MovieLens Capstone Project

Lidia Almazan

5th June 2019

Introduction

This is a report on movie recommendation system using the MovieLens dataset. The data is obtained from GroupLens, a research lab at the University of Minnesota. We will use the 10M version of the MovieLens dataset to make the computation a bit faster because we use less data.

First of all we load the data from the corresponding link provided by grouplens:

```
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")

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

The data set is built by:

```
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")
```

We create a validation set of 10% of MovieLens data and a 90% set for training:

```
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

We 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")
```

and finnally we add the 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)
```

We want to create our own recommendation system using the MovieLens data set. In the next secctions, it will be shown how an analysis of the data and how in the R script are implemented each proposed model for the prediction rating and the RMSE obtained from each of them.

The libraries used in the present project are:

```
library(tidyverse)
library(caret)
library(ggplot2)
```

Methods and analysis

Data Analysis

First of all we need to know the data we are working with, this is why we need to analyse the edx data obtained from the previous code lines.

The dimension of the file we are working with is:

```
dim(edx)
```

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

There are 9 million of rows (observations) with 6 columns (variables), it means that we are working with more than 36 millions of data.

Using the command head(), we can see an overview of the data:

```
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
```

The 6 columns presents on the edx set are:

- **userId**: identification number given to each user.
- **movieId**: identification number given to each movie.
- **rating**: stars given to a movie for each user going from 0 the worst rating to 5 the best rating in intervas of 0.5.
- **timestamp**: date and time where the rating was given.
- **title_**: title of the movie and year of the first release.
- **genre**: categories associated to the movie with a separation of a vertical bar.

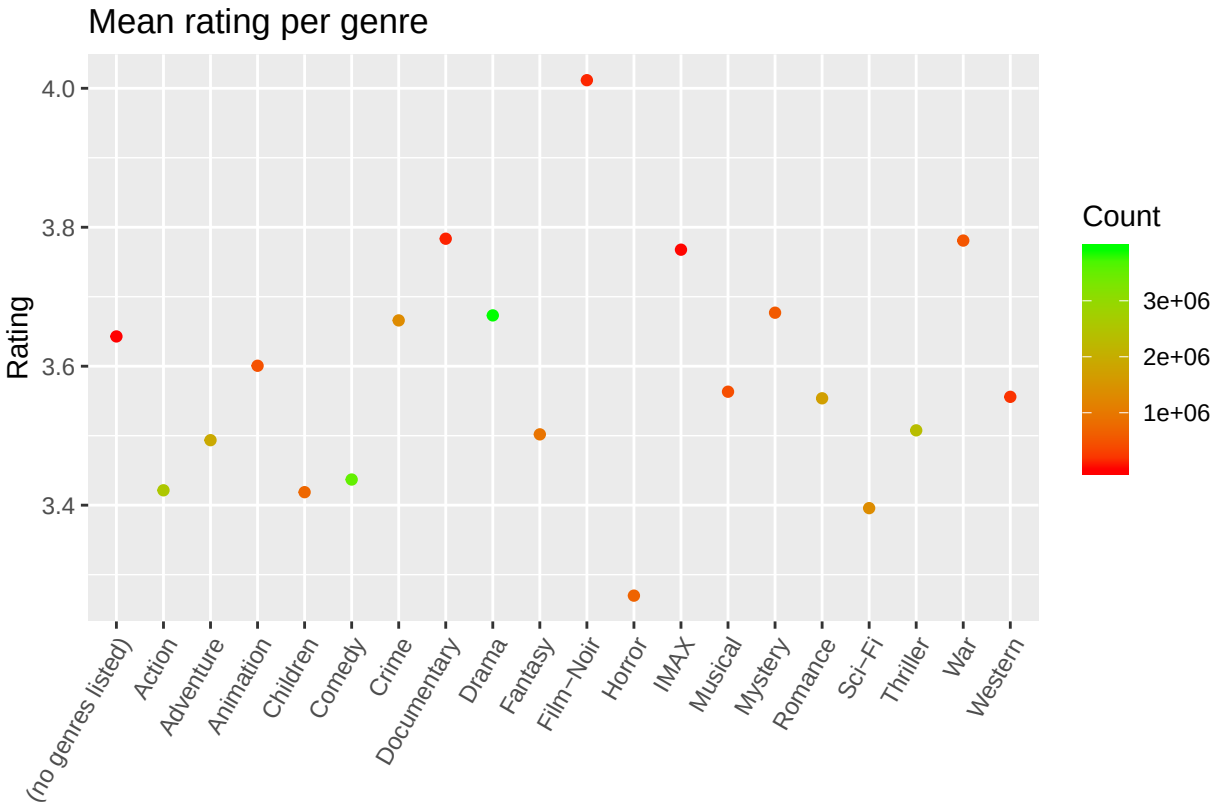
In the edx set, there are many users rating many movies, therefore we use the following command to show how many unique users are providing ratings and how many unique movies are rated:

```
edx %>% summarize(n_movies = n_distinct(edx$movieId),
                  n_users = n_distinct(edx$userId))
```

```
##   n_movies n_users
## 1    10677   69878
```

