Movielens Recommendation System

Dr. Ivan S. Zapreev

2019-12-31

According to Wikipedia:

> A recommender system or a recommendation system (sometimes replacing 'system' with a synonym such as platform or engine) is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.

Introduction

Recommendation systems are widely used in commercial application to provide targeted commercials or suggest online content. In the latter, one can think of *Spotify* suggesting songs or *Youtube* proposing videos. The main goal of a recommendation system is to accurately predict client's preferences, based on available data on the previous user behavior. Note that, such data includes the bahaviour of all the clients.

The goal of this work is to build a movie recommendation system based on the MovieLens 10M dataset available from https://grouplens.org/datasets/movielens/10m/. To reach our goal we will use supervised machine learning techniques studied within the *"PH125.8x Data Science: Machine Learning"* course (a part of the broader HarvardX *Data Science Professional* certification program). The recommendation system will be therefore based on (linear) statistical models trained on the subset of the MovieLens 10M dataset.

The rest of the document is organized as follows. The *"Analysis"* section explains the process and techniques used, such as data cleaning, data exploration and visualization, any insights gained, and your modeling approach. The *"Results"* presents the modeling results and discusses the model performance. The *"Conclusions"* section gives a brief summary of the report, its limitations and future work.

The remainder of this section first present the data set, next analyze the dataset properties, then defines the goal of this project more concretely, and finally identifies the main steps that have been taken to reach the goal.

Dataset overview

As stated in the README of the original MovieLens 10M dataset, it contains `10000054` ratings and `95580` tags (not used in this study) applied to `10681` movies by `71567` users of the MovieLens service.

All ratings are contained in the file `ml-10M100K/ratings.dat`. Each line of this file represents one rating specified by the rating value (`rating`), the movie identifier (`movieId`), the identifier of the user (`userId`) and the submission timestamp (`timestamp`). Ratings are made on a `5`-star scale, with half-star increments. Timestamps represent seconds since midnight *Coordinated Universal Time (UTC) of January 1, 1970.*

Movie information is contained in the file `ml-10M100K/movies.dat`. Each line of this file represents one movie specified by its identifier (`movieId`), movie title (`title`) and the corresponding genres (`genres`). The movie titles are entered manually, so errors and inconsistencies may exist. Genres are pipe-separated lists, of the individual genres from the following set:

("Action", "Adventure", "Animation", "Children's", "Comedy", "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror", "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western").

In this project, we use a pre-processed subset of the original dataset, using only the data from the `ratings.dat` and `movies.dat` files thereof. To facilitate supervised learning, the data is pre-processed in the following way:

1. The `ratings.dat` and `movies.dat` are loaded into `ratings` and `movies` data frames
    1. `ratings` – with columns named userId, movieId, rating, timestamp
    2. `movies` – columns named movieId, title, genres
2. The ratings are coupled with the movies information by the `movieId` into a new `movielens` data frame
3. The `movielens` data frame is randomly split into two parts:
    2. `edx` – a training set storing the `90%` of the observations from the `movielens`
    3. `validation` – a testing set storing the `10%` of the observations from the `movielens`
4. The `validation` set is filtered to only contain movies ans users present in `edx`
5. The observations filtered out in the previous step are added back to the `edx` set

For more details, see the `create_movielens_sets` function located in the `movielens_project.R` script. As a result of pre-processing we have two sets with the following metrics, the training set `edx`:

```
## # A tibble: 1 x 3
##   num_observations num_movies num_users
##              <int>      <int>     <int>
## 1          9000055      10677     69878
```

