FinalProject_Report

Kriti Gupta

29/09/2020

1.Introduction

The purpose for this project is creating a recommender system using MovieLens dataset.

The version of movielens dataset used for this final assignment contains approximately 10 Milions of movies ratings, divided in 9 Milions for training and one Milion for validation. It is a small subset of a much larger (and famous) dataset with several millions of ratings. After a initial data exploration, the recommender systems builted on this dataset are evaluated and choosen based on the RMSE - Root Mean Squared Error that should be at least lower than **0.89999**.

#Installing essential packages

First the working envrionment is set up by installing essential packages:

Install all needed libraries if it is not present

```
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(kableExtra)) install.packages("kableExtra")
if(!require(tidyr)) install.packages("tidyr")
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(stringr)) install.packages("stringr")
if(!require(forcats)) install.packages("forcats")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(data.table)) install.packages("data.table")
```

Loading all needed libraries

```
library(dplyr)
library(tidyverse)
library(kableExtra)
library(tidyr)
library(stringr)
library(forcats)
library(ggplot2)
library(lubridate)
library(caret)
library(tinytex)
library(data.table)
```

2. Data downloading and preparation

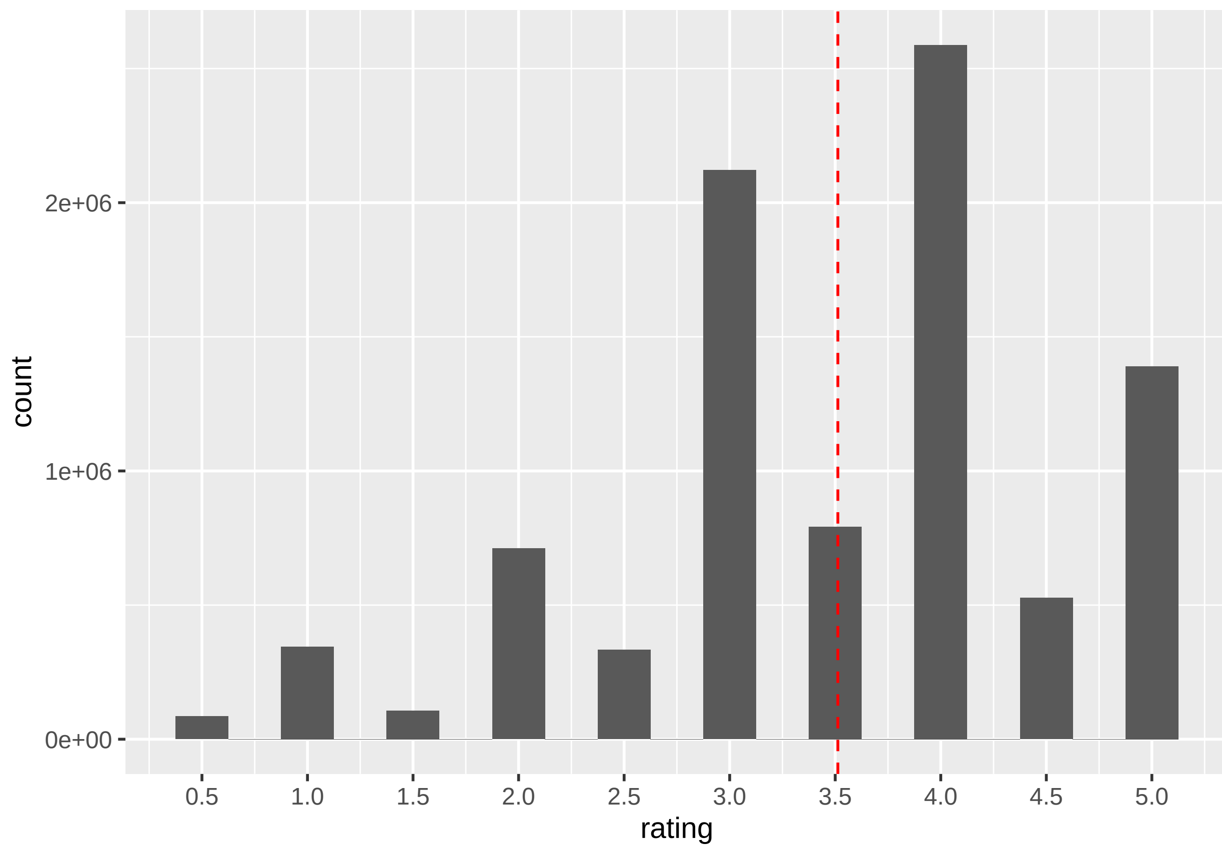
Now we split the MovieLens dataset into Training (*edx*) and Validation (*validation*) sets. The Validation set will be 10% of MovieLens data.

