Movielens Project

Iván de Luna Aldape

June 1, 2021

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

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

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

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

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

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

```
library(ggplot2)
```

Introduction

The Movie Lens project is a model that predicts the user rating based on different variables such as the movie, genre and the user behavior regarding ratings.

```
### EDX Code ###
##############################################################################
#
```

```r
# Create edx set, validation set (final hold-out test 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")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(data.table)
library(lubridate)

# 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")

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(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") # if using R 3.5 or earlier, use `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)
```

```
# RMSE function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2,na.rm=T))
}
```

Methods and Analysis

Data Exploratory Analysis

The number of rows in the edx dataset is 9,000,055 with 6 variables

```
#Rows
nrow(edx)
```

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

```
# Variables
ncol(edx)
```

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

The variables and their class is and their summary

```
# Class
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
# Summary
summary(edx)
```

```
##      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
##
##
##
```

The variable timestamp is of class integer, but it needs to be parsed into datetime format, se we use the lubridate library that will help to do it in an easier way.

```
class(edx$timestamp)
```

```
## [1] "integer"
```

```
edx$timestamp <- as_datetime(edx$timestamp)
validation$timestamp <- as_datetime(validation$timestamp)
```

Check formatted variables is in new POSIXt format

```
class(edx$timestamp)
```

```
## [1] "POSIXct" "POSIXt"
```

```
class(validation$timestamp)
```

```
## [1] "POSIXct" "POSIXt"
```

Initially the variables userId, movieId, timestamp may not be meaningful but will be used as identification and aggregation variables in order to expand the analysis and have a better organization of the datasets. The title variable has a format of "title (year)", so we need to create two columns, for title and for year and apply this to both edx and validation sets. We can use RegEx or, given that the year component of the title variable is in the last part of the string, as (xxxx), we can subtract it without the parenthesis. We also need to subtract the title name from title and create the title_name variable

```
edx <- edx %>% mutate(year = as.numeric(str_sub(title, -5,-2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(title, -5,-2)))
edx <- edx %>% mutate(title_name = as.character(str_sub(title, end = -8)))
validation <- validation %>% mutate(title_name = as.character(str_sub(title, end = -8)))
```

The rating date can also be separated for better understanding

```
edx$rating_year <- format(edx$timestamp, "%Y")
validation$rating_year <- format(validation$timestamp, "%Y")
```

Genres are also grouped by different names in the same movie title a movie can have multiple genre classifications, so we need to separate them. Also, trying to manipulate data with such large strings and classifications can become a problem with memory management.

```
edx <- edx %>% separate_rows(genres, sep="\\|")
validation <- validation %>% separate_rows(genres, sep="\\|")
```

Analysis and Results

Analysis

There are 69,878 unique users and 10,676 unique titles, of which the most common rating is 4 with a 2,588,430 count, followed by 3 and 5.

```
edx %>% select(userId) %>%
  n_distinct()
```

```
## [1] 69878
```

```
edx %>% select(title) %>%
  n_distinct()
```

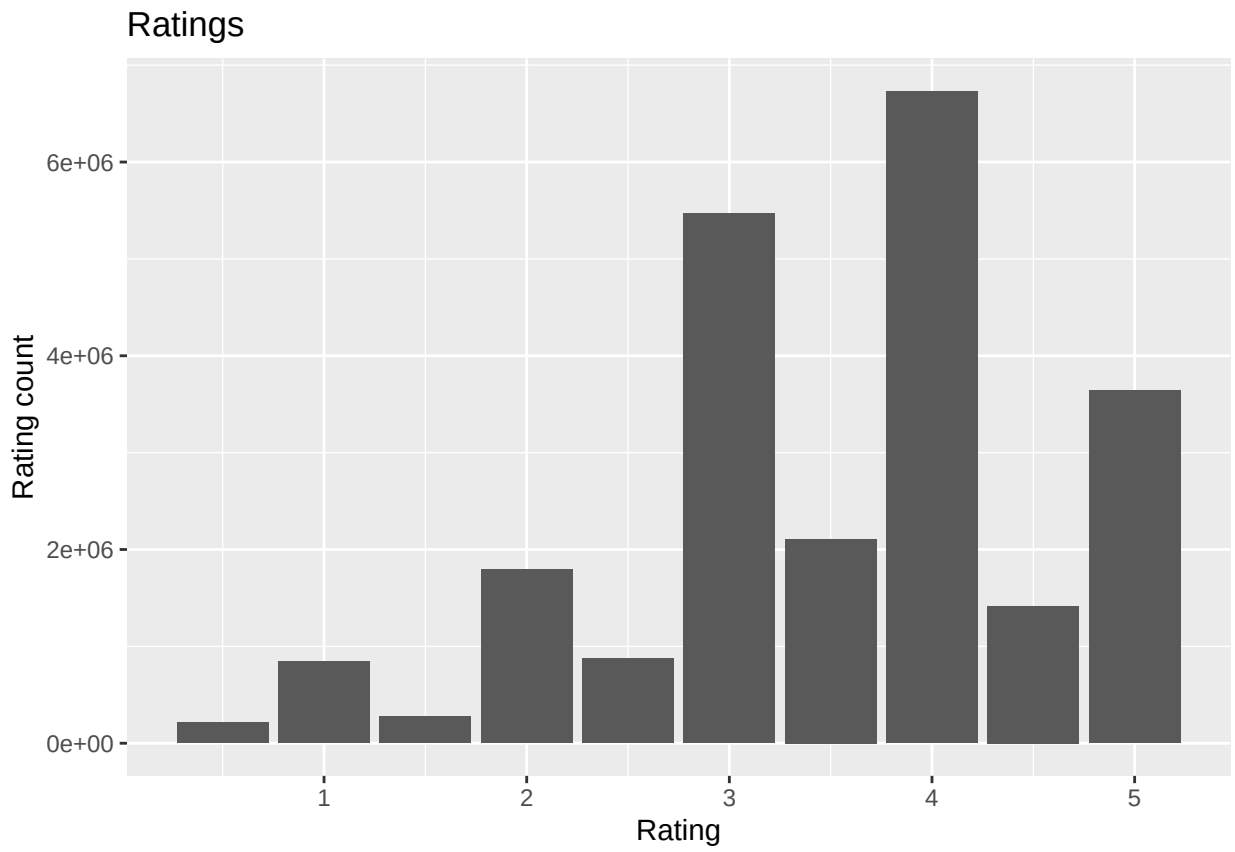
```
## [1] 10676
```

```
edx %>% select(rating) %>%
  group_by(rating) %>%
  summarise(n = n()) %>%
  arrange(-n)
```

```
## # A tibble: 10 x 2
##    rating       n
##     <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
```