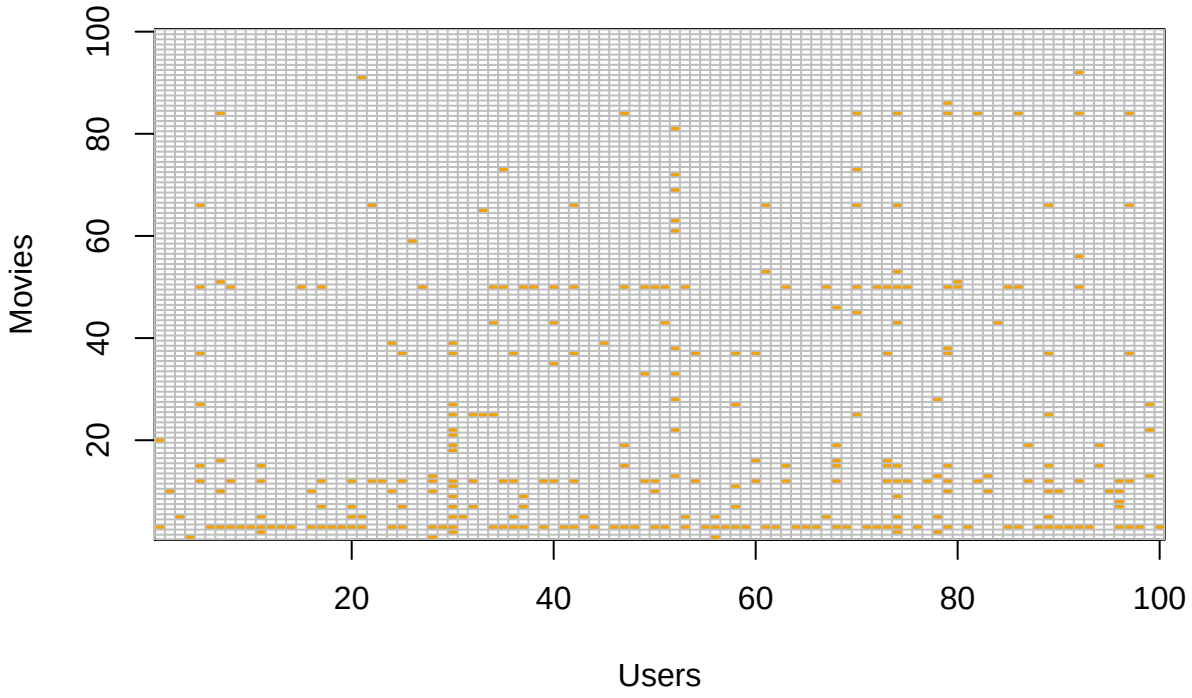
and the testing set `validation`:

```
## # A tibble: 1 x 3
##   num_observations num_movies num_users
##              <int>      <int>     <int>
## 1           999999       9809     68534
```

As one can see the `edx` set stores 90% of the total number of observations (`num_observations`) and the `validation` set contains 10%. The difference in the numer of distinct movies (`num_movies`) and the number of distinct users (`num_users`) is explained by the observations being moved back from `validation` to `edx` in the last data preprocessing step, described above.

Dataset analysis

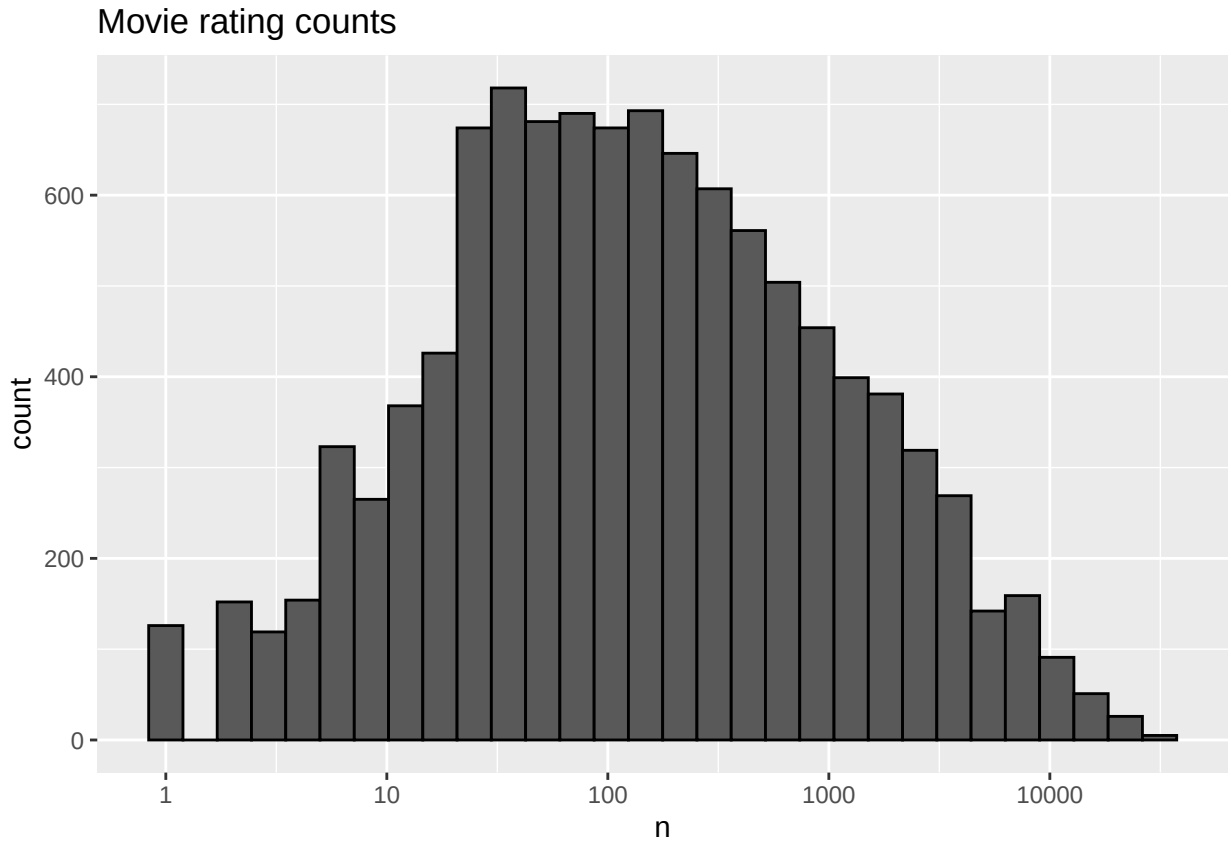
Note that, from the number of distinct user, movies and observations (ratings) in the `edx` set it is clear that not every user has rated every movie as 69878 * 10677 $<$ 9000055. The same can be seen via the next plot for randomly sampled `100` unique users and `100` unique movies:



The the task of a recommendation system we are to build can be seen as filling in the missing ratings. To stress the complexity of the undertaking, below we list a number of different circumstances that can influence rating predictions, and may need to be taken into account when building the statistical model.

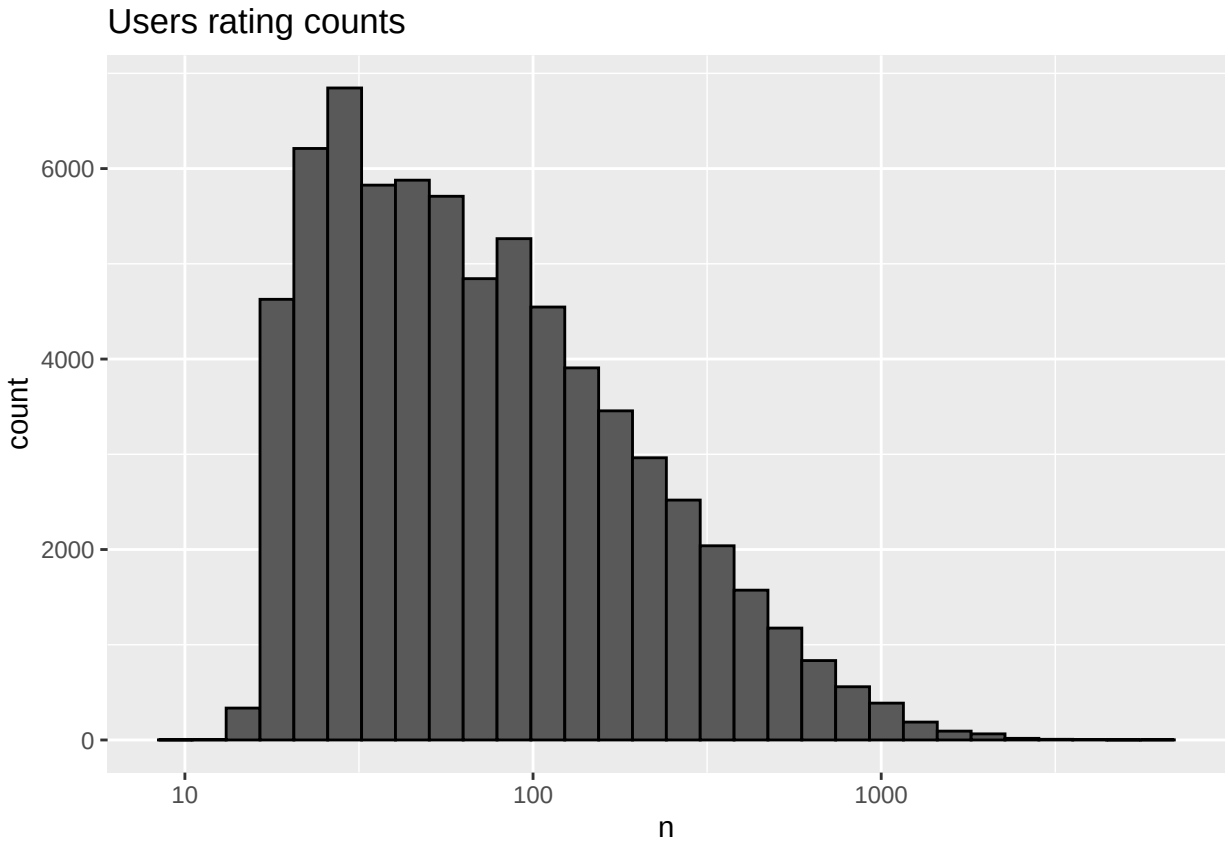
(I) Some movies are rated more often than the others:

```
movielens_data$edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movie rating counts")
```