3.Data exploration

##3.1 Overall profile of the dataset Let's first have a general overview of the dataset:

```
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
```



We can see the overall distribution of all of the ratings. It is screwed to the right. All half stars are less frenquient than full stars. A red dased line of the overall average rating is also plotted here as a reference.

```
dim(edx) # 9000055       6
n_distinct(edx$movieId) # 10677
n_distinct(edx$title) # 10676: there might be movies of different IDs with the same title
n_distinct(edx$userId) # 69878
n_distinct(edx$movieId)*n_distinct(edx$userId) # 746087406
n_distinct(edx$movieId)*n_distinct(edx$userId)/dim(edx)[1] # 83
```

3.2 Extracting age of movies

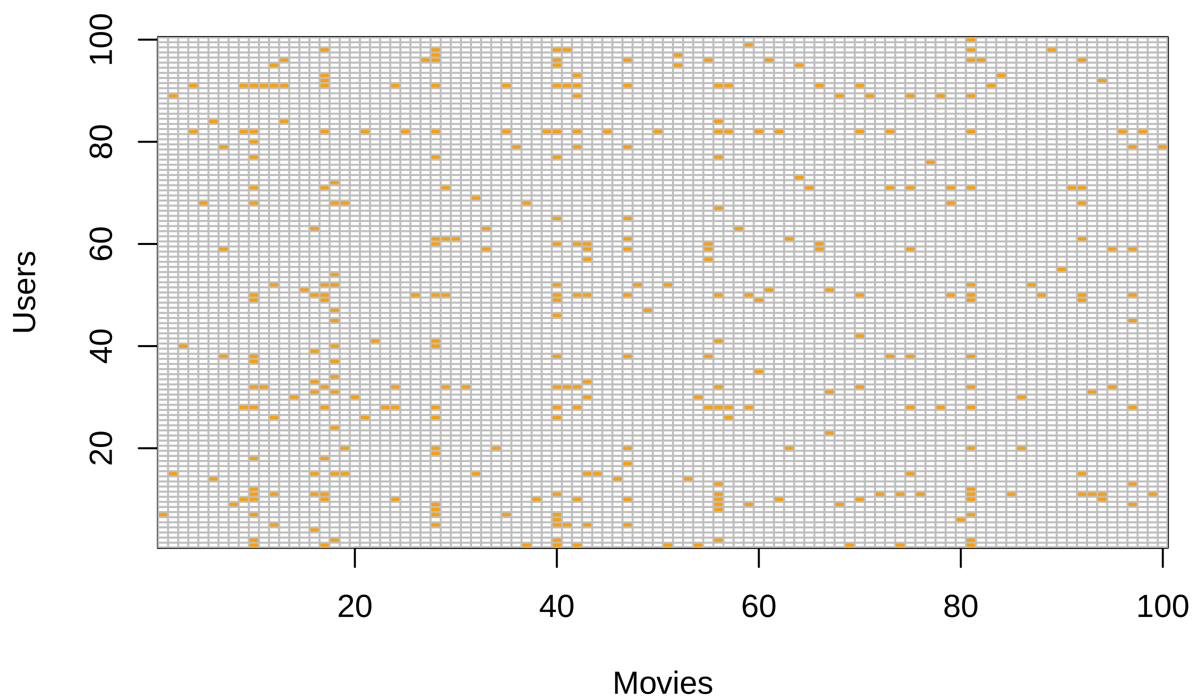
```
edx_1 <- edx %>% mutate(year_rated = year(as_datetime(timestamp)))
# extract the release year of the movie
# edx_1 has year_rated, year_released, age_at_rating, and titles without year information
edx_1 <- edx_1 %>% mutate(title = str_replace(title,"^(.+)\\s\\((\\d{4})\\)$","\\1__\\2" )) %>%
  separate(title,c("title","year_released"),"__") %>%
  select(-timestamp)
edx_1 <- edx_1 %>% mutate(age_at_rating= as.numeric(year_rated)-as.numeric(year_released))
head(edx_1)
```

