DS09_Capstone_Project01_MC

Marc Camprodon

2023-11-24

Project_01. INDEX.

Project_01. IMPLEMENTATION.

1.INTRODUCTION.

We want to build a recommendation system, for movies, through being able to predict movie ratings for users. Before we build our recommendation system, we'll work on understanding the available data, through data exploration and visualization.

The data source of our study is the provided 'movielens' data frame. Let's prepare it (according to instructions):

Create edx and final_holdout_test sets:

Note: this process could take a couple of minutes

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

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

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   1.0.0
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.5.0
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

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

```
library(tidyverse)
library(caret)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

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

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

2. DATA exploration and visualization.

On structure of the "movilens" data.frame.

```
str(movielens)
```

```
## 'data.frame':    10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

Movielens has 10000054 observations of 6 variables (3 integers, 2 character, 1 numeric (rating)) .

On structure of the "edx" data.frame:

```
head(edx)
```

```
##   userId movieId rating timestamp                          title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

The edx data.frame has 9000047 observations of (the same movilens) 6 variables.

Let's find out how many unique values are per variable in edx:

```
variables <- colnames(edx)

nr_values_vars <-edx %>% summarize(
    nr_users=n_distinct(userId),
    nr_movies=n_distinct(movieId),
    nr_ratings=n_distinct(rating),
    nr_timestamps=n_distinct(timestamp),
    nr_titles=n_distinct(title),
    nr_genres=n_distinct(genres)
    )
nr_values_vars
```

```
##   nr_users nr_movies nr_ratings nr_timestamps nr_titles nr_genres
## 1    69878     10677         10       6519590     10676       797
```

```
print(nr_values_vars)
```

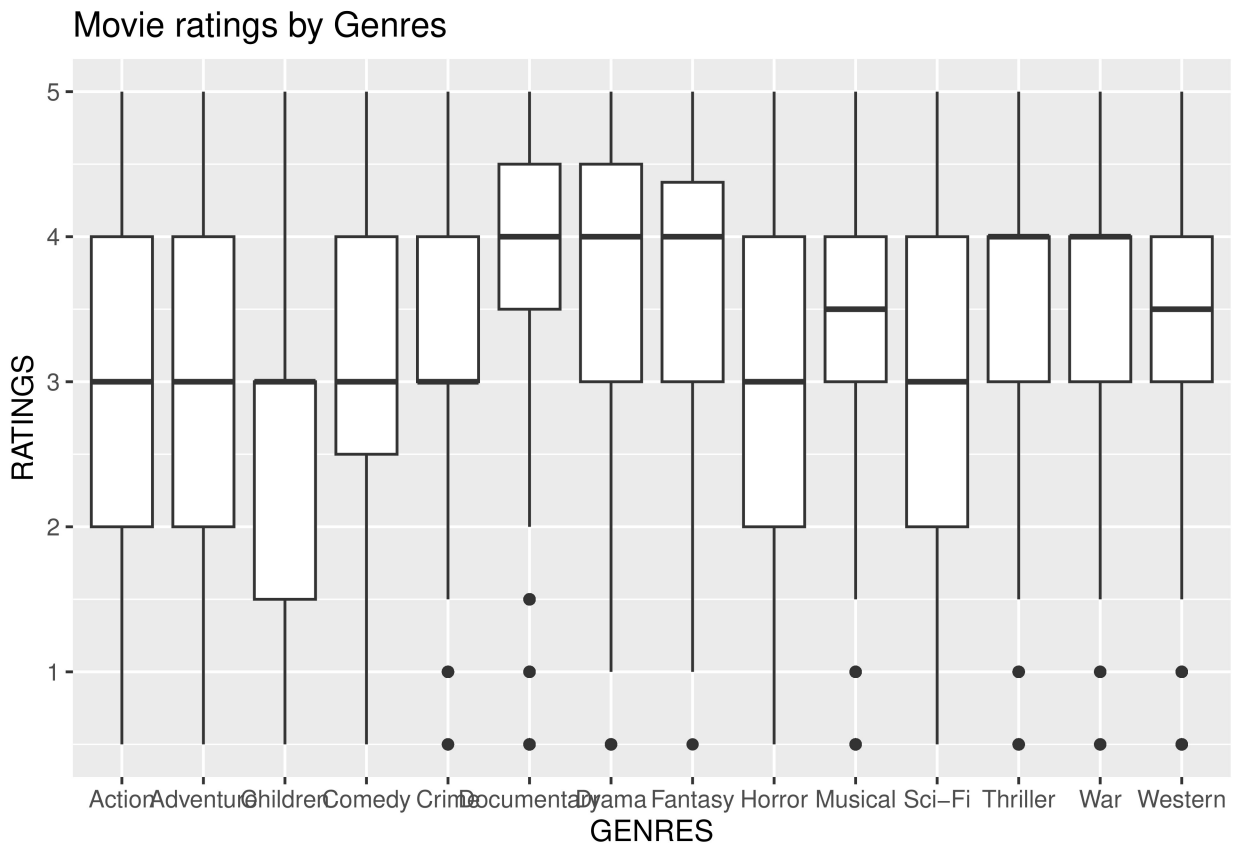
```
##   nr_users nr_movies nr_ratings nr_timestamps nr_titles nr_genres
## 1    69878     10677         10       6519590     10676       797
```

Data is Tidy, ready for exploration and model building. We don't need to (further) data wrangling.

Notes: -'movieId' and 'title' are associated, as each title is unique (nr_movies should be the same as nr_titles). -The combinations of user-movie should produce the population of ratings being reviewed. It would also be interesting knowing how the 'genres' variable affects ratings, in combination with 'users'.

-A quick review on a few generic 'genres', might provide some insights on 'genres' significance:

```
edx %>%
  filter(genres %in% c("Action","Adventure","Children","Comedy","Crime","Documentary","Drama","Fantasy"
  ggplot(aes(genres, rating)) +
  geom_boxplot()+
  xlab("GENRES") +
  ylab("RATINGS")+
  ggtitle("Movie ratings by Genres")
```

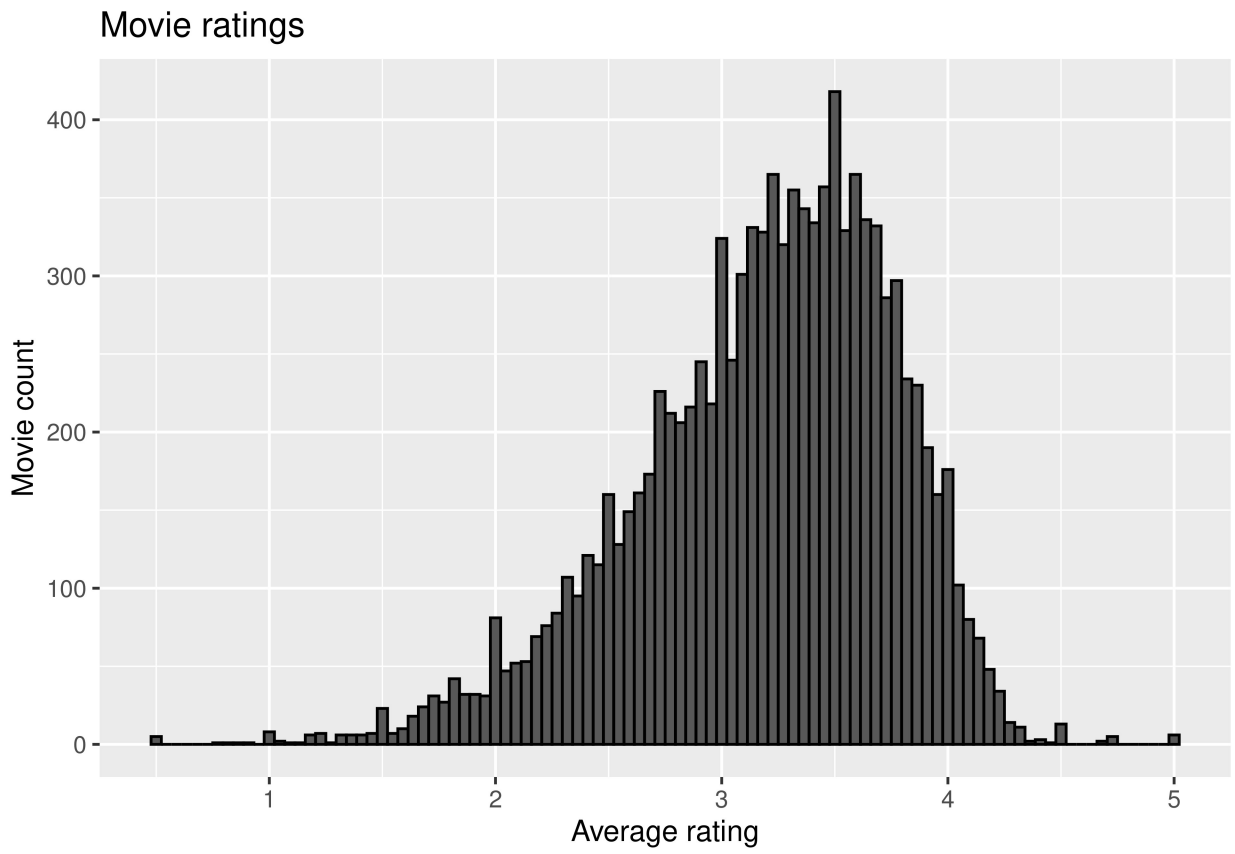


-We can observe that the distribution of a few genres-among the selected for the plot (Documentary, Dama, Fantasy) have their median rating value on "4".Whereas 5 genres are centered around 3, all with quartiles in 2 and 4, with range from about 0 to 5. -The 'Children' genre's ratings distribution shows the worst results among the selected genres. -Genre and ratings are related. The relationship is significant (at sight), versus rating averages, for a few genres, 'Documentary', 'Drama', 'Fantasy' (positive effect of genre on ratings) and also for 'Children'(negative effect).

Let's now work on a visual representation of 'movieID' and 'users', vs 'ratings'. Building distributions of ratings by movie (1 to 5), and by user (1 to 5):

-By movie:

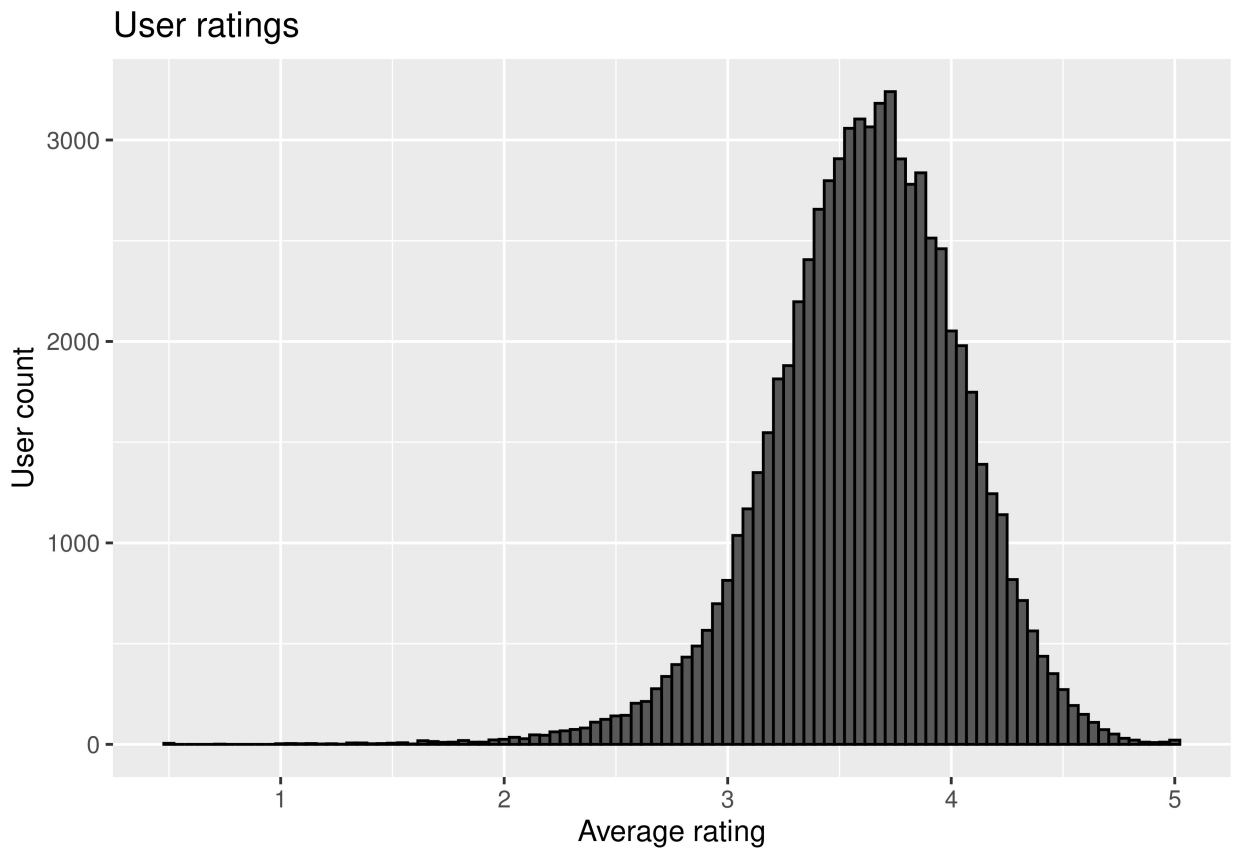
```
edx %>%
  group_by(movieId) %>%
  summarize(avg = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(avg)) +
  geom_histogram(bins = 100, color = "black")+
  xlab("Average rating") +
  ylab("Movie count") +
  ggtitle("Movie ratings")
```



We find that the most repeated mean-rating for movies is 3.5, with over 400 movies averaging 3.5 rating. Not too many movies average is over 4, around 100 movies average a 4 rate, only some 20movies average 4.5, and around 10 average 5. Not many movies average ratings below 2.

-By user:

```
edx %>%
  group_by(userId) %>%
  summarize(avg = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(avg)) +
  geom_histogram(bins = 100, color = "black")+
  xlab("Average rating") +
  ylab("User count") +
  ggtitle("User ratings")
```



Average ratings by user show a quite 'normal distribution', centered around a rating average of 3.75.

These plots help understand that both userId and movieId have an effect on a movie rating. Ratings distribution for different users is different, also different movies are differently rated (dispersion on avg rating per movie).

3.MODELING.

Building the Recommendation System. We'll use ratings given by users to movies, to predict ratings. We will create a simple model -algorithm- which we will improve through a sequence of steps (successive models). Using a training set (edx) and a test set (temp) to assess the accuracy of the models. The RMSE function (residual mean square error) will help us evaluate model accuracy.By calculating the error made between prediction of ratings, and true ratings.

Our first model(1) will consider the same rating for any movie and user:

Model [#1].Same rating for any movie and user.

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

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

Following we'll define the RMSE function: RMSE <- function(true_ratings, predicted_ratings){ sqrt(mean((true_ratings - predicted_ratings)^2)) } Then we'll calculate the 'rmse' for this first recommendation morel:

```
naive_rmse <- RMSE(temp$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061205
```

We'll present results for each model on a table, model - rmse: data.frame with method & RMSE:

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

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## i Please use `tibble()` instead.
```

```
rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

We will now start improving the model:

Model [#2]. Modeling movie effects:

```
mu_hat_2 <- mean(edx$rating)
mu_hat_2
```

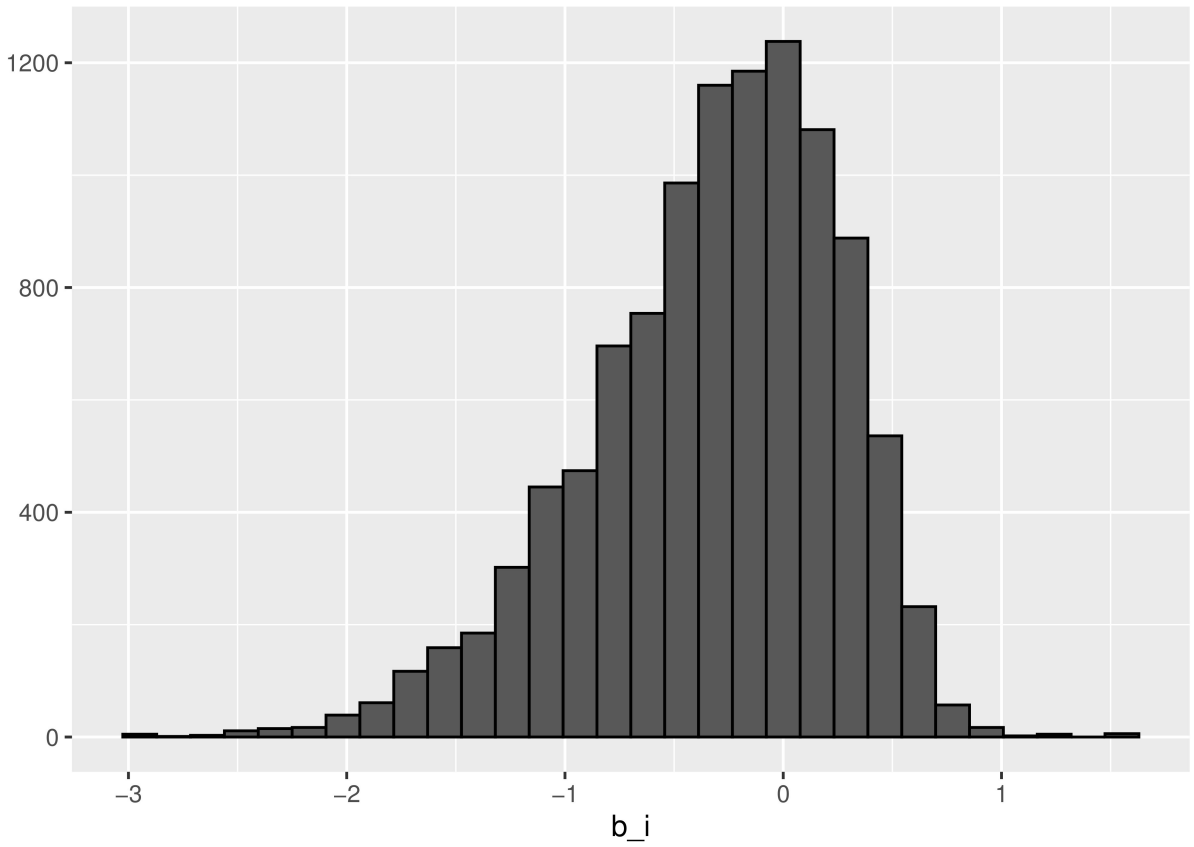
```
## [1] 3.512465
```

```
movie_avgs <- edx %>%
   group_by(movieId) %>%
   summarize(b_i = mean(rating - mu_hat_2))
head(movie_avgs)
```

```
## # A tibble: 6 x 2
##   movieId    b_i
##     <int>  <dbl>
## 1       1  0.415
## 2       2 -0.307
## 3       3 -0.365
## 4       4 -0.648
## 5       5 -0.444
## 6       6  0.303
```

```
movie_avgs %>%
   qplot(b_i, geom ="histogram", bins = 30, data = ., color = I("black"))
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
```




```
predicted_ratings <- mu_hat_2 + temp %>%
   left_join(movie_avgs, by='movieId') %>%
   pull(b_i)
head(predicted_ratings)
```

```
## [1] 2.935121 3.663522 3.055652 3.530058 4.415366 2.945278
```

```
pred_rat_2 <-replace_na(predicted_ratings, mu_hat_2)
head(pred_rat_2)
```

```
## [1] 2.935121 3.663522 3.055652 3.530058 4.415366 2.945278
```

```
model_2_rmse <- RMSE(pred_rat_2, temp$rating)
model_2_rmse
```

```
## [1] 0.9439049
```

```
rmse_results <- bind_rows(rmse_results,
                data_frame(method="Movie Effect Model",
                RMSE = model_2_rmse ))
rmse_results
```

```
## # A tibble: 2 x 2
##   method              RMSE
##   <chr>              <dbl>
## 1 Just the average   1.06
## 2 Movie Effect Model 0.944
```

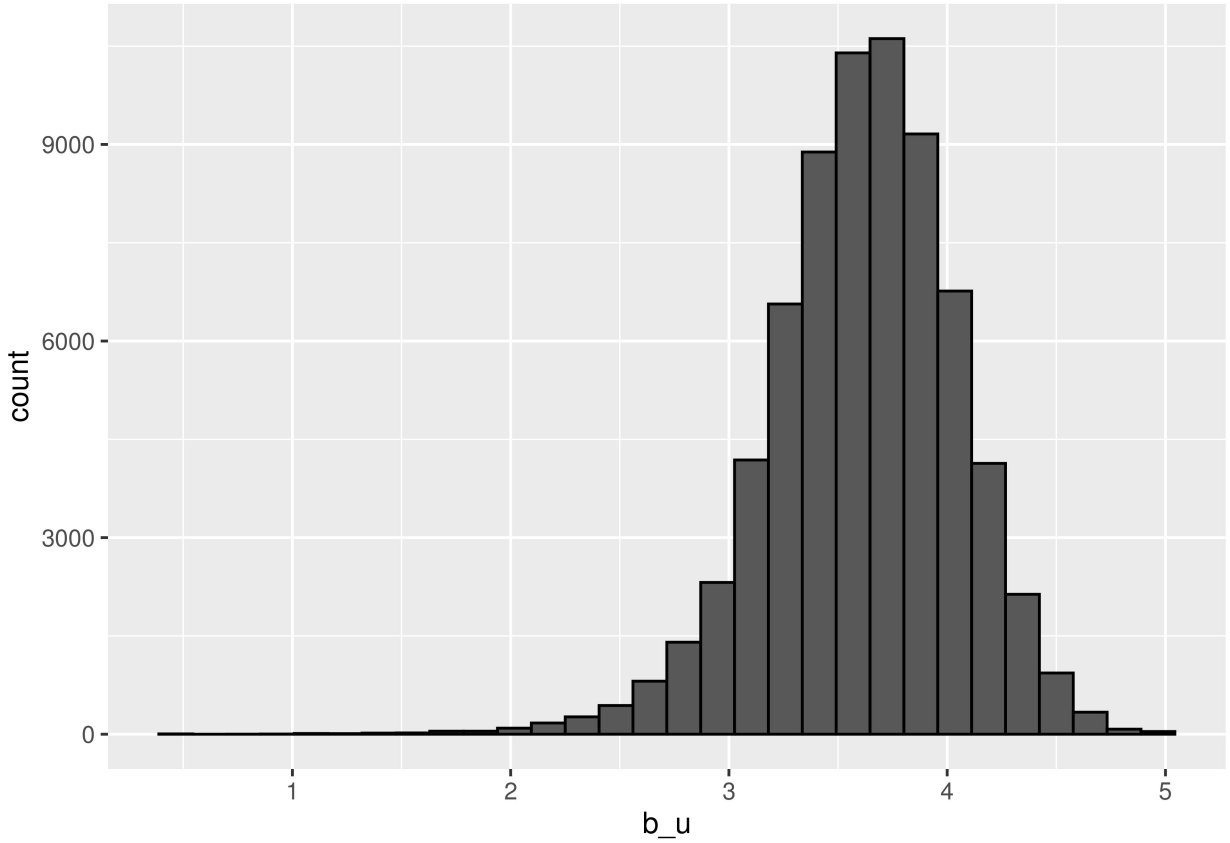
```
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612048
Movie Effect Model	0.9439049

