HarvardX; Data Science Capstone

MovieLens Recommndation System

Tali Behar

December 2020

1 Executive Summary

1.1 Overview

Recommendation systems are systems that are designed to recommend things to the user based on many different factors.

The goal of a recommender system is to generate meaningful recommendations by learning the user preferences and predicting the rating a user would give to an item. Then project that learning on future users of the system as well, in a way that keeps improving over time.

Suggestions for books on Amazon, or movies on Netflix, are real-world examples of the operation of industry-strength recommender systems.

Recommender systems are an important class of machine learning algorithms.

1.2 Background

MovieLens dataset was collected over a long period of time by GroupLens Research.

Here we are using a version that contains about 10 million rows.

Each row represents a movie rating given by a user.

We will use the MovieLens dataset in order to implement a movie recommendation system that will help us predict which movies a user is more likely to watch, based on his history ratings.

The dataset contains the following column for each row:

- UserId (contains ~70k different users)
- MovieId (contains ~11k different movies)
- Rating (varies between 0.5-5)
- Timestamp (when the rating was give)
- Title + Release year
- Genre (each movie can be categorized with multiple genres)

On average, each user rates 103 movies. This means that each user rated only a subset of the movies based on his preferences, and not all of them.

1.3 Goal

The goal of the project is to analyze movie i characteristics that can affect the rating of a given user u, and by that predict the rating that the user u will rate that movie.

1.4 Variable Selection

1. The outcome of the movie we are predicting; **Y = rating**
2. The characteristics we will use for the prediction:

- **Mean**: average of all rated movies in the system, regardless of the user
- **User**: users are different with their personal preferences, as well as in their engagement and excitement level. We'll check the user specific effect on the mean.
- **Movie**: some movies tend to get rated higher on average than others (regardless of the user).we'll check the movie specific effect on the mean.
- **Movie age**: older movies tend to be rated higher than new movies. We'll check how the age of the movie affect the user ratings.
- **Time effect**: newer movies have less ratings than older movies. We'll use that by adding the time effect to the average rating for movie i and the specific effect of user u.
- **Genre**: different genres get rated differently, both in the score and quantity of ratings. We'll add the genres effect and the time effect to the average rating for movie i and the specific effect of user u.

1.5 Method

- Divide the MovieLens dataset randomly to train set ("edx") and validation set (the final hold-out test set)
- Divide "edx" randomly to train and test set in order to train and examine the different models.
- Penalize outliers - cross validation in order to find penalty term- diving the data once again in order to prevent overfiting

1.6 Model Estimation

Evaluate model performance based on RMSE (Root Mean Square Error).

RMSE is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are. RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

The desired RMSE goal we get is **RMSE $<$ 0.8649**

1.7 Model Results

The chosen model was test on the validation set and resulted in **RMSE = 0.8632**

1.8 Conclusion

The most accurate model is the one that took into account all the features:

Reg. Movie + User Effect + Time Effect + Genres Effect

The test validation showed even better results than the results we obtained while training the model, which means that the algorithm can act standalone in the future.

2 Data Preperation

2.1 Installing Required Packeges

```
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(scales)
library(gridExtra)
library(knitr)
library(DescTools)
library(colorspace)
library(cowplot)
library(formattable)
```

2.2 Loading MovieLens Dataset

```
#Since we don't want to download and process the dataset every time we can run this report as it takes
# The file can be downloaded once from OneDrive and placed in the same folder as this Rmd file: https:/
# To use this optimization, simply uncomment this line, and comment out the following download code chu

#load("movielens_dataset.RData")
####
```

```
# Disable this code chunk if using the optimized process with an RData file

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

### 3.Create Edx set (training set) and Validation Sets (final hold-out test set)

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"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
```

```
# if using R 3.6 or earlier:
#movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
#                                           title = as.character(title),
#                                          genres = as.character(genres))

# if using R 4.0 or later:
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")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `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 <- 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)
edx <- rbind(edx, removed)

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

2.3 Get A Glimpse Of Edx Data

```
## 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, ...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ rating    <dbl> 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, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|...
```

'edx' data set contains 6 columns;

1. userId - Unique ID for the user
2. movieId - Unique ID for the movie
3. rating - A rating between 0.5 and 5 for the movie
4. timestamp - Date and time the rating was given
5. title - Movie title and year the movie was released.
6. genres - Genres associated with the movie

And 9 million rows. Each row provides a given rating to one movie by specific user.

2.4 Sanitizing the data - timestamp column

The date-time in the 'edx' timestamp column is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC.

In order to be able to use the time information and for better convenience, I converted the timestamp into rate date, added rate year column, extracted the release year from the movie title and deleted the release year from movie title

```
# Disable this code chunk if using the optimized process with an RData file

edx_year_sanitized <- edx %>%
  mutate(rate_year = year(as_datetime(timestamp)),
         rate_date = date(as_datetime(timestamp)),
         release_year =as.numeric(str_extract(title, "(?<=\\()(\\d{4})(?=\\))")),
         title= str_remove(as.character(title),"(\\(\\d{4}\\))")) %>%
  select(-timestamp)
```

Get a glimpse of the sanitized data

```
## Rows: 9,000,055
## Columns: 8
## $ userId       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId      <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377,...
## $ rating       <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ title        <chr> "Boomerang ", "Net, The ", "Outbreak ", "Stargate ", "...
## $ genres       <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Dra...
## $ rate_year    <dbl> 1996, 1996, 1996, 1996, 1996, 1996, 1996, 1996, 1996, ...
## $ rate_date    <date> 1996-08-02, 1996-08-02, 1996-08-02, 1996-08-02, 1996-...
## $ release_year <dbl> 1992, 1995, 1995, 1994, 1994, 1994, 1994, 1994, 1994, ...
```

2.5 The Dataset Overview

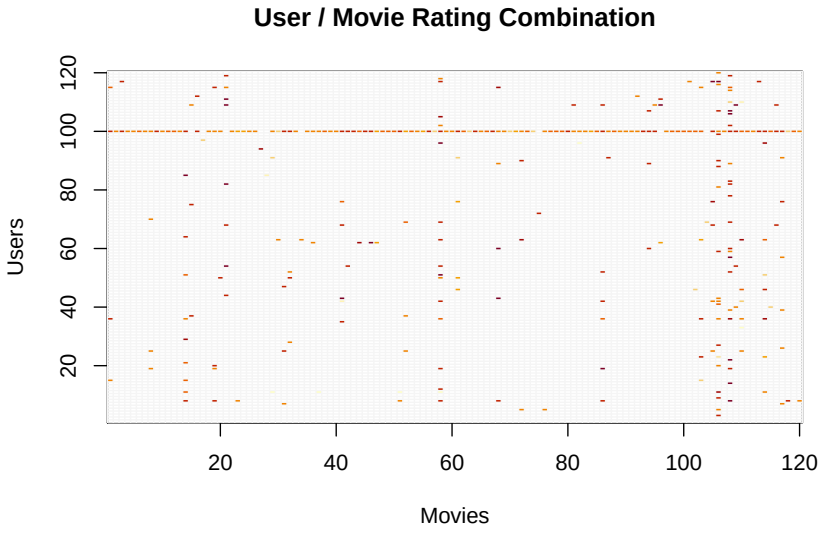
Number of users	Number of movies	Number of ratings (M)	Number of 'missing' ratings (M)
69878	10677	9.000055	737.0874

'edx_year_sanitized' contains about 70k different users that provided ratings and about 11k different rated movies.

The following matrix contain random sample of 120 movies and 120 users.

If we were to look at the entire matrix, the empty cells (about 737 million cells in the data set overall), represent the unrated movies by all users. Users rated only the selected movies by their personal preferences, therefore we have "only" 9 million ratings.

Each of the colored cells represent one rating and the color varies according to its score.



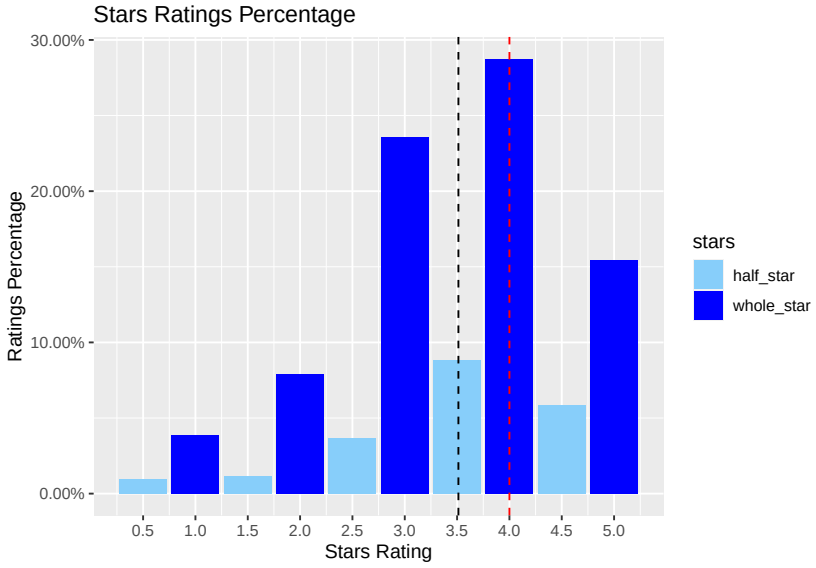
Evaluate the movie - Rating Score

mean	median	mode
3.512465	4	4

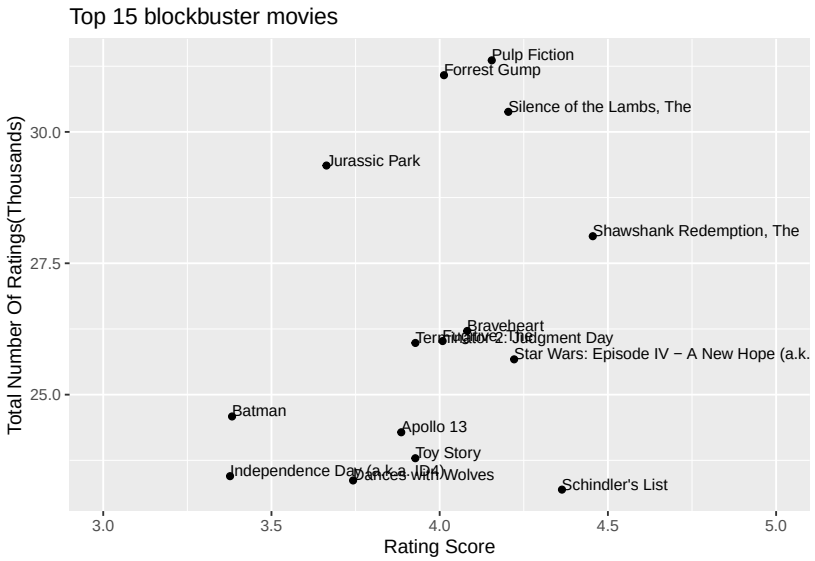
The average rate score is 3.512 stars while both median and mode rate score is 4 stars (distribution is negatively skewed).

About 2.5 milion ratings, which is 29% of the total ratings, are 4 stars and only 41% of the ratings is 3 stars and less.

In general, whole star ratings are more common than half star ratings, and are 79.5 percent of the total ratings, as shown in the next plot.



- Top 15 blockbuster movies



The plot above demonstrates the top 15 blockbuster movies with overall number of ratings higher than 22k, most of them are rated above the average score.

The top leader is Pulp Fiction, directed by Quentin Tarantino, and was released on 1994.

The challenge of recommendation systems

Since our goal is to predict the future ratings, we can treat this as a machine learning challenge and "fill" the empty cells from the above matrix.

From the user's prespective, we'll want to fit a rating which is as close as possible to the actual rating the user would have rated the movie, and use that for future users with similiar preferences outside of our control.

From the movie's prespective, we'll want to predict if the movie could become a blockbuster, or less than that.

We'll use the following features: UserId, MovieId, Age of the movie at rating, time (release year) and genre.

3 Analysis And Modeling

3.1 Create additional partition of training and test set

In this step, we'll create additional training and test sets, in order to evaluate the accuracy of the models we build and to prevent overfitting of the RMSE.

Once we hit **RMSE < 0.8649**, we'll use the final holdout test set (validation) and determine the final model RMSE.