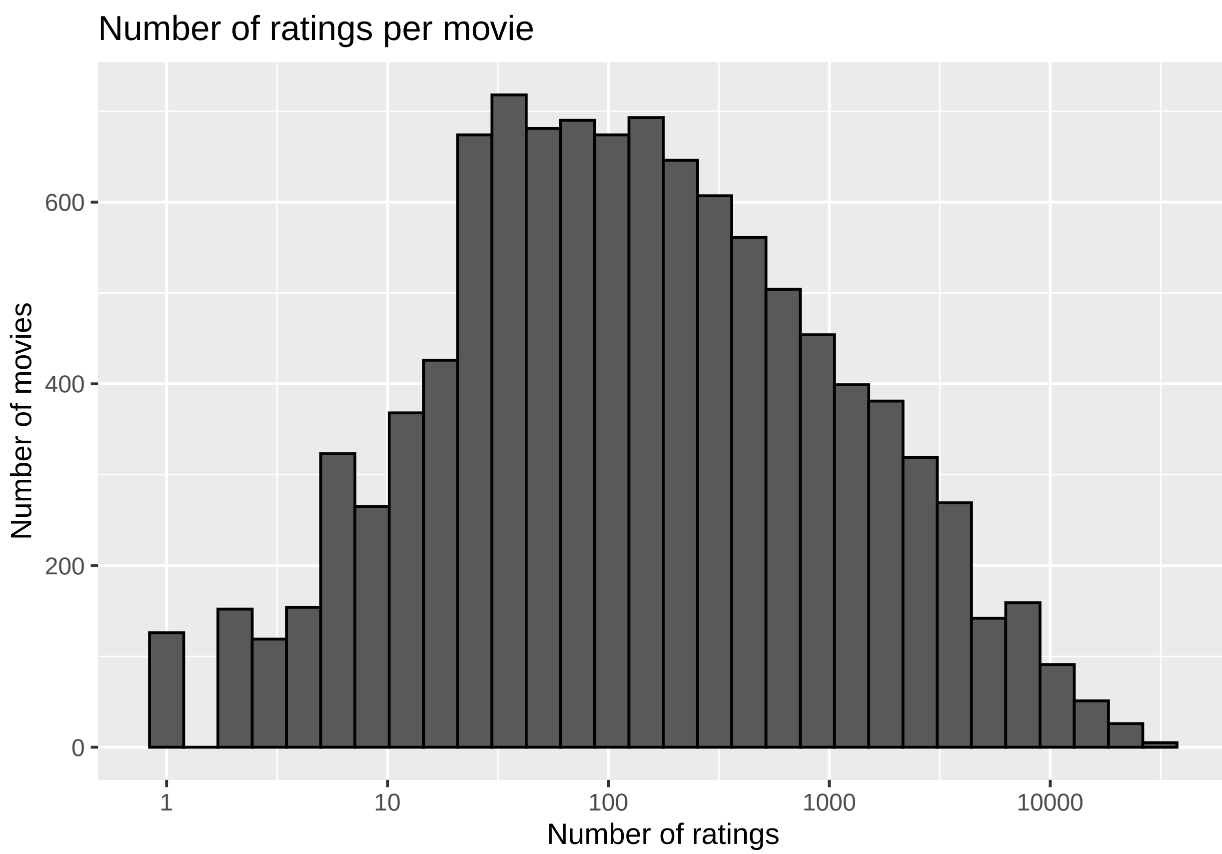
```
##   userId movieId rating                     title year_released
## 1      1     122      5               Boomerang          1992
## 2      1     185      5                Net, The          1995
```

