Movielens Project Report

Manuel Alexis Mena Nieves

June 23, 2020

1. Introduction

This project report is about creating a movie recommendation system for the HarvardX Data Science professional certificate program, using the 10M version of the Movielens Dataset.

This dataset describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service, with a 'movieId' for each movie, and an anonymized 'userId' for each user who provided one or more movie reviews.

The goal of the project is to create an algorithm to predict how users will rate movies using all the tools shown throughout the courses in this series. To accomplish this goal an exploratory analysis was done, in order to understand the data and summarize their main characteristics with tables and visual methods. After this, a machine learning model was created to predict how each user would rate movies.

2. Getting the data

The following code will be used to download the dataset. We begin loading the libraries tidyverse, caret and data.table:

```
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")
```

As the file containing the dataset is large, the script checks its availability in project folder. If not, we proceed to download the file from Grouplens site and make it a dataframe:

```
if (!file.exists("ml-10M100K/ratings.dat") || !file.exists("ml-10M100K/movies.dat")) {
  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)

} else {
  ratings <- fread(text = gsub("::", "\t", readLines("ml-10M100K/ratings.dat")),
                   col.names = c("userId", "movieId", "rating", "timestamp"))
  movies <- str_split_fixed(readLines("ml-10M100K/movies.dat"), "\\::", 3)
```

```
}

colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
```

We proceed to divide the movielens dataframe created into two datasets, one for training and the other one for validation.

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

# if using R 3.5 or earlier, use `set.seed(1)` instead
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 training set name is edx, and it will be used for exploratory analysis and modelling, according to the findings we will make.

3. Exploratory Analysis

The first step in the data analysis process is to make sense of the data we have and then figure out what questions we want to ask and how to frame them, as well as how best to manipulate the data to get the answers we need. This process is called exploratory analysis, where we make a broad look at patterns, trends, outliers and unexpected results in the existing data. Firstly, we want to know the dimension of the edx dataset:

```
dim(edx)
```

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

There are 9,000,055 observations of 6 variables. Let's look the name of each variable:

```
as_tibble(edx)
```

```
## # A tibble: 9,000,055 x 6
##    userId movieId rating timestamp title                  genres
##     <int>   <dbl>  <dbl>     <int> <chr>                  <chr>
##  1      1     122      5 838985046 Boomerang (1992)       Comedy|Romance
##  2      1     185      5 838983525 Net, The (1995)        Action|Crime|Thriller
##  3      1     292      5 838983421 Outbreak (1995)        Action|Drama|Sci-Fi|T~
##  4      1     316      5 838983392 Stargate (1994)        Action|Adventure|Sci-~
##  5      1     329      5 838983392 Star Trek: Generation~ Action|Adventure|Dram~
##  6      1     355      5 838984474 Flintstones, The (199~ Children|Comedy|Fanta~
##  7      1     356      5 838983653 Forrest Gump (1994)    Comedy|Drama|Romance|~
##  8      1     362      5 838984885 Jungle Book, The (199~ Adventure|Children|Ro~
##  9      1     364      5 838983707 Lion King, The (1994)  Adventure|Animation|C~
## 10      1     370      5 838984596 Naked Gun 33 1/3: The~ Action|Comedy
## # ... with 9,000,045 more rows
```

We can see that we have the variables userId, movieId, rating, timestamp, title and genres. Let's make a summary to understand the rating data:

```
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 mean rating is 3.512, and the median rating is 4. We can see that the lowest rating is 0.5 and the highest is 5. The interquartile range is 1.

We have a timestamp variable which is not easy to understand, and the title variable has the year in parenthesis, so we must convert the timestamp to a common date value, and extract the year from the title using regex:

```
# Extract the date from edx and validation sets
edx <- edx %>%
  mutate(date = as.Date.POSIXct(timestamp)) %>% select(-timestamp)
validation <- validation %>%
  mutate(date = as.Date.POSIXct(timestamp)) %>% select(-timestamp)
```

