Edx HarvardX Data Science Capstone Project

Daniel Yoo

3/5/2020

Contents

1 Introduction **1**

2 Exploratoy Data Analysis (EDA) **2**
2.1 EDA by movie 4
2.2 EDA by user 12

3 Modeling **14**
3.1 Simplest Model 14
3.2 Adding movie effects 15
3.3 Adding user effects 16
3.4 Regularization 17
3.5 Matrix Factorization 20

4 Results (validation) **22**

5 Conclusion **25**

1 Introduction

Recommendation systems use ratings that users have given items to make specific recommendations. Companies that sell many products to many customers and permit these customers to rate their products, like Amazon, are able to collect massive datasets that can be used to predict what rating a particular user will give a specific item. Items for which a high rating is predicted for a given user are then recommended to that user.

Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie. Here, we provide the basics of how these recommendations are made, motivated by some of the approaches taken by the winners of the Netflix challenges.

In this project, we use the movielens dataset by the GroupLens research lab. Our primary goal of this study is to build a robust movie recommendation system using the offered dataset to reach the aimed perfomance root mean squared error (RMSE) less than 0.86490 on the validation set. The project is part of the HarvardX's Data Science Professional Certificate program.

This report is composed of as follows.

1. Introduction
2. Exploratory Data Analysis
3. Modeling
4. Results (validation)
5. Conclusion

2 Exploratoy Data Analysis (EDA)

First, we load the dataset as the project instruction describes.

```
# First load packages
library(ggplot2)
library(tidyverse)
library(kableExtra)
library(recosystem)
library(data.table)
```

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

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

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 3.5.2
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

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

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

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[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")
set.seed(1)
# if using R 3.5 or earlier, use `set.seed(1)` instead
```

```
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)
```

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

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

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

Now, we see the basic structure of the dataset.

```
edx %>% class()
```

```
## [1] "data.frame"
```

```
dim(edx)
```

```
## [1] 9000055       6
```

```
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
```

```
edx %>%
  summarise(users = n_distinct(userId),
            movies = n_distinct(movieId))
```

```
##   users movies
## 1 69878  10677
```

```
edx %>%
  select(genres) %>%
  separate_rows(genres, sep = "\\|") %>%
  pull(genres) %>% unique()
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Crime"              "Thriller"           "Drama"
##  [7] "Sci-Fi"             "Adventure"          "Children"
## [10] "Fantasy"            "War"                "Animation"
## [13] "Musical"            "Western"            "Mystery"
## [16] "Film-Noir"          "Horror"             "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

There are 69878 users and 10677 movies included in the dataset. There are 20 genres. However, one genre is (no genres listed) which is obscure.

```
# check any missing data
any(is.na(edx))
```

```
## [1] FALSE
```

2.1 EDA by movie

Let's look at the basics of the rated movies.

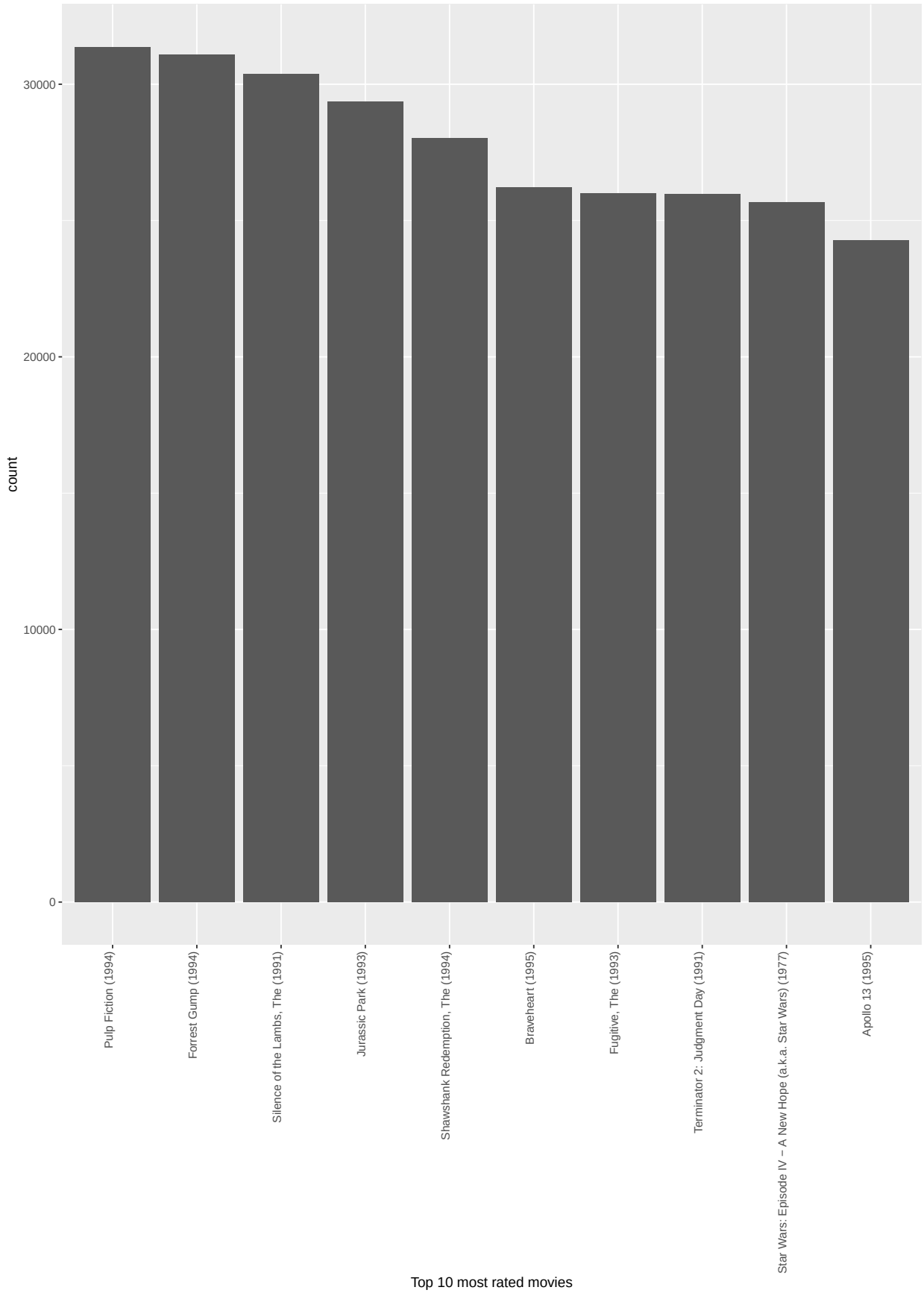
```
edx_movies <-
  edx %>%
  group_by(movieId) %>%
  summarise(count = n()) %>%
  arrange(-count)
edx_movies %>% summary()
```

```
##     movieId          count
##  Min.   :    1   Min.   :    1.0
##  1st Qu.: 2754   1st Qu.:   30.0
##  Median : 5434   Median :  122.0
##  Mean   :13105   Mean   :  842.9
##  3rd Qu.: 8710   3rd Qu.:  565.0
##  Max.   :65133   Max.   :31362.0
```