```
set.seed(755, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(755)`

# randomly splitting edx data set into 80% training set and 20% testing set
test_index <- createDataPartition(y = edx_year_sanitized$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_edx <- edx_year_sanitized %>% slice(-test_index)
temp <- edx_year_sanitized %>% slice(test_index)

# making sure that the test set includes users and movies that appear in the training set.
```

```
test_edx <- temp %>%
  semi_join(train_edx, by = "movieId") %>%
  semi_join(train_edx, by = "userId")
removed <- anti_join(temp,test_edx)
train_edx <- rbind(train_edx, removed)
rm(temp, removed)
```

3.2 RMSE function

- **MSE Function**

  The MSE measures the average of the squares of the errors that is, the average squared difference between the estimated values and the actual value. Therefore, is a measure of the quality of an estimator, it is always non-negative, and values closer to zero are better.

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

- **RMSE** is a good measure of how accurately the model predicts the response, and it is the most important criterion for fit if the main purpose of the model is prediction.

  Lower values of RMSE indicate better fit.

  RMSE squares the residual, and is affected by outliers.

  The RMSE is analogous to the standard deviation (MSE to variance) and is a measure of how large the residuals are spread out.

- 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;

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

$$RMSE = \sqrt{MSE}$$

- N - the number of user/movie combinations and the sum occurring over all of these combinations.
- If the RMSE is larger than 1, it means our typical error is larger than one star.
- The function that computes the RMSE for vectors of ratings and their corresponding predictors;

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

3.3 Naive Model

3.3.1 The model

The naive model is the simplest possible recommendation system that predicts the same rating for all movies regardless of the user preferences.

The naive model takes into account a similar rating for all movies and users, and the differences are explained by random variation which will be defined by:

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

$\epsilon_{u,i}$; independent errors sampled from the same distribution centered at 0.

μ ; true rating for all movies

The estimate that minimizes the RMSE is the least squares estimate of μ and which is the $\hat{m}u$: the average rating of all movies across all users

```
mu_hat <- mean(train_edx$rating)
mu_hat
```

```
## [1] 3.512523
```

3.3.2 The model results

if we predict all unknown ratings with μ we obtain the following RMSE

```
naive_rmse <- RMSE(test_edx$rating, mu_hat)
naive_mse <- MSE(test_edx$rating, mu_hat)

#creating results table
naive_model_results <- tibble(method = "Average only",
                              MSE=naive_mse, RMSE = naive_rmse)
naive_model_results %>% knitr::kable()
```

method	MSE	RMSE
Average only	1.12479	1.060561

The RMSE we got is 1.06, which means our typical error is larger than one star, which is not good enough!

To remind ourselves, the goal is to aspire for RMSE < 0.8649

3.4 userId

3.4.1 User Activity

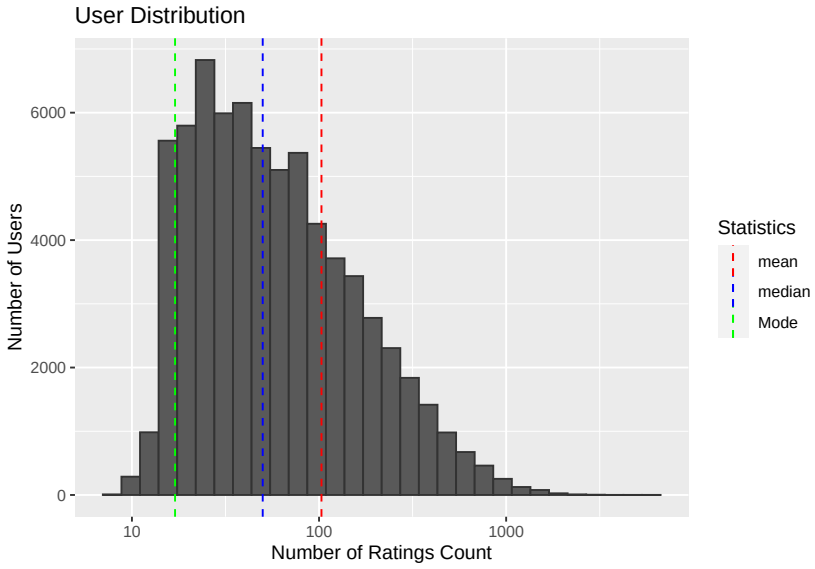
mean	median	mode	min	max
103	50	17	7	5327

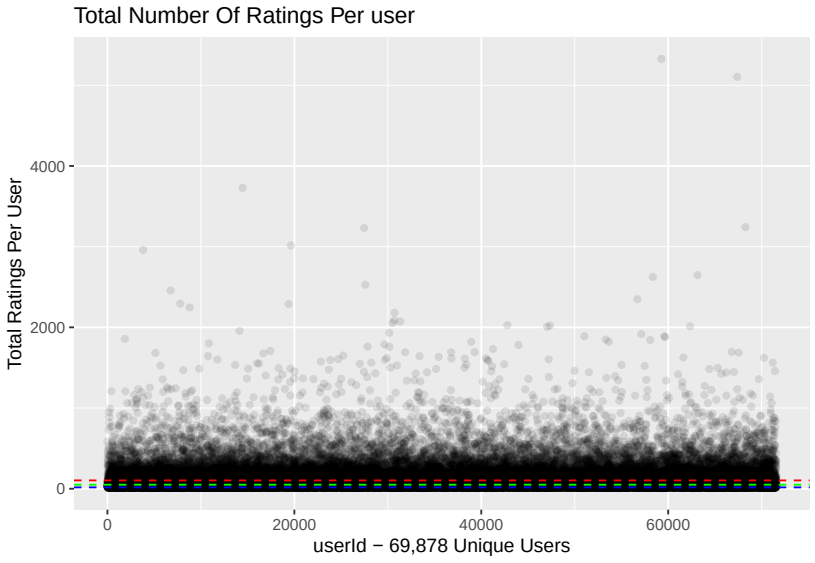
Each of the 70K different usesrs rated in average 103 different movies. The most frequent value of ratings is only 17 movies, but there are 3% users who rated more than 500 movies!

The distribution is positively skewed, as the mean is larger than the mode and median.

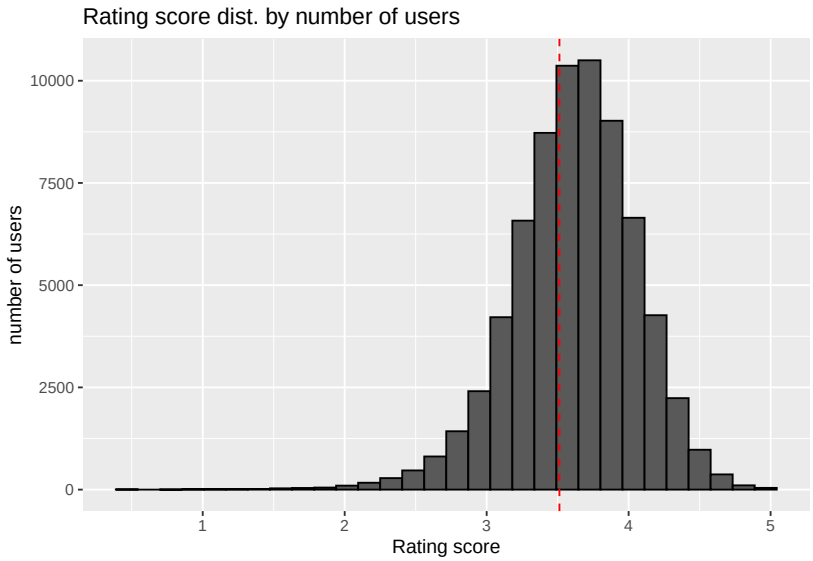
Both plots demonstrates extreme observations (outliers) e.g. very low or high number of ratings, which indicates that some users are more active than others at rating movies.

Outliers affect RMSE which is highly sensitive to them, and therefore we will have to treat this and set penalty term going forward.





50% of the ratings are between 3 to 4 stars. Some users love every movie they watch and simply rate most of them 5 stars, but the left tail inidicate users that are very critic and rate lot of 1-2 stars.



3.4.2 Model 1 - Modeling User Effect

The previous analysis showed that users are rating different from each other. also, some are very active while others rarely active.

We can augment the naive model by adding the term b_u to represent average ranking by user u:

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

b_u - the user specific effect/"bias" ; average rating by user u regardless of movie i

we can compute the predicted ratings using the next;

$$\hat{b_u} = mean(y_{u,i} - \hat{mu})$$

since the least squares estimate b_u is just the average of $y_{u,i} - \hat{mu}$ for each user u.

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

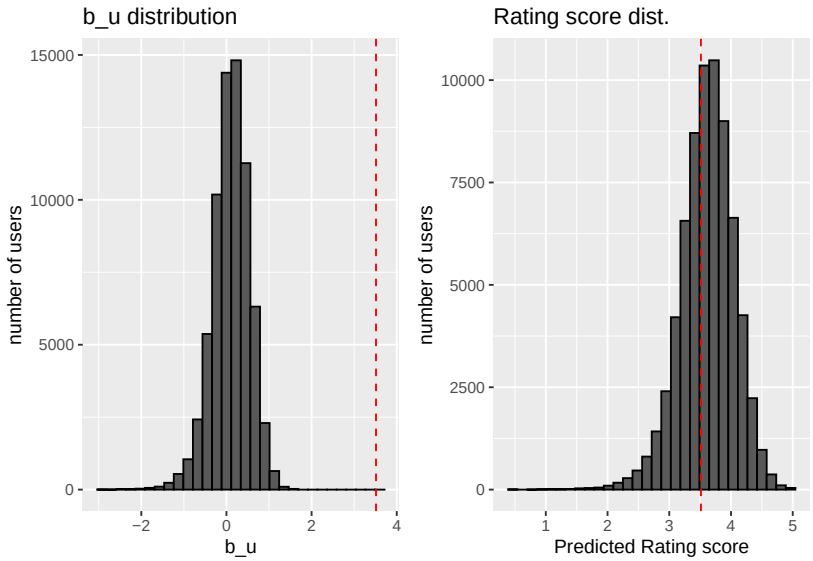
- Fit the model

```
fit_user_ave <-
  train_edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu))
```

- Predict the ratings - how much our prediction improves once using y=mu+b_u

```
predicted_ratings <-
  test_edx %>%
  left_join(fit_user_ave, by='userId') %>%
  mutate(predicted=mu+b_u) %>%
  pull(predicted)
```

We can see that the b_u distribution spans a wide symmetric range around 0, which indicates that the users rate equaly for good and bad.



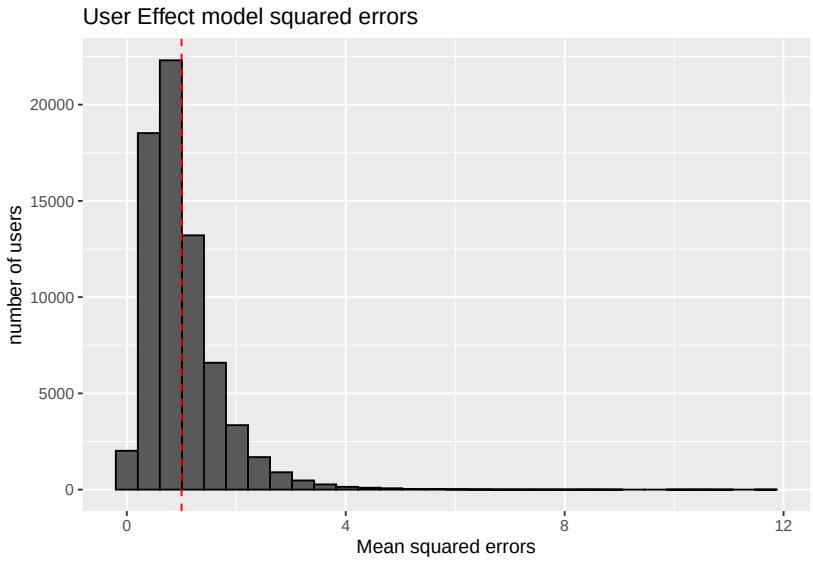
- Model results

```
# model RMSE
model_1_rmse <- RMSE(true_ratings=test_edx$rating,
                     predicted_ratings=predicted_ratings)
# model MSE
model_1_mse <- MSE(test_edx$rating,predicted_ratings)

#add the results to the table
model_1_results <- tibble(method = "User Effect",
                          MSE=model_1_mse, RMSE = model_1_rmse)
model_1_results %>% knitr::kable()
```

method	MSE	RMSE
User Effect	0.9587157	0.9791403

we obtain RMSE = 0.9791, only 8% improvement from the naive model.

- The model mean squared errors



- Model 1 squared error summarize;

```
##  squared_errors
##  Min.   : 0.00000
##  1st Qu.: 0.09288
##  Median : 0.39062
##  Mean   : 0.95872
##  3rd Qu.: 1.20695
##  Max.   :18.25620
```

The large distribution of the errors in the plot, with the heavy right tail, and MSE closer to 1 indicate low accuracy.

3.4.3 Model 1.1 - Modeling Reg. User Effect

Regularization

As mentioned before, RMSE is sensetive to outliers.

outliers can be interuped as b's "mistakes" -

- Users that systematically rate only 1 stars or only 5 stars
- movies that have only one rating and scored 5 stars will apear as "best" movie, the same for "worst" (1 rating of 1 star).

The supposed "best" and "worst" movies were rated by very few users, in most cases just 1. This is because with just a few users, we have more uncertainty. Therefore, larger estimates of b's, negative or positive, are more likely.

These are noisy estimates that we should not trust, especially when it comes to prediction.

Large errors can increase our RMSE, so we would rather be conservative when unsure.

Regularization permits us to penalize large estimates that are formed using small sample sizes.

The general idea is to add a penalty for large values of b's to the sum of squars equation that we minimize, since having many large b's makes it harder to minimize the equation that were trying to minimize.

We control the total variability of the user effects specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

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

The first term is just least squares and the second is a penalty that gets larger when many b's are large. Using calculus we can actually show that the values of bi that minimize this equation are:

$$\hat{b_i}(\lambda)=\frac{1}{\lambda+n_i}\sum_{u}(y_{u,i}-\hat{mu})$$

where n_i is the number of ratings made for movie i.

This approach will have our desired effect: when our sample size n_i is very large, a case which will give us a stable estimate, then the penalty is effectively ignored since .

However, when the n_i is small, then the estimate is shrunken towards 0. The larger, the more we shrink.

lambda is a tuning parameter. we will use cross-validation to choose it. we'll create additional partition of the training for cross validation, pick the lambda and evaluate the performance on the previos train+test.