It seems that people usually rate movies in whole numbers (i.e. 1,2,3,4,5 vs 0.5,1.5,2.5, etc.)

```
edx %>% select(rating) %>%
  group_by(rating) %>%
  summarise(n = n()) %>%
  arrange(-n) %>%
  ggplot(aes(x = rating, y = n)) + geom_bar(stat = "identity") +
  labs(x = "Rating",
       y = "Rating count",
       title = "Ratings")
```



There seems to by a correlation between rating year and average ratings which started pretty high, 4.33, but it has normalized between 3.4 and 3.6 range. We need to cast the variable rating_year into numeric for the

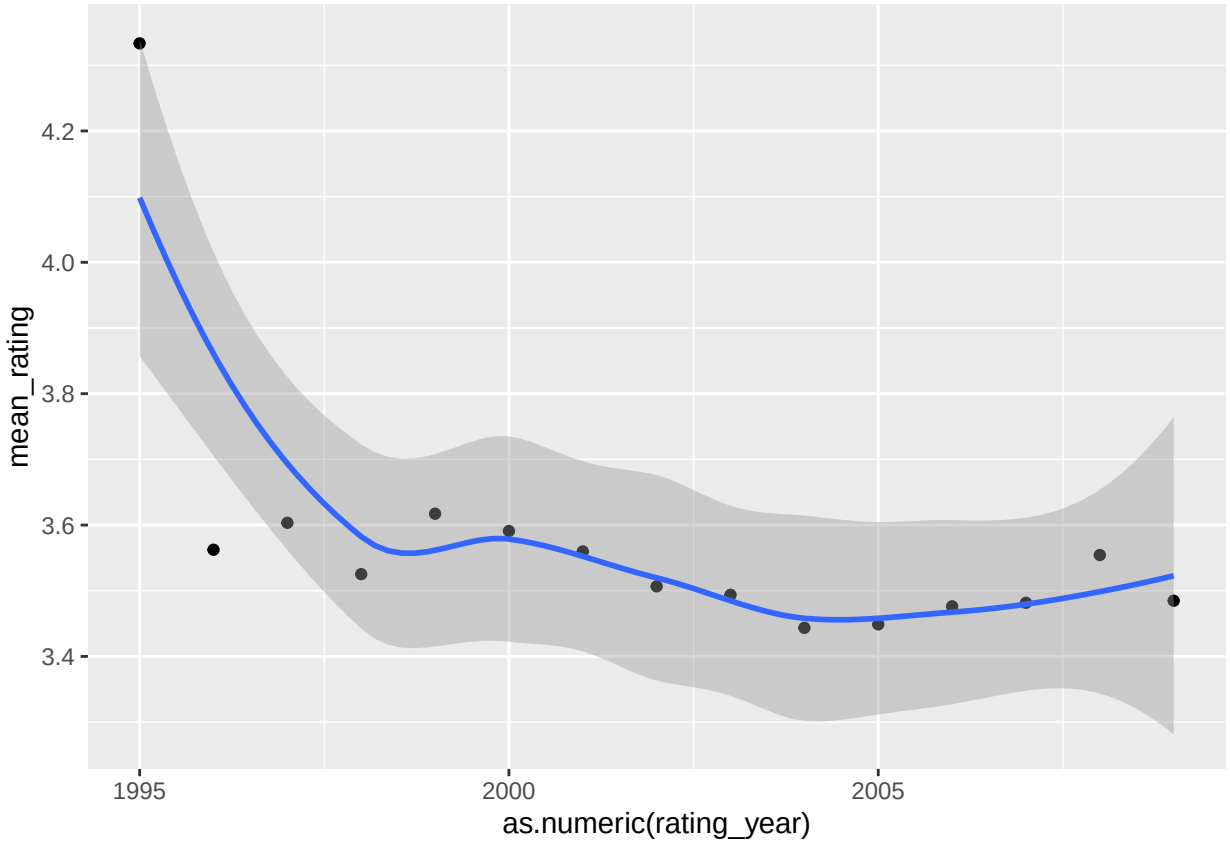
smoothing process.

```
edx %>% select(rating, rating_year) %>%
  group_by(rating_year) %>%
  summarise(mean_rating = mean(rating)) %>%
  arrange(rating_year)
```

```
## # A tibble: 15 x 2
##    rating_year mean_rating
##    <chr>             <dbl>
##  1 1995               4.33
##  2 1996               3.56
##  3 1997               3.60
##  4 1998               3.53
##  5 1999               3.62
##  6 2000               3.59
##  7 2001               3.56
##  8 2002               3.51
##  9 2003               3.49
## 10 2004               3.44
## 11 2005               3.45
## 12 2006               3.48
## 13 2007               3.48
## 14 2008               3.55
## 15 2009               3.48
```

```
edx %>% select(rating, rating_year) %>%
  group_by(rating_year) %>%
  summarise(mean_rating = mean(rating)) %>%
  ggplot(aes(x = as.numeric(rating_year), y = mean_rating)) +
  geom_point() +
  geom_smooth()
```