As we have seen with the head() command, the movies have multiple genres, therefore we use the following commands to separate them and show in a plot how many genres are, which ones are better rated and how many times have been rated.

```
edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(Count = n(), mean_rating = mean(rating)) %>%
  arrange(desc(Count)) %>% ggplot(aes(genres)) +
  geom_point(aes(genres, mean_rating, color = Count)) +
  theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  labs(x = "", y = "Rating", title = "Mean rating per genre") +
  scale_color_gradient(low = "red", high = "green")
```

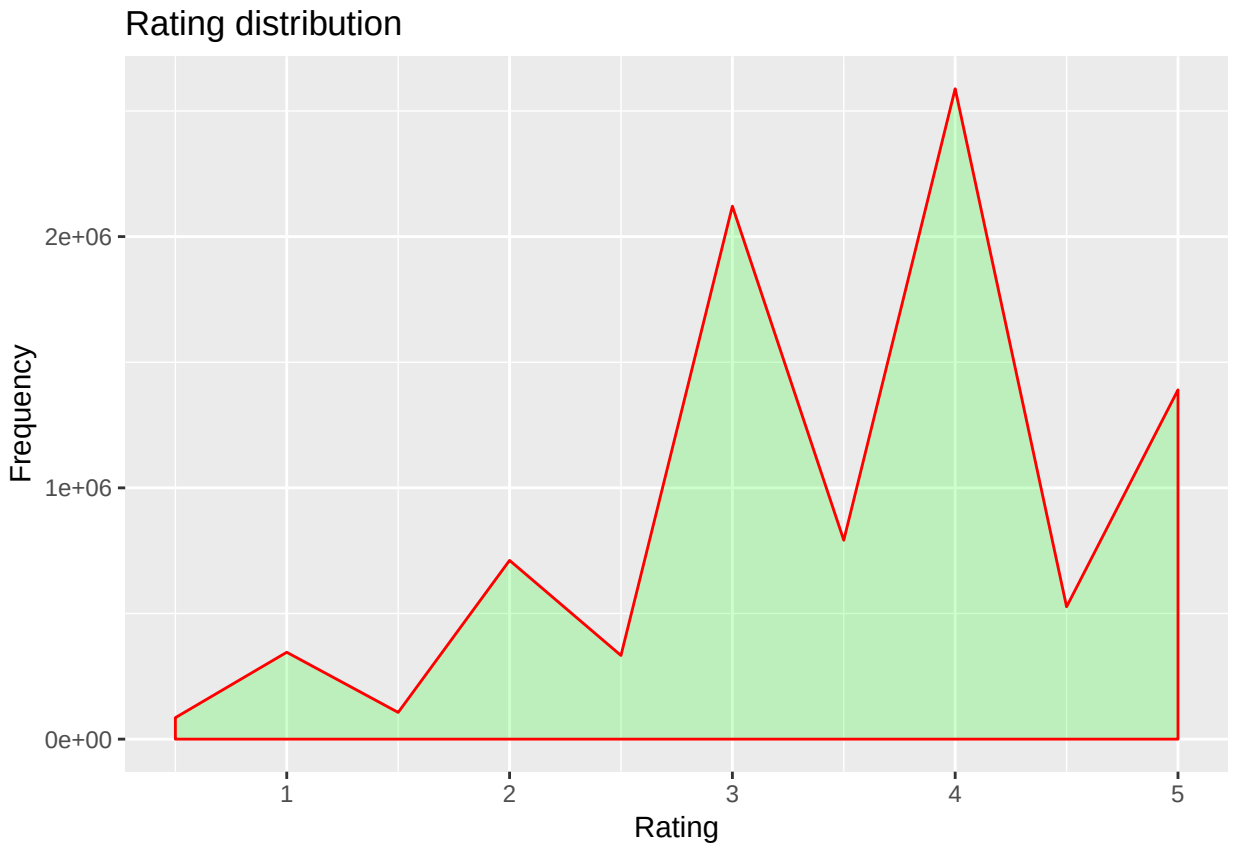


From the previous plot we can see that the most rated genres are Drama followed by Comedy, and the better rated genre is Film-Noir followed by Documentary and War movies.