```
set.seed(2020, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(2020)`

# randomly splitting train set into 90% training set and 10% testing set
test_index_cv <-
  createDataPartition(y = train_edx$rating,
                      times = 1, p = 0.1, list = FALSE)

train_edx_cv <- train_edx %>% slice(-test_index_cv)
temp <- train_edx %>% slice(test_index_cv)

# making sure that the test set includes users and movies that appear in the training set.

test_edx_cv <-
  temp %>%
  semi_join(train_edx_cv, by = "movieId") %>%
  semi_join(train_edx_cv, by = "userId")

removed <- anti_join(temp, test_edx_cv)
train_edx_cv <- rbind(train_edx_cv, removed)
rm(temp, removed)
```

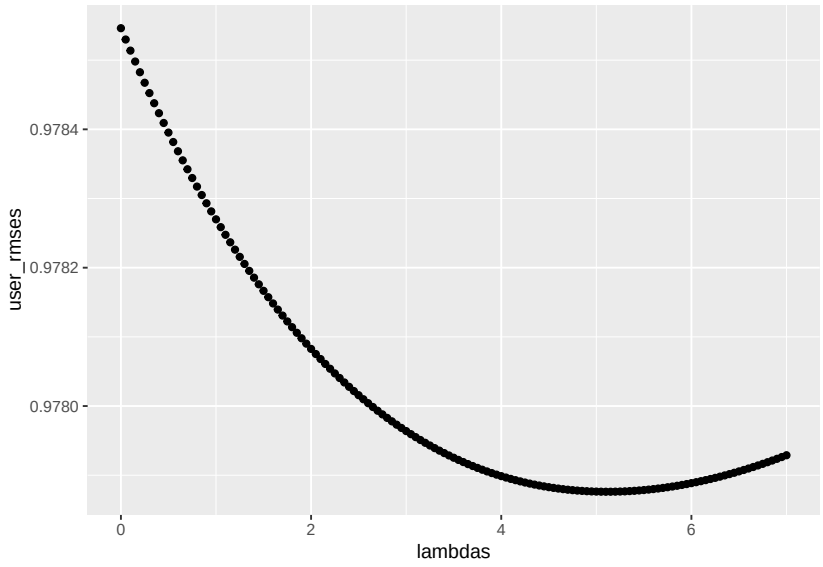
- The model will look like this:

$$Y_{u,i}=\mu+\ \lambda b_u+\epsilon_{u,i}$$

λ = Penalty term

- Choosing penalty term (lambda) for user effect;

```
equation_mu <- mean(train_edx_cv$rating)

equation_sum_u <-
  train_edx_cv %>%
  group_by(userId) %>%
  summarize(n_i=n(),
            s=sum(rating-equation_mu))

lambdas <- seq(0,7,0.05)

user_rmses <-
  sapply(lambdas,function(lambda){
    reg_predicted_ratings <-
      test_edx_cv %>%
      left_join(equation_sum_u, by="userId") %>%
      mutate(reg_b_u=(s/(n_i+lambda)),
             predicted=(equation_mu+reg_b_u)) %>%
      pull(predicted)

    return(RMSE(true_ratings=test_edx_cv$rating,
                predicted_ratings=reg_predicted_ratings))
  })

qplot(lambdas,user_rmses)
```




```
penalty_term <- lambdas[which.min(user_rmses)]
penalty_lambda_rmse <- c(lambda=penalty_term,
                         cv_rmse=user_rmses[lambda=penalty_term])
penalty_lambda_rmse
```

```
##    lambda   cv_rmse
## 5.1000000 0.9784825
```

- Apply lambda on edx train and test set

```
fit_reg_user_ave <-
  train_edx %>%
  group_by(userId) %>%
  summarize(n_i=n(),
            reg_b_u=(sum(rating - mu)/(n_i+penalty_term)))
```

- Penalized prediction

```
reg_predicted_ratings <-
  test_edx %>%
  left_join(fit_reg_user_ave, by='userId') %>%
  mutate(predicted=mu+reg_b_u) %>%
  pull(predicted)
```

- Penalized model results

```
model_1_1_rmse <- RMSE(true_ratings=test_edx$rating,
                       predicted_ratings=reg_predicted_ratings)
model_1_1_mse <- MSE(test_edx$rating,reg_predicted_ratings)

# add the results to the table
model_1_1_results <- tibble(method = "Reg. User Effect",
                            MSE=model_1_1_mse, RMSE = model_1_1_rmse)
model_1_1_results %>% knitr::kable()
```

method	MSE	RMSE
Reg. User Effect	0.957556	0.9785479

- Model 1.1 squared error summarize

```
test_edx %>%
  left_join(fit_reg_user_ave, by='userId') %>%
  select(userId,rating,reg_b_u) %>%
  mutate(predicted=reg_b_u+mu,
         reg_squared_errors=(rating-predicted)^2) %>%
  select(reg_squared_errors) %>%
  summary()
```

```
##  reg_squared_errors
##  Min.   : 0.00000
##  1st Qu.: 0.09459
##  Median : 0.39040
##  Mean   : 0.95756
##  3rd Qu.: 1.21577
##  Max.   :16.88294
```

We obtain RMSE= 0.9785479, the penalized estimates does not provide much improvement to the RMSE. moving forward by testing the movie specific effect on the predicrion.

3.5 movieId

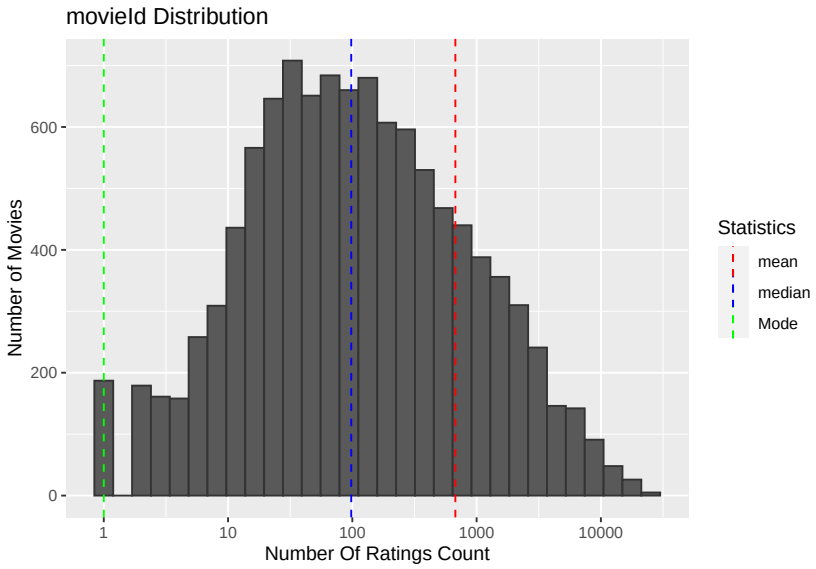
3.5.1 Movie Rating

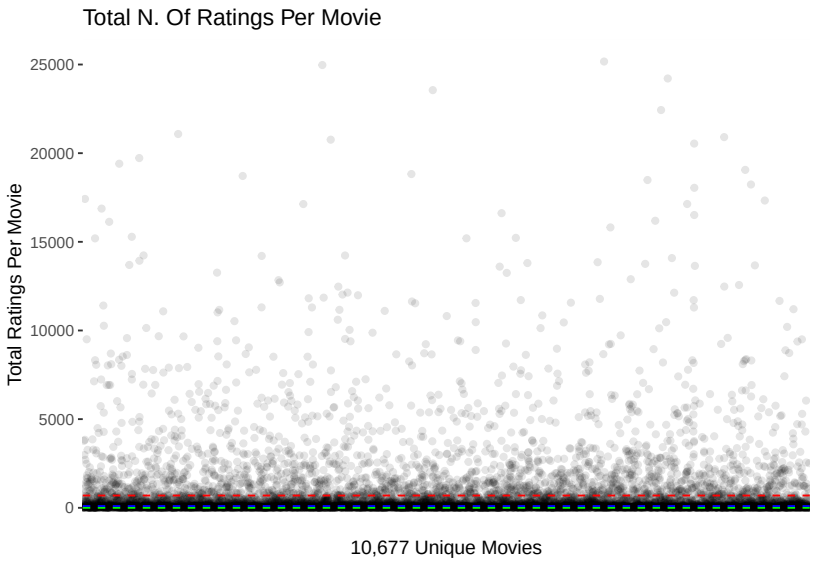
mean	median	mode	min	max
674	98	1	1	25169

In average, each one of the 10.6k different movies get rated 674 times.

The movies most frequently gets only 1 ratings, but there are exeptional like “Pulp Fiction” that got rated 3734% (!) more than average with 25169 users ratings.

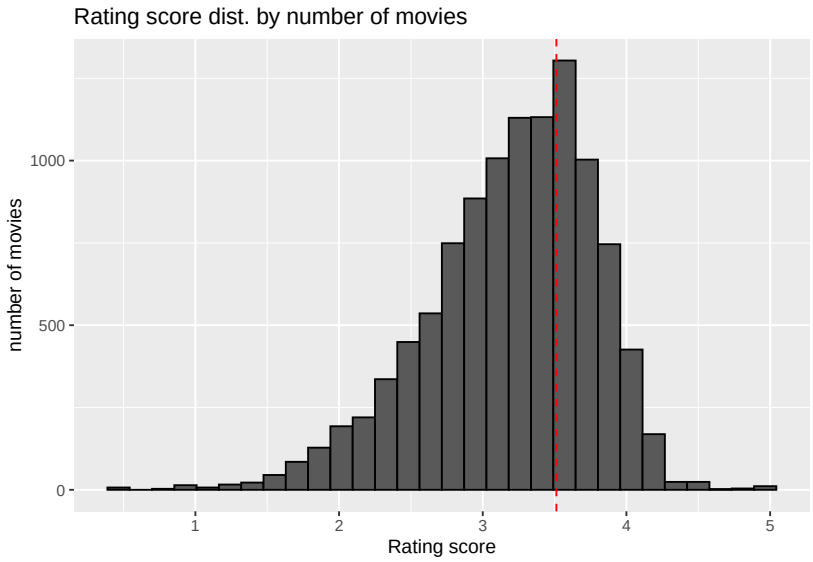
the distribution is positively skewed, like the user activity dist., as the mean is larger than the mode and median.





Some movies are just generally rated higher than others.

50% of the ratints are between 2.8 to 3.6 stars. very few gets prefect 5 srars, respectively very few got the shady 1 star.



3.5.2 Model 2 - Modeling Movie Effect

We can augment the naive model by adding the term b_i to represent average ranking for movie i:

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

b_i - the movie specific effect/"bias" ; average rating for movie i regardless of user

we can compute the predicted ratings using the next;

$$\hat{b_i} = mean(y_{u,i} - \hat{mu})$$

since the least squares estimate b_i is just the average of $y_{u,i} - \hat{mu}$ for each movie i.

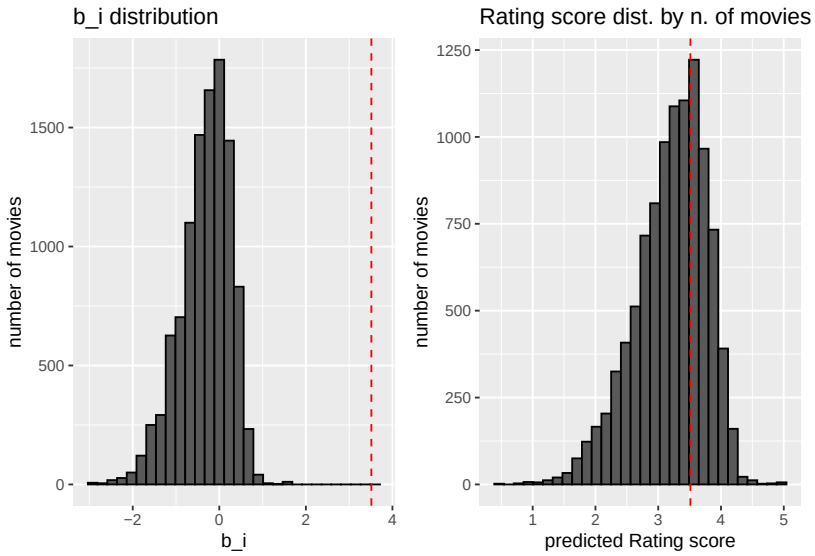
- Fit the model

