HarvardX: PH125.9x Data Science
MovieLens_Predicing Ratings Project

Tran Phu Hoa

March 25, 2019

Contents

1 Introduction

The main purpose of writing this report is to apply what I have learnt about Data Science in eight previous courses of HarvardX Data Science series. It will report the solution of the first challenge of its final course named Data Science: "Capstone".

The report is expected to replicate a full process of a Data Scientist doing a project, and it will adapt part of the structure which is recommended by Dr. Roger D.Peng in his book called "Report Writing for Data Science in R". Thus, those steps are.

1. Defining the question
2. Defining the ideal dataset
3. Obtaining the data
4. Cleaning the data
5. Exploratory data analysis
6. Statistical prediction/modelling

7. Interpretation of results
8. Challenging of results
9. Synthesis and write up
10. Creating reproducible code

2 Defining the question

The main goal is to find out a model that can make a prediction for the scores rated by users for different movies. The Root Mean Square Error (RMSE) will be used to evaluate how well models perform. Models have the lowest RMSE will rank in first place as the best predicting model.

3 Defining the ideal dataset

The dataset will be the 10M version of the MovieLens dataset. In order to obtain this dataset, the project will apply codes provided by the course as below.

```
##########################################################
# Create edx set, validation set, and submission file
##########################################################

# 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")

# 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"))

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")
```

In addition, the course gives codes to make validation sets as below.

```
# Validation set will be 10% of MovieLens data

set.seed(1)
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)
edx <- rbind(edx, removed)

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

The "edx" subset will be the main dataset for this analysis and modelling, while the "validation" subset will be used to calculate RMSE for final algorithm.

Save the dataset to the local system, so that it saves the loading times for further analysis.

```
save(edx, file = "edx.RData")

save(validation, file = "validation.RData")
```

4 Exploring the "edx" dataset

Loading useful packages

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

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

Take a look at the dataset using "glimpse" function in "tidyverse" package, and it shows that the "edx" dataset has 9,000,055 observations and 6 variables. There are three types of data including integer, double and character. In fact, "userId" and "timestamp" are coded in integer, "movieId" and "rating" are double, and "title" and "genres" are character.

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|D...
```

Look for NA as the missing value in the dataset using "summary" function. The result indicates that there is no missing value in the "edx" dataset.

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

There are about 70,000 unique users, and 797 groups of genres. The rates are continuous variables, however, they would be classified as categorical variables with 10 different scores. The number of distinct movie is expected to be the same with the number of titles, however it seems that two movies had the same name.

```
##   n_users n_movies n_rating n_title n_genres n_timestamp
## 1   69878    10677       10   10676      797     6519590
```

The dataset does not seem to be in the tidy format. The first reason is more than one variables are included in the same column, for example "Boomerang (1992)" can be divided into two columns which are "Movie title" and "Released year". Furthemore, the column "timestamp" looks meaningless, and it should be transformed to date format. In "genres" column, instead of clumping those genres together, it may be better to separate them.

The following activities will tidy up the "edx" dataset.

Separate "title" column into "title" and "movie_year"

```
#Select object "year" in the "title" column, and create "movie_year" variable
tidy_edx <- tidy_edx_date %>%
  mutate(movie_year = str_extract(tidy_edx_date$title, regex("\\(\\d{4}\\)")))

#Detect "year" in the title column
tidy_edx <- tidy_edx %>%
  mutate(movie_year = str_extract(tidy_edx$movie_year, regex("\\d{4}")))
```

```
tidy_edx <- tidy_edx %>%
  mutate(movie_year = as.integer(movie_year), movieId = as.integer(movieId))
```

Save tidy_edx to the local system.

Remove unnecessary dataset "tidy_edx_date", "tidy_edx_sep_genres"

```
rm("tidy_edx_date", "tidy_edx_sep_genres")

gc()
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  11059676  590.7   35700160 1906.6  34488676 1841.9
## Vcells 219543715 1675.0  461757116 3523.0 439605848 3354.0
```

Take a look at the dataset

```
glimpse(tidy_edx)
```

```
## Observations: 23,371,423
## Variables: 7
## $ userId         <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ movieId        <int> 122, 122, 185, 185, 185, 292, 292, 292, 292, 31...
## $ rating         <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,...
## $ title          <chr> "Boomerang (1992)", "Boomerang (1992)", "Net, T...
## $ genres         <chr> "Comedy", "Romance", "Action", "Crime", "Thrill...
## $ date_of_rating <dttm> 1996-08-02 11:24:06, 1996-08-02 11:24:06, 1996...
## $ movie_year     <int> 1992, 1992, 1995, 1995, 1995, 1995, 1995, 1995,...
```

Identify any missing values

```
summary(tidy_edx)
```

```
##      userId         movieId          rating         title
##  Min.   :    1   Min.   :    1   Min.   :0.500   Length:23371423
##  1st Qu.:18140   1st Qu.:  616   1st Qu.:3.000   Class :character
##  Median :35784   Median : 1748   Median :4.000   Mode  :character
##  Mean   :35886   Mean   : 4277   Mean   :3.527
##  3rd Qu.:53638   3rd Qu.: 3635   3rd Qu.:4.000
##  Max.   :71567   Max.   :65133   Max.   :5.000
##     genres          date_of_rating                movie_year
##  Length:23371423    Min.   :1995-01-09 11:46:49   Min.   :1915
##  Class :character   1st Qu.:2000-01-06 22:51:28   1st Qu.:1987
##  Mode  :character   Median :2003-01-03 00:17:38   Median :1995
##                     Mean   :2002-10-20 01:19:24   Mean   :1990
##                     3rd Qu.:2005-11-03 12:57:53   3rd Qu.:1998
##                     Max.   :2009-01-05 05:02:16   Max.   :2008
```

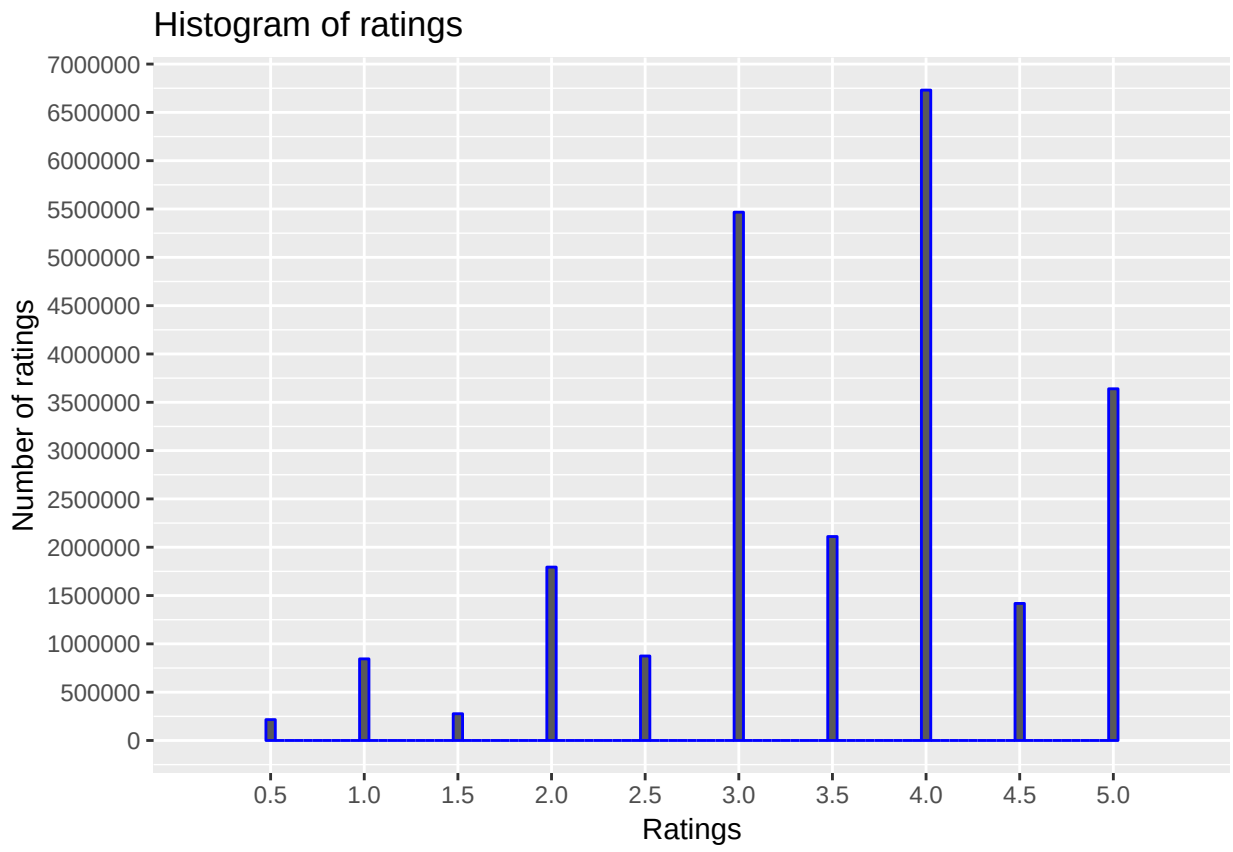
There are about 20 different genres and 90 unique released years for 10,676 movie titles.

```
tidy_edx %>% summarize(n_users = n_distinct(userId),
                  n_movies = n_distinct(movieId),
                  n_rating = n_distinct(rating),
                  n_title = n_distinct(title),
                  n_genres = n_distinct(genres),
                  n_date = n_distinct(date_of_rating),
                  n_movieyear = n_distinct(movie_year))
```

```
##   n_users n_movies n_rating n_title n_genres  n_date n_movieyear
## 1   69878    10677       10   10676       20 6519590          94
```

Plot the histogram of rating

```
tidy_edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.05, color = "blue") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 7000000, 500000))) +
  xlab("Ratings") +
  ylab("Number of ratings") +
  ggtitle("Histogram of ratings")
```