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



If we do the same with movie ratings, there seems to be a preference for classic movies, which realased around 1931-1962.

```
edx %>% select(rating, year) %>%
  group_by(year) %>%
  summarise(mean_rating = mean(rating)) %>%
  arrange(-mean_rating)
```

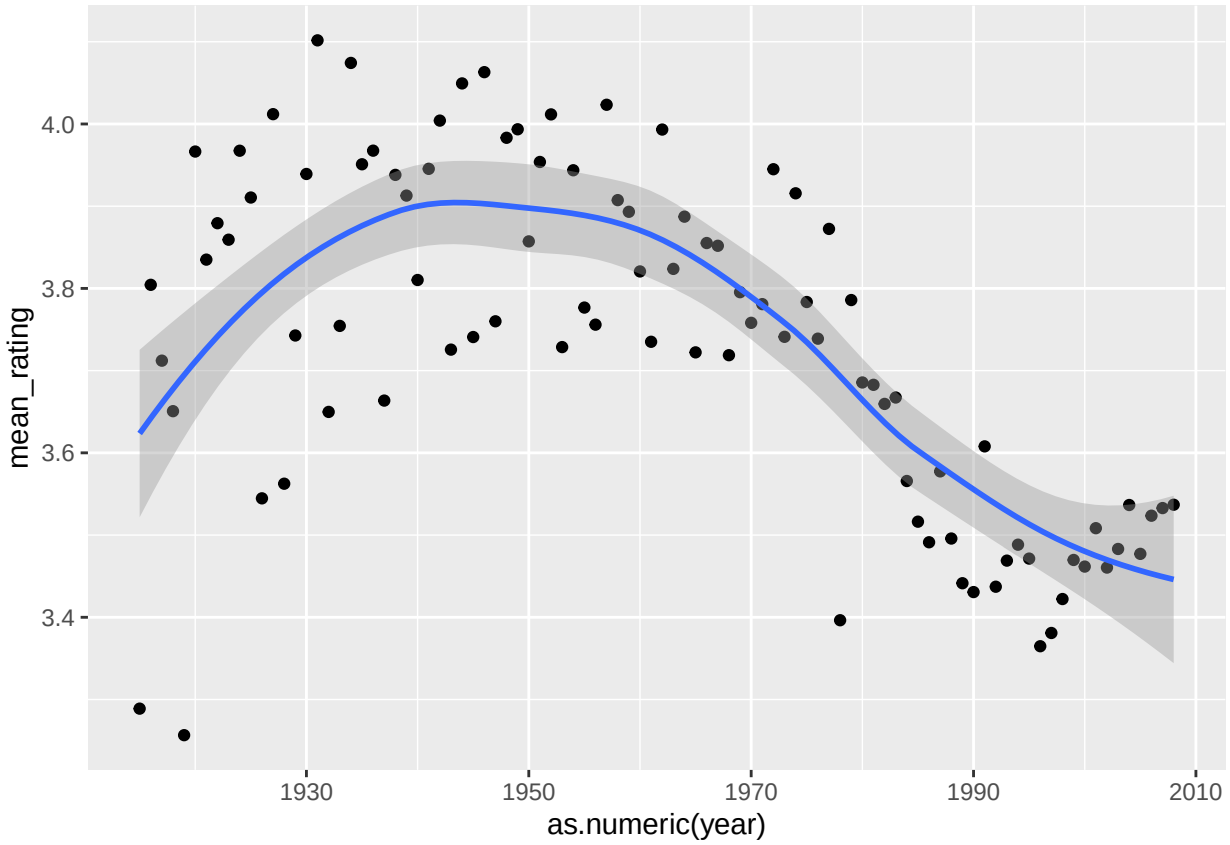
```
## # A tibble: 94 x 2
##     year mean_rating
##    <dbl>       <dbl>
##  1  1931        4.10
##  2  1934        4.07
##  3  1946        4.06
##  4  1944        4.05
##  5  1957        4.02
##  6  1927        4.01
##  7  1952        4.01
##  8  1942        4.00
##  9  1949        3.99
## 10  1962        3.99
## # ... with 84 more rows
```

And movies in general get lower ratings over time or year of release

```
edx %>% select(rating, year) %>%
  group_by(year) %>%
  summarise(mean_rating = mean(rating)) %>%
```

```
  ggplot(aes(x = as.numeric(year), y = mean_rating)) +
  geom_point() +
  geom_smooth()
```

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



The average rating doesn't change over time, but it can be an effect of 4 being the most common rating. It seems that 1995 had the best ratings overall.

```
edx %>% select(title_name, rating) %>%
  group_by(title_name) %>%
  summarise(mean_rating = mean(rating),
            ratings = n()) %>%
  arrange(-ratings)
```

