MovieLens Project

ongmq

2022-09-06

Section 1: Introduction

This project is part of the final course of the HarvardX Data Science Professional Certificate program. The aim is to create a movie recommendation system using the MovieLens data set, which contains millions of movie ratings by users.

The MovieLens data set will be split into (i) a edx set and (ii) a validation set.

- The edx set will be split further into a training set to train our algorithms, and a testing set to evaluate the algorithms.
- The performance of the algorithms will be determined by their root mean square error (RMSE), with smaller RMSE indicating better performance.
- The best-performing model will then be evaluated against the validation set.

Section 1.1: Generating the edx and validation sets

We start by loading the required R packages, namely the tidyverse, caret, and data.table packages.

```
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)
```

We then download the MovieLens data set using the code provided in the HarvardX course.

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("https://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")

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")
```

We will now generate the `validation` set using the code provided in the HarvardX course.

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

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,]
```

```
# Make sure userId and movieId in `validation` set are also in `edx` set

validation_set <- 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_set)
edx <- rbind(edx, removed)
```

Section 1.2: Summary statistics of the edx set

The following code shows that the `edx` set is a data.table with over 9 million observations and 6 variables.

```
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|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
# We can see that many movies fall into multiple genres.

head(edx$genres)
```

```
## [1] "Comedy|Romance"                "Action|Crime|Thriller"
## [3] "Action|Drama|Sci-Fi|Thriller"  "Action|Adventure|Sci-Fi"
## [5] "Action|Adventure|Drama|Sci-Fi" "Children|Comedy|Fantasy"
```

```
# We can tease out the unique genres using the following code.

genres <- str_replace(edx$genres, "\\|.*", "")
genres <- genres[!duplicated(genres)]

# There are 20 unique genres, including a category of "no genres listed".

genres
```

```
##  [1] "Comedy"             "Action"             "Children"
```

```
##  [4] "Adventure"          "Animation"          "Drama"
##  [7] "Crime"              "Sci-Fi"             "Horror"
## [10] "Thriller"           "Film-Noir"          "Mystery"
## [13] "Western"            "Documentary"        "Romance"
## [16] "Fantasy"            "Musical"            "War"
## [19] "IMAX"               "(no genres listed)"
```

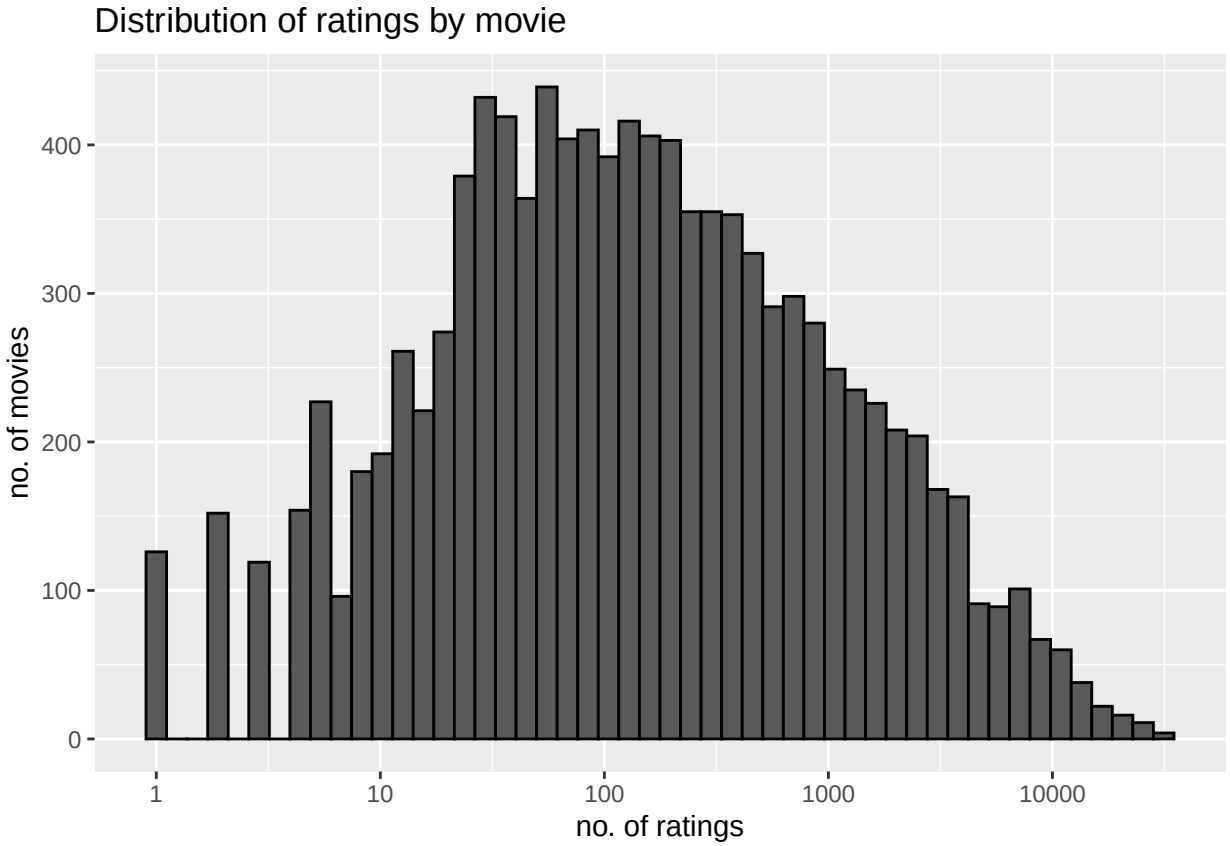
We will now compile some simple summary statistics of the `edx` set before going into data exploration.

```
##   n_ratings n_movies n_users n_genres avg_rating
## 1   9000055    10677   69878       20   3.512465
```

