HarvardX PH125.9x MovieLens Ratings Prediction Project

Rogelio Montemayor

June 3, 2021

1 Introduction and overview

This project is part of the "PH125.9x Data Science: Capstone" course on HarvardX. The goal of this project is to create a recommendation system using the MovieLens dataset. For this project, to make computation simpler, we will be using the 10M version of the MovieLens dataset. This dataset contains 10 million ratings of more than 10,000 movies given by about 70,000 users.

We will build an algorithm that will predict the rating a particular user would give movie. To make the prediction, our model uses the movie, the user, and the year the movie was released.

To score how accurate the model predicts a given rating, we use root mean square error (RMSE) as a metric. Our final model has a **RMSE (*Root Mean Square Error*) of 0.8645.**

1.1 Overview

These are the steps we will follow to go from raw dataset to model and insights:

- Download and build the dataset
- Analysis
  - Initial exploratory analysis
  - Data cleaning and feature engineering
  - Further visual exploration
  - Modeling approach
    - Baseline model
    - Iterations of ever more complex models
- Results and final model
- Conclusion and insights

1.2 Download the raw dataset

We will download the data from http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

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

1.3 Build the base dataset, split into base and validation set

First we build the base dataset:

```
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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

Then we split the base dataset into the main (edx) and validation sets. The validation set is 10% of the base dataset. The validation set will **only** be used to test our final model. We will use the edx set for training and model selection.

```
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,]
```

We need to make sure that *userId* and *movieId* in the validation set are also in de edx set:

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

Then we add the rows removed back into the edx set:

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

2 Analysis

Our first step is to explore the edx dataset to get a sense of the way the information is presented, better understand the features and to get ideas for data cleaning or feature engineering.

2.1 Initial exploratory analysis

Let's take a look at a few rows of the file. We can see that the file uses the following six variables:

- userId
- movieId
- rating
- timestamp
- title
- genres

```
head(edx)
```

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

```
glimpse(edx)
```

```
Rows: 9,000,055
Columns: 6
$ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
$ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
$ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
$ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
$ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
$ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

We can see that the title includes the year the movie came out, and we can also see that the

genres combines genre names for each movie. Also, the *timestamp* column is in the format where the value is the number of seconds since January 1st, 1970.

We can now look at some summary statistics for the dataset:

```
     userId         movieId          rating        timestamp
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
 1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
 Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
 Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
 3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
    title              genres
 Length:9000055     Length:9000055
 Class :character   Class :character
 Mode  :character   Mode  :character
```

It is also valuable to calculate how many unique users and movies are in the set:

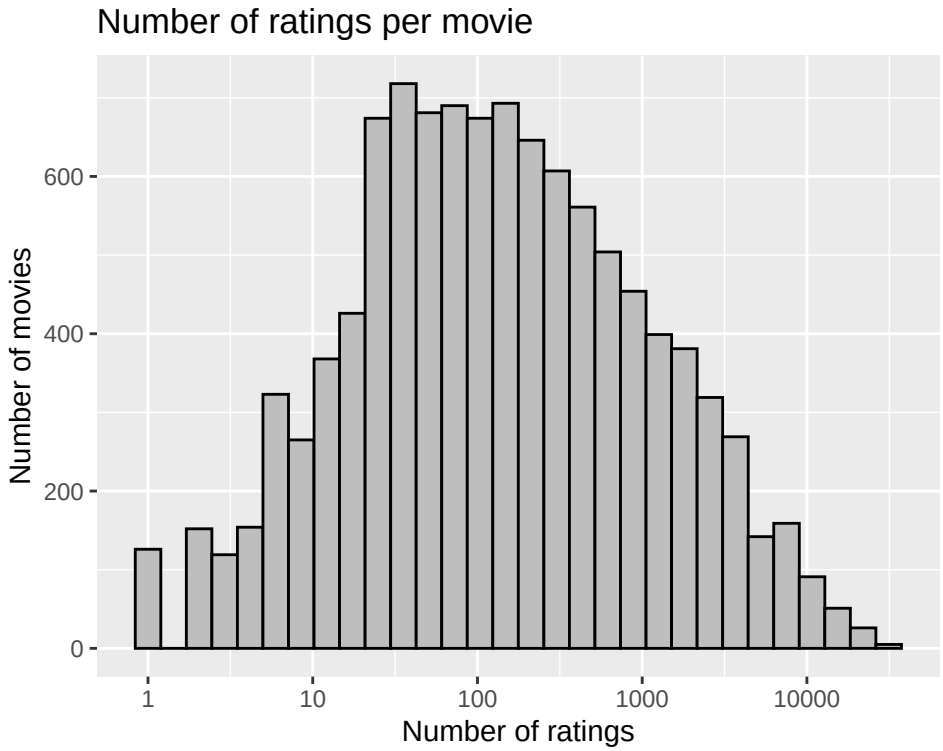
```
  unique_users unique_movies
1        69878         10677
```

Next, we have a table of the top 10 movies by number of ratings:

title	n_reviews
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015
Braveheart (1995)	26212
Fugitive, The (1993)	25998
Terminator 2: Judgment Day (1991)	25984
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
Apollo 13 (1995)	24284

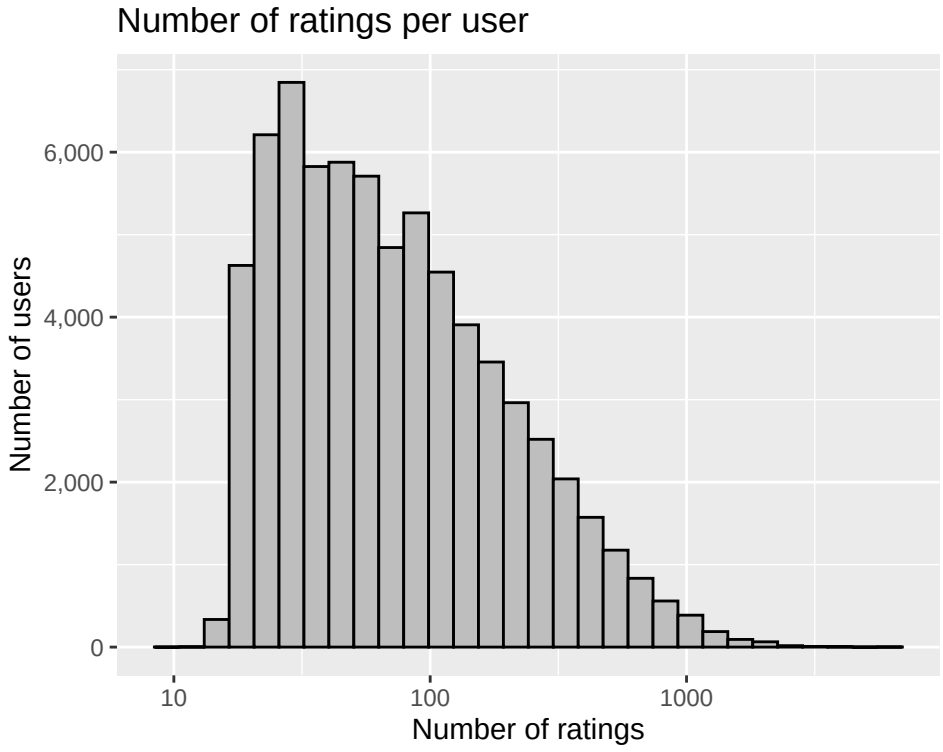
As it stands, Pulp Fiction is the most reviewed movie in our set.