Not all the ratings are equally distributed, as we can see in the next plot. It could mean different things, one of the most obvious is that users tend to rate movies that they like more. Also from the plot we can see how the rating given is usually integer, half star is less common.

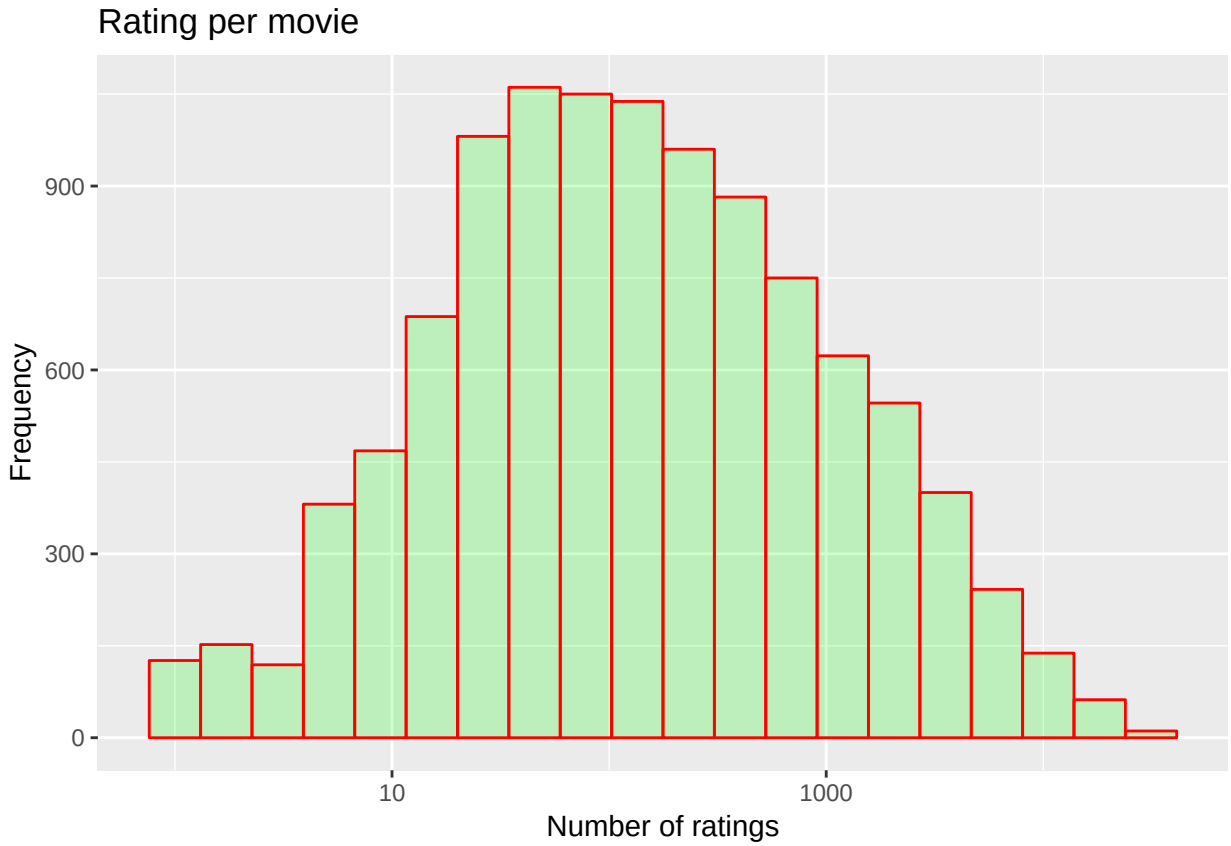
```
edx %>%
  group_by(rating) %>%
  summarize(count = n())  %>%
  ggplot(aes(x = rating, y = count)) +
```

```
  geom_area(color = "red", fill = "green", alpha = .2) +
  labs(x = "Rating", y = "Frequency", title = "Rating distribution")
```



Also the number of ratings per movie is no uniform, it means, as it is shown in the next plot, there are movies that are almost no rated.

```
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 20, col = "red", fill = "green", alpha = .2) +
  scale_x_log10() +
  labs(x = "Number of ratings", y = "Frequency", title = "Rating per movie")
```



