MovieLens Project

Dr. Ankur Awadhiya, IFS

15 August 2021

Contents

Executive summary

Recommendation systems, or recommender systems are a group of techniques and algorithms that can provide recommendations. They work by analysing the known data to make predictions about the unknown, primarily in the form of recommendations. Recommender systems are widely used by websites including movie streaming services and online market places, to the benefit of both the consumers and the service providers.

In this project, we implement a movie recommendation system using the 10M version of the MovieLens dataset, using tools shown throughout the course. We download and clean this dataset, and partition it into working (train_set and test_set) and validation datasets.

Exploratory data analysis is used to gain insights about the datasets. We observe that the datasets have a distinct and significant movie effect: some movies are much better and some are much worse rated than the average. Similarly, the datasets also show a distinct and significant user effect: some users give much higher ratings and some give much lower ratings than the average. Movie ratings also show a distinct genre effect: some genres have very high average ratings, while movies in some other genres have quite low average ratings.. Full ratings are much more common than half ratings. There is a perceptible, but not very strong, time effect. Movies of different time periods have different average ratings.

Next, we train different machine learning algorithms on the train_set dataset and test them on the test_set dataset. To evaluate the algorithms, we compute the root mean square errors between the predictions and the known values of ratings. The algorithm that provides the lowest RMSE value is then chosen and tested on the

validation dataset.

We find that we get the lowest RMSE values when movie effects, user effects, genre effects and time effects are simultaneously incorporated in a regularised algorithm using penalised least squares. On evaluating the validation dataset, we achieve the RMSE value of 0.8647, showing that machine learning algorithms can be used to get very accurate predictions about movie ratings.

Introduction

The aim of this project is to create a movie recommendation system using the MovieLens dataset. Recommendation systems are a group of techniques and algorithms that can provide recommendations, typically in the form of relevant items to the users, based on user preferences. They process past data provided by users (say, by clicking videos that interest them on Youtube) to make predictions about future behaviours and choices.

Recommender systems are of two kinds - content-based, and collaborative-filtering-based. The content-based recommender systems make suggestions based on the similarity of movie attributes. For example, if I watch action movies starring Arnold Schwarzenegger, the recommender will suggest other action movies, movies starring Arnold Schwarzenegger, or action movies starring Arnold Schwarzenegger. On the other hand, collaborative-filtering-based systems analyse past interactions between users and movies to make suggestions. For instance, if I watch movies similar to individuals A and B, but very different from movies watched by individuals C and D, the recommender system will suggest movies that are watched by A and B, but not by C or D. In this project, we will implement a collaborative-filtering-based recommender system, which predicts the ratings that a given user would give to a movie, thus recommending to users those movies that they would have rated highly.

While movie recommendation system has been taught in the course, it was based on a small, abridged dataset. This project aims to construct a movie recommendation system using the 10M version of the MovieLens dataset, using tools shown throughout the course.

Goals

The objective of this project is to train machine learning algorithms on the 10M version of the MovieLens dataset, finally using the best algorithm to predict movie ratings in the validation dataset.

We have been provided codes to generate datasets. We need to develop our algorithm using the edx set. For a final test of our final algorithm, we will predict movie ratings in the validation set (the final hold-out test set) as if they were unknown. RMSE will be used to evaluate how close our predictions are to the true values in the validation set (the final hold-out test set).

Dataset

The dataset is to be downloaded from MovieLens, a research site of the University of Minnessota's GroupLens Social Computing Research. This research site uses collaborative filtering technology to recommend movies that users would enjoy, based on the kinds of movie ratings that they have provided on the site. The algorithm predicts those movies that would have received very low and very high ratings by the user, as discerned by the user's preferences and prior data. Once this is done, the user may avoid those predicted movies that the system flags as not as per the user's preferences, and may preferentially watch those movies that have been marked by the algorithm as matching the user's preferences. Thus, the site generates personalised predictions about movies that the user has not yet watched.

The data that have been generated over the years can be utilised by researchers to analyse several aspects of filtering and personalisation. In the current project as well, we make use of this MovieLens data. The dataset can be downloaded from the following sites:

1. https://grouplens.org/datasets/movielens/10m/
2. http://files.grouplens.org/datasets/movielens/ml-10m.zip

The dataset is in the form of comma-separated values.

Variables

The important variables in the dataset are:

1. userId: Anonymised ids that are consistent between ratings.csv and tags.csv files.
2. movieId: Unique ids for movies. Only movies with at least one rating or tag are included in the dataset. Movie ids are consistent between ratings.csv, tags.csv, movies.csv, and links.csv files.
3. rating: Ratings are made on a 5-star scale, with half-star increments.
4. timestamp: Timestamps are represented in seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
5. tag: Tags are user-generated metadata about movies, typically a single word or a short phrase.
6. title: The title of the movie.
7. genres: Genres of the movies are a pipe-separated list selected from the following:

a. Action

b. Adventure

c. Animation

d. Children’s

e. Comedy

f. Crime

g. Documentary

h. Drama

i. Fantasy

j. Film-Noir

k. Horror

l. Musical

m. Mystery

n. Romance

o. Sci-Fi

p. Thriller

q. War

r. Western

s. (no genres listed)

8. imdbId: This is an identifier for movies, as used by http://www.imdb.com.

9. tmdbId: This is an identifier for movies, as used by https://www.themoviedb.org.

Key steps

The key steps are the following:

1. Installation of required packages and loading of libraries

2. Downloading the dataset

3. Cleaning the dataset to construct a dataset with requisite attributes

4. Partitioning of dataset into working (training and testing) and validation datasets

5. Exploratory data analysis to understand the datasets and get insights about their salient attributes, to be used to train machine learning algorithms

6. Training different algorithms on the training dataset, and testing them on the testing dataset

7. Evaluation of different algorithms through computation of root mean square error (or some other suitable parameter) between the predictions and the actual ratings in the testing dataset

8. Selection of the best (or set of best) algorithm(s) on the basis of evaluation

9. Utilisation of the best (or set of best) algorithm(s) selected on the basis of evaluation to compute predictions for the validation dataset

10. Reporting of results and compilation of report

Methods and analysis

Basic options

Before we begin our analysis, we clear the existing variables in the workspace using the rm(list=ls()) command. We also set the number of decimal places for results using the options(digits=4) command.

```
### Basic options: clear list, show 4 decimal places
rm(list=ls())
options(digits=4)
```

Installing requisite packages

When submitting a code to a third party, it is a good practice to incorporate commands that install the requisite packages automatically, so that the code runs smoothly on the target machine. We do this using the if(!require(PACKAGE_NAME)) install.packages("PACKAGE_NAME", repos = "REPO_NAME") command. This command checks if the requisite packages are installed on the target machine, and if not, installs them. We make a list of requisite packages and incorporate the installation commands.

```
### Installing requisite 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")
```

```r
if(!require(data.table)) install.packages("data.table",
    repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate",
    repos = "http://cran.us.r-project.org")
```

Loading requisite libraries

Next, we load the requisite package libraries using the library(LIB_NAME) command. Once the libraries are loaded, we can make use of them in the subsequent code.

```r
### Loading required libraries
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
```

Dataset download

It is a good practice to incorporate the commands for dataset download in the script itself, so that the script can run autonomously. The required dataset can be downloaded from the following sites:

1. https://grouplens.org/datasets/movielens/10m/
2. http://files.grouplens.org/datasets/movielens/ml-10m.zip

The dataset is in the form of comma-separated values, and we can load the dataset using the read.csv() or fread() command.

```r
### Downloading the 10M dataset
dl <- tempfile()
file <- "http://files.grouplens.org/datasets/movielens/ml-10m.zip"
download.file(file, dl)
```

Data cleaning

The process of data cleaning creates a cleaned dataset with the requisite attributes as are required for processing.

We begin by unzipping the dataset using the unzip() command. The contents of the files are read using the fread() command. The fread() command is similar to the read.table() command, but is faster and more convenient. It reads the input data using the readLines() command, and we can specify attributes such as the separator used between columns, the number of rows to be read, the header column names, whether strings should be converted to factors, etc.

Next we make use of the str_split_fixed() command to get movies data. This command separates strings into fixed number of pieces, as defined by a regular expression. We add the column names using the colnames() command.

Next we convert the movies data into a dataframe using the as.data.frame() command, convert movieIds into numbers, and titles and genres into characters, and then join the movies and ratings data into the movielens data using the left_join() command, keeping movieId as the joining attribute.

This completes the construction of a composite file that can be used in further analyses. By combining the movies and ratings data together and having the column headers, we have a clean file. If required, we may also remove the NA values in this step.

```r
### Unzipping the dataset and construction of a composite file
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")
```

```r
# Construction of a composite dataset
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")
```

Next, we use createDataPartition() function from the caret package to partition the movielens dataset into working (called edx) and validation datasets. We set the seed to 1 to ensure replicability of the submitted results. We partition the dataset in 90:10

manner, with temp validation dataset being 10% and working dataset being 90%. Since we have a large number of data points, a 9:1 ratio ensures that we have sufficient amount of data in validation dataset for final validation, and also large amount of data in the edx dataset to train and test various machine learning algorithms.

The createDataPartition() function outputs 10% indices randomly; these are stored in the test_index variable. movielens[test_index,] outputs the rows with the selected indices; these are stored as the temp dataset. movielens[-test_index,] outputs the rows without the selected indices; these are stored as the edx dataset.

We make sure that userId and movieId in validation dataset are also in edx set. To do this, we use semi_join() command on the temp dataset to get a cleaned validation dataset. We then add rows removed from the validation dataset back into the edx dataset using the anti_join() and rbind() commands.

```r
### Construction of edx and validation datasets
# Validation dataset will be 10% of MovieLens data
set.seed(1) # if using R 4.0 or later,
# use `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 the validation dataset are
# also in the edx dataset
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation dataset back into edx dataset
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

We use the createDataPartition() function from the caret package to partition the edx dataset into test_set and train_set datasets. We set the seed to 1 to ensure replicability of the submitted results. We partition the dataset in a 90:10 manner, with the test_set dataset being 10% and the train_set dataset being 90%. Since we have a large number of data points, a 9:1 ratio ensures that we have sufficient amount of data in test_set dataset to test various machine learning algorithms, and also large amount of data in the train_set dataset to train various machine learning algorithms.

The createDataPartition() function outputs 10% indices randomly; these are stored in the test_index variable. edx[test_index,] outputs the rows with the selected indices; these are stored as the test_set dataset. edx[-test_index,] outputs the rows without the selected indices; these are stored as the train_set dataset.

Finally, we remove the temporary files no longer required: dl, ratings, movies, test_index, temp, removed.

```
# Test set will be 10% of edx data
set.seed(1) # if using R 4.0 or later,
# use `set.seed(1, sample.kind="Rounding")`
test_index <- createDataPartition(y = edx$rating,
    times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

### Removing temporary files
rm(dl, ratings, movies, test_index, temp, removed)
```

This completes the data cleaning stage of the analysis.

Data exploration and visualisation

To make sense of the constructed datasets, we look at their characteristics. We begin by observing the classes of the edx, train_set, test_set, and validation datasets using the class() command.

```
### Exploratory data analysis
# Class of edx, train_set, test_set and validation
class(edx)
```

```
## [1] "data.table" "data.frame"
```

```
class(train_set)
```

```
## [1] "data.table" "data.frame"
```

```r
class(test_set)
```

```
## [1] "data.table" "data.frame"
```

```r
class(validation)
```

```
## [1] "data.table" "data.frame"
```

We observe that the classes of the edx, train_set, test_set, and validation datasets are "data.table" "data.frame" A data.table is an extension of data.frame, and is compatible with R functions working on dataframes. However, it has the added advantage of being better with large datasets, and offering fast subset, grouping, update, and joins.

Next, we explore the dimensions of the edx, train_set, test_set, and validation datasets using the dim() command.

```r
# Dimensions of edx, train_set, test_set and validation datasets
# Note: the values differ on my Macbook Pro and Chromebook.
dim(edx)
```

```
## [1] 9000061       6
```

```r
dim(train_set)
```

```
## [1] 8100054       6
```

```
dim(test_set)
```

```
## [1] 900007      6
```

```
dim(validation)
```

```
## [1] 999993      6
```

We observe that the edx dataset has 9000061 (around 9 million) rows and 6 columns, and the validation dataset has 999993 (around 1 million) rows and 6 columns. This shows that the movielens dataset was correctly partitioned into edx and validation datasets in a 90:10 ratio, as we had desired.

Similarly, we observe that the train_set dataset has 8100054 (around 8.1 million) rows and 6 columns, and the test_set dataset has 900007 (around 0.9 million) rows and 6 columns. This shows that the edx dataset was correctly partitioned into train_set and test_set datasets in a 90:10 ratio, as we had desired.

Next, we explore the structure of the edx, train_set, test_set, and validation datasets using the str() command. To save space, we list the first 3 variables using the list.len=3 argument.

```
# Structure of edx, train_set, test_set and validation datasets
# Argument list.len=3 lists first 3 variables only
str(edx, list.len=3)
```

```
## Classes 'data.table' and 'data.frame':   9000061 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##   [list output truncated]
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
str(train_set, list.len=3)
```

```
## Classes 'data.table' and 'data.frame':   8100054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##   [list output truncated]
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
str(test_set, list.len=3)
```

```
## Classes 'data.table' and 'data.frame':   900007 obs. of  6 variables:
##  $ userId   : int  1 3 3 3 3 4 4 4 5 5 ...
##  $ movieId  : num  589 590 1552 1564 5505 ...
##  $ rating   : num  5 3.5 2 4.5 2 3 3 5 3 1 ...
##   [list output truncated]
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
str(validation, list.len=3)
```

```
## Classes 'data.table' and 'data.frame':   999993 obs. of  6 variables:
##  $ userId   : int  1 2 2 3 3 3 4 4 4 5 ...
##  $ movieId  : num  588 1210 1544 151 1288 ...
##  $ rating   : num  5 4 3 4.5 3 3 3 3 5 3 ...
##   [list output truncated]
##  - attr(*, ".internal.selfref")=<externalptr>
```

We observe that all the four datasets have 6 variables. The userId variable is an integer, the movieId variable is numeric, the rating variable is numeric, and so on. Thus, the variables are in their correct classes for further analyses.

Next, we explore the names in the edx, train_set, test_set, and validation datasets using the names() command.

```
# Names in edx, train_set, test_set and validation datasets
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
names(train_set)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
names(test_set)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
names(validation)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

We observe that all four datasets have the same 6 variable names: “userId”, “movieId”, “rating”, “timestamp”, “title”, and “genres”.

Next, we explore the first 6 rows in the edx, train_set, test_set, and validation datasets using the head() command.

```
# First 6 rows in edx, train_set, test_set and validation datasets
edx %>% head()
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:                        Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

```
train_set %>% head()
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                            genres
## 1:                 Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:                         Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:        Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

```
test_set %>% head()
```

```
##    userId movieId rating  timestamp                                  title
## 1:      1     589    5.0  838983778      Terminator 2: Judgment Day (1991)
## 2:      3     590    3.5 1136075494              Dances with Wolves (1990)
## 3:      3    1552    2.0 1133571139                         Con Air (1997)
## 4:      3    1564    4.5 1136418605 For Roseanna (Roseanna's Grave) (1997)
## 5:      3    5505    2.0 1136075848                  Good Girl, The (2002)
## 6:      4      21    3.0  844416980                     Get Shorty (1995)
##                        genres
## 1:              Action|Sci-Fi
## 2:   Adventure|Drama|Western
## 3: Action|Adventure|Thriller
## 4:      Comedy|Drama|Romance
## 5:              Comedy|Drama
## 6:       Action|Comedy|Drama
```

```
validation %>% head()
```

```
##    userId movieId rating  timestamp
## 1:      1     588    5.0  838983339
## 2:      2    1210    4.0  868245644
## 3:      2    1544    3.0  868245920
## 4:      3     151    4.5 1133571026
## 5:      3    1288    3.0 1133571035
## 6:      3    5299    3.0 1164885617
##                                                      title
## 1:                                          Aladdin (1992)
## 2:       Star Wars: Episode VI - Return of the Jedi (1983)
## 3: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
## 4:                                          Rob Roy (1995)
## 5:                               This Is Spinal Tap (1984)
## 6:                         My Big Fat Greek Wedding (2002)
##                                        genres
## 1: Adventure|Animation|Children|Comedy|Musical
## 2:                    Action|Adventure|Sci-Fi
## 3:     Action|Adventure|Horror|Sci-Fi|Thriller
## 4:                   Action|Drama|Romance|War
## 5:                                   Comedy|Musical
## 6:                                   Comedy|Romance
```

We observe that the datasets appear fine. Next, we summarise the edx, train_set, test_set, and validation datasets using the summary() command.

```
# Summary of edx, train_set, test_set and validation datasets
summary(edx)
```

```
##      userId          movieId           rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.50   Min.   :7.90e+08
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.00   1st Qu.:9.47e+08
##  Median :35743   Median : 1834   Median :4.00   Median :1.04e+09
##  Mean   :35869   Mean   : 4120   Mean   :3.51   Mean   :1.03e+09
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.00   3rd Qu.:1.13e+09
##  Max.   :71567   Max.   :65133   Max.   :5.00   Max.   :1.23e+09
##     title               genres
##  Length:9000061     Length:9000061
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
summary(train_set)
```

```
##      userId          movieId           rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.50   Min.   :7.90e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.00   1st Qu.:9.47e+08
##  Median :35754   Median : 1834   Median :4.00   Median :1.04e+09
##  Mean   :35871   Mean   : 4121   Mean   :3.51   Mean   :1.03e+09
##  3rd Qu.:53605   3rd Qu.: 3626   3rd Qu.:4.00   3rd Qu.:1.13e+09
##  Max.   :71567   Max.   :65133   Max.   :5.00   Max.   :1.23e+09
##     title               genres
##  Length:8100054     Length:8100054
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
summary(test_set)
```

```
##      userId         movieId          rating       timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.50   Min.   :8.23e+08
##  1st Qu.:18090   1st Qu.:  648   1st Qu.:3.00   1st Qu.:9.47e+08
##  Median :35693   Median : 1833   Median :4.00   Median :1.04e+09
##  Mean   :35853   Mean   : 4115   Mean   :3.51   Mean   :1.03e+09
##  3rd Qu.:53576   3rd Qu.: 3623   3rd Qu.:4.00   3rd Qu.:1.13e+09
##  Max.   :71567   Max.   :65088   Max.   :5.00   Max.   :1.23e+09
##     title               genres
##  Length:900007      Length:900007
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
summary(validation)
```

```
##      userId         movieId          rating       timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.50   Min.   :7.90e+08
##  1st Qu.:18127   1st Qu.:  653   1st Qu.:3.00   1st Qu.:9.47e+08
##  Median :35719   Median : 1835   Median :4.00   Median :1.04e+09
##  Mean   :35878   Mean   : 4121   Mean   :3.51   Mean   :1.03e+09
##  3rd Qu.:53649   3rd Qu.: 3633   3rd Qu.:4.00   3rd Qu.:1.13e+09
##  Max.   :71567   Max.   :65133   Max.   :5.00   Max.   :1.23e+09
##     title              genres
##  Length:999993      Length:999993
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

The summary() command gives the minimum, maximum, mean, median, first quartile, and third quartile values for numeric data columns, and length, class, and mode for character data columns.

We can get the number of unique movies in the edx, train_set, test_set, and validation datasets using the n_distinct() command.