```
# Use regex to extract the year from the movie title
edx <- edx %>%
  mutate(year = as.integer(str_extract(str_extract(title,"\\((\\d{4})\\)$"),"\\d{4}")))
validation <- validation %>%
  mutate(year = as.integer(str_extract(str_extract(title,"\\((\\d{4})\\)$"),"\\d{4}")))
```

We can see the number of unique users and how many unique movies were rated:

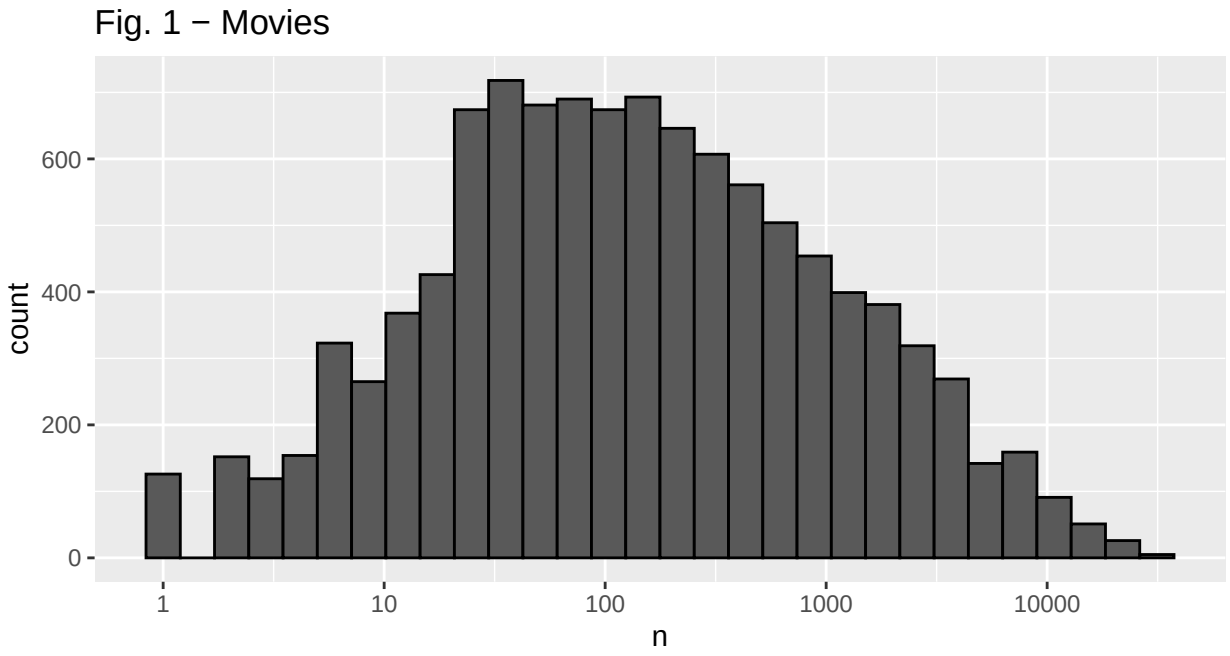
```
edx %>% summarize(n_movies = n_distinct(movieId),
                  n_users = n_distinct(userId),
                  n_years = n_distinct(year))
```