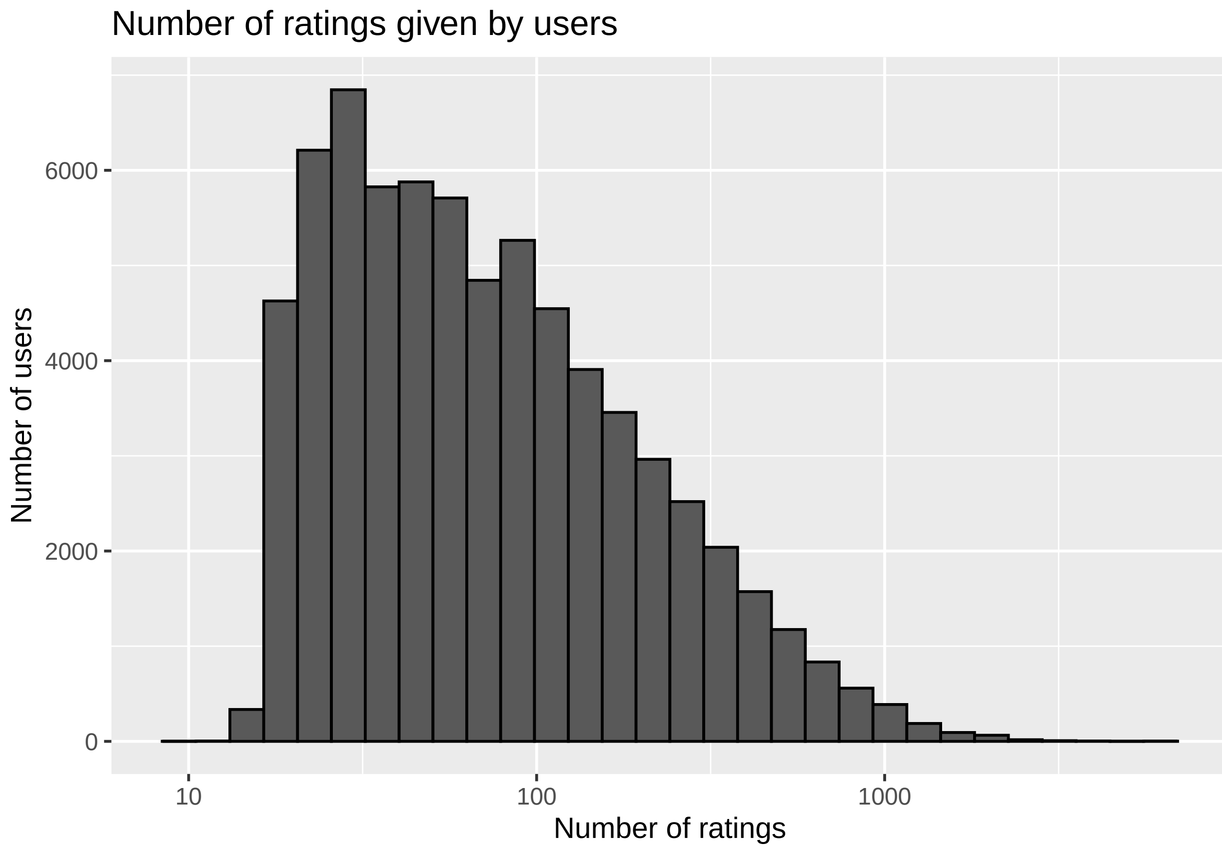
```
## 3      1     292      5               Outbreak          1995
## 4      1     316      5               Stargate          1994
## 5      1     329      5 Star Trek: Generations          1994
## 6      1     355      5       Flintstones, The          1994
##                              genres year_rated age_at_rating
## 1                    Comedy|Romance       1996             4
## 2             Action|Crime|Thriller       1996             1
## 3  Action|Drama|Sci-Fi|Thriller       1996             1
## 4        Action|Adventure|Sci-Fi       1996             2
## 5 Action|Adventure|Drama|Sci-Fi       1996             2
## 6        Children|Comedy|Fantasy       1996             2
```

