 $ n             : int [1:9000055] 2178 13469 14447 17030 14550 4831 31079 3612 18921 7331 ...
 $ years         : num [1:9000055] 26 23 23 24 24 24 24 24 24 24 ...
 $ n_year        : num [1:9000055] 84 586 628 710 606 ...
```

Modify validation data for Release Date Effect We need to do the above ***avg_rating_rel_effect*** update for the validation data as well. Let's update the ***validation*** table to get Table 11 below:

All Years

Table 11: Movielens validation data with average rating due to release date effect

userId	movieId	rating	title	genres	rating_date	movie_dt	date	avg_rating	avg_rating_rel	n	years	n_year
1	231	5	Dumb & Dumber	Comedy	1996-08-02 10:56:32	1994	1996-08-04	3.555820	2.953281	1798	24	75
1	480	5	Jurassic Park	Action|Adventure|Sci-Fi|Thriller	1996-08-02 11:00:53	1993	1996-08-04	3.555820	3.643993	3271	25	131
1	586	5	Home Alone	Children|Comedy	1996-08-02 11:07:48	1990	1996-08-04	3.555820	3.074550	1556	28	56
2	151	3	Rob Roy	Action|Drama|Romance|War	1997-07-07 03:34:10	1995	1997-07-06	3.606571	3.571984	771	23	34
2	858	2	Godfather, The	Crime|Drama	1997-07-07 03:20:45	1972	1997-07-06	3.606571	4.412675	2067	46	45

```
tibble [999,999 x 13] (S3: tbl_df/tbl/data.frame)
 $ userId        : int [1:999999] 1 1 1 2 2 2 3 3 4 4 ...
 $ movieId       : num [1:999999] 231 480 586 151 858 ...
 $ rating        : num [1:999999] 5 5 5 3 2 3 3.5 4.5 5 3 ...
 $ title         : chr [1:999999] "Dumb & Dumber " "Jurassic Park " "Home Alone " "Rob Roy " ...
 $ genres        : chr [1:999999] "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action
 $ rating_date   : POSIXct[1:999999], format: "1996-08-02 10:56:32" "1996-08-02 11:00:53" ...
 $ movie_dt      : num [1:999999] 1994 1993 1990 1995 1972 ...
 $ date          : POSIXct[1:999999], format: "1996-08-04" "1996-08-04" ...
 $ avg_rating    : num [1:999999] 3.56 3.56 3.56 3.61 3.61 ...
 $ avg_rating_rel: num [1:999999] 2.95 3.64 3.07 3.57 4.41 ...
 $ n             : int [1:999999] 1798 3271 1556 771 2067 862 2545 947 1869 776 ...
 $ years         : num [1:999999] 24 25 28 23 46 21 28 17 23 24 ...
 $ n_year        : num [1:999999] 75 131 56 34 45 41 91 56 81 32 ...
```

3 Analysis - Model Building and Evaluation

3.1 Split the edx data into separate training and test sets

We will develop our algorithm using the edx set only.
We will split the edx data into separate training and test sets to design and test our algorithm, namely train_set and test_set.

3.1.1 Loss function

For a final test of our algorithm, we predict movie ratings in the test set as if they were unknown. RMSE[7] (residual mean squared error/root mean square error), the typical error loss, will be used to evaluate how close our predictions are to the true values in the validation set.
We define $y_{u,i}$ as the rating for movie i by user u and denote our prediction with $\hat{y}_{u,i}$.

The RMSE is then defined as Equation 2:

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

with N being the number of user/movie combinations and the sum occurring over all these combinations.

Remember that we can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good.

Let's write a function that computes the RMSE for vectors of ratings and their corresponding predictors:

```r
RMSE <- function(true_ratings, predicted_ratings) {
    sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

[7] https://en.wikipedia.org/wiki/Root-mean-square_deviation

3.2 Model 1: A first naive "mean" model

Let's start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like Equation 3:

$$Y_{i,i} = \mu + \epsilon_{u,i} \tag{3}$$

with $\epsilon_{u,i}$ independent errors sampled from the same distribution centered at 0 and μ the "true" rating for all movies. We know that the estimate that minimizes the RMSE is the least squares estimate of μ and, in this case, is the average of all ratings:

```
(mu_hat <- mean(train_set$rating))
[1] 3.512482
```

If we predict all unknown ratings with $\hat{\mu}$ we obtain the following RMSE:

```
(model_1_rmse <- RMSE(test_set$rating, mu_hat))
[1] 1.059904
```

Keep in mind that if we plug in any other number, we get a higher RMSE. For example:

```
predictions <- rep(2.5, nrow(test_set))
RMSE(test_set$rating, predictions)
[1] 1.465736

predictions <- rep(3, nrow(test_set))
RMSE(test_set$rating, predictions)
[1] 1.177271

predictions <- rep(4, nrow(test_set))
RMSE(test_set$rating, predictions)
[1] 1.166678
```

From looking at the distribution of ratings, we can visualize that this is the standard deviation of that distribution. We get a RMSE of about 1. Our target is RMSE < 0.86490. So we can definitely do better!

3.2.1 Results Model 1

As we go along, we will be comparing different approaches. Let's start by creating a results table with this naive approach to get Table 12:

Table 12: RMSE Results Model 1

Index	Method	RMSE
1	Just the average	1.059904

3.3 Model 2: Movie effects

We know from experience that some movies are just generally rated higher than others. This intuition, that different movies are rated differently, is confirmed by data. We can augment our previous model by adding the term b_i to represent average ranking for movie i and would look like Equation 4:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i} \tag{4}$$

Statistics textbooks refer to the bs as effects or *"bias"*.