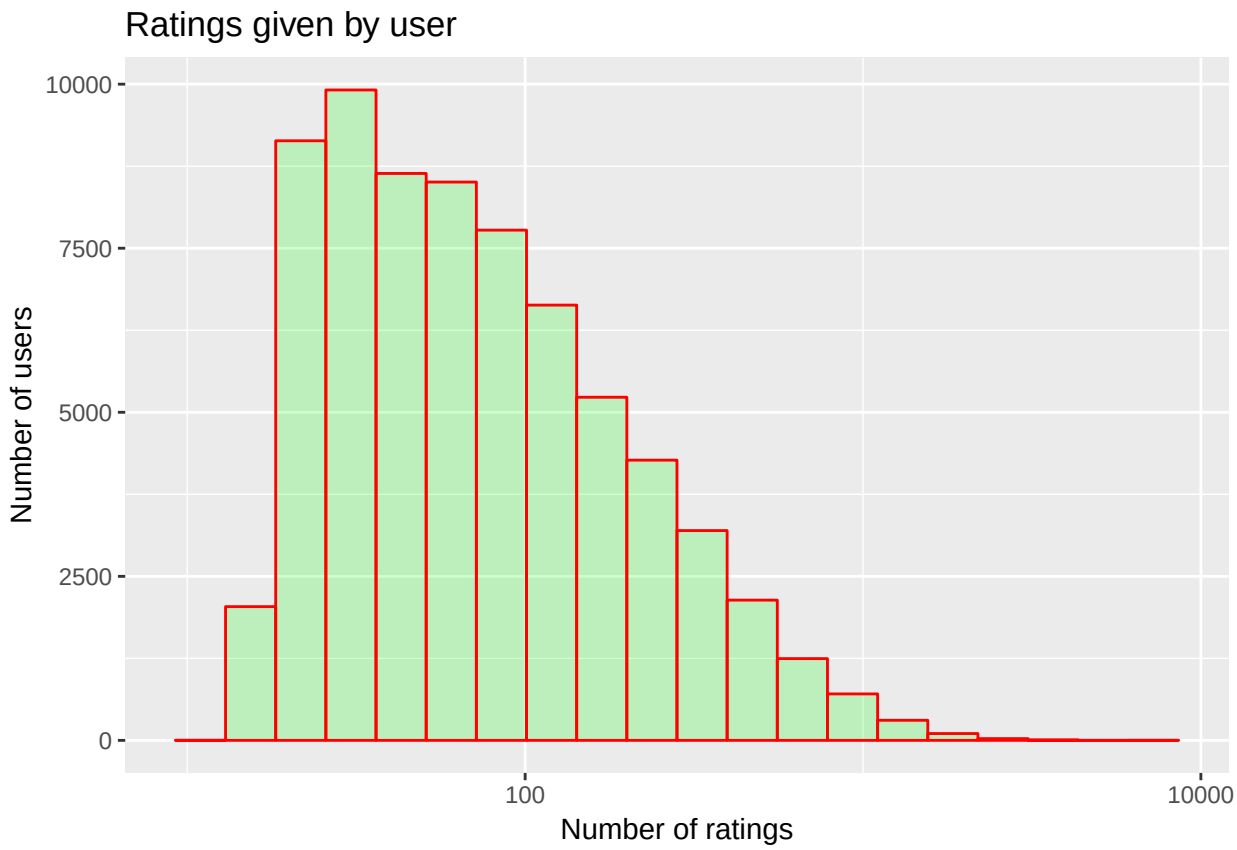
Going more into detail to the movies that have a few ratings, we see how there are 126 movies that have been rated only once. This could be important when we look for a model, making us to use a regularization technique to give less weight to movies less rated to avoid possible noise.

```
edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = rating, n_rating = count)
```

```
## # A tibble: 126 x 3
##    title                                                rating n_rating
##    <chr>                                                 <dbl>    <int>
##  1 1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)             2          1
##  2 100 Feet (2008)                                           2          1
##  3 4 (2005)                                                  2.5        1
##  4 Accused (Anklaget) (2005)                                 0.5        1
##  5 Ace of Hearts (2008)                                      2          1
##  6 Ace of Hearts, The (1921)                                 3.5        1
##  7 Adios, Sabata (Indio Black, sai che ti dico: Sei un gr~    1.5        1
##  8 Africa addio (1966)                                       3          1
##  9 Aleksandra (2007)                                         3          1
## 10 Bad Blood (Mauvais sang) (1986)                           4.5        1
## # ... with 116 more rows
```