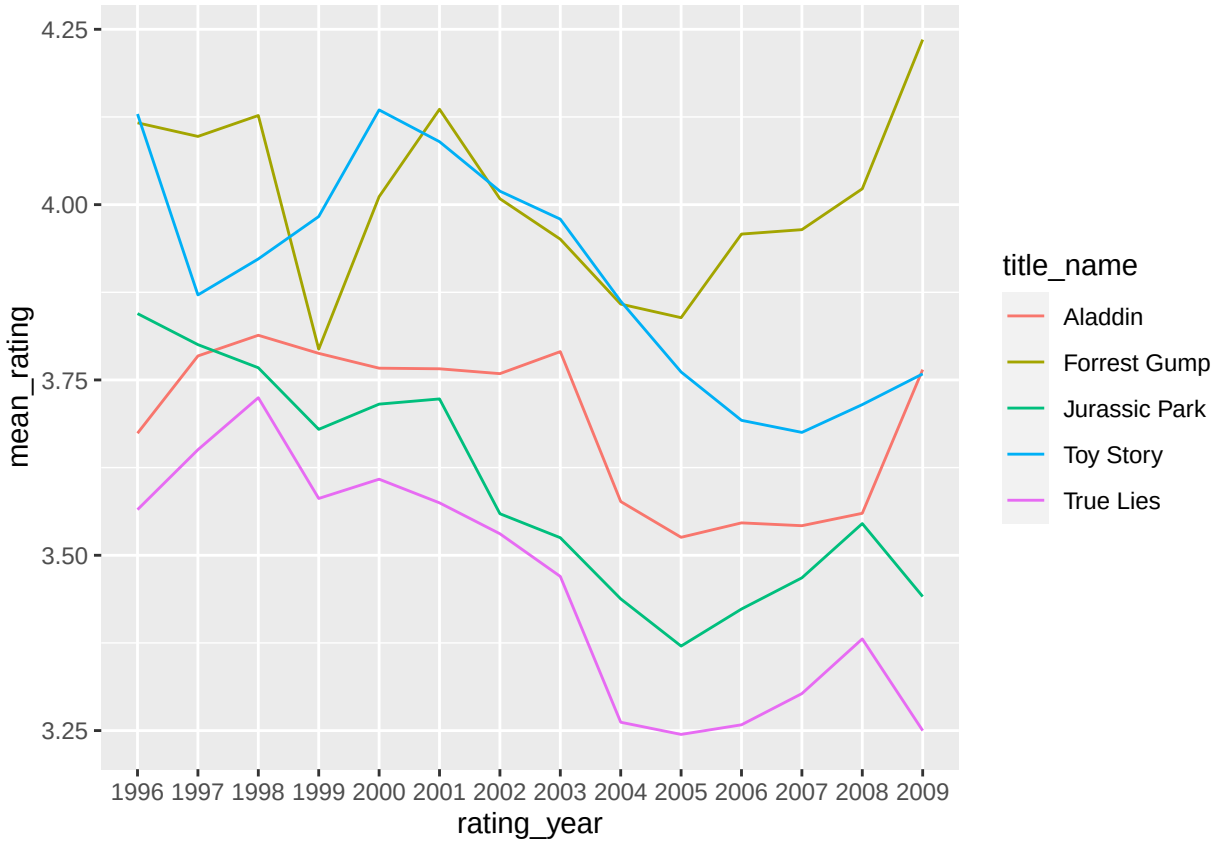
```
## # A tibble: 10,407 x 3
##    title_name                    mean_rating ratings
##    <chr>                               <dbl>   <int>
##  1 Forrest Gump                         4.01  124316
##  2 Toy Story                            3.93  118950
##  3 Jurassic Park                        3.66  117440
##  4 True Lies                            3.50  114115
##  5 Aladdin                              3.67  105865
##  6 Batman                               3.38   98340
##  7 Lion King, The                       3.75   94605
##  8 Pulp Fiction                         4.15   94086
##  9 Independence Day (a.k.a. ID4)        3.38   93796
```

```
## 10 Silence of the Lambs, The            4.20   91146
## # ... with 10,397 more rows
```

If we concentrate in the top 5 rated movies, there is a downward trend in time for a decrease in average ratings until around 2005, then a considerable increase in the avg rating for the next years. Only Toy Story and True Lies have a descending rating in 2009.

```
edx %>% select(title_name, rating_year, rating) %>%
  group_by(title_name) %>%
  filter(title_name %in% c("Forrest Gump",
                           "Toy Story",
                           "Jurassic Park",
                           "True Lies",
                           "Aladdin")) %>%
  group_by(title_name, rating_year) %>%
  summarise(mean_rating = mean(rating)) %>%
  ggplot(aes(x = rating_year, y = mean_rating, color = title_name,
             group = title_name)) +
  geom_line()
```

```
## `summarise()` has grouped output by 'title_name'. You can override using the `.groups` argument.
```