We can again use least squares to estimate the b_i in the following way to get Equation 5:

$$fit \leftarrow lm(rating \ \sim \ as.factor(userId), \ data = train\_set) \tag{5}$$

Because there are thousands of b_i as each movie gets one, the lm() function will be very slow here. We therefore will not run the code above.

But in this particular situation, we know that the least squares estimate $\hat{b_i}$ is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i. ***So we can compute them this way (we will drop the hat notation in the code to represent estimates going forward)***:

$$\hat{b_i} = \overline{y_{u,i} - \hat{\mu}} \tag{6}$$

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

We can see that these estimates vary substantially, see Figure 11

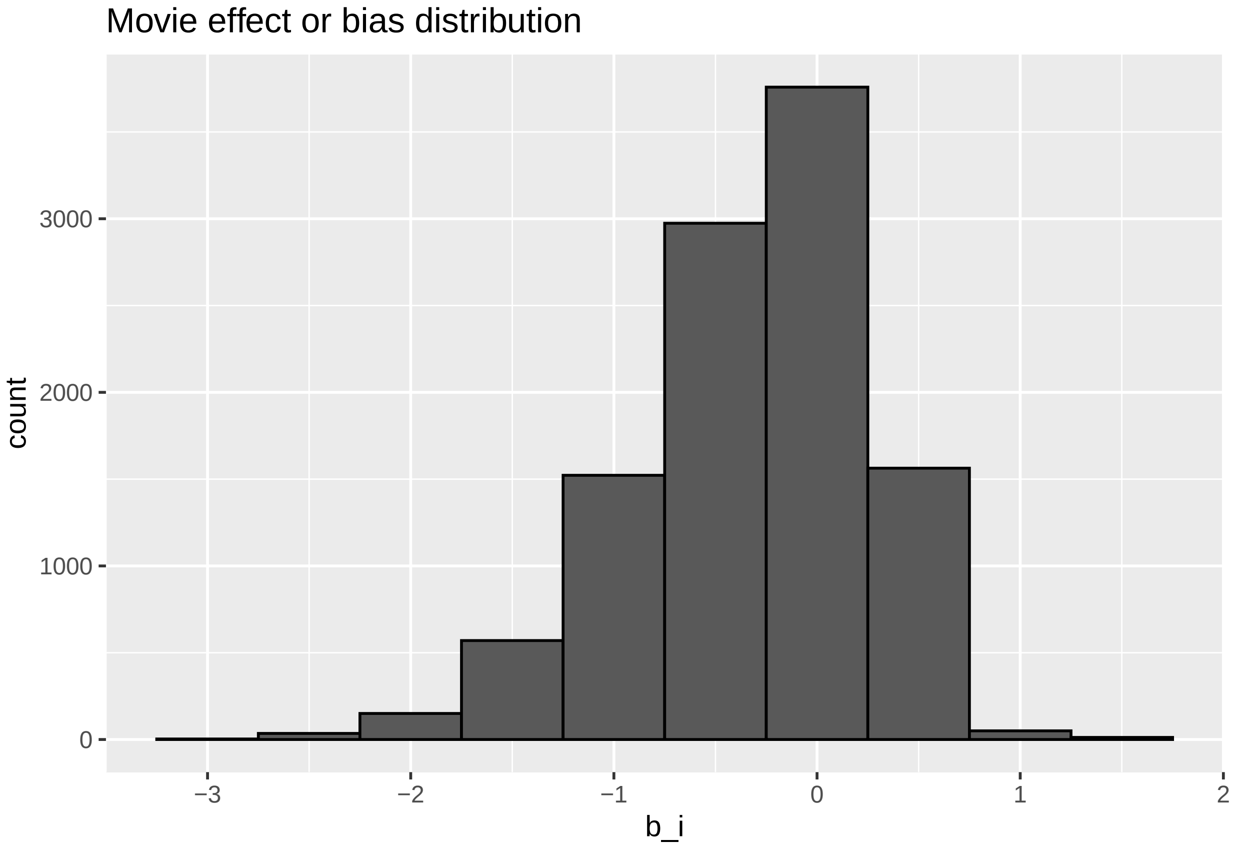


Figure 11: Movie effect or bias distribution

Remember $\hat{\mu}$=3.5 so a b_i=1.5 implies a perfect five star rating. Let's see how much our prediction improves once we use $\hat{y_{u,i}} = \hat{\mu} + \hat{b_i}$:

```
predicted_ratings_model_2 <- mu + test_set %>% left_join(movie_avgs,
    by = "movieId") %>% .$b_i
(model_2_rmse <- RMSE(predicted_ratings_model_2, test_set$rating))
[1] 0.9437429
```

3.3.1 Results Model 1-2

Let's add the movie effects model to our results table to get Table 13

Table 13: RMSE Results Model 1-2

Index	Method	RMSE
1	Just the average	1.0599043
2	Movie Effect Model	0.9437429

3.4 Model 3: User effects

Let's compute b_u the average rating for user u for those that have rated over 100 movies, see Figure 12

```
train_set %>% group_by(userId) %>% summarize(b_u = mean(rating)) %>%
    filter(n() >= 100) %>% ggplot(aes(b_u)) + geom_histogram(bins = 30,
    color = "black") + ggtitle("Average rating for users who have rated over 100 movies")
```