The next step is to perform some visual analysis of the main variables, to get a feeling of the distribution of the data.



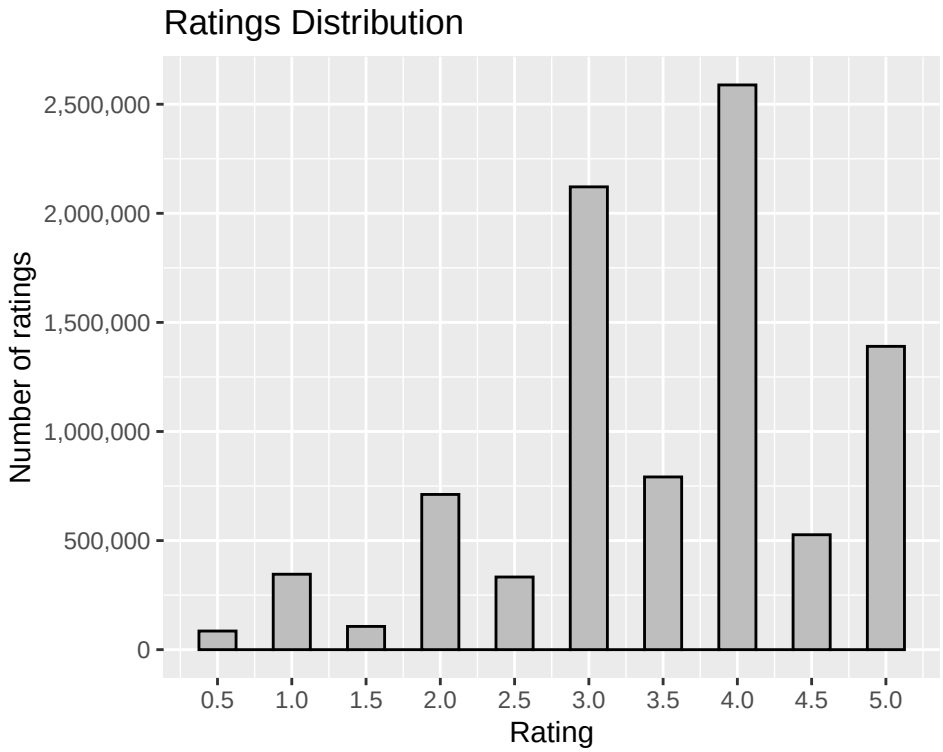
We can see the number of ratings per movie, and it shows that while some blockbusters (on the right) get over 10,000 reviews, there are a good number of more obscure movies with very few ratings (on the left).

Another useful graph is ratings per user:



Some users rated over a thousand movies, but most rate between 30 and 100 movies.

It is also interesting to look at the ratings distribution. Where we can see that the most common rating is 4.



In fact, the average rating for the entire dataset is 3.5124652 stars.

2.2 Data cleaning and feature engineering

The first data cleaning step is to add a new column with the release year (year) and to remove that text from the title value.

```
edx <- edx %>% mutate(year = as.numeric(str_sub(title, -5,-2)))
edx <- edx %>% mutate(title = str_sub(title, 1, -8))

validation <- validation %>% mutate(year = as.numeric(str_sub(title, -5,-2)))
validation <- validation %>% mutate(title = str_sub(title, 1, -8))
```

We also create a new column called movie_age, with the age of the movie calculated as the number of years from 2020 to the release date.

```
edx <- edx %>% mutate(movie_age= 2020 - year)
validation <- validation %>% mutate(movie_age= 2020 - year)
```

I also want to create a new version of the dataset with the genres disaggregated.

```
# This step takes a long time. For now I will only split the edx dataset
edx_split_genres <- edx %>% separate_rows(genres, sep = "\\|")
```

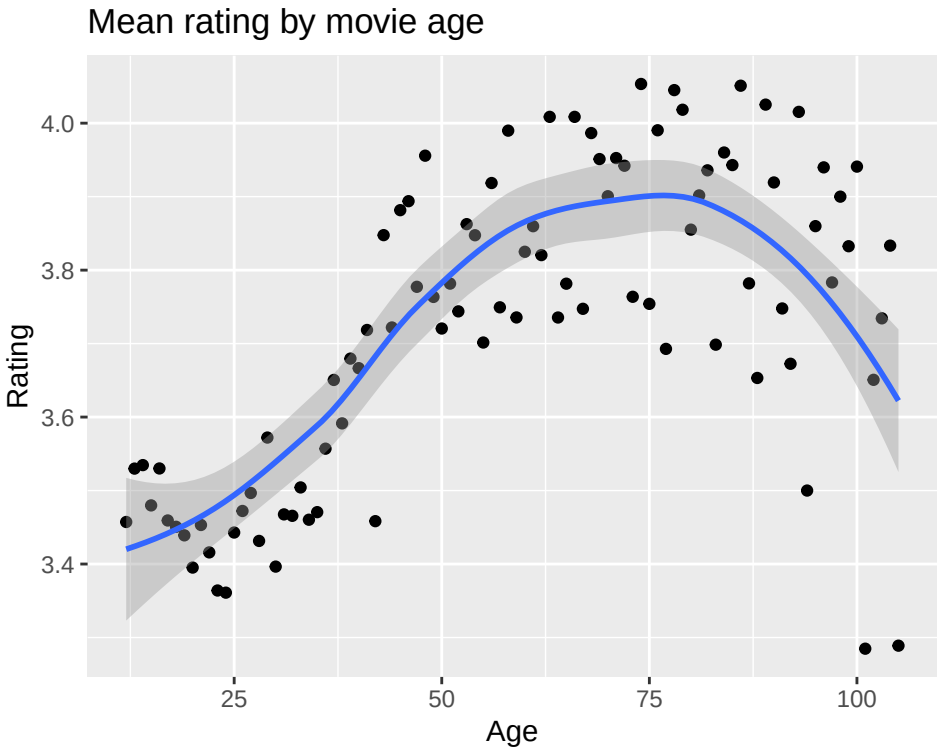
And also create a new column with the year where the movie was rated. We get that information from the timestamp column.

```
library(lubridate)
# The timestamp is the number of seconds elapsed since January 1st, 1970
edx <- edx %>%
  mutate(year_rated= year(as.Date(as.POSIXct(timestamp,
                                             origin= "1970-01-01"))))
validation <- validation %>%
  mutate(year_rated= year(as.Date(as.POSIXct(timestamp,
                                             origin= "1970-01-01"))))
```

