Professional Certificate in Data Science

Movielens Project

Elaborated by: Marco Méndez Atienza

July 20, 2021

Contents

1 Introduction

A recommendation system is a process that tries to predict user preferences. They can cover a large range of techniques and be useful from simple predictions to more advanced topics like matrix factorization. They function as a data filtering tool and on the principle of finding patterns in behavior data.

These days, recommendation systems are present in a large amount of places. The most important sites in the world use it: Amazon, YouTube, Netflix, etc. Speaking of the last, the Netflix prize ("an open competition for the best collaborative filtering algorithm to predict user ratings for films" ("Netflix Prize," 2021)) is a very good example of rating-based recommendation systems used in a high demand context.

In the next pages, de 10M data set of MovieLens, created by the University of Minnesota, will be used. The report is split into four sections: first, the data set is loaded and observed for further statistical analysis; second, an exploratory data analysis (EDA) is performed, so the structure of the data set is clear; finally, the machine learning techniques are applied in order to achieve an acceptable algorithm of prediction.

As we know, a common value used to evaluate an algorithm is the Root Mean Square Error (RMSE), which measures the quantity of the error between two data sets. In other words, it compares a predicted value with an observed or known value. Thus, RMSE is a measure of accuracy and very useful in predictive algorithms.

Because of that, four different models will be fitted and compared between them using their RMSE in order to assess their accuracy and predictive power. The following formula will be used as the computation for RMSE:

$$\mathbf{RMSE} = \sqrt{\frac{\mathbf{1}}{\mathbf{N}}\sum_{\mathbf{u,i}}(\hat{\mathbf{y}}_{\mathbf{u,i}} - \mathbf{y}_{\mathbf{u,i}})^{\mathbf{2}}}$$

Thus, the main goal is to minimize this function.

2 Data set

The following code, provided in the course materials ("HarvardX Data Science Professional Certificate," n.d.), is used to generate the data set of MovieLens:

```
############################################################
# 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")

library(tidyverse)
library(caret)
library(data.table)
library(ggthemes)
library(ggrepel)
library(viridis)
library(RColorBrewer)

# 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)
```

3 Data structure

Firstly, we can see that the data set is, in fact, very large. It contains 9 000 055 observations and 6 different features.

```
dim(edx)
```

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

This represents a challenge in terms of the reproducibility and practicality of the code, since it can take a lot of time to compute the information wanted. Also, as we can see, the data set gives an ID to every unique user and movie, so a movie can be rated by several users, and a single user can rate a lot of movies.

```
#use of head() to overview the general setup of the dataset

head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

It is worth noticing that the ratings include no zeros and they take values between 0.5 and 5:

```
#table() shows all the frequencies for every value "rating" takes

table(edx$rating)
```

```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
##   85374  345679  106426  711422  333010 2121240  791624 2588430  526736 1390114
```

So it is a very large data set, but how many unique movies and users includes?

Unique movies:

```
# unique() creates a vector or unique elements in "movieId"; length() displays its size

length(unique(edx$movieId))
```

```
## [1] 10677
```

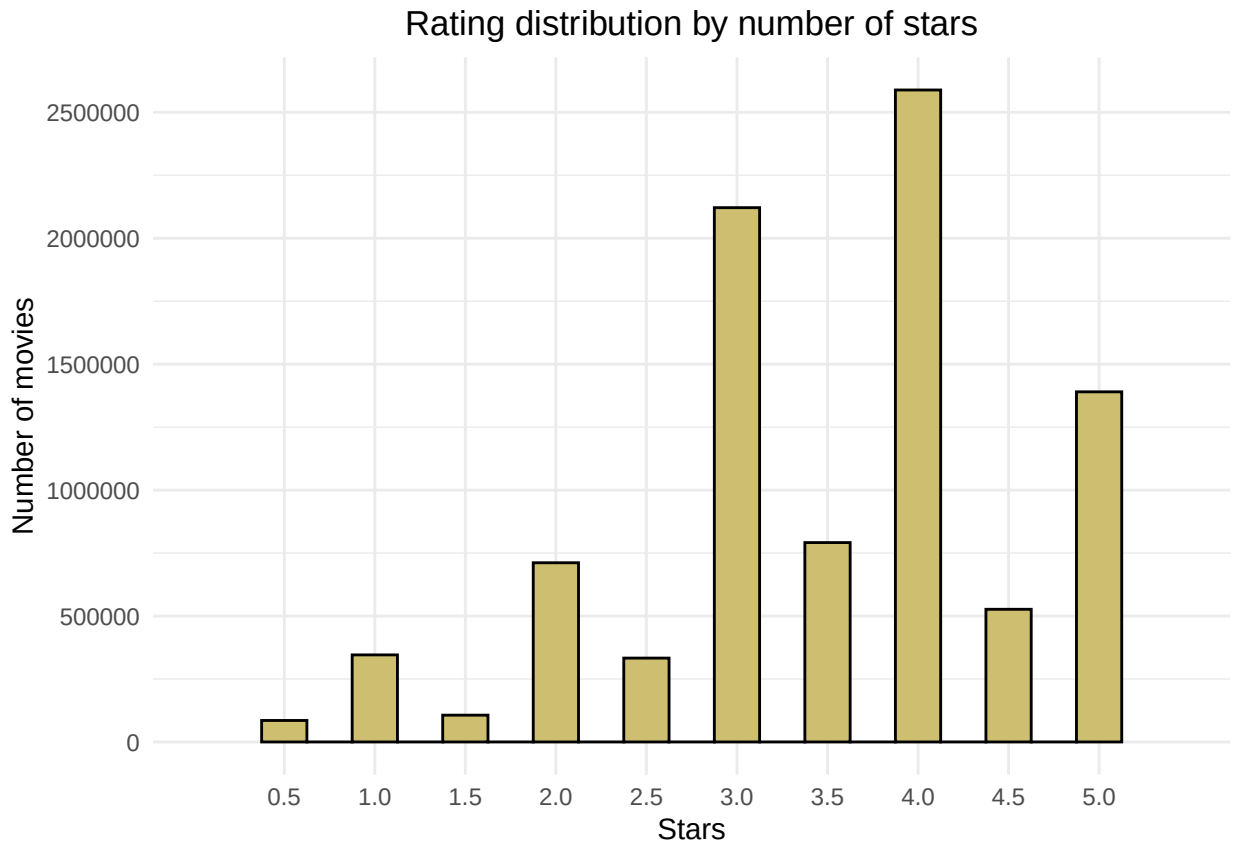
Unique users:

```
length(unique(edx$userId))
```

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

Is there something to say about the distributions of the ratings? As we can see in the graph below, there is a tendency to rate the movies with higher number of stars:

```
#use of ggplot and ggthemes

edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, color = "black", fill = "lightgoldenrod3") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  ggtitle("Rating distribution by number of stars") +
  xlab("Stars") +
  ylab("Number of movies") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



As it can be seen, 4 and 3 stars are the most common ratings given to a movie:

```
# the following code shows the frequency of every rating in descending order

edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  top_n(10, count) %>%
  arrange(desc(count))
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
```

```
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

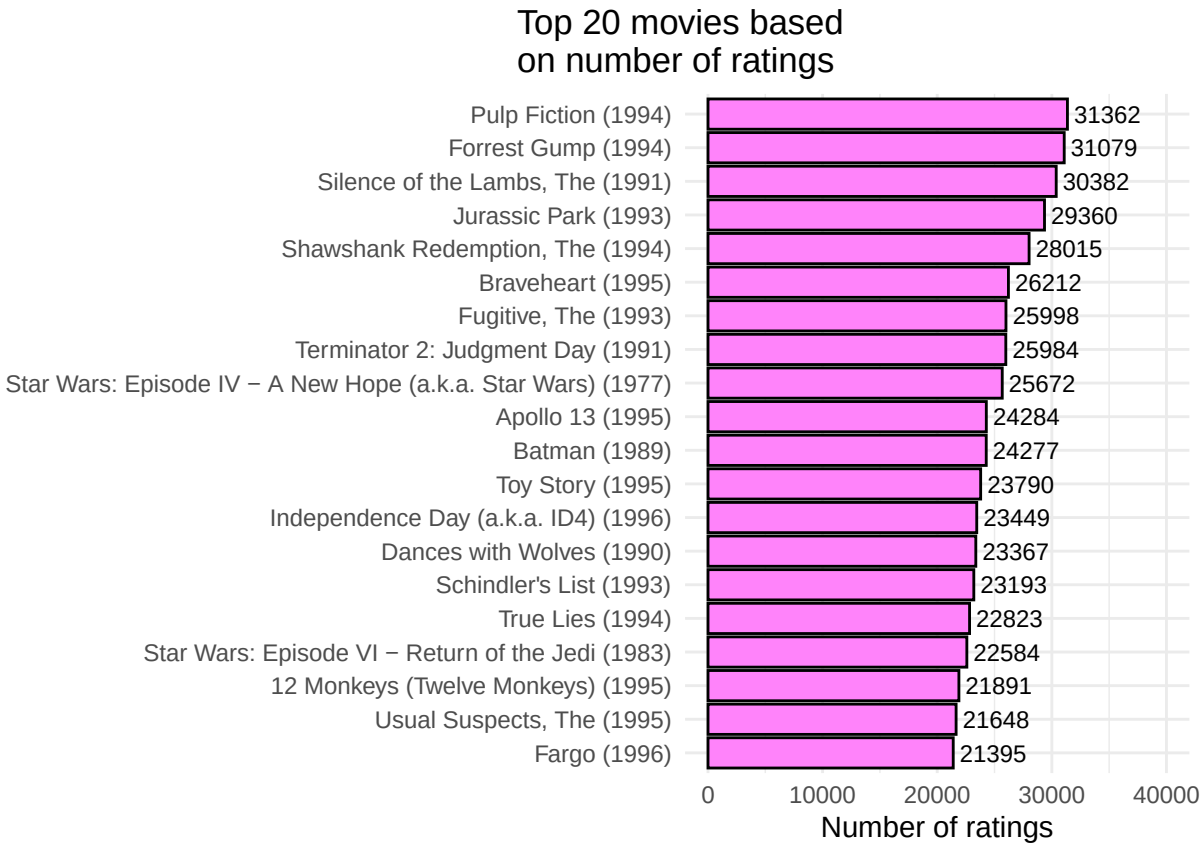
We can also see which movies received the largest number of ratings:

```
# this shows the movies with most ratings in descending order

edx %>%
  group_by(movieId) %>%
  summarize(Ratings = n(), Title = first(title)) %>%
  arrange(desc(Ratings)) %>%
  top_n(15, Ratings)
```