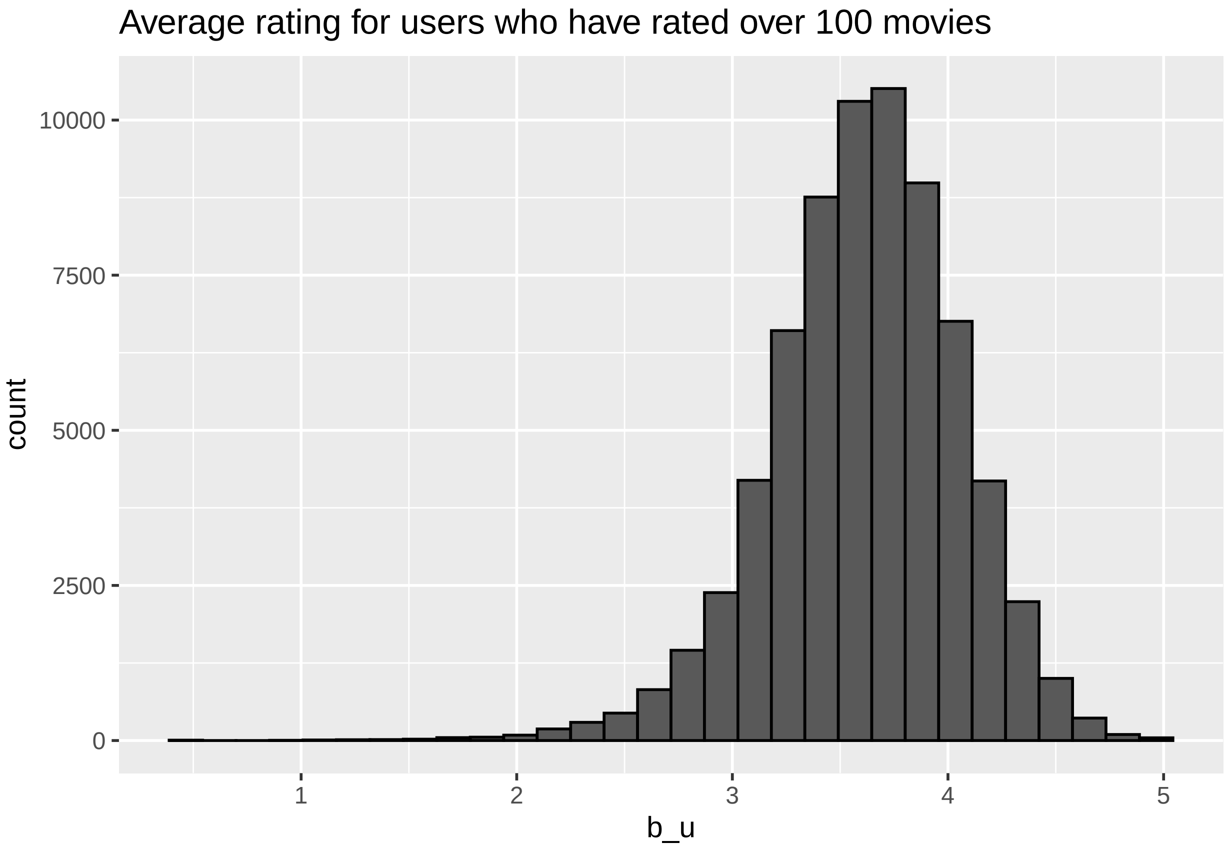


Figure 12: Average rating for users who have rated over 100 movies

Let's compute b_u the average rating for user u for those that have rated any movies, see Figure 13

```
train_set %>% group_by(userId) %>% summarize(b_u = mean(rating)) %>%
    ggplot(aes(b_u)) + geom_histogram(bins = 30, color = "black") +
    ggtitle("Average rating for users who have rated any movies")
```