```
# Number of unique movies in edx, train_set, test_set and validation
# datasets
n_distinct(edx$movieId)
```

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

```
n_distinct(train_set$movieId)
```

```
## [1] 10665
```

```
n_distinct(test_set$movieId)
```

```
## [1] 9758
```

```
n_distinct(validation$movieId)
```

```
## [1] 9796
```

We observe that the number of movies in every dataset is close to 10,000, even though the datasets have vastly different number of rows. This is expected since a single movie may be rated several times by different users.

Similarly, we can get the number of unique users in the edx, train_set, test_set, and validation datasets using the n_distinct() command.

```r
# Number of unique users in edx, train_set, test_set and validation
# datasets
n_distinct(edx$userId)
```

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

```r
n_distinct(train_set$userId)
```

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

```r
n_distinct(test_set$userId)
```

```
## [1] 68096
```

```r
n_distinct(validation$userId)
```

```
## [1] 68531
```

Here again, we observe that the number of distinct users in every dataset is close to 69,000, even though the datasets have vastly different number of rows. This is expected since a single user may rate several movies.

We can get the number of movies in different genres in the edx, train_set, test_set, and validation datasets by grouping the data by genres and using the summarise() command. We arrange the number of movies in different genres in descending order to get an idea of the genres with the most number of movies.

```r
# Classification of movies by genres in edx, train_set, test_set and
# validation datasets
edx %>% group_by(genres) %>% summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 797 x 2
##    genres                          n
##    <chr>                       <int>
##  1 Drama                      733353
##  2 Comedy                     700883
##  3 Comedy|Romance             365894
##  4 Comedy|Drama               323518
##  5 Comedy|Drama|Romance       261098
##  6 Drama|Romance              259735
##  7 Action|Adventure|Sci-Fi    220363
##  8 Action|Adventure|Thriller 148933
##  9 Drama|Thriller             145359
## 10 Crime|Drama                137424
## # ... with 787 more rows
```

```r
train_set %>% group_by(genres) %>% summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 797 x 2
##    genres                          n
##    <chr>                       <int>
##  1 Drama                       659889
##  2 Comedy                      630957
##  3 Comedy|Romance              329379
##  4 Comedy|Drama                291356
##  5 Comedy|Drama|Romance        234869
##  6 Drama|Romance               233643
##  7 Action|Adventure|Sci-Fi     198231
##  8 Action|Adventure|Thriller 133879
##  9 Drama|Thriller              130770
## 10 Crime|Drama                 123613
## # ... with 787 more rows
```

```
test_set %>% group_by(genres) %>% summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 770 x 2
##    genres                        n
##    <chr>                     <int>
##  1 Drama                     73464
##  2 Comedy                    69926
##  3 Comedy|Romance            36515
##  4 Comedy|Drama              32162
##  5 Comedy|Drama|Romance      26229
##  6 Drama|Romance             26092
##  7 Action|Adventure|Sci-Fi   22132
##  8 Action|Adventure|Thriller 15054
##  9 Drama|Thriller            14589
## 10 Crime|Drama               13811
## # ... with 760 more rows
```

```
validation %>% group_by(genres) %>% summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 769 x 2
##    genres                        n
##    <chr>                     <int>
##  1 Drama                     81731
##  2 Comedy                    77713
##  3 Comedy|Romance            40167
##  4 Comedy|Drama              35976
##  5 Comedy|Drama|Romance      29133
##  6 Drama|Romance             28804
##  7 Action|Adventure|Sci-Fi   24223
##  8 Action|Adventure|Thriller 16738
##  9 Drama|Thriller            16250
## 10 Crime|Drama               15403
## # ... with 759 more rows
```

We observe that in all the four datasets, the largest number of movies are in the Drama genre, followed by Comedy and Comedy|Romance genres. This gives us an indication that all the four datasets are fairly representative of each other.

Next, we explore the movies with greatest number of ratings in edx, train_set, test_set, and validation datasets. We do this by grouping the datasets by title, counting the number of number of ratings for each title, and then arranging the results in descending order to get the titles with the highest number of ratings on the top.

```
# Movies with greatest number of ratings in edx, train_set, test_set
# and validation datasets
edx %>% group_by(title) %>% summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 10,676 x 2
##    title                                                            n
##    <chr>                                                        <int>
##  1 Pulp Fiction (1994)                                          31336
##  2 Forrest Gump (1994)                                          31076
##  3 Silence of the Lambs, The (1991)                             30280
##  4 Jurassic Park (1993)                                         29291
##  5 Shawshank Redemption, The (1994)                             27988
##  6 Braveheart (1995)                                            26258
##  7 Terminator 2: Judgment Day (1991)                            26115
##  8 Fugitive, The (1993)                                         26050
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25809
## 10 Batman (1989)                                                24343
## # ... with 10,666 more rows
```

```
train_set %>% group_by(title) %>% summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 10,664 x 2
##    title                                                         n
##    <chr>                                                     <int>
##  1 Pulp Fiction (1994)                                       28233
##  2 Forrest Gump (1994)                                       27934
##  3 Silence of the Lambs, The (1991)                          27272
##  4 Jurassic Park (1993)                                      26351
##  5 Shawshank Redemption, The (1994)                          25120
##  6 Terminator 2: Judgment Day (1991)                         23554
##  7 Braveheart (1995)                                         23534
##  8 Fugitive, The (1993)                                      23443
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 23203
## 10 Batman (1989)                                             21930
## # ... with 10,654 more rows
```

```
test_set %>% group_by(title) %>% summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 9,757 x 2
##    title                                                         n
##    <chr>                                                     <int>
##  1 Forrest Gump (1994)                                        3142
##  2 Pulp Fiction (1994)                                        3103
##  3 Silence of the Lambs, The (1991)                               3008
##  4 Jurassic Park (1993)                                           2940
##  5 Shawshank Redemption, The (1994)                               2868
##  6 Braveheart (1995)                                              2724
##  7 Fugitive, The (1993)                                           2607
##  8 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  2606
##  9 Terminator 2: Judgment Day (1991)                              2561
## 10 Apollo 13 (1995)                                               2424
## # ... with 9,747 more rows
```

```
validation %>% group_by(title) %>% summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 9,795 x 2
##    title                                                              n
##    <chr>                                                          <int>
##  1 Pulp Fiction (1994)                                             3528
##  2 Silence of the Lambs, The (1991)                                3388
##  3 Forrest Gump (1994)                                             3381
##  4 Jurassic Park (1993)                                            3340
##  5 Shawshank Redemption, The (1994)                                3138
##  6 Fugitive, The (1993)                                            2901
##  7 Braveheart (1995)                                               2896
##  8 Terminator 2: Judgment Day (1991)                               2833
##  9 Apollo 13 (1995)                                                2758
## 10 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  2757
## # ... with 9,785 more rows
```

We observe that in all the four datasets, the top 10 most rated movies are similar. This gives us an indication that all the four datasets are fairly representative of each other.

Next, we explore the number of entries with each rating in edx, train_set, test_set, and validation datasets. We do this by grouping the datasets by rating, counting the number of number of entries for each rating, and then arranging the results in descending order of rating.

```
# Number of movies with given ratings in edx, train_set, test_set
# and validation datasets
edx %>% group_by(rating) %>% summarise(n = n()) %>%
  arrange(desc(rating))
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    5   1390541
##  2    4.5  526309
##  3    4   2588021
##  4    3.5  792037
##  5    3   2121638
##  6    2.5  332783
##  7    2    710998
##  8    1.5  106379
## 9      1    345935
## 10     0.5   85420
```

```
train_set %>% group_by(rating) %>% summarise(n = n()) %>%
  arrange(desc(rating))
```

```
## # A tibble: 10 x 2
##    rating       n
##     <dbl>   <int>
##  1    5   1251507
##  2    4.5  473658
##  3    4   2329171
##  4    3.5  712881
##  5    3   1908878
##  6    2.5  299643
##  7    2    640198
##  8    1.5   95636
##  9    1    311337
## 10    0.5   77145
```

```
test_set %>% group_by(rating) %>% summarise(n = n()) %>%
  arrange(desc(rating))
```

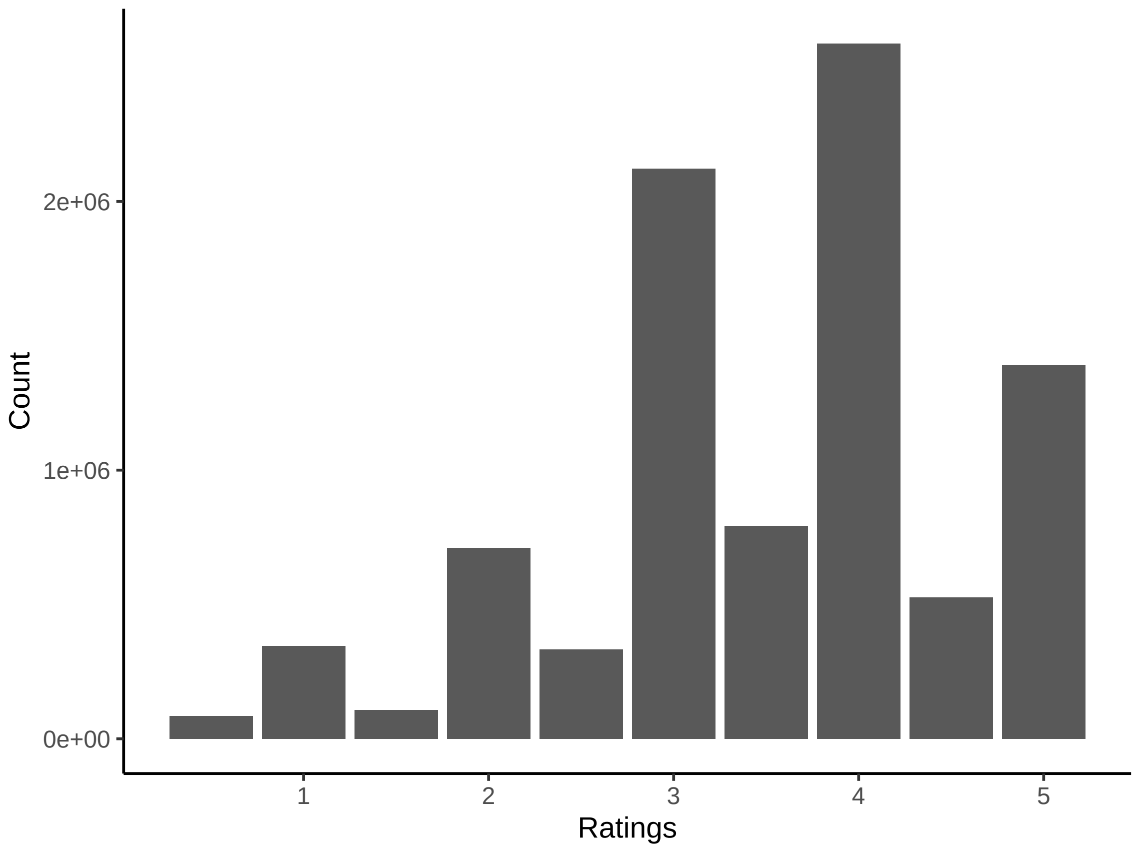
```
## # A tibble: 10 x 2
##    rating      n
##     <dbl>  <int>
## 1     5   139034
## 2     4.5  52651
## 3     4   258850
## 4     3.5  79156
## 5     3   212760
## 6     2.5  33140
## 7     2    70800
## 8     1.5  10743
## 9     1    34598
## 10    0.5   8275
```

