MovieLens Capstone Project

Sergio Grueso Sáez

10/10/2019

INTRODUCTION:

MovieLens dataset contains more than 10 million ratings; each row contains information about the rating like the user ID, the movie ID, its title, the day it was rated and the genre it belongs to.

The aim of this project is to create a model that can predict ratings that users would do if they have just watched a certain movie. That is how MovieLens recommendation system works, if the model predicts for a specific user a rating of, for example, four or five stars out of five, then that movie will be recommended to that user.

First we analyse the dataset by exploring how all the variables are distributed and how are they related to each other. Then we proceed to create a model that takes into account all the insights we've made from the data analysis, and makes a prediction of the rating that a certain user in a certain date will make to a concrete film.

To evaluate our model we use the Root Mean Square Error (RMSE). Our predictions will be accurate enough if the RMSE is lower than 0.8649.

METHOD

DATASET CREATION AND PARTITION:

The MovieLens dataset is downloaded and all the packages required are installed. The dataset is separated in two partitions:

- Training set ("edx"): that we will use to analyse and visualize the data, to create and train our algorithm and to create our predicted ratings.
- Test set ("validation"): that we will use to test our predictions.

The split of the data is made randomly, selecting 90% of data for the edx dataset and 10% for the validation dataset. We made sure that the training set and test set do not

have any user ID or movie ID that is not found in both datasets. The following code creates the MovieLens dataset and generates both test set and training set:

```
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")

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zi
p", dl)
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M10
0K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timesta
mp"))
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")

set.seed(1, sample.kind="Rounding")

test_index <- createDataPartition(y = movielens$rating, times = 1, p =
0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
validation <- temp %>%
     semi_join(edx, by = "movieId") %>%
     semi_join(edx, by = "userId")
removed <- anti_join(temp, validation)

edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

DATA ANALYSIS AND DATA VISUALIZATION

With a summary of the edx and validation dataset we can see their dimensions and what variables they contain:

```
str(edx)

## '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 83898339
2 838984474 838983653 838984885 838983707 838984596 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (
1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action
|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci-Fi" ...

dim(edx)

## [1] 9000055       6

str(validation)

## 'data.frame':    999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 86824564
5 868245920 1136075494 1133571200 844416936 844417070 ...
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "H
ome Alone (1990)" "Rob Roy (1995)" ...
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Chi
ldren|Comedy" "Action|Drama|Romance|War" ...

dim(validation)

## [1] 999999      6
```

Any missing values can be searched with anyNA() function which indicates if there is any elements missing (FALSE means no elements missing).

```
anyNA(edx)

## [1] FALSE

anyNA(validation)

## [1] FALSE
```

Movie ratings:

Our analysis starts by looking how the variables interact with the variable we want to predict (rating) in order to see if there is any pattern we need to include in our model.

We can compute the top ten most rated films on our date set with this piece of code:

```
edx %>% group_by(title) %>% summarize(n = n()) %>% arrange(desc(n)) %>
% head(10)
```

```
## # A tibble: 10 x 2
##    title                                                        n
##    <chr>                                                    <int>
##  1 Pulp Fiction (1994)                                      31362
##  2 Forrest Gump (1994)                                      31079
##  3 Silence of the Lambs, The (1991)                         30382
##  4 Jurassic Park (1993)                                     29360
##  5 Shawshank Redemption, The (1994)                         28015
##  6 Braveheart (1995)                                        26212
##  7 Fugitive, The (1993)                                     25998
##  8 Terminator 2: Judgment Day (1991)                        25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10 Apollo 13 (1995)                                         24284
```

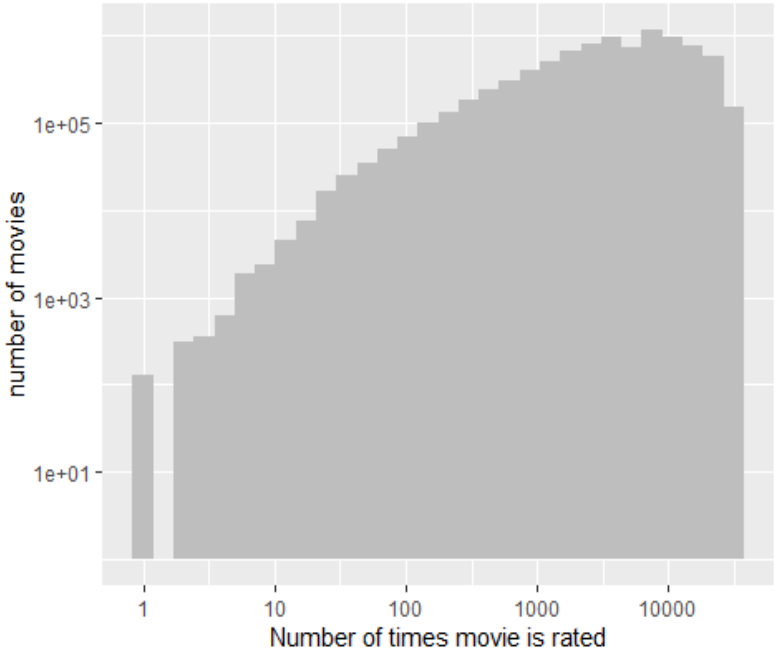
How many movies have been rated only once (total of 126 movies)