2.3 Further visual exploration

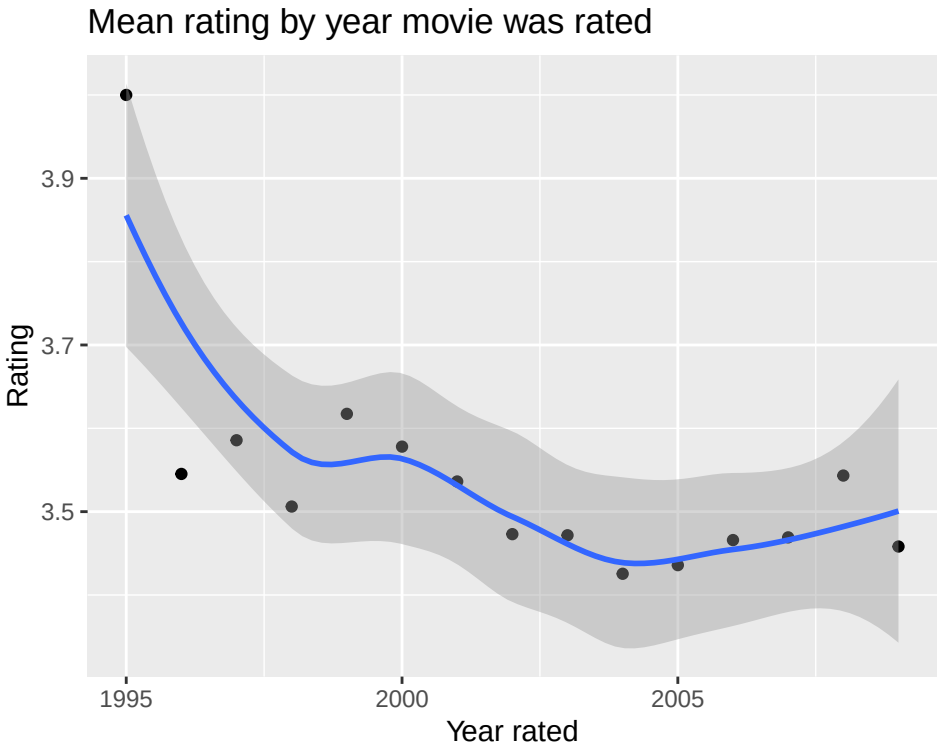
After cleaning and creating new features (columns), we can perform some additional visual exploration of our data.

First we can see the average rating of movies of the same age (released in the same year):



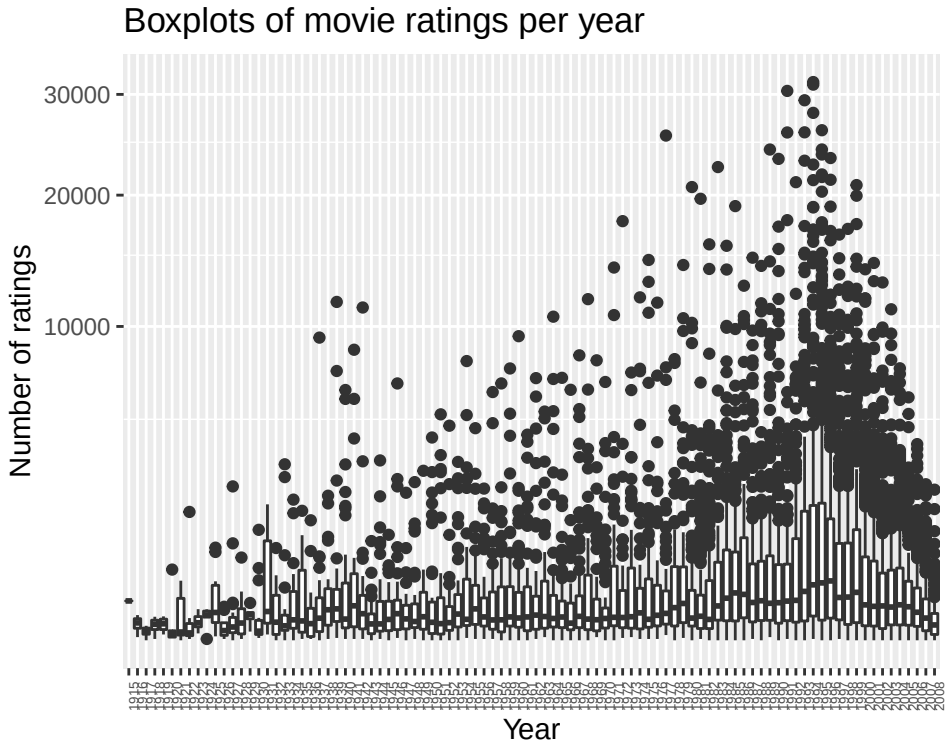
This shows an interesting pattern, where it seems average ratings do vary across decades.

We can see the average rating of movies based on the year they were rated. This shows a smaller effect.