The most reviewed movie was rated by 31362 users and the least reviewed movie was rated by 1 user. The median and mean of the number of rating for the entire movie was 122 and 842.9385595, respectively.

```
# What are the most rated movies?
edx %>%
  group_by(title) %>%
  summarise(count = n()) %>%
  arrange(-count) %>%
  top_n(10) %>%
  ggplot(aes(x = reorder(title, -count), y = count)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0, hjust = 1)) +
  xlab("Top 10 most rated movies")
```

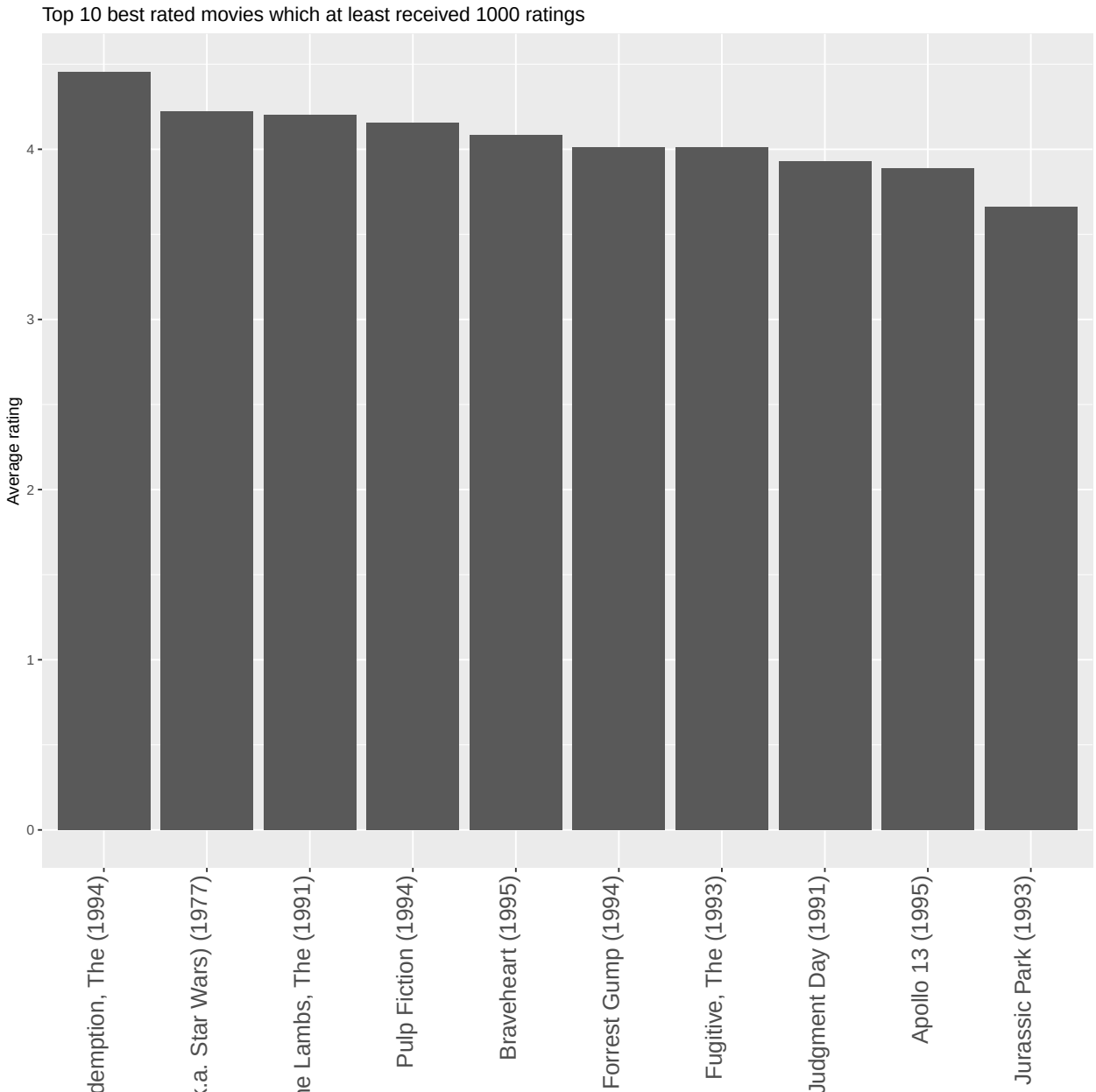
```
## Selecting by count
```



Top 10 most rated movies

```r
# What are the best rated movies?
edx %>%
  group_by(title) %>%
  filter(n() >= 1000) %>%
  summarise(avg = mean(rating), median = median(rating), n = n()) %>%
  arrange(-avg) %>%
  top_n(10) %>%
  ggplot(aes(reorder(title, -avg), avg)) +
  geom_bar(stat = "identity") +
  xlab("") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0, hjust = 1, size = 15)) +
  ylab("Average rating") + ggtitle("Top 10 best rated movies which at least received 1000 ratings")
```