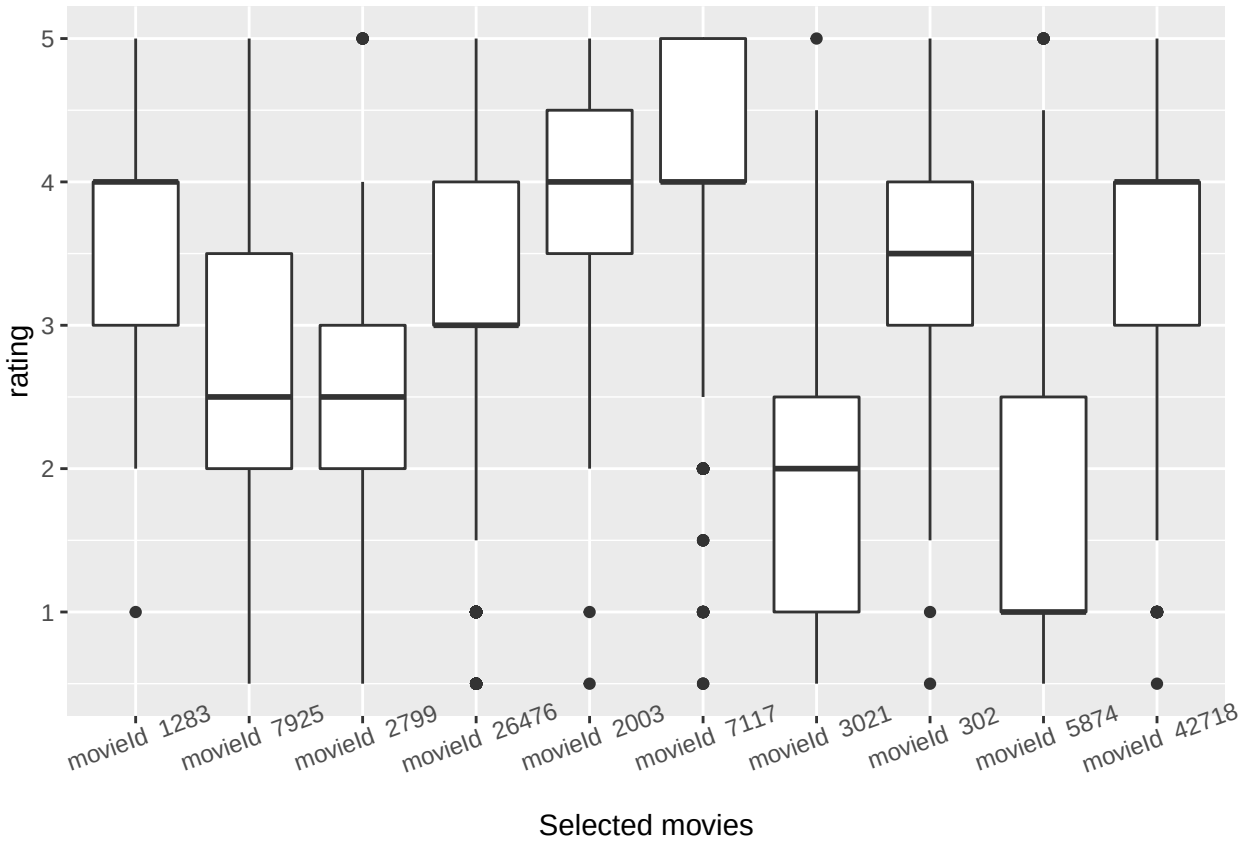
(II) Some users are more active in rating that the others:

```
movielens_data$edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Users rating counts")
```