Further to modeling Movie effects, we will add user effect variable to the model.

Model [#3]. Modeling User effects.

```
edx %>%
   group_by(userId) %>%
   summarize(b_u = mean(rating)) %>%
   filter(n()>=100) %>%
   ggplot(aes(b_u)) +
   geom_histogram(bins = 30, color = "black")
```




```
   #ok, graphic shows distribution around b_u of ca.3,6

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

```
## # A tibble: 69,878 x 2
##    userId     b_u
##     <int>   <dbl>
##  1      1  1.68
##  2      2 -0.236
##  3      3  0.264
##  4      4  0.652
##  5      5  0.0853
##  6      6  0.346
##  7      7  0.0238
##  8      8  0.203
##  9      9  0.232
## 10     10  0.0833
## # ... with 69,868 more rows
```

```
predicted_ratings <- temp %>%
   left_join(movie_avgs, by='movieId') %>%
```

```
   left_join(user_avgs, by='userId') %>%
   mutate(pred = mu_hat + b_i + b_u) %>%
   pull(pred)
head(predicted_ratings)
```

```
## [1] 4.614356 5.342757 4.734887 3.293650 4.178957 2.708870
```

```
              #Removing any na's (if any)
              pred_rat_3 <-replace_na(predicted_ratings, mu_hat_2)
              head(pred_rat_3)
```

```
## [1] 4.614356 5.342757 4.734887 3.293650 4.178957 2.708870
```

```
model_3_rmse <- RMSE(pred_rat_3, temp$rating)
model_3_rmse
```

```
## [1] 0.8653458
```

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

method	RMSE
Just the average	1.0612048
Movie Effect Model	0.9439049
Movie + User Effects Model	0.8653458

By considering the movie and user effect in our model, we have improved model accuracy, now well below the 1.0 value (0.8653458«1). We could further improve model accuracy, by using regularization, removing effects of noisy estimates (penalizing large estimates coming from small sample sizes).

Model [#4]. Regularization on the Movie Effects model:

Regularization should help us penalize large estimates derived from small samples. Thus, soften 'anomalies' influence. Computing regularized estimates:

```
lambda <- 3.2
mu <- mean(edx$rating)
movie_reg_avgs <- edx %>%
   group_by(movieId) %>%
   summarize(b_i=sum(rating-mu)/(n()+lambda),n_i=n())

pred_rat_4 <- temp %>%
   left_join(movie_reg_avgs, by = "movieId") %>%
   mutate(pred=mu + b_i) %>%
   pull(pred)
head(pred_rat_4)
```

```
## [1] 2.935236 3.663505 3.055758 3.530051 4.415203 2.945526
```

```
model_4_rmse <- RMSE(pred_rat_4, temp$rating)
model_4_rmse
```

```
## [1] 0.943856
```

```
          sum(is.na(pred_rat_4)) #[1] 0
```

```
## [1] 0
```

```
          #Removing na's in any:
          pred_rat_4na <-replace_na(pred_rat_4, mu_hat_2)
          head(pred_rat_4na)
```

```
## [1] 2.935236 3.663505 3.055758 3.530051 4.415203 2.945526
```

```
#RMSE(predicted_ratings,temp$rating)
model_4_rmse <- RMSE(pred_rat_4na, temp$rating)
model_4_rmse
```