```
##   n_movies n_users n_years
## 1    10677   69878      94
```

We have 10,677 movies, 69,878 users and 94 years of movies.If we multiply the number of movies and users we get a number larger than the observations in edx dataset. This implies that not every user rated every movie.

Let's look at some of the general properties of the edx dataset, we start with an histogram of movies:

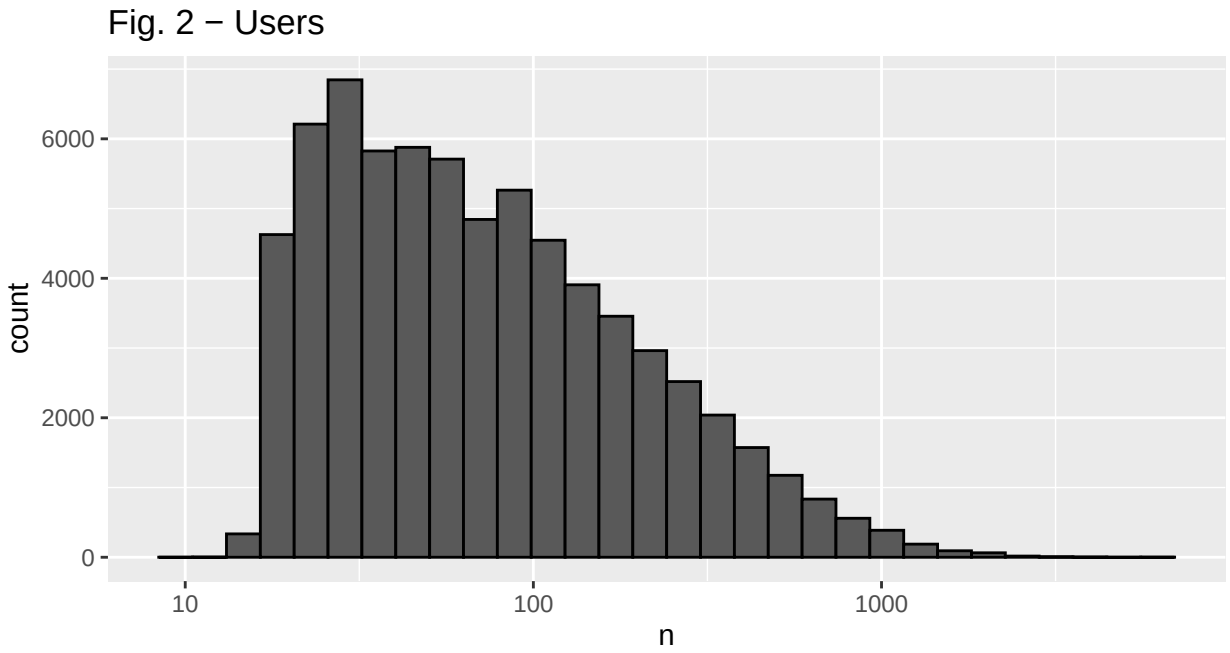
```
edx %>% count(movieId) %>% ggplot(aes(n)) +
  geom_histogram(bins = 30, color="black") + scale_x_log10() + ggtitle("Fig. 1 - Movies")
```



The figure 1 shows that some movies get rated more than others. This is because some movies are more popular and are watched by millions, while some artsy, independent movies are watched by just a few.

Now let's take a look at a histogram of users, with the following plot:

```
edx %>% count(userId) %>% ggplot(aes(n)) +
  geom_histogram(bins = 30, color="black") + scale_x_log10() + ggtitle("Fig. 2 - Users")
```



In figure 2 we see that some users are more active than others at rating movies. We must remember that not all movies are reviewed by all users, as seen in this histogram.

We already know the mean and median of ratings, let's see how the ratings are distributed:

```
edx %>% group_by(rating) %>% tally()
```

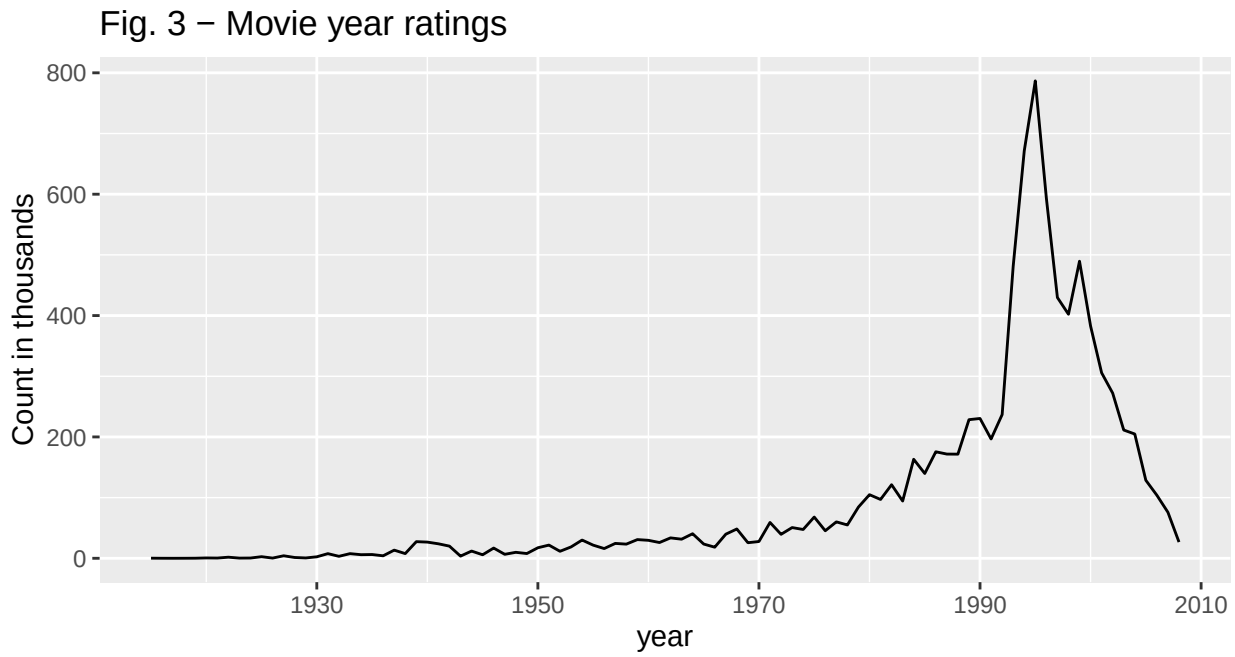
```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    0.5   85374
##  2    1    345679
##  3    1.5  106426
##  4    2    711422
##  5    2.5  333010
##  6    3   2121240
##  7    3.5  791624
##  8    4   2588430
##  9    4.5  526736
## 10    5   1390114
```

Half star ratings are less common than whole star ratings, as we see for example 2,121,240 three star rating versus 330,010 two and half rating, or 791,624 three and half rating.

Let's see now the count of ratings per movie year, in the following line plot:

```
edx %>% group_by(year) %>% summarize(count=n()/1000) %>% ggplot(aes(year,count)) +
  geom_line() + ggtitle("Fig. 3 - Movie year ratings") + ylab("Count in thousands")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