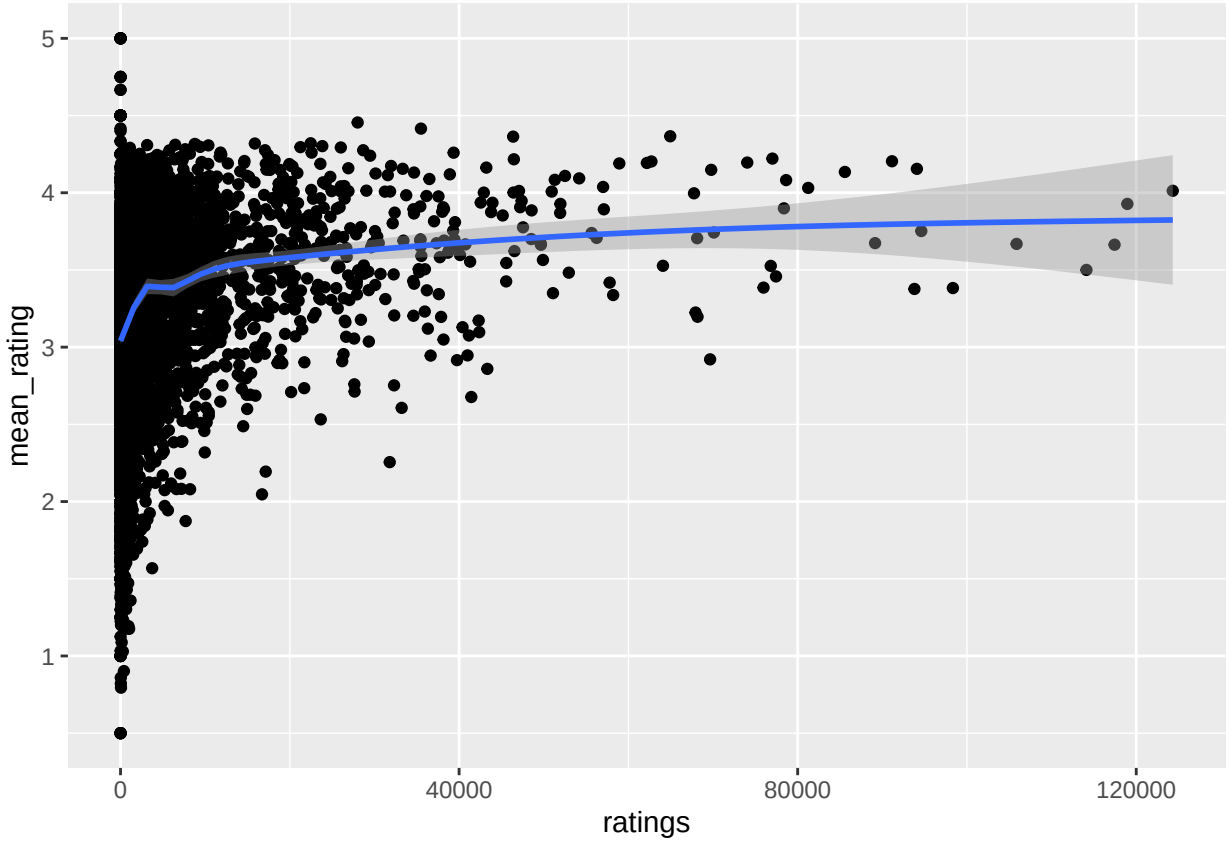


We can also check that there is a correlation between number of ratings and mean rating, which shows a trend into the values between 3.5 and 4, the more ratings the movie has.

```
edx %>% select(title_name, rating) %>%
  group_by(title_name) %>%
  summarise(mean_rating = mean(rating), ratings = n()) %>%
```

```
  ggplot(aes(x = ratings, y = mean_rating)) +
  geom_point() +
  geom_smooth()
```

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



The most amount of reviews by year of release is 1995, followed by 1994 and 1996.

```
edx %>%  select(year) %>%
  group_by(year) %>%
  summarise(n = n()) %>%
  arrange(-n)
```

```
## # A tibble: 94 x 2
##     year       n
##    <dbl>   <int>
##  1  1995 2083655
##  2  1994 1732877
##  3  1996 1561069
##  4  1999 1158834
##  5  1997 1137184
##  6  1998 1086421
##  7  1993 1085520
##  8  2000  931118
##  9  2001  832400
## 10  2002  710565
## # ... with 84 more rows
```

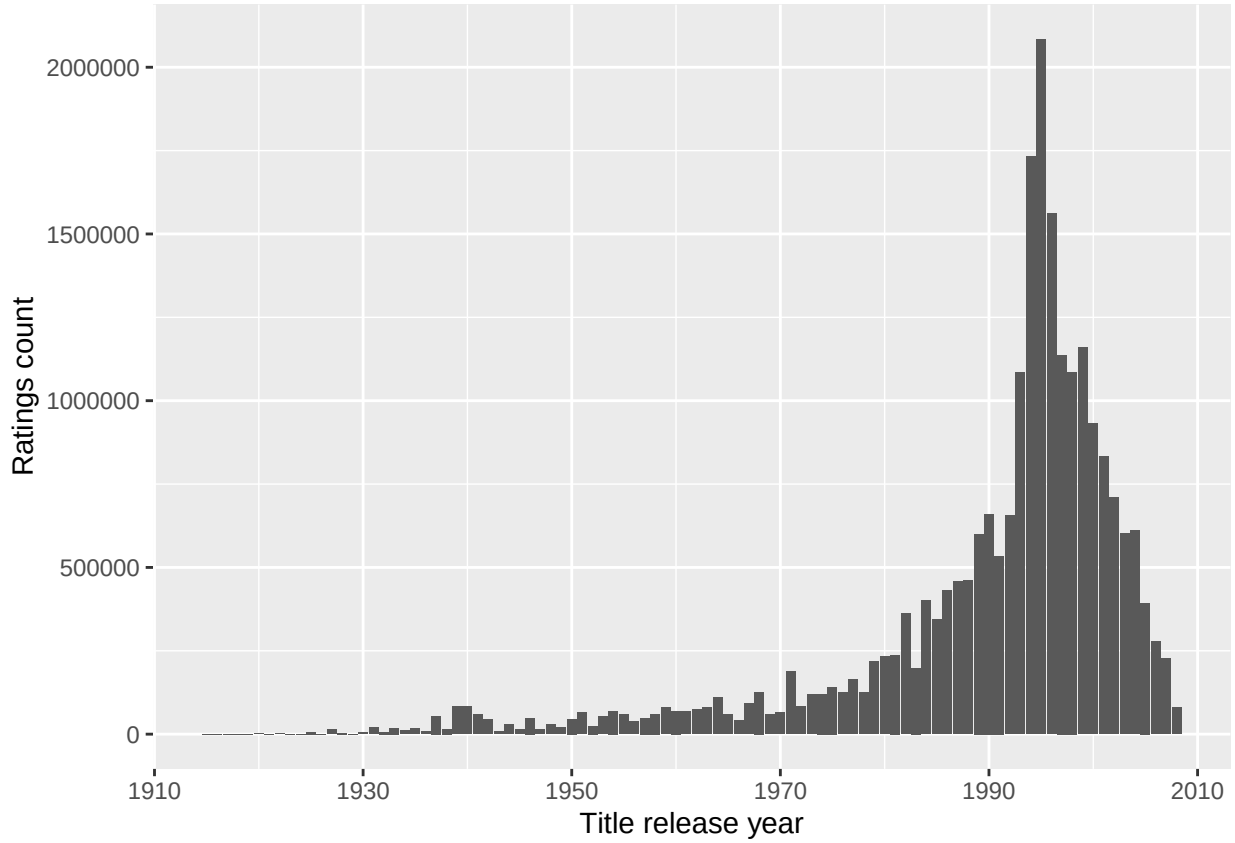
If we arrange by frequency, 10 years have half the reviews and all of them are within 1993 and 2002.

```
edx %>%  select(year) %>%
  group_by(year) %>%
  summarise(n = n()) %>%
  mutate(freq = n/sum(n)) %>%
  arrange(-n)
```

```
## # A tibble: 94 x 3
##     year       n   freq
##    <dbl>   <int>  <dbl>
##  1  1995 2083655 0.0892
##  2  1994 1732877 0.0741
##  3  1996 1561069 0.0668
##  4  1999 1158834 0.0496
##  5  1997 1137184 0.0487
##  6  1998 1086421 0.0465
##  7  1993 1085520 0.0464
##  8  2000  931118 0.0398
##  9  2001  832400 0.0356
## 10  2002  710565 0.0304
## # ... with 84 more rows
```