```
## [1] 0.943856
```

```
rmse_results <- bind_rows(rmse_results,
                data_frame(method="Regularized Movie Effect Model",
                RMSE = model_4_rmse ))
rmse_results %>% knitr::kable()
```

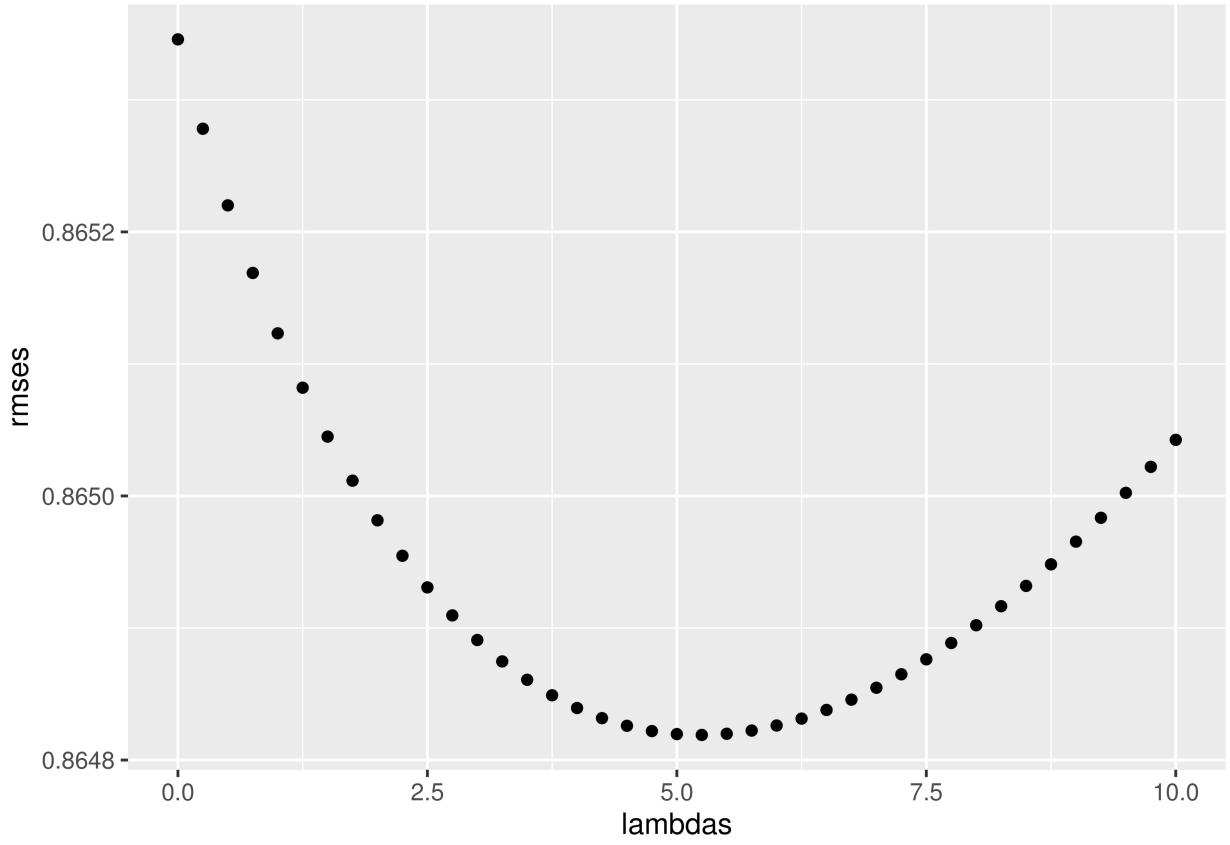
method	RMSE
Just the average	1.0612048
Movie Effect Model	0.9439049
Movie + User Effects Model	0.8653458
Regularized Movie Effect Model	0.9438560

Model [#5]. Regularization on the Movie + User Effects model:

Penalized least squares, constraining variability of effect sizes. This time we'll use cross-validation for choosing an optimal lambda:

```
lambdas <- seq(0, 10, 0.25)
    #Check for na values:
    #sum(is.na(edx$rating)) #[1] 0
    #sum(is.na(temp$rating)) #[1] 0

rmses <- sapply(lambdas, function(l){
    mu <- mean(edx$rating)
    b_i <- edx %>%
          group_by(movieId) %>%
```

```
        summarize(b_i = sum(rating - mu)/(n()+l))
    b_u <- edx %>%
        left_join(b_i, by="movieId") %>%
        group_by(userId) %>%
        summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <- temp %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        mutate(pred = mu + b_i + b_u) %>%
        pull(pred)
        return(RMSE(predicted_ratings, temp$rating))
    })

qplot(lambdas, rmses)
```




```
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 5.25
```

Now we calculate rmse applying lambda = 5.25 we found minimizes rmse:

```
lambda<-5.25
rmse_5 <- sapply(lambda, function(l){
```

```
        mu <- mean(edx$rating)
        b_i <- edx %>%
              group_by(movieId) %>%
              summarize(b_i = sum(rating - mu)/(n()+l))
        b_u <- edx %>%
              left_join(b_i, by="movieId") %>%
              group_by(userId) %>%
              summarize(b_u = sum(rating - b_i - mu)/(n()+l))
        predicted_ratings <- temp %>%
              left_join(b_i, by = "movieId") %>%
              left_join(b_u, by = "userId") %>%
              mutate(pred = mu + b_i + b_u) %>%
              pull(pred)
        return(RMSE(predicted_ratings, temp$rating))
       })
rmse_5
```

```
## [1] 0.864819
```

```
rmse_results <- bind_rows(rmse_results,
                data_frame(method="Regularized Movie+User Effect Model",
                RMSE = rmse_5 ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612048
Movie Effect Model	0.9439049
Movie + User Effects Model	0.8653458
Regularized Movie Effect Model	0.9438560
Regularized Movie+User Effect Model	0.8648190

Final model applied to the holdout_test.

Model [#6].FINAL _ Regularization on the Movie + User Effects model.

We'll now apply our latest MODEL using the final_holdout_test:

Make sure userId and movieId in final hold-out test set are also in edx set.

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

  #we'll briefly review the 'final_holdout_test' structure:
  str(final_holdout_test)
```

```
## 'data.frame':    1000007 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : int  231 480 586 151 858 1544 590 4995 34 432 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
```