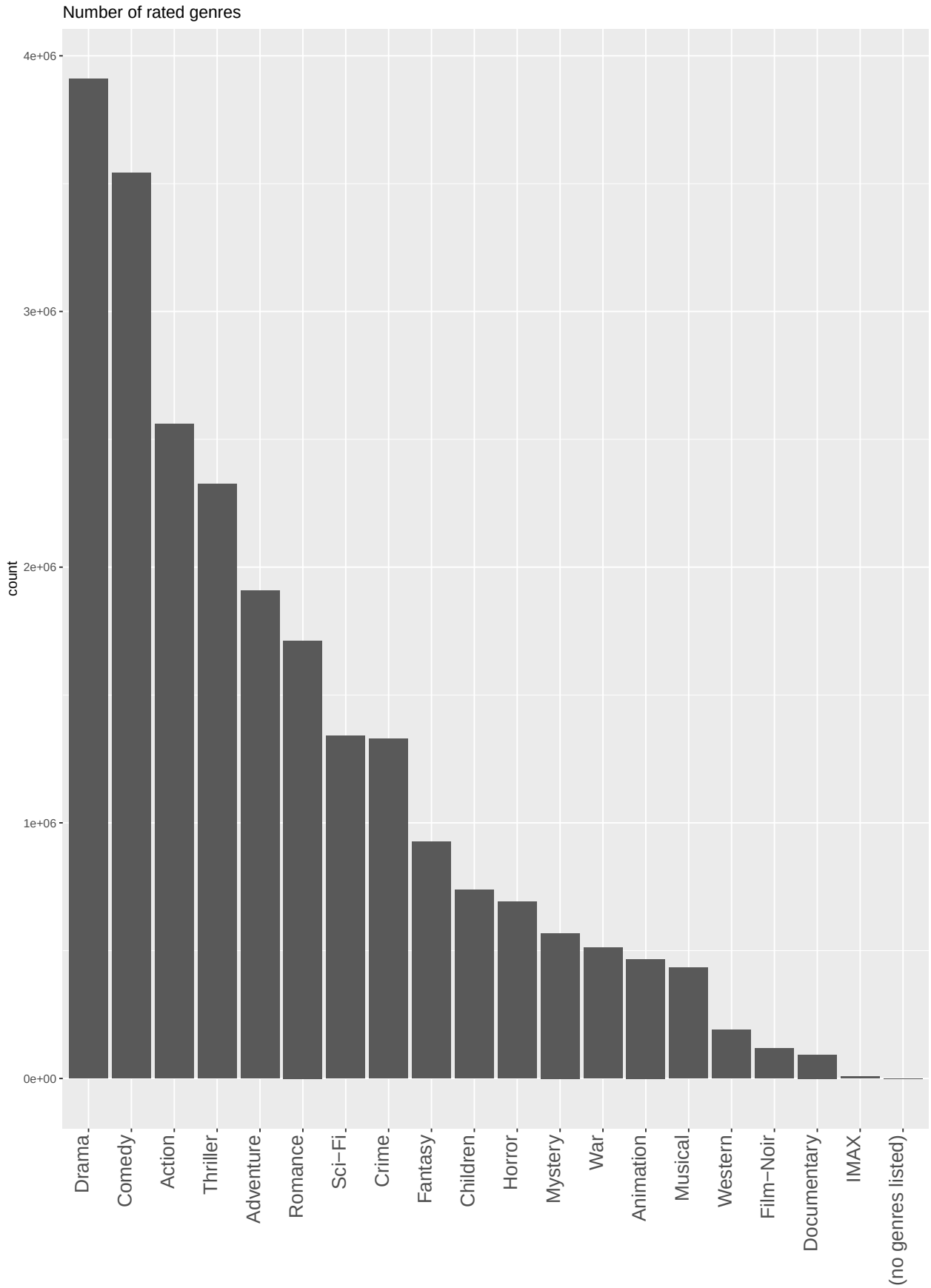
```
## Selecting by n
```


Top 10 best rated movies which at least received 1000 ratings
Average rating
0
1
2
3
4
Shawshank Redemption, The (1994)
Star Wars: Episode IV – A New Hope (a.k.a. Star Wars) (1977)
Silence of the Lambs, The (1991)
Pulp Fiction (1994)
Braveheart (1995)
Forrest Gump (1994)
Fugitive, The (1993)
Terminator 2: Judgment Day (1991)
Apollo 13 (1995)
Jurassic Park (1993)

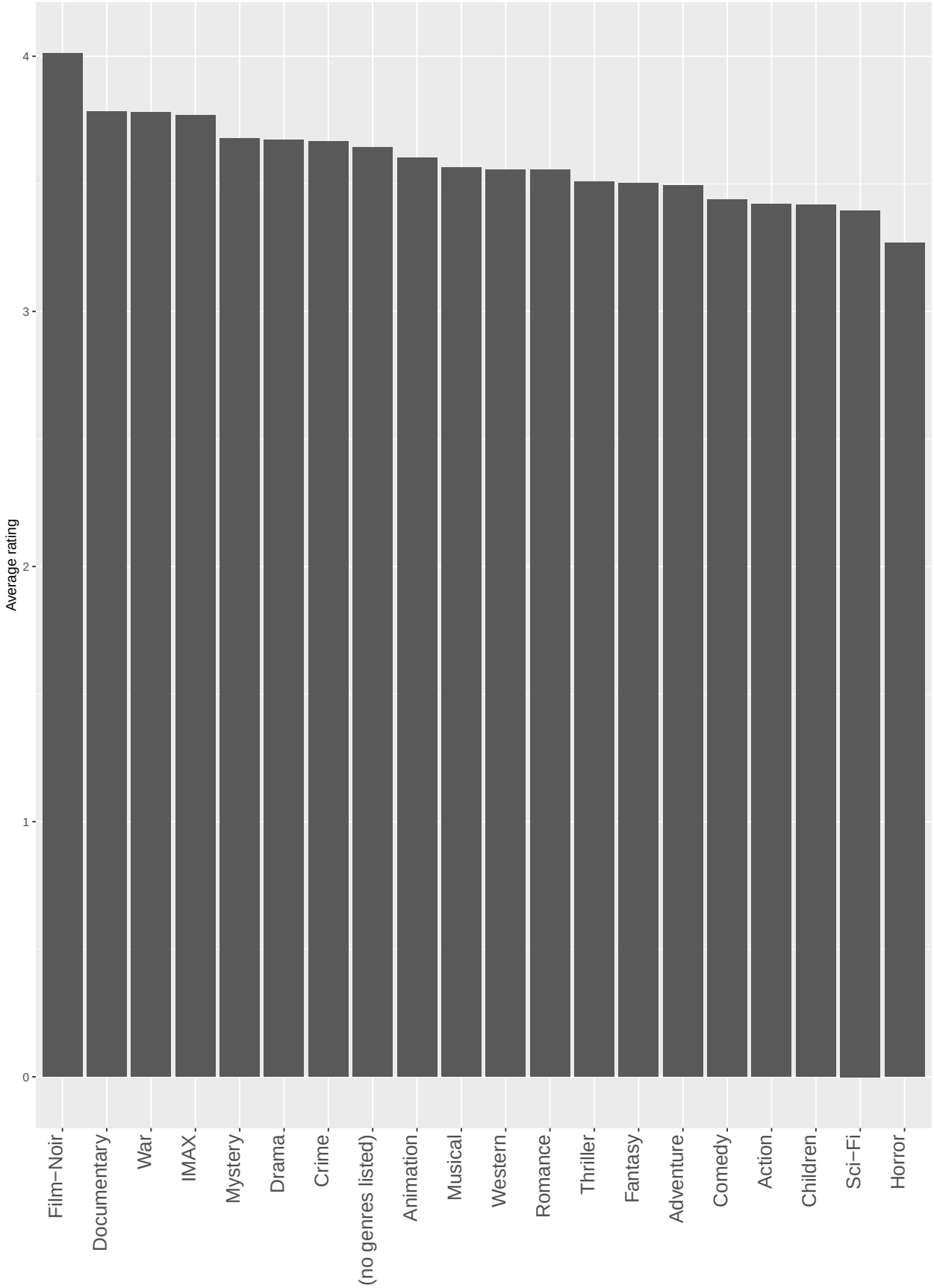
```r
# What are the most rated genres?
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(count = n()) %>%
  arrange(-count)
```