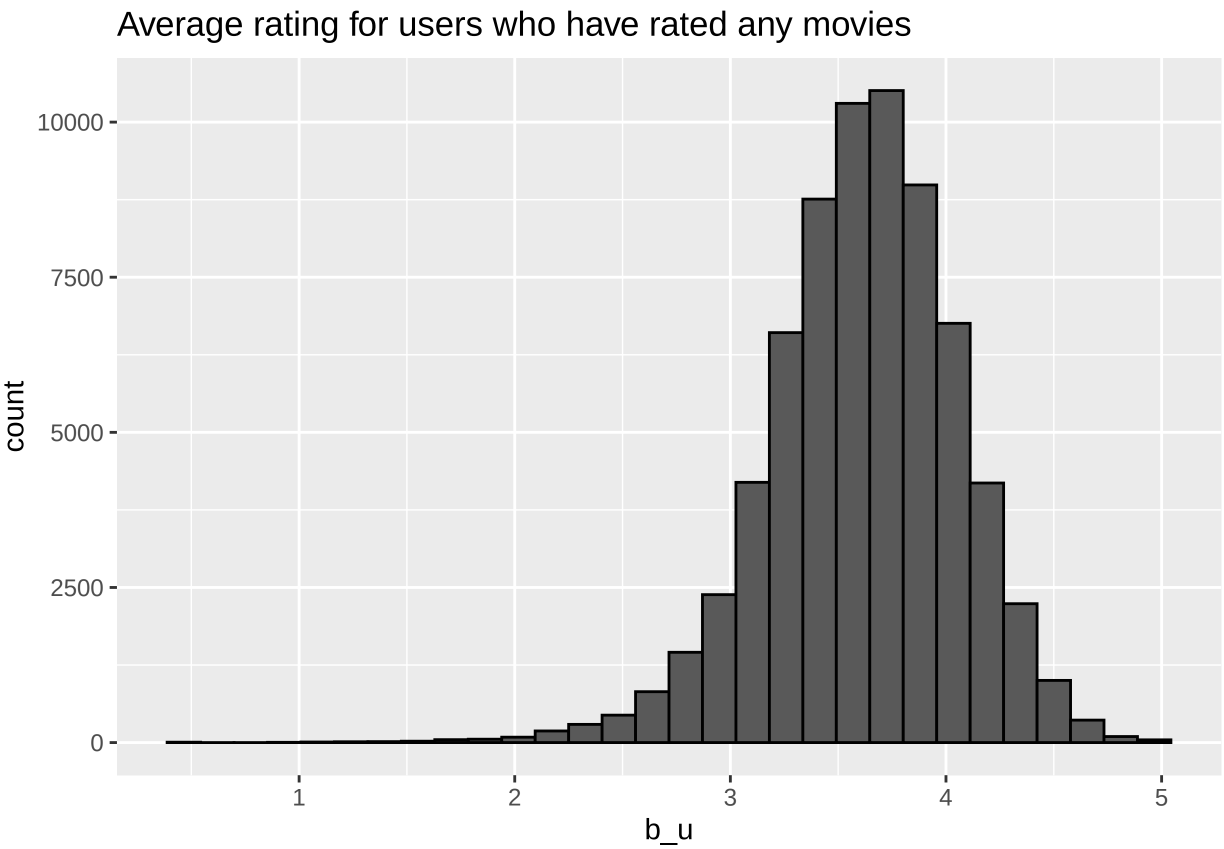


Figure 13: Average rating for users who have rated any movies

Notice that there is substantial variability across users as well: some users are very cranky and others love every movie. This implies that a further improvement to our model may be as shown in Equation 7:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i} \tag{7}$$

where b_u is a user-specific effect. Now if a cranky user (negative b_u) rates a great movie (positive b_i), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

To fit this model, we could again use lm() as shown in Equation 8:

$$fit \leftarrow lm(rating\ \sim\ as.factor(movieId) + as.factor(userId),\ data = train\_set) \tag{8}$$

but, for the reasons described earlier, we won't. Instead, we will compute an approximation by computing $\hat{\mu}$ and $\hat{b_i}$ and estimating $\hat{b_u}$ as the average of $y_{u,i} - \hat{\mu} - \hat{b_i}$:

$$\hat{b_u} = \overline{y_{u,i} - \hat{\mu} - \hat{b_i}} \tag{9}$$

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

We can see that these estimates vary substantially, see Figure 14

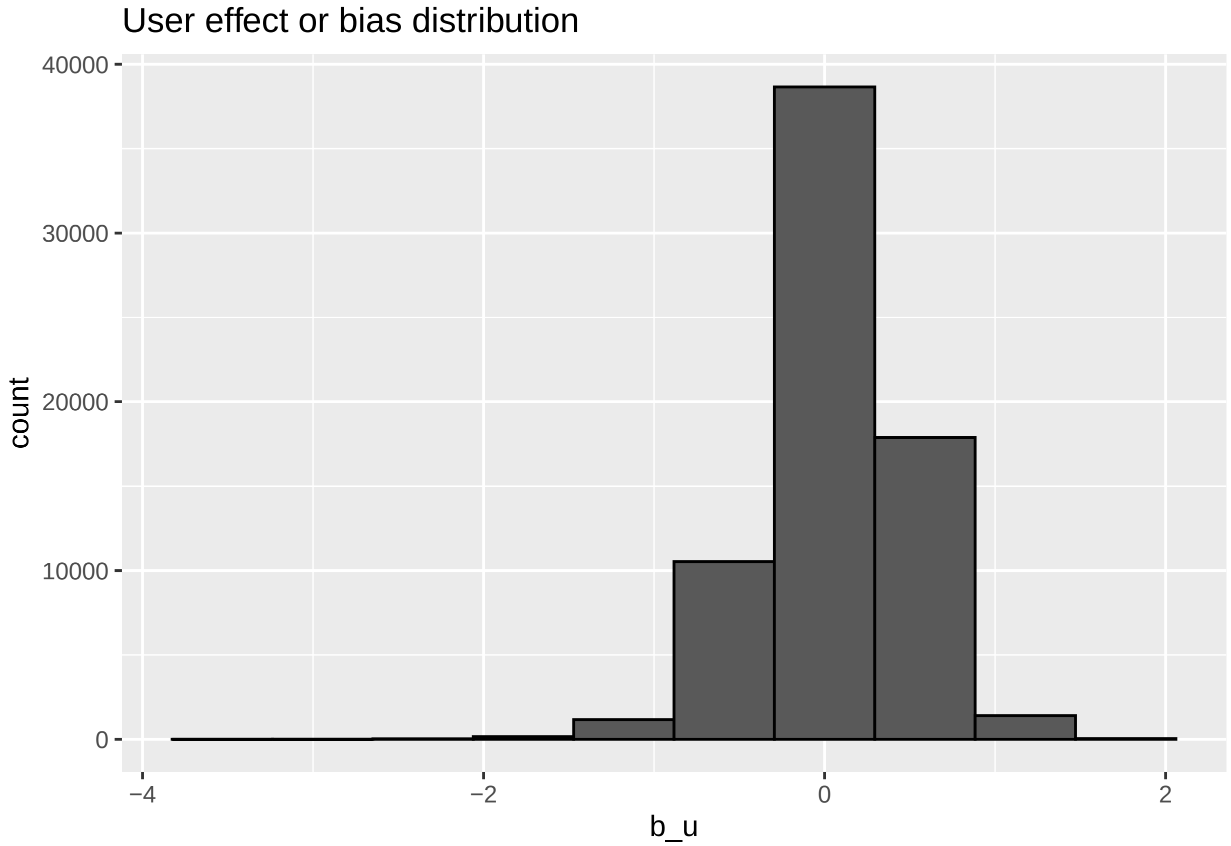


Figure 14: User effect or bias distribution

We can now construct predictors and see how much the RMSE improves:

```
predicted_ratings_model_3 <- test_set %>% left_join(movie_avgs,
    by = "movieId") %>% left_join(user_avgs, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>% .$pred
(model_3_rmse <- RMSE(predicted_ratings_model_3, test_set$rating))
[1] 0.865932
```

3.4.1 Results Table Model 1-3

Let’s add the user effects model to our results table to get Table 14

Table 14: RMSE Results Model 1-3

Index	Method	RMSE
1	Just the average	1.0599043
2	Movie Effect Model	0.9437429
3	Movie + User Effects Model	0.8659320

3.5 Model 4: Genre effects

The movielens data also has a genres column. This column includes every genre that applies to the movie. Some movies fall under several genres. Define a category of genres as whatever combination of genres appears in this column. We will refer to this category as simply "genres".