```
## # A tibble: 15 x 3
##    movieId Ratings Title
##      <dbl>   <int> <chr>
##  1     296   31362 Pulp Fiction (1994)
##  2     356   31079 Forrest Gump (1994)
##  3     593   30382 Silence of the Lambs, The (1991)
##  4     480   29360 Jurassic Park (1993)
##  5     318   28015 Shawshank Redemption, The (1994)
##  6     110   26212 Braveheart (1995)
##  7     457   25998 Fugitive, The (1993)
##  8     589   25984 Terminator 2: Judgment Day (1991)
##  9     260   25672 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
## 10     150   24284 Apollo 13 (1995)
## 11     592   24277 Batman (1989)
## 12       1   23790 Toy Story (1995)
## 13     780   23449 Independence Day (a.k.a. ID4) (1996)
## 14     590   23367 Dances with Wolves (1990)
## 15     527   23193 Schindler's List (1993)
```

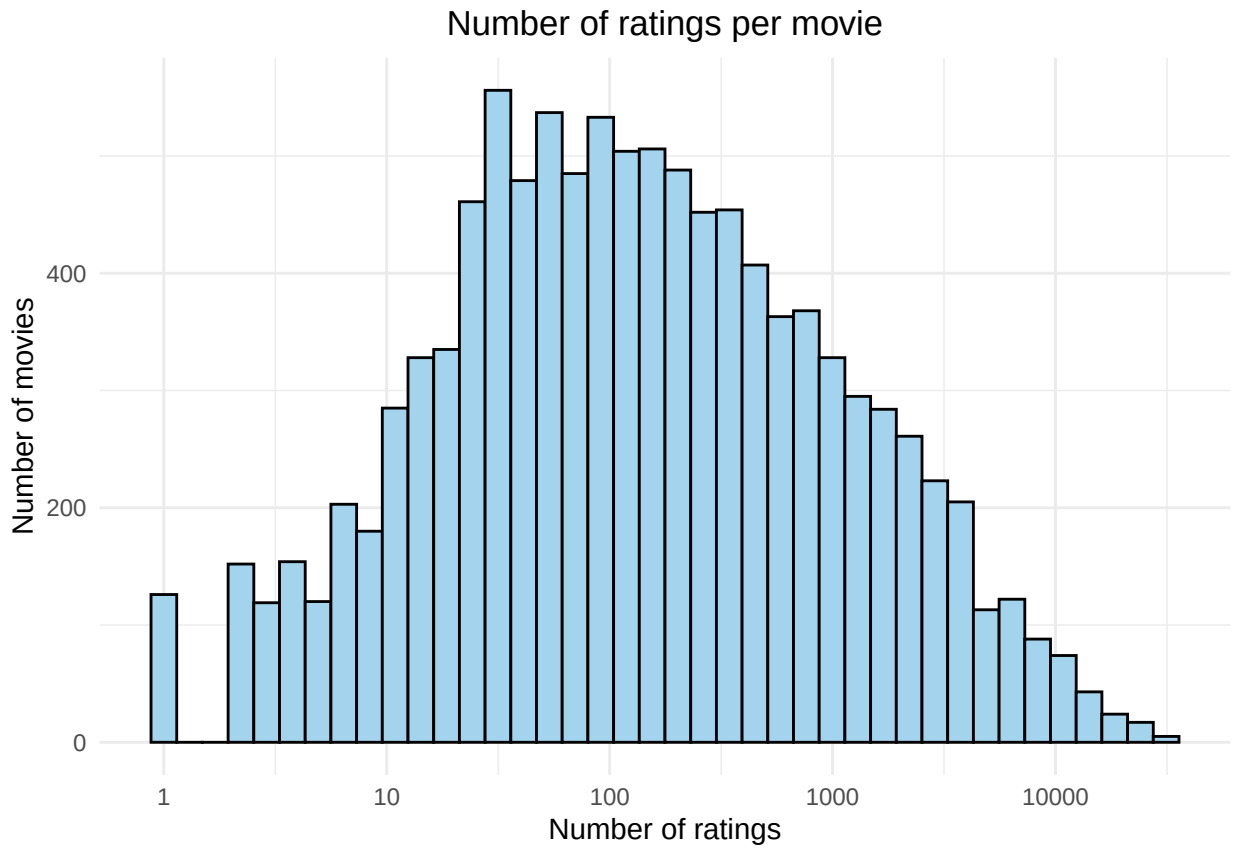
```
# first, creation of an object with the 20 most rated movies

top_rated_movies <- edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  top_n(20,count) %>%
  arrange(desc(count))

# then, graph of it reordering the list so the movies are in descending order


  top_rated_movies %>%
    ggplot(aes(x=reorder(title, count), y=count)) +
    geom_bar(stat = "identity", color = "black", fill = "orchid1") +
    coord_flip(y=c(0, 40000)) +
    xlab("") +
    ylab("Number of ratings") +
```