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

```r
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(count = n()) %>%
  ggplot(aes(reorder(genres, -count), count)) +
  geom_bar(stat = "identity") +
  xlab("") +
  theme(axis.text.x = element_text(size = 15, angle = 90, hjust = 1, vjust = 0)) +
  ggtitle("Number of rated genres")
```


Number of rated genres
count
4e+06
3e+06
2e+06
1e+06
0e+00
Drama
Comedy
Action
Thriller
Adventure
Romance
Sci−Fi
Crime
Fantasy
Children
Horror
Mystery
War
Animation
Musical
Western
Film−Noir
Documentary
IMAX
(no genres listed)

```r
# What are the best rated genres?
edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(avg = mean(rating), median = median(rating), count = n()) %>%
  ggplot(aes(reorder(genres, -avg), avg)) +
  geom_bar(stat = "identity") +
  xlab("") +
  theme(axis.text.x = element_text(size = 15, angle = 90, hjust = 1, vjust = 0)) +
  ylab("Average rating")
```


Average rating
0
1
2
3
4
Film−Noir
Documentary
War
IMAX
Mystery
Drama
Crime
(no genres listed)
Animation
Musical
Western
Romance
Thriller
Fantasy
Adventure
Comedy
Action
Children
Sci−Fi
Horror

```
  ggtitle("Average Rates of genres")
```

```
## $title
## [1] "Average Rates of genres"
##
## attr(,"class")
## [1] "labels"
```

We can see that drama, comedy and action are the most commonly rated genres. We can assume this is by the popular hollywood movies. The top rated movies are film-noir, documentary, and war movies. We can safely assume that this is because of the users who watch these movies tend to rate better. Note that there are 7 unspecified genre movies. The average rating for this movie is not meaningful due to the lack of users.

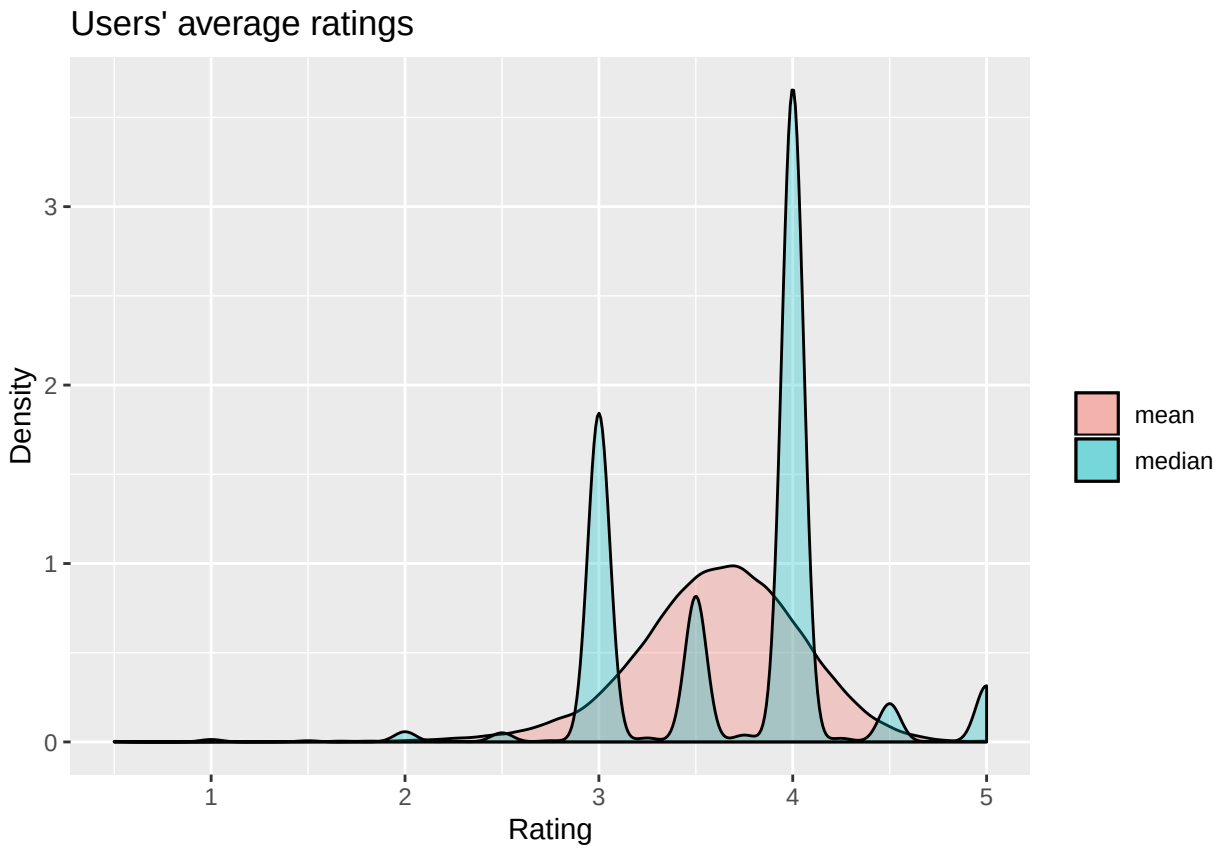
2.2 EDA by user

```
# Who has rated the most movies?
edx %>%
  group_by(userId) %>%
  summarise(avg = mean(rating),
            median = median(rating),
            count = n()) %>%
  summary()
```