Section 2: Analysis - Exploring the edx set

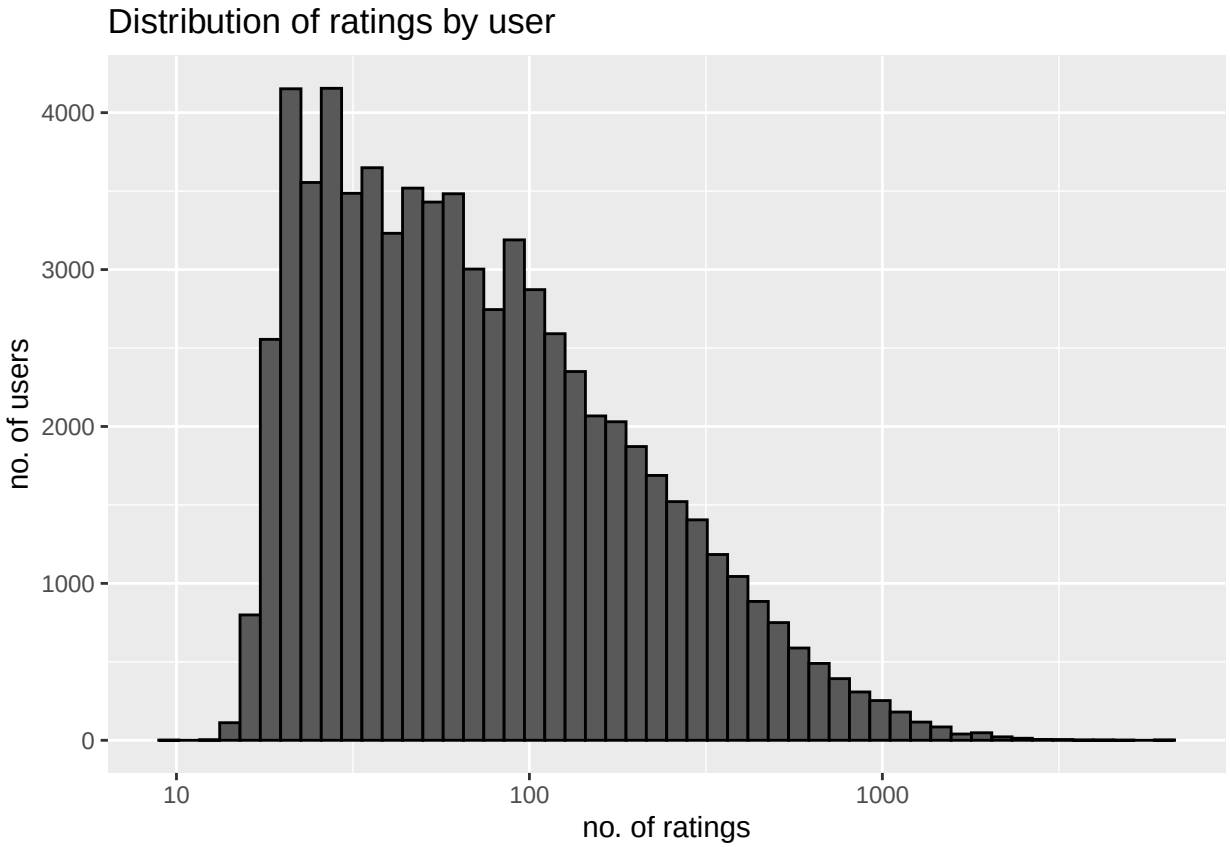
Section 2.1: Identifying movie effect

A quick data exploration shows that some movies are rated more than others.