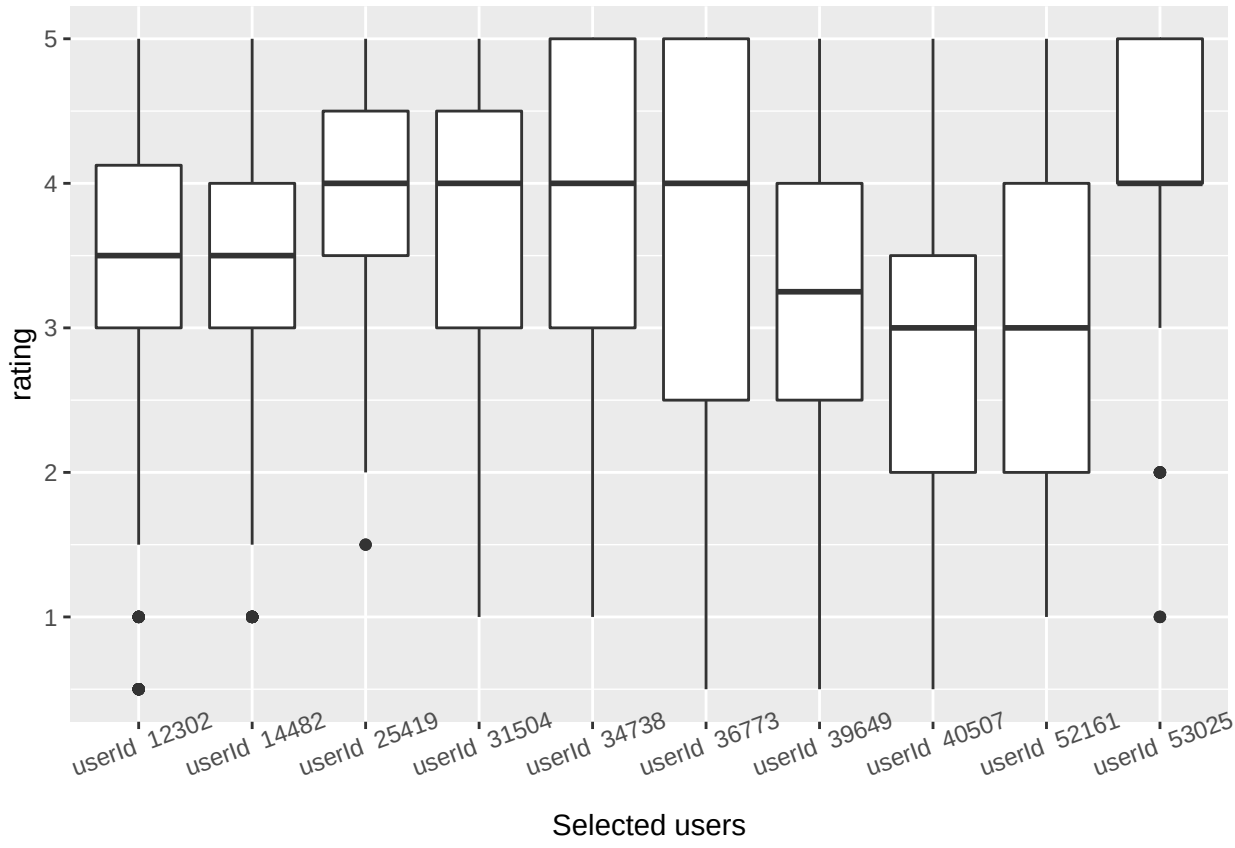
(III) Movies differ from eachother in given ratings, lets consider 10 movies with at least 100 ratings each:

```
#Sample 10 individual movie ids for the movies with more than 100 ratings
movie_ids <- movielens_data$edx %>%
  group_by(movieId) %>% summarise(cnt = n()) %>%
  filter(cnt > 100) %>% pull(movieId) %>% unique(.)
set.seed(1)
sample_movie_ids <- sample(movie_ids, 10)

#Use the boxplot to show variations in ratings between different movies
movielens_data$edx %>%
  filter(movieId %in% sample_movie_ids) %>%
  ggplot(aes(title, rating, group=movieId)) +
  geom_boxplot() +
  scale_x_discrete(labels = paste("movieId ",as.character(sample_movie_ids)))+
  theme(axis.text.x = element_text(angle = 20)) +
  labs(x = "Selected movies")
```