"4.0" score had the highest number of ratings (6,730,401), and "0.5" had the lowest one (215,932).

```
tidy_edx %>% group_by(rating) %>%
  summarize(n_rating = n()) %>%
  arrange(desc(n_rating))
```

```
## # A tibble: 10 x 2
##    rating n_rating
##     <dbl>    <int>
##  1    4    6730401
##  2    3    5467061
##  3    5    3639511
##  4    3.5  2110690
##  5    2    1794243
##  6    4.5  1418248
##  7    2.5   874290
##  8    1     844336
##  9    1.5   276711
## 10    0.5   215932
```

Summary statistic of rating

```
tidy_edx %>% summarize(mean_rating = mean(rating),
                       median_rating = median(rating),
                       sd_rating = sd(rating))
```

```
##   mean_rating median_rating sd_rating
## 1    3.527019             4  1.052262
```

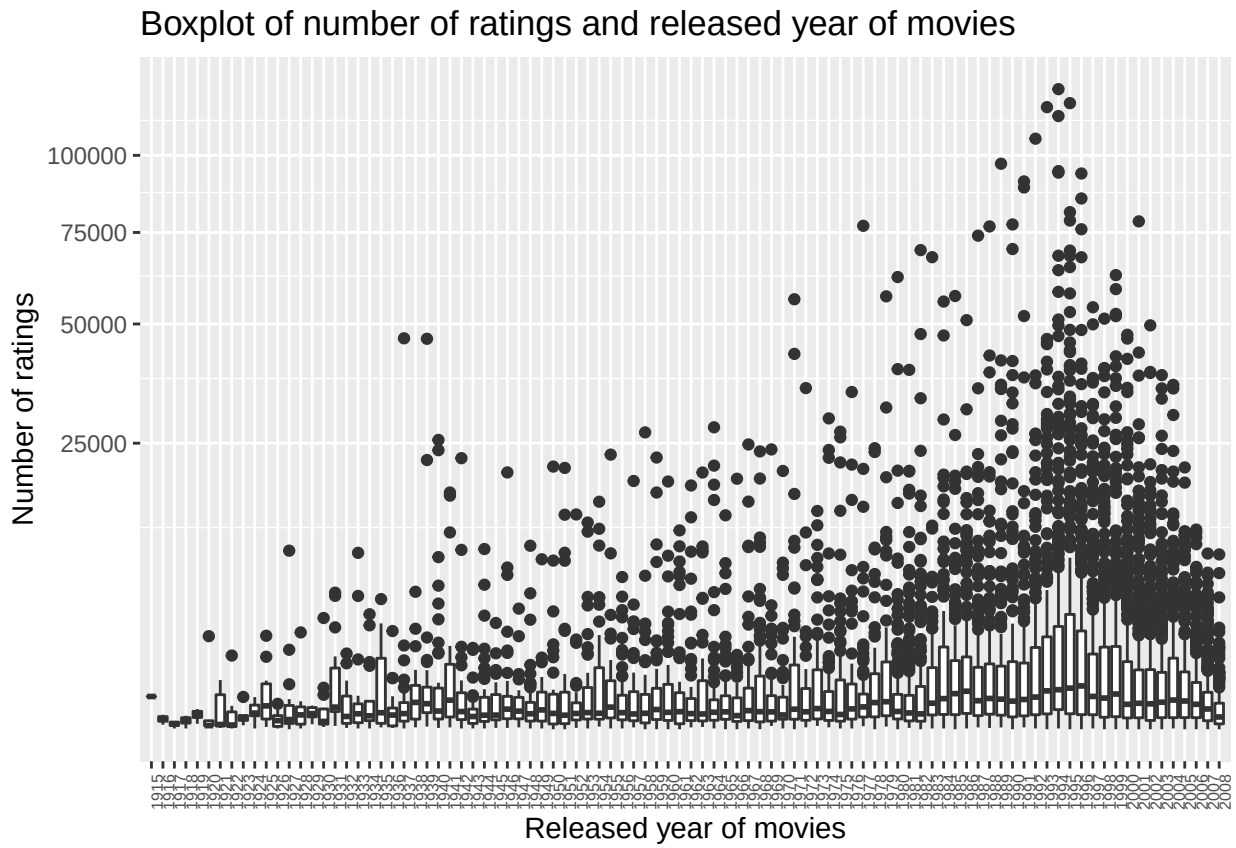
```
tidy_edx %>% group_by(movieId, title) %>%
  summarize(n_movie = n()) %>%
  arrange(desc(n_movie))
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                n_movie
##      <int> <chr>                                  <int>
##  1     356 Forrest Gump (1994)                   124316
##  2       1 Toy Story (1995)                      118950
##  3     480 Jurassic Park (1993)                  117440
##  4     380 True Lies (1994)                      114115
##  5     588 Aladdin (1992)                        105865
##  6     592 Batman (1989)                          97108
##  7     364 Lion King, The (1994)                  94605
##  8     296 Pulp Fiction (1994)                    94086
##  9     780 Independence Day (a.k.a. ID4) (1996)   93796
## 10     593 Silence of the Lambs, The (1991)       91146
## # ... with 10,667 more rows
```

Compute the number of ratings for each movie and then plot it against the year the movie came out. Use the square root transformation on the counts.

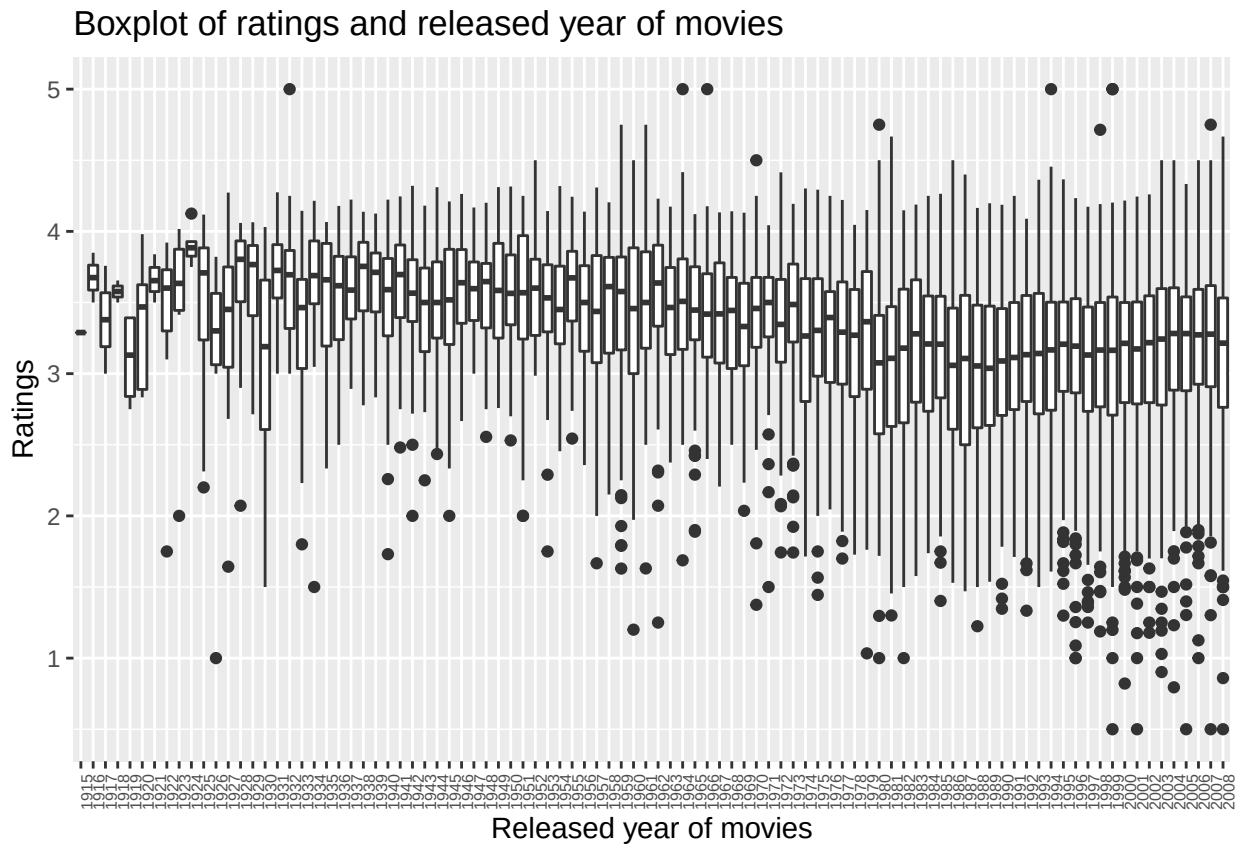
Boxplot of number of ratings and released year of movies

```
tidy_edx %>% group_by(movieId) %>%
  summarize(n_ratings = n(),
            year = as.character(first(movie_year))) %>%
  qplot(year, n_ratings, data = ., geom = "boxplot") +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 6)) +
  labs(title = "Boxplot of number of ratings and released year of movies",
       x = "Released year of movies",
       y = "Number of ratings")
```