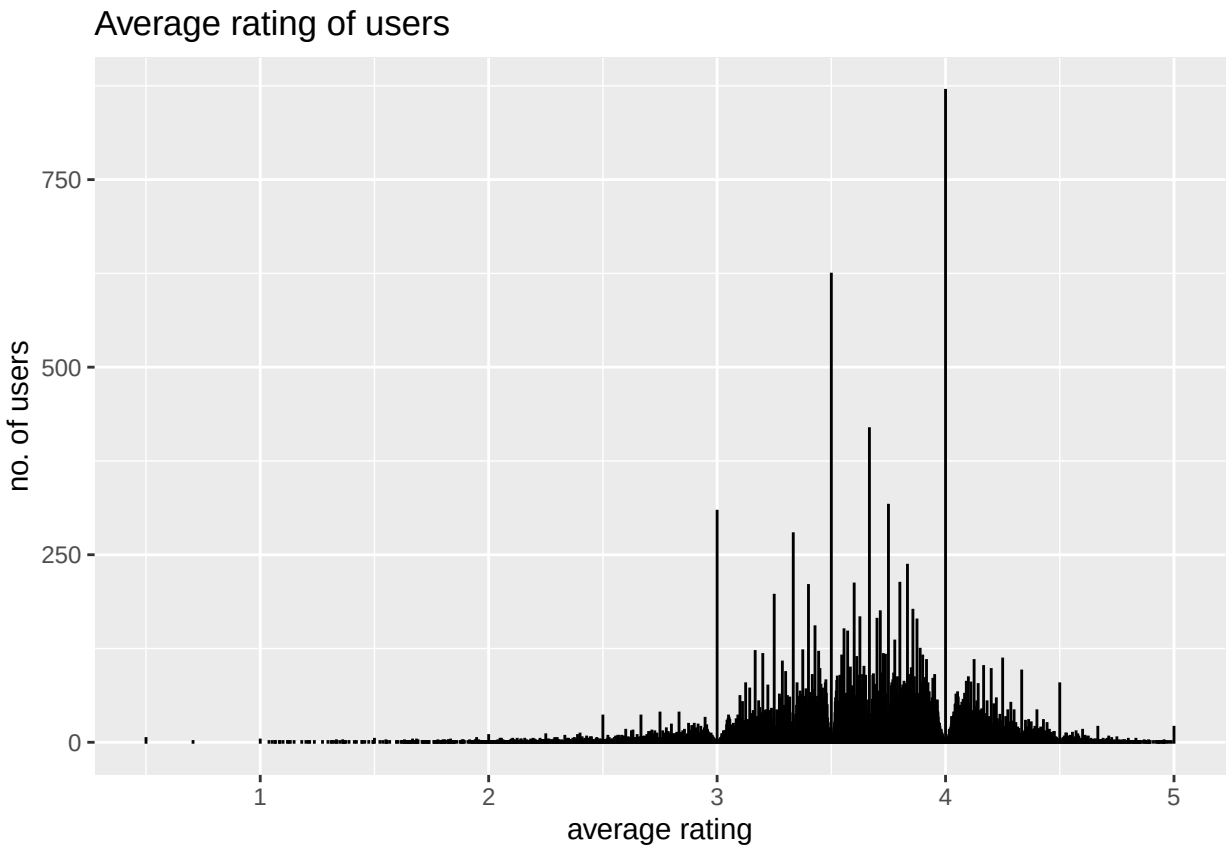


Section 2.2: Identifying user effect

Some users are also more active than others at rating movies.



Different users also have different rating patterns, with some tend to rate movies more favourably in general.



Section 2.3: Identifying genre effect

Ratings are also affected by the movie's genre.

```
# Below is the number of movies for each genre,

n_movies_genres <- sapply(genres, function(i){
  index <- str_which(edx$genres, i)
  length(edx$rating[index])
})
n_movies_genres
```