```
    geom_text(aes(label = count), hjust = -0.1, size = 3) +
    ggtitle("Top 20 movies based \n on number of ratings") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = -1))
```



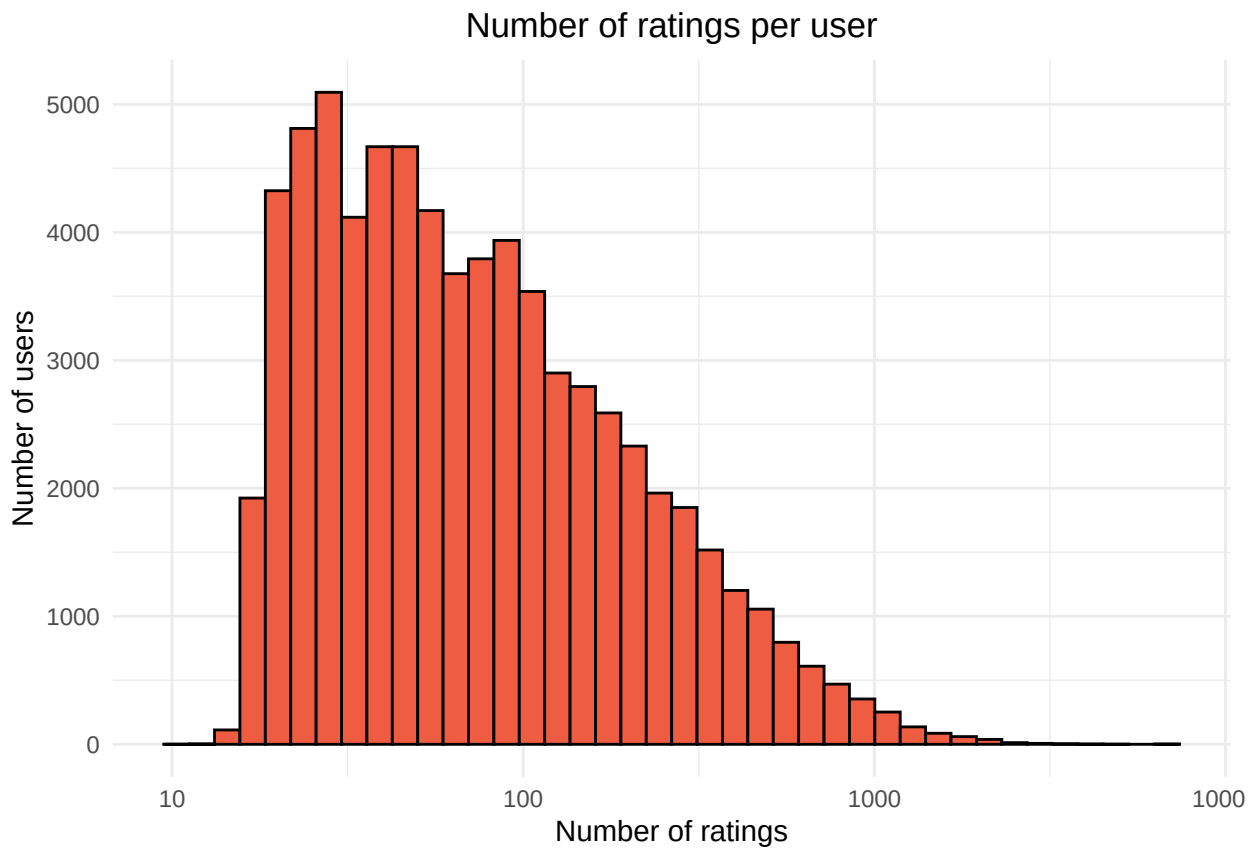
It is useful to also see the distribution of ratings: some movies have been rated much more often that others:

```
# the following graph shows the distributions of number of ratings per movie

edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 40, color = "black", fill = "lightskyblue2") +
  scale_x_log10() +
  ggtitle("Number of ratings per movie") +
  xlab("Number of ratings") +
  ylab("Number of movies") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



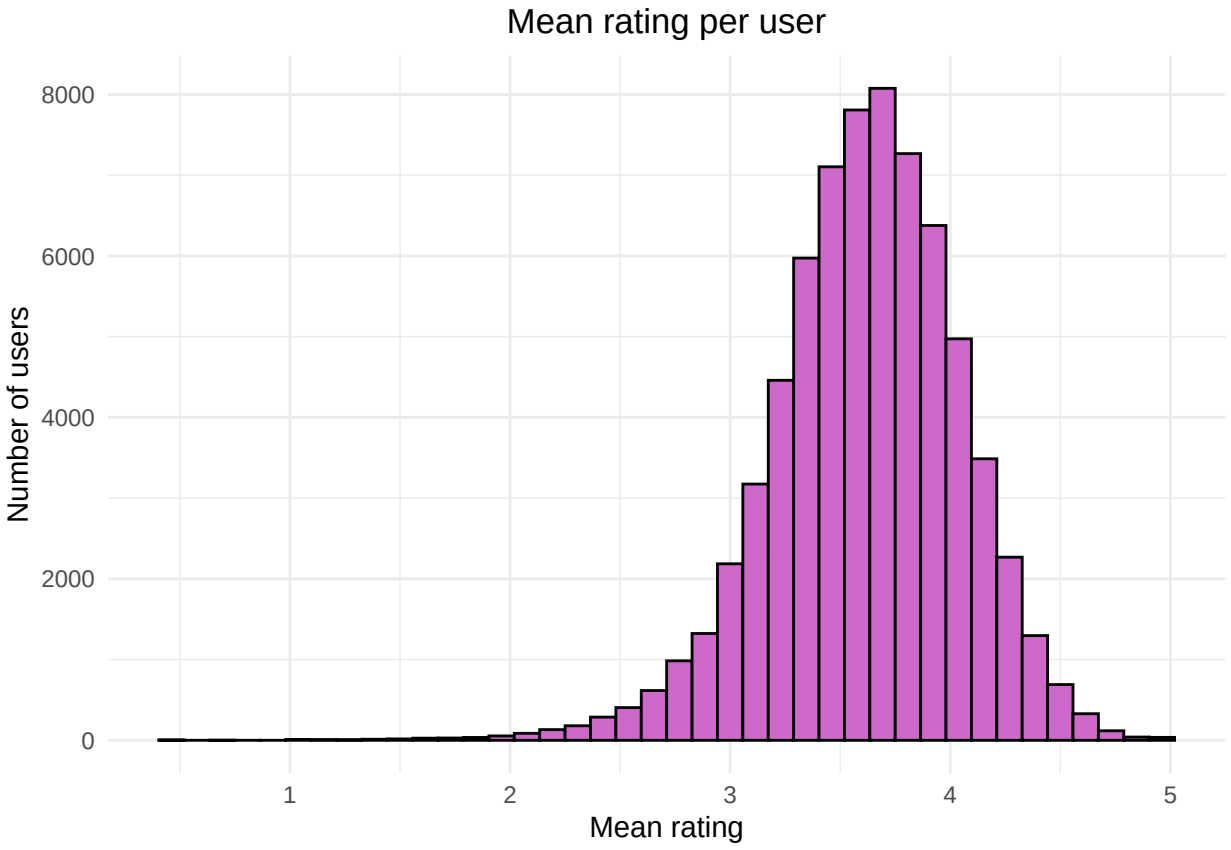
How often the users rated movie? How many movies each user rated? To answer these questions, consider the next graph, where it is obvious that the majority of users rated between 10 and 30 movies, while a few rated more that 500:

```
# this shows the distribution of number of ratings per user

edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 40, color = "black",  fill = "tomato2") +
  scale_x_log10() +
  xlab("Number of ratings") +
  ylab("Number of users") +
  ggtitle("Number of ratings per user") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



Finally, to wrap up this exploration data analysis, lets see how the mean movie ratings are distributed by user, since some users are far more critical with the movies than others:

```
# this graph is to show how the stars are distributed (it is clearly normal)

edx %>%
  group_by(userId) %>%
  summarize(avg = mean(rating)) %>%
  ggplot(aes(avg)) +
  geom_histogram(bins = 40, color = "black", fill = "orchid3") +
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Mean rating per user") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```


Mean rating per user
Number of users
8000
6000
4000
2000
0
1
2
3
4
5
Mean rating

4 Modeling

As said before, the loss function to minimize is defined as follows:

$$\mathbf{RMSE} = \sqrt{\frac{1}{\mathbf{N}}\sum_{\mathbf{u,i}}(\hat{\mathbf{y}}_{\mathbf{u,i}} - \mathbf{y}_{\mathbf{u,i}})^{\mathbf{2}}}$$

With N defined as the number of user/movie combinations and the sum running through all of them. This RMSE being the model accuracy measure, one can think of it as the error occurring when predicting some movie rating. Thus, if $RMSE > 1$, then the prediction is off by more than one star (given that the ratings $\in [0.5, 5]$, this would make a poor estimate).

Now, the computation of the RMSE can be coded as follows:

```
# creation of the pertinent function to use onwards

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

4.1 Average model

As seen during the course classes, a very basic model to predict the movie ratings is by calculating the mean rating. By computing the same rating for all movies, we can define a basic model that predicts ratings based on the mean and explains the differences with a random variation:

$$\mathbf{Y}_{\mathbf{u,i}} = \mu + \varepsilon_{\mathbf{u,i}}$$

Where $\varepsilon \sim (0, \sigma)$ and μ is the true rating. But how well does this approach minimize the RMSE? As seen in class, the estimate that minimize the RMSE is the least square of $Y_{u,i}$. For this particular case, it happens to be the mean of all ratings:

```
# first estimator, just the rating mean

mu_hat <- mean(edx$rating)

mu_hat
```

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

Then, we can predict all movie ratings using this μ through a naive approach:

```
# first RMSE, a naive approach

naive_RMSE <- RMSE(edx$rating, mu_hat)

naive_RMSE
```

```
## [1] 1.060331
```

Where it is clear that our prediction is pretty high. Also note that if we use any other number for our estimate $\hat{\mu}$, the RMSE will rise (this is obvious as the RMSE is minimized with our $\hat{\mu}$):

```
# example of higher RMSE using 3.5 (a number very close to mu_hat)

predictions <- rep(3.5, nrow(edx))
```

```
RMSE(edx$rating, predictions)
```

```
## [1] 1.060405
```

Before continuing with the improvement of our estimate, lets build a table to store all the results that are being obtained:

```
# storing of results in table using kable

RMSE_results <- data.frame(Method = "Average model", RMSE = naive_RMSE)