Boxplot of ratings and released year of movies

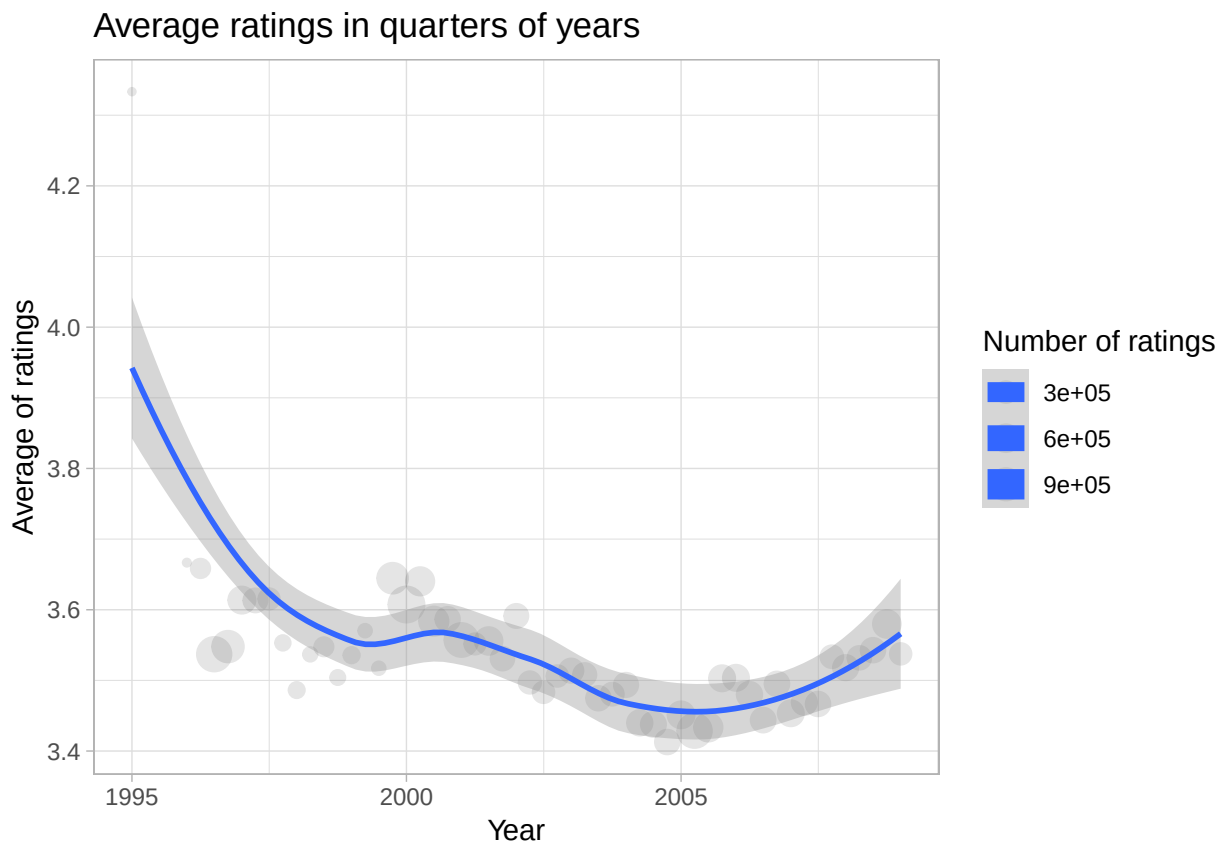
```
tidy_edx %>% group_by(movieId) %>%
  summarize(mean_rating = mean(rating),
            year = as.character(first(movie_year))) %>%
  qplot(year, mean_rating, data = ., geom = "boxplot") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 6)) +
  labs(title = "Boxplot of ratings and released year of movies",
       x = "Released year of movies",
       y = "Ratings")
```



Average ratings in quarters of years

```
tidy_edx %>% mutate(date = round_date(date_of_rating, unit = "quarter")) %>%
  group_by(date) %>%
  summarize(mean_rating = mean(rating), n_rating = n()) %>%
  ggplot(aes(date, mean_rating, size = n_rating)) +
  geom_point(alpha = 0.1) +
  scale_size(name  = "Number of ratings") +
  geom_smooth() +
  theme_light() +
  labs(title = "Average ratings in quarters of years",
       x = "Year",
       y = "Average of ratings")
```

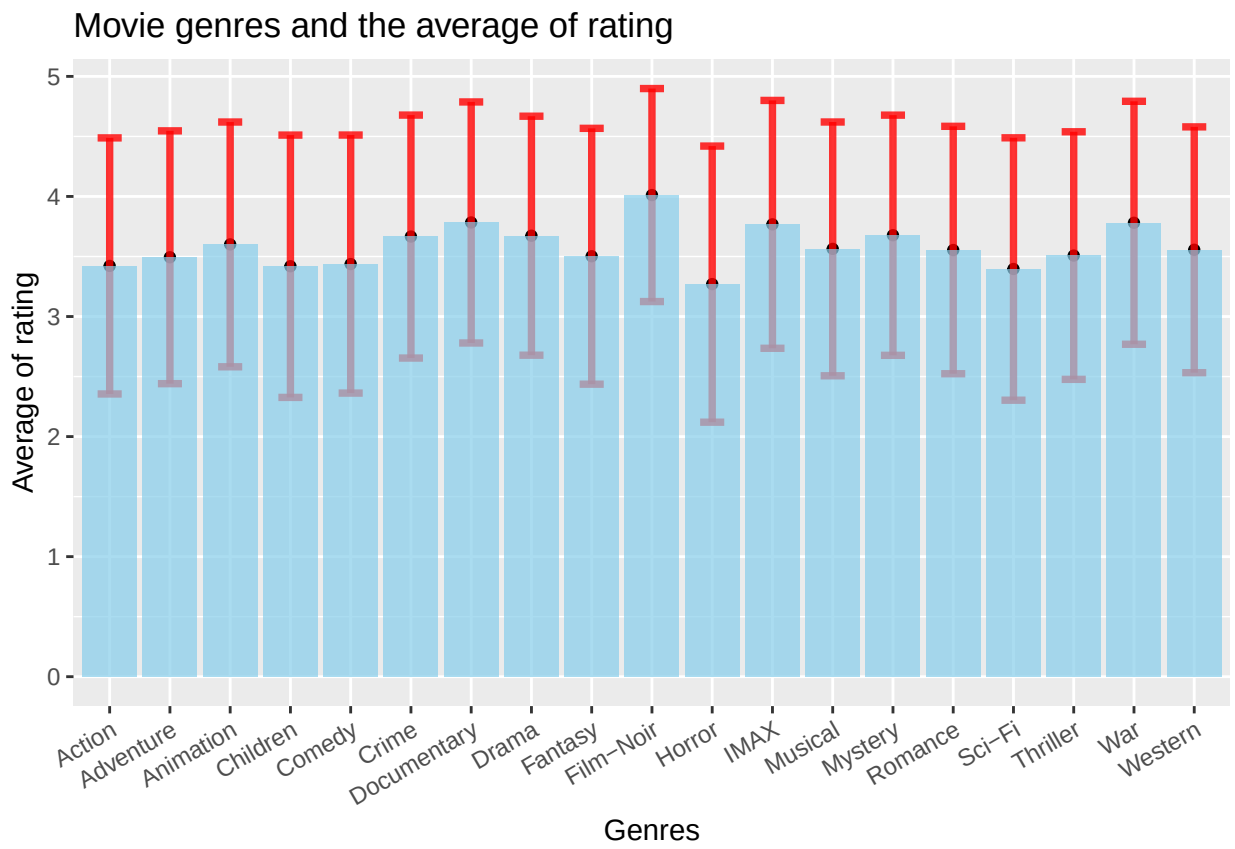
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



"Genres" seems to effect how users rate, for example, "Film-noir" was scored the highest rate (4.2 points) on average. "Horror" movies had smallest score which was about 0.8.

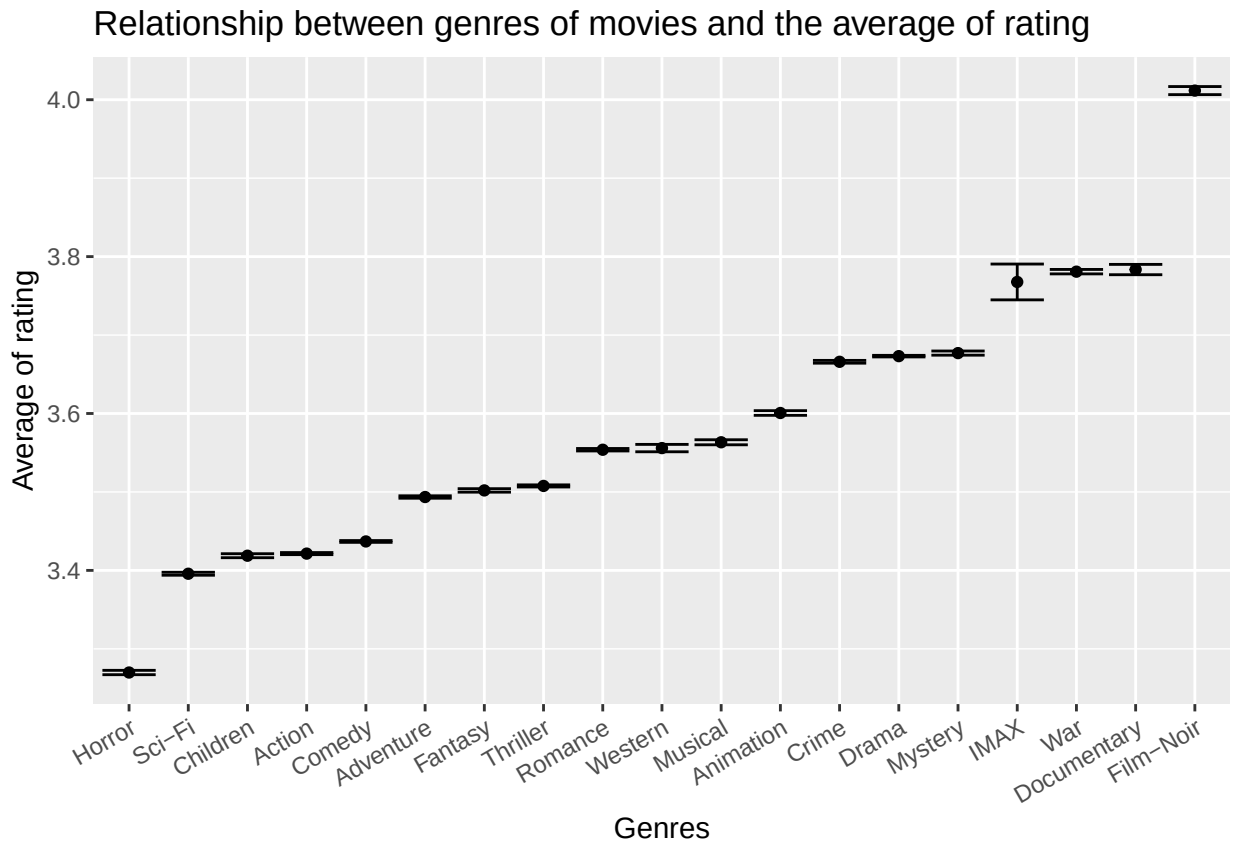
A first look at relationship between genres of movies and the average of rating

```
tidy_edx %>% group_by(genres) %>%
  summarize(n_rating = n(),
            avg_rating = mean(rating),
            se_rating = sd(rating)) %>%
  filter(n_rating >= 1000) %>%
  arrange(desc(n_rating)) %>%
  ggplot (aes(x=genres, y=avg_rating)) +
  geom_point() +
  geom_errorbar(aes(ymin=avg_rating - se_rating,
                    ymax=avg_rating + se_rating),
                width=0.4, colour="red",
                alpha=0.8, size=1.3) +
  geom_bar(stat="identity", fill="skyblue", alpha=0.7) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1)) +
  labs(title = "Movie genres and the average of rating",
       x = "Genres",
       y = "Average of rating")
```