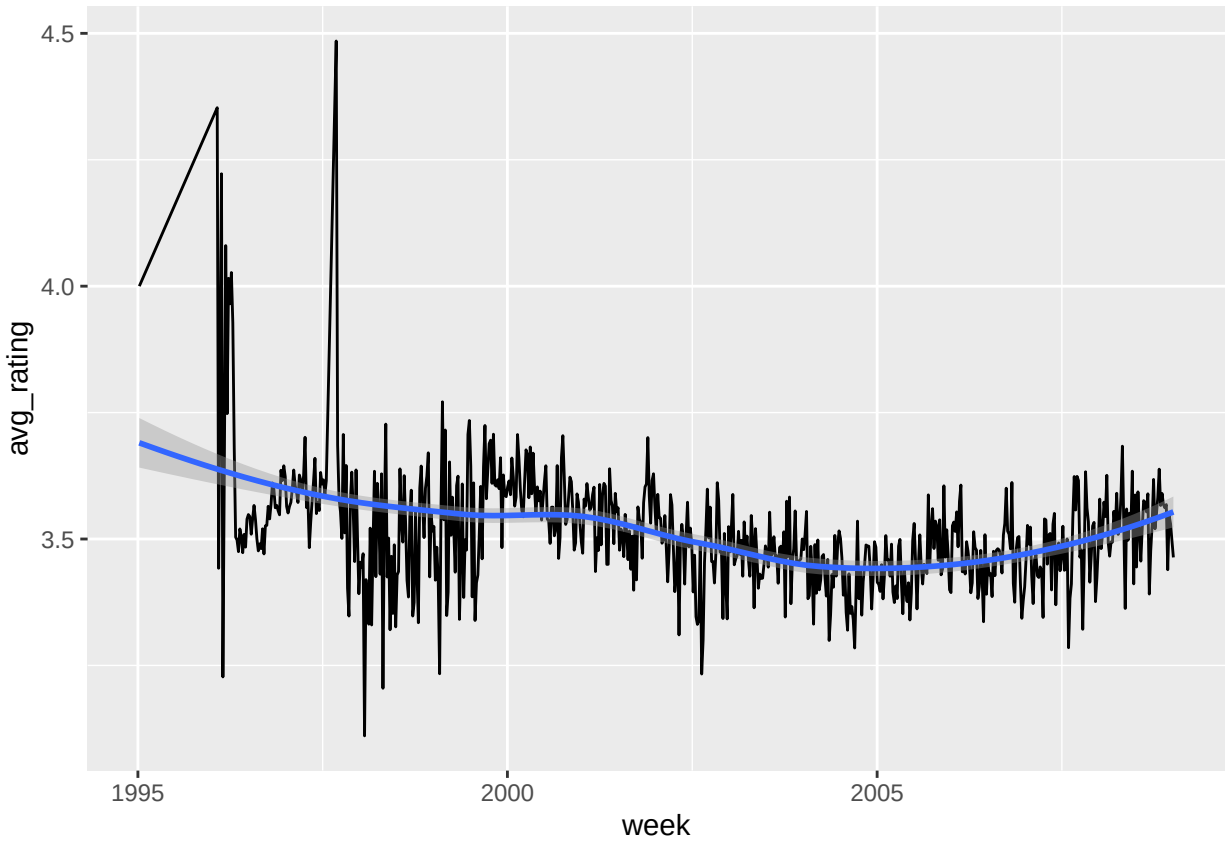
(IV) Users differ from each other in giving ratings, lets consider 10 users with at least 100 ratings each:

```
#Sample 10 individual user ids for the users with more than 100 ratings
user_ids <- movielens_data$edx %>%
  group_by(userId) %>% summarise(cnt = n()) %>%
  filter(cnt > 100) %>% pull(userId) %>% unique(.)
set.seed(1)
sample_user_ids <- sample(user_ids, 10)

#Use the boxplot to show variations in ratings between different users
movielens_data$edx %>%
  filter(userId %in% sample_user_ids) %>%
  mutate(userName = paste("userId ", as.character(userId))) %>%
  group_by(userId) %>%
  ggplot(aes(userName, rating, group=userId)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 20)) +
  labs(x = "Selected users")
```



(V) The rating scores depend on the time when the rating is given:

```
#Convert the timestamp into a week (date) and then plot
#the mean rating for all the movies per week
movielens_data$edx %>%
  mutate(date = as_datetime(timestamp), week = round_date(date, "week")) %>%
  group_by(week) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(week, avg_rating)) +
  geom_line() +
  geom_smooth(method = 'loess')
```