```
fit_movie_ave <-
  train_edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

- Predict the rarings - how much our prediction improves once using y=mu+b_i

```
predicted_ratings <-
  test_edx %>%
  left_join(fit_movie_ave, by='movieId') %>%
  mutate(predicted = mu + b_i) %>%
  pull(predicted)
```

We can see that the b_i distribution more negatively which means moives get lower ratings.



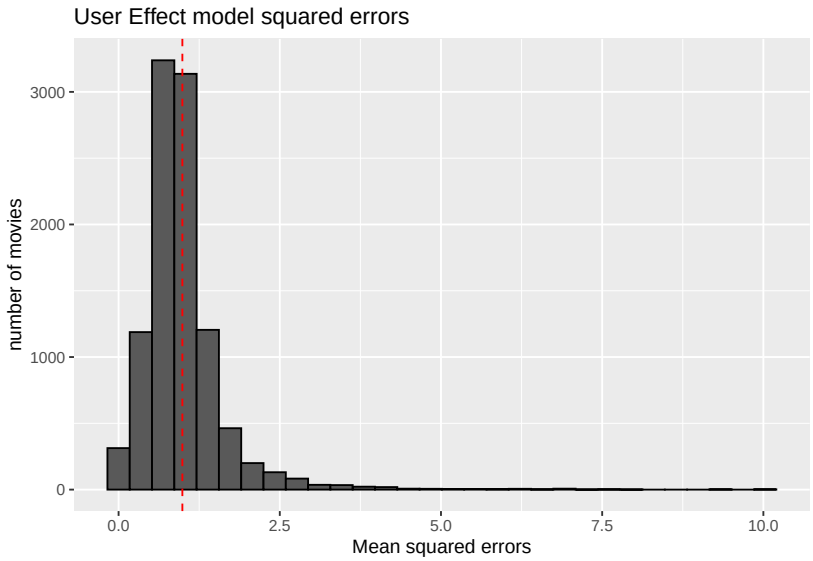
- Model results

```
# model RMSE
model_2_rmse <- RMSE(true_ratings=test_edx$rating,
                     predicted_ratings=predicted_ratings)
# model MSE
model_2_mse <- MSE(test_edx$rating,predicted_ratings)

#add the results to the table
model_2_results <- tibble(method = "Movie Effect",
                          MSE=model_2_mse, RMSE = model_2_rmse)
model_2_results %>% knitr::kable()
```

method	MSE	RMSE
Movie Effect	0.8911112	0.9439868

We obtain RMSE=0.9439868, improvement of 11% from the naive model RMSE, but it's better than the model with just the user effect.

- The model mean squared errors



- The model mean squared errors summary

```
test_edx %>%
  left_join(fit_movie_ave, by='movieId') %>%
  select(movieId,rating,b_i) %>%
  mutate(predicted=b_i+mu,
         squared_errors=(rating-predicted)^2) %>%
  select(squared_errors) %>%
  summary()
```

```
##  squared_errors
##  Min.   : 0.00000
##  1st Qu.: 0.08024
##  Median : 0.38220
##  Mean   : 0.89111
##  3rd Qu.: 1.09453
##  Max.   :16.00000
```

The squared errors distribution is large, which indicate low accuracy. The MSE distribution tail quite shorter than the user effect model, but can be improved by penalize the outliers.

3.5.3 Model 2.1 - Modeling Reg. Movie Effect

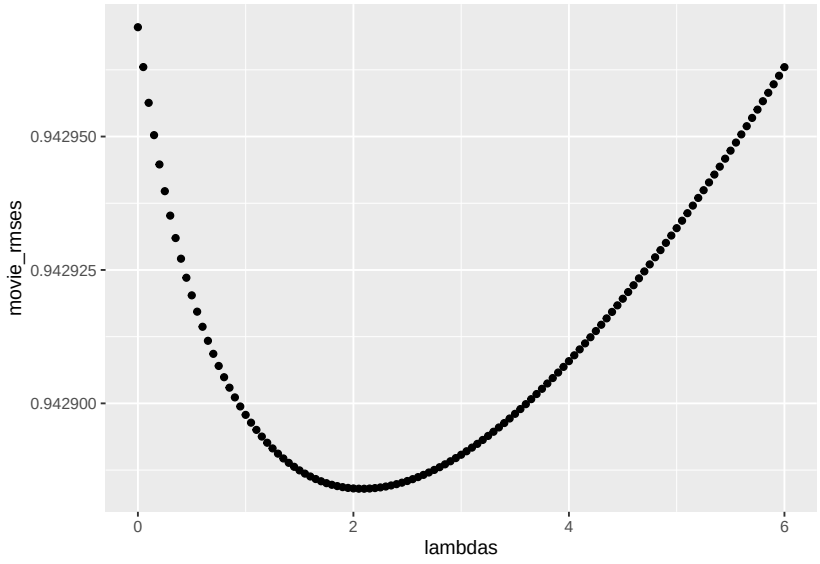
- The model will look like this:

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

λ = Penalty term

- Choosing penalty term (lambda) for movie effect

```r
equation_mu <- mean(train_edx_cv$rating)

equation_sum_m <-
  train_edx_cv %>%
  group_by(movieId) %>%
  summarize(n_i=n(),
            s=sum(rating-equation_mu))

lambdas <- seq(0,6,0.05)

movie_rmses <-
  sapply(lambdas,function(lambda){
    reg_predicted_ratings <-
      test_edx_cv %>%
      left_join(equation_sum_m, by="movieId") %>%
      mutate(reg_b_i=(s/(n_i+lambda)),
             predicted=(equation_mu+reg_b_i)) %>%
      pull(predicted)
    return(RMSE(true_ratings=test_edx_cv$rating,
                predicted_ratings=reg_predicted_ratings))
  })

qplot(lambdas,movie_rmses)
```




```r
penalty_term <- lambdas[which.min(movie_rmses)]
penalty_lambda_rmse <- c(lambda=penalty_term,
                         cv_rmse=movie_rmses[lambda=penalty_term])
penalty_lambda_rmse
```

```
##   lambda  cv_rmse
## 2.100000 0.942963
```

- Apply lambda on edx train and test set

```
fit_reg_movie_ave <-
  train_edx %>%
  group_by(movieId) %>%
  summarize(n_i=n(),
            reg_b_i=(sum(rating - mu)/(n_i+penalty_term)))
```

- Penalized prediction

```
reg_predicted_ratings <-
  test_edx %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
  mutate(predicted=mu+reg_b_i) %>%
  pull(predicted)
```

- Penalize model results

```
model_2_1_rmse <- RMSE(true_ratings=test_edx$rating,
                       predicted_ratings=reg_predicted_ratings)

model_2_1_mse <- MSE(test_edx$rating,reg_predicted_ratings)

#add the results to the table
model_2_1_results <- tibble(method = "Reg. Movie Effect",
                            MSE=model_2_1_mse, RMSE = model_2_1_rmse)
model_2_1_results %>% knitr::kable()
```

method	MSE	RMSE
Reg. Movie Effect	0.8909883	0.9439218

The penalized model obtain RMSE= 0.9439218. There is no improvement with the penalty term.

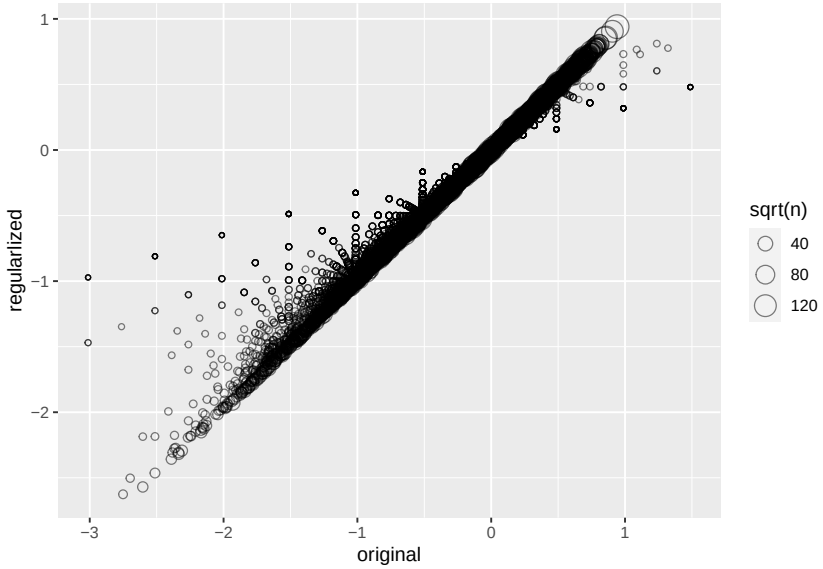
```
test_edx %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
  select(movieId,rating,reg_b_i) %>%
  mutate(predicted=reg_b_i+mu,
         reg_squared_errors=(rating-predicted)^2) %>%
  select(reg_squared_errors) %>%
  summary()
```

```
##  reg_squared_errors
##  Min.   : 0.00000
##  1st Qu.: 0.08015
##  Median : 0.38201
##  Mean   : 0.89099
##  3rd Qu.: 1.09525
##  Max.   :15.62615
```

- Comparing between the estimates

To see how the estimates shrink, we'll plot the regularized estimates vs. least squared estimates

```
data_frame(original = fit_movie_ave$b_i,
           regularlized = fit_reg_movie_ave$reg_b_i,
           n = fit_reg_movie_ave$n_i) %>%
  ggplot(aes(original, regularlized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```



The size of the circle represent the size of n_i (count of ratings).

When n_i is small, the values are shrinking more toward zero.

3.6 movieId and userId

3.6.1 Model 3 - Modeling Movie + User effect

Modeling the movie effect and the user effect seperatly did not offer more than 7%-10% of improvement to the naive RMSE.

Common sense tells that although each one has its own effect, users tend to watch and rate movies for a reason.

Therefore it is much right to model them together.

The model will look like this:

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

$b_{u,i}$ - average rating for movie i with user specific effect.

In this case, if a cranky user (= negative $b_{u,i}$) rates a great movie (= positive b_i), the effects counter each other and we may be able to correctly predict that this user rated this great movie with 3 stars rather than 5 stars, and that should improve our prediction.

- Fit the model

```
fit_user_movie_ave <-
  train_edx %>%
  left_join(fit_movie_ave, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_ui = mean(rating - mu - b_i))
```

- Predict the ratings - how much our prediction improves once using y=mu+b_i+b_ui