Arranging by year of release, we can see more clearly the concentration of the movie ratings in time.

```
edx %>%  select(year) %>%
  group_by(year) %>%
  summarise(n = n()) %>%
  mutate(freq = n/sum(n)) %>%
  ggplot() +
  geom_bar(aes(x = year,
               y = n),
               stat = "identity") +
  labs( x = "Title release year",
        y = "Ratings count")
```



In the case of specific titles Forrest Gump has the most reviews with 124,316 followed by Toy Story and Jurassic Park.

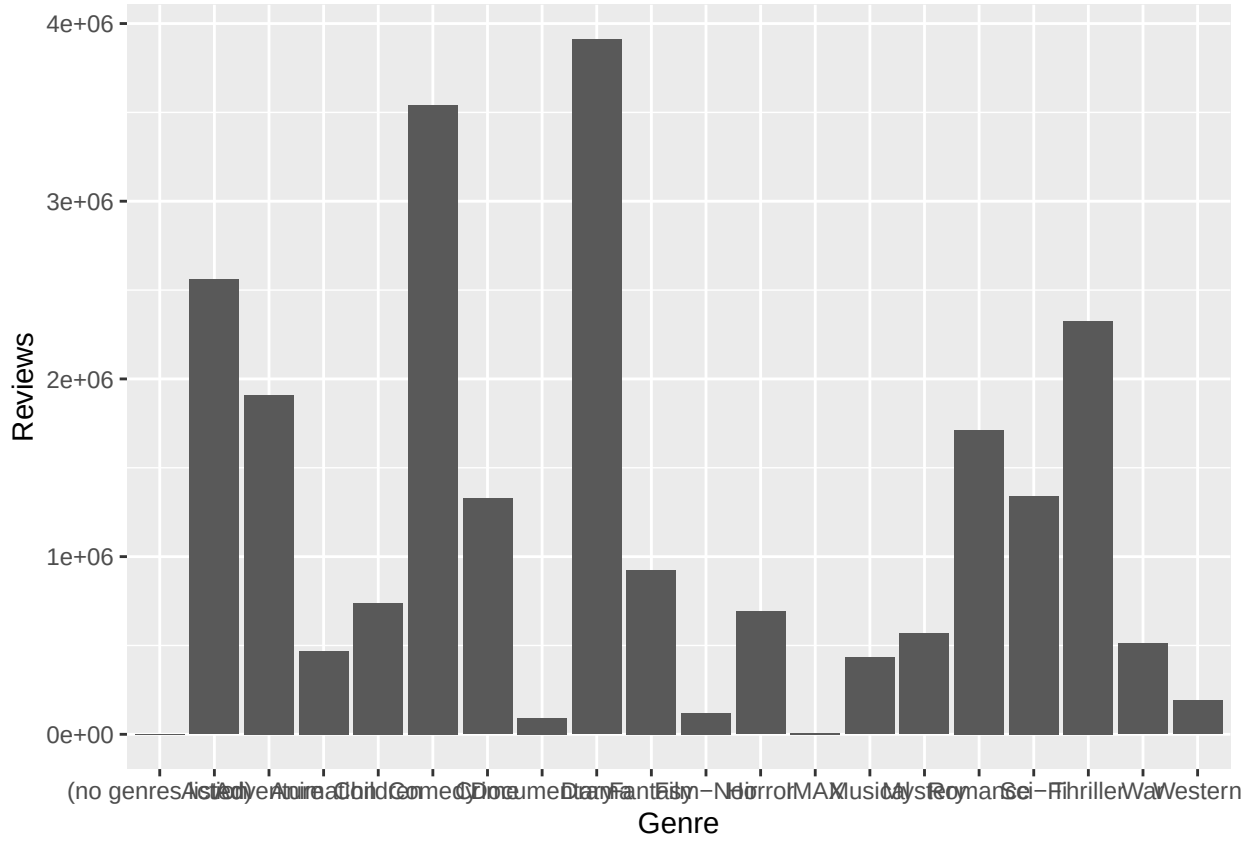
```
edx %>% select(title_name) %>%
  group_by(title_name) %>%
  summarise(n = n()) %>%
  mutate(freq = n/sum(n)) %>%
  arrange(-n)
```

```
## # A tibble: 10,407 x 3
##    title_name                          n    freq
##    <chr>                           <int>   <dbl>
##  1 Forrest Gump                   124316 0.00532
##  2 Toy Story                      118950 0.00509
##  3 Jurassic Park                  117440 0.00502
##  4 True Lies                      114115 0.00488
##  5 Aladdin                        105865 0.00453
##  6 Batman                          98340 0.00421
##  7 Lion King, The                  94605 0.00405
##  8 Pulp Fiction                    94086 0.00403
##  9 Independence Day (a.k.a. ID4)   93796 0.00401
## 10 Silence of the Lambs, The       91146 0.00390
## # ... with 10,397 more rows
```

In the case of genres there is a considerable preference for some such as Drama, Comedy and Action.