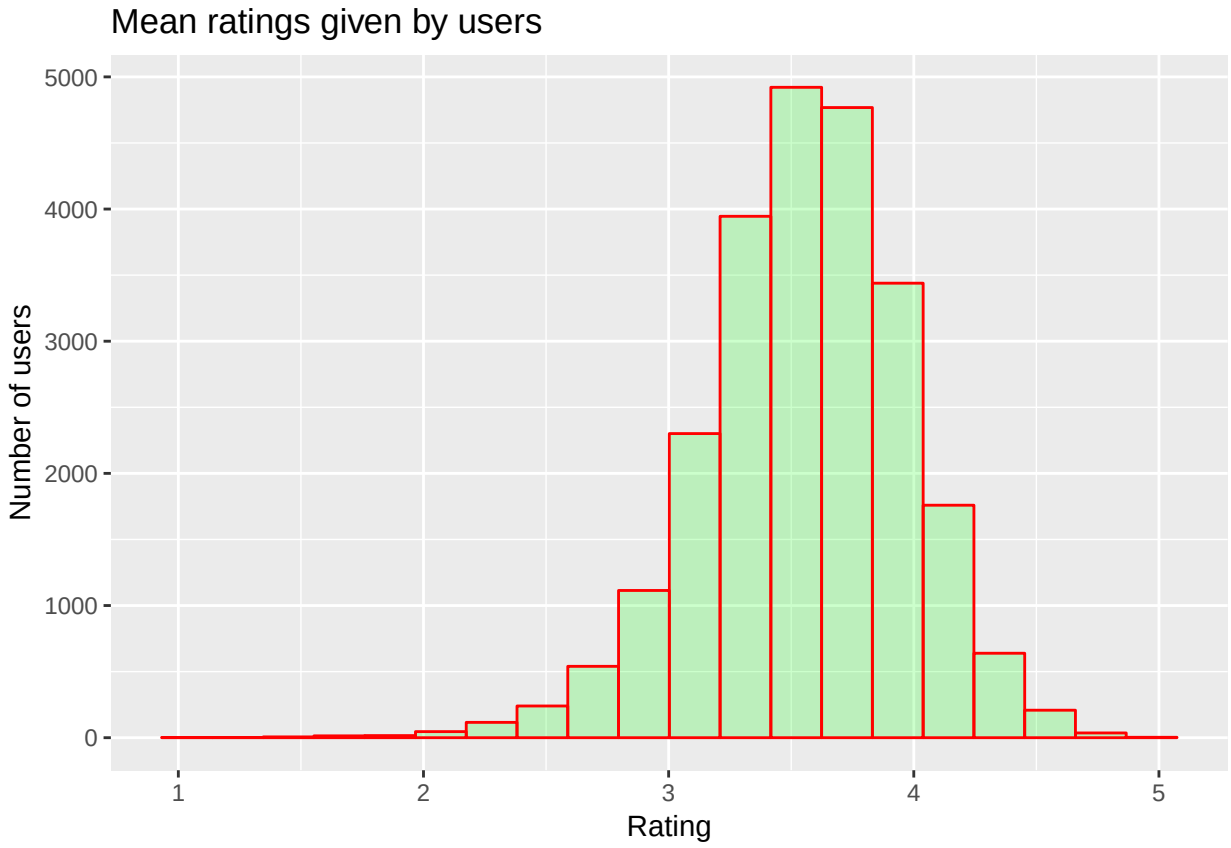
Looking for each user, we can observe that the majority of users have rated between 40 and 100 movies.

```r
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 20, col = "red", fill = "green", alpha = .2) +
  scale_x_log10() +
  labs(x = "Number of ratings", y = "Number of users", title = "Ratings given by user")
```



Also there is some bias in the ratings that each user gives, this means that there are users that usually gives higher ratings than others, as we can see in the next plot.

```r
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(mean_rating = mean(rating)) %>%
  ggplot(aes(mean_rating)) +
  geom_histogram(bins = 20, col = "red", fill = "green", alpha = .2) +
  labs(x = "Rating", y = "Number of users", title = "Mean ratings given by users")
```



Machine Learning models

Model 1 - Mean

We start with a simple model which predict the same rating for all movies and users, with all the differences explained by random variation.

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

with $\varepsilon_{u,i}$ being the independent error from the same distribution centered at 0 and μ. The user is denoted by the subscript u, and the movie is denoted by the subscript i.

The average of all ratings is:

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

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

We compute the residual mean squared error (RMSE) to compare the different models, which is the typical error made while predicting the movie rating. If the number is larger than 1, it means our typical error is larger than one star, which is not good. This happens in our basic model, where our RMSE is 1.061202.

```
model1_rmse <- RMSE(validation$rating, mu_hat)

rmse_results <- data_frame(Model = "1 - Mean", RMSE = model1_rmse)
rmse_results %>% knitr::kable()
```

Model	RMSE
1 - Mean	1.061202

Model 2 - Movie effect

In the second model, we add a bias called b_i that represent the average ranking for the movie i, because there are some movies that are rated higher than others. In that case the model used is:

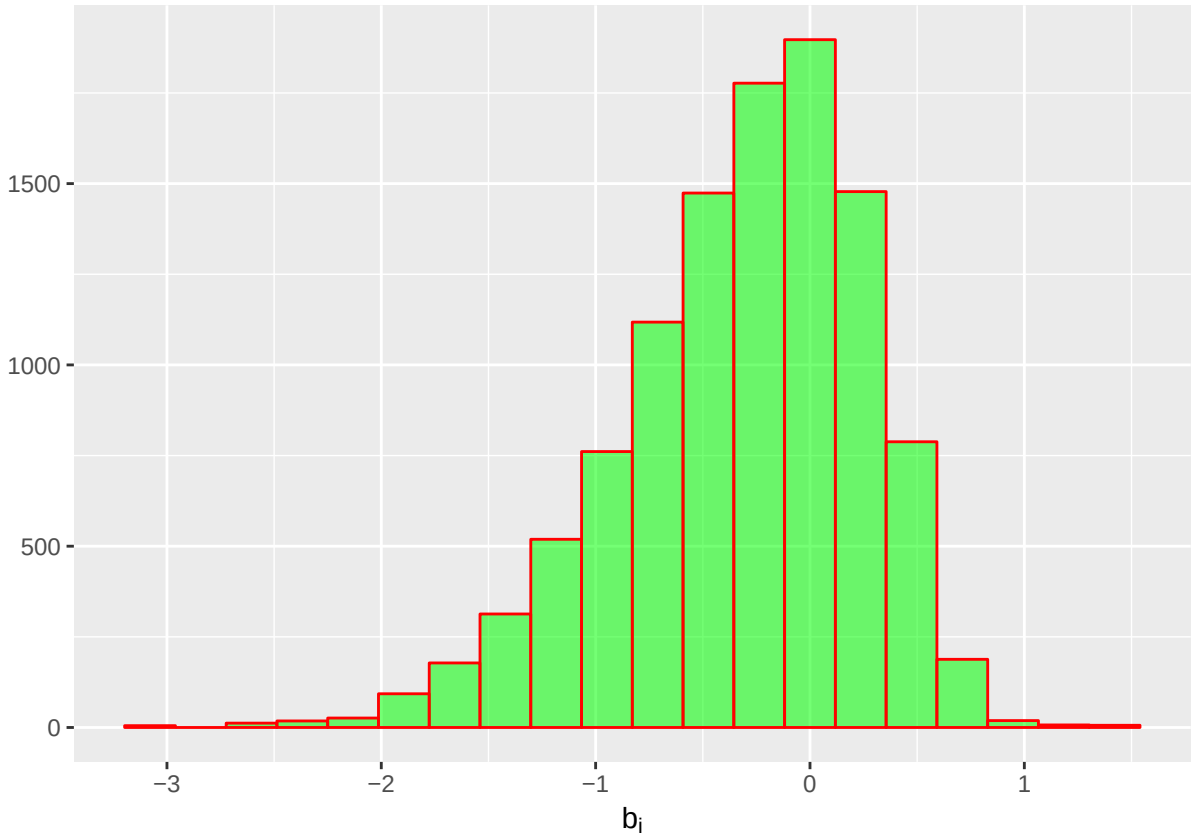
$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

Other method as least square could be used to estimate b_i, however we are dealing with a very large data set and the use of the function 'r lm() would take so much time. Therefore, we use the following code which is an approximation:

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

Plotting the b_i we see that these estimates vary substantially:

```
movie_avgs %>% qplot(b_i, geom = "histogram", bins = 20, data = .,
                     color = I("red"), fill = I("green"), alpha = .1) +
  theme(legend.position = "none") + xlab(bquote('b'['i']))
```



In the next step we apply the method to the validation set and we see that the resulting RMSE is slightly better compared with the previous model.