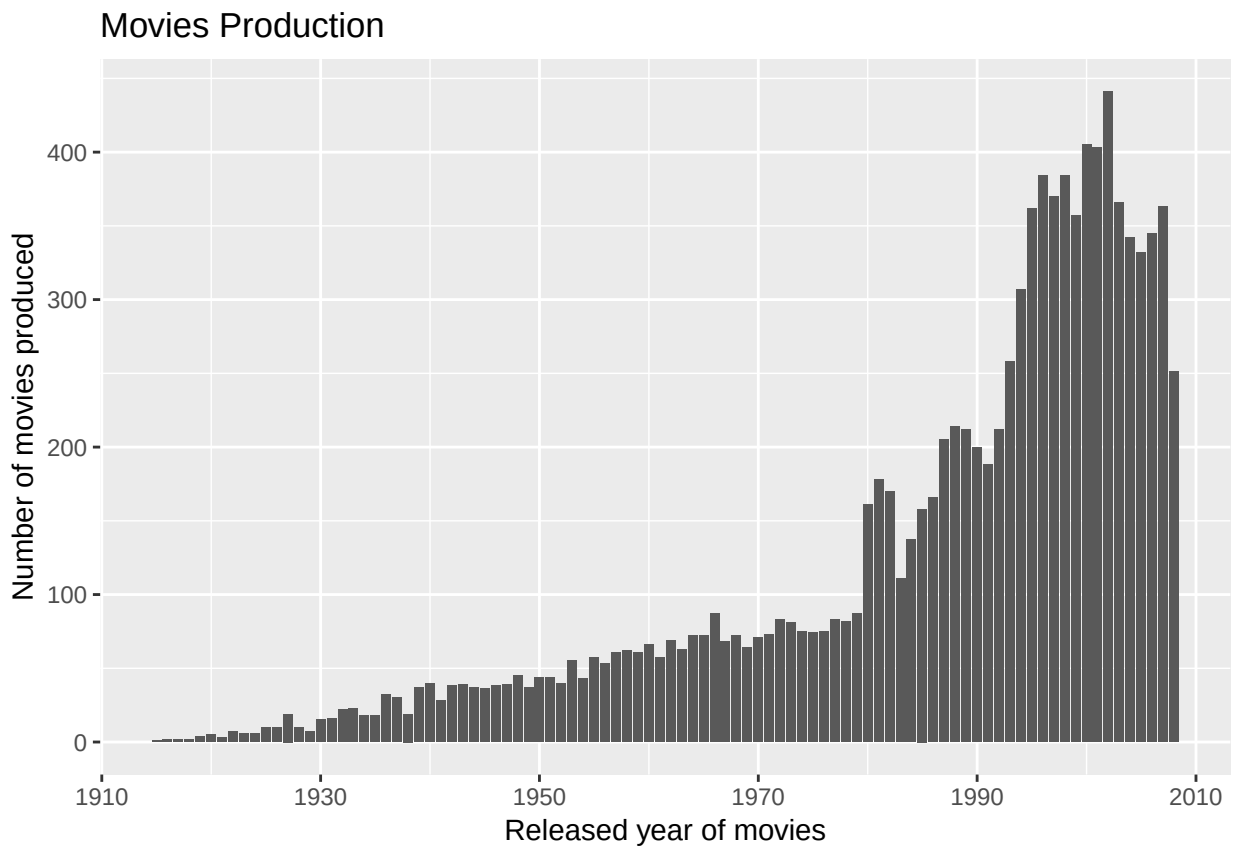
Other view at the relationship between genres of movies and the average of rating

```
tidy_edx %>% group_by(genres) %>%
  summarize(n = n(),
            avg_rating = mean(rating),
            se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000) %>%
  mutate(genres = reorder(genres, avg_rating)) %>%
  ggplot(aes(x = genres,
             y = avg_rating,
             ymin = avg_rating - 2*se,
             ymax = avg_rating + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 30, hjust = 1)) +
  labs(title = "Relationship between genres of movies and the average of rating",
     x = "Genres",
     y = "Average of rating")
```



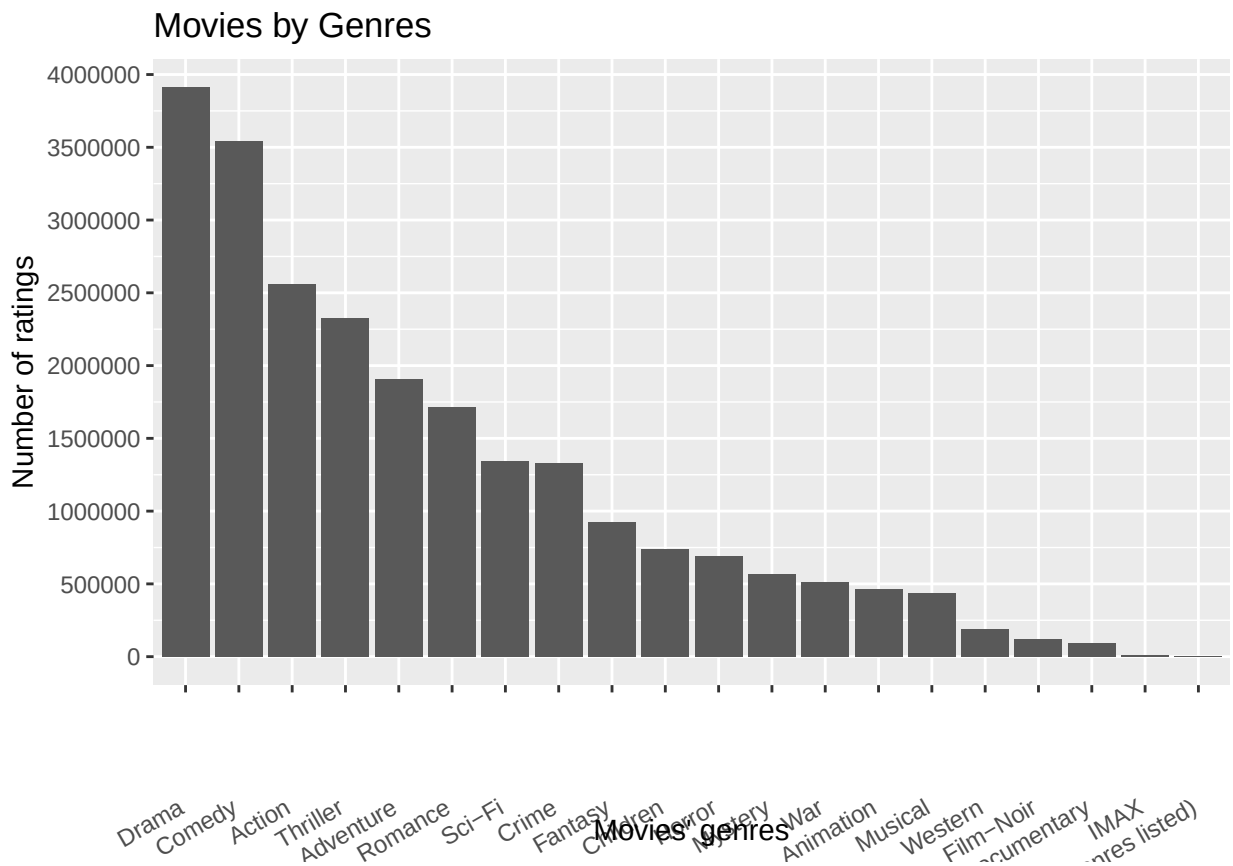
Number of movies produced per year

```
tidy_edx %>%
select(movie_year, movieId) %>%
group_by(movie_year) %>%
summarise(count = n_distinct(movieId)) %>%
arrange(desc(count)) %>%
ggplot(aes(x = movie_year, y = count)) +
geom_bar(stat = "identity") +
labs(title = "Movies Production",
     x = "Released year of movies",
     y = "Number of movies produced")
```