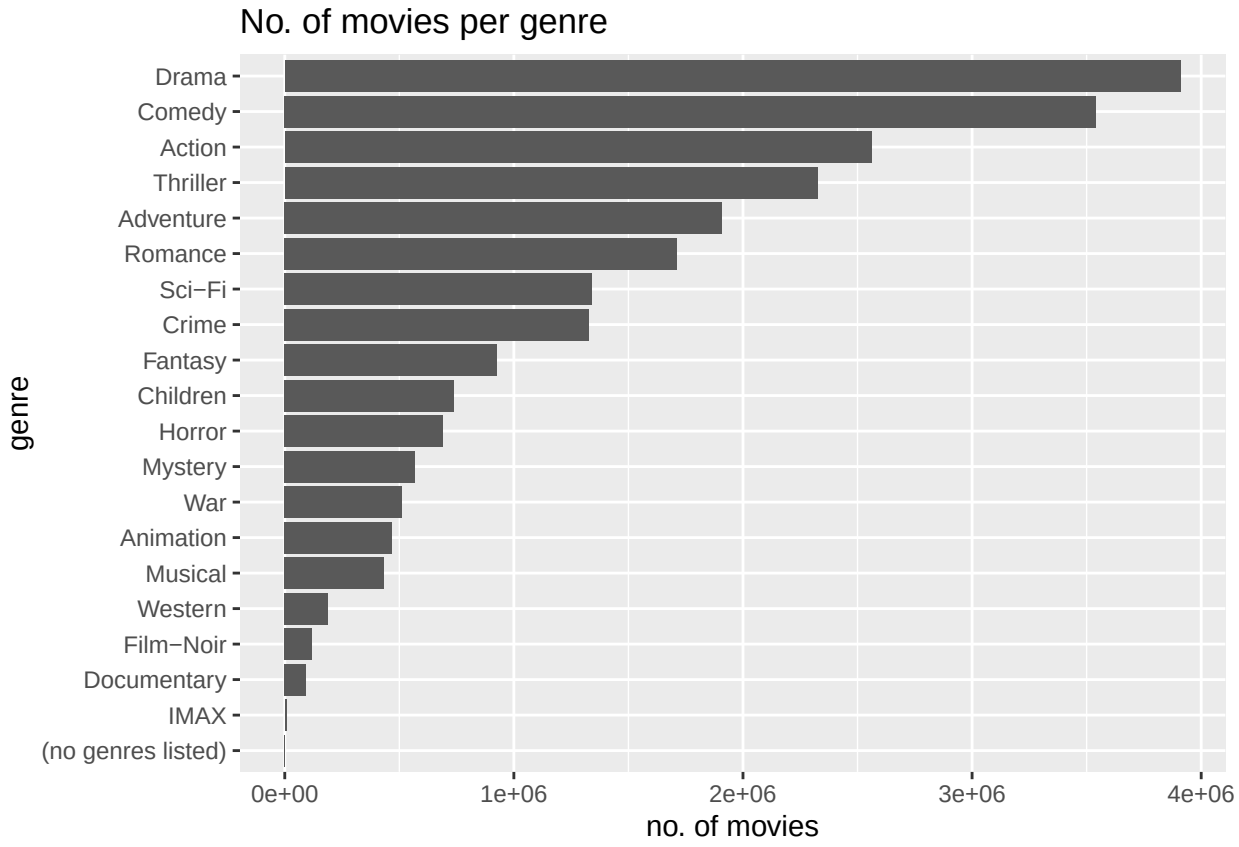
```
##            Comedy            Action          Children         Adventure
##           3540930           2560545            737994           1908892
##         Animation             Drama             Crime             Sci-Fi
##            467168           3910127           1327715           1341183
##            Horror          Thriller         Film-Noir           Mystery
##            691485           2325899            118541            568332
##           Western       Documentary           Romance           Fantasy
##            189394             93066           1712100            925637
##           Musical               War              IMAX (no genres listed)
##            433080            511147              8181                 7
```