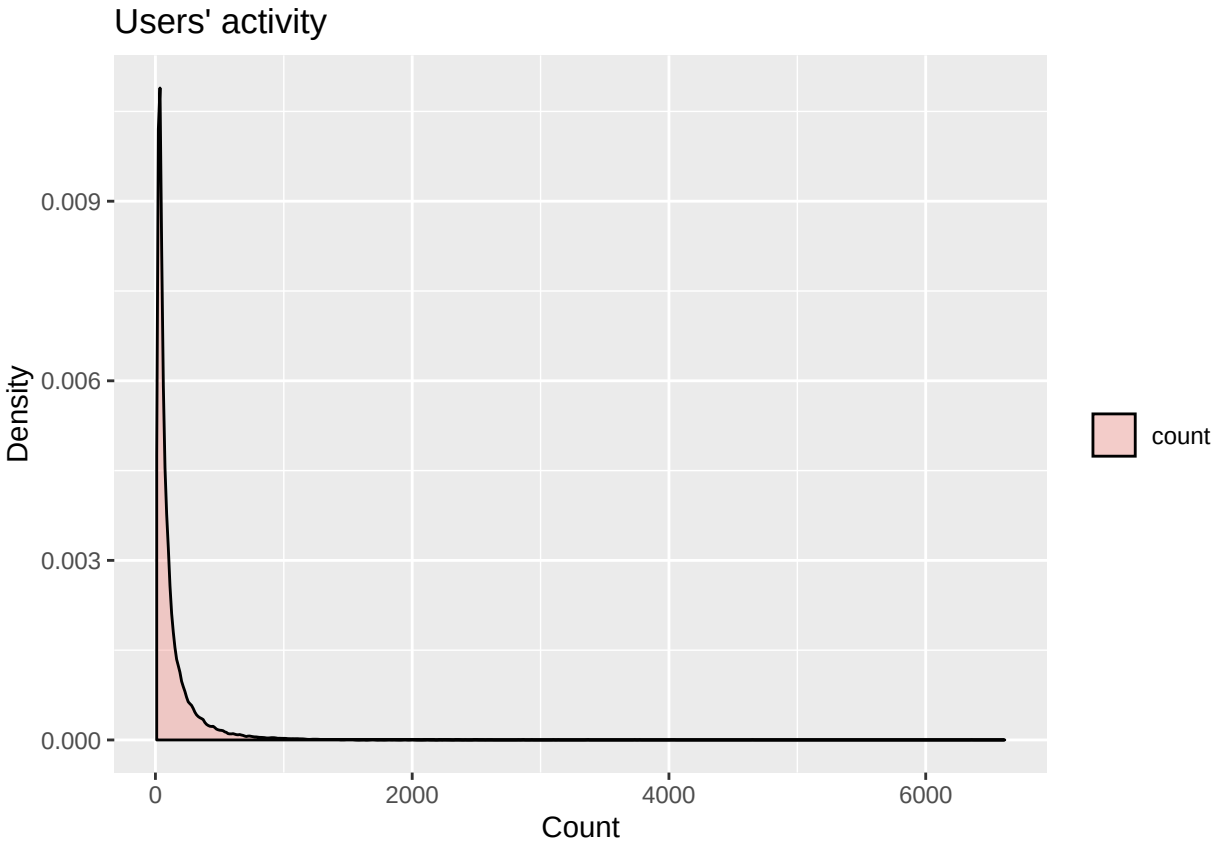
```
##      userId           avg            median          count
##  Min.   :    1   Min.   :0.500   Min.   :0.500   Min.   :  10.0
##  1st Qu.:17943   1st Qu.:3.357   1st Qu.:3.000   1st Qu.:  32.0
##  Median :35798   Median :3.635   Median :4.000   Median :  62.0
##  Mean   :35782   Mean   :3.614   Mean   :3.699   Mean   : 128.8
##  3rd Qu.:53620   3rd Qu.:3.903   3rd Qu.:4.000   3rd Qu.: 141.0
##  Max.   :71567   Max.   :5.000   Max.   :5.000   Max.   :6616.0
```

We can see that the most active user rated 6616 movies. The least active user rated 10 movies. The average rating is 3.6. The average number of rating per user is 129.

```
# Ratings are normally distributed
edx %>%
  group_by(userId) %>%
  summarise(mean = mean(rating),
            median = median(rating)) %>%
  ggplot() +
  geom_density(aes(mean, fill = "mean"), alpha = 0.3) +
  geom_density(aes(median, fill = "median"), alpha = 0.3) +
  xlab("Rating") +
  ylab("Density") +
  ggtitle("Users' average ratings") +
  labs(fill = "")
```




```
# Users' activity
edx %>%
  group_by(userId) %>%
  summarise(count = n()) %>%
  ggplot() +
  geom_density(aes(count, fill = "count"), alpha = 0.3) +
  xlab("Count") +
  ylab("Density") +
  ggtitle("Users' activity") +
  labs(fill = "")
```



As expected, users' ratings are normally distributed. Also, users' activities are poisson distribution, highly skewed.

3 Modeling

We use root mean square error as a metric to judge our model performance.

First, we will make an extra test set to check the performance of our models before testing on the final validation dataset.

```
# split train and test sets
set.seed(1)
ind <- createDataPartition(y = edx$rating, times = 1, p = 0.3, list = F)
train_set <- edx[-ind, ]
test_set <- edx[-ind, ]

# semi_join to double check that all users and movies in the test set are also included in the train se
test_set <-
  test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

3.1 Simplest Model

Here, we will build the simplest model. Using the arithmetic mean to predict.

```
mu_hat <- mean(train_set$rating)
naive_rmse <- RMSE(mu_hat, test_set$rating)

# create a data frame to keep the performances for different models
rmse_results <- tibble(
  method = c("Project goal", "Simple average"),
  RMSE = c(0.86490, naive_rmse)
)
mu_hat
```

```
## [1] 3.512596
```

```
rmse_results %>% kable()
```

method	RMSE
Project goal	0.864900
Simple average	1.060214

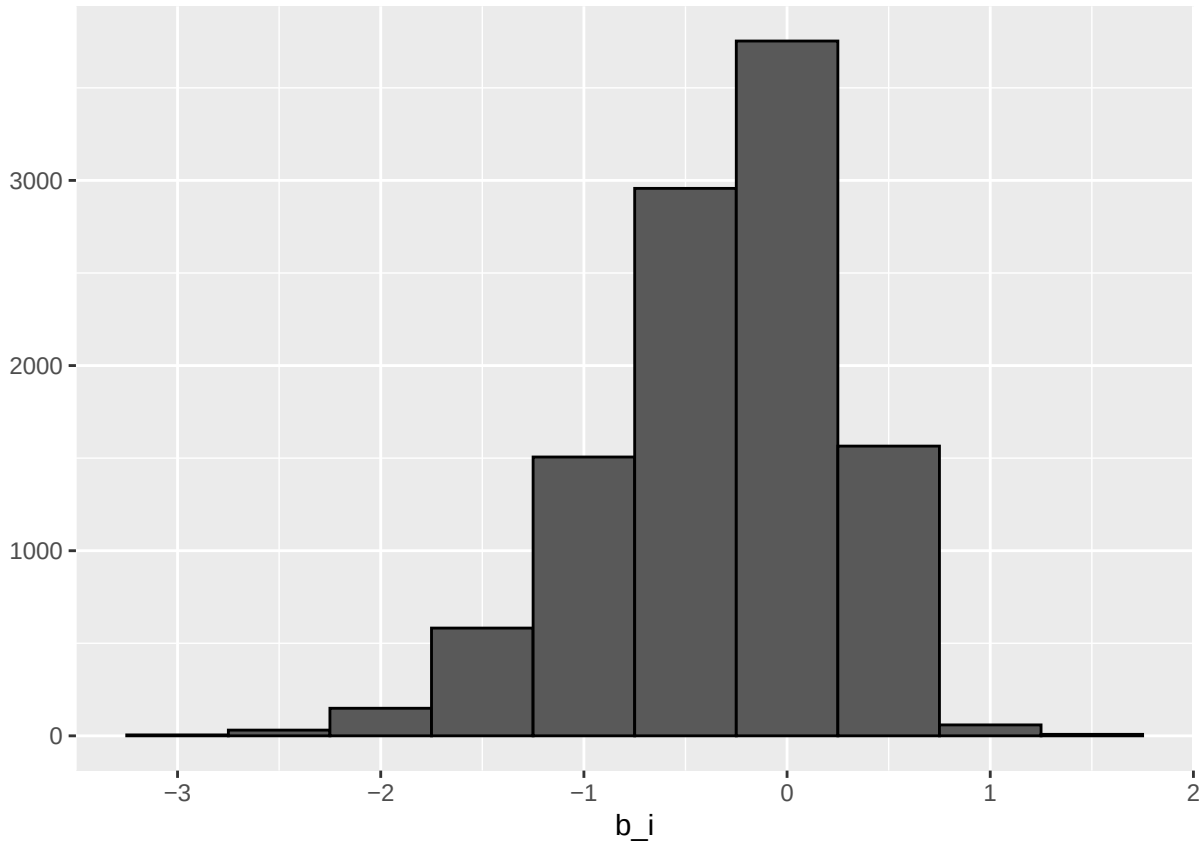
The average rating of all movies across all users is 3.5125959.

The RMSE with the simple mean is 1.0602141.

3.2 Adding movie effects

We can do better with adding effects. First, we add movie effects.

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

qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```