(VI) The best and the worst (on-average) movies were not rated very often:

```
#Get the movies with the top 10 average scores
true_top_10 <- avg_movie_ratings %>%
  arrange(desc(avg_rating)) %>%
  slice(1:10)
true_top_10
```

```
## # A tibble: 10 x 3
##    movieId avg_rating num_ratings
##      <dbl>      <dbl>       <int>
##  1    3226       5               1
##  2   33264       5               2
##  3   42783       5               1
##  4   51209       5               1
##  5   53355       5               1
##  6   64275       5               1
##  7    5194       4.75            4
##  8   26048       4.75            4
##  9   26073       4.75            4
## 10   65001       4.75            2
```

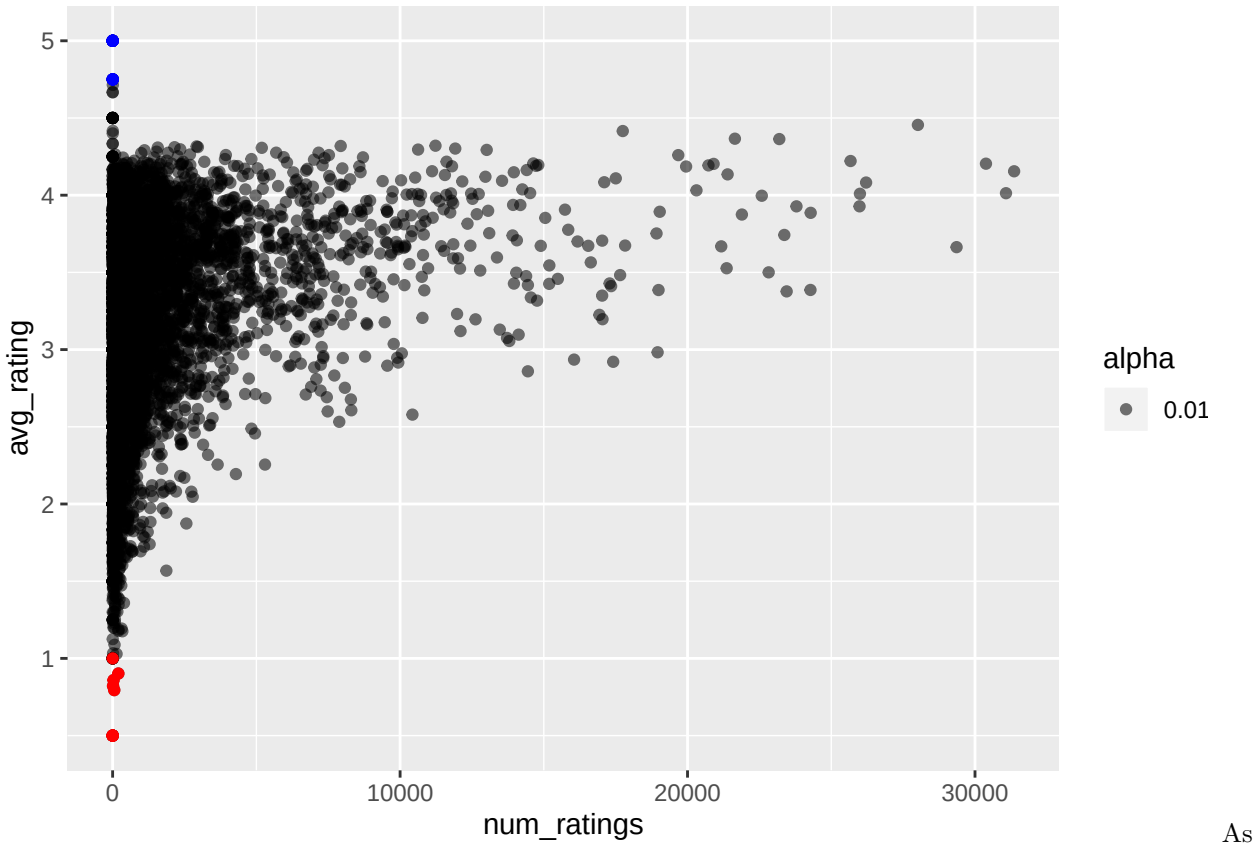
```
#Get the movies with the bottom 10 average scores
true_bottom_10 <- avg_movie_ratings %>%
  arrange(avg_rating) %>%
  slice(1:10)
true_bottom_10
```

```
## # A tibble: 10 x 3
```

```
##    movieId avg_rating num_ratings
##      <dbl>      <dbl>       <int>
##  1    5805      0.5             2
##  2    8394      0.5             1
##  3   61768      0.5             1
##  4   63828      0.5             1
##  5   64999      0.5             2
##  6    8859      0.795          56
##  7    7282      0.821          14
##  8   61348      0.859          32
##  9    6483      0.902         199
## 10     604      1               2
```