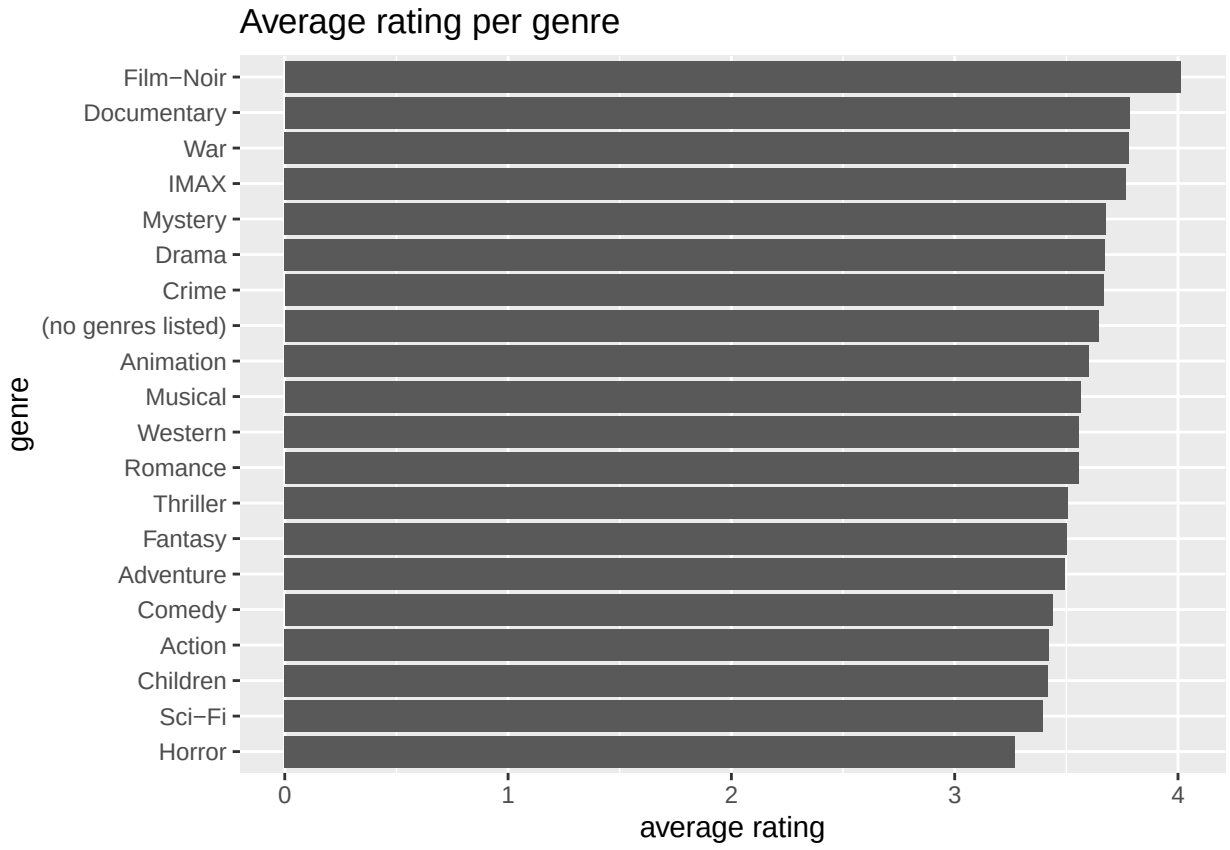
```
# and the average rating for each genre.

genres_avg_rating <- sapply(genres, function(i){
  index <- str_which(edx$genres, i)
```

```
  mean(edx$rating[index])
})
genres_avg_rating
```

```
##            Comedy            Action          Children         Adventure
##          3.436908          3.421405          3.418715          3.493544
##         Animation             Drama             Crime            Sci-Fi
##          3.600644          3.673131          3.665925          3.395743
##            Horror          Thriller         Film-Noir           Mystery
##          3.269815          3.507676          4.011625          3.677001
##           Western       Documentary           Romance           Fantasy
##          3.555918          3.783487          3.553813          3.501946
##           Musical               War  IMAX (no genres listed)
##          3.563305          3.780813          3.767693          3.642857
```

We can see that certain genres tend to be rated more favourably than others.





Section 2.4: Identifying time effect

There could also be time effect on movie ratings.

```
# The timestamp variable in the `edx` set, which represents the time and date in which the rating was p

head(edx$timestamp)
```