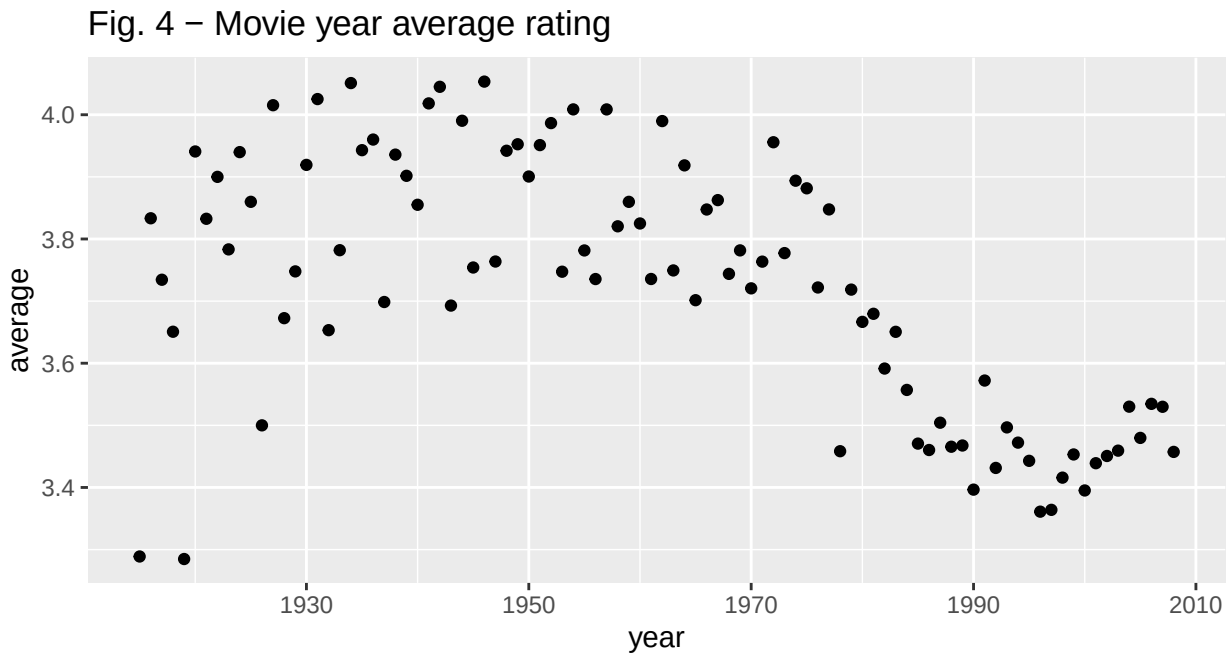


The figure 3 shows that movies that were released from 1980 and after have more ratings than the movies released before. We can imply that the exposure of the users is higher for recent movies.

Now we want to know what is the average rating per movie year, using a dot plot:

```
edx %>% group_by(year) %>%
  summarize(average = mean(rating)) %>% ggplot(aes(year,average)) +
  geom_point() + ggtitle("Fig. 4 - Movie year average rating")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

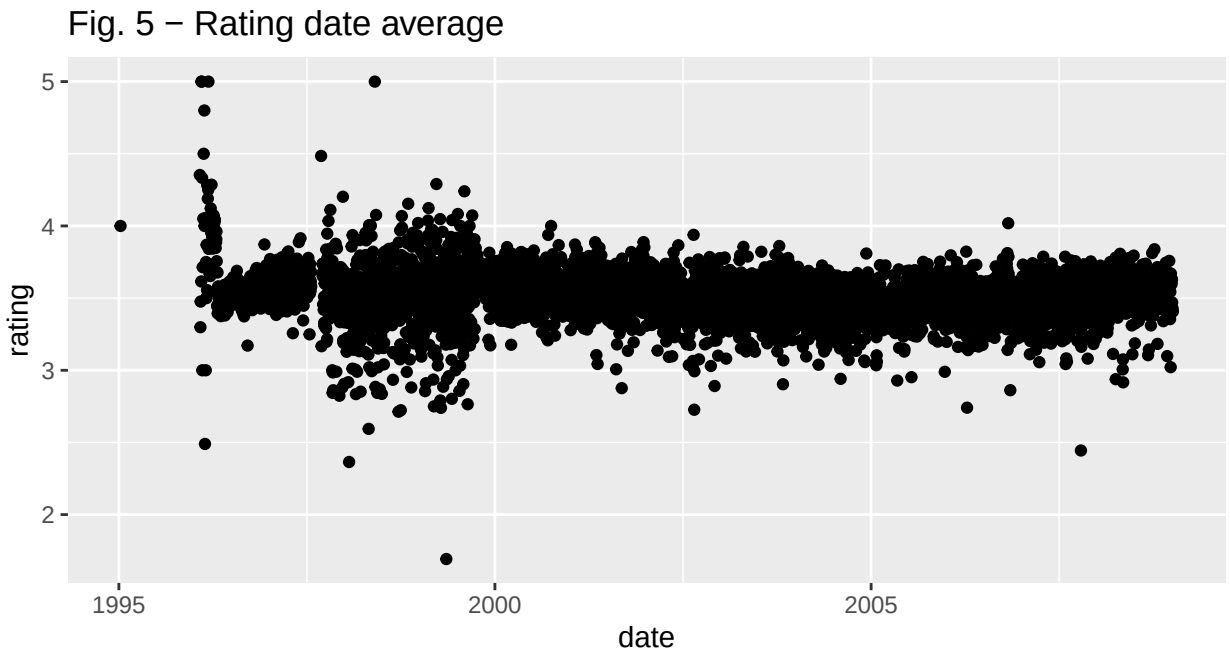


Interestingly, in figure 4 we see the older the movie the better average rating it has.

The edx dataset has ratings from 1995 onward, let's see if there is a variation in the average rating using a dot plot:

```
edx %>% group_by(date) %>%
  summarize(rating = mean(rating)) %>% ggplot(aes(date,rating)) + geom_point() +
  ggtitle("Fig. 5 - Rating date average")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



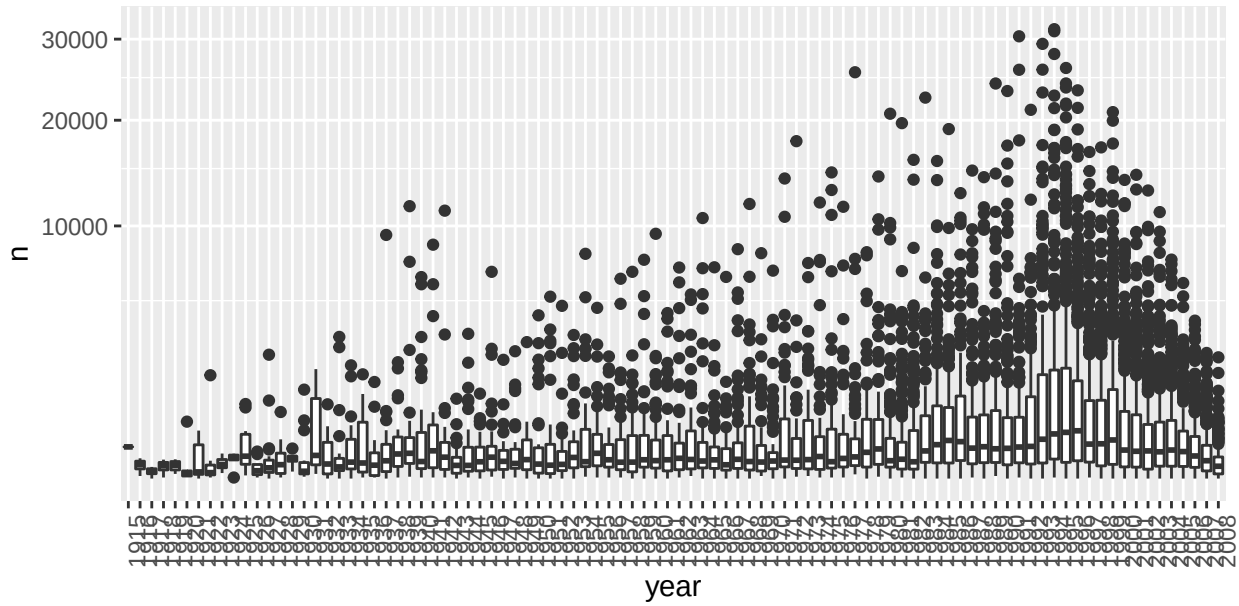
In figure 5 we don't see any noticeable changing trend in average rating per date, as opposed to what we have seen in figure 4.