RMSE_results %>%
  knitr::kable(label = "RMSEs per method")
```

Method	RMSE
Average model	1.060331

So how can the estimate improve? Through Section 3 we obtained meaningful insights about the date, so the next models will use them to try to minimize our RMSE.

4.2 Movie effect model

Given the previous model, the movie effect can be incorporated by adding a factor that accounts for the mean rating for each movie i. This implies that the model is nos defined:

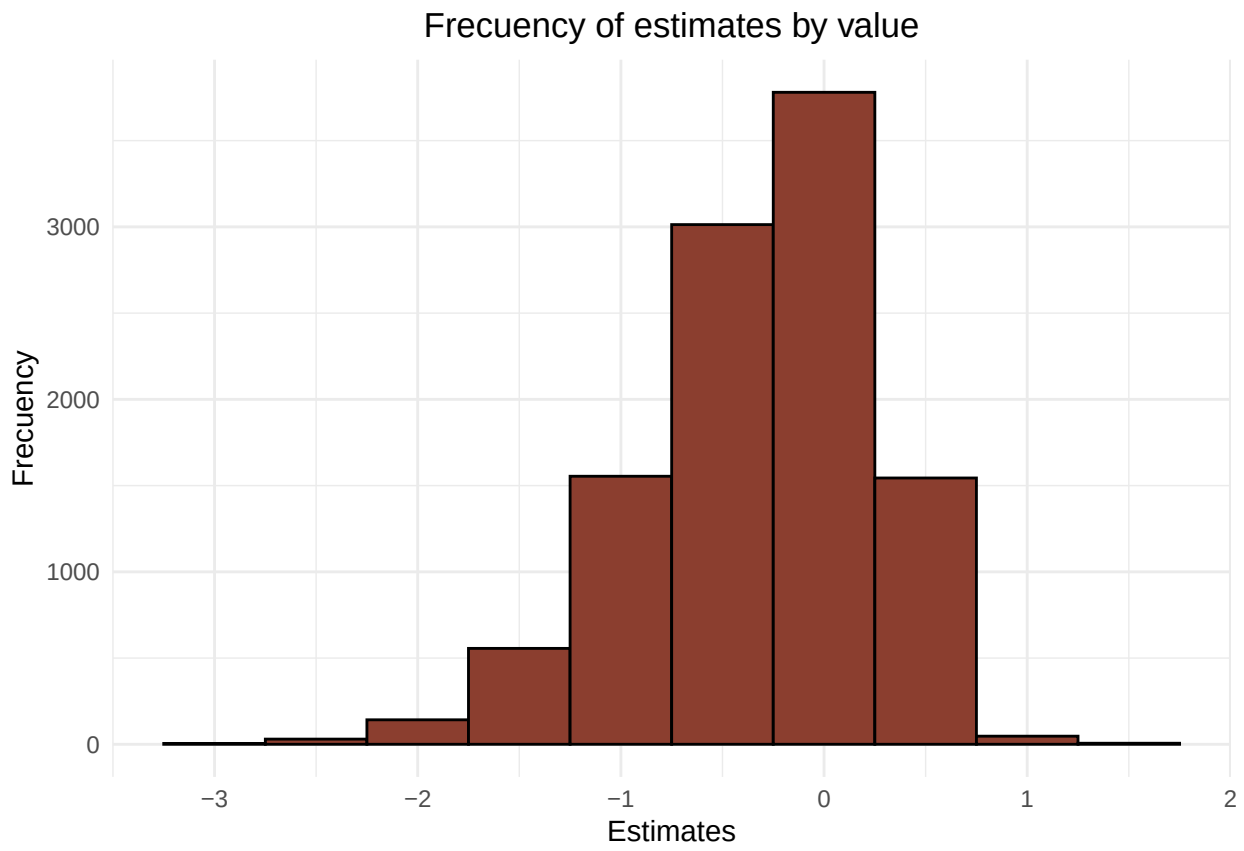
$$\mathbf{Y_{u,i}} = \mu + \mathbf{b_i} + \varepsilon_{\mathbf{u,i}}$$

Thus, it is known that the estimate $\hat{b}_i$ is just the mean of $y_{u,i}$ minus the overall mean for each movie i, so the code is as follows:

```
# second estimator calculated according to the upper formula

mu <- mean(edx$rating)

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

As it can be seen in the next graph, the estimates vary substantially:

```
# distribution of estimates

movie_avgs %>%
  ggplot(aes(b_i)) +
  geom_histogram(bins = 10, color = "black",  fill = "coral4") +
  xlab("Estimates") +
  ylab("Frecuency") +
  ggtitle("Frecuency of estimates by value") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



So how much the prediction is improved?

```
# calculation of second RMSE and storage of results

predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  .$b_i

model_1_RMSE <- RMSE(predicted_ratings, validation$rating)

RMSE_results <- bind_rows(RMSE_results,
                          tibble(Method = "Movie effect model",
                                 RMSE = model_1_RMSE))