```
predicted_ratings <-
  test_edx %>%
  left_join(fit_movie_ave, by='movieId') %>%
  left_join(fit_user_movie_ave, by='userId') %>%
  mutate(predicted = mu+b_i+b_ui) %>%
  pull(predicted)
```

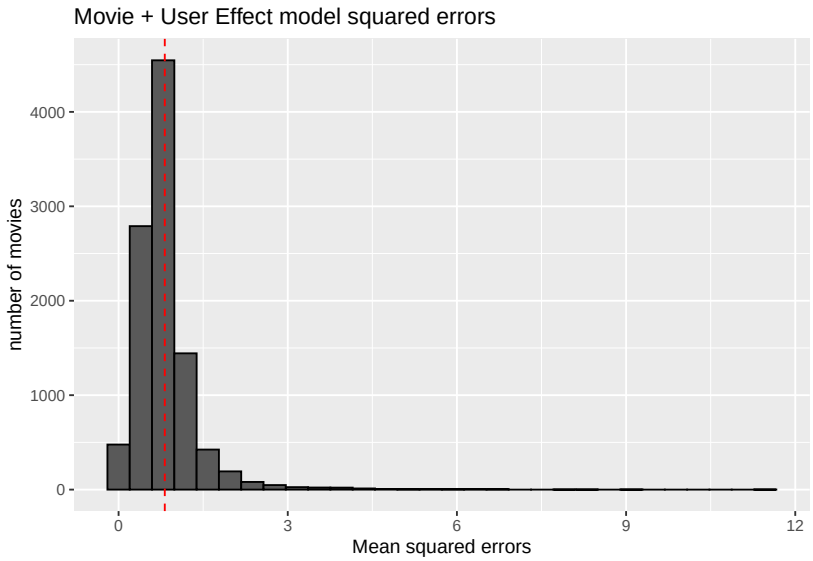
- Model results

```
# model RMSE
model_3_rmse <- RMSE(true_ratings=test_edx$rating,
                     predicted_ratings=predicted_ratings)
# model MSE
model_3_mse <- MSE(test_edx$rating,predicted_ratings)

#add the results to the table
model_3_results <- tibble(method = "Movie + User Effect",
                          MSE=model_3_mse, RMSE = model_3_rmse)
model_3_results %>% knitr::kable()
```

method	MSE	RMSE
Movie + User Effect	0.7510663	0.8666408

The model RMSE= 0.8666408, 18.2% improvement from the naive model, which is already very significant and shows good correlation between user and a movie.

- The model mean squared errors



- The model squared errors summary

```
##  squared_errors
##  Min.   : 0.00000
##  1st Qu.: 0.06343
##  Median : 0.29341
##  Mean   : 0.75107
##  3rd Qu.: 0.89520
##  Max.   :25.48224
```

Althogh we saw RMSE improvement, the squared error shows large distribution. We will try to "fix" it by the penalizing, and reduce very large absolute values of b_i and $b_{u,i}$

3.6.2 Model 3.1 - Modeling Reg. Movie + User effect

- The model will look like this:

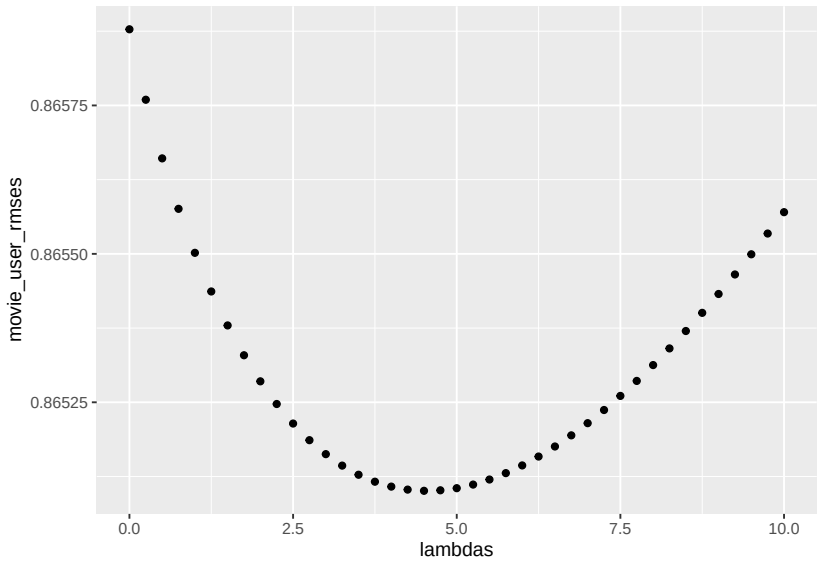
$$Y_{u,i} = \mu + \ \lambda(b_i + b_{u,i}) + \epsilon_{u,i}$$

λ = Penalty term

- Choosing penalty term (lambda) for movie + user effect

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

equation_mu <- mean(train_edx_cv$rating)

movie_user_rmses <-
  sapply(lambdas,function(lambda){
    fit_reg_movie_ave <-
      train_edx_cv %>%
      group_by(movieId) %>%
```

```
      summarize(n_i=n(),
                s= sum(rating - equation_mu),
                reg_b_i=(s/(n_i+lambda)))

    fit_reg_user_movie_ave <-
      train_edx_cv %>%
      left_join(fit_reg_movie_ave, by='movieId') %>%
      group_by(userId) %>%
      summarize(n_i=n(),
                s= sum(rating -reg_b_i -equation_mu),
                reg_b_ui=(s/(n_i+lambda)))

    reg_predicted_ratings <-
      test_edx_cv %>%
      left_join(fit_reg_movie_ave, by='movieId') %>%
      left_join(fit_reg_user_movie_ave, by='userId') %>%
      mutate(predicted = equation_mu+reg_b_i+reg_b_ui) %>%
      pull(predicted)

    return(RMSE(true_ratings=test_edx_cv$rating,
                predicted_ratings=reg_predicted_ratings))

  })

qplot(lambdas,movie_user_rmses)
```




```
penalty_term <- lambdas[which.min(movie_user_rmses)]
penalty_lambda_rmse <- c(lambde=penalty_term,
                         cv_rmse=movie_user_rmses[lambda=penalty_term])
penalty_lambda_rmse
```

```
##    lambde   cv_rmse
## 4.5000000 0.8655759
```

- Apply lambda on edx train and test set

```
fit_reg_movie_ave <-
  train_edx %>%
  group_by(movieId) %>%
  summarize(n_i=n(),
            s= sum(rating - mu),
            reg_b_i=(s/(n_i+penalty_term)))

fit_reg_user_movie_ave <-
  train_edx %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
  group_by(userId) %>%
  summarize(n_i=n(),
            s= sum(rating -reg_b_i -mu),
            reg_b_ui=(s/(n_i+penalty_term)))
```

- Penalized prediction

```
reg_predicted_ratings <-
  test_edx %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
  left_join(fit_reg_user_movie_ave, by='userId') %>%
  mutate(predicted = mu+reg_b_i+reg_b_ui) %>%
  pull(predicted)
```

- Penalized model results

```
model_3_1_rmse <- RMSE(true_ratings=test_edx$rating,
                       predicted_ratings=reg_predicted_ratings)

model_3_1_mse <- MSE(test_edx$rating,reg_predicted_ratings)

#add the results to the table
model_3_1_results <- tibble(method = "Reg. Movie + User Effect",
                            MSE=model_3_1_mse, RMSE = model_3_1_rmse)
model_3_1_results %>% knitr::kable()
```

method	MSE	RMSE
Reg. Movie + User Effect	0.7498952	0.8659649

The penalized model obtain RMSE = 0.8659649 which takes us even closer to our goal.

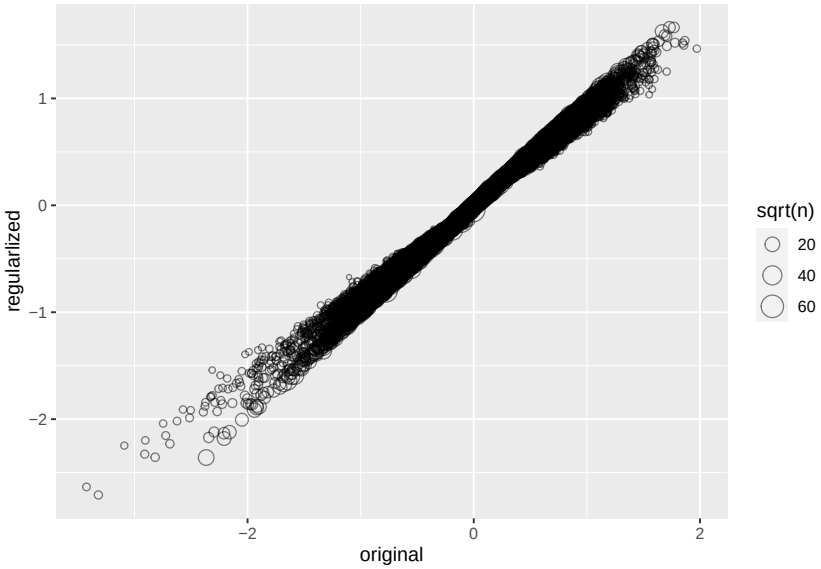
- The model reg squared errors summary

```
##  reg_squared_errors
##  Min.   : 0.00000
##  1st Qu.: 0.06373
##  Median : 0.29473
##  Mean   : 0.74989
##  3rd Qu.: 0.89648
##  Max.   :24.70511
```