Now we can compute the number of ratings for each movie and plot it against the year it came out in the following graphic:

```
edx %>% group_by(movieId) %>% summarize(n=n(),year=as.character(first(year))) %>%
  qplot(year,n,data=.,geom="boxplot", main = "Fig. 6 - Rating quantity per year Boxplot ") +
  coord_trans(y="sqrt") + theme(axis.text.x=element_text(angle=90,hjust=1))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

Fig. 6 − Rating quantity per year Boxplot



In figure 6 we see that the year 1996 has the highest median number of ratings.

4. Modelling the data

The goal of developing models in machine learning is to extract insights from data that you can then use to make better decisions and in our case, to make better recommendations. Algorithmic models tell you which outcome is likely to hold true for your target variable based on your training data. They construct a representation of the relationships and tease out patterns between all the different features in our dataset that you can apply to similar data you collect in the future, allowing you to make decisions based on those patterns and relationships.

Since we have a training data set named edx, we further divide it into a train set of 80% observations and a test set of 20%. This is to avoid using our validation dataset because like a real model we use the validation as we don't know its outcomes. The train and test set will be used for tuning our model.

```
#   Firstly we create a training and test set from edx dataset, in order to tune our model
set.seed(1996)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,list = FALSE)
train_set <- edx[-test_index,]
test_set_tmp <- edx[test_index,]
test_set <- test_set_tmp %>%
  semi_join(train_set, by = "movieId") %>% semi_join(train_set, by = "userId")
train_set <- rbind(train_set, anti_join(test_set_tmp, test_set))
```

```
## Joining, by = c("userId", "movieId", "rating", "title", "genres", "date", "year")
```

```
rm(test_set_tmp)
```

Then we create a basic model, which is simply the mean of the rating.

```
#   Sencondly, we apply a basic model on training dataset:
mu <- mean(edx$rating)
mu
```

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

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

Then we create a function for the RMSE, using the difference of the tue ratings versus our first prediction.

```
#   And a basic RMSE function to evaluate
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Since we have this naive approach, we can use this for to see how well we did by seeing the RMSE results.

```
#   3.1 Naive approach
mu <- mean(edx$rating)
naive_rmse <- RMSE(test_set$rating,mu)
rmse_results <- data_frame(Method = "Just the average", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

Now we evaluate what is the effect of the movie ratings.

```
#   3.2 Movie effect
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)
movie_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Movie effect",
                                     RMSE = movie_effect_rmse))
```

Then we calculate the effect of the users on the rating.

```
#   3.3 User effect
user_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>% mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
user_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "User effect",
                                     RMSE = user_effect_rmse))
```

As we saw before, the year of release of the movie is important in our model, so lets use the year effect in our model:

```
#   3.4 Year effect
year_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "year") %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)
RMSE(predicted_ratings, test_set$rating)
```

```
## [1] 0.8659845
```

Now we have to regularize our model using a lambda sequence.

```
#   3.4 Regularized movie + user effect
lambdas <- seq(0,10,0.25)

rmses <- sapply(lambdas, function(l){
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating -  mu)/(n()+l))

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

  # Year effect
  b_y <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
```

```r
    group_by(year) %>%
    summarize(b_y = sum(rating - b_u - b_i - mu)/(n()+l))