See also the next plot where the best movies are marked as blue and the worst as red:

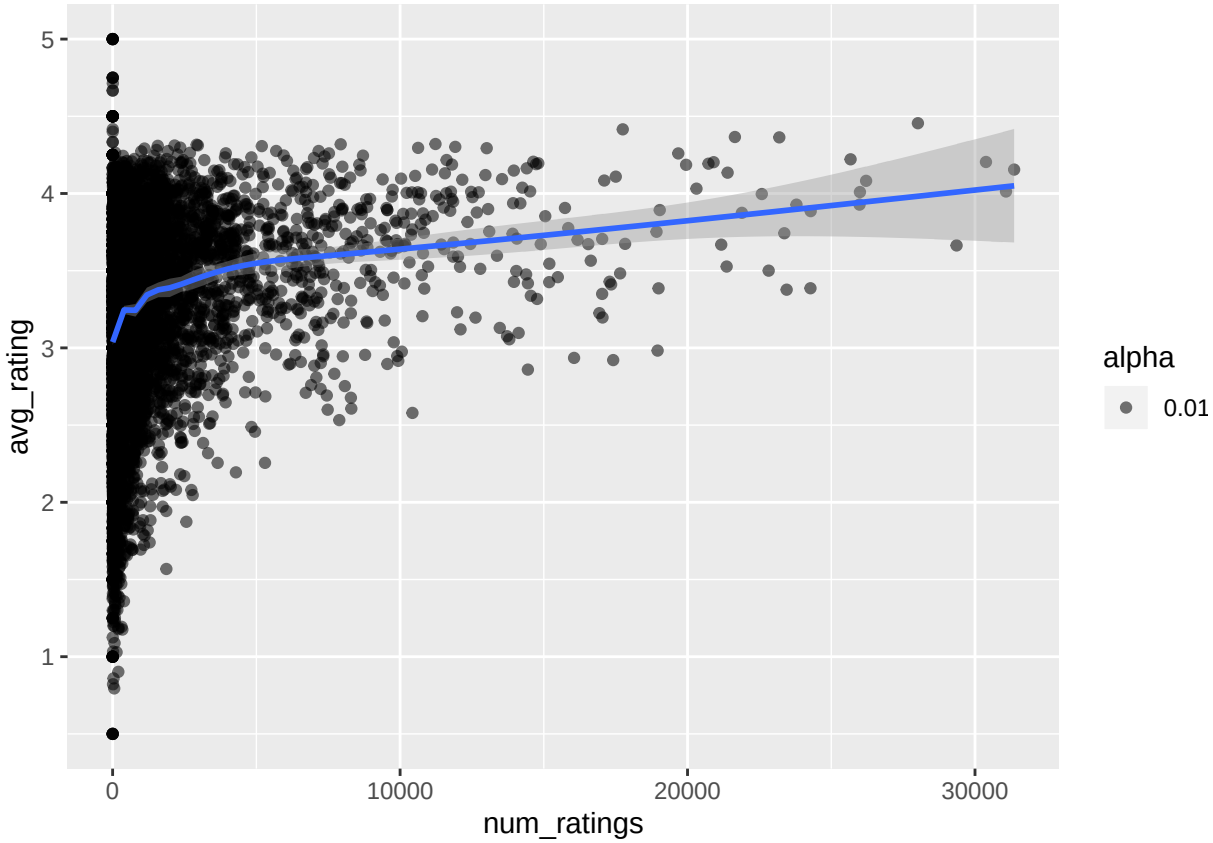
```
#Plot the movies
avg_movie_ratings %>%
  ggplot(aes(x = num_ratings, y = avg_rating)) +
  geom_point(aes(color = I("black"), alpha=0.01))+
  geom_point(data = true_top_10, aes(num_ratings, avg_rating), color="blue") +
  geom_point(data = true_bottom_10, aes(num_ratings, avg_rating), color="red")
```



As we see from the plot above there is more variability in the movies with fewer ratings but above that, the movies that are rated more often also seem to be on-average rated higher:

```
#Plot the movies
avg_movie_ratings %>%
  ggplot(aes(x = num_ratings, y = avg_rating)) +
  geom_point(aes(color = I("black"), alpha=0.01)) + geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Project goal

Execution plan

Analysis

Results

Conclusions