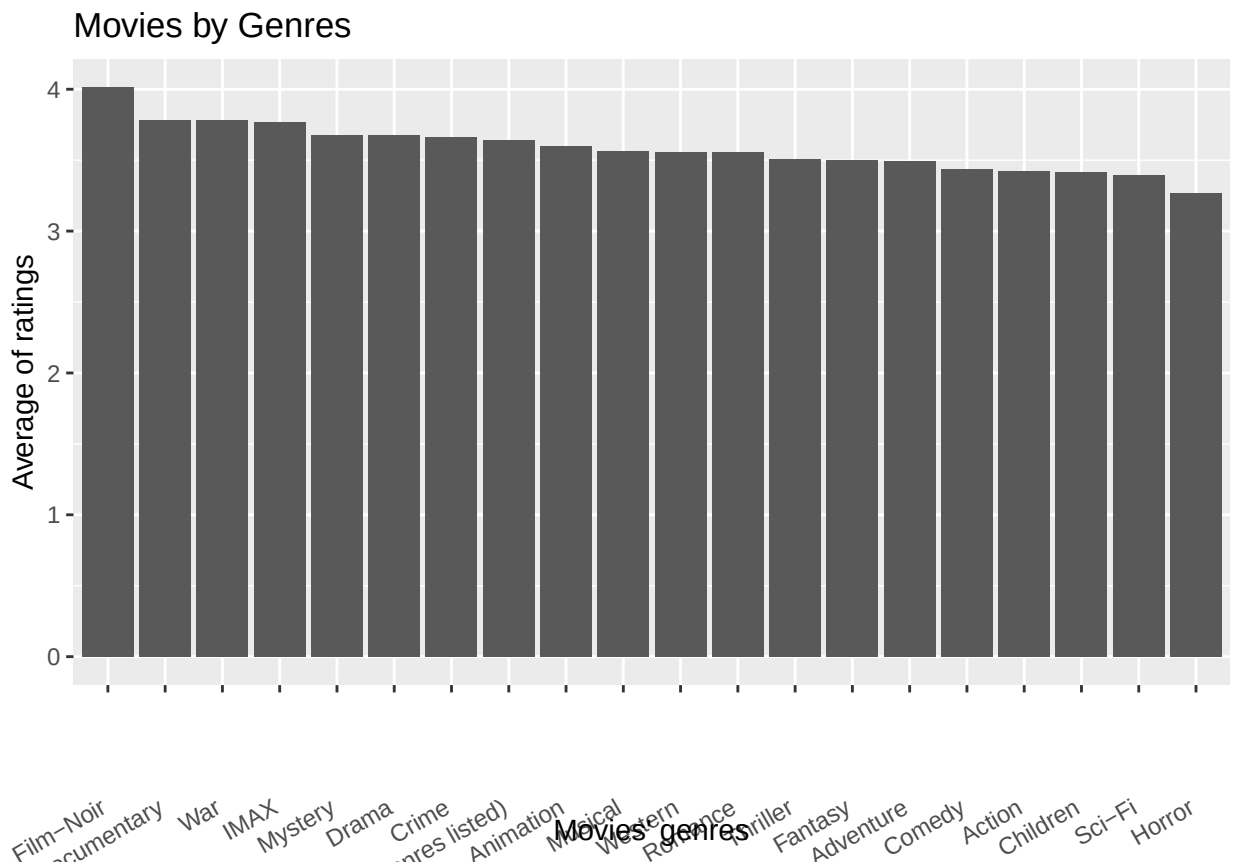
The relationship between movie genres and number of ratings

```
tidy_edx %>%
group_by(genres) %>%
summarise(count = n()) %>%
arrange(desc(count)) %>%
ggplot(aes(x = reorder(genres, -count), y = count)) +
geom_bar(stat = "identity") +
scale_y_continuous(breaks = c(seq(0, 7000000, 500000))) +
theme(axis.text.x = element_text(angle = 30, hjust = 1, vjust = 0)) +
labs(title = "Movies by Genres",
     x = "Movies' genres",
     y = "Number of ratings")
```



The relationship between movie genres and average of ratings

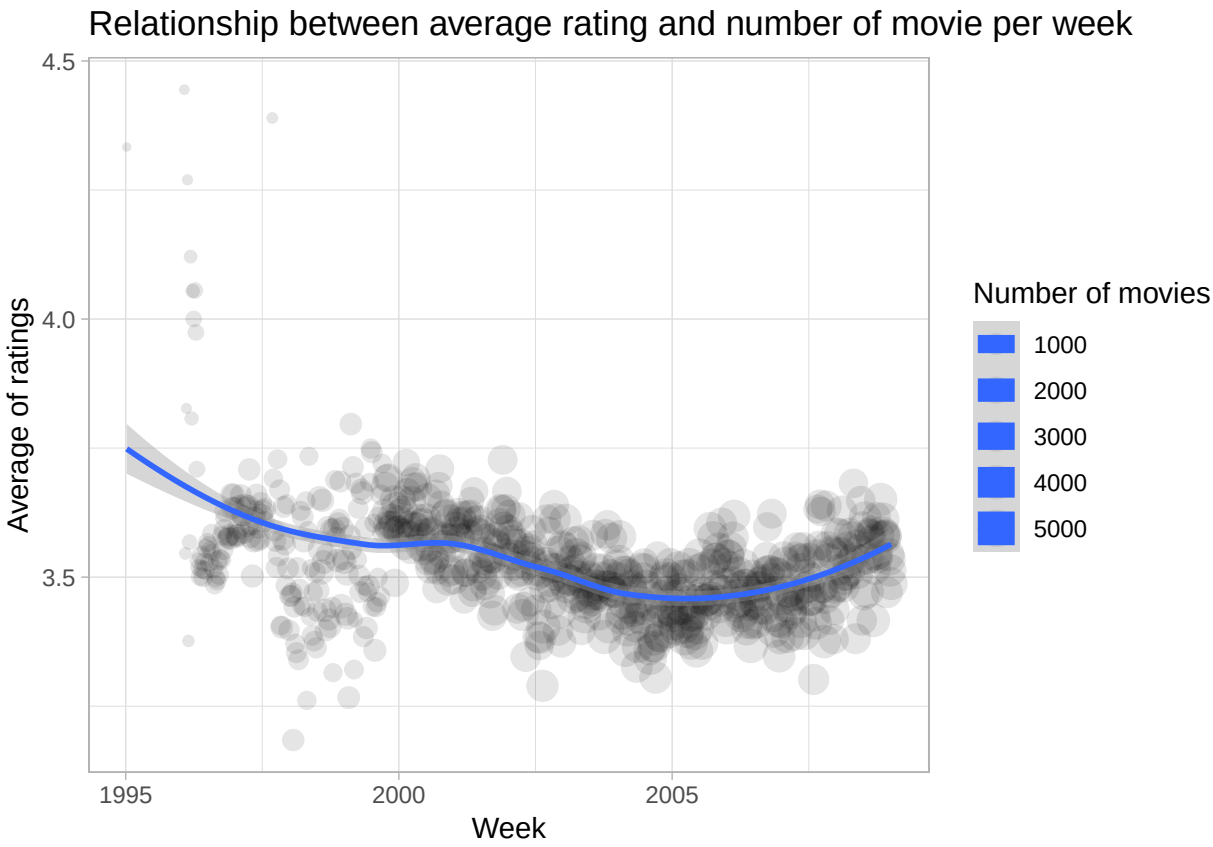
```
tidy_edx %>%
group_by(genres) %>%
summarise(mean_rating = mean(rating)) %>%
arrange(desc(mean_rating)) %>%
ggplot(aes(x = reorder(genres, -mean_rating), y = mean_rating)) +
geom_bar(stat = "identity") +
theme(axis.text.x = element_text(angle = 30, hjust = 1, vjust = 0)) +
labs(title = "Movies by Genres",
     x = "Movies' genres",
     y = "Average of ratings")
```



Exploring the affect of number of movies in a single period on average rating of that interval

```
weekly_rating <- tidy_edx %>%
  mutate(date = round_date(date_of_rating, unit = "week")) %>%
  group_by(date) %>%
  nest() %>%
  mutate(n_movie = map(data, ~n_distinct(.x$movieId)),
         mean_rating = map(data, ~mean(.x$rating)))

weekly_rating <- weekly_rating %>% unnest(n_movie,mean_rating)

weekly_rating %>% ggplot(aes(date, mean_rating, size = n_movie)) +
  geom_point(alpha = 0.1) +
  scale_size(name  = "Number of movies") +
  geom_smooth() +
  theme_light() +
  labs(title = "Relationship between average rating and number of movie per week",
     x = "Week",
     y = "Average of ratings")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```


Relationship between average rating and number of movie per week
Average of ratings
4.5
4.0
3.5
1995
2000
2005
Week
Number of movies
1000
2000
3000
4000
5000

5 Modellling

5.1 RMSE

```
# "Metrics" package will be used to calculate RMSE in this project.
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
# "caret" package will help to create train and test datasets.
library(caret)
```

5.2 Create training data and testing data for machine learning

```
set.seed(2019)

test_index <- createDataPartition(tidy_edx$rating, time = 1, p = 0.8, list = FALSE)

train_data <- tidy_edx %>% slice(test_index)
test_data <- tidy_edx %>% slice(-test_index)
```

Use semi_join() to make all users and movies in the testing data are in the training data

```
test_data <- test_data %>%
  semi_join(train_data, by = "movieId") %>%
  semi_join(train_data, by = "userId")
```

5.3 Matrix factorization approach

5.3.1 Load useful packages

```
library(tidyverse)
library(tictoc)
library(Metrics)
```

5.3.2 Benchmark model

By using the same predited value for all movies and users, this approach expected to be the simplest possible recommendation system. The model will be the baseline for models comparisons.

A model-based approach seems to fix the expectation as its assumption is to predict the same rating for all movies and all users, and those variations in the prediction would be explained by random variables.

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

μ is the true rating for all movies and users. $\epsilon_{u,i}$ are variations contained everythings the model does not explain, and they are assumed to be identical and independent which are sampled from the same distribution centered at zero.

The mean of rating is about 3.5 points.

```
mean_rating <- mean(train_data$rating)

mean_rating
```

```
## [1] 3.527082
```

As all rating will be equal to mean of rating (μ), RMSE of this averaging model will be about 1.061.