- Comparing between the estimates

To see how the estimates shrink, we'll plot the regularized estimates vs. least squre estimates

```
data_frame(original = fit_user_movie_ave$b_ui,
           regularlized = fit_reg_user_movie_ave$reg_b_ui,
           n=fit_reg_user_movie_ave$n_i) %>%
  ggplot(aes(original, regularlized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```



3.7 Age of the movie at rating

3.7.1 How old was the movie during rating

Having the joint effect of movie and user is not enough since the users ratings are affected by other factors.

One factor for exampe, is the age of the movie at rating. There are users that likes the quality of "old times" movies and favors aged movies, regardless of the specific movie. It can be considered as "Old Movies Genre".

We will model the effect of the age of the movie on the user.

We also need to rememmber that a movie can be rated multiple times by different users at different years.

Age of a movie at rating summary;

mean	median	mode	min	max
11.97702	7	1	0	93

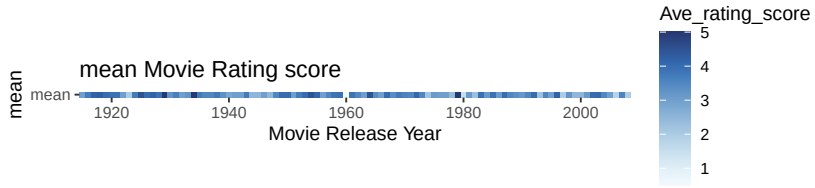
The average age of the movies is 12 years.

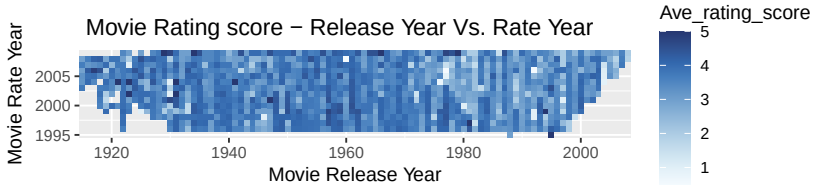
The youngest movies being rated were the ones that rated the same year of release. The oldest is 93 years old.

Most frequently rated are movies one year after they released.

The next tile plot shows the average score of the movies at rating vs. their release year. The tiled colors represent the rate score; darker tile indicates higher score.

We can see that older movies score higher in average, but the effect is not much significant.





The next table show example of movies with more than total of 1000 ratings, that released in 1984.

The rate years are 1999-2000, which means the age of the movies at rating is around 3 to 4 years.

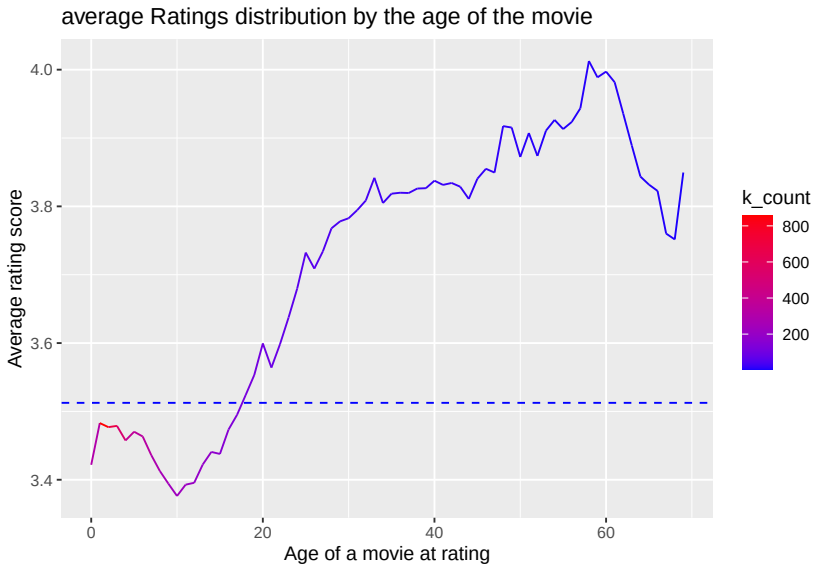
In this random example we can see the change of the average rating score for each movie by its age, and the average rate score of the entire rate year.

- 1984 release year example (filter movies with less than 1000 ratings)

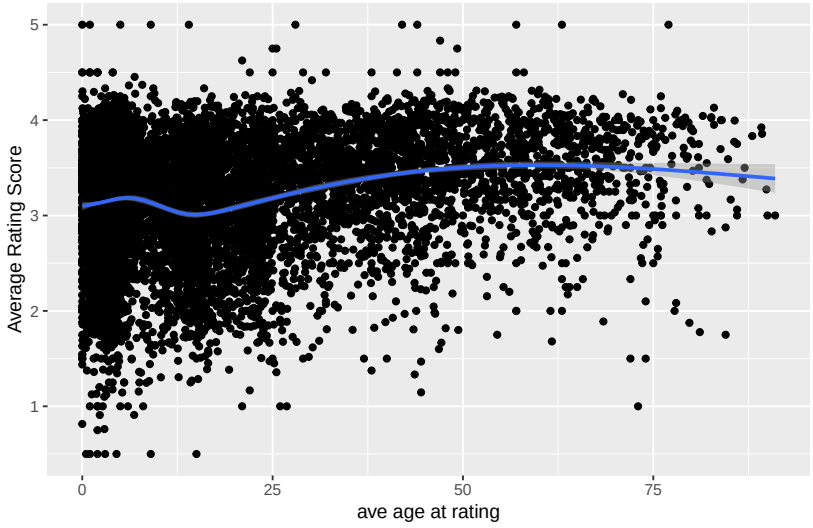
title	rate_year	count	Ave_rating_score	release_year	ave
Amadeus	1999	1100	4.261	1984	3.882
Ghostbusters (a.k.a. Ghost Busters)	1999	1514	3.716	1984	3.882
Romancing the Stone	1999	1186	3.525	1984	3.882
Terminator, The	1999	1371	4.028	1984	3.882
Amadeus	2000	1307	4.233	1984	3.876
Ghostbusters (a.k.a. Ghost Busters)	2000	1991	3.858	1984	3.876
Indiana Jones and the Temple of Doom	2000	1091	3.664	1984	3.876
Romancing the Stone	2000	1214	3.657	1984	3.876
Splash	2000	1026	3.442	1984	3.876
Terminator, The	2000	1922	4.095	1984	3.876

The next plot represent the averge rate score of the age at rating (for movies with 5k total ratings and up).

Old movies score higher, but they also rated less times, which need to take into penalize considaration.



The points at the plot represent different movies. There are many more newer movies than older, however there is some evidence for age effect



3.7.2 Classical Hollywood cinema

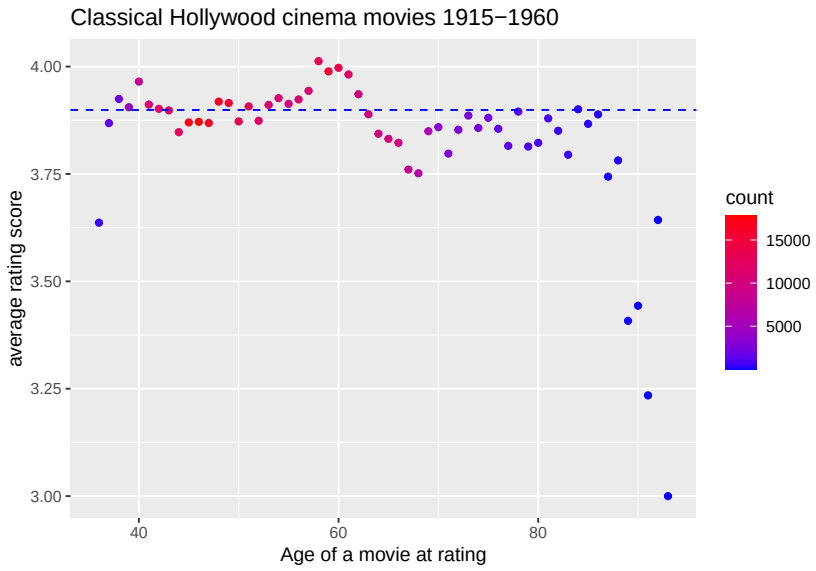
Classical Hollywood cinema, a genre of age, is a term used in film criticism to describe both a narrative and visual style of filmmaking which became characteristic of American cinema between the 1910s (rapidly after World War I) and the 1960s.

(https://en.wikipedia.org/wiki/Classical_Hollywood_cinema)

There are 1.3k Classical Hollywood cinema movies in our data with average age of 54 years old and average rating of 3.9 stars, 0.4 star more than the general average.

n_movies	ave_rating	ave_age	min_age	max_age
1287	3.8977	54.42969	36	93

Classical Hollywood cinema 36-93 years old movies and their average score as shown;



3.7.3 Model 4 - Reg. User and Age of the movie at rating Effect

The model will group the age of the movie at rating regardles of its title since each movie rated several times over the years.

Previous analysis on the user effect taught us that the model should be penalized, also, the analysis on the age at rating indicate of large variation between the rating counts and the rating scores.

The model will look like this:

$$Y_{u,i} = \mu + \ \lambda b_u + b_a + \epsilon_{u,i}$$

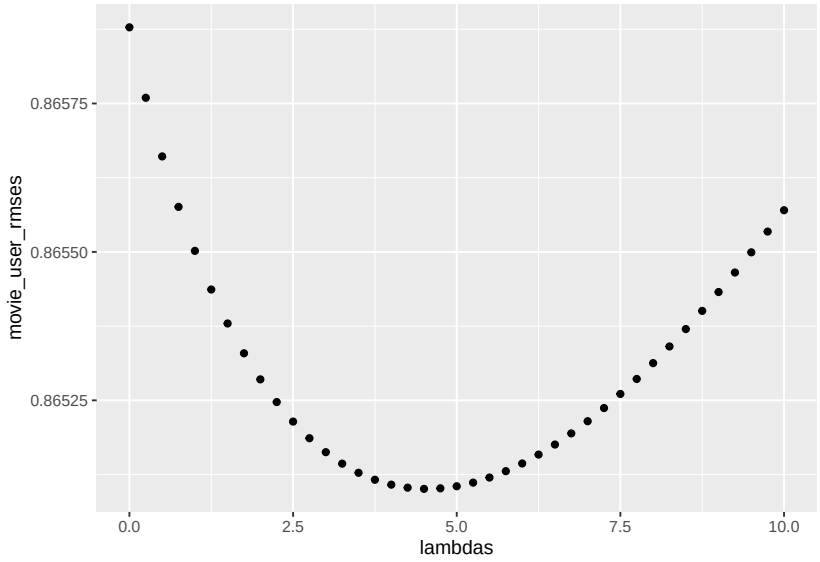
b_a - average rating of a movie based on his age at rating.

We'll choose penalize model in advance;

- Choosing penalty term (lambda) for User and Age effect

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

equation_mu <- mean(train_edx_cv$rating)

user_age_rmses <-
  sapply(lambdas,function(lambda){
    fit_reg_user_ave <-
      train_edx_cv %>%
      mutate(age_at_rating= abs(rate_year-release_year)) %>%
      filter(age_at_rating>=0)%>%
      group_by(userId) %>%
      summarize(n_i=n(),
                s= sum(rating - equation_mu),
                reg_b_u=(s/(n_i+lambda)))

    fit_reg_user_age_ave <-
      train_edx_cv %>%
```

```
      mutate(age_at_rating= abs(rate_year-release_year)) %>%
      filter(age_at_rating>=0)%>%
      left_join(fit_reg_user_ave, by='userId') %>%
      group_by(age_at_rating) %>%
      summarize(n_i=n(),
                s= sum(rating -reg_b_u -equation_mu),
                reg_b_ua=(s/(n_i+lambda)))

    reg_predicted_ratings <-
      test_edx_cv %>%
      mutate(age_at_rating= abs(rate_year-release_year)) %>%
      filter(age_at_rating>=0)%>%
      left_join(fit_reg_user_ave, by='userId') %>%
      left_join(fit_reg_user_age_ave, by='age_at_rating') %>%
      mutate(predicted = equation_mu+reg_b_u+reg_b_ua) %>%
      pull(predicted)

    return(RMSE(true_ratings=test_edx_cv$rating,
                predicted_ratings=reg_predicted_ratings))
  })

qplot(lambdas,movie_user_rmses)
```




```
penalty_term <- lambdas[which.min(user_age_rmses)]
penalty_lambda_rmse <- c(lambda=penalty_term,
                         cv_rmse=user_age_rmses[lambda=penalty_term])
penalty_lambda_rmse
```

```
##    lambda   cv_rmse
## 5.7500000 0.9710351
```

- Apply lambda on edx train and test set

```
fit_reg_user_ave <-
  train_edx %>%
  mutate(age_at_rating= abs(rate_year-release_year)) %>%
  filter(age_at_rating>=0)%>%
  group_by(userId) %>%
  summarize(n_i=n(),
            s= sum(rating - mu),
            reg_b_u=(s/(n_i+penalty_term)))

fit_reg_user_age_ave <-
  train_edx %>%
  mutate(age_at_rating= abs(rate_year-release_year)) %>%
  filter(age_at_rating>=0)%>%
  left_join(fit_reg_user_ave, by='userId') %>%
  group_by(age_at_rating) %>%
  summarize(n_i=n(),
            s= sum(rating -reg_b_u -mu),
            reg_b_ua=(s/(n_i+penalty_term)))
```

- Penalized predicted ratings

```
reg_predicted_ratings <-
  test_edx %>%
  mutate(age_at_rating= abs(rate_year-release_year)) %>%
  filter(age_at_rating>=0)%>%
  left_join(fit_reg_user_ave, by='userId') %>%
  left_join(fit_reg_user_age_ave, by='age_at_rating') %>%
  mutate(predicted = mu+reg_b_u+reg_b_ua) %>%
  pull(predicted)
```

- Penalized model results

```
model_4_rmse <- RMSE(true_ratings=test_edx$rating,
                     predicted_ratings=reg_predicted_ratings)

model_4_mse <- MSE(test_edx$rating,reg_predicted_ratings)

#add the results to the table
model_4_results <-
  tibble(method = "Reg. User + Age of the movie at rating Effect",
         MSE=model_4_mse, RMSE = model_4_rmse)
model_4_results %>% knitr::kable()
```

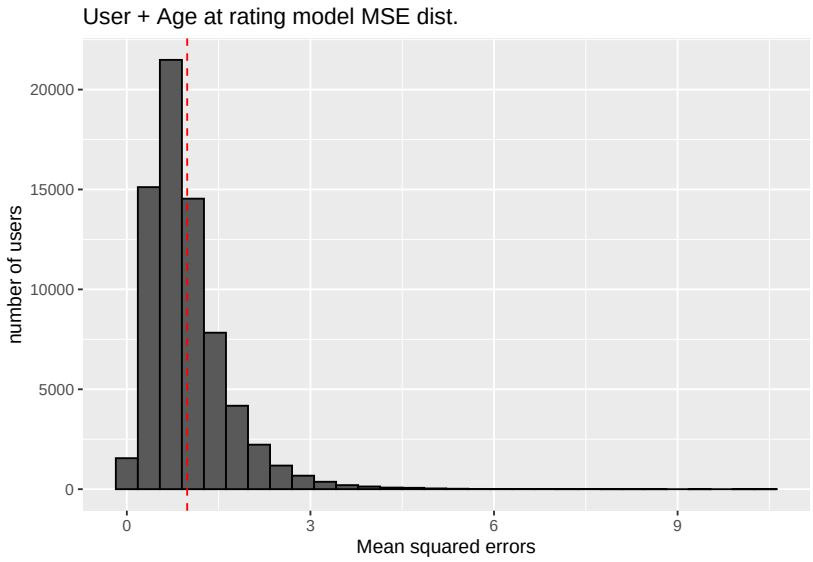
method	MSE	RMSE
Reg. User + Age of the movie at rating Effect	0.9433008	0.9712367

The penalized model obtain RMSE 0.9712367, improvement of only 8.3% than the naive model and only 1% better than the user effect model.