RMSE_results %>%
  knitr::kable(label = "RMSEs per method")
```

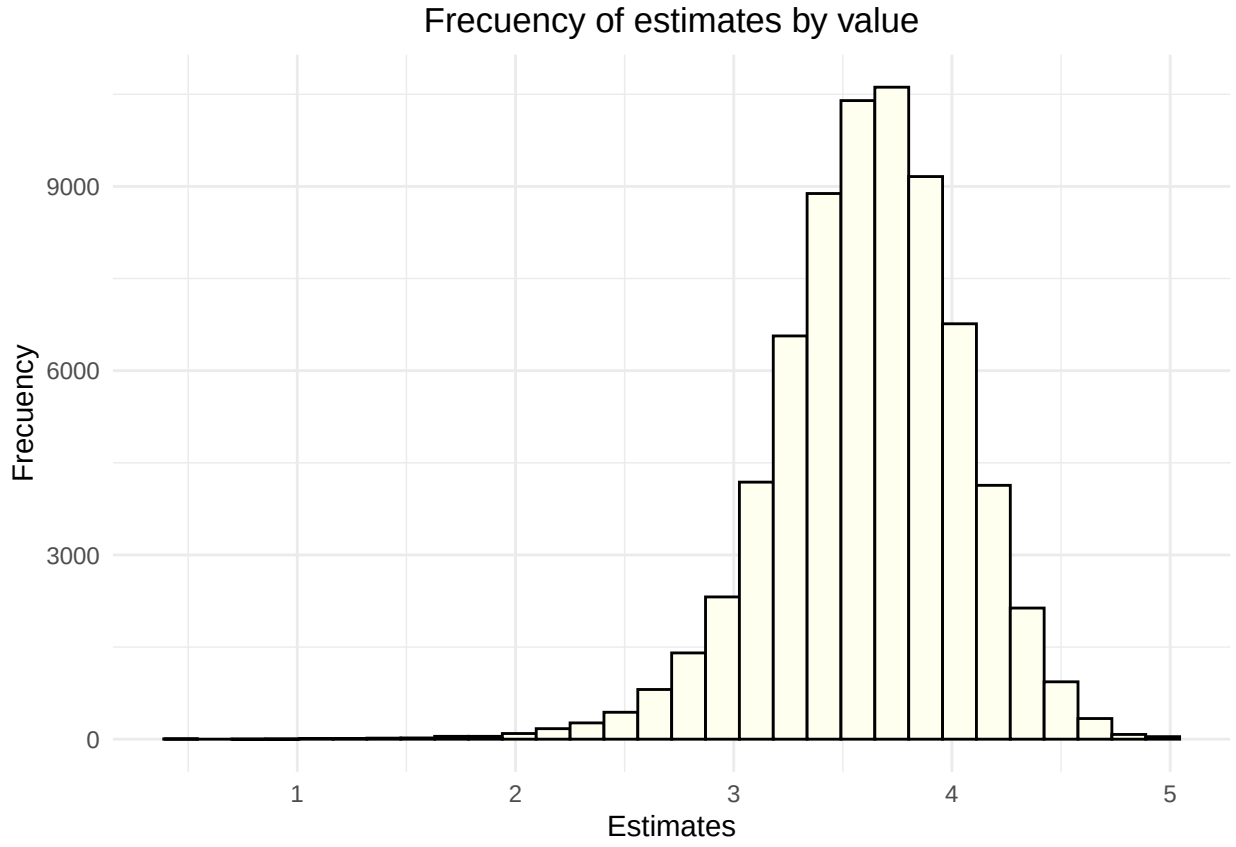
Method	RMSE
Average model	1.0603313
Movie effect model	0.9439087

This procedure is using a penalty term, in this case, a "penalty movie term." It is clear that our RMSE is in fact falling. Intuitively, the same task can be performed with users, as described in the next subsection

4.3 User + movie effects model

In this subsection, the mean rating for users that have rated more than 100 movies will be computed as follows:

```
# distribution of stars (users with more than 100 ratings)

edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n() >= 100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black",  fill = "ivory1") +
  xlab("Estimates") +
  ylab("Frecuency") +
  ggtitle("Frecuency of estimates by value") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



Assuming that the estimates follow a normal distribution, our model is now improved as follows:

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

For the model fitting, the general mean $\hat{\mu}$ and movie effect $\hat{b}_i$ will be calculated, so then the user effect $\hat{b}_u$ is estimated by subtracting the residual average obtained after removing the overall mean and movie effect from $y_{u,i}$:

```
# computation of third estimate

user_avgs <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

How much we improved?

```
# third RMSE and storage of results

predicted_ratings <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_2_rmse <- RMSE(predicted_ratings, validation$rating)

RMSE_results <- bind_rows(RMSE_results,
                          tibble(Method = "Movie + user effects model",
                                 RMSE = model_2_rmse))