```
edx %>% group_by(title) %>% mutate(n = n()) %>% filter(n == 1)
```

```
## # A tibble: 126 x 7
## # Groups:   title [126]
##    userId movieId rating  timestamp title                 genres          n
##     <int>   <dbl>  <dbl>      <int> <chr>                 <chr>       <int>
##  1    826   64153    2.5 1230750043 Devil's Chair, The (~ Horror          1
##  2   3457    3561    1   1051371256 Stacy's Knights (198~ Drama           1
##  3   5227    5616    3.5 1219467370 Mesmerist, The (2002) Comedy|Fan~     1
##  4   5947    6941    2.5 1073321135 Just an American Boy~ Documentary     1
##  5   6905   60880    4   1222805003 Family Game, The (Ka~ Comedy|Dra~     1
##  6   7304   31547    3.5 1230321725 Lessons of Darkness ~ Documentar~     1
##  7   7304   38435    3.5 1230589804 Forty Shades of Blue~ Drama           1
##  8   8041   61970    2   1222779824 Moonbase (1998)       Sci-Fi          1
##  9   9212   63312    4   1226684912 Krabat (2008)         Drama|Fant~     1
## 10  10057   58520    4   1230245584 Mala Noche (1985)     Drama           1
## # ... with 116 more rows
```

And the distribution of the number of rates.