```
validation %>% group_by(rating) %>% summarise(n = n()) %>%
  arrange(desc(rating))
```

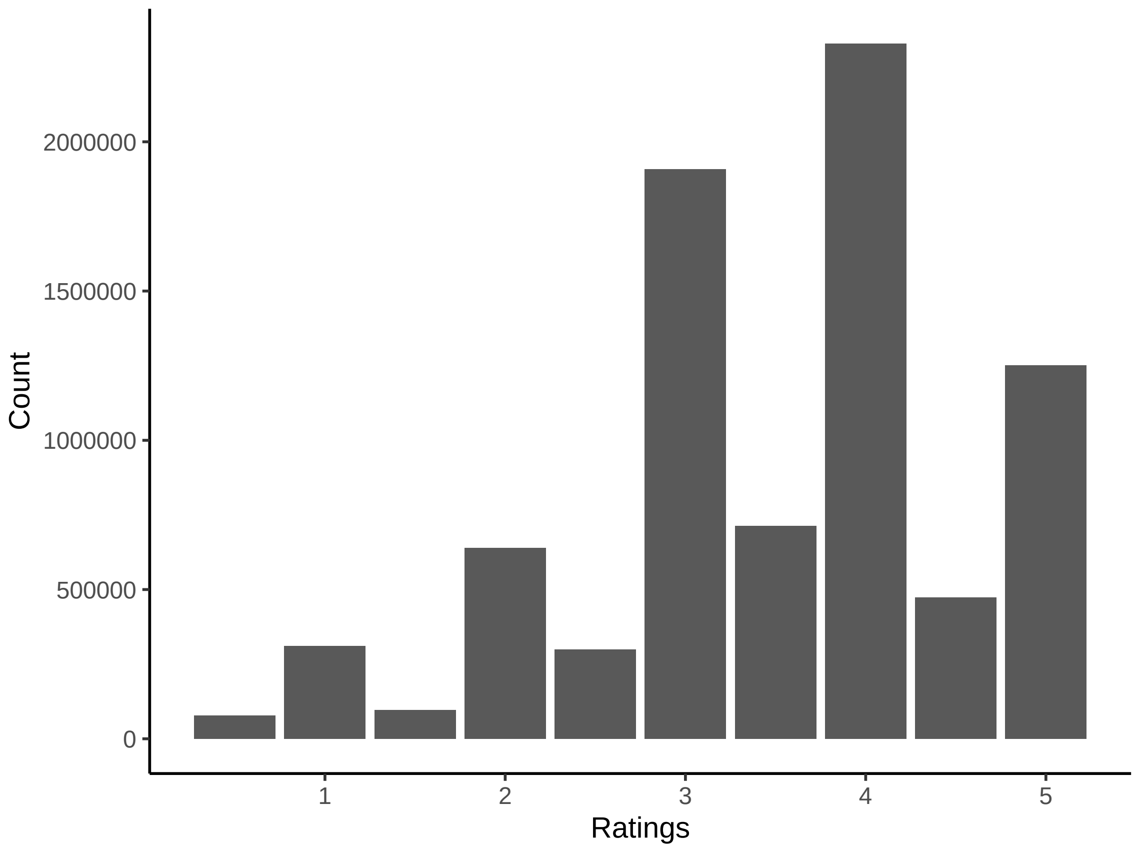
```
## # A tibble: 10 x 2
##    rating      n
##     <dbl>  <int>
## 1     5   154271
## 2     4.5  58713
## 3     4   287829
## 4     3.5  87727
## 5     3   235038
## 6     2.5  37395
## 7     2    79308
## 8     1.5  11899
## 9     1    38245
## 10    0.5   9568
```

We observe the number of movies of different ratings in all the datasets. We can note that the ratings go from 0.5 to 5 in 0.5 increments. Further, the number of entries for each rating is different in each dataset. This is expected since each dataset has a different size. However, the pattern of the ratings appears similar, in that half ratings are much less prevalent than full ratings. To observe this relationship more clearly, we plot the ratings using the ggplot2 library.

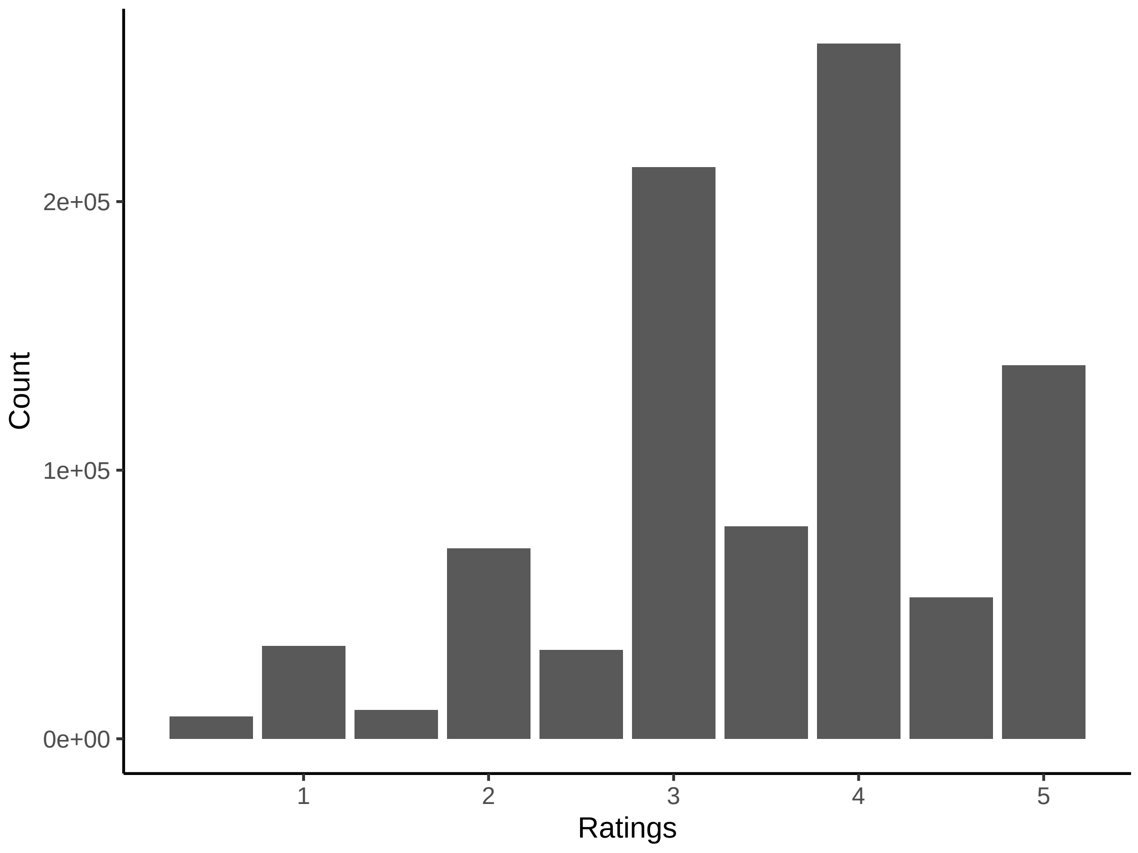
```
# Plot of distribution of ratings
edx %>% group_by(rating) %>% summarize(count = n()) %>%
  ggplot(aes(x=rating, y=count)) +
  geom_col() +
  xlab("Ratings") +
  ylab("Count") +
  theme_classic()
```



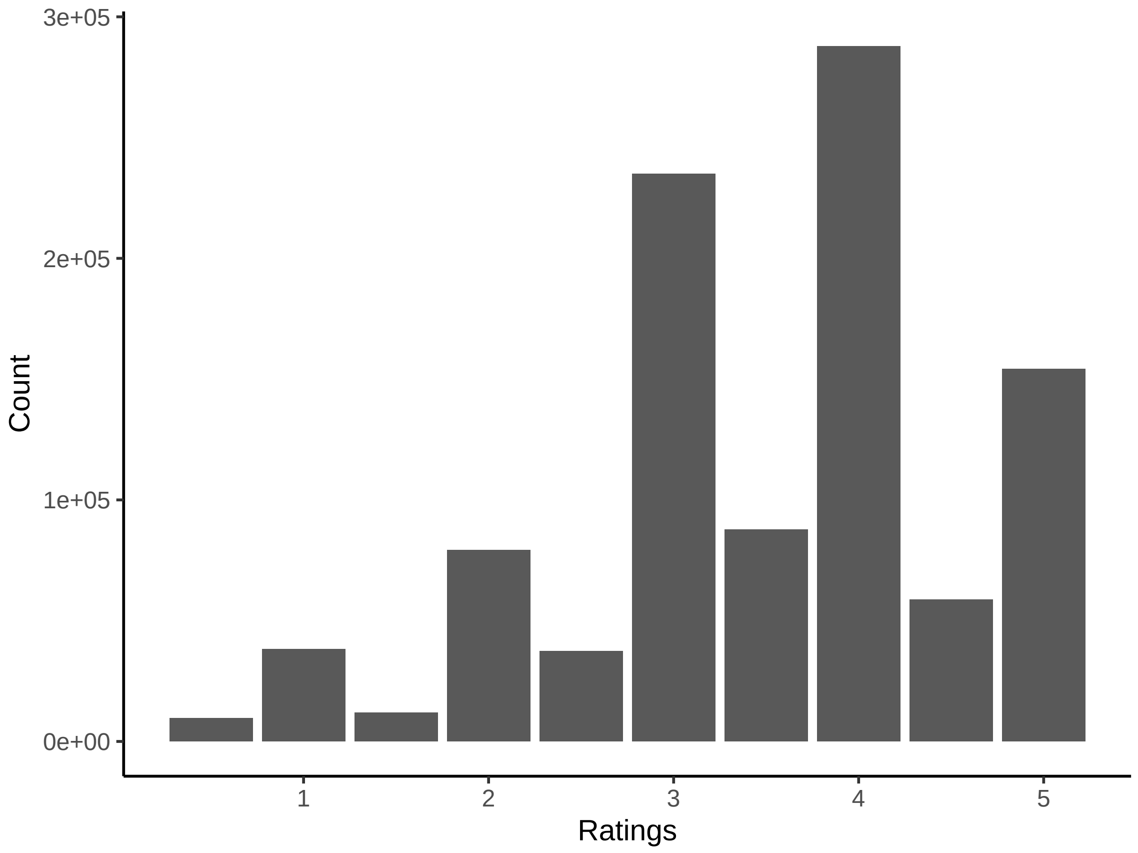
```
train_set %>% group_by(rating) %>% summarize(count = n()) %>%
  ggplot(aes(x=rating, y=count)) +
  geom_col() +
  xlab("Ratings") +
  ylab("Count") +
  theme_classic()
```



```r
test_set %>% group_by(rating) %>% summarize(count = n()) %>%
  ggplot(aes(x=rating, y=count)) +
  geom_col() +
  xlab("Ratings") +
  ylab("Count") +
  theme_classic()
```



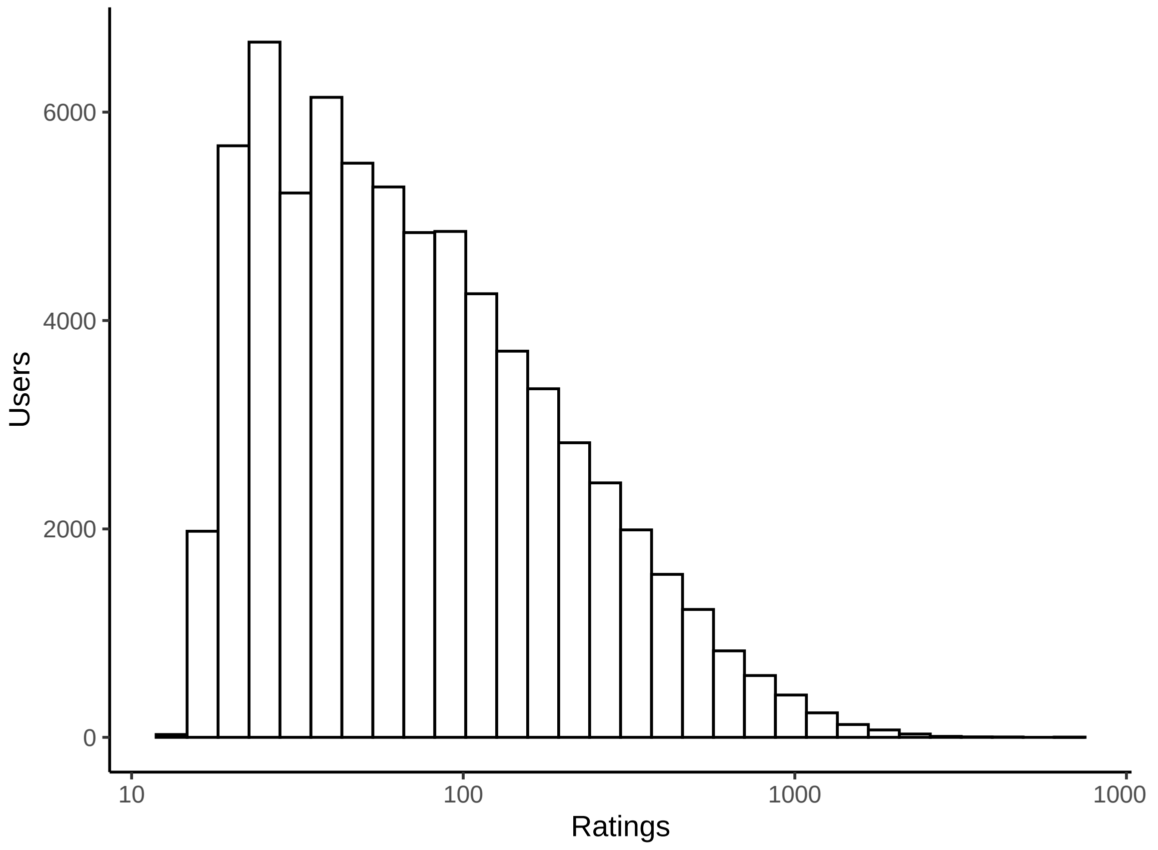
```r
validation %>% group_by(rating) %>% summarize(count = n()) %>%
  ggplot(aes(x=rating, y=count)) +
  geom_col() +
  xlab("Ratings") +
  ylab("Count") +
  theme_classic()
```



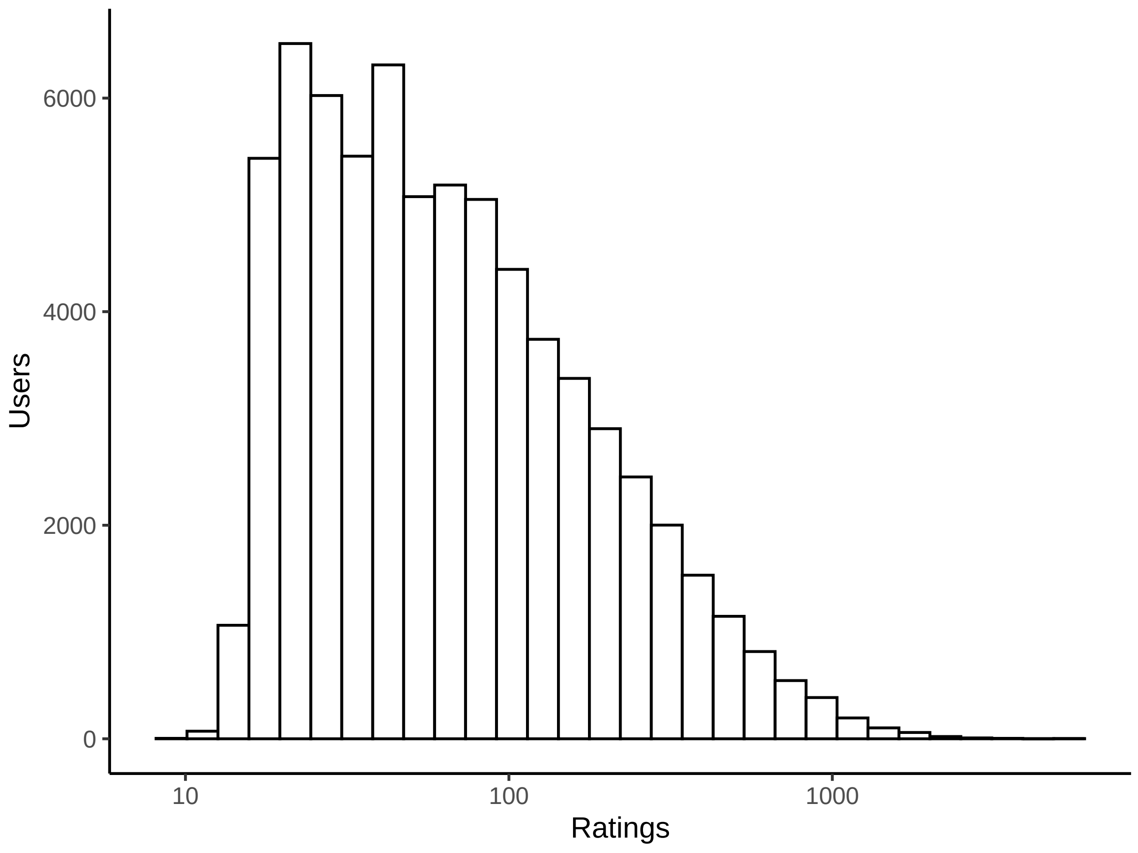
We observe that the ratings in all four datasets show a similar pattern. A rating of 4 has the largest number of entries in all the four datasets, and a rating of 0.5 has the least number of entries in all the four datasets. The bars for full ratings are much higher than their adjacent half ratings.

These similar patterns reveal that the datasets are representative of each other. Next, we explore the skew in the number of ratings done by the users in edx, train_set, test_set, and validation datasets. We do this by grouping the datasets by userId, counting the number of entries for each user, and then plotting the results in the form of a histogram using the ggplot2 library.

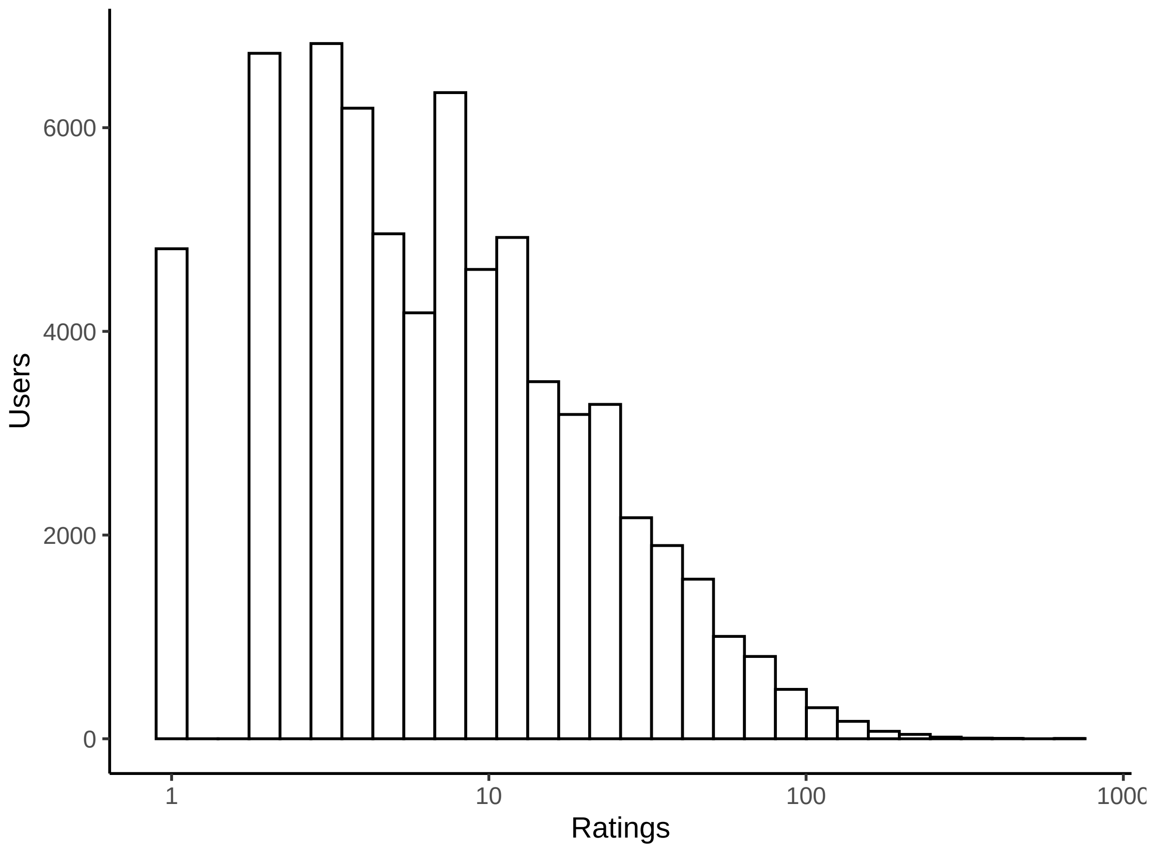
```
# Skew in distribution of ratings by users
edx %>% group_by(userId) %>% summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color="black", fill="white") +
  xlab("Ratings") +
  ylab("Users") +
  scale_x_log10() + theme_classic()
```



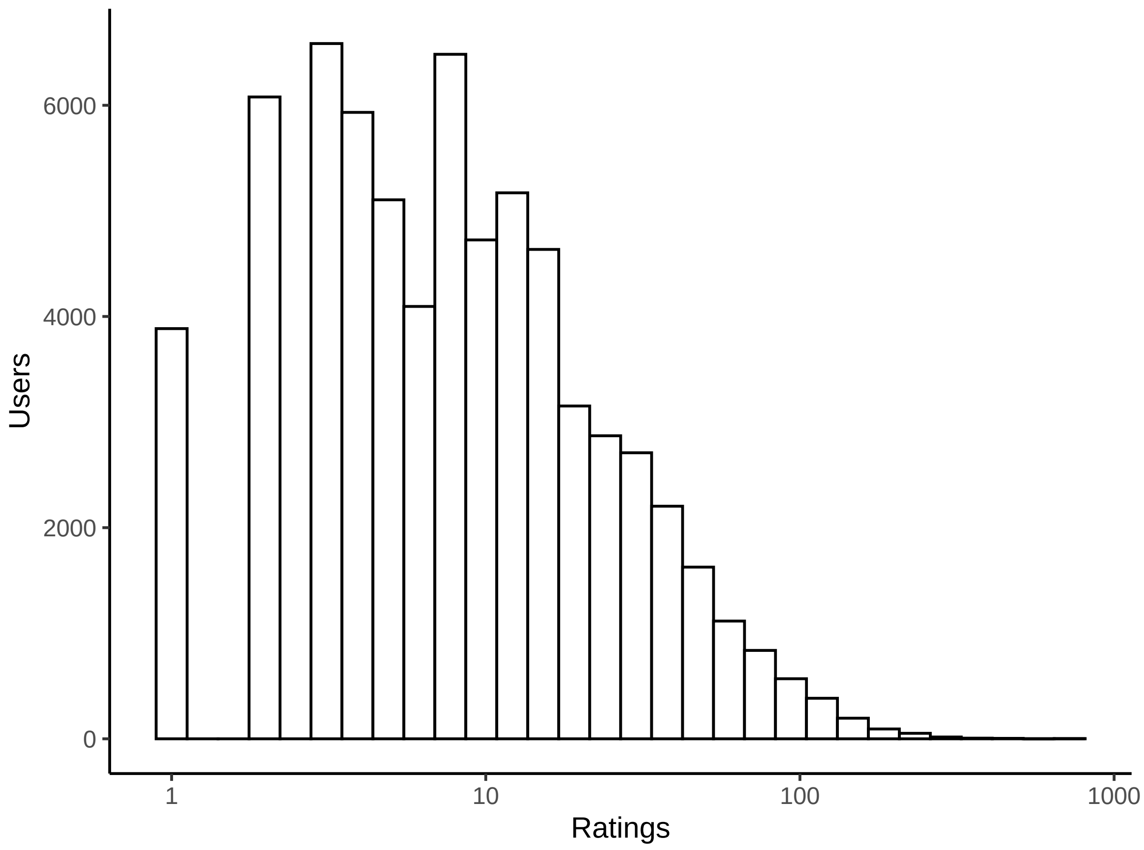
```
train_set %>% group_by(userId) %>% summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color="black", fill="white") +
  xlab("Ratings") +
  ylab("Users") +
  scale_x_log10() + theme_classic()
```