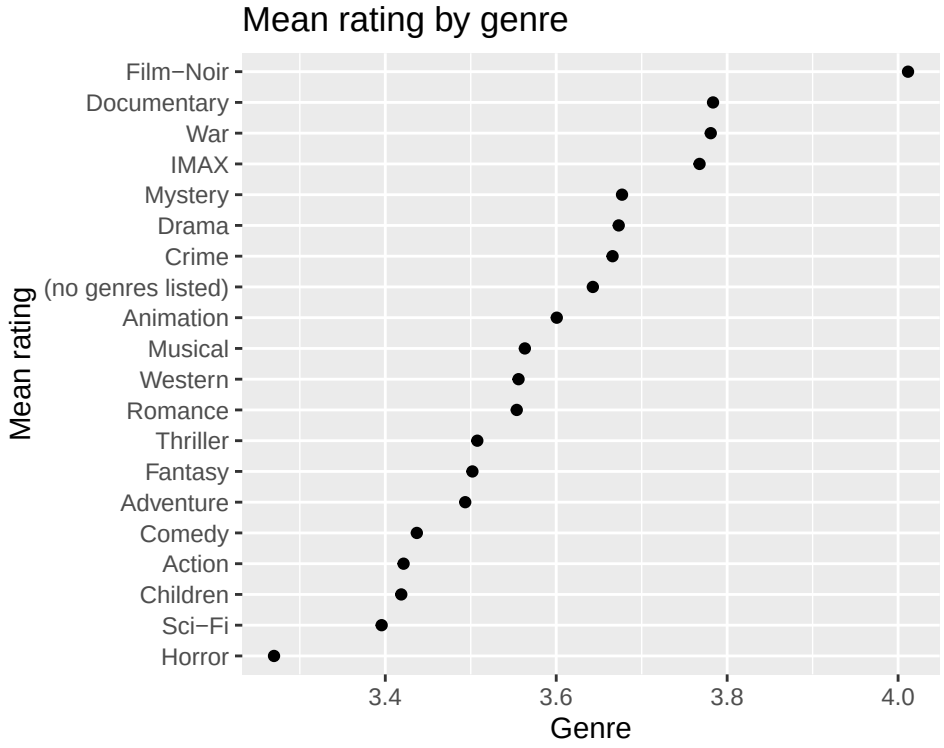
We can also see that the first ratings were made in 1995.

Another analysis that is interesting is to look at the distribution of ratings by year. Here are the boxplots for every year in the dataset. We can see that the year with the highest median rating is 1995.



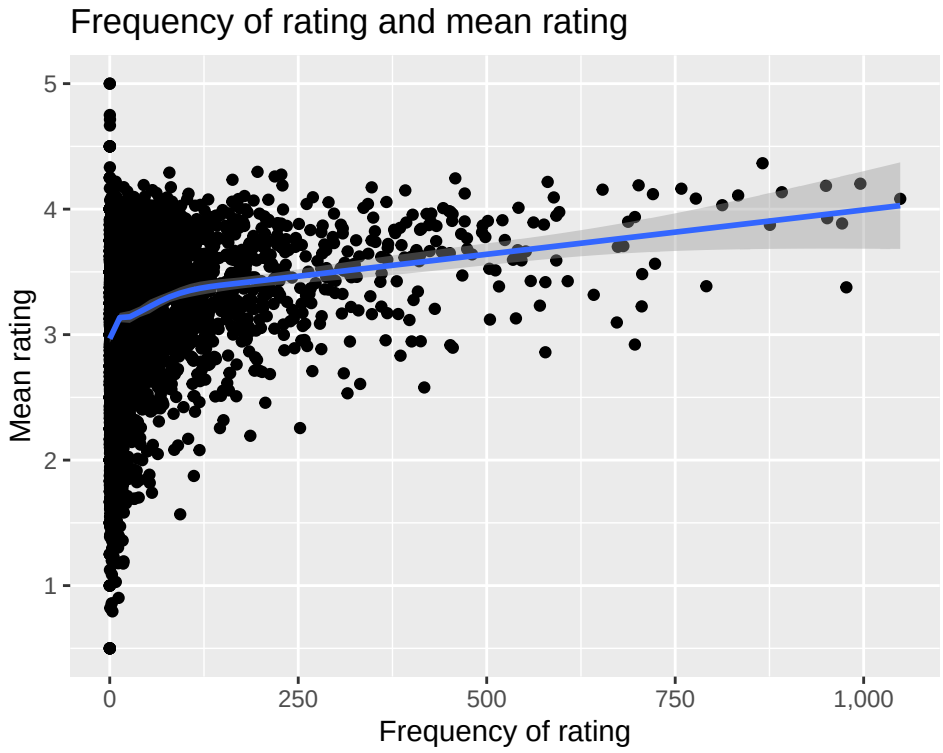
We can also see that from that year, with newer movies the number of ratings decreases every year: the more recent a movie is, the less time users have had to rate it.

Are ratings influenced by the movie genre? In the next plot we can see that the average rating is different for movies of different ratings.



Horror movies have the lowest average rating, and one could imagine that this is due to the many low-budget horror films out there.

Finally, using the age of the movie, we can see how the most frequently rated movies (more popular movies) have a higher average rating.



We filtered for movies released since 1995 since we previously learned that the number of ratings decreased from that point for recent movies.

2.4 Modeling approach

With our exploration, we can start to see the effect that movie, user and year have on the ratings. Our model should try to use those features to inform its prediction.

We need to split the edx dataset into training and test sets.

```
set.seed(157, sample.kind= "Rounding")
test_index <- createDataPartition(y= edx$rating, times= 1,
                                  p= 0.2, list= FALSE)
edx_train <- edx[-test_index,]
edx_test <- edx[test_index,]

# Ensure users and movies on the test set are also on the train set
edx_test <- edx_test %>%
  semi_join(edx_train, by= "movieId") %>%
  semi_join(edx_train, by= "userId")
```

In our training and model selection, we need to constantly evaluate the performance of each of our intermediate models. As we have mentioned, the loss function we are going to minimize is the root mean squared error (RMSE). RMSE is the standard deviation of the prediction

errors (residuals):

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

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

2.4.1 Baseline model

We will start with a very simple baseline model where we predict every new rating to be just the average of all the ratings. We will name this mean rating mu.

```
mu <- mean(edx_train$rating)
mu
```

```
[1] 3.512504
```

We now can calculate the RMSE for this prediction versus all the actual predictions in our test set

```
m_1_rmse <- RMSE(edx_test$rating, mu)
```

With this very simplistic approach, our ratings are off an average of 1.0607351 stars.