```
predicted_ratings <- mu + test_set %>%
    left_join(movie_avgs, by = "movieId") %>%
    pull(b_i)
movie_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results, tibble(
    method = "Movie Effect",
    RMSE = movie_rmse)
)
rmse_results %>% kable()
```

method	RMSE
Project goal	0.8649000
Simple average	1.0602141
Movie Effect	0.9418562

We see that adding movie effect has made great improvements in our metric. Now our RMSE is under 0.95.

3.3 Adding user effects

Second, we add user effects.

```
# Calculate user effects
user_avgs <-
  train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))
```

```r
# Predict
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

movie_user_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <-
  rmse_results %>%
  bind_rows(tibble(
    method = "Movie + User Effects",
    RMSE = movie_user_rmse
  ))
rmse_results %>% kable()
```

method	RMSE
Project goal	0.8649000
Simple average	1.0602141
Movie Effect	0.9418562
Movie + User Effects	0.8548301

We have made another great achievement by adding user effects on top of the movie effect. It is less than our first aimed RMSE 0.86490. However, we can do better by trying regularization.

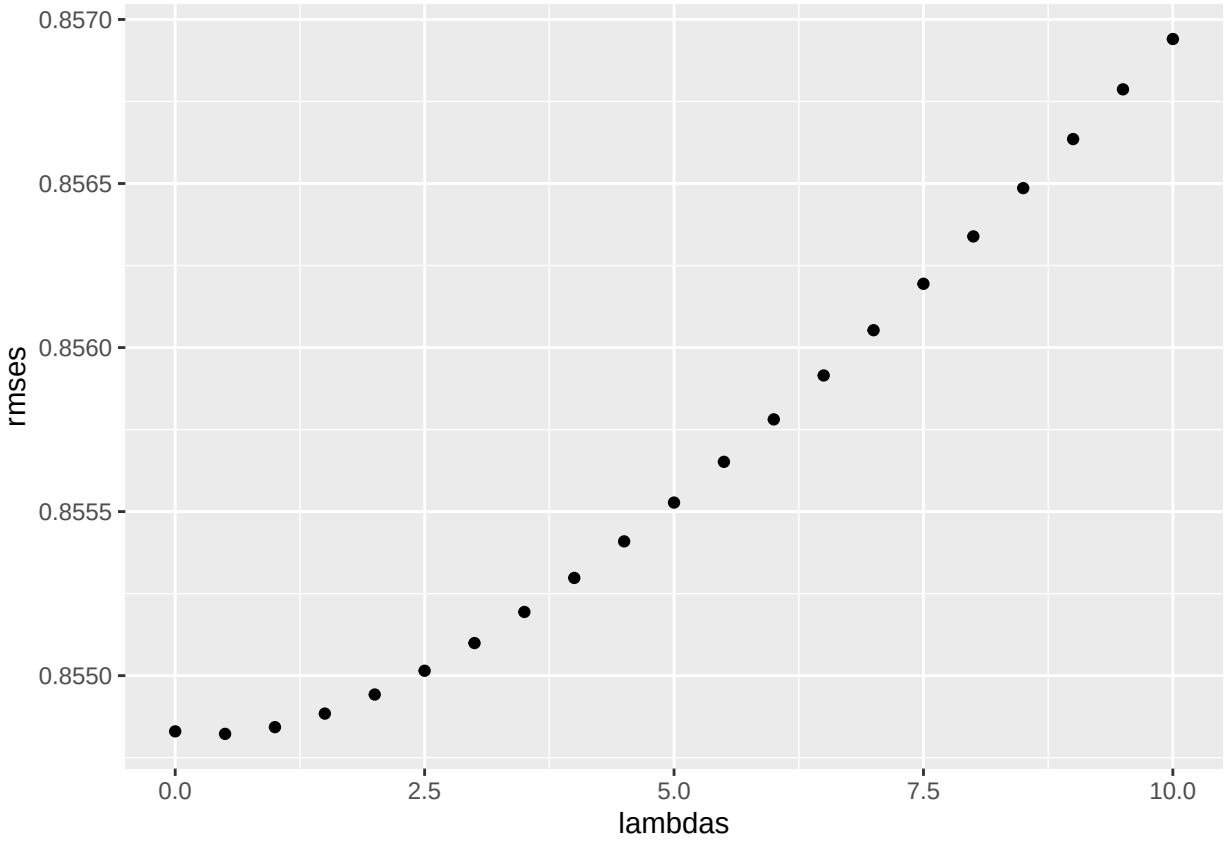
3.4 Regularization

The largest and smallest movie and user effects are with movies and users with only few ratings. So we use regularization to penalize such estimates that come from small sizes. We use cross-validation to find the best tuning parameter.

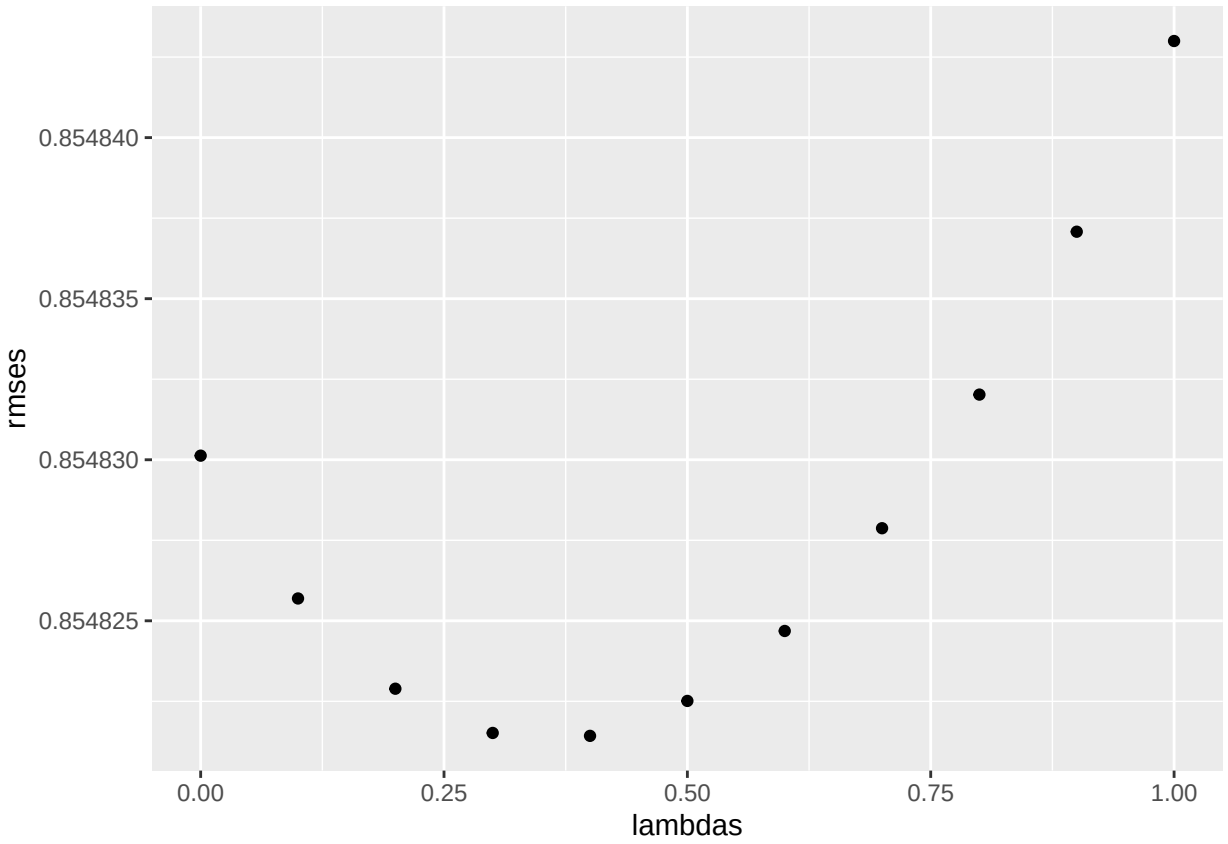
Note: this process might take some time.