```
test_set %>% group_by(userId) %>% summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color="black", fill="white") +
  xlab("Ratings") +
  ylab("Users") +
  scale_x_log10() + theme_classic()
```



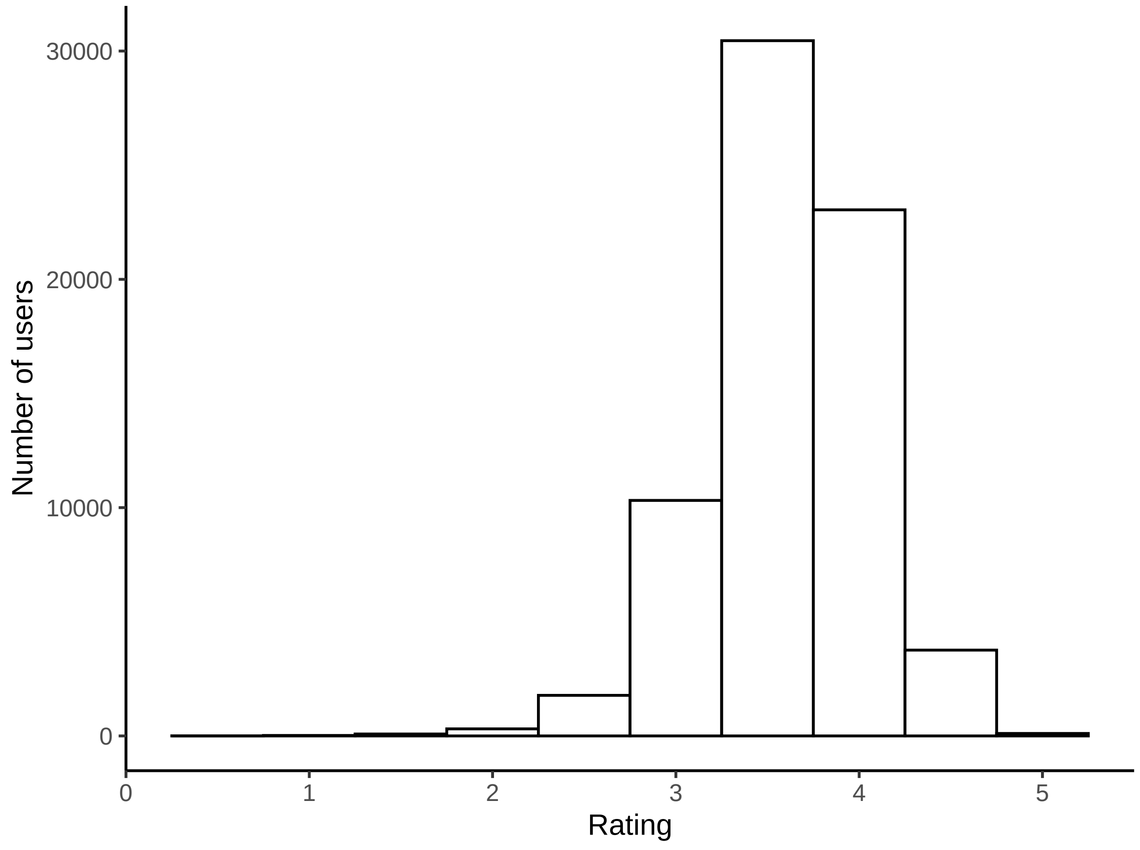
```
validation %>% group_by(userId) %>% summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color="black", fill="white") +
  xlab("Ratings") +
  ylab("Users") +
  scale_x_log10() + theme_classic()
```



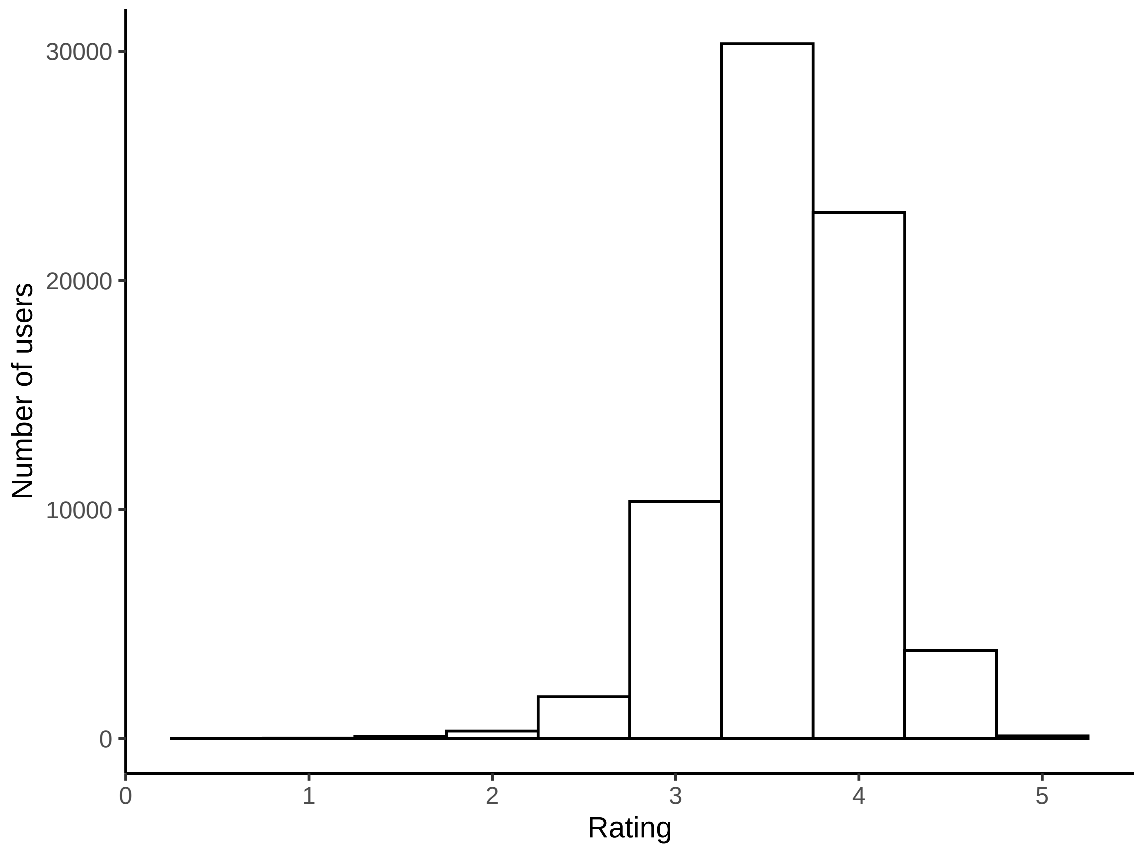
We observe that there are very few users that rate very few movies, or very large number of movies. Most of the users rate a sizeable number of movies, neither very small, nor very large. The distribution has a right tail, showing that there are a small number of users that do a large number of ratings. These users could be movie buffs and rating buffs.

We can also have a look at the average ratings given by different users.

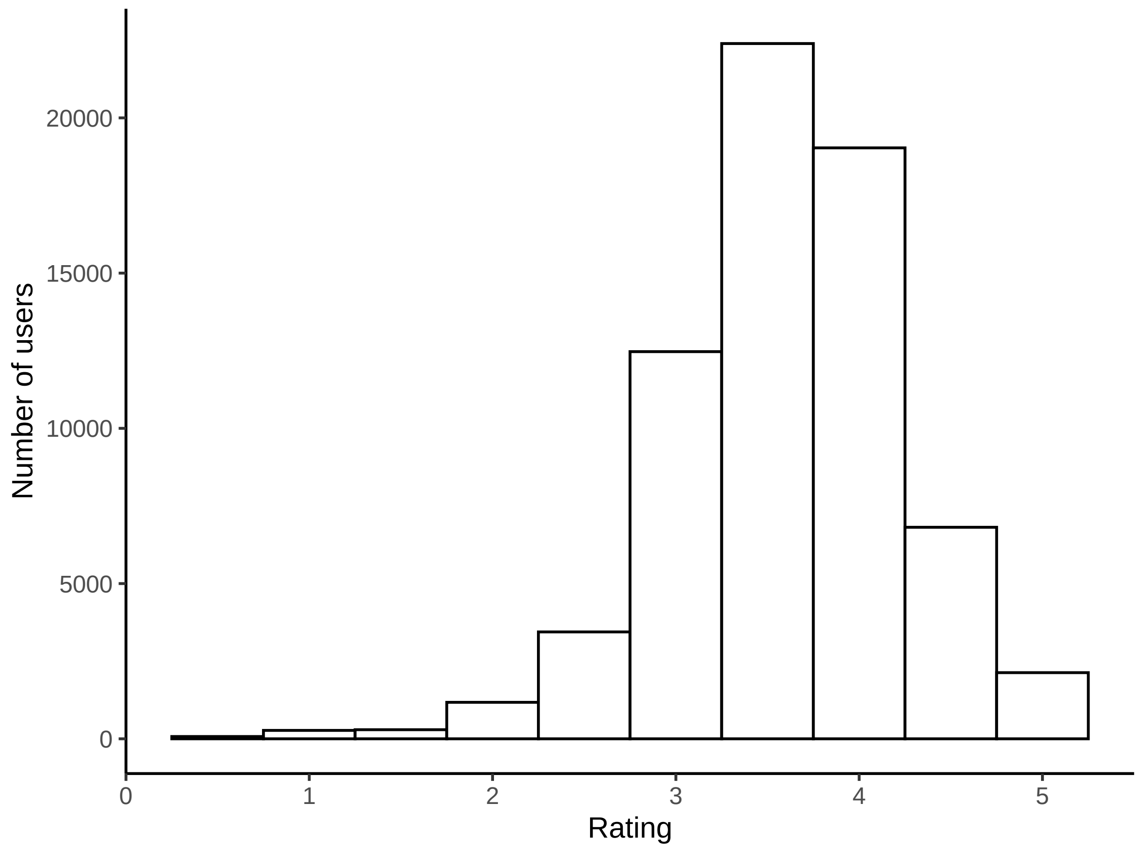
```
# Plot of average ratings by userId
edx %>% group_by(userId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(color="black", fill="white", binwidth=0.5) +
  xlab("Rating") +
  ylab("Number of users") +
  theme_classic()
```



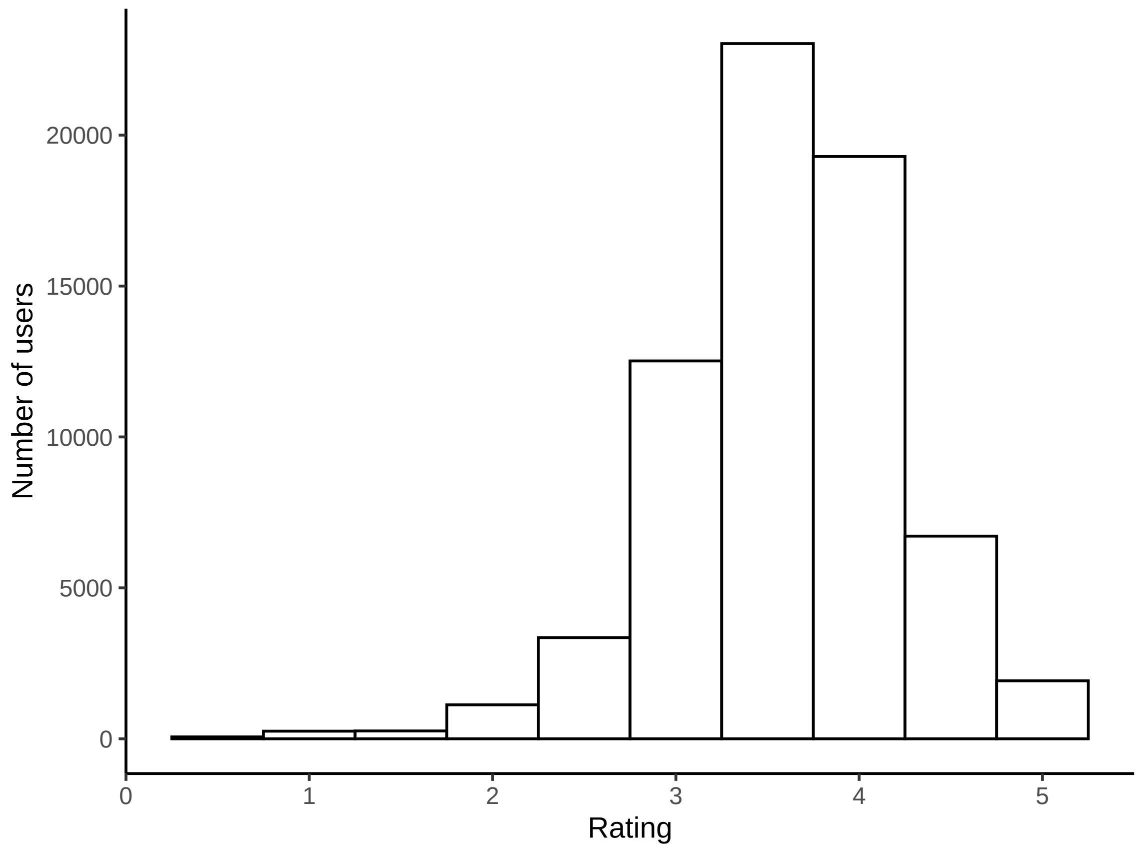
```
train_set %>% group_by(userId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(color="black", fill="white", binwidth=0.5) +
  xlab("Rating") +
  ylab("Number of users") +
  theme_classic()
```