3.3 Important Plots

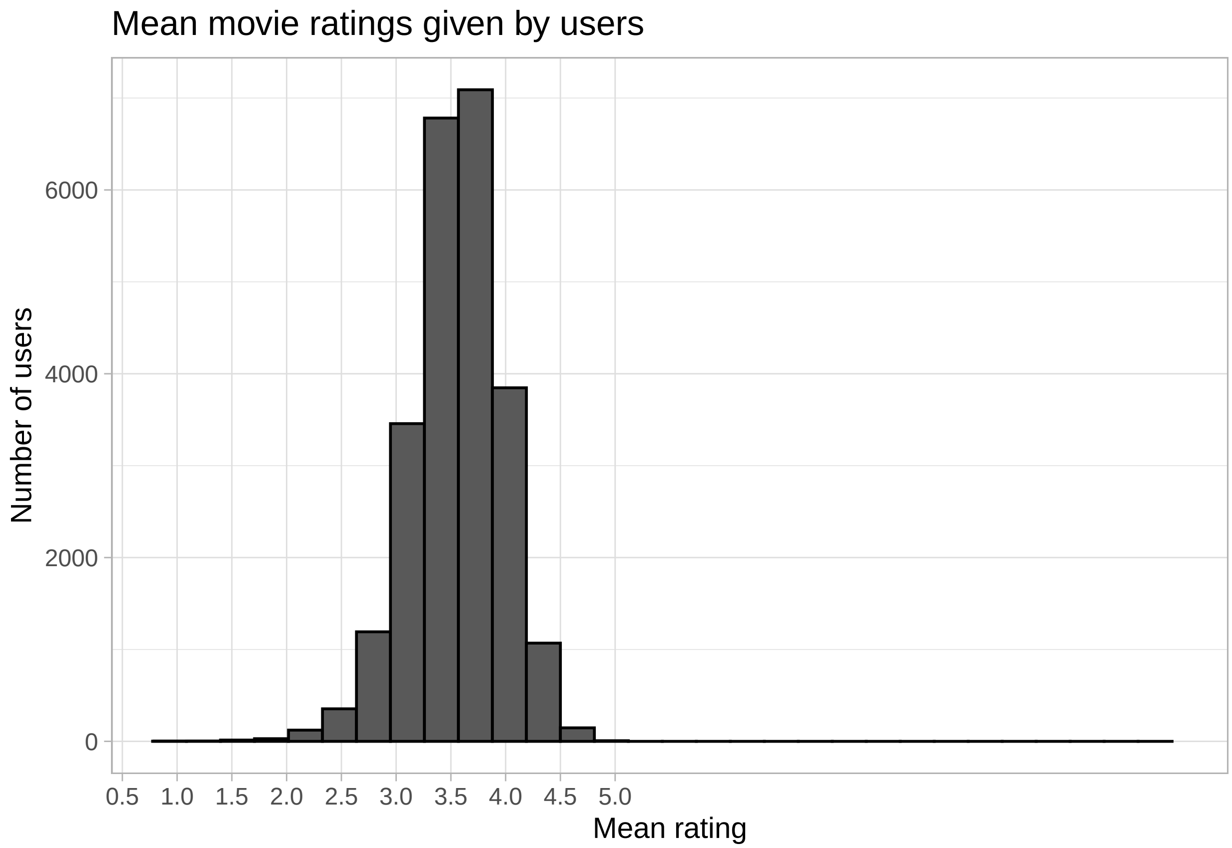
Movies vs Users - Shows sparsity




Number of ratings per movie
Number of movies
600
400
200
0
1
10
100
1000
10000
Number of ratings