```r
# cross-validation

lambdas <- seq(0, 10, 0.5)
lambda_look <- function(l, train_set, test_set) {
  mu <- mean(train_set$rating)
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu) / (n() + l))

  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - mu - b_i) / (n() + l))

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  RMSE(predicted_ratings, test_set$rating)
```

```
}

rmses <- sapply(lambdas, lambda_look, train_set = train_set, test_set = test_set)
qplot(lambdas, rmses)
```




```
# more specified search
lambdas <- seq(0, 1, 0.1)
rmses <- sapply(lambdas, lambda_look, train_set = train_set, test_set = test_set)
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 0.4
```

```
l <- 0.4
mu <- mean(train_set$rating)
b_i <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu) / (n() + l))

b_u <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu - b_i) / (n() + l))

predicted_ratings <-
  test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
regularized_RMSE <- RMSE(predicted_ratings, test_set$rating)
rmse_results <-
  rmse_results %>%
  bind_rows(tibble(
    method = "Movie, User Effects, Regularized",
    RMSE = regularized_RMSE
  ))
```

```
rmse_results %>% kable()
```

method	RMSE
Project goal	0.8649000
Simple average	1.0602141
Movie Effect	0.9418562
Movie + User Effects	0.8548301
Movie, User Effects, Regularized	0.8548214

We made a very tiny improvement by searching lambda. This may due to our high proportion, 70% of training partition.

3.5 Matrix Factorization

Matrix factorization is a widely used concept in machine learning. It is very much related to factor analysis, singular value decomposition (SVD), and principal component analysis (PCA).

We use `recosystem` R package developed by Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and Chih-Jen Lin, an open source library for recommender system using parallel matrix factorization. github page

```
library(recosystem)
set.seed(123)
train_data <- with(train_set,
                   data_memory(user_index = userId,
                               item_index = movieId,
                               rating = rating))
test_data <- with(test_set,
                   data_memory(user_index = userId,
                               item_index = movieId,
                               rating = rating))

r <- recosystem::Reco()

# we perform without extra tuning process for the sake of simplicity
r$train(train_data, opts = c(niter = 60))
```

```
## iter      tr_rmse          obj
##    0       0.9839   1.0632e+07
##    1       0.8837   9.3799e+06
##    2       0.8630   9.2278e+06
##    3       0.8478   9.0856e+06
##    4       0.8406   9.0308e+06
##    5       0.8350   8.9937e+06
##    6       0.8305   8.9647e+06
##    7       0.8272   8.9427e+06
##    8       0.8249   8.9260e+06
##    9       0.8232   8.9164e+06
##   10       0.8219   8.9065e+06
##   11       0.8209   8.9006e+06
##   12       0.8201   8.8944e+06
##   13       0.8194   8.8883e+06
##   14       0.8189   8.8849e+06
##   15       0.8184   8.8812e+06
##   16       0.8180   8.8797e+06
```

```
##   17       0.8176   8.8770e+06
##   18       0.8173   8.8737e+06
##   19       0.8170   8.8734e+06
##   20       0.8168   8.8687e+06
##   21       0.8166   8.8692e+06
##   22       0.8163   8.8662e+06
##   23       0.8161   8.8649e+06
##   24       0.8159   8.8612e+06
##   25       0.8158   8.8631e+06
##   26       0.8156   8.8599e+06
##   27       0.8154   8.8572e+06
##   28       0.8153   8.8581e+06
##   29       0.8152   8.8570e+06
##   30       0.8150   8.8552e+06
##   31       0.8149   8.8539e+06
##   32       0.8148   8.8538e+06
##   33       0.8147   8.8524e+06
##   34       0.8146   8.8526e+06
##   35       0.8145   8.8510e+06
##   36       0.8144   8.8497e+06
##   37       0.8143   8.8499e+06
##   38       0.8143   8.8487e+06
##   39       0.8142   8.8488e+06
##   40       0.8141   8.8482e+06
##   41       0.8140   8.8476e+06
##   42       0.8139   8.8471e+06
##   43       0.8139   8.8462e+06
##   44       0.8138   8.8450e+06
##   45       0.8137   8.8441e+06
##   46       0.8137   8.8440e+06
##   47       0.8136   8.8442e+06
##   48       0.8135   8.8433e+06
##   49       0.8135   8.8428e+06
##   50       0.8134   8.8427e+06
##   51       0.8133   8.8400e+06
##   52       0.8133   8.8413e+06
##   53       0.8132   8.8400e+06
##   54       0.8131   8.8407e+06
##   55       0.8131   8.8396e+06
##   56       0.8130   8.8394e+06
##   57       0.8129   8.8370e+06
##   58       0.8129   8.8382e+06
##   59       0.8129   8.8382e+06
```