There is strong evidence of a genre effect as we have shown earlier in Figure 4, and in this section below in Figure 16. If we define $g_{u,i}$ as the genre for u user's rating of movie i, then the following model as shown in Equation 10 is most appropriate:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}\beta_k + \epsilon_{u,i} \tag{10}$$

$$\text{with } x_{u,i}^k = 1 \text{ if } g_{u,i} \text{ is genre *k*}$$

```
train_set %>% group_by(genres) %>% summarize(mu_g = mean(rating)) %>%
    ggplot(aes(mu_g)) + geom_histogram(bins = 30, color = "black") +
    ggtitle("Average rating for movies of category genres")
```

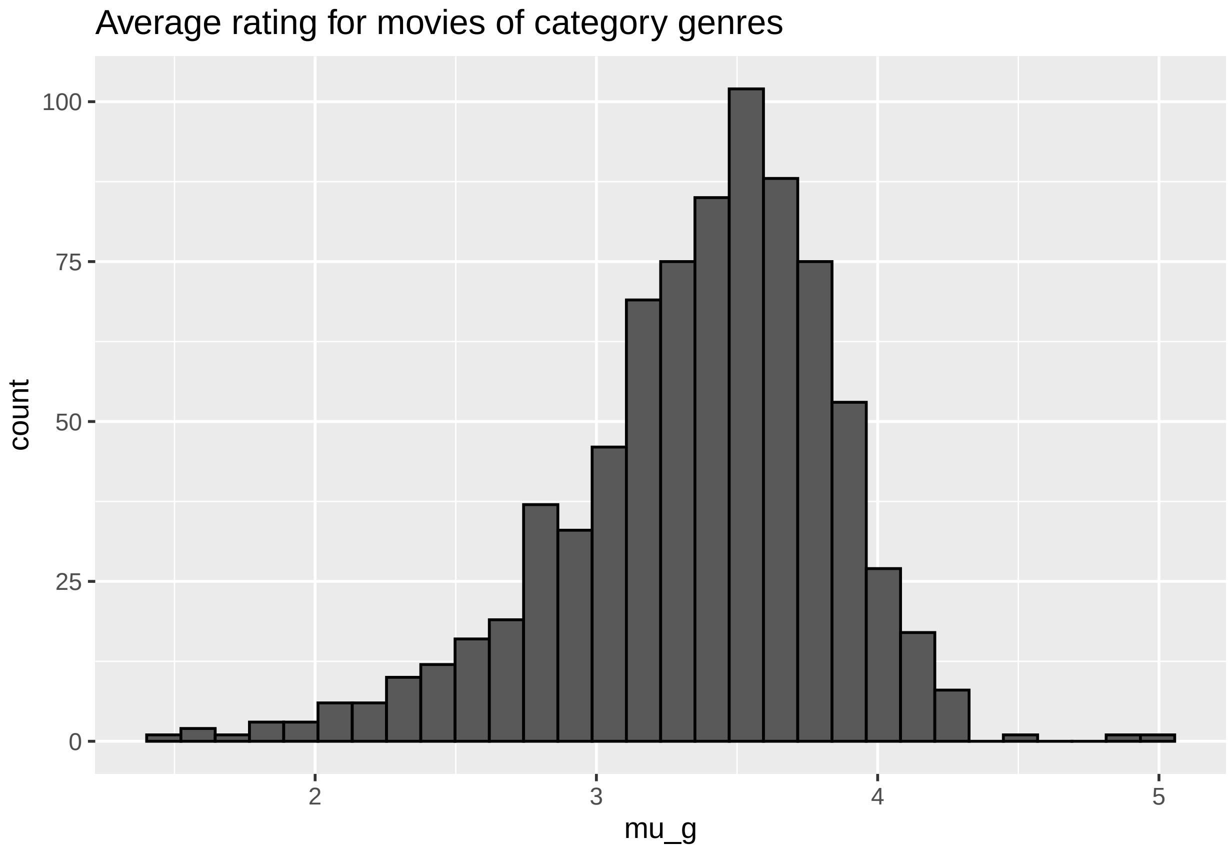


Figure 15: Average rating for movies of category genres

To fit this model, we could again use the lm() function as shown in Equation 11:

$$\begin{aligned} fit \leftarrow lm(rating\ \sim as.factor(movieId) + as.factor(userId) + \\ as.factor(genres),\ data = train\_set) \end{aligned} \tag{11}$$

but, for the reasons described earlier, we won't. Instead, we will compute an approximation by computing $\hat{\mu}$, $\hat{b_i}$, $\hat{b_u}$ and estimating $\hat{b_g}$ as the average of $y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_u}$:

$$\hat{b_g} = \overline{y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_u}} \tag{12}$$

where:

$$\hat{b_g} = \sum_{k=1}^{K} x_{u,i}\beta_k \tag{13}$$

$$\text{with } x_{u,i}^k = 1 \text{ if } g_{u,i} \text{ is genre *k*}$$

where $\hat{b_g}$ is genre specific effect.

```r
genres_avgs <- train_set %>% left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>% group_by(genres) %>%
    summarize(b_g = mean(rating - mu - b_i - b_u))
```