```
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Roman
```

```
  head(final_holdout_test)
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                  title
## 1                                  Dumb & Dumber (1994)
## 2                                  Jurassic Park (1993)
## 3                                     Home Alone (1990)
## 4                                        Rob Roy (1995)
## 5                                  Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                  genres
## 1                                Comedy
## 2        Action|Adventure|Sci-Fi|Thriller
## 3                          Children|Comedy
## 4                Action|Drama|Romance|War
## 5                             Crime|Drama
## 6 Action|Adventure|Horror|Sci-Fi|Thriller
```

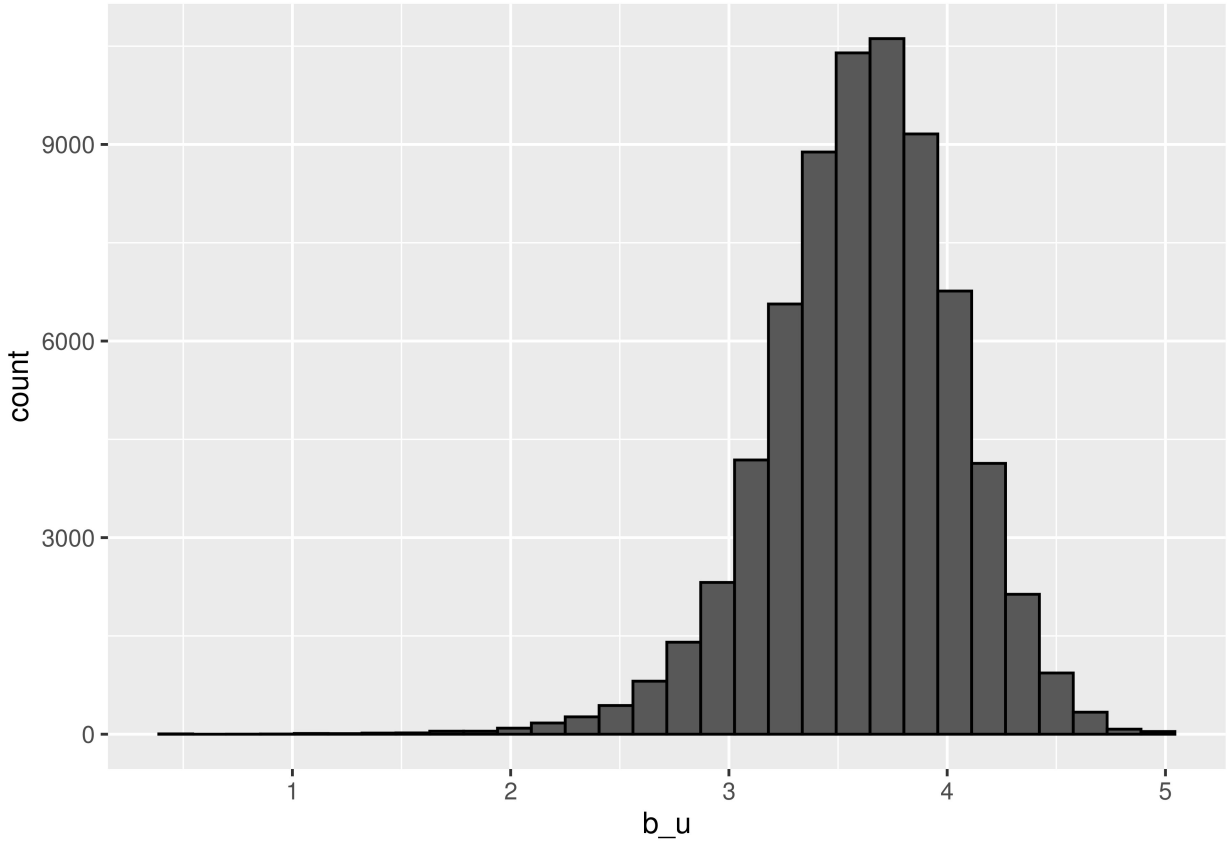
```
# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

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

Model:

```
edx %>%
   group_by(userId) %>%
   summarize(b_u = mean(rating)) %>%
   filter(n()>=100) %>%
   ggplot(aes(b_u)) +
   geom_histogram(bins = 30, color = "black")
```




```
  #The graphic shows distribution around b_u of ca.3,6

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

```
## # A tibble: 69,878 x 2
##    userId     b_u
##     <int>   <dbl>
##  1      1  1.68
##  2      2 -0.236
##  3      3  0.264
##  4      4  0.652
##  5      5  0.0853
##  6      6  0.346
##  7      7  0.0238
##  8      8  0.203
##  9      9  0.232
## 10     10  0.0833
## # ... with 69,868 more rows
```

```
predicted_ratings <- final_holdout_test %>%
            left_join(movie_avgs, by='movieId') %>%
```

```
            left_join(user_avgs, by='userId') %>%
            mutate(pred = mu_hat + b_i + b_u) %>%
            pull(pred)
head(predicted_ratings)
```

```
## [1] 4.614356 5.342757 4.734887 3.293650 4.178957 2.708870
```

```
model_f_rmse <- RMSE(predicted_ratings, final_holdout_test$rating)
model_f_rmse
```

```
## [1] 0.8653458
```

```
          #Removing any na's:
          pred_rat_f <-replace_na(predicted_ratings, mu_hat_2)
          head(pred_rat_f)
```

```
## [1] 4.614356 5.342757 4.734887 3.293650 4.178957 2.708870
```

```
model_f_rmse <- RMSE(pred_rat_f, final_holdout_test$rating)
model_f_rmse
```

```
## [1] 0.8653458
```

```
rmse_results <- bind_rows(rmse_results,
                data_frame(method="holdout_Movie + User Effects Model",
                RMSE = model_f_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612048
Movie Effect Model	0.9439049
Movie + User Effects Model	0.8653458
Regularized Movie Effect Model	0.9438560
Regularized Movie+User Effect Model	0.8648190
holdout_Movie + User Effects Model	0.8653458

We'll apply REGULARIZATION to this last model (Movie and User Effects Model) on the final holdout test. We'll use cross-validation for choosing a lambda:

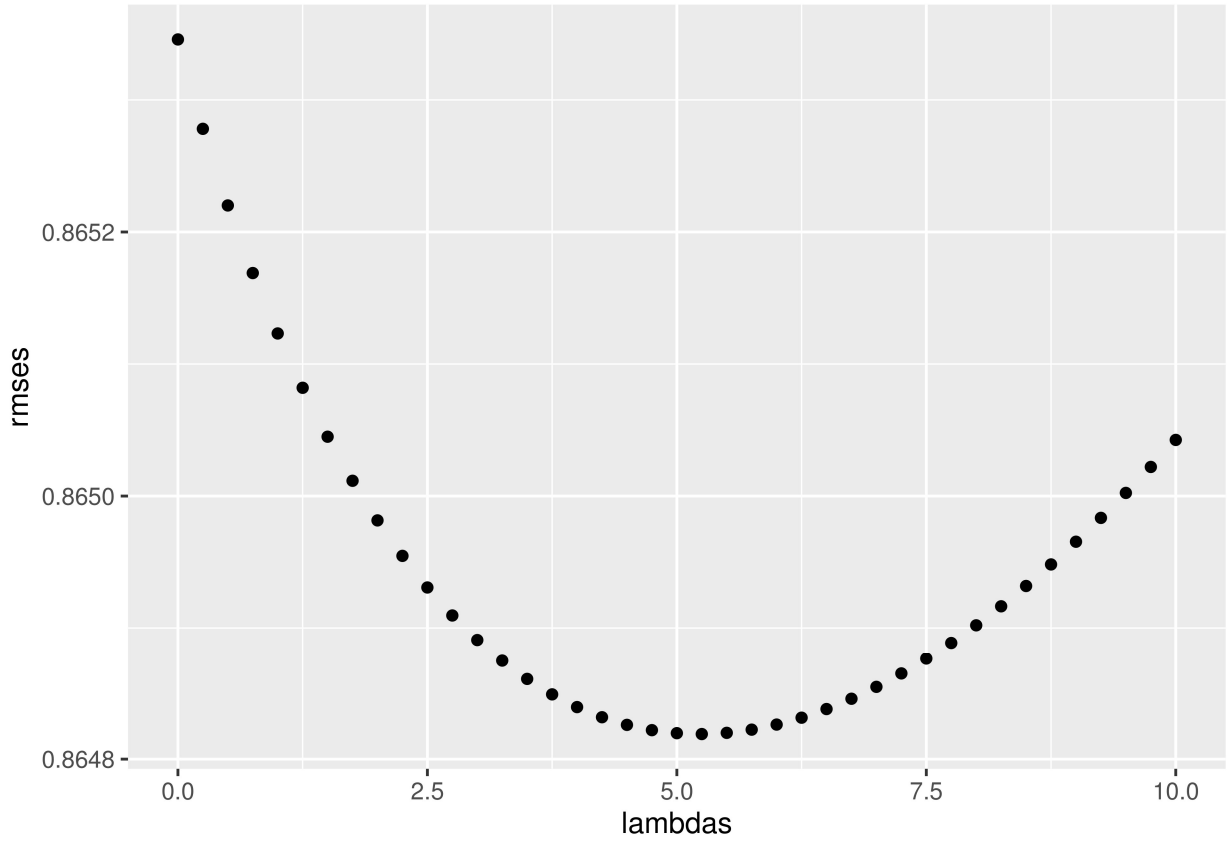
```
lambdas <- seq(0, 10, 0.25)

sum(is.na(edx$rating)) #[1] 0
```

```
## [1] 0
```

```
sum(is.na(final_holdout_test$rating)) #[1] 0
```

```
## [1] 0
```

```
rmses <- sapply(lambdas, function(l){
            mu <- mean(edx$rating)
            b_i <- edx %>%
              group_by(movieId) %>%
              summarize(b_i = sum(rating - mu)/(n()+l))
            b_u <- edx %>%
              left_join(b_i, by="movieId") %>%
              group_by(userId) %>%
              summarize(b_u = sum(rating - b_i - mu)/(n()+l))
            predicted_ratings <-
              final_holdout_test %>%
              left_join(b_i, by = "movieId") %>%
              left_join(b_u, by = "userId") %>%
              mutate(pred = mu + b_i + b_u) %>%
              pull(pred)
            return(RMSE(predicted_ratings, final_holdout_test$rating))
          })
qplot(lambdas, rmses)
```



```
lambda
```

```
## [1] 5.25
```

Applying lambda = 5.25, to calculate the minimized rmse:

```
lambda<-5.25
rmse_6 <- sapply(lambda, function(l){
            mu <- mean(edx$rating)
            b_i <- edx %>%
              group_by(movieId) %>%
              summarize(b_i = sum(rating - mu)/(n()+l))
            b_u <- edx %>%
              left_join(b_i, by="movieId") %>%
              group_by(userId) %>%
              summarize(b_u = sum(rating - b_i - mu)/(n()+l))
            predicted_ratings <-
              temp %>%
              left_join(b_i, by = "movieId") %>%
              left_join(b_u, by = "userId") %>%
              mutate(pred = mu + b_i + b_u) %>%
              pull(pred)
            return(RMSE(predicted_ratings, temp$rating))
          })

rmse_6
```

```
## [1] 0.864819
```

```
rmse_results <- bind_rows(rmse_results,
                data_frame(method="holdout_Regularized Movie+User Effect Model",
                RMSE = rmse_6 ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0612048
Movie Effect Model	0.9439049
Movie + User Effects Model	0.8653458
Regularized Movie Effect Model	0.9438560
Regularized Movie+User Effect Model	0.8648190
holdout_Movie + User Effects Model	0.8653458
holdout_Regularized Movie+User Effect Model	0.8648190

4.CONCLUSIONS, REPORT SUMMARY.

By considering User and Movie Effects, and constraining the variability of size effects through regularization, we obtain an RMSE of 0.8648190 for our final holdout test, on our latest model "holdout_regularized Movie+User Effect Model". This model provides a much improved (rmse<0.8649) estimation for ratings for a movie, by a user, than our first approximation ('Just the average (ratings)').