```
benchmark_rmse <- rmse(test_data$rating, mean_rating)

benchmark_rmse
```

```
## [1] 1.052582
```

```
save(benchmark_rmse, file = "benchmark.RData")
```

This gives the baseline RMSE to compare with next modelling approaches.

In order to do better than simply predicting the average rating, we incorporate some of insights we gained during the exploratory data analysis.

5.3.3 Movie effect model

The Explanatory analysis suggests that some movies were rated higher than the others, and the next model will capture this effect by adding b_i term which is the average rating for movie i.

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

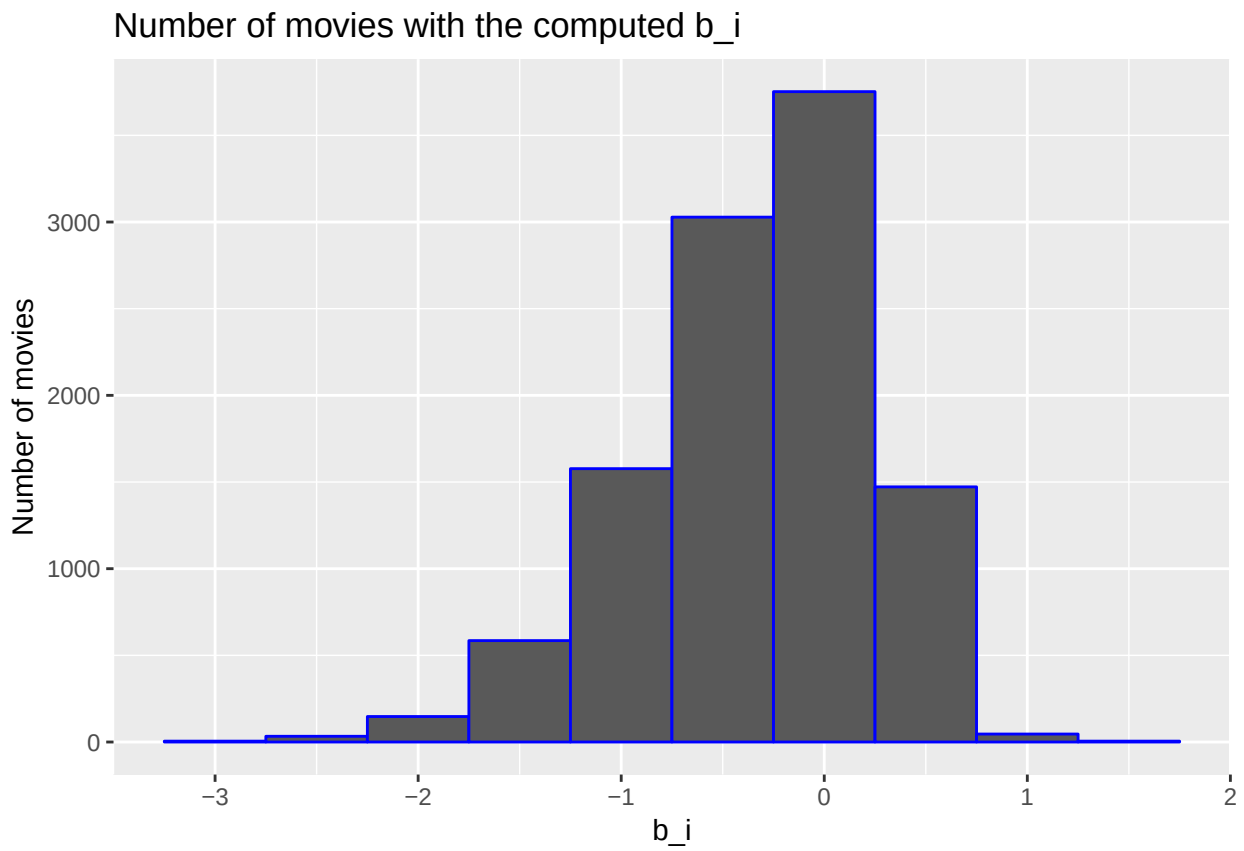
b_i is average rating for movie i. b_i can be calculated by using linear regression function like this.

However, this technique will crash the computer as the dataset is big. In fact, it is more efficient to treat b_i as the function of the mean of rating minus the overall mean for each movie.

```
movie_effect_avgs <- train_data %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mean_rating))
```

Draw the histogram of b_i

```
movie_effect_avgs %>%
  qplot(b_i,
        geom ="histogram",
        bins = 10, data = .,
        color = I("blue"),
        ylab = "Number of movies",
        main = "Number of movies with the computed b_i")
```



Predicted values of ratings in test_date using Movie effect model

```
tic()

movie_effect_predicted_ratings <- mean_rating + test_data %>%
  left_join(movie_effect_avgs, by = "movieId") %>%
  .$b_i

toc()
```

```
## 0.52 sec elapsed
```

Calculate RMSE of the Movie effect model

```
movie_effect_model_rmse <- rmse(movie_effect_predicted_ratings, test_data$rating)

movie_effect_model_rmse
```

```
## [1] 0.9415179
```

```
save(movie_effect_model_rmse, file = "movie_effect_rmse.RData")
```

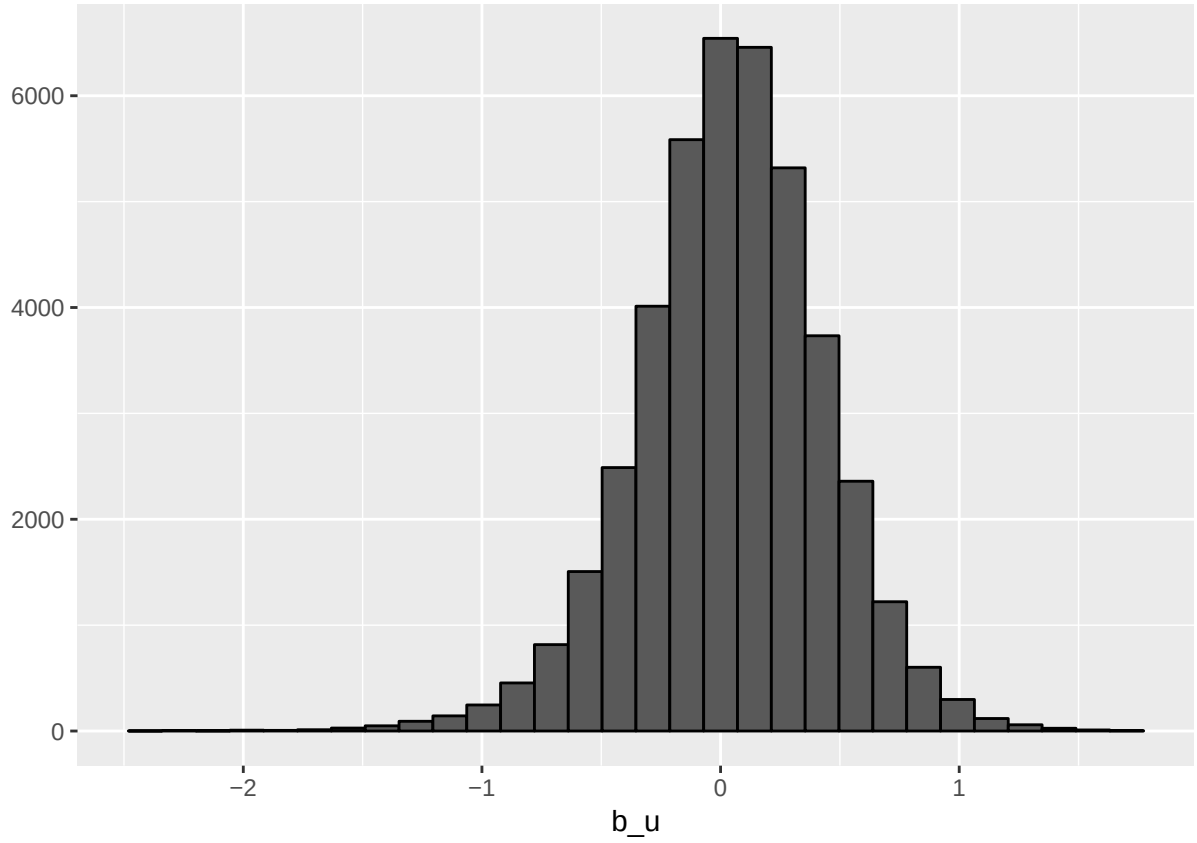
5.3.4 Movie-User effect model

It is observed that each user will rate differently. The graph using the train_data will show this user effect.

Plot the histogram of b_u with users who rated more than 100 times

```
train_data %>%
  left_join(movie_effect_avgs, by='movieId') %>%
```

```
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mean_rating - b_i)) %>%
  qplot(b_u,
        geom ="histogram",
        bins = 30, data = .,
        color = I("black"))
```



The movie-user effect model would be

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

b_u is a user effect.

To avoid potential crash using linear regression approach, the movie-user effect model are computed as the sum of overall mean of rating μ, the movie effect b_i and the user effect $b_{u}. The $b_{u} are the average of the leftover obtained after removing the overall mean and the movie effect from the ratings $Y_{u, i}.

```
tic()

# Calculate b_u

user_effect_avgs <- train_data %>%
  left_join(movie_effect_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mean_rating - b_i))
```

```
# Predict the rating in test_data using Movie-user effect model

movie_user_effect_predicted_ratings <- test_data %>%
  left_join(movie_effect_avgs, by='movieId') %>%
  left_join(user_effect_avgs, by='userId') %>%
  mutate(pred = mean_rating + b_i + b_u) %>%
  .$pred

toc()
```

```
## 9.2 sec elapsed
```

Calculate RMSE of Movie-user effect model

```
movie_user_effect_rmse <- rmse(movie_user_effect_predicted_ratings, test_data$rating)