We can see that these estimates vary substantially, see Figure 16

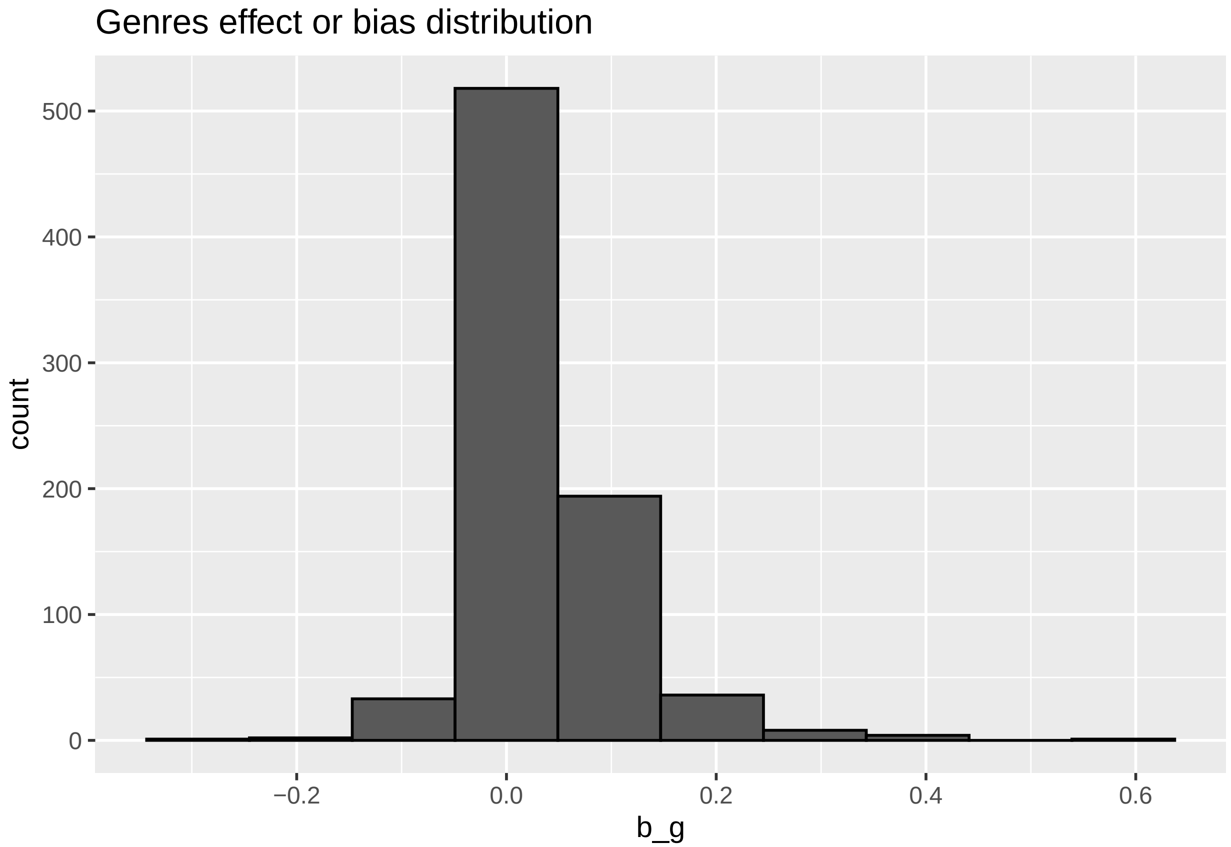


Figure 16: Genres effect or bias distribution

We can now construct predictors and see how much the RMSE improves:

```
predicted_ratings_model_4 <- test_set %>% left_join(movie_avgs,
    by = "movieId") %>% left_join(user_avgs, by = "userId") %>%
    left_join(genres_avgs, by = "genres") %>% mutate(pred = mu +
    b_i + b_u + b_g) %>% .$pred
(model_4_rmse <- RMSE(predicted_ratings_model_4, test_set$rating))
[1] 0.8655941
```

3.5.1 Results Table Model 1-4

Let's add the genres effects model to our results table to get Table 15

Table 15: RMSE Results Model 1-4

Index	Method	RMSE
1	Just the average	1.0599043
2	Movie Effect Model	0.9437429
3	Movie + User Effects Model	0.8659320
4	Movie + User + Genres Effects Model	0.8655941

3.6 Model 5 : Rating Time effect

The movielens dataset also includes a time stamp. This variable represents the time and date in which the rating was provided. Earlier in the EDA/Data wrangling section we created a new column date with the time stamp.

We computed the average rating for each week and plotted this average against day.