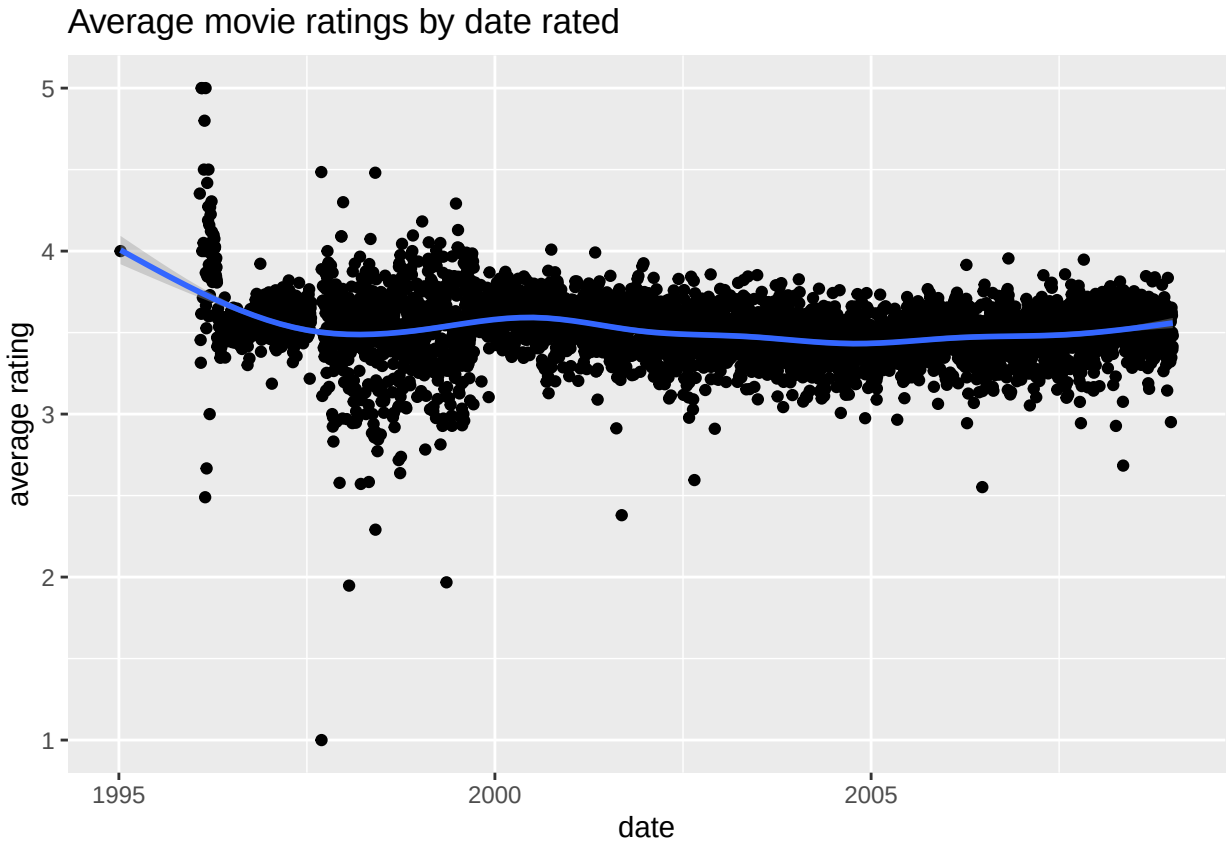
```
## [1] 838985046 838983525 838983421 838983392 838983392 838984474
```

```
# We will change it to the ISO 8601 format, and compute the average rating for each date.

library(lubridate)
edx <- mutate(edx, date = as_datetime(timestamp))
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                date
## 1:                Comedy|Romance 1996-08-02 11:24:06
## 2:          Action|Crime|Thriller 1996-08-02 10:58:45
## 3:  Action|Drama|Sci-Fi|Thriller 1996-08-02 10:57:01
## 4:       Action|Adventure|Sci-Fi 1996-08-02 10:56:32
```

```
## 5: Action|Adventure|Drama|Sci-Fi 1996-08-02 10:56:32
## 6:       Children|Comedy|Fantasy 1996-08-02 11:14:34
```

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



Data exploration tells us that there are (i) movie, (ii) user, (iii) genre, and (iv) time effects affecting ratings. We will train an algorithm building on these effects one by one.

Section 3: Results - Building our models

We start by splitting the `edx` set further into training and testing sets. The `test` set will be 10% of `edx` data.

```
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]
```

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

removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)
```

Section 3.1: Model 1 - predict the same rating for all movies

We start by building the simplest model, i.e., predicting the same rating for all movies.

```
# In this case, the estimate that minimises the RMSE is the average of all ratings.

mu <- mean(train_set$rating)

# If we predict all ratings with the estimate mu, we obtain an RMSE of 1.06, which is too high.

rmse_model_1 <- RMSE(mu, test_set$rating)
```

```
# We will put the RMSE result in a table to see how the RMSE decreases as we improve our model.

rmse_results <- data.frame(method = "just the average",
                           RMSE = rmse_model_1)
rmse_results
```

```
##             method     RMSE
## 1 just the average 1.060054
```

Section 3.2: Model 2 - adding movie effect

Based on our data exploration, we know that different movies are rated differently. We can improve our model by including the movie effect, i.e., adding a term b_i to represent the average rating for movie i.

```
# We can obtain the least squares estimate (LSE) of b_i using the following code.
# Note that we are using a linear model for our algorithm.

mu_movie <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))

# We can see how much our prediction improves using the following code.

preds_movie <- test_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

rmse_model_2 <- RMSE(preds_movie, test_set$rating)
```

```
# The RMSE falls to 0.94, but we can do much better by taking into account of other effects.

rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "adding movie effect",
                                     RMSE = rmse_model_2))
rmse_results
```

```
##                method      RMSE
## 1    just the average 1.0600537
## 2 adding movie effect 0.9429615
```

Section 3.3: Model 3 - adding user effect

Our data exploration also tells us that different users tend to rate movies differently. We can further improve our model by adding the user effect, i.e., including a term b_u to represent the average rating for user u.