movie_user_effect_rmse
```

```
## [1] 0.8580505
```

```
save(movie_user_effect_rmse, file = "movie_user_effect_rmse.RData")
```

5.3.5 Regularized Movie-User effect model

Set up a Lambdas vector

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

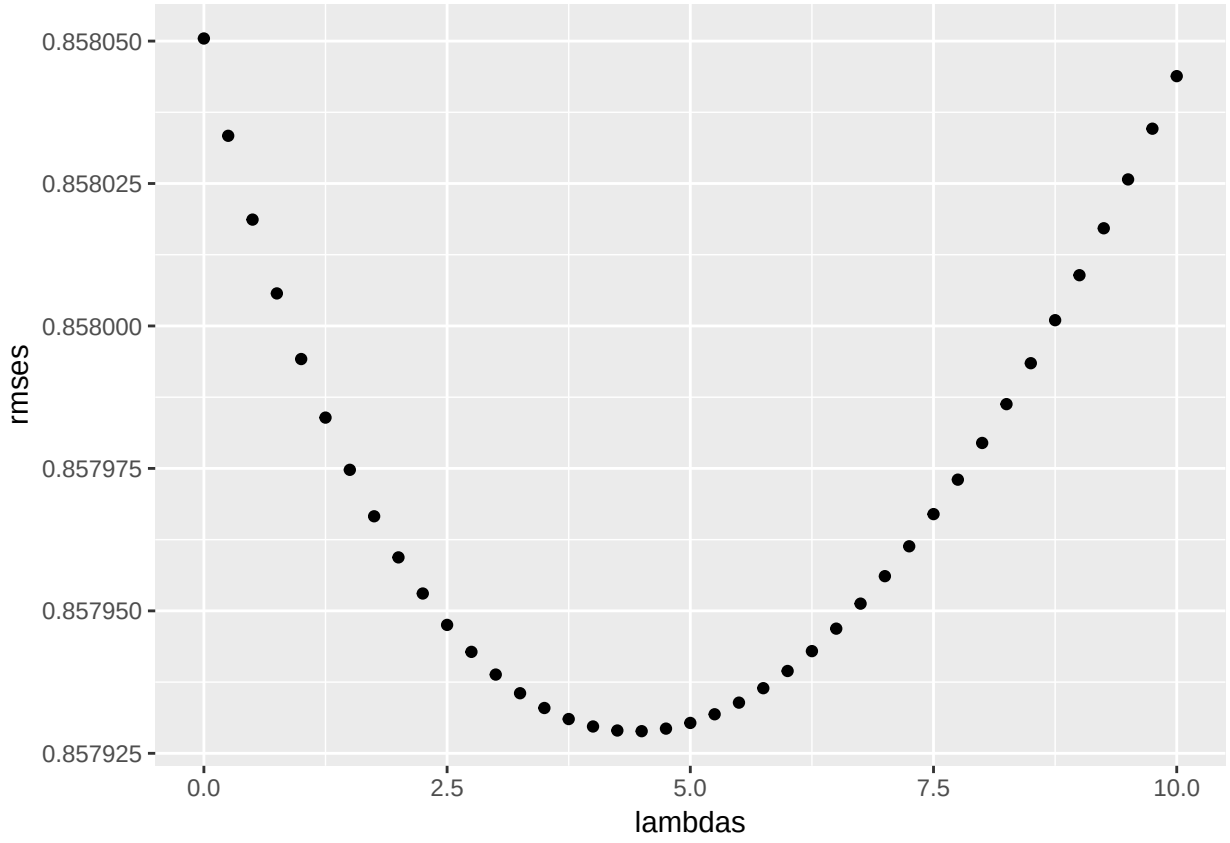
Build a function to run Movie-user effect model with different values of lambdas, and calculate their RMSE

```
tic()
rmses <- sapply(lambdas, function(l){

  mean_rating <- mean(train_data$rating)

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

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

  movie_user_effect_predicted_ratings <-
    test_data %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mean_rating + b_i + b_u) %>%
    .$pred

  return(rmse(movie_user_effect_predicted_ratings, test_data$rating))
})

toc()
```

```
## 414.1 sec elapsed
```

Visualize the rmse and lambdas to observe the optimal value of lambda

```
qplot(lambdas, rmses)
```



The optimal lambda is:

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

```
## [1] 4.5
```

```
save(lambda, file = "lambda.RData")
```

Report the RMSE of the Movie-User effect model with the optimal lambda

```
regularized_movie_user_effect_rmse <- min(rmses)

regularized_movie_user_effect_rmse
```

```
## [1] 0.8579289
```

```
save(regularized_movie_user_effect_rmse, file = "regularized_movie_user_effect_rmse.RData")
```

6 Running other approaches

This section will show the steps to conduct four machine learning algorithms. The first one is to run Random forest using "Ranger" package. The second one is k-nearest neighbor approach in the "caret" package. The following approaches are also in the "caret" package including Quantile Regression Neural Network and Bayesian Regularized Neural Networks.

The purpose of doing this is to explore the power of other algorithms in predicting the "rating" variable and learn how to run and tune them to get better results. In fact, the expectation is to know some limitations in those machine learning "blackbox" such as how they use the memory of the computer and how long it takes to get the predicted results.

6.1 Random forest approach using "Ranger" package

6.1.1 Random forest with 300 trees using a subset of tidy_edx

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))

gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2217488 118.5   28560128 1525.3  35700160 1906.6
## Vcells 21016818 160.4  792355065 6045.2 908251674 6929.5
```

```
# Load necessary packages
library(tidyverse)
library(ranger)
```

```
## Warning: package 'ranger' was built under R version 3.5.3
```

```
library(rsample)
library(tictoc)
library(Metrics)
library(caret)


# Load the datasets
tic()
load("tidy_edx.RData")


# Create a subset of tidy_edx dataset
set.seed(2019)

sample <- sample_n(tidy_edx, 0.001 * nrow(tidy_edx))

rm(tidy_edx)

gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2227027 119.0   22848102 1220.3  35700160 1906.6
## Vcells 21171685 161.6  633884052 4836.2 908251674 6929.5
```

```
# Create validation set
cv_split <- vfold_cv(sample, v = 5) #v: how many times the data should split
```

```r
# Mapping train & validate
cv_data <- cv_split %>%
  mutate(train = map(splits, ~training(.x)),
         validate = map(splits, ~testing(.x)))


#Tune the Hyper-Parameters
cv_tune <- cv_data %>% crossing(mtry = 1:6)


# Random forest model with 300 trees
cv_models_tunerf <- cv_tune %>%
  mutate(model = map2(train,
                      mtry,
                      ~ ranger(formula =
                                 rating ~.,
                               data = .x,
                               mtry = .y,
                               num.trees = 300)))


# Generate validate predictions for each model
cv_prep_tunerf <- cv_models_tunerf %>%
  mutate(validate_actual = map(validate, ~.x$rating),
         validate_predicted = map2(.x = model,
                                   .y = validate,
                                   ~predict(.x, .y)$predictions))


# Calculate validate RMSE for each fold and mtry combination
cv_eval_tunerf <- cv_prep_tunerf %>%
  mutate(validate_rmse = map2_dbl(.x = validate_actual,
                                  .y = validate_predicted,
                                  ~rmse(actual = .x, predicted = .y)))


# Calculate the mean validate_mae for each mtry used
mean_rmse_by_mtry <- cv_eval_tunerf %>%
  group_by(mtry) %>%
  summarise(mean_rmse = mean(validate_rmse))

min_mean_rmse <- min(mean_rmse_by_mtry$mean_rmse)


# Result of Random Forest with 300 trees

mean_rmse_by_mtry
```

```
## # A tibble: 6 x 2
##    mtry mean_rmse
##   <int>     <dbl>
## 1     1      1.03
```

```
## 2     2      1.03
## 3     3      1.03
## 4     4      1.04
## 5     5      1.04
## 6     6      1.04
```

```
# Save the minimum of rmse to conduct Table of results
rf1_rmse <- min(mean_rmse_by_mtry$mean_rmse)

save(rf1_rmse, file = "rf1_rmse.RData")


# Memory usage of running Random Forest with 300 trees
memory.size()
```

```
## [1] 3406.3
```

```
# Time spending in running Random Forest with 300 trees
toc()
```

```
## 290.6 sec elapsed
```

6.1.2 Run random forest with 500 trees using a subset of tidy_edx

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))

gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2238737 119.6   18278481  976.2  35700160 1906.6
## Vcells 21060556 160.7  507107241 3869.0 908251674 6929.5
```

```
# Load necessary packages
library(tidyverse)
library(ranger)
library(rsample)
library(tictoc)
library(Metrics)
library(caret)


# Load the datasets
tic()
load("tidy_edx.RData")
load("tidy_validation.RData")


# Create a subset of tidy_edx dataset
set.seed(2019)

sample <- sample_n(tidy_edx, 0.001 * nrow(tidy_edx))

rm(tidy_edx)

gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2248768 120.1   14622784  781.0  35700160 1906.6
## Vcells 35513067 271.0  405685792 3095.2 908251674 6929.5
```

```
# Create validation set
cv_split <- vfold_cv(sample, v = 5) #v: how many times the data should split


# Mapping train & validate
cv_data <- cv_split %>%
  mutate(train = map(splits, ~training(.x)),
         validate = map(splits, ~testing(.x)))


# Tune the Hyper-Parameters
cv_tune <- cv_data %>% crossing(mtry = 1:6)


# Random forest model with 500 trees
cv_models_tunerf <- cv_tune %>%
```

```
  mutate(model = map2(train,
                      mtry,
                      ~ ranger(formula =
                                 rating ~.,
                               data = .x,
                               mtry = .y,
                               num.trees = 500)))


# Generate validate predictions for each model
cv_prep_tunerf <- cv_models_tunerf %>%
  mutate(validate_actual = map(validate, ~.x$rating),
         validate_predicted = map2(.x = model,
                                   .y = validate,
                                   ~predict(.x, .y)$predictions))


# Calculate validate RMSE for each fold and mtry combination
cv_eval_tunerf <- cv_prep_tunerf %>%
  mutate(validate_rmse = map2_dbl(.x = validate_actual,
                                  .y = validate_predicted,
                                  ~rmse(actual = .x, predicted = .y)))


# Calculate the mean validate_mae for each mtry used
mean_rmse_by_mtry <- cv_eval_tunerf %>%
  group_by(mtry) %>%
  summarise(mean_rmse = mean(validate_rmse))


# Result of Random Forest with 500 trees

mean_rmse_by_mtry
```

```
## # A tibble: 6 x 2
##    mtry mean_rmse
##   <int>     <dbl>
## 1     1      1.03
## 2     2      1.03
## 3     3      1.03
## 4     4      1.04
## 5     5      1.04
## 6     6      1.04
```

```
# Save the minimum of rmse to conduct Table of results
rf2_rmse <- min(mean_rmse_by_mtry$mean_rmse)

save(rf2_rmse, file = "rf2_rmse.RData")