The plot shows some evidence of a time effect. If we define $d_{u,i}$ as the day for user's u rating of movie i, then the following updated model as shown in Equation 14 is most appropriate:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}\beta_k + f(d_{u,i}) + \epsilon_{u,i} \tag{14}$$

with f a smooth function of $d_{u,i}$

To fit this model, we could again use lm() function as shown in Equation 15:

$$\begin{aligned} fit \leftarrow lm(rating\ \sim\ & as.factor(movieId) + as.factor(userId) + \\ & as.factor(genres) + as.factor(date),\ data = train\_set) \end{aligned} \tag{15}$$

but, for the reasons described earlier, we won't. Instead, we will compute an approximation by computing $\hat{\mu}$, $\hat{b_i}$, $\hat{b_u}$, $\hat{b_g}$ and estimating $\hat{b_d}$ as the average of $y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_u} - \hat{b_g}$:

$$\hat{b_d} = \overline{y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_u} - \hat{b_u}} \tag{16}$$

where:

$$\hat{b_d} = f(d_{u,i}) \tag{17}$$

where $\hat{b_d}$ is rating time specific effect.

```r
time_effect_avgs <- train_set %>% left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>% left_join(genres_avgs,
    by = "genres") %>% group_by(date) %>% summarize(b_d = mean(avg_rating -
    mu - b_i - b_u - b_g))
```

We can see that these estimates vary substantially, see Figure 16

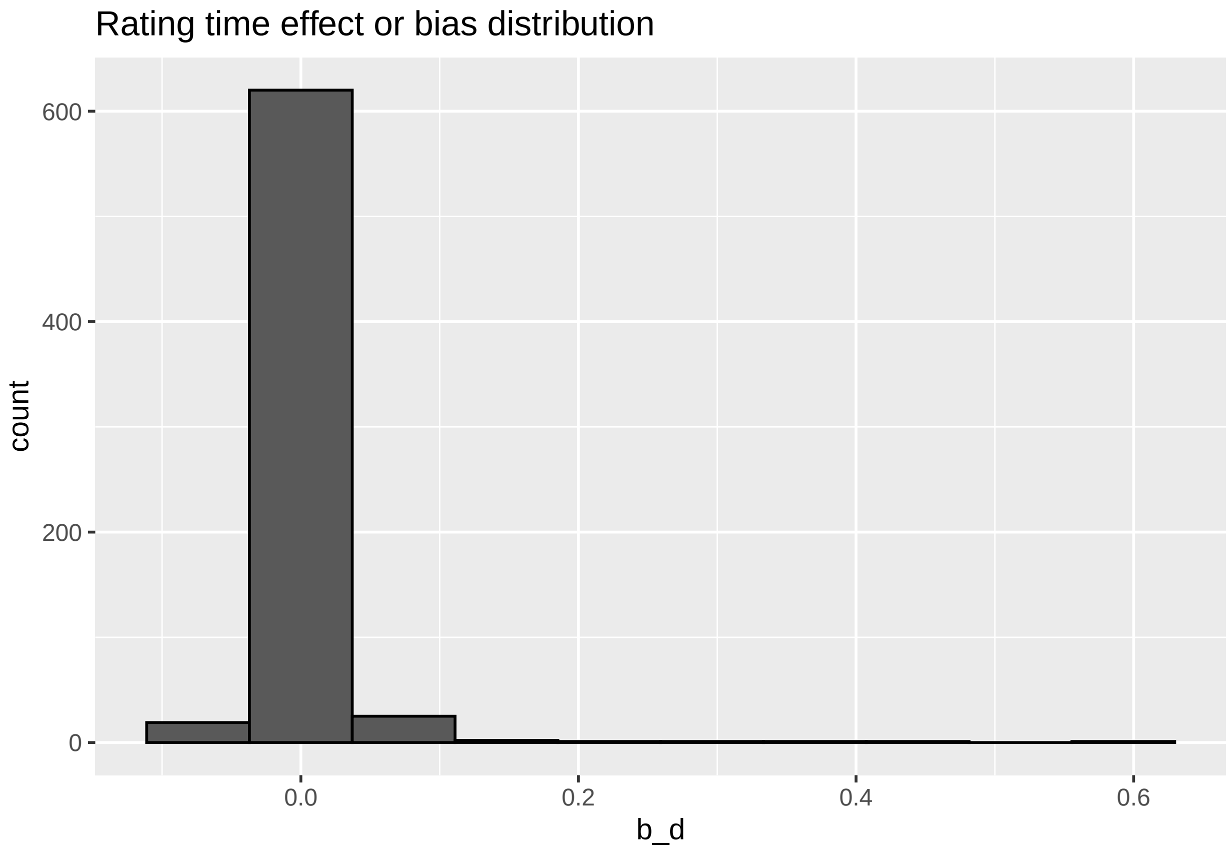


Figure 17: Rating time effect or bias distribution

We can now construct predictors and see how much the RMSE improves

```
predicted_ratings_model_5 <- test_set %>% left_join(movie_avgs,
    by = "movieId") %>% left_join(user_avgs, by = "userId") %>%
    left_join(genres_avgs, by = "genres") %>% left_join(time_effect_avgs,
    by = "date") %>% mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
    .$pred
(model_5_rmse <- RMSE(predicted_ratings_model_5, test_set$rating))
[1] 0.8654205
```

3.6.1 Results Table Model 1-5

Let's add the Rating Time effects model to our results table to get Table 16

Table 16: RMSE Results Model 1-5

Index	Method	RMSE
1	Just the average	1.0599043
2	Movie Effect Model	0.9437429
3	Movie + User Effects Model	0.8659320
4	Movie + User + Genres Effects Model	0.8655941
5	Movie + User + Genres + Rating Time Effects Model	0.8654205