```r
test_set %>% group_by(userId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(color="black", fill="white", binwidth=0.5) +
  xlab("Rating") +
  ylab("Number of users") +
  theme_classic()
```



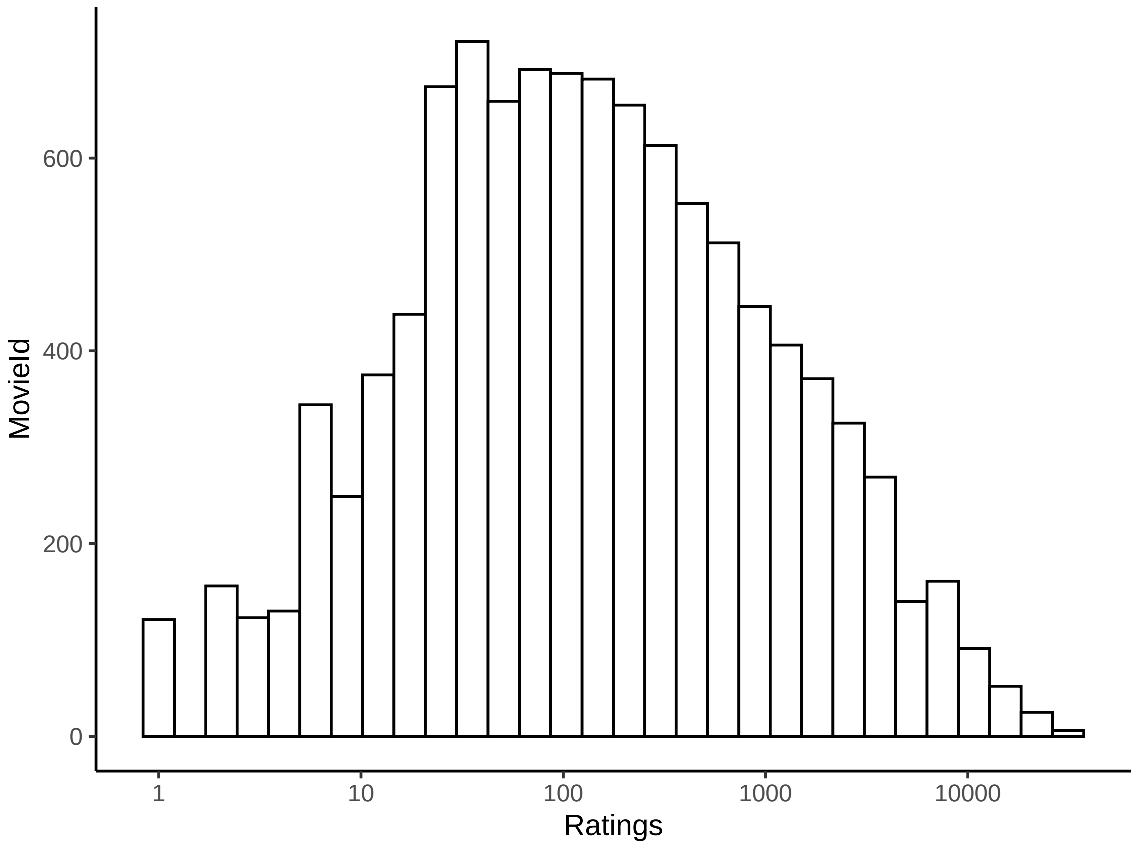
```
validation %>% group_by(userId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(color="black", fill="white", binwidth=0.5) +
  xlab("Rating") +
  ylab("Number of users") +
  theme_classic()
```



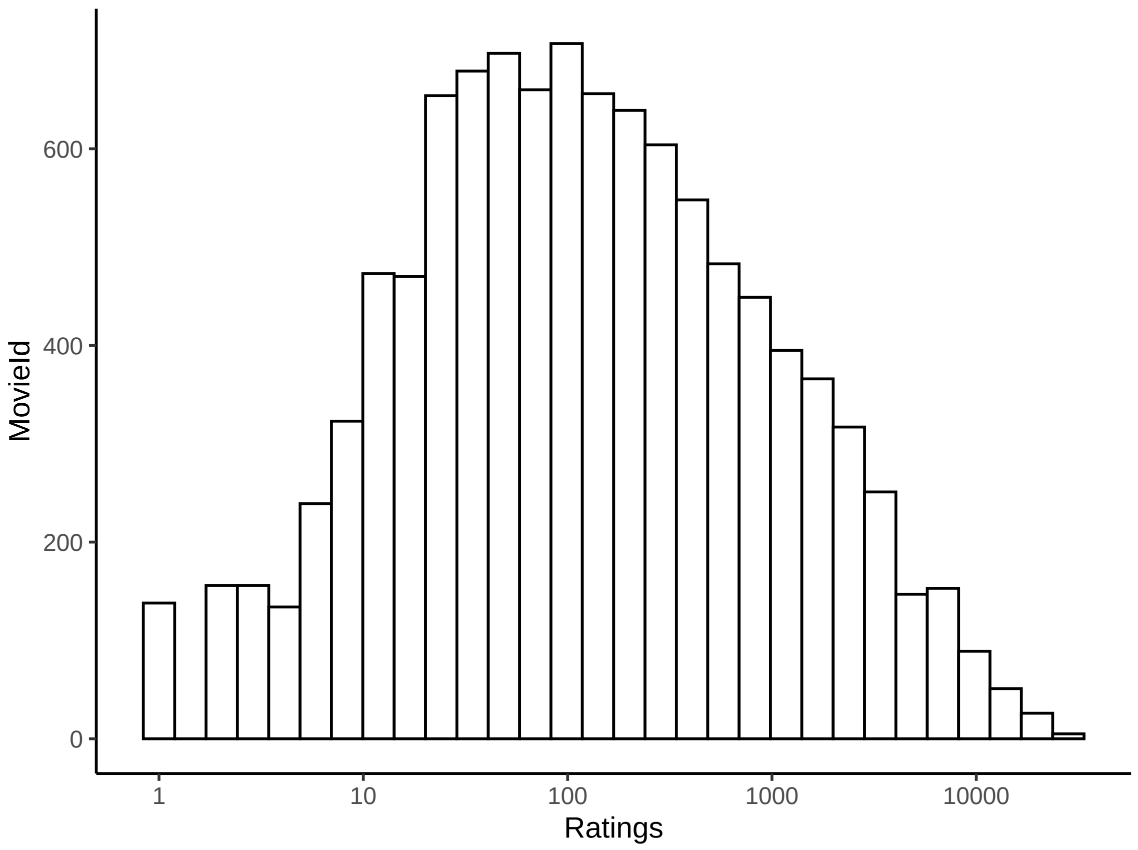
We can observe that there are some users who give very high ratings and some others who give very low ratings. The majority of users give average ratings. Thus, there is a very prominent user effect.

We can perform an exploration to understand the skew in the number of ratings received by different movies.

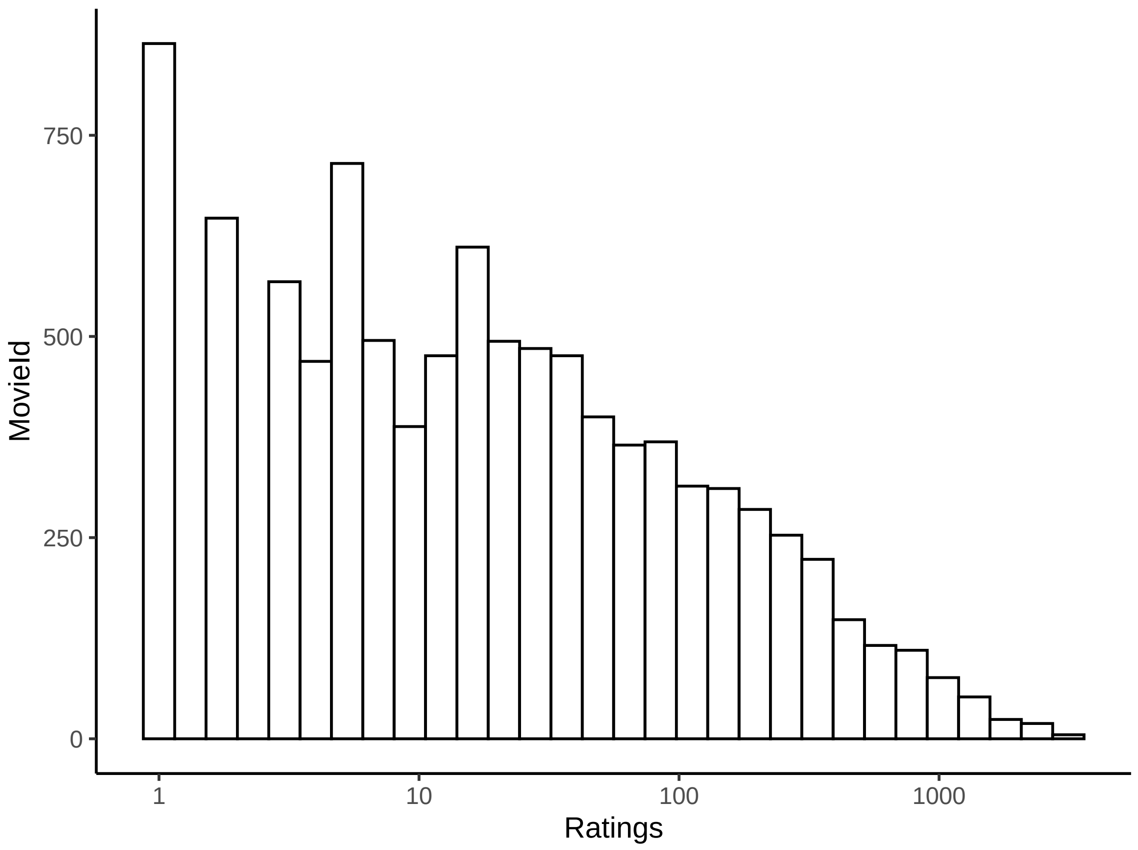
```
# Skew in distribution of ratings by movieId
edx %>% group_by(movieId) %>% summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color="black", fill="white") +
  xlab("Ratings") +
  ylab("MovieId") +
  scale_x_log10() + theme_classic()
```



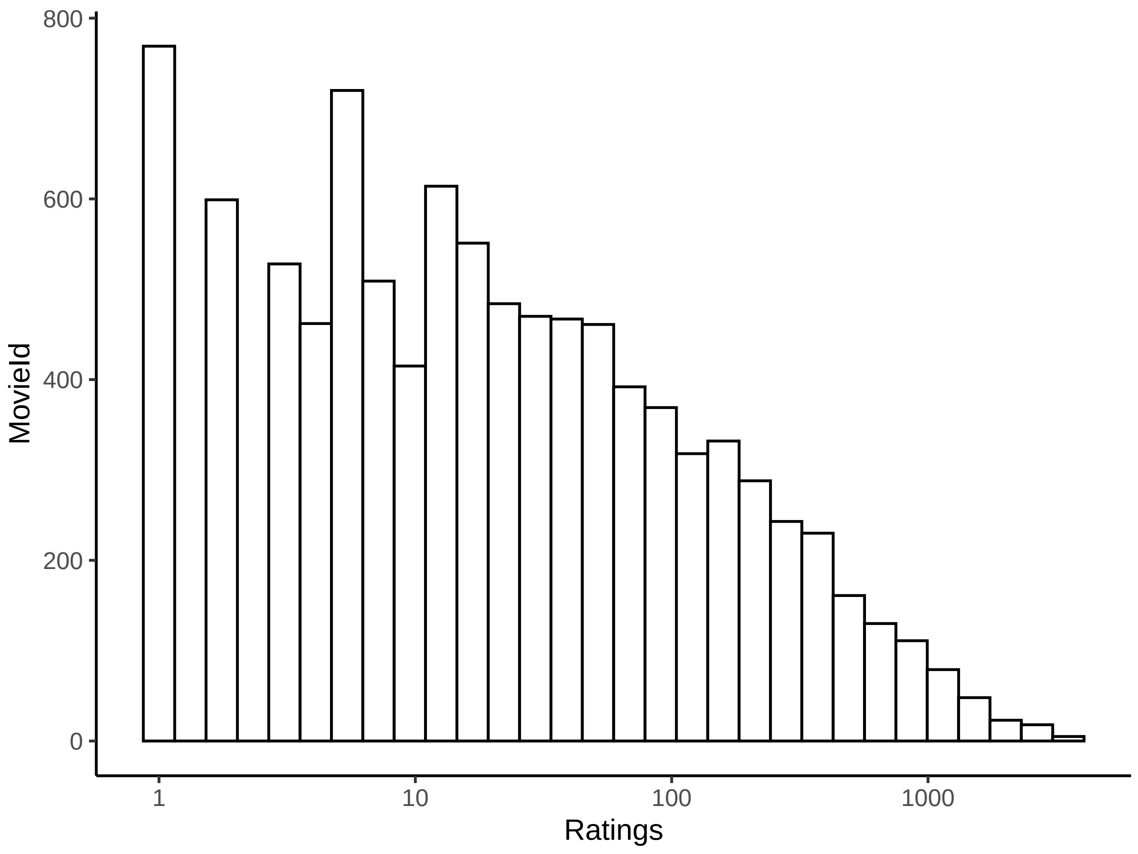
```
train_set %>% group_by(movieId) %>% summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color="black", fill="white") +
  xlab("Ratings") +
  ylab("MovieId") +
  scale_x_log10() + theme_classic()
```



```
test_set %>% group_by(movieId) %>% summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color="black", fill="white") +
  xlab("Ratings") +
  ylab("MovieId") +
  scale_x_log10() + theme_classic()
```