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
    mutate(pred = mu + b_i + b_u + b_y) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
```

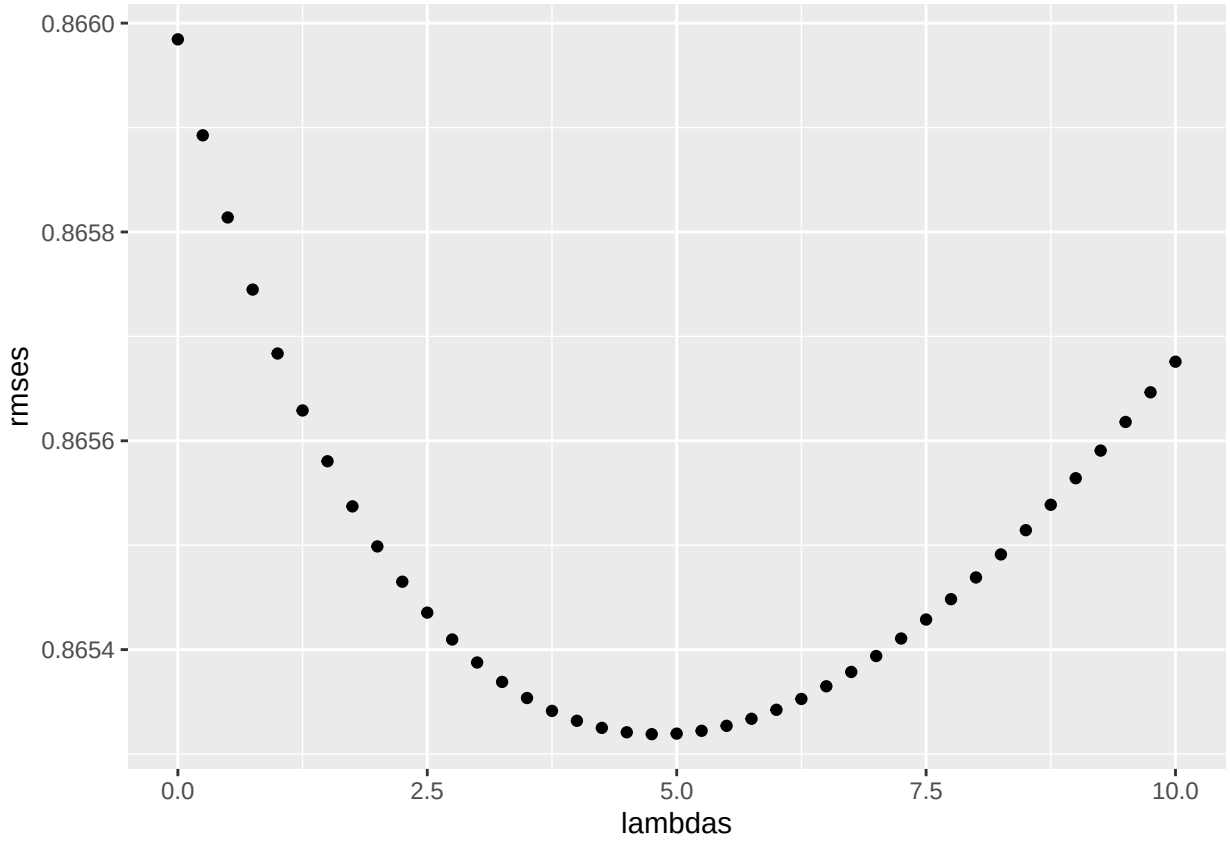
```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
## `summarise()` ungrouping output (override with `.groups` argument)
```

Now we can see what is the best

```r
# Lets make a lambdas vs rmses plot to see where is the best lambda
qplot(lambdas,rmses)
```




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

```
## [1] 4.75
```

```
min(rmses)
```

```
## [1] 0.865319
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Regularized movie + user + year effect",
                                     RMSE = min(rmses)))
```

```
# Test the result against validation set

b_i_tuned <- edx %>%
  group_by(movieId) %>%
  summarize(b_i_tuned = sum(rating - mu)/(n() + best_lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
b_u_tuned <- edx %>%
  left_join(b_i_tuned, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u_tuned = sum(rating - b_i_tuned - mu)/(n() + best_lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
b_y_tuned <- edx %>%
  left_join(b_i_tuned, by = "movieId") %>%
  left_join(b_u_tuned, by = "userId") %>%
  group_by(year) %>%
  summarize(b_y_tuned = sum(rating - b_i_tuned - b_u_tuned - mu)/(n() + best_lambda))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings <- validation %>%
  left_join(b_i_tuned, by = "movieId") %>%
  left_join(b_u_tuned, by = "userId") %>%
  left_join(b_y_tuned, by = "year") %>%
  mutate(pred = mu + b_i_tuned + b_u_tuned + b_y_tuned) %>%
  pull(pred)
final_rmse <- RMSE(predicted_ratings, validation$rating)
final_rmse
```

```
## [1] 0.8645223
```

5. Conclusions

- By applying a prediction based on ratings by user, movie and year we could get an accuracy of
- It's tempting to think that this model can be used for users who has not rated yet a movie, but it can be useful as a common recommendation from other users.