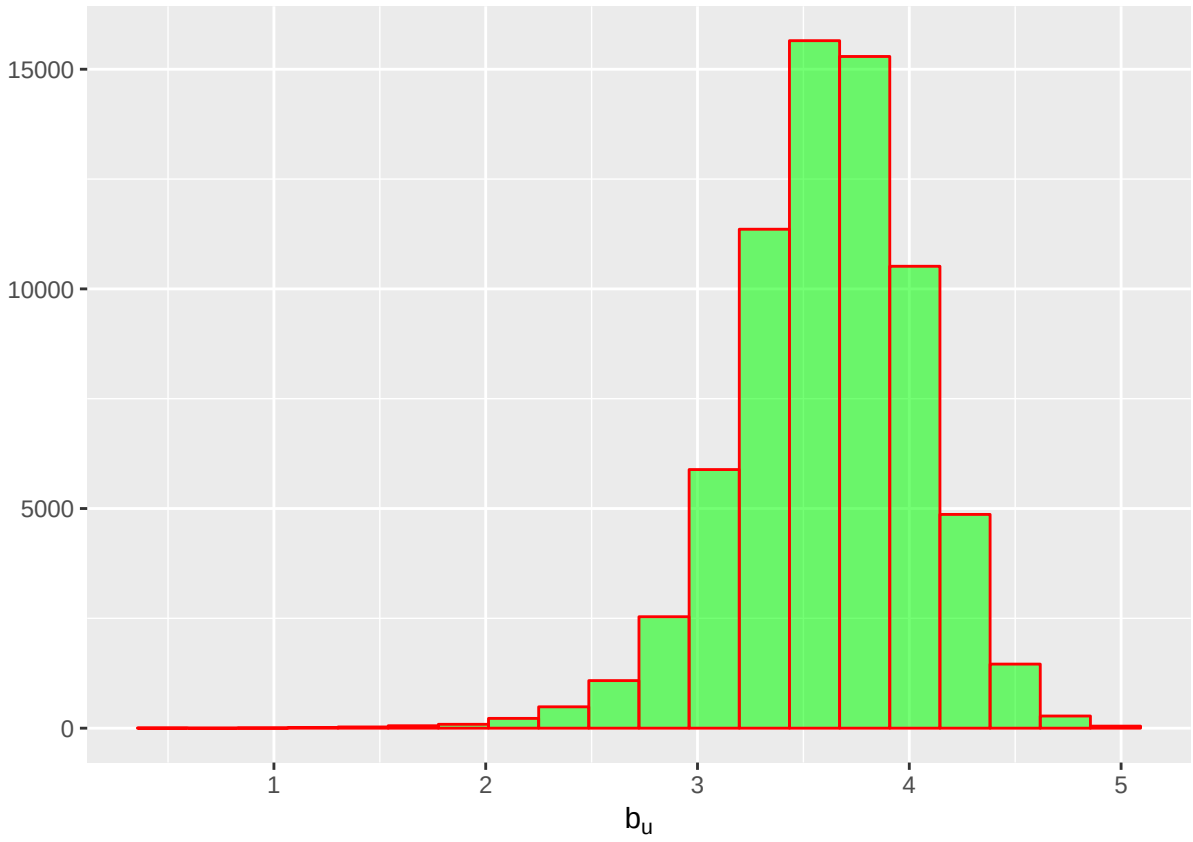
```
predicted_ratings <- mu_hat + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model2_rmse <- RMSE(predicted_ratings, validation$rating)


rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="2 - Movie effect",
                                     RMSE = model2_rmse ))

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

Model	RMSE
1 - Mean	1.0612018
2 - Movie effect	0.9439087

Model 3 - Movie and user effect

As we have seen in the analysis of data, and as it is shown in the next plot, the users are not evaluating the movies impartially and we see from the plot that some users are more "haters" and others love every movie.

```
edx %>% group_by(userId) %>% summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>% qplot(b_u, geom = "histogram", bins = 20, data = .,
      color = I("red"), fill = I("green"), alpha = .1) +
  theme(legend.position = "none") + xlab(bquote('b'['u']))
```



In the next model we use the bias effect associated to each user b_u, in that case the model is:

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

We compute the b_u using:

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

and construct the predictors and compute the RMSE:

```
predicted_ratings <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)

model3_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="3 - Movie and user effect",
                                     RMSE = model3_rmse))

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

Model	RMSE
1 - Mean	1.0612018
2 - Movie effect	0.9439087

Model	RMSE
3 - Movie and user effect	0.8653488

which is much better than the other two models, we have achieve the goal of the project which was to get a RMSE lower than 0.87750. **The RMSE obtained using the movie and user effect is 0.8653488.**

Model 4 - Regularized movie and user effect

We have already achieve our goal, which was reducing the RMSE under 0.87750. However, we could improve a bit the prediction taking into account that there are some movies rated just once, as we have seen in a table in the data analysis section.