To keep tabs on our different models, we are going to build a table with the model name and the RMSE for each model

Model	RMSE
Using just the average	1.060735

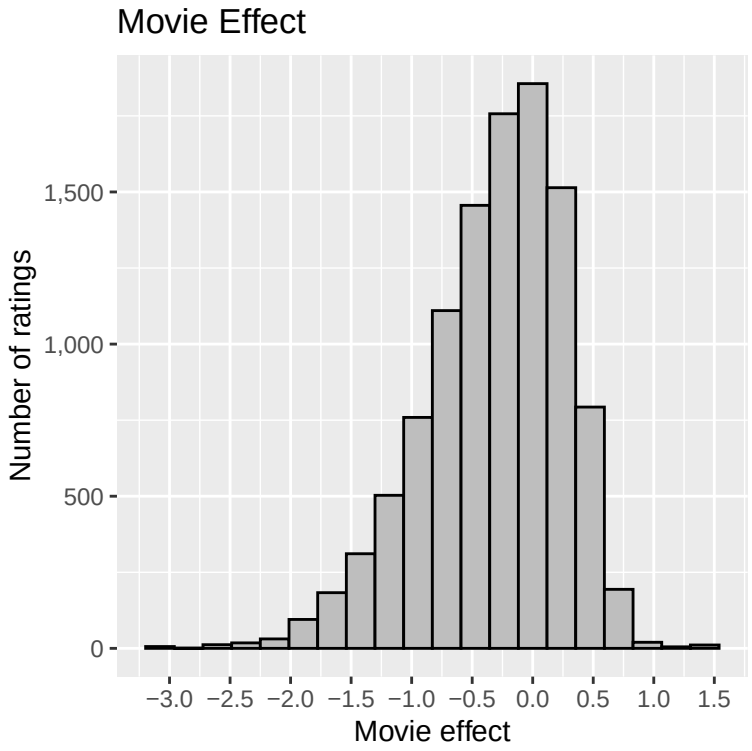
2.4.2 Model 2: Movie effect model

It is now time to start making our model better by using all the data we have. We saw on our exploration that some movies are more generally liked than others, and thus different movies have different average ratings.

It is possible to improve our model by calculating for each movie its estimated deviation from the average rating for all movies. Let's call this the movie effect (m_e).

```
movie_effect <- edx_train %>%
  group_by(movieId) %>%
  summarize(m_e= mean(rating - mu))
```

We can visualize this effect:



We can also see that the deviations skew to the negative side.

Once we have calculated this deviation for each movie, we can now predict ratings and calculate the RMSE for this model. We will call it "Movie effect model".

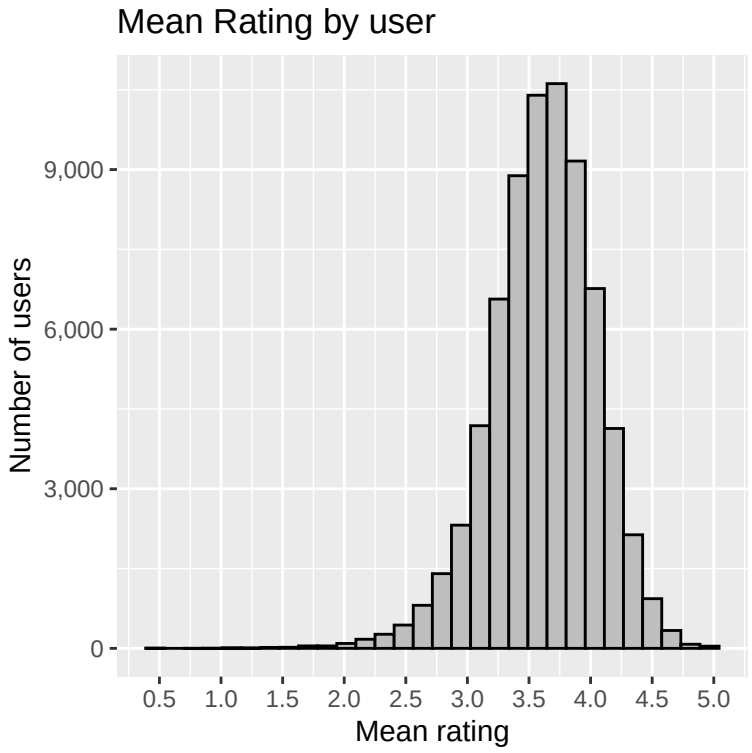
```
predicted_ratings <- edx_test %>%
  left_join(movie_effect, by= "movieId") %>%
  mutate(pred= mu + m_e) %>%
  .$pred
m_2_rmse <- RMSE(edx_test$rating, predicted_ratings)
```

Our RMSE has now dropped to 0.9442851. We update our results table.

Model	RMSE
Using just the average	1.0607351
Movie effect model	0.9442851

2.4.3 Model 3: Movie + user effect model

To further improve our model, we can take into account that different users rate movies with a different scale. Some might give 5 stars to every movie, and some give most movies they rate 1 or 2 stars:

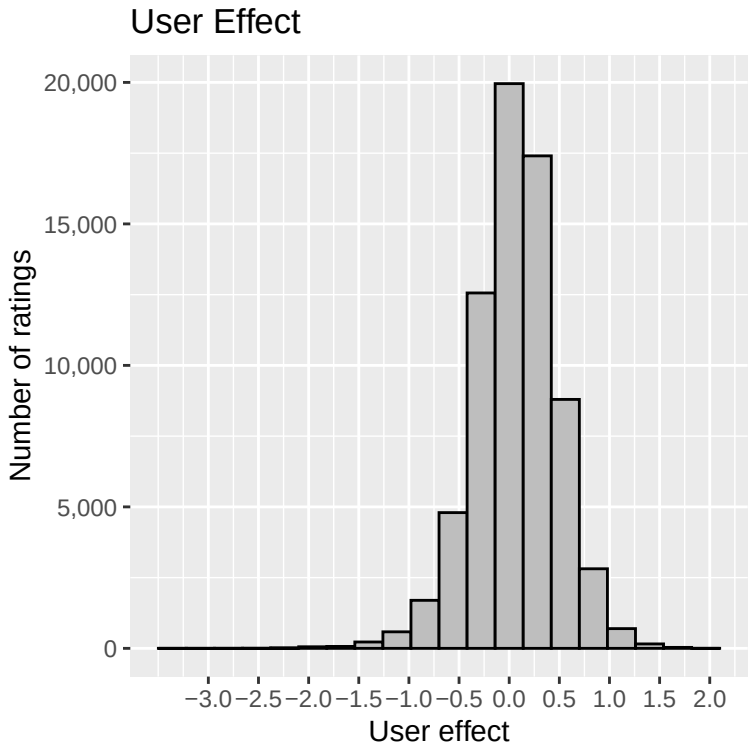


We can see how some users on the right have an average rating that is a lot higher than those users on the left.