Number of ratings given by users
Number of users
6000
4000
2000
0
10
100
1000
Number of ratings



4.Analysis - Model Building and Evaluation

Define RMSE: residual mean squared error

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

Model 1

Naive Mean-Baseline Model

In the first model, just based on the ratings itself, to minimize the RMSE, the best prediction of ratings for each movie will be the overall average of all ratings. The average rating is mu = 3.51247, and the naive RMSE is 1.0612.

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

```
## [1] 3.51247
```

```r
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

```
## [1] 1.0612
```

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

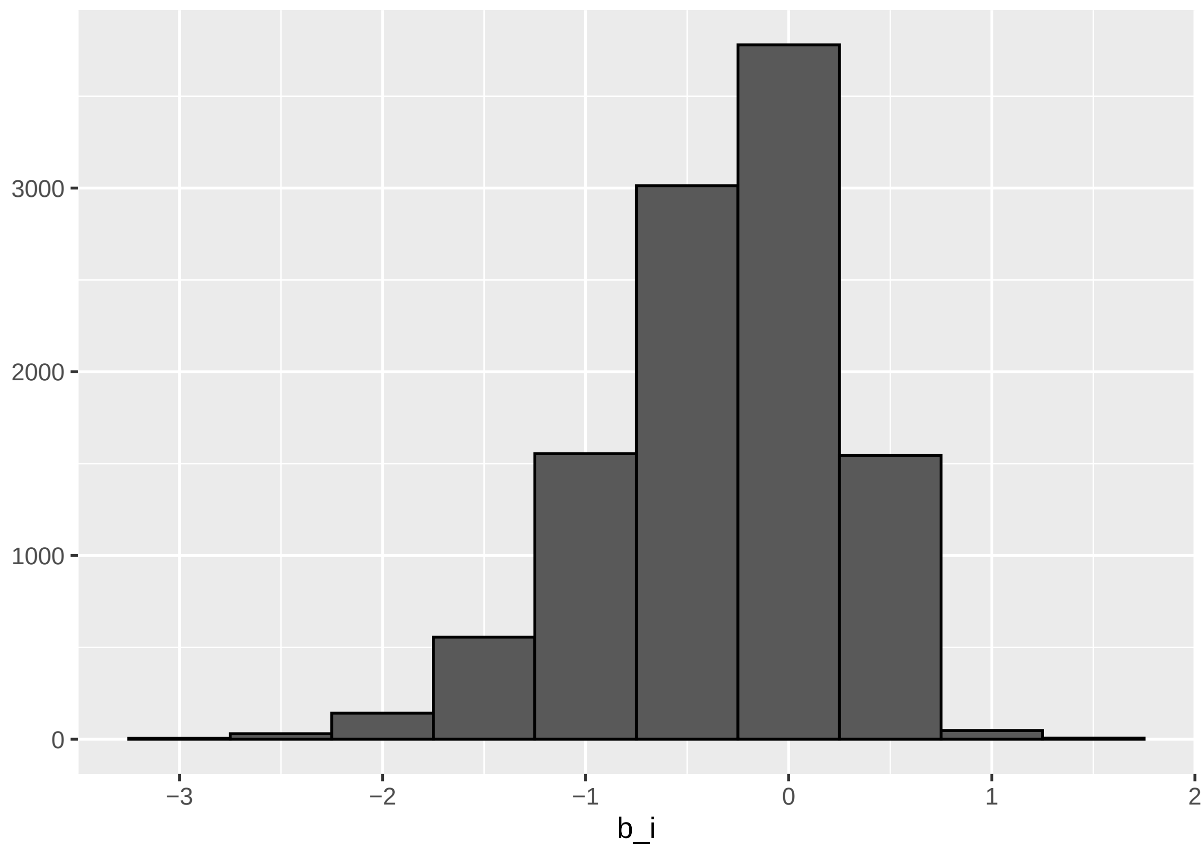
```
## # A tibble: 1 x 2
##   Model               RMSE
##   <chr>              <dbl>
## 1 Just the average 1.06120
```

Model 2

Modeling movie effects: adding b__i to represent average ranking for movie__i

Since the intrinsic features of a movie could obviously affect the ratings of a movie, we add the bias of movie/item (b__i) to the model, i.e., for each movie, the average of the ratings on that specific movie will have a difference from the overall average rating of all movies. We can plot the distribution of the bias and calculate the RMSE of this model.