```
edx %>% group_by(title) %>% mutate(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(fill = "grey") +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Number of times movie is rated") + ylab("number of movies")
```

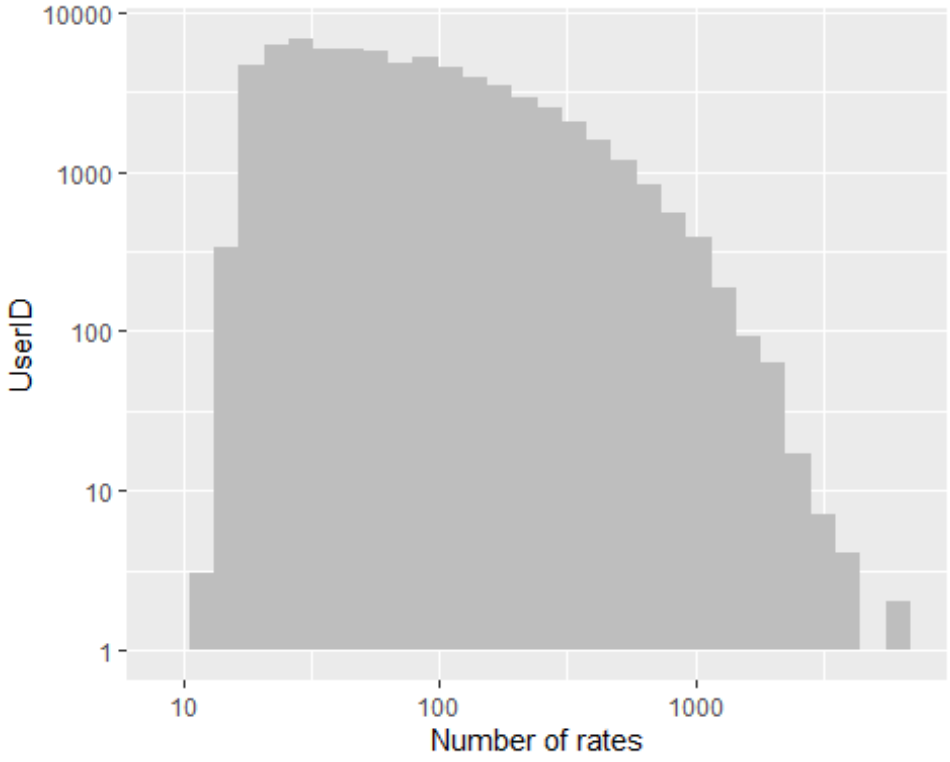


User ratings:

Another variable in our dataset is UserId. Analysis on this variable can help us determine if we need to include a parameter in our model that counters the effect of a particular type of user in the ratings prediction.

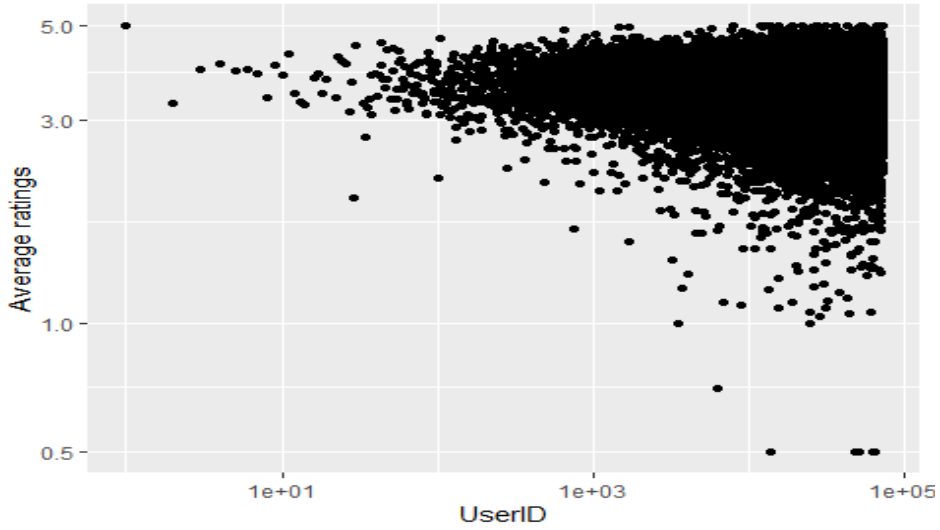
For example we can look if all the users rate with the same frequency by comparing the number of rates per user.

```
edx %>% group_by(userId) %>% summarize(n = n()) %>%
    ggplot(aes(n)) +
    geom_histogram(fill = "grey")  +
    scale_x_log10() +
    scale_y_log10() +
    xlab("Number of rates") + ylab("UserID")
```



The frequency of rates is quite distributed so now we can plot the average rating of every user to see if the is any pattern:

```
edx %>% group_by(userId) %>% summarize(avg= mean(rating)) %>%
    ggplot(aes(userId, avg)) +
    geom_point() +
    scale_x_log10() +
    scale_y_log10() +
    xlab("UserID") + ylab("Average ratings")
```

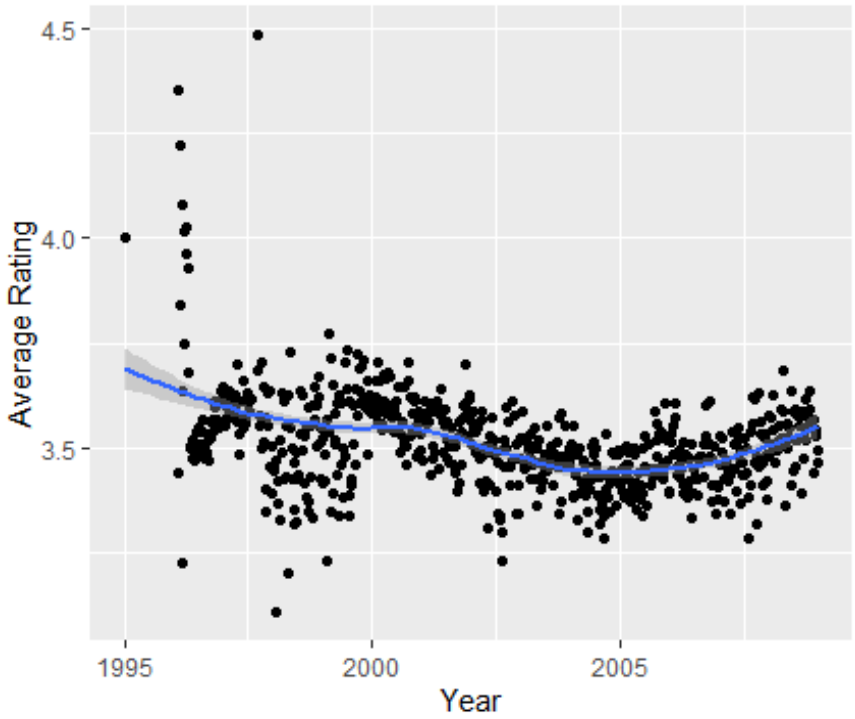


From the previous plot can be inferred that some users usually make low ratings and some others make high ratings, but in average the ratings are above 3 stars.

Ratings over time:

Next variable is time; does the passage of time affect rates? We can figure that out by plotting the number of rates per year:

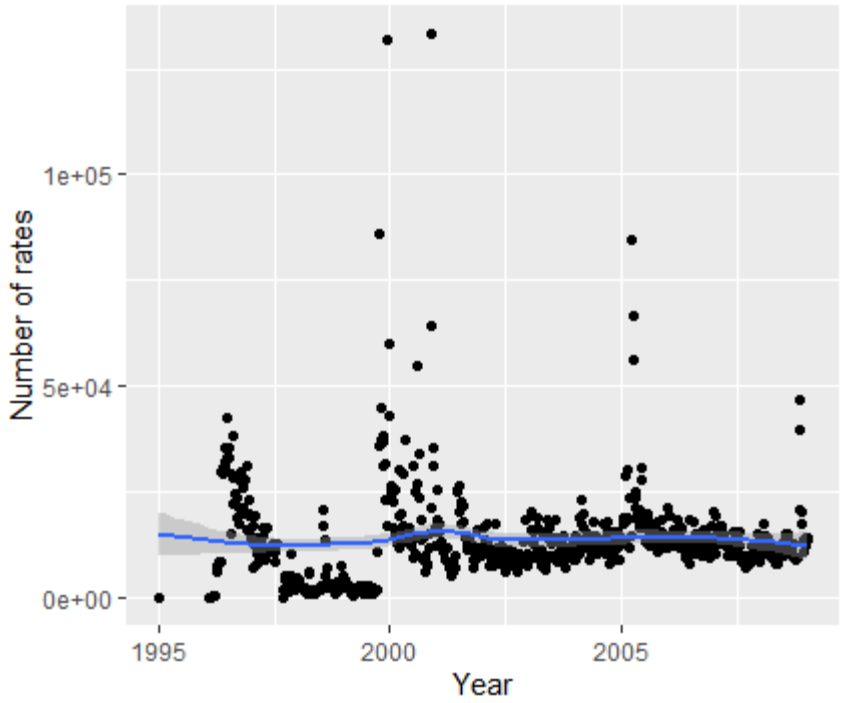
```
edx %>% mutate(date = round_date(date, unit = "week")) %>%
    group_by(date) %>%
    summarize(rating = mean(rating)) %>%
    ggplot(aes(date, rating)) +
    geom_point() +
    geom_smooth() +
    xlab("Year") + ylab("Average Rating")
```



And the mean rates per year:

```
edx %>% mutate(date = round_date(date, unit = "week")) %>%
    group_by(date) %>%
    summarize(n = n()) %>%
    ggplot(aes(date, n)) +
```

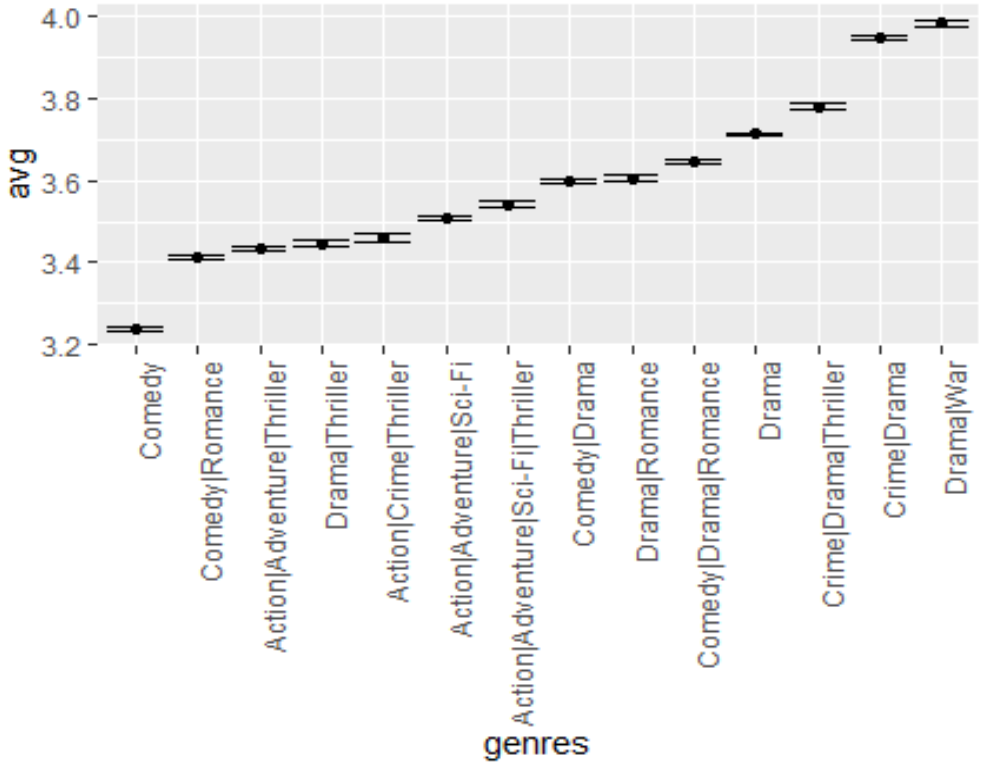
```
    geom_point() +
    geom_smooth() +
    xlab("Year") + ylab("Number of rates")
```



Genre ratings:

We continue our analysis by searching patterns when films are grouped by genres. We define “genre” as any the description given in the variable “genre”. This means that some movies will have several labels but we keep them that way for simplicity. Let’s see the average ratings of the most rated genres:

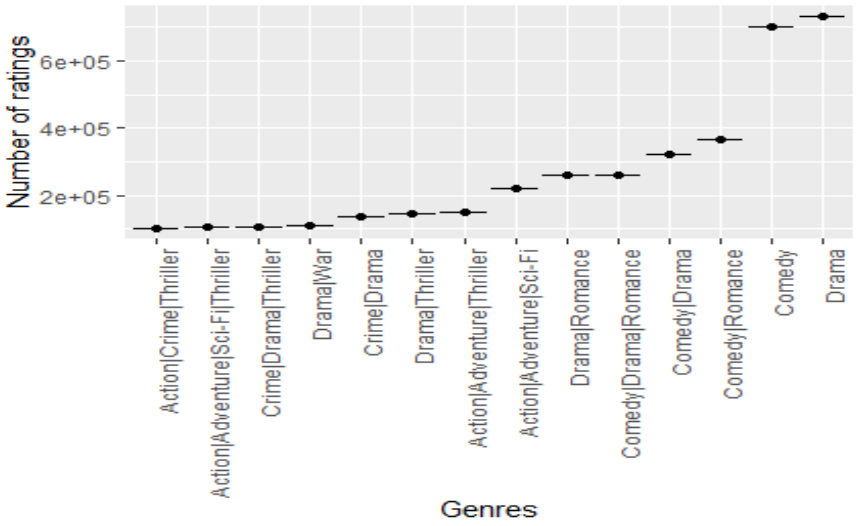
```
edx %>% group_by(genres) %>%
    summarize(n = n(),
              avg = mean(rating),
              se = sd(rating)/sqrt(n())) %>%
    filter(n >= 100000) %>%
    mutate(genres = reorder(genres, avg)) %>%
    ggplot(aes(x = genres,
               y = avg,
               ymin = avg - 2*se,
               ymax = avg + 2*se)) +
    geom_point() +
    geom_errorbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Comedy has the lowest rates on average whereas Drama|War seems to have the highest ratings on average. For this plot, genres with less than 10000 rates have been removed.