3.7 Model 6 Release Date Effect

The plots in Figures 6, 7, 8, 9 above shows some evidence of a Release Date effect based on the when the movie was released and it's popularity given by the mean rating. If we define $arr_{r,i,y}$ as the average rating *r=mean(rating)* since release date *y=n_year* for movie i (in the formula for plots above), then the following updated model is most appropriate:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}\beta_k + f(d_{u,i}) + f(arr_{r,i,y}) + \epsilon_{u,i} \tag{18}$$

with f a smooth function of $arr_{r,i,y}$

To fit this model, we could again use lm() function as shown in Equation 19:

$$\begin{aligned} fit \leftarrow lm(rating\ \sim & as.factor(movieId) + as.factor(userId) + \\ & as.factor(genres) + as.factor(date) + \\ & as.factor(movie_dt),\ data = train\_set) \end{aligned} \tag{19}$$

but, for the reasons described earlier, we won't. Instead, we will compute an approximation by computing $\hat{\mu}$, $\hat{b_i}$, $\hat{b_u}$, $\hat{b_g}$, $\hat{b_d}$ and estimating $\hat{b_r}$ as the average of $y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_u} - \hat{b_g} - \hat{b_d}$ where:

$$\hat{b_r} = \overline{y_{u,i} - \hat{\mu} - \hat{b_i} - \hat{b_u} - \hat{b_g} - \hat{b_d}} \tag{20}$$

where:

$$\hat{b_r} = f(arr_{r,i,y}) \tag{21}$$

where $\hat{b_r}$ is Release date specific effect.

```
rel_effect_avgs <- train_set %>% left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>% left_join(genres_avgs,
    by = "genres") %>% left_join(time_effect_avgs, by = "date") %>%
    group_by(movieId) %>% summarize(b_r = mean(avg_rating_rel -
    mu - b_i - b_u - b_g - b_d))
```

We can see that these estimates vary substantially, see Figure 18

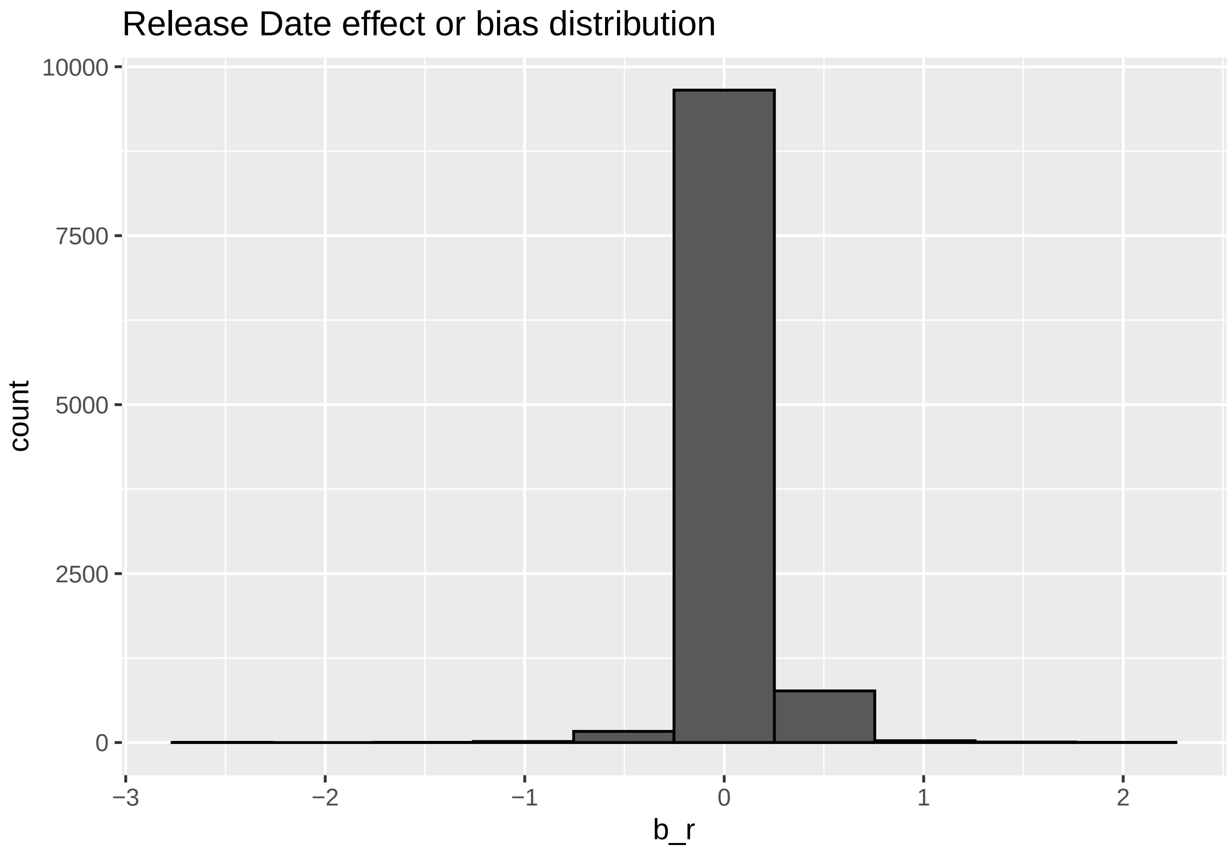


Figure 18: Release Date effect or bias distribution

We can now construct predictors and see how much the RMSE improves

```
predicted_ratings_model_6 <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by='genres') %>%
  left_join(time_effect_avgs, by = "date") %>%
  left_join(rel_effect_avgs, by='movieId') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_d + b_r) %>%
  .$pred

(model_6_rmse <- RMSE(predicted_ratings_model_6, test_set$rating))
```

```
[1] 0.863333
```

3.7.1 Results Table

Let's add the Release Date effects model to our results table

Table 17: RMSE Results Model 1-6

Index	Method	RMSE
1	Just the average	1.0599043
2	Movie Effect Model	0.9437429
3	Movie + User Effects Model	0.8659320
4	Movie + User + Genres Effects Model	0.8655941
5	Movie + User + Genres + Rating Time Effects Model	0.8654205
6	Movie + User + Genres + Time + Releasedate Effects Model	0.8633330

```
knitr::knit_exit()
```