```r
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```




```r
predicted_ratings_2 <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
model_2_rmse <- RMSE(validation$rating,predicted_ratings_2) # 0.943909
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Movie Effect Model",
                                     RMSE = model_2_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   Model                  RMSE
##   <chr>                 <dbl>
## 1 Just the average   1.06120
## 2 Movie Effect Model 0.943909
```

Adding the movie bias successfully brought the RMSE to lower than 1.

Model 3

User effects: adding b__u to represent average ranking for user__u

Similar to the movie effect, intrinsic features of a given user could also affect the ratings of a movie. We now further add the bias of user (b__u) to the movie effect model.

```r
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_ratings_3 <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_3_rmse <- RMSE(validation$rating,predicted_ratings_3) # 0.865349
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Movie + User Effects Model",
                                     RMSE = model_3_rmse))
rmse_results
```

```
## # A tibble: 3 x 2
##   Model                          RMSE
##   <chr>                         <dbl>
## 1 Just the average           1.06120
## 2 Movie Effect Model         0.943909
## 3 Movie + User Effects Model 0.865349
```

Model 4

regularization of movie effect

A) perform cross validation to determine the parameter lambda

To train the parameter lambda, 10-fold cross validation is used here within only the *edx* set, because the *validation* set should not be used to train any parameter.

```r
# set.seed(2019)
# cv_splits <- caret::createFolds(edx$rating, k=10, returnTrain =TRUE)
#
# # define a matrix to store the results of cross validation
# rmses <- matrix(nrow=10,ncol=51)
# lambdas <- seq(0, 3, 0.1)
#
# # perform 10-fold cross validation to determine the optimal lambda
# for(k in 1:10) {
#   train_set <- edx[cv_splits[[k]],]
#   test_set <- edx[-cv_splits[[k]],]
#
#   # Make sure userId and movieId in test set are also in the train set
#   test_final <- test_set %>%
#     semi_join(train_set, by = "movieId") %>%
#     semi_join(train_set, by = "userId")
#
#   # Add rows removed from validation set back into edx set
#   removed <- anti_join(test_set, test_final)
#   train_final <- rbind(train_set, removed)
#
```

```
#   mu <- mean(train_final$rating)
#   just_the_sum <- train_final %>%
#     group_by(movieId) %>%
#     summarize(s = sum(rating - mu), n_i = n())
#
#   rmses[k,] <- sapply(lambdas, function(l){
#     predicted_ratings <- test_final %>%
#       left_join(just_the_sum, by='movieId') %>%
#       mutate(b_i = s/(n_i+l)) %>%
#       mutate(pred = mu + b_i) %>%
#       pull(pred)
#     return(RMSE(predicted_ratings, test_final$rating))
#   })
# }
#
# rmses
# rmses_cv <- colMeans(rmses)
# rmses_cv
# qplot(lambdas,rmses_cv)
# lambdas[which.min(rmses_cv)]   #2.2
```