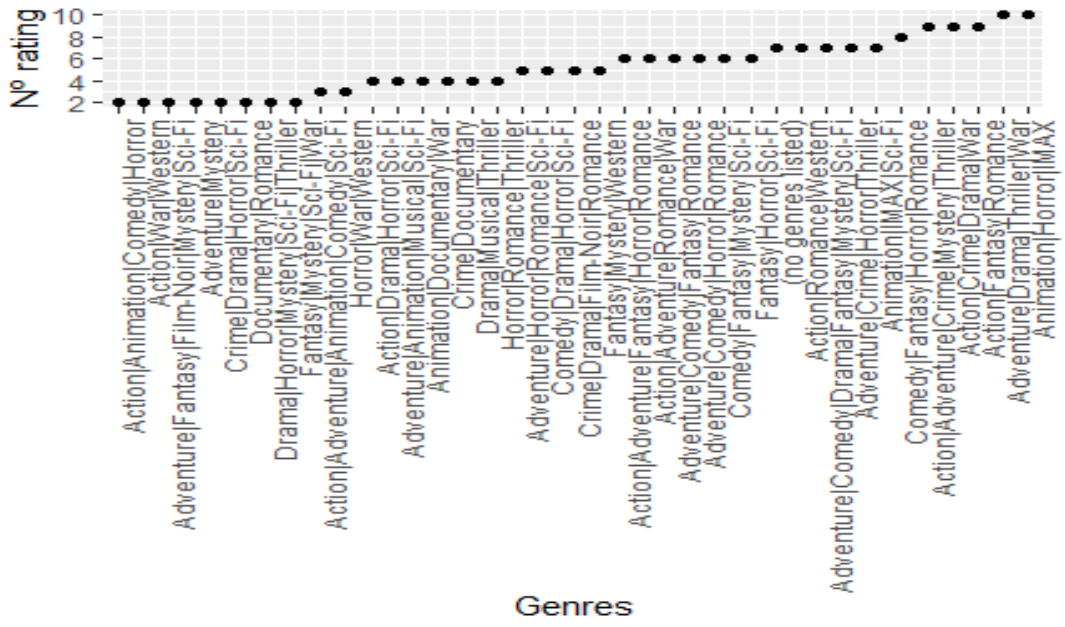
Instead of average, number of rates can also be plotted and the most rated genres are the following (only genres with more than 100000 rates were displayed):

```
edx %>% group_by(genres) %>%
    summarize(n = n(), se = sd(rating)/sqrt(n())) %>%
    filter(n >= 100000) %>%
    mutate(genres = reorder(genres, n)) %>%
    ggplot(aes(x = genres, y = n, ymin = n - 2*se, ymax = n + 2*se)) +
    geom_point() +
    geom_errorbar() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    xlab("Genres") + ylab("Number of ratings")
```



On the other hand, the least rated genres are the following and we see that the probably are the least rated due to their arbitrary combination of diferent genres (only genres with less than 10 rates were displayed):

```
edx %>% group_by(genres) %>%
    summarize(n = n()) %>%
    filter(n <= 10) %>%
    mutate(genres = reorder(genres, n)) %>%
    ggplot(aes(x = genres, y = n)) +
    geom_point() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    xlab("Genres") + ylab("Nº of ratings")
```



Prediction model:

Considering what we have seen in the data exploration and data visualization we are now able to construct an algorithm that considers all the patterns observed.

The approach in order to create the algorithm is to observe how variables behave in the edx dataset and create a parameter for each variable. These parameters will then be used to make predictions of ratings in the validation set.

Our algorithm will be something like this:

$$Y = \mu + \beta_1 + \beta_2 + \beta_3 + \ldots$$

"Y" is a variable called "predicted_ratings" which contains the predicted ratings of our model in the validation dataset, "μ" is the mean of all ratings, and "β_x" the parameter for the variable "x".