```
validation %>% group_by(movieId) %>% summarize(count = n()) %>%
  ggplot(aes(count)) +
  geom_histogram(color="black", fill="white") +
  xlab("Ratings") +
  ylab("MovieId") +
  scale_x_log10() + theme_classic()
```

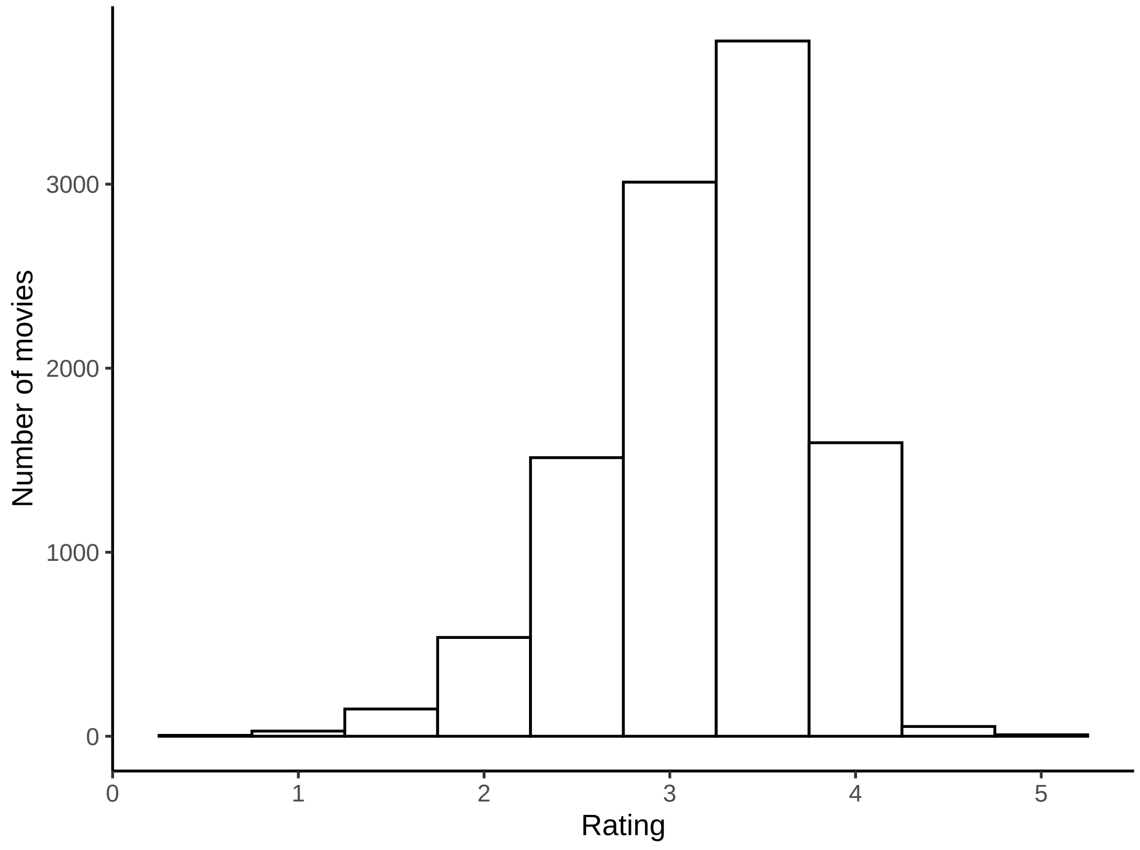


We observe that a majority of movies receive between 50 and 500 ratings. A few movies receive ratings in excess of 10,000. These are the blockbuster movies that have probably been watched and rated by a large number of users. There are some movies that receive very few ratings, less than 50. These are obscure movies that very few people have watched and rated.

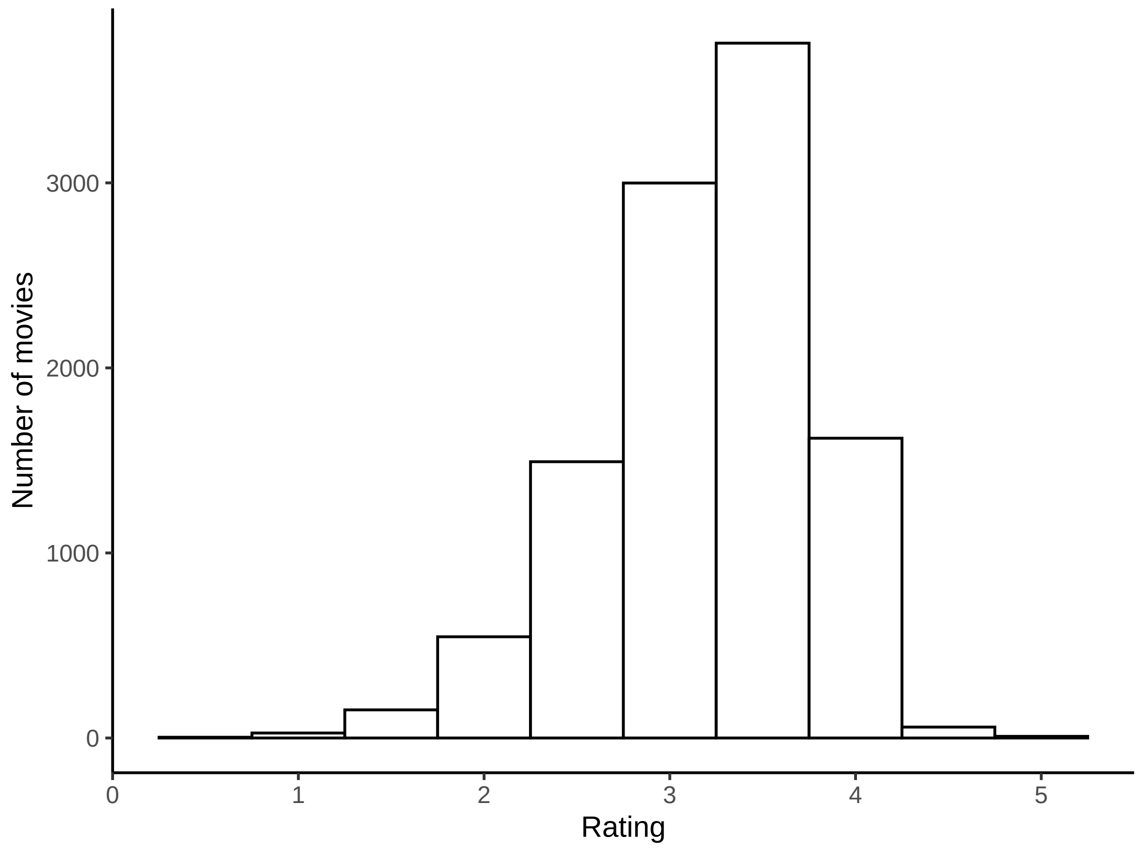
We observe that while all the four datasets have a right tail, the number of ratings are different in different datasets. This is expected since the datasets have different sizes.

We can also have a look at the average ratings received by different movies.

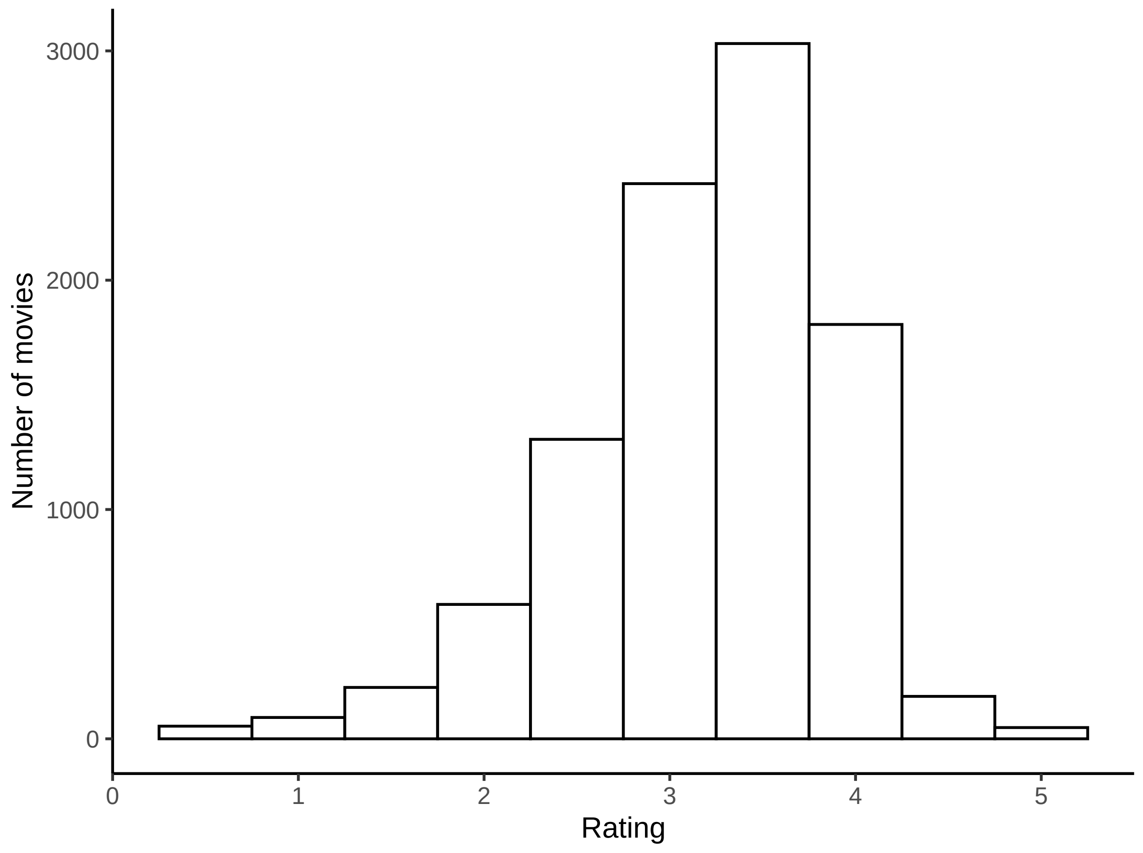
```r
# Plot of average ratings by movieId
edx %>% group_by(movieId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(color="black", fill="white", binwidth=0.5) +
  xlab("Rating") +
  ylab("Number of movies") +
  theme_classic()
```



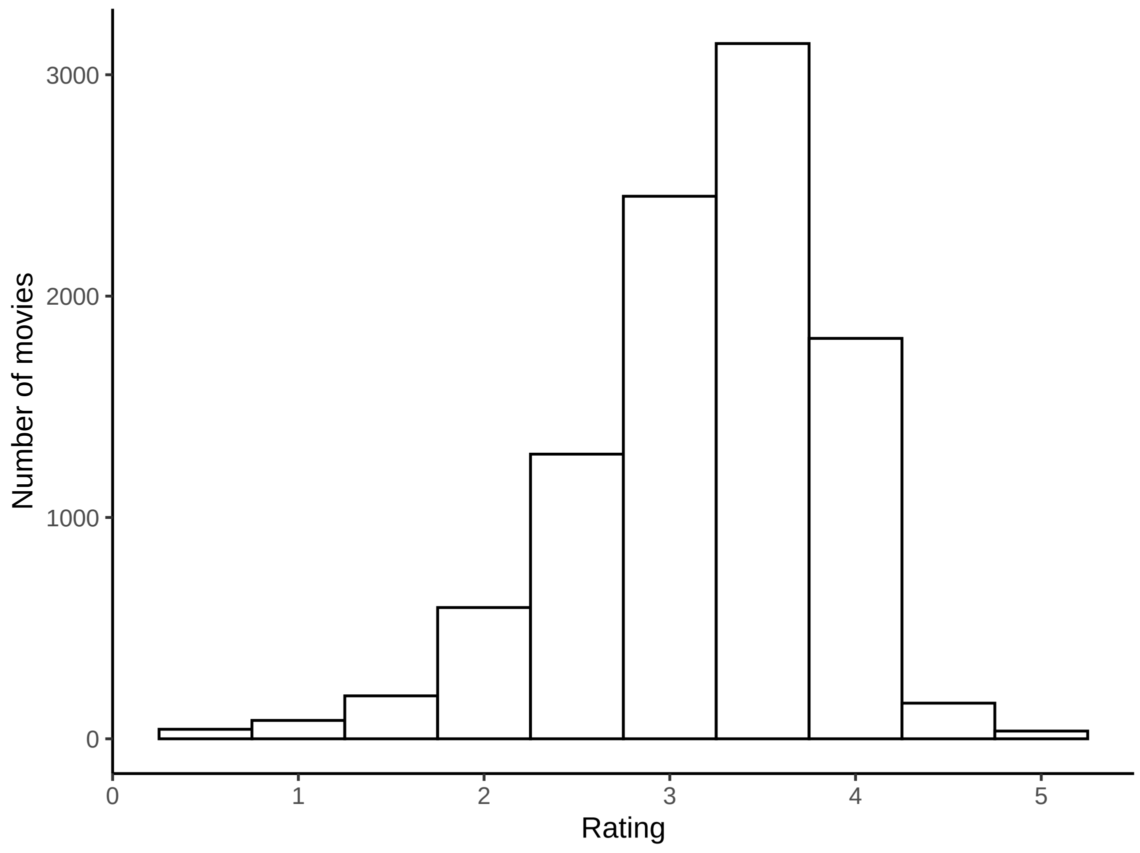
```r
train_set %>% group_by(movieId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(color="black", fill="white", binwidth=0.5) +
  xlab("Rating") +
  ylab("Number of movies") +
  theme_classic()
```