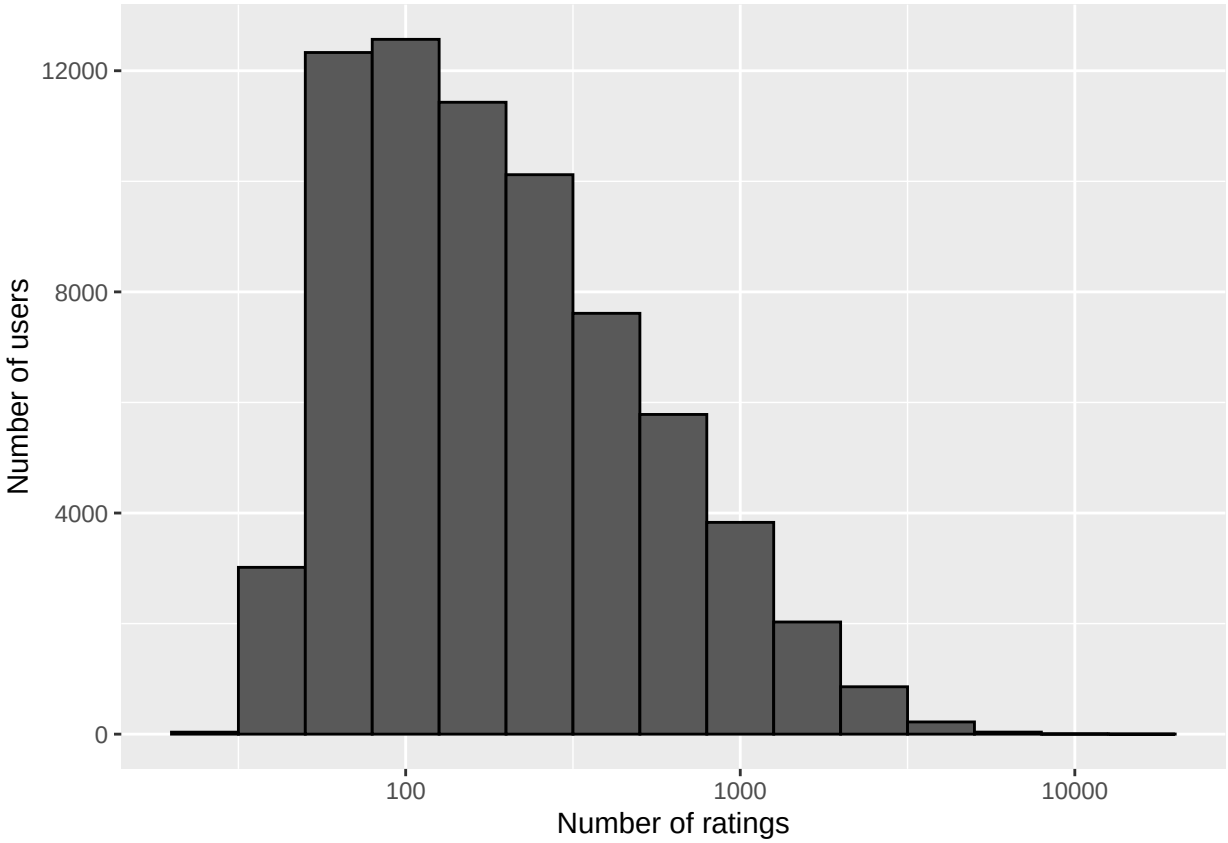
```
edx %>% group_by(genres) %>%
  summarize(n = n()) %>%
```

```
  arrange(desc(n)) %>%
  ggplot(aes(x = genres, y = n)) +
  geom_bar(stat = "identity")+
  labs(x = "Genre", y = "Reviews")
```



In the case of users, the number of ratings is around 100

```
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, binwidth=0.2, color="black", show.legend = FALSE) +
  scale_x_log10() +
  labs(x = "Number of ratings", y = "Number of users")
```



Results

First we create a test set to train the models, according to the specifications

```
set.seed(1)
test_index <- createDataPartition(y = edx$rating,
                                  times = 1,
                                  p = 0.2,
                                  list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

Remove probable coincidences from the other sets so it doesn't repeat values.

```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

The first Model, Average or Naive Average, gives us an average movie rating of 3.5269

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

```
## [1] 3.526972
```

The RMSE for mu is not really good, this can be affected by a lot of variables and considerations, such as the trend in time or preference for newer or older movies

```
rmse_mu <- RMSE(test_set$rating, mu)
rmse_mu
```

```
## [1] 1.051984
```

Create a table to store the results

```
rmse_results <- tibble(method = "Average", RMSE = rmse_mu)
```

Least Squares tries to create a model taking into account some other variables, in this case we will predict our value Y with some values b_variable, which in this case will be defined as the mean difference between the actual rating in the training set vs the mean value.

The second model will thus be defined by the simple or naive average and a movie effect

```
movie_mu <- train_set %>% group_by(movieId) %>%
  summarize(b_movie = mean(rating - mu))

b_movie <- test_set %>% left_join(movie_mu, by = "movieId") %>%
  pull(b_movie)

y_movie <- mu + b_movie
```