The variable "predicted_ratings" will be then be fitted in our RMSE function that calculates the Root Mean Squared Error (RMSE):

```
RMSE <- function(true_ratings, predicted_ratings){
sqrt(mean((true_ratings - predicted_ratings)^2))}


model_rmse <- RMSE(predicted_ratings, validation$rating)
```

The RMSE is a measure of how much error we are making when estimating with our predictions. A good prediction and our goal will be an RMSE lower than 0.864999.
Anytime we add a parameter to our model we can compute the RMSE function to see how close we are getting to our objective, but here the code is omitted until the last parameter is added to avoid reiteration (RSME calculation will be shown at the end).

We start to model by implementing movie effects. This means that some movies are simply rated better just for being that movie. Here we do not care if the movie was actually good or bad. Including the parameter “b_1” in our model will improve our predictions because we will be taking into account that effect. This parameter is calculated by the average of the extraction of “μ” (average of ratings in edx dataset) from each rating (including this parameter reduced RMSE to 0.9439).

```
mu <- mean(edx$rating)
movie_avgs <- edx %>%
    group_by(movieId) %>%
    summarize(b_1 = mean(rating - mu))
```

The next step is to implement the user effect. Maybe some users tend to rate the movies they watch with a lower rate than expected, considering how much they liked it. On the other hand, some other users may do the opposite by always rating higher than expected. We can control this effect by adding to our model an extra parameter “b_2”. We do this the exact same way as before but now using b_1 as well. The code to calculate b_2 and reduce the RMSE to 0.8653 is:

```
user_avgs <- edx %>%
    left_join(movie_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_2 = mean(rating - mu - b_1))
```

Plotting the average rating per year, we saw that there is a slightly effect of time over the average rating. With this information it is a good idea that we include a parameter to our model that takes into account the passage of time. To include this parameter we follow the same steps as before. The problem here is that we will find a lot of “NA” if we have our timestamp in seconds, that is because not every film is rated every second. If we divide timestamp by 86400 (or by 60, then 60 and then 24) and round it to the nearest integer we transform timestamp from seconds to days.
Once we have the new parameter to include the passage of time to our model (b_3) the RSME is reduced to 0.8648345.

```
time_avgs <- edx %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by= 'userId') %>%
    mutate(timestamp = timestamp/60/60/24) %>%
    mutate(timestamp = round(timestamp, digits = 0)) %>%
    group_by(timestamp) %>%
    summarize(b_3 = mean(rating - mu - b_1 - b_2))
```

Another variable we can take into account is the genre of the films. As discussed before, a lot of movies have more than one genre attached to them but to simplify we considered every combination of different genres that every movie has as one genre. Grouping by genre and taking the average rating we saw that some genres have lower or higher ratings than others. This code generates our fourth parameter "b_4" that reduces the RMSE to 0.8644346.

```
genres_avgs <- edx %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by= 'userId') %>%
    mutate(timestamp = timestamp/60/60/24) %>%
    mutate(timestamp = round(timestamp, digits = 0)) %>%
    left_join(time_avgs, by= 'timestamp') %>%
    group_by(genres) %>%
    summarize(b_4 = mean(rating - mu - b_1 - b_2 - b_3))
```

RESULTS:

Now we have four parameters than we can use with "mu" to predict ratings in the validation set.

We can calculate the RMSE with the code below (Remember that we have to change the timestamp from seconds to days in "validation" dataset too so our averages datasets can be joined):

```
predicted_ratings <- validation %>%
    mutate(timestamp = timestamp/60/60/24) %>%
    mutate(timestamp = round(timestamp, digits = 0)) %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
```

```
    left_join(time_avgs, by='timestamp') %>%
    left_join(genres_avgs, by= 'genres') %>%
    mutate(pred = mu + b_1 + b_2 + b_3 + b_4) %>%
    pull(pred)

model_rmse <- RMSE(predicted_ratings, validation$rating)
model_rmse

## [1] 0.8644346
```

CONCLUSION:

We achieved a 0.8644346 RMSE, this means that on average our prediction will be accurate and only miss by 0.8644346 stars when we predict the rating. Now, for a user that we predict that he or she will rate a movie around 4 stars, we can recommend that movie to that user because he or she will probably like it.