```
test_set %>% group_by(movieId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(color="black", fill="white", binwidth=0.5) +
  xlab("Rating") +
  ylab("Number of movies") +
  theme_classic()
```



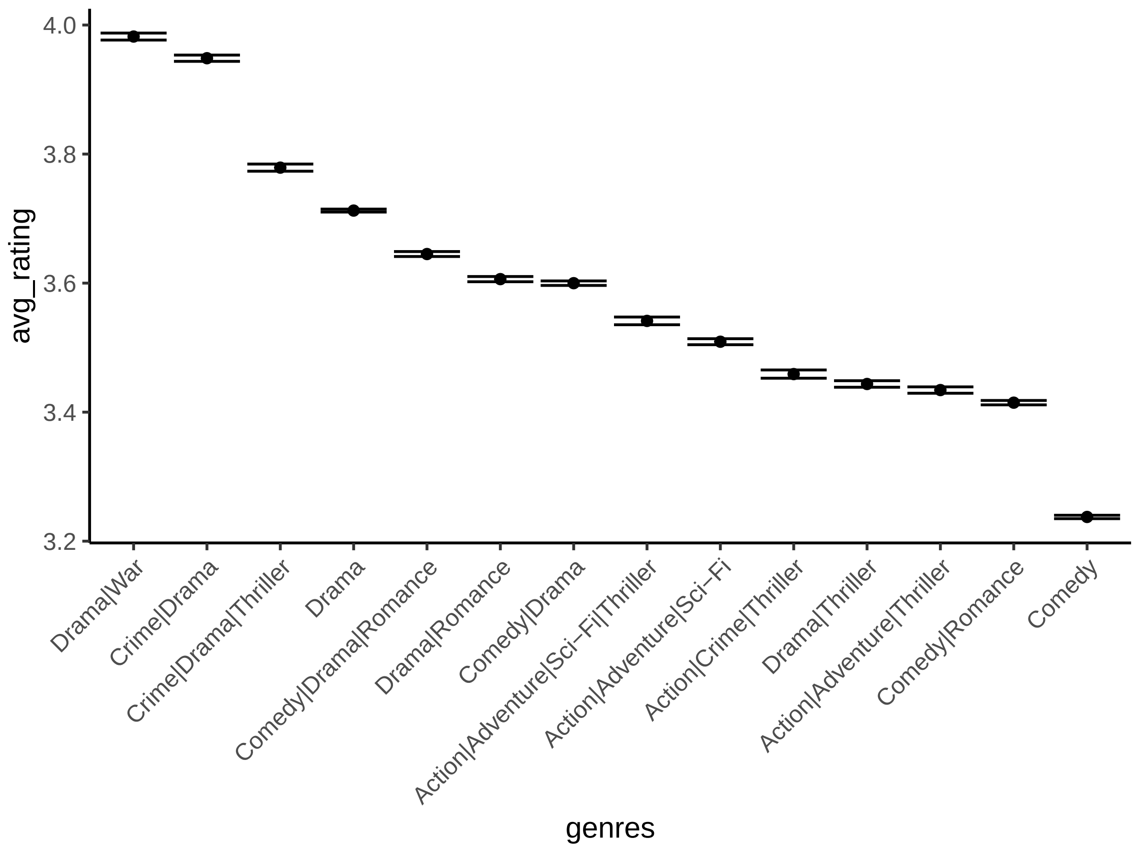
```
validation %>% group_by(movieId) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(avg_rating)) +
  geom_histogram(color="black", fill="white", binwidth=0.5) +
  xlab("Rating") +
  ylab("Number of movies") +
  theme_classic()
```



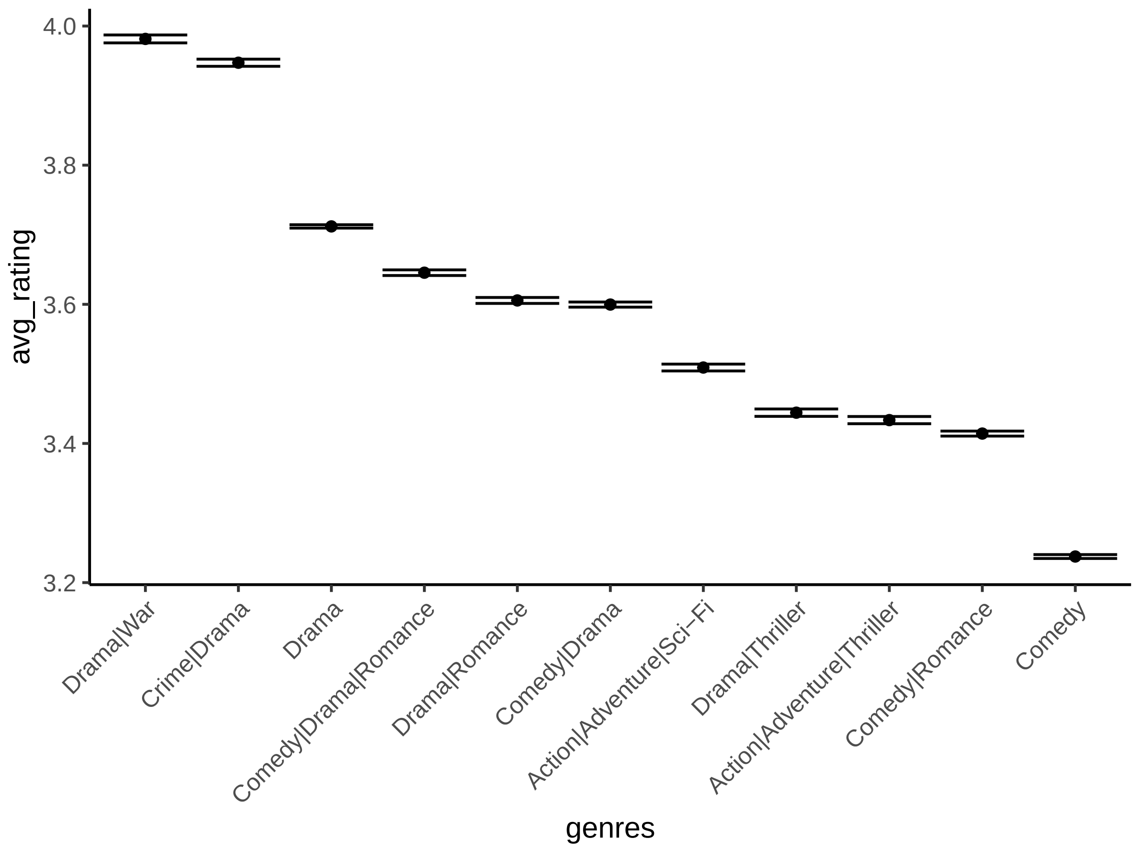
We can observe that there are some movies with very high ratings and some others with very low ratings. The majority of movies receive average ratings. Thus, there is a very prominent movie effect.

Next, we ask the question: Is the average rating of a movie dependent on the genre it belongs to? To answer this question, we can group the edx, train_set, test_set, and validation datasets by genres, and calculate the number of ratings, the average rating, and the standard error of ratings for each genre. To ensure that we only take the most representative data points, we filter those genres with more than 100,000 ratings in the larger datasets - edx and train_set, and genres with more than 10,000 ratings in the smaller datasets - test_set and validation. We then reorder the genres in descending order of average ratings, and plot the average ratings of each genre in the four datasets.

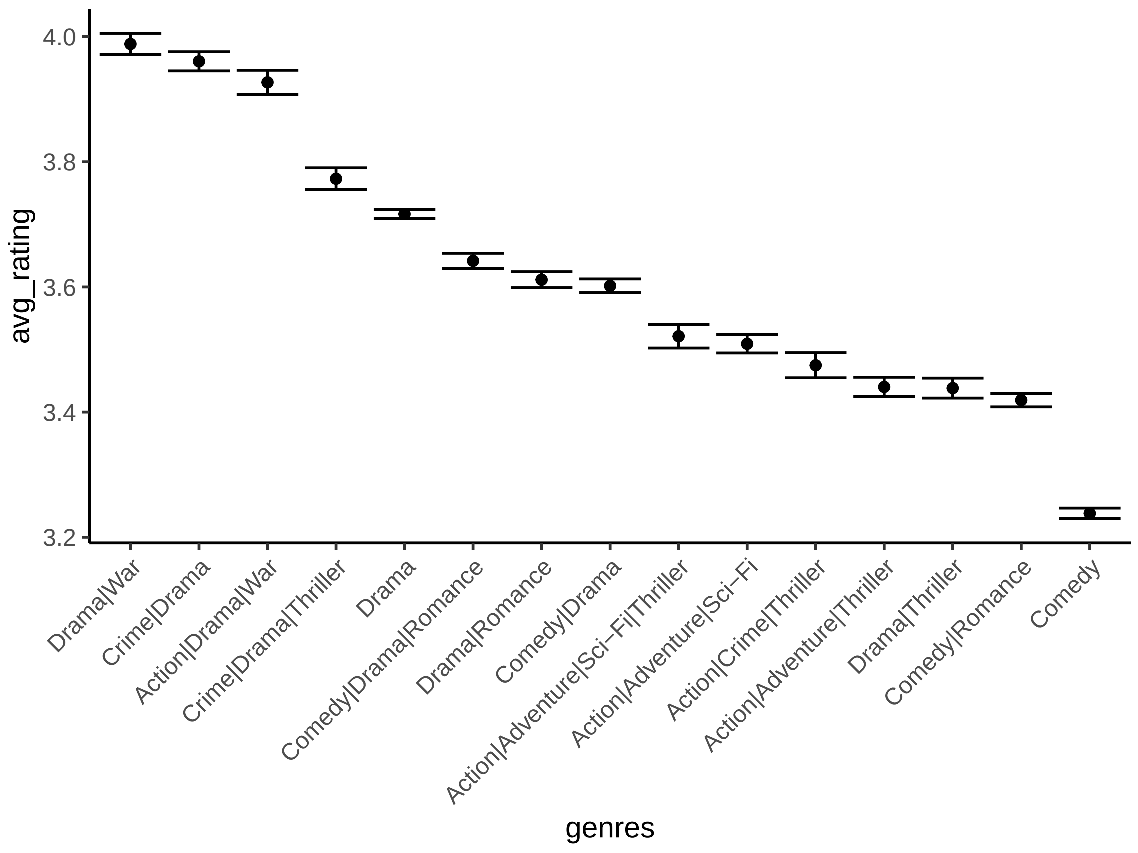
```
# Evidence of genre effect on average ratings
# Using a smaller size of n as filter when there are lesser data
# points
edx %>% group_by(genres) %>%
  summarize(n = n(), avg_rating = mean(rating),
  se_rating = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, desc(avg_rating))) %>%
  ggplot(aes(x = genres, y = avg_rating,
  ymin = avg_rating - 2*se_rating,
  ymax = avg_rating + 2*se_rating)) +
  geom_point() + geom_errorbar() + theme_classic() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))
```



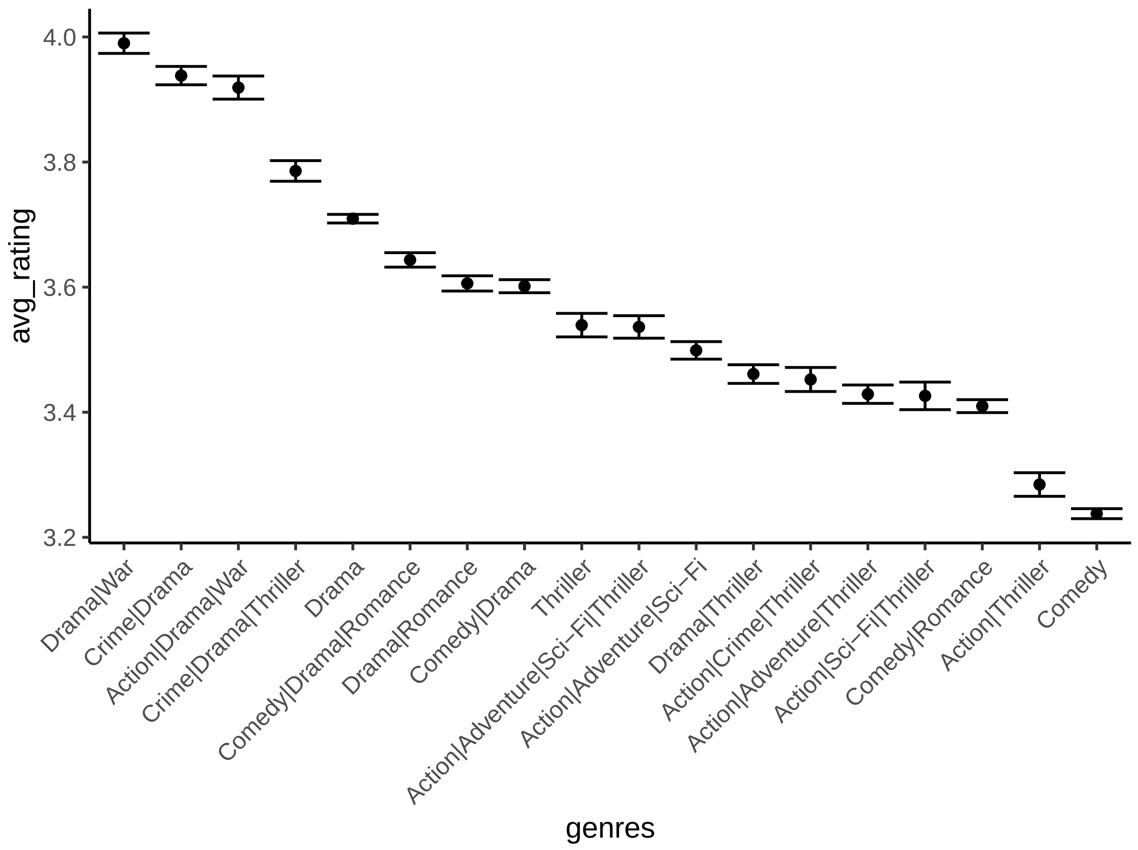
```r
train_set %>% group_by(genres) %>%
  summarize(n = n(), avg_rating = mean(rating),
  se_rating = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, desc(avg_rating))) %>%
  ggplot(aes(x = genres, y = avg_rating,
  ymin = avg_rating - 2*se_rating,
  ymax = avg_rating + 2*se_rating)) +
  geom_point() + geom_errorbar() + theme_classic() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))
```



```
test_set %>% group_by(genres) %>%
  summarize(n = n(), avg_rating = mean(rating),
  se_rating = sd(rating)/sqrt(n())) %>%
  filter(n >= 10000) %>%
  mutate(genres = reorder(genres, desc(avg_rating))) %>%
  ggplot(aes(x = genres, y = avg_rating,
  ymin = avg_rating - 2*se_rating,
  ymax = avg_rating + 2*se_rating)) +
  geom_point() + geom_errorbar() + theme_classic() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))
```



```
validation %>% group_by(genres) %>%
  summarize(n = n(), avg_rating = mean(rating),
  se_rating = sd(rating)/sqrt(n())) %>%
  filter(n >= 10000) %>%
  mutate(genres = reorder(genres, desc(avg_rating))) %>%
  ggplot(aes(x = genres, y = avg_rating,
  ymin = avg_rating - 2*se_rating,
  ymax = avg_rating + 2*se_rating)) +
  geom_point() + geom_errorbar() + theme_classic() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust=1))
```



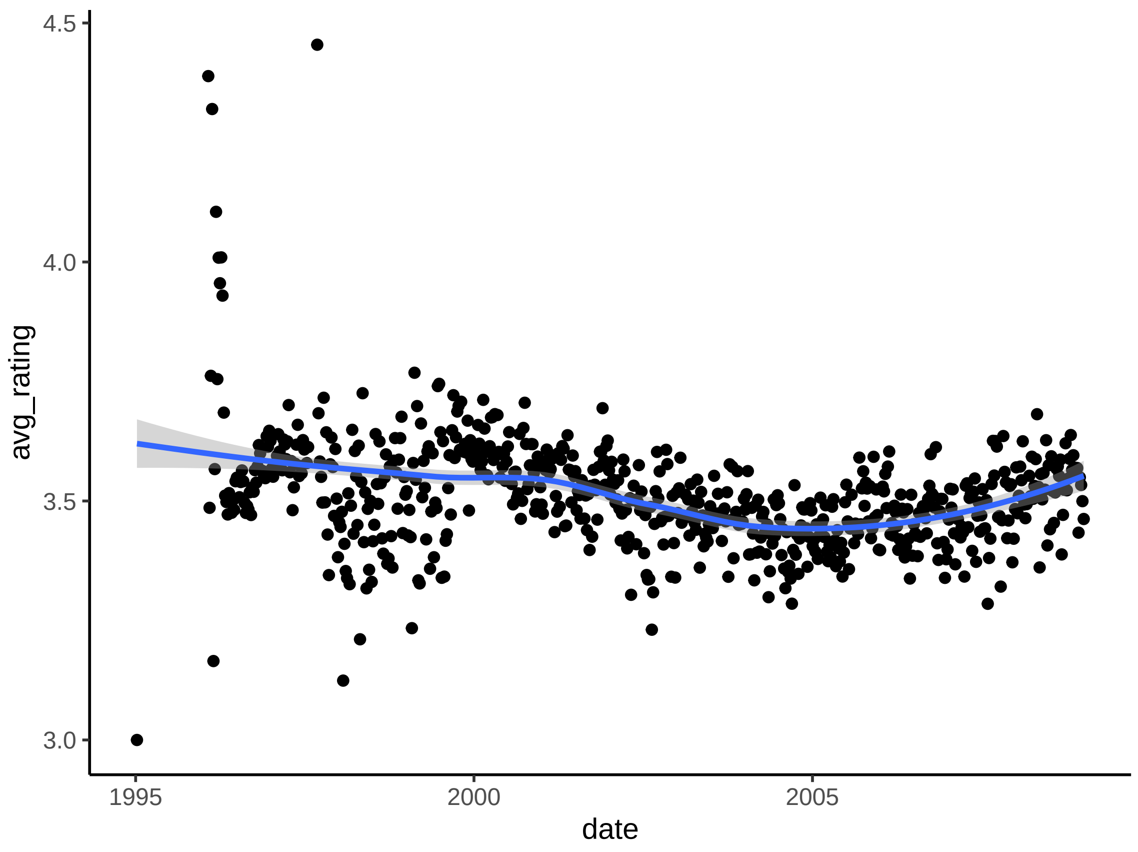
We observe that there is a prominent genre effect for ratings. In all the four datasets, the Drama|War genre receives the highest ratings, followed by the Crime|Drama genre. In all the four datasets, the average ratings of the movies in the comedy genre are around 0.8 points lower than the average ratings of the movies in the Drama|War genre.

This gives us a powerful insight: When deciding the rating of a new movie, the genre that the movie belongs to will have a large impact.

We can perform a similar exploration with the time effect. To explore the effect of time on ratings, we group the entries in the four datasets - edx, train_set, test_set and validation, by the week of the timestamp information. We then compute the average ratings of the movies in each week, and plot them using the ggplot2 library. We plot the data in the form of points and a smoothed conditional means line.

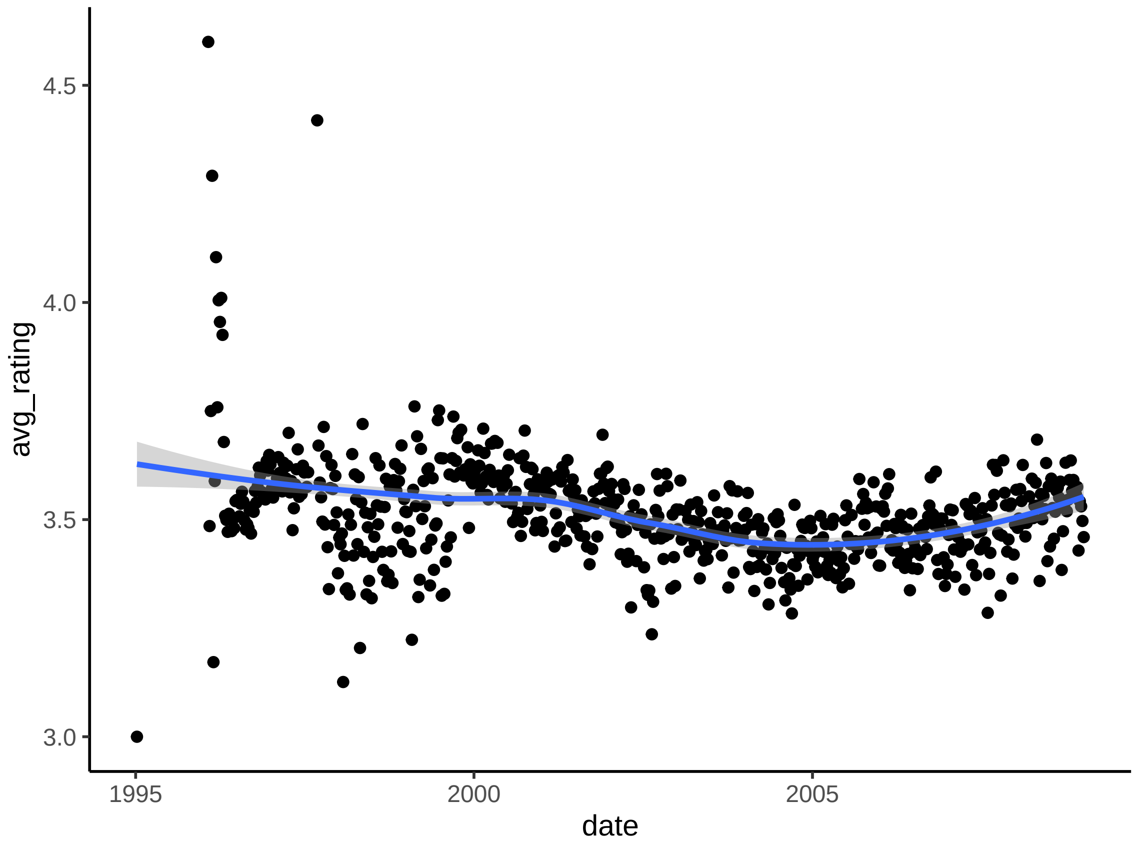
```
# A look at time effect
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(date, avg_rating)) +
  geom_point() +
  geom_smooth() + theme_classic()
```

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



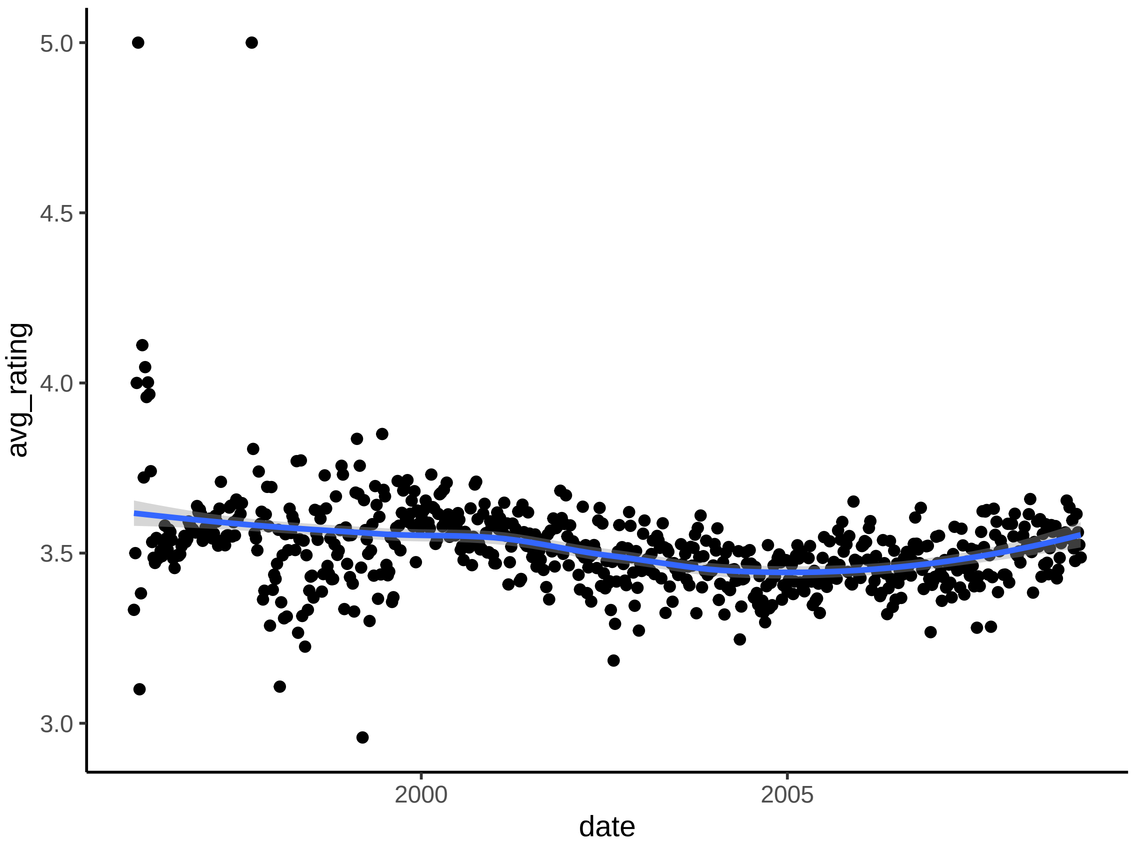
```
train_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(date, avg_rating)) +
  geom_point() +
  geom_smooth() + theme_classic()
```

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



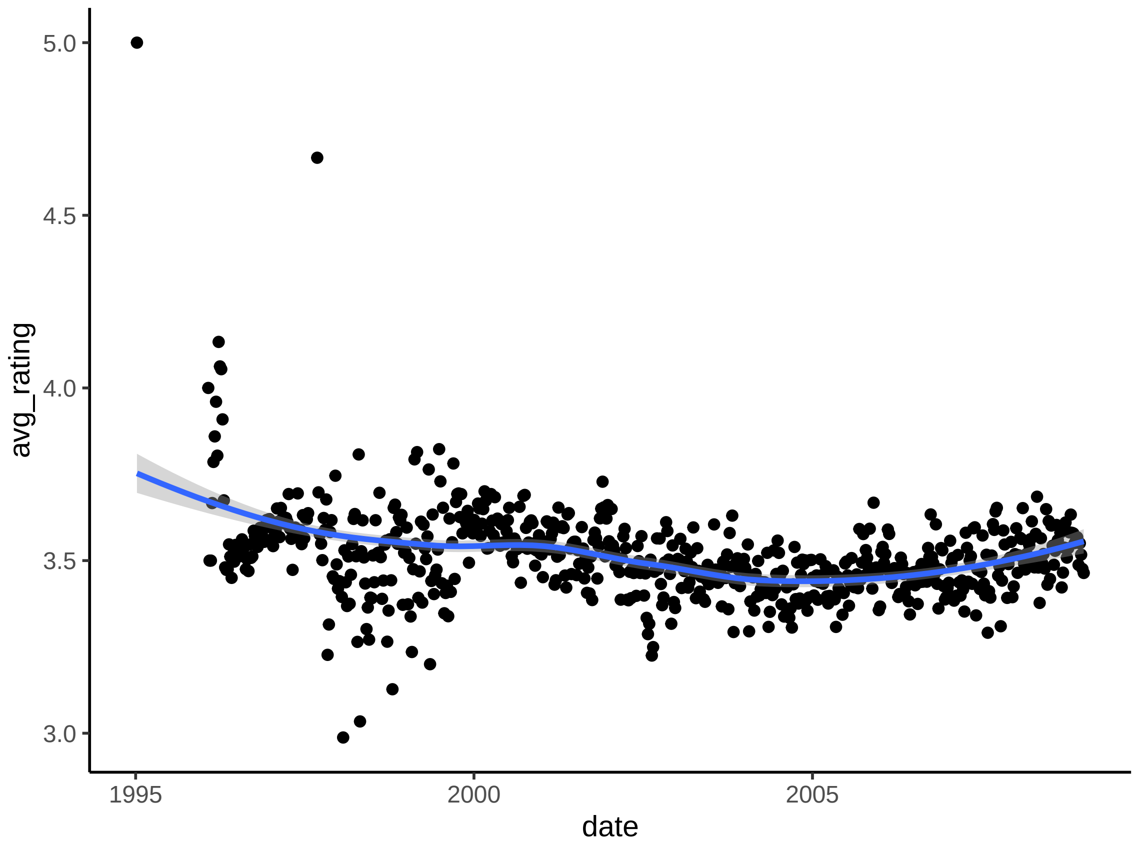
```
test_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(date, avg_rating)) +
  geom_point() +
  geom_smooth() + theme_classic()
```

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