Lets include this effect in our model. We start from our movie effect model and now add the effect of the average rating for each user to the model.

```
user_effect <- edx_train %>%
  left_join(movie_effect, by= "movieId") %>%
  group_by(userId) %>%
  summarize(u_e= mean(rating - mu - m_e))
```

The plot shows that the distribution is tighter for this effect:



Once we have calculated the effect of the particular user on our model, we can calculate the predicted ratings and the RMSE for this "Movie + user effect model".

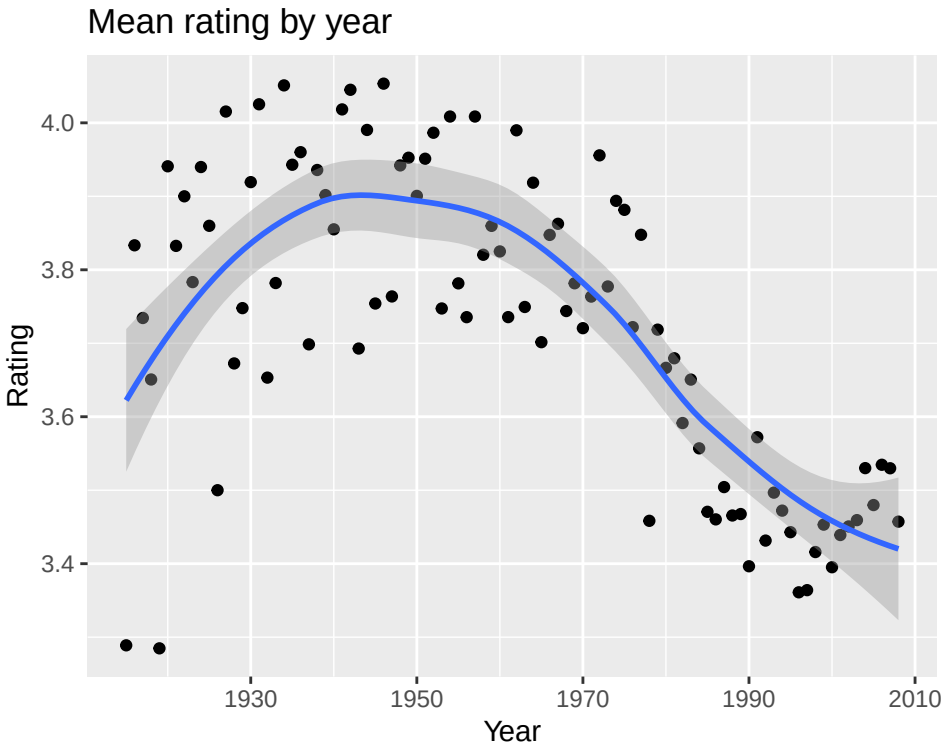
```
predicted_ratings <- edx_test %>%
  left_join(movie_effect, by= "movieId") %>%
  left_join(user_effect, by= "userId") %>%
  mutate(pred= mu + m_e + u_e) %>%
  .$pred
m_3_rmse <- RMSE(edx_test$rating, predicted_ratings)
```

The RMSE is now 0.866729:

Model	RMSE
Using just the average	1.0607351
Movie effect model	0.9442851
Movie + user effect model	0.8667290

2.4.4 Model 4: Movie + user + year effect model

Let us now turn our efforts to the effect a movie's release year has on ratings. We saw on the visual exploration stage that average ratings seemed to vary through the years. Lets model the mean rating for a movie by their release year. We will use release year instead of age for simplicity:

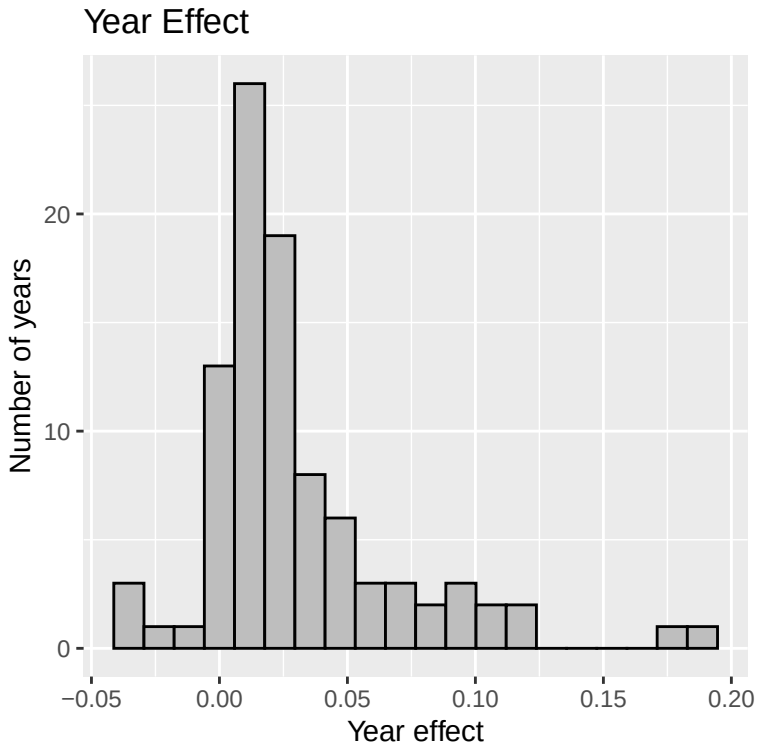


As we can see that average ratings rise as the movie is older, and declines again for movies released before 1950.

In the same way we added the deviation for movie and user, we can add the effect of release year on a movie average rating.

```
year_effect <- edx_train %>%
  left_join(movie_effect, by= "movieId") %>%
  left_join(user_effect, by= "userId") %>%
  group_by(year) %>%
  summarize(y_e= mean(rating - mu - m_e - u_e))
```

And we can visualize this deviataion using a histogram:



Clearly it is a very small effect compared to the previous two.