```
# The LSE of b_u and the RSME are obtained using the following code.
```

```r
mu_user <- train_set %>%
  left_join(mu_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))

preds_user <- test_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

rmse_model_3 <- RMSE(preds_user, test_set$rating)
```

```r
# The RMSE falls to 0.86, but we can do even better by taking into account of more effects.

rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "adding user effect",
                                     RMSE = rmse_model_3))
rmse_results
```

```
##                method      RMSE
## 1    just the average 1.0600537
## 2 adding movie effect 0.9429615
## 3  adding user effect 0.8646843
```

Section 3.4: Model 4 - adding genre effect

We know that different movie genres also tend to be rated differently. To take into account of the genre effect, we include a term b_g to represent the average rating for genre g.

```r
# The LSE of b_g and the RMSE are obtained using the following code.

mu_genre <- train_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  group_by(genres) %>%
  summarise(b_g = mean(rating - mu - b_i - b_u))

preds_genre <- test_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  left_join(mu_genre, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred

rmse_model_4 <- RMSE(preds_genre, test_set$rating)
```

```r
# The RMSE remains somewhat the same.
# We will check if we can improve our model further by adding the time effect.

rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "adding genre effect",
                                     RMSE = rmse_model_4))
rmse_results
```

```
##                 method      RMSE
## 1    just the average 1.0600537
## 2 adding movie effect 0.9429615
## 3  adding user effect 0.8646843
## 4 adding genre effect 0.8643241
```

Section 3.5: Model 5 - adding time effect

To include the time effect, we add a term b_t to represent the average rating for date rated t.

```
# We first round the ISO 8601 date we created earlier to day,

train_set <- mutate(train_set, date = round_date(date, unit = "day"))
test_set <- mutate(test_set, date = round_date(date, unit = "day"))

# and then we calculate the LSE of b_t and the RMSE using the following code.

mu_time <- train_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  left_join(mu_genre, by = 'genres') %>%
  group_by(date) %>%
  summarise(b_t = mean(rating - mu - b_i - b_u - b_g))

preds_time <- test_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  left_join(mu_genre, by = "genres") %>%
  left_join(mu_time, by = "date") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  .$pred

rmse_model_5 <- RMSE(preds_time, test_set$rating)
```

```
# The RMSE falls slightly further, suggesting an improvement in our model.

rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "adding time effect",
                                     RMSE = rmse_model_5))
rmse_results
```

```
##                 method      RMSE
## 1    just the average 1.0600537
## 2 adding movie effect 0.9429615
## 3  adding user effect 0.8646843
## 4 adding genre effect 0.8643241
## 5  adding time effect 0.8638314
```

Section 3.6: Model 6 - Regularisation

It is likely that some of our estimates are noisy, i.e., skewed by small sample sizes. We will use regularisation to penalise large estimates that are formed using small sample sizes.

We start by using cross-validation to select a lambda (tuning variable) by taking into account movie, user, genre and time effects.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  mu_movie <- train_set %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu) / (n()+l))

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

  mu_genre <- train_set %>%
    left_join(mu_movie, by = 'movieId') %>%
    left_join(mu_user, by = 'userId') %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - mu - b_i - b_u) / (n()+l))

  mu_time <- train_set %>%
    left_join(mu_movie, by = 'movieId') %>%
    left_join(mu_user, by = 'userId') %>%
    left_join(mu_genre, by = 'genres') %>%
    group_by(date) %>%
    summarise(b_t = sum(rating - mu - b_i - b_u - b_g) / (n()+l))

  preds_reg <- test_set %>%
    left_join(mu_movie, by = 'movieId') %>%
    left_join(mu_user, by = 'userId') %>%
    left_join(mu_genre, by = "genres") %>%
    left_join(mu_time, by = "date") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
    .$pred

  return(RMSE(preds_reg, test_set$rating))
})
```

The optimal lambda is as follow:

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

We will now use the optimal lambda to evaluate the RMSE of our regularised model.

```
mu_movie <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu) / (n()+lambda))

mu_user <- train_set %>%
  left_join(mu_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu - b_i) / (n()+lambda))

mu_genre <- train_set %>%
  left_join(mu_movie, by = 'movieId') %>%
```

```
  left_join(mu_user, by = 'userId') %>%
  group_by(genres) %>%
  summarise(b_g = sum(rating - mu - b_i - b_u) / (n()+lambda))

mu_time <- train_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  left_join(mu_genre, by = 'genres') %>%
  group_by(date) %>%
  summarise(b_t = sum(rating - mu - b_i - b_u - b_g) / (n()+lambda))

preds_reg <- test_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  left_join(mu_genre, by = "genres") %>%
  left_join(mu_time, by = "date") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  .$pred

rmse_model_6 <- RMSE(preds_reg, test_set$rating)
```

```
# The RMSE remains somewhat the same.
# Nonetheless, the regularised model is preferred as it takes into account the effect of small sample s

rmse_results <- bind_rows(rmse_results,
                          data.frame(method = "regularisation",
                                     RMSE = rmse_model_6))
rmse_results
```

```
##                method      RMSE
## 1    just the average 1.0600537
## 2 adding movie effect 0.9429615
## 3  adding user effect 0.8646843
## 4 adding genre effect 0.8643241
## 5  adding time effect 0.8638314
## 6      regularisation 0.8632662
```