RMSE_results %>%
  knitr::kable(label = "RMSEs per method")
```

Method	RMSE
Average model	1.0603313
Movie effect model	0.9439087
Movie + user effects model	0.8653488

Where it is clear that the last model used is the one that minimizes the most the RMSE. Nonetheless the very good estimation, one must consider that the models used take into account movies that are not very popular, which implies very few ratings for each one. Thus, the uncertainty is large, so larger estimates of b_i, either negative or positive, are more likely.

Because confidence intervals are not adequate for this models (given that the prediction must be a single value), regularization will be used to improve the estimations.

4.4 Regularization

To estimate the parameters b, now the objective function to minimize, which includes a penalty, is defined as follows:

$$\frac{\mathbf{1}}{\mathbf{N}}\sum_{\mathbf{u,i}}(\mathbf{u_{u,i}} - \mu - \mathbf{b_i})^{\mathbf{2}} + \lambda\sum_{\mathbf{i}}\mathbf{b_i^2}$$

Where the first term is the RMSE and the second the penalty given to the terms that increase when b does. Thus, the values of b which minimize this equation are:

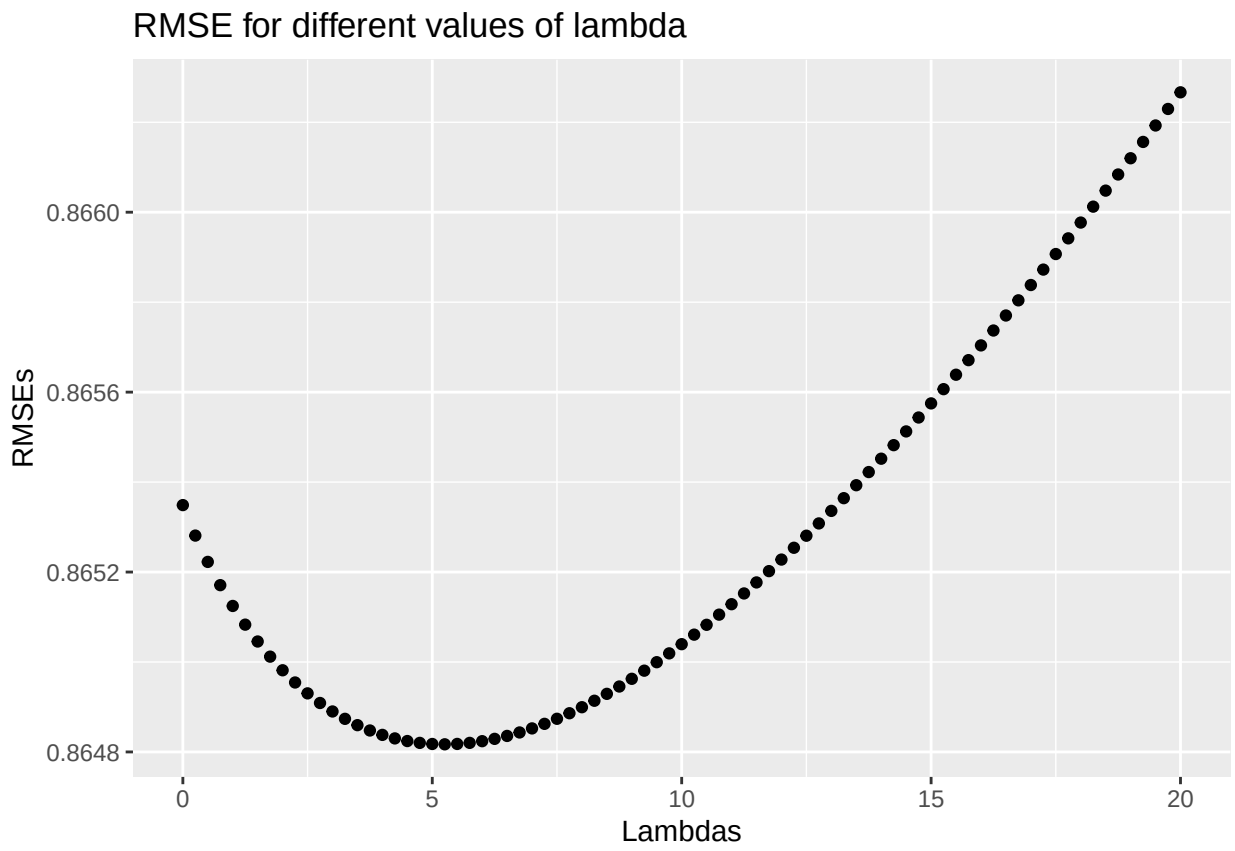
$$\hat{\mathbf{b}}_\mathbf{i}(\lambda) = \frac{\mathbf{1}}{\lambda + \mathbf{n_i}}\sum_{\mathbf{u=1}}^{\mathbf{n_i}}(\mathbf{Y_{u,i}} - \hat{\mu})$$

So the parameter to optimize is , in this case, λ:

```
# definition of function to test 80 different lambdas, then computation of estimates, then computation

lambdas <- seq(0, 20, 0.25)

RMSEs <- sapply(lambdas, function(l){

  mu <- mean(edx$rating)

  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l), n_i=n())

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

  predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred= mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, validation$rating))
})
```

One can graph these λ to see which value minimizes RMSE:

```
# behavior of lambdas vs. RMSEs

qplot(lambdas, RMSEs, main = "RMSE for different values of lambda", xlab = "Lambdas", ylab = "RMSEs")
```



Specifically, the best λ is:

```
# extraction of optimal lambda

lambda_star <- lambdas[which.min(RMSEs)]

lambda_star
```

```
## [1] 5.25
```

Thus, our final results for all four models are the following:

```
# final storage of results

RMSE_results <- bind_rows(RMSE_results,
                          tibble(Method = "Regularized movie + user effects model",
                                 RMSE = min(RMSEs)))

RMSE_results %>%
  knitr::kable(label = "RMSEs per method")
```

Method	RMSE
Average model	1.0603313
Movie effect model	0.9439087
Movie + user effects model	0.8653488
Regularized movie + user effects model	0.8648170

One can see that the last improvement is marginal, nonetheless, is fair to say that it is robust and adequate.

5 Conclusion

Throughout this document, it has been shown several approaches to predict movie ratings based on a quite large data set. With machine learning algorithms, three models were discarded so the regularized model showed the lowest value of RMSE. This was made through the addition of movie and user effects, given that these variables show certain behavior patterns and tendencies that were modeled to improve the estimations.

Despite this, one can argue that the model could be improved even more by adding other effects like movie genre, year, user age, user sex, etc. Only repeating the procedures an arbitrary number of times would shed light on how much the RMSE can be minimized. Unfortunately, time and software constraints prevent the modeling of more sophisticated estimates.

Nevertheless, this exercise was useful and valid, since 4 repetitions were performed and an acceptable RMSE level was reached.

6 References

HarvardX data science professional certificate. (n.d.). In *edX*. https://www.edx.org/es/professional-certificate/harvardx-data-science?index=spanish_product&queryID=5be4100055746931457d0566e5c3bf26&position=2

Netflix prize. (2021). In *Wikipedia*. Wikimedia Foundation. https://en.wikipedia.org/wiki/Netflix_Prize