We get lambda = 2.2

##B) Model generation and prediction

```
lambda <- 2.2
mu <- mean(edx$rating)
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
predicted_ratings_4 <- validation %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
model_4_rmse <- RMSE(predicted_ratings_4, validation$rating)   # 0.943852 not too much improved
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Regularized Movie Effect Model",
                                     RMSE = model_4_rmse))
rmse_results
```

```
## # A tibble: 4 x 2
##   Model                              RMSE
##   <chr>                             <dbl>
## 1 Just the average                1.06120
## 2 Movie Effect Model              0.943909
## 3 Movie + User Effects Model      0.865349
## 4 Regularized Movie Effect Model 0.943852
```

Model 5

Regularization of both movie and user effects (use the same lambda for both movie and user effects)

1. **Perform cross validation to determine the parameter lambda**

```
# define a matrix to store the results of cross validation
# lambdas <- seq(4, 8, 0.1)
# rmses_2 <- matrix(nrow=10,ncol=length(lambdas))
# # perform 10-fold cross validation to determine the optimal lambda
# for(k in 1:10) {
#   train_set <- edx[cv_splits[[k]],]
#   test_set <- edx[-cv_splits[[k]],]
#
#   # Make sure userId and movieId in test set are also in the train set
#   test_final <- test_set %>%
#     semi_join(train_set, by = "movieId") %>%
#     semi_join(train_set, by = "userId")
#
#   # Add rows removed from validation set back into edx set
#   removed <- anti_join(test_set, test_final)
#   train_final <- rbind(train_set, removed)
#
#   mu <- mean(train_final$rating)
#
#   rmses_2[k,] <- sapply(lambdas, function(l){
#     b_i <- train_final %>%
#       group_by(movieId) %>%
#       summarize(b_i = sum(rating - mu)/(n()+l))
#     b_u <- train_final %>%
#       left_join(b_i, by="movieId") %>%
#       group_by(userId) %>%
#       summarize(b_u = sum(rating - b_i - mu)/(n()+l))
#     predicted_ratings <-
#       test_final %>%
#       left_join(b_i, by = "movieId") %>%
#       left_join(b_u, by = "userId") %>%
#       mutate(pred = mu + b_i + b_u) %>%
#       pull(pred)
#     return(RMSE(predicted_ratings, test_final$rating))
#   })
# }
# rmses_2
# rmses_2_cv <- colMeans(rmses_2)
# rmses_2_cv
# qplot(lambdas,rmses_2_cv)
# lambda <- lambdas[which.min(rmses_2_cv)]   #4.9
```

###2. Model generation and prediction

Regularized Movie Effect and User Effect Model

```r
mu <- mean(edx$rating)
b_i_reg <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u_reg <- edx %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted_ratings_5 <-
    validation %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
model_5_rmse <- RMSE(predicted_ratings_5, validation$rating)   # 0.864818
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Regularized Movie + User Effect Model",
                                     RMSE = model_5_rmse))
rmse_results
```

```
## # A tibble: 5 x 2
##   Model                                  RMSE
##   <chr>                                 <dbl>
## 1 Just the average                      1.06120
## 2 Movie Effect Model                    0.943909
## 3 Movie + User Effects Model            0.865349
## 4 Regularized Movie Effect Model        0.943852
## 5 Regularized Movie + User Effect Model 0.864818
```

5.Conclusion

From the summarized RMSEs of different models, we can see that Regularization of Movie+User Model largely improved the accuracy of the prediction.

```
## # A tibble: 5 x 2
##   Model                                  RMSE
##   <chr>                                 <dbl>
## 1 Just the average                      1.06120
## 2 Movie Effect Model                    0.943909
## 3 Movie + User Effects Model            0.865349
## 4 Regularized Movie Effect Model        0.943852
## 5 Regularized Movie + User Effect Model 0.864818
```

The final accuracy is 0.864818