Let's now predict ratings using all three effects. We will call this "Movie + user + year effect model".

```
predicted_ratings <- edx_test %>%
  left_join(movie_effect, by= "movieId") %>%
  left_join(user_effect, by= "userId") %>%
  left_join(year_effect, by= "year") %>%
  mutate(pred= mu + m_e + u_e + y_e) %>%
  .$pred

# Calculate the RMSE for this model
m_4_rmse <- RMSE(edx_test$rating, predicted_ratings)
```

Our RMSE is now 0.8663849. As we can see the improvement is very small:

Model	RMSE
Using just the average	1.0607351
Movie effect model	0.9442851
Movie + user effect model	0.8667290
Movie + user + year effect model	0.8663849

2.4.5 Model 5: Regularized movie + user effect model

Up to now, we are giving the same weight in our models to every movie whether it has a thousand ratings or just a handful. So, obscure movies with few ratings can distort our model and add to the measured error.

Here are the 10 movies with the biggest deviation from our model:

title	m_e	n
Hellhounds on My Trail	1.487496	1
Marathon Family, The (Maratonci Trce Pocasni Krug)	1.487496	2
Titicut Follies	1.487496	1
Satan's Tango (Sátántangó)	1.487496	2
Shadows of Forgotten Ancestors	1.487496	1
Money (Argent, L')	1.487496	1
Sun Alley (Sonnenallee)	1.487496	1
Aerial, The (La Antena)	1.487496	1
Blue Light, The (Das Blaue Licht)	1.487496	1
More	1.404163	6

We can see they are all rather obscure movies with very few ratings. This is the table for 10 movies with the biggest negative deviation:

title	m_e	n
Besotted	-3.012504	1
Grief	-3.012504	1
Under the Lighthouse Dancing	-3.012504	1
Accused (Anklaget)	-3.012504	1
Confessions of a Superhero	-3.012504	1
War of the Worlds 2: The Next Wave	-3.012504	2
Hip Hop Witch, Da	-2.876140	11
SuperBabies: Baby Geniuses 2	-2.719026	46
Disaster Movie	-2.615952	29
From Justin to Kelly	-2.611230	157

These large deviations by movies with very few ratings are adding to our overall RMSE.

One technique to fix this problem is to use **regularization**, where we add a penalty that will have a large effect for movies with a very few ratings and a very small effect for movies with many reviews.

In the equation below, if the number of ratings for movie i is large, the effect of lambda is

very small.

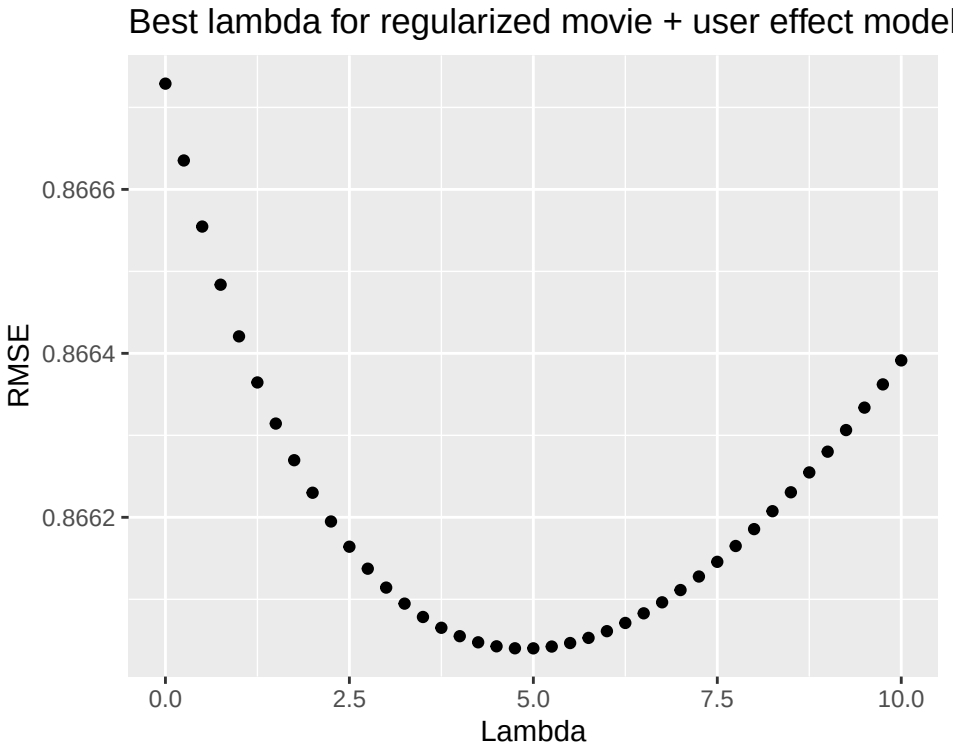
$$m_e(regularized) = \frac{1}{(lambda + n_i)} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{mu})$$

The higher the value of lambda, the higher the regularization strength. To find out what is the value of best value of lambda to use, we iterate over a range of different values of lambda until we find the one that minimizes the RMSE. For our first try, we will only use our biggest effects: movie and user.