- The residual summary as shown;

```
##  reg_squared_errors
##  Min.   : 0.00000
##  1st Qu.: 0.09205
##  Median : 0.38008
##  Mean   : 0.94330
##  3rd Qu.: 1.19302
##  Max.   :18.85896
```

- The mean squared residual distribution



The model MSE distribution show long right tail that indicates low accuracy.

We'll move forward and look for more influencing factors.

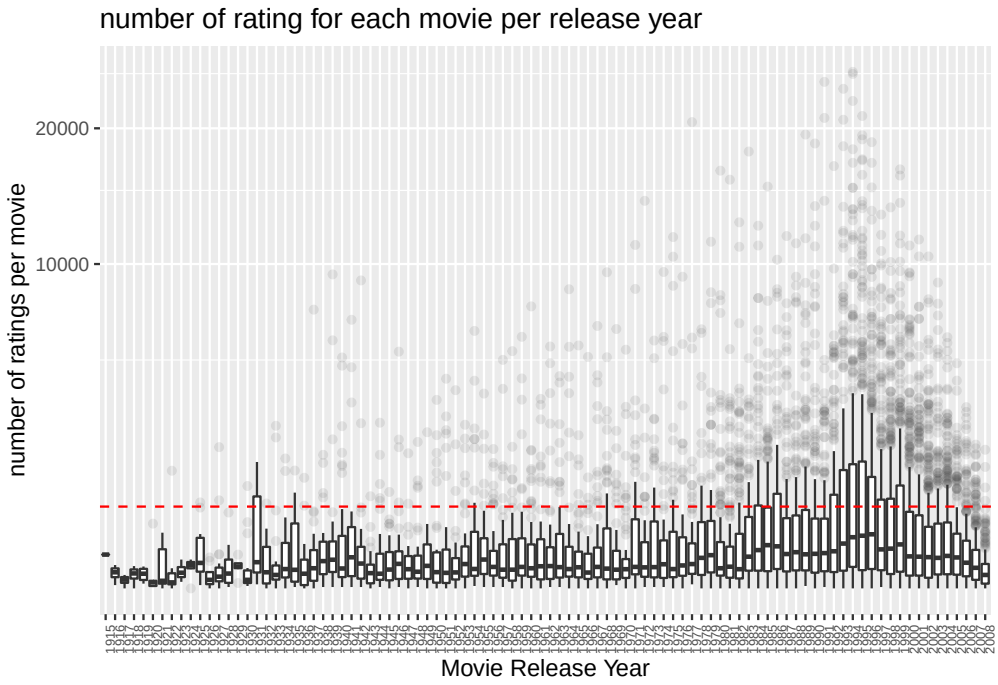
3.8 Time Frame Ranges

Movie release stretched over 93 years, while they first started to be rated after 80 years

Year	First	Last	Range in years
Release	1915	2008	93
rate	1995	2009	14

3.8.1 Release year

The number of ratings for each movie against the year the movie came out (each point represents a different movie);



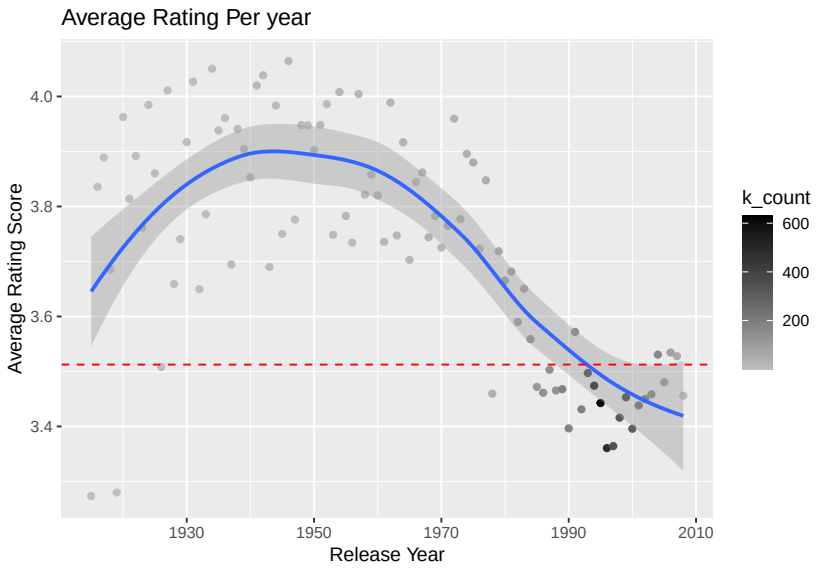
362 movies that releasd in 1995 got the highest ratings count.

Movies that released between the years 1992-1999 shows above average ratings count while starting in 1993 the number of the movies being rated decreases with year.

The more recent movie is, the less time users have had to rate it.

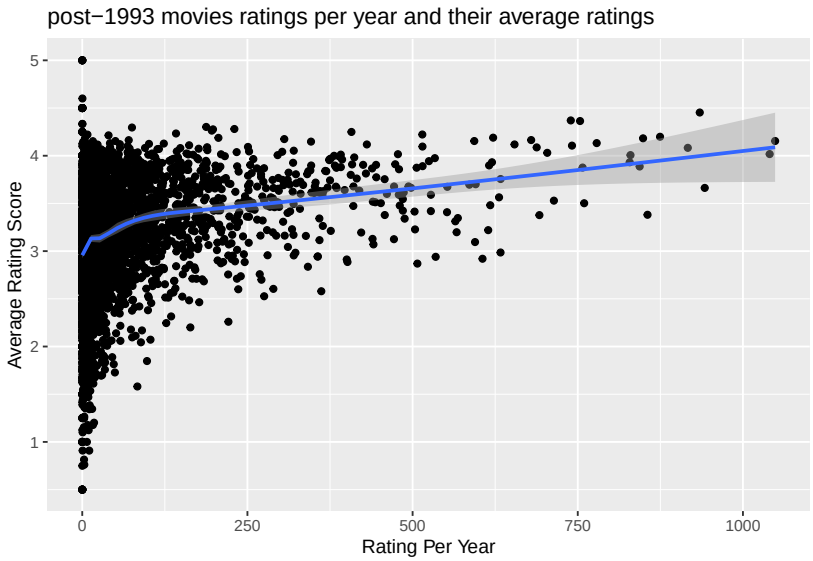
In addition, as the total rating count increases, the total rating score decreases.

Newer released movies gets more rated, but the amount of the ratings affects the average score and lower it.



The next plot shows post-1993 movies - ratings per year and their average ratings. Each point represent unique movie.

As the rating per year increases, the average rate score increases, in other words, the more often a movie is rated, the higher its average rating.



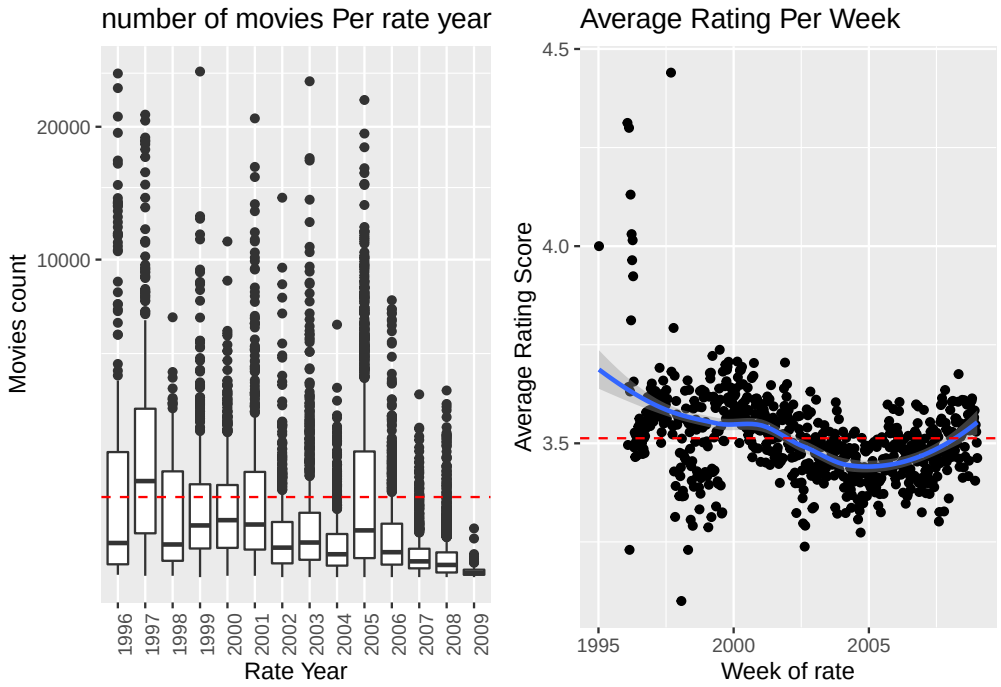
Here are the top 10 movies with the most ratings per year, along with their average ratings (represent the upper right part of the previous plot)

title	Total_ratings_count	years	ave_rating	rating_per_year
Pulp Fiction	25169	24	4.15	1048.71
Forrest Gump	24971	24	4.02	1040.46
Jurassic Park	23555	25	3.66	942.20
Shawshank Redemption, The	22432	24	4.45	934.67
Braveheart	21081	23	4.08	916.57
Matrix, The	16616	19	4.20	874.53
Independence Day (a.k.a. ID4)	18823	22	3.38	855.59
American Beauty	16128	19	4.18	848.84
Apollo 13	19405	23	3.89	843.70
Fugitive, The	20745	25	4.01	829.80

3.8.2 Rate year

The number of the movies being rated generally decreases every year.

The trend of the average rating score is not distinct, but both plots indicate that there is some evidence of a time effect on average rating.



3.8.3 Model 5 - Reg. Movie and User Effect + Time effect

Time effect along with movie effect and user effect supported by previous analysis.

We'll create a model that adds a time effect to the regularized movie and user joint model.

We define d_ui as the day for user's u rating of movie i.

The model will look like this:

$$Y_{u,i} = \mu + \ \lambda(b_i + b_{u,i}) + f(d_{u,i}) + \epsilon_{u,i}$$

f a smooth function of $d_{u,i}$, since time is continuous

- Fit the model

```
fit_time_ave <-
  train_edx %>%
  mutate(week = round_date(rate_date, unit = "week")) %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
  left_join(fit_reg_user_movie_ave, by='userId') %>%
  group_by(week) %>%
  summarize(d_ui=mean(rating-mu-reg_b_i-reg_b_ui))
```

- Predict the ratings

```
predicted_ratings <-
  test_edx %>%
  mutate(week = round_date(rate_date, unit = "week")) %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
  left_join(fit_reg_user_movie_ave, by='userId') %>%
```

```
  left_join(fit_time_ave, by="week") %>%
  mutate(predicted = mu+reg_b_i+reg_b_ui+d_ui) %>%
  pull(predicted)
```

- Model results

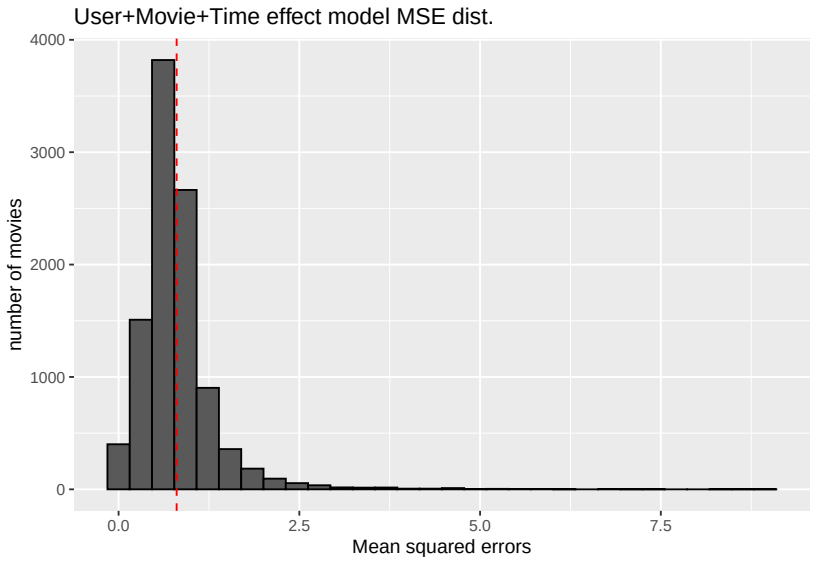
```
model_5_rmse <- RMSE(true_ratings=test_edx$rating,
                     predicted_ratings=predicted_ratings)

model_5_mse <- MSE(test_edx$rating,predicted_ratings)

#add the results to the table
model_5_results <- tibble(method = "Reg. Movie + User Effect + Time effect",
                          MSE=model_5_mse, RMSE = model_5_rmse)
model_5_results %>% knitr::kable()
```

method	MSE	RMSE
Reg. Movie + User Effect + Time effect	0.7497151	0.8658609

- The residual summary as shown

```
##  reg_squared_errors
##  Min.   : 0.00000
##  1st Qu.: 0.06375
##  Median : 0.29459
##  Mean   : 0.74972
##  3rd Qu.: 0.89537
##  Max.   :24.89020
```

- The model mean squared errors distribution



The model obtain RMSE= 0.8658609, improvement of 18.31501% from the naive RMSE and just 0.01% improvement from the regularized movie and user effect model.