Section 3.7: Testing final model using the validation set

We will now test our final model–which is the regularised model including the average rating plus the movie, user, genre and time effects–using the validation set.

```
# We start by wrangling the timestamp variable of the `validation` set.

str(validation_set)
```

```
## Classes 'data.table' and '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 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Roman
##  - attr(*, ".internal.selfref")=<externalptr>
```

```r
validation_set <- validation_set %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(date = round_date(date, unit = "day"))
head(validation_set)
```

```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                    title
## 1:                                     Dumb & Dumber (1994)
## 2:                                     Jurassic Park (1993)
## 3:                                        Home Alone (1990)
## 4:                                           Rob Roy (1995)
## 5:                                    Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                      genres       date
## 1:                                   Comedy 1996-08-02
## 2:        Action|Adventure|Sci-Fi|Thriller 1996-08-02
## 3:                          Children|Comedy 1996-08-02
## 4:                Action|Drama|Romance|War 1997-07-07
## 5:                             Crime|Drama 1997-07-07
## 6: Action|Adventure|Horror|Sci-Fi|Thriller 1997-07-07
```

```r
# Next, we use cross-validation to select a lambda.
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){

  mu <- mean(validation_set$rating)

  mu_movie <- validation_set %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu) / (n()+l))

  mu_user <- validation_set %>%
    left_join(mu_movie, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - mu - b_i) / (n()+l))

  mu_genre <- validation_set %>%
    left_join(mu_movie, by = 'movieId') %>%
    left_join(mu_user, by = 'userId') %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - mu - b_i - b_u) / (n()+l))

  mu_time <- validation_set %>%
    left_join(mu_movie, by = 'movieId') %>%
    left_join(mu_user, by = 'userId') %>%
    left_join(mu_genre, by = 'genres') %>%
    group_by(date) %>%
    summarise(b_t = sum(rating - mu - b_i - b_u - b_g) / (n()+l))
```

```
  preds_valid <- validation_set %>%
    left_join(mu_movie, by = 'movieId') %>%
    left_join(mu_user, by = 'userId') %>%
    left_join(mu_genre, by = "genres") %>%
    left_join(mu_time, by = "date") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
    .$pred

  return(RMSE(preds_valid, validation_set$rating))
})
```

```
## [1] 0
```

```
# Finally, we calculate the RMSE using the following code.
mu_movie <- validation_set %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu) / (n()+lambda))

mu_user <- validation_set %>%
  left_join(mu_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu - b_i) / (n()+lambda))

mu_genre <- validation_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  group_by(genres) %>%
  summarise(b_g = sum(rating - mu - b_i - b_u) / (n()+lambda))

mu_time <- validation_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  left_join(mu_genre, by = 'genres') %>%
  group_by(date) %>%
  summarise(b_t = sum(rating - mu - b_i - b_u - b_g) / (n()+lambda))

preds_valid <- validation_set %>%
  left_join(mu_movie, by = 'movieId') %>%
  left_join(mu_user, by = 'userId') %>%
  left_join(mu_genre, by = "genres") %>%
  left_join(mu_time, by = "date") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  .$pred

RMSE(preds_valid, validation_set$rating)
```

```
## [1] 0.8233911
```

We obtained an RMSE of 0.8234, which is smaller than the threshold of 0.8649 required by the HarvardX course.

Section 4: Conclusion

We have successfully trained a machine learning algorithm to predict movie ratings, taking into account movie, user, genre and time effects.

- Our analysis shows that movie and user effects are more important predictors of movie ratings, since they reduce the RMSE by a larger extent.
- Genre and time effects also help to predict movie ratings, though they reduce the RSME by a smaller extent.
- Meanwhile, regularisation helps to improve our model by taking into account the effect of small sample sizes.

Future work can explore the data set and improve the algorithm further (e.g., including the release date of each movie in the model).

Section 5: References

Being new to R and machine learning, I have consulted the following materials when doing up this movielens project:

- https://rafalab.github.io/dsbook/index.html