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

rmses <- sapply(lambdas, function(l) {
  mu <- mean(edx_train$rating)
  m_e <- edx_train %>%
    group_by(movieId) %>%
    summarize(m_e= sum(rating - mu)/(n()+l))
  u_e <- edx_train %>%
    left_join(m_e, by= "movieId") %>%
    group_by(userId) %>%
    summarize(u_e= sum(rating - m_e - mu)/(n()+l))
  predicted_ratings <- edx_test %>%
    left_join(m_e, by= "movieId") %>%
    left_join(u_e, by= "userId") %>%
    mutate(pred= mu + m_e + u_e) %>%
    .$pred
  return(RMSE(edx_test$rating, predicted_ratings))
})
```

We can visualize how the RMSE changes as we use different values for lambda:



The best lambda in this version is 5. With this lambda we get an RMSE of: 0.8660402:

Model	RMSE
Using just the average	1.0607351
Movie effect model	0.9442851
Movie + user effect model	0.8667290
Movie + user + year effect model	0.8663849
Regularized movie + user effect model	0.8660402

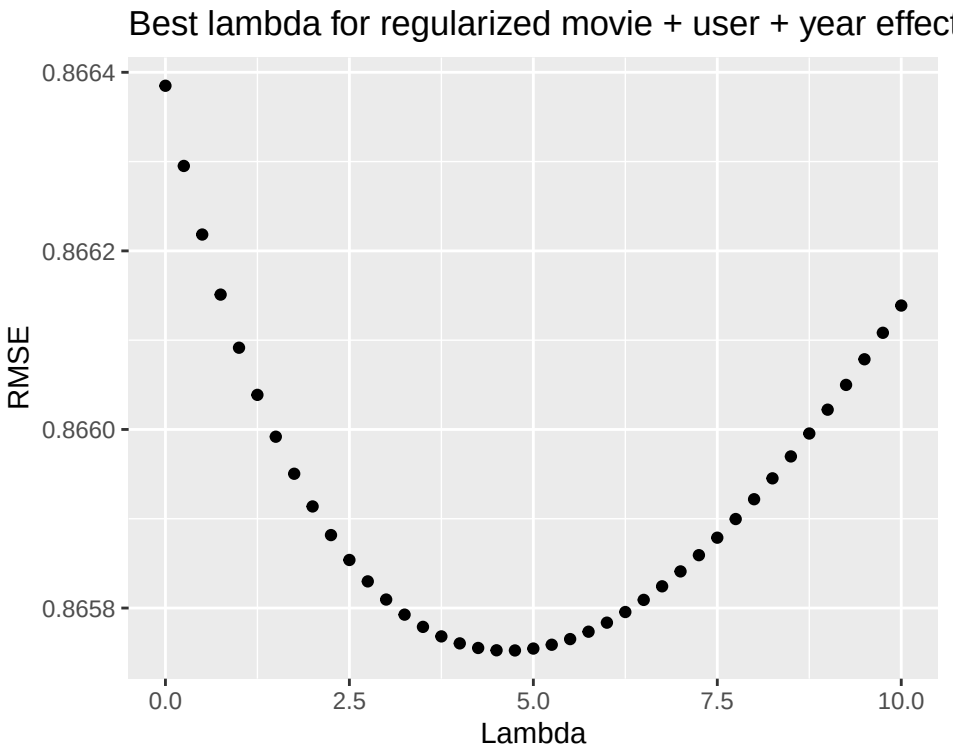
2.4.6 Model 6: Regularized movie + user + year effect model

We will improve our model one last time by using regularization for all three effects: movie, user, and year. We can run our iterations to find the value of lambda that minimizes the RMSE.

```
rmses_2 <- sapply(lambdas, function(l) {
  mu <- mean(edx_train$rating)
  m_e <- edx_train %>%
    group_by(movieId) %>%
    summarize(m_e= sum(rating - mu)/(n()+l))
  u_e <- edx_train %>%
    left_join(m_e, by= "movieId") %>%
    group_by(userId) %>%
    summarize(u_e= sum(rating - m_e - mu)/(n()+l))
  y_e <- edx_train %>%
```

```
    left_join(m_e, by= "movieId") %>%
    left_join(u_e, by= "userId") %>%
    group_by(year) %>%
    summarize(y_e= sum(rating - m_e - u_e - mu)/(n()+l))
  predicted_ratings <- edx_test %>%
    left_join(m_e, by= "movieId") %>%
    left_join(u_e, by= "userId") %>%
    left_join(y_e, by= "year") %>%
    mutate(pred= mu + m_e + u_e + y_e) %>%
    .$pred
  return(RMSE(edx_test$rating, predicted_ratings))
})
```

We can use the plot to find the new best value for lambda:



The best lambda is now 4.75. With this value our RMSE is: 0.8657525:

Model	RMSE
Using just the average	1.0607351
Movie effect model	0.9442851
Movie + user effect model	0.8667290
Movie + user + year effect model	0.8663849
Regularized movie + user effect model	0.8660402
Regularized movie + user + year effect model	0.8657525

3 Results

Up to this point, we have exclusively used the edx set to find our best performing algorithm. It is now time to test the final version of our model on the validation set to get the true measure of error in a set of data that we have not seen before. Our best model is the regularized movie + user + year effect model.

3.1 Final model

Now that we have the optimal value of lambda for our model, we will use it for regularization. We also need to recalculate the mean (mu) using the whole edx data (not just the train split).

```
mu <- mean(edx$rating)

movie_effect_final <- edx %>%
  group_by(movieId) %>%
  summarize(m_e= sum(rating - mu)/(n()+lambda_2))

user_effect_final <- edx %>%
  left_join(movie_effect_final, by= "movieId") %>%
  group_by(userId) %>%
  summarize(u_e= sum(rating - mu - m_e)/(n()+lambda_2))

year_effect_final <- edx %>%
  left_join(movie_effect_final, by= "movieId") %>%
  left_join(user_effect_final, by= "userId") %>%
  group_by(year) %>%
  summarize(y_e= sum(rating - mu - m_e - u_e)/(n()+lambda_2))
```

Having calculated all three effects, we can make predictions on the validation set, and get our final RMSE:

```
predicted_ratings <- validation %>%
  left_join(movie_effect_final, by= "movieId") %>%
  left_join(user_effect_final, by= "userId") %>%
  left_join(year_effect_final, by= "year") %>%
  mutate(pred= mu + m_e + u_e + y_e) %>%
  .$pred

# Calculate the RMSE for this model
m_7_rmse <- RMSE(validation$rating, predicted_ratings)
```

The **final RMSE** for our model is: **0.8645223**.

Our final table looks like this:

Model	RMSE
Using just the average	1.0607351
Movie effect model	0.9442851
Movie + user effect model	0.8667290
Movie + user + year effect model	0.8663849
Regularized movie + user effect model	0.8660402
Regularized movie + user + year effect model	0.8657525
Final RMSE (validation set, regularized movie + user + year effect model)	0.8645223

4 Conclusion

By building a base model and then creating increasingly more complicated models we were able to go from a RMSE of 1.0607351 to an **RMSE of 0.8645223**. This is an improvement of over **18.5%**.

To go further, we could try include more variables like genre, year rated, etc to see if that could improve our model.

For further exploration, we could try to include the important fact that group of users and groups of movies have similar rating patterns. We could perform a PCA analysis to group our users and movies and then use those groups to refine the predictions.