The time effect is barely significant.

3.9 Genres

3.9.1 Genres Overview

Movies are classified to different genres. Some genres are very popular and contain large amount of movies and some are barely heard of.

In addition, different users tend to favor certain genres over others and target their watching and rating habits toward them.

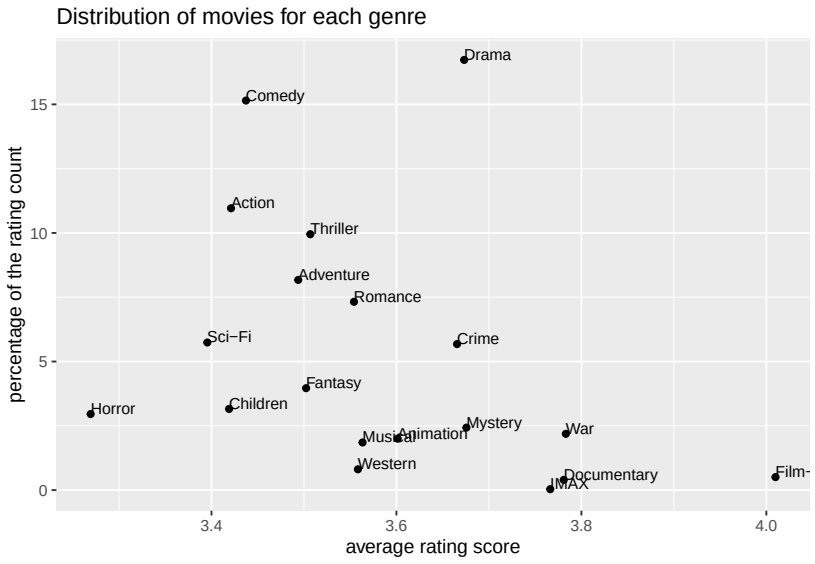
The genres column include every genre that apllies to the movie. Some movies fall under several genres as folowing;

```
##                    title                              genres
## 1: Sleepless in Seattle                Comedy|Drama|Romance
## 2:   Dances with Wolves              Adventure|Drama|Western
## 3:  Mission: Impossible  Action|Adventure|Mystery|Thriller
## 4:         Multiplicity                              Comedy
```

We'll separate the edx train and test sets rows for unique genres. Movie with more than one genre will split into several rows, each row represents a unique genre.

```
train_edx_genres <- train_edx %>%  separate_rows(genres, sep="\\|")
test_edx_genres <- test_edx %>%  separate_rows(genres, sep="\\|")
```

The distribution of the genres by their total ratings and total average score as follow;



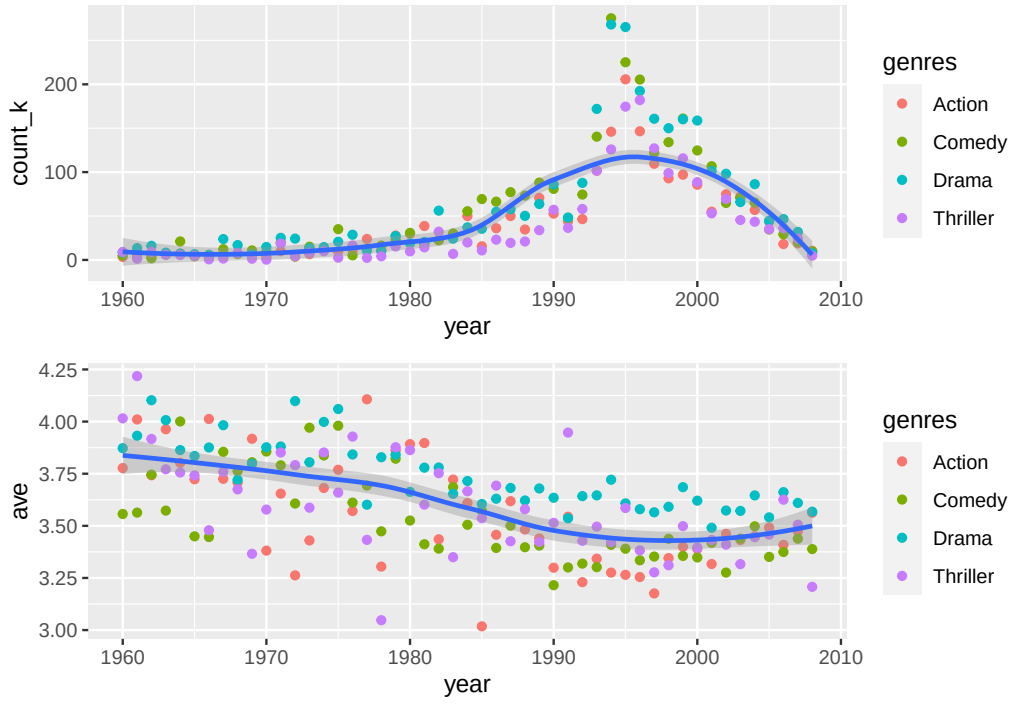
There are 19 uniqe genres.

Drama genre leads with 17% of total ratings and the highst average score.

The genres with average score over 3.4 stars and on top 20 quantile of total ratings are the following;

genres	count_m	ave_rating	percentage
Drama	3.13	3.67	17
Comedy	2.83	3.44	15
Action	2.05	3.42	11
Thriller	1.86	3.51	10

We can see that those 4 genres has changed over the years, but in general their ratings respectivly increases until the mid 90's and their average rate score decreases constantly.



The analysis shows evidance of genre effect. We'll add the effect to our previous model.

3.9.2 Model 6 - Reg. Movie + User Effect + Time Effect + Genres Effect

The model will look like this:

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

$x_{u,i}^k = 1$ if $g_{u,i}$ is genre k

We simply define b_g as the unique genre effect (average rating of the genre) correspondent with movie i

- Fit the model

```
fit_genres_ave <-
  train_edx_genres %>%
  mutate(week = round_date(rate_date, unit = "week")) %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
```

```
  left_join(fit_reg_user_movie_ave, by='userId') %>%
  left_join(fit_time_ave, by="week") %>%
  group_by(genres) %>%
  summarize(b_g=mean(rating-mu-reg_b_i-reg_b_ui-d_ui))
```

- Predict the ratings

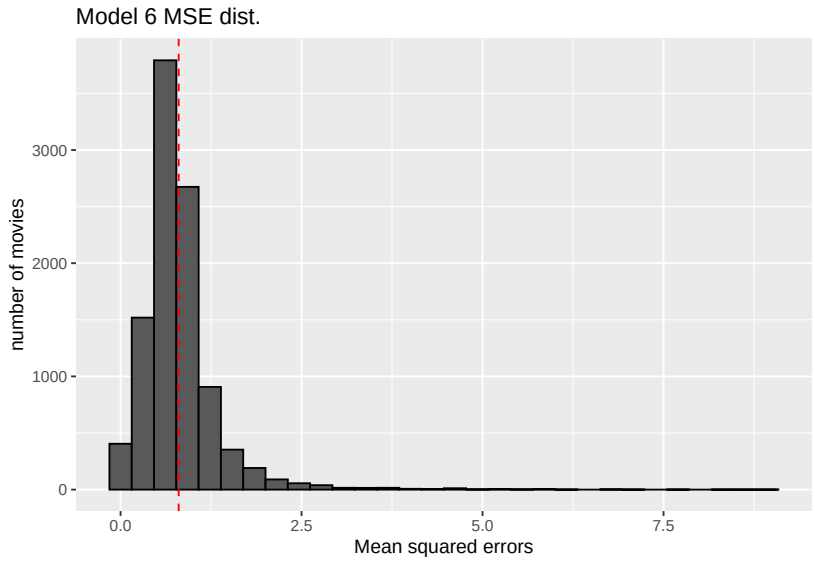
```
predicted_ratings <-
  test_edx_genres %>%
  mutate(week = round_date(rate_date, unit = "week")) %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
  left_join(fit_reg_user_movie_ave, by='userId') %>%
  left_join(fit_time_ave, by="week") %>%
  left_join(fit_genres_ave, by='genres') %>%
  mutate(predicted = mu+reg_b_i+reg_b_ui+d_ui+b_g) %>%
  pull(predicted)
```

- The model results

```
model_6_rmse <- RMSE(true_ratings=test_edx_genres$rating,
                     predicted_ratings=predicted_ratings)

model_6_mse <- MSE(test_edx_genres$rating,predicted_ratings)

#add the results to the table
model_6_results <-
  tibble(method = "Reg. Movie + User Effect + Time Effect + Genres Effect",
         MSE=model_6_mse, RMSE = model_6_rmse)
model_6_results %>% knitr::kable()
```

method	MSE	RMSE
Reg. Movie + User Effect + Time Effect + Genres Effect	0.7466395	0.864083

- The model mean squared error distribution



We have reached our final goal, and all the features got included in the model which indicates good correlation between them. We'll examine the model on the validation set.

4 Test On Validation

Following is a summary of all models and their RMSE, and the most accurate one is the one that took into account all the features:

Reg. Movie + User Effect + Time Effect + Genres Effect

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

method	MSE	RMSE
Reg. Movie + User Effect + Time Effect + Genres Effect	0.7466395	0.8640830
Reg. Movie + User Effect + Time effect	0.7497151	0.8658609
Reg. Movie + User Effect	0.7498952	0.8659649
Movie + User Effect	0.7510663	0.8666408
Reg. Movie Effect	0.8909883	0.9439218
Movie Effect	0.8911112	0.9439868
Reg. User + Age of the movie at rating Effect	0.9433008	0.9712367
Reg. User Effect	0.9575560	0.9785479
User Effect	0.9587157	0.9791403
Average only	1.1247902	1.0605613

4.1 Data Preparation

```
test_validation <-
  validation %>%
  mutate (rate_date = date(as_datetime(timestamp)),
```

```
          release_year =as.numeric(str_extract(title,"(?<=\\()(\\d{4})(?=\\))")),
          title= str_remove(as.character(title), "(\\(\\d{4}\\))")) %>%
  select(-timestamp) %>%
  separate_rows(genres, sep="\\|")
```

4.2 Final Model Examination

- Fit the model

```
fit_genre_ave <-
  train_edx_genres %>%
  mutate(week = round_date(rate_date, unit = "week")) %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
  left_join(fit_reg_user_movie_ave, by='userId') %>%
  left_join(fit_time_ave, by="week") %>%
  group_by(genres) %>%
  summarize(b_g=mean(rating-mu-reg_b_i-reg_b_ui-d_ui))
```

- Predict the ratings

```
predicted_ratings <-
  test_validation %>%
  mutate(week = round_date(rate_date, unit = "week")) %>%
  left_join(fit_reg_movie_ave, by='movieId') %>%
  left_join(fit_reg_user_movie_ave, by='userId') %>%
  left_join(fit_time_ave, by="week") %>%
  left_join(fit_genres_ave, by='genres') %>%
  mutate(predicted = mu+reg_b_i+reg_b_ui+d_ui+b_g) %>%
  pull(predicted)
```

- Final model results

```
model_final_rmse <- RMSE(true_ratings=test_validation$rating,
                         predicted_ratings=predicted_ratings)
model_final_mse <- MSE(test_validation$rating,predicted_ratings)

#add the results to the table
model_final_results <-
  tibble(method = "final",
         MSE=model_final_mse, RMSE = model_final_rmse)
model_final_results %>% knitr::kable()
```

method	MSE	RMSE
final	0.7452454	0.8632759

5 Conclusions

The test validation showed even better results than the results we obtained while training the model, which means that the algorithm can act standalone in the future.

- Future work: we can further add more personalized info on the user, such as age, sex, origin in order to make the model even more accurate.
- Limitations: the dataset size impacted our ability to try and fit more targeted models such as KNN and Matrix Factorization, simply because our peronal computer couldn't handle the load. From the same reasons, it was also challenging to run statistical queries on our current model. Running these in a cloud computing environment would simplify this process.