```
y_hat_reco <- r$predict(test_data, out_memory())
rmse_results <- bind_rows(rmse_results,
                          tibble(
                              method = "Matrix Factorization",
                              RMSE = RMSE(y_hat_reco, test_set$rating)
                          ))
rmse_results %>% kable()
```

method	RMSE
Project goal	0.8649000
Simple average	1.0602141
Movie Effect	0.9418562
Movie + User Effects	0.8548301
Movie, User Effects, Regularized	0.8548214
Matrix Factorization	0.8088490

We observe a great improvements from the matrix factorization. Now we are ready to test this in the real validation set.

4 Results (validation)

We already observed that the RMSE by linear model successfully achieved the aimed RMSE on the test set. In addition, we went further to minimize RMSE value by using the matrix factorization. Here, we test whether our models perform as we expect or not.

```
# Since we finished finding our hyperparameter, lambda, we can use the whole edx set from now on.
mu_final <- mean(edx$rating)

# movie effect
b_i_final <-
  edx %>%
  group_by(movieId) %>%
  summarise(b_i_final = sum(rating - mu_final) / (n() + l))

# user effect
b_u_final <-
  edx %>%
  left_join(b_i_final, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u_final = sum(rating - mu_final - b_i_final) / (n() + l))

pred_validation <-
  validation %>%
  left_join(b_i_final, by = "movieId") %>%
  left_join(b_u_final, by = "userId") %>%
  mutate(pred_final = mu_final + b_i_final + b_u_final) %>%
  pull(pred_final)

RMSE_validation_regularized <- RMSE(pred_validation, validation$rating)
rmse_results <-
  rmse_results %>%
  bind_rows(
    tibble(
      method = "Movie, User Effects, Regularized (validation)",
      RMSE = RMSE_validation_regularized
    )
            )
rmse_results %>% kable()
```

method	RMSE
Project goal	0.8649000
Simple average	1.0602141
Movie Effect	0.9418562
Movie + User Effects	0.8548301
Movie, User Effects, Regularized	0.8548214
Matrix Factorization	0.8088490
Movie, User Effects, Regularized (validation)	0.8652450

Here, we observe that our regularized model does not perform as good as on the test set. Let's try our matrix factorization model.

```
set.seed(123)
train_data <- with(edx,
                   data_memory(user_index = userId,
                               item_index = movieId,
                               rating = rating))
test_data <- with(validation,
                   data_memory(user_index = userId,
                               item_index = movieId,
                               rating = rating))

r <- recosystem::Reco()

# we perform without extra tuning process for the sake of simplicity
r$train(train_data, opts = c(niter = 60))
```

```
## iter      tr_rmse          obj
##    0       0.9554   1.4624e+07
##    1       0.8788   1.3358e+07
##    2       0.8540   1.3069e+07
##    3       0.8415   1.2929e+07
##    4       0.8347   1.2861e+07
##    5       0.8305   1.2816e+07
##    6       0.8278   1.2789e+07
##    7       0.8262   1.2773e+07
##    8       0.8249   1.2759e+07
##    9       0.8239   1.2744e+07
##   10       0.8233   1.2739e+07
##   11       0.8227   1.2734e+07
##   12       0.8223   1.2730e+07
##   13       0.8219   1.2723e+07
##   14       0.8215   1.2726e+07
##   15       0.8212   1.2717e+07
##   16       0.8209   1.2714e+07
##   17       0.8207   1.2712e+07
##   18       0.8205   1.2710e+07
##   19       0.8203   1.2705e+07
##   20       0.8202   1.2708e+07
##   21       0.8200   1.2702e+07
##   22       0.8198   1.2701e+07
##   23       0.8197   1.2699e+07
##   24       0.8196   1.2698e+07
##   25       0.8194   1.2697e+07
##   26       0.8193   1.2695e+07
```

```
##   27       0.8192   1.2693e+07
##   28       0.8191   1.2692e+07
##   29       0.8190   1.2691e+07
##   30       0.8189   1.2688e+07
##   31       0.8189   1.2690e+07
##   32       0.8188   1.2688e+07
##   33       0.8187   1.2688e+07
##   34       0.8186   1.2686e+07
##   35       0.8186   1.2687e+07
##   36       0.8185   1.2684e+07
##   37       0.8184   1.2684e+07
##   38       0.8183   1.2682e+07
##   39       0.8182   1.2681e+07
##   40       0.8182   1.2682e+07
##   41       0.8181   1.2681e+07
##   42       0.8181   1.2679e+07
##   43       0.8180   1.2678e+07
##   44       0.8180   1.2678e+07
##   45       0.8179   1.2677e+07
##   46       0.8179   1.2675e+07
##   47       0.8178   1.2676e+07
##   48       0.8178   1.2676e+07
##   49       0.8177   1.2675e+07
##   50       0.8177   1.2675e+07
##   51       0.8176   1.2672e+07
##   52       0.8176   1.2673e+07
##   53       0.8175   1.2671e+07
##   54       0.8175   1.2673e+07
##   55       0.8174   1.2672e+07
##   56       0.8173   1.2669e+07
##   57       0.8173   1.2670e+07
##   58       0.8172   1.2669e+07
##   59       0.8172   1.2669e+07
```

```
y_hat_reco <- r$predict(test_data, out_memory())
rmse_results <- bind_rows(rmse_results,
                    tibble(
                        method = "Matrix Factorization (validation)",
                        RMSE = RMSE(y_hat_reco, validation$rating)
                    ))
rmse_results %>% kable()
```

method	RMSE
Project goal	0.8649000
Simple average	1.0602141
Movie Effect	0.9418562
Movie + User Effects	0.8548301
Movie, User Effects, Regularized	0.8548214
Matrix Factorization	0.8088490
Movie, User Effects, Regularized (validation)	0.8652450
Matrix Factorization (validation)	0.8311711

The final RMSE with matrix factorization on the validation set is less than our aimed RMSE score.

5 Conclusion

We used movielens dataset to construct a recommender system through HarvardX Data Science Capstone project. We built both linear models and matrix factorized model. We observed that adding movie and user effects could significantly improve the performance of model. We could finally achieved to reach our aimed RMSE score by using our final matrix factozied model. Our model has several weaknesses. First, we only used two predictors, user and movie information. Second, our model is built for existing users so the implementation on a new user would be time-consuming process by running a new model each time.