This can be improved using regularization, which permits us to penalize large estimates that are formed using small sample sizes. For that purpose we use a tuning parameter λ, which regularize the estimate user effects as well as the movie effects, that is done minimizing the following expression:

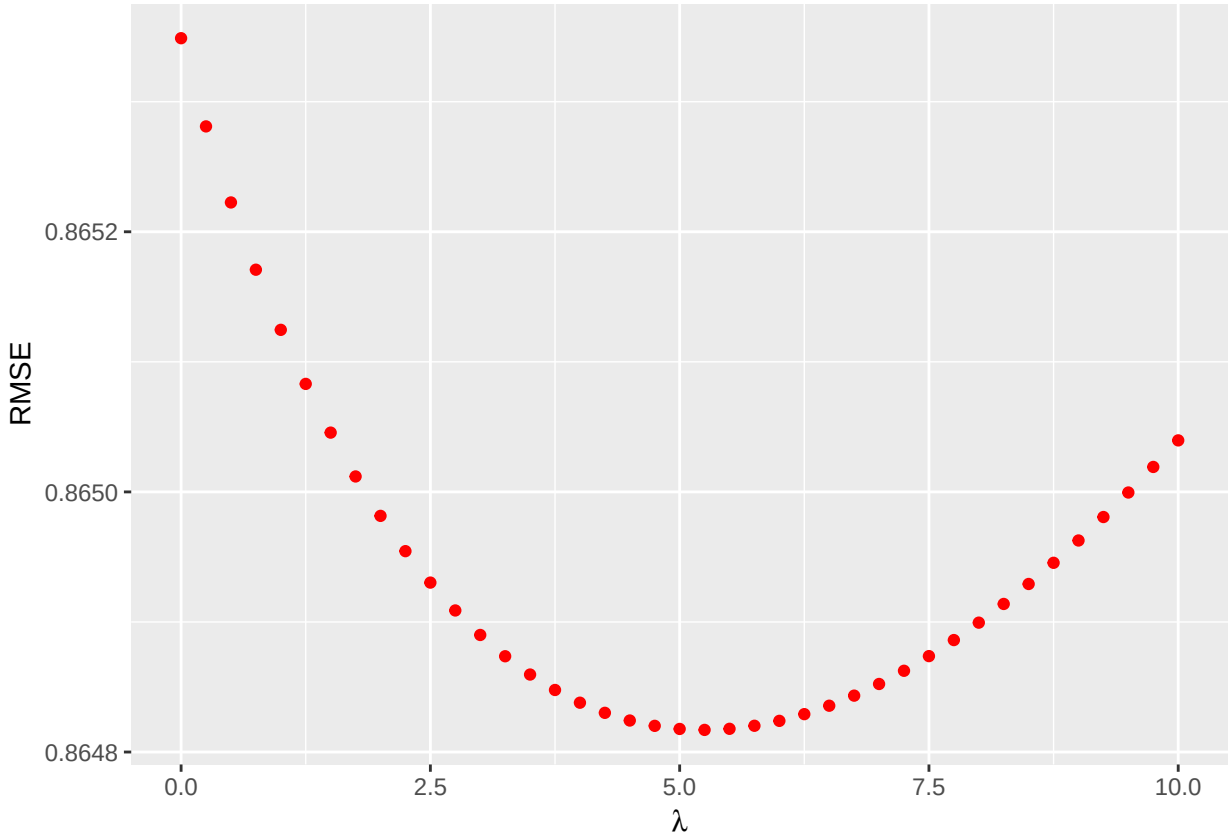
$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu+b_i+b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

To do so, we use cross-validation to choose the optimal λ to get the smaller RMSE.

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


# For each lambda,find b_i & b_u, followed by rating prediction & testing
# note:the below code could take some time
rmses <- sapply(lambdas, function(l){

  mu_hat <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu_hat)/(n()+l))

  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu_hat)/(n()+l))

  predicted_ratings <-
    validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu_hat + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation$rating))
})
```

Comparing the obtained RMSE with the tuning parameter λ, we see that the best RMSE is obtained for $\lambda = 5.25$.

```
qplot(lambdas, rmses, color = I("red")) + xlab(expression(lambda)) + ylab("RMSE")
```




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

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

The RMSE obtained with this model is 0.8648170 slightly better than with the model 3.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="4 - Regularized movie and user effect",
                                     RMSE = min(rmses)))

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

Model	RMSE
1 - Mean	1.0612018
2 - Movie effect	0.9439087
3 - Movie and user effect	0.8653488
4 - Regularized movie and user effect	0.8648170

Results

In our project we have used different models to predict the rating given by users for different movies, and the RMSE obtained for each model are:

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

Model	RMSE
1 - Mean	1.0612018
2 - Movie effect	0.9439087
3 - Movie and user effect	0.8653488
4 - Regularized movie and user effect	0.8648170

with a **lowest RMSE found in the regularized movie and user effect model being 0.8648170**, but not being so different from the movie and user effect model.

Conclusions

After analysing the data through graphical representations and computing the RMSE for each proposed machine learning model, we found the **best predictor for ratings was considering the bias given by users and movies with a regularization which gave us a RMSE of 0.8648170.**