The RMSE is a little bit better than a simple average with a value of 0.94

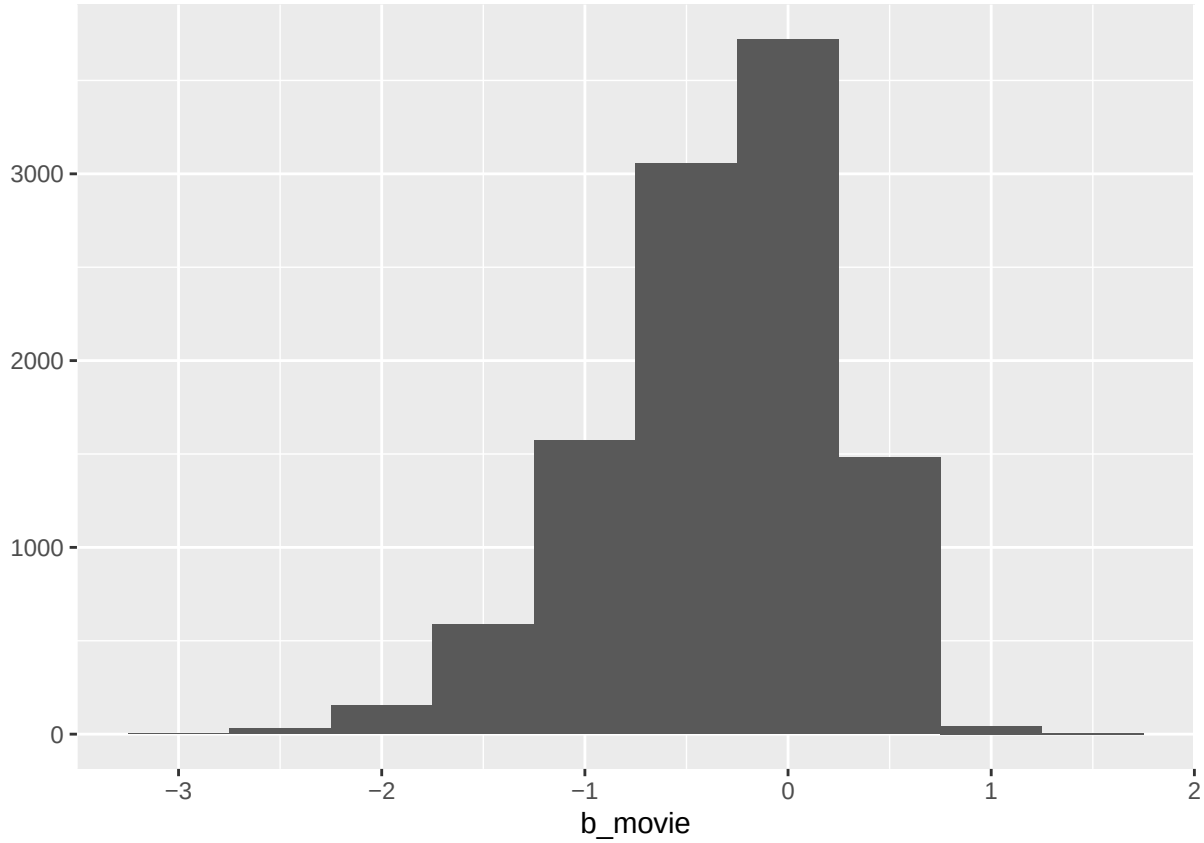
```
rmse_movie <- RMSE(test_set$rating, y_movie)
rmse_movie
```

```
## [1] 0.9409964
```

```
rmse_results <- rbind(rmse_results, tibble(method = "Movie mu",
                                           RMSE = rmse_movie))
```

Visualizing it in an histogram, most of the values are near 0, with a skewness towards negative values.

```
movie_mu %>% qplot(b_movie,
                   geom= "histogram",
                   bins = 10,
                   data = .)
```



Now we add the average rating the user gives to the movies it reviews

```
user_mu <- train_set %>%  group_by(userId) %>%
  left_join(movie_mu, by = "movieId") %>%
  summarise(b_user = mean(rating - mu - b_movie))

b_user <- test_set %>% left_join(user_mu,
                                 by = "userId") %>% .$b_user
```

We estamite the ratings with simple average, movie effect and user effect And we get a better RMSE of 0.85749 compared to the previous model.

```
y_user <- mu + b_movie + b_user
rmse_user <- RMSE(test_set$rating, y_user)
rmse_user
```

```
## [1] 0.8574998
```

```
rmse_results <- rbind(rmse_results, tibble(method = "User mu",
                                           RMSE = rmse_user))
```

We improved substantialy in the RMSE, we can add now a genre effect in order to try to find a better result.

```
genre_mu <- train_set %>%  group_by(genres) %>%
  left_join(movie_mu, by = "movieId") %>%
  summarise(b_genre = mean(rating - mu))

b_genre <- test_set %>% left_join(genre_mu,
                                 by = "genres") %>% .$b_genre
```

We now estimate the model with past effects plus the genre effect and we get a worse RMSE than before (0.8646)

```
y_genre <- mu + b_movie + b_user + b_genre
rmse_genre <- RMSE(test_set$rating, y_genre)
rmse_genre
```

```
## [1] 0.8646397
```

```
rmse_results <- rbind(rmse_results, tibble(method = "Genre mu",
                                           RMSE = rmse_genre))
```

So we will use the second model which takes into account a movie and user effect and try it in the validation set

```
validation_prediction <- validation %>%
  left_join(movie_mu, by = "movieId" ) %>%
  left_join(user_mu , by = "userId") %>%
  mutate(pred = mu + b_movie + b_user) %>%
  pull(pred)
```

We check the RMSE and we get 0.8637, which is a little bit worse than in our test set but not much different.

```
model_valid <- RMSE(validation$rating, validation_prediction)
model_valid
```

```
## [1] 0.8637861
```

```
rmse_results <- rbind(rmse_results, tibble(method = "Validation",
                                           RMSE = model_valid))
rmse_results
```

```
## # A tibble: 5 x 2
##   method      RMSE
##   <chr>      <dbl>
## 1 Average    1.05
## 2 Movie mu   0.941
## 3 User mu    0.857
## 4 Genre mu   0.865
## 5 Validation 0.864
```

Conclusion

We can see that the model can be preditive enough with the average movie and user effect. This can be improved with further considerations that may count towards the specific usage of the platform, given that there are some heavy users in terms of ratings, which may imply that the more they rate, the more considerations they take into account for such rating.