# Memory usage of running Random Forest with 500 trees
memory.size()
```

```
## [1] 5219.55
```

```r
# Time spending in running Random Forest with 500 trees
toc()
```

```
## 461.11 sec elapsed
```

6.1.3 Run random forest with better subset of tidy_edx with 300 trees

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))

gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2238797 119.6    7486864  399.9  35700160 1906.6
## Vcells 21061610 160.7  566581254 4322.7 908251674 6929.5
```

```
# Load necessary packages
library(tidyverse)
library(ranger)
library(rsample)
library(tictoc)
library(Metrics)
library(caret)


# Load the datasets
tic()
load("tidy_edx.RData")


# Create a subset of tidy_edx dataset
set.seed(2019)

subset_train_data <- tidy_edx %>%
  group_by(movieId) %>%
  mutate(count = n()) %>%
  filter(count >= 1000) %>%
  ungroup(movieId) %>%
  select(-count) %>%
  group_by(userId) %>%
  mutate(count = n()) %>%
  filter(count >= 100) %>%
  ungroup(userId) %>%
  select(-count)

sample <- sample_n(subset_train_data, 0.001 * nrow(subset_train_data))

rm(tidy_edx)

gc()
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   2241836  119.8    7486864  399.9  35700160 1906.6
## Vcells 132668801 1012.2  453265003 3458.2 908251674 6929.5
```

```
# Create validation set
cv_split <- vfold_cv(sample, v = 5) #v: how many times the data should split


# Mapping train & validate
```

```r
cv_data <- cv_split %>%
  mutate(train = map(splits, ~training(.x)),
         validate = map(splits, ~testing(.x)))


# Tune the Hyper-Parameters
cv_tune <- cv_data %>% crossing(mtry = 1:6)

# Random forest model with 300 trees
cv_models_tunerf <- cv_tune %>%
  mutate(model = map2(train,
                      mtry,
                      ~ ranger(formula = rating ~.,
                                       data = .x,
                                       mtry = .y,
                                       num.trees = 300)))

# Generate validate predictions for each model
cv_prep_tunerf <- cv_models_tunerf %>%
  mutate(validate_actual = map(validate, ~.x$rating),
         validate_predicted = map2(.x = model,
                                   .y = validate,
                                   ~predict(.x, .y)$predictions))

# Calculate validate RMSE for each fold and mtry combination
cv_eval_tunerf <- cv_prep_tunerf %>%
  mutate(validate_rmse = map2_dbl(.x = validate_actual,
                                  .y = validate_predicted,
                                  ~rmse(actual = .x, predicted = .y)))

# Calculate the mean validate_mae for each mtry used
mean_rmse_by_mtry <- cv_eval_tunerf %>%
  group_by(mtry) %>%
  summarise(mean_rmse = mean(validate_rmse))


# Result of Random Forest using better dataset with 300 trees

mean_rmse_by_mtry
```

```
## # A tibble: 6 x 2
##    mtry mean_rmse
##   <int>     <dbl>
## 1     1      1.02
## 2     2      1.02
## 3     3      1.02
## 4     4      1.03
## 5     5      1.03
## 6     6      1.03
```

```r
# Save the minimum of rmse to conduct Table of results
rf3_rmse <- min(mean_rmse_by_mtry$mean_rmse)

save(rf3_rmse, file = "rf3_rmse.RData")
```

```r
# Memory usage of running Random Forest using better dataset with 300 trees
memory.size()
```

```
## [1] 3681.41
```

```r
# Time spending in running Random Forest using better dataset with 300 trees
toc()
```

```
## 238.86 sec elapsed
```

6.2 k-Nearest Neighbors approach

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))

gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2238621 119.6    7486864  399.9  35700160 1906.6
## Vcells 21059661 160.7  436816568 3332.7 908251674 6929.5
```

```
# Load necessary packages
library(tidyverse)
library(rsample)
library(tictoc)
library(Metrics)
library(caret)


# Load the datasets
tic()
load("tidy_edx.RData")


# Create a subset of tidy_edx dataset
set.seed(2019)

sample <- sample_n(tidy_edx, 0.001 * nrow(tidy_edx))

rm(tidy_edx)

gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2242613 119.8    7486864  399.9  35700160 1906.6
## Vcells 21206080 161.8  349453254 2666.2 908251674 6929.5
```

```
# Set up cross validation method for the knn model
fitControl <- trainControl(method = "cv",
                           number = 5,
                           p = 0.8)

# Run the knn model
model_knn <- train(rating ~ movieId + userId + genres + date_of_rating,
           data = sample,
           method = "knn",
           trControl = fitControl,
           verbose= FALSE)


# Result of K-nearest neighbors approach

model_knn$results[2]
```

```
##       RMSE
## 1 1.138431
```

```
## 2 1.113371
## 3 1.099573
```

```
# Save the minimum of rmse to conduct Table of results
knn_rmse <- min(model_knn$results[2])

save(knn_rmse, file = "knn_rmse.RData")


# Memory usage of running Knn model
memory.size()
```

```
## [1] 387.25
```

```
# Time spending in running Knn model
toc()
```

```
## 53.97 sec elapsed
```

7 Choosing a model for validation

The table of RMSEs of different models will help in deciding which model should be use in the validation step. This project will take the model that generates the lowest RMSE in the testing data (test_data).

```
load("benchmark.RData")
load("movie_effect_rmse.RData")
load("movie_user_effect_rmse.RData")
load("regularized_movie_user_effect_rmse.RData")
load("rf1_rmse.RData")
load("rf2_rmse.RData")
load("rf3_rmse.RData")
load("knn_rmse.RData")


# Table of RMSEs of different machine learning approaches
RMSE_results <- data_frame(Model = "Benchmark", RMSE = benchmark_rmse)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
RMSE_results <- bind_rows(RMSE_results,
                          data_frame(Model = "Movie effect",
                                     RMSE = movie_effect_model_rmse))

RMSE_results <- bind_rows(RMSE_results,
                          data_frame(Model = "Movie-User effect",
                                     RMSE = movie_user_effect_rmse))

RMSE_results <- bind_rows(RMSE_results,
                          data_frame(Model = "Regularized Movie-User effect",
                                     RMSE = regularized_movie_user_effect_rmse))

RMSE_results <- bind_rows(RMSE_results,
                          data_frame(Model = "Random Forest with 300 trees",
                                     RMSE = rf1_rmse))

RMSE_results <- bind_rows(RMSE_results,
                          data_frame(Model = "Random Forest with 500 trees",
                                     RMSE = rf2_rmse))

RMSE_results <- bind_rows(RMSE_results,
                          data_frame(Model = "Random Forest in better dataset",
                                     RMSE = rf3_rmse))

RMSE_results <- bind_rows(RMSE_results,
                          data_frame(Model = "K-nearest neighbors",
                                     RMSE = knn_rmse))

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

Model	RMSE
Benchmark	1.0525823
Movie effect	0.9415179
Movie-User effect	0.8580505

Model	RMSE
Regularized Movie-User effect	0.8579289
Random Forest with 300 trees	1.0271242
Random Forest with 500 trees	1.0267229
Random Forest in better dataset	1.0164987
K-nearest neighbors	1.0995727

8 Validate the models

8.1 Remove and free up working space

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2277248 121.7    7486864  399.9  35700160 1906.6
## Vcells 21143162 161.4  223650082 1706.4 908251674 6929.5
```

```
## [1] 8084
```

```
## [1] 342.24
```

8.2 Load necessary packages

8.3 Load "validation" dataset

```
load("validation.RData")
```

8.4 Tidy up the "validation" dataset.

Separate "title" column into "title" and "movie_year"

```
tidy_validation <- tidy_validation %>%
  mutate(movie_year = as.integer(movie_year), movieId = as.integer(movieId))
```

Save tidy_edx to the local system.

Remove unnecessary dataset "tidy_edx_date", "tidy_edx_sep_genres"

```
rm("validation_date", "validation_sep_genres")

gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  2287782 122.2    7486864  399.9  35700160 1906.6
## Vcells 40470418 308.8  178920065 1365.1 908251674 6929.5
```

8.5 Validate three best models that have the lowest RMSEs

```
tic()

load("tidy_edx.RData")
load("lambda.RData")

mean_rating <- mean(tidy_edx$rating)

b_i <- tidy_edx %>%
    group_by(movieId) %>%
```

```
    summarize(b_i = sum(rating - mean_rating)/(n()+lambda))

b_u <- tidy_edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mean_rating)/(n()+lambda))

movie_user_effect_predicted_ratings <-
    tidy_validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mean_rating + b_i + b_u) %>%
    .$pred

# Memory usage
memory.size()
```

```
## [1] 3190.27
```

```
# Time spending in running the model
toc()
```

```
## 30.17 sec elapsed
```

RMSE of the best model in the validation step

```
movie_user_effect_rmse <- rmse(movie_user_effect_predicted_ratings, tidy_validation$rating)

Final_result <- data_frame(Model = "Movie-User effect", RMSE = movie_user_effect_rmse)

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

Model	RMSE
Movie-User effect	0.8630011

9 Conclusion

The report had described the progress of finding the best predicting model for rating variable in MovieLens dataset. In fact, it is the Regularized Movie-User effect model, which is belonged to Factorization approach. The smallest Root Mean Square Error that this model can archive in the validation dataset is 0.8630.

In addition, this approach outperforms two others such Random forest and K-neaesr neighbors in both memory usage and time consumption. In fact, factorization approach allows the computer to run full training dataset which can not achieve in using two others.

Although Random forest model and Knn model fail to get the smallest RMSE in this project, there are rooms for these models to improve the scores through an increase in computer capacity and knowledge of the user in understanding these models. For example, if the computer can raise its memory from 8,000 mb to 32,000 mb, it can process not only 0.1 percent of training dataset but the full one which would promise better RMSE. In addition, the ability to exploit these models and their packages by tuning their parameters could also produce better RMSE.

In conclusion, the main purpose of doing this project is to apply lessons in Harvards courses to conduct a real question. It had provided valuable time for practicing and testing what we have learnt, and the most interested thing is that it opens more questions and paths to get deeper in the machine learning field.