```
validation %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(date, avg_rating)) +
  geom_point() +
  geom_smooth() + theme_classic()
```

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



We observe that in all the four datasets, the average ratings of the movies hover around 3.5. This fact corroborates with the insight gained from plotting the column plots of movie ratings. We can see that the average ratings show a small, but significant time effect. For instance, the ratings around the year 2005 are lesser than the ratings before and after. We also observe that movies in the 1990s have much higher ratings on average than the movies made after. This could be due to nostalgia, but could also represent the fact that the availability of cheap digital movie making technology in the 2000s permitted the creation of a large number of movies, many of which were not great. On the other hand, in the 1990s, the high cost of movie making meant that only the best scripts, directors, actors, etc. had a chance to appear in a movie, making for a much better movie. Yet another reason could be that the movies of 1990s had begun experimenting in new technologies and styles, making them much more memorable than the later movies when the technologies and styles had lost their novelty. Compare, for example, Jurassic Park (1993) with Jurassic Park III (2001). Watching the magnificent dinosaurs in 1993 was a very different experience than watching them again in 2001. Jurassic Park III (2001) also did not have the benefit of a Michael Crichton story, unlike Jurassic Park (1993) and The Lost World (1997), which did.

From the point of view of making a recommender system, we can note that there is a distinct and significant time effect, particularly when moving from the 1990s to the 2000s and beyond, where the time effect on ratings becomes more diluted.

Insights gained

1. The datasets have been correctly generated, and are fairly representative of each other. Thus, algorithms trained on one dataset can be tested or validated on the other datasets.

2. Some movies are much better and some are much worse rated than the average. Thus, there is a distinct and significant movie effect.

3. Some users give much higher ratings and some give much lower ratings than the average. Thus, there is a distinct and significant user effect.

4. Full ratings are much more common than half ratings.

5. Movies in some genres have very high average ratings, while movies in some other

genres have quite low average ratings. Thus, there is a distinct and significant genre effect.

6. There is a perceptible, but not very strong, time effect. Movies of different time periods have different average ratings.

Approach to modelling

To model and predict the ratings, we have, so far, downloaded and cleaned the dataset, partitioned the dataset into working (training and testing) and validation datasets, and performed exploratory data analysis to understand the datasets and get insights about their salient attributes.

These insights have revealed that movie ratings demonstrate significant and strong movie effect, user effect, and genre effect, and significant but not very strong, time effect. This gives us an idea that if we were to incorporate all four of these effects - movie effect, user effect, genre effect, and time effect - into our algorithm, our predictions should be able to come very close to the actual ratings. But we must also be mindful of using penalised least squares so that the training data with very few ratings do not overpower our computations.

With this in mind, now we will train different algorithms on the training dataset, and test them on the testing dataset. We will evaluate the different algorithms through computation of root mean square error between predictions and the ratings in the testing dataset, to select the best (or set of best) algorithm(s).

The algorithms will include:

1. Using a random rating as the best prediction: Without any detailed knowledge,

if we were to predict the rating of an unknown movie, we can use a random rating (say 2.5, since it lies in the middle of the rating range 0-5) as the first best approximate rating, and try to improve the RMSE values from here.

2. Using the average movie rating: With some knowledge of the data, if we were to predict the rating of an unknown movie, we would best utilise the average rating of the known rated movies. We will use this as the second best approximate rating.

3. Utilising the movie effect: We have observed that some movies are better rated, and some others are worse rated than the average. Thus, if we take the known ratings of the movies, we can use these ratings as the third best approximate.

4. Utilising the movie + user effect: Similar to the movie effect, we have observed that there is a significant user effect. Thus, if we take the known ratings of the movies together with the known propensities of different users to give good or bad ratings, we will come to a fourth best approximate.

5. Utilising the movie + user + genre effect: Similar to the movie and user effects, we have observed that there is a significant genre effect. Thus, if we take the known ratings of the movies together with the known propensities of different users to give good or bad ratings, and adjust for the genre effect, we will come to a fifth best approximate.

6. Utilising the movie + user + genre + time effect: Similar to the movie, user and genre effects, we have observed that there is a significant time effect. Thus, if we take the known ratings of the movies together with the known propensities of different users to give good or bad ratings, adjust for the genre effect, and the time effect, we will come to a sixth best approximate.

7. Regularisation of movie effect: We will use a penalty lambda for those movies that have less number of ratings. This is because for those movies that have a large number of ratings, we can be more confident about their average readings being representative, while for those movies that have fewer ratings, it is possible that the average ratings will fluctuate when more users rate them. Thus, the known average ratings of those movies will be less representative. Through the process of regularisation, we will try to improve upon the predictions by incorporating the representativeness of the known ratings. This regularisation will be attempted through a randomly selected lambda, and also through a lambda selected through the data.

8. Regularisation of movie + user effect: We will regularise both the movie and user effects to get a good prediction of ratings.

9. Regularisation of movie + user + genre effect: We will regularise the movie, user and genre effects to get a good prediction of ratings.

10. Regularisation of movie + user + genre + time effect: We will regularise the movie, user, genre and time effects to get a good prediction of ratings.

Finally, the algorithm that gives the lowest RMSE will be selected and tested on the validation dataset to check how well the algorithm does on an out-sample.

Results and discussion

RMSE generation function

We begin by creating a function to generate root mean square error (RMSE) values. This function takes the true ratings and predicted ratings as inputs and returns the RMSE value between the true and predicted ratings. The root mean square error is computed as the square root of the mean of squares of differences between true and predicted ratings.

```r
# Creating function to generate RMSE values
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm=TRUE))
}
```

Next we employ different methods discussed in the course, and other methods based on the course to build the recommendation system.

Training and testing machine learning algorithms

The simplest system will predict any randomly selected rating as the expected rating for all the movies of the test_set dataset. Since the ratings go in half points from 0.5 to 5 (both inclusive), let us take a randomly selected value, say 2.5 as the predicted rating, mu_hat.

To test how good this random value predicts the ratings of the test_set dataset, we compute the RMSE value using the RMSE function created earlier. We store the results in a data frame named rmse_results.

```r
### Building the Recommendation System: Using a random value (2.5)
# for all the movies of the test_set dataset
mu_hat <- 2.5
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.466
```

```r
rmse_results <- data_frame(method = "Random value (2.5)",
    RMSE = naive_rmse)
```

How can we improve upon the value? Probably by taking a better value. We know through statistics that the mean rating is a measure of central tendency, or a representative value for a set of data. We can use the mean rating of the train_set dataset as the most representative rating for any movie. Thus, we create a second recommendation system, where we use the mean rating of the train_set dataset as the suggested rating for every movie in the test_set dataset.

To deploy this recommendation system, we use the value of mean(train_set$rating) as the expectation of the movie rating, mu_hat.

To test how good this recommendation system predicts the ratings of the test_set dataset, we compute the RMSE value using the RMSE function created earlier. We store the results in the data frame named rmse_results.

```r
### Building the Recommendation System: Using the average rating of
# the train_set for all the movies of the test_set dataset
```

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

```
## [1] 3.512
```

```
mu_rmse <- RMSE(test_set$rating, mu_hat)
mu_rmse
```

```
## [1] 1.059
```

```
rmse_results <- bind_rows(rmse_results,
  data_frame(method = "Using the average",
  RMSE = mu_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Random value (2.5)	1.466
Using the average	1.059

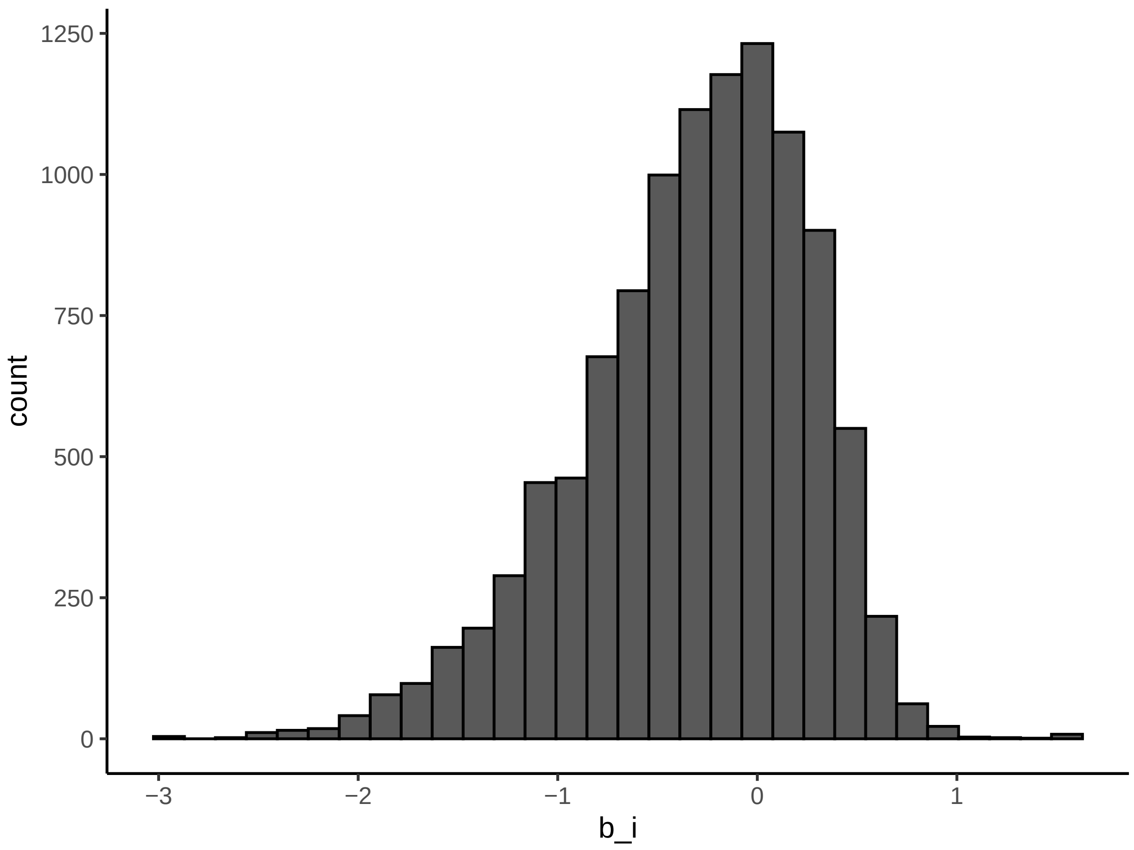
We find that the RMSE has improved (reduced) a lot. Can we improve the RMSE value further? We can make use of the fact that not all movies are the same. Some are great movies, and they receive high ratings. Others are not so good movies, and they receive lesser ratings. We can use the average rating of a movie as the expectation of the rating for that movie.

To deploy this recommendation system, we group the train_set dataset by movieId, and compute the average difference between the rating of the movie and the mean rating of every movie. If we plot this difference as a histogram, we will find that most of the movies will have a zero average difference (because most of the movies will have around average ratings, making the difference from average rating nearly zero). And the histogram will taper towards both ends.

To predict the ratings in the test_set dataset, we add the values obtained for each movie to the mean value of ratings. For those movies that are not there in the test_set dataset, the method will predict the average rating of every movie as the expected rating for that particular movie.

To test how good this recommendation system predicts the ratings of the test_set dataset, we compute the RMSE value using the RMSE function created earlier. We store the results in the data frame named rmse_results.

```
### Improving by incorporating the fact that some movies are great,
# some are not. Movie effect model.
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))


movie_avgs %>% ggplot(aes(b_i)) +
                geom_histogram(bins = 30, color = "black") +
                theme_classic()
```



```r
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i


model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
  data_frame(method="Movie Effect Model",
  RMSE = model_1_rmse))
rmse_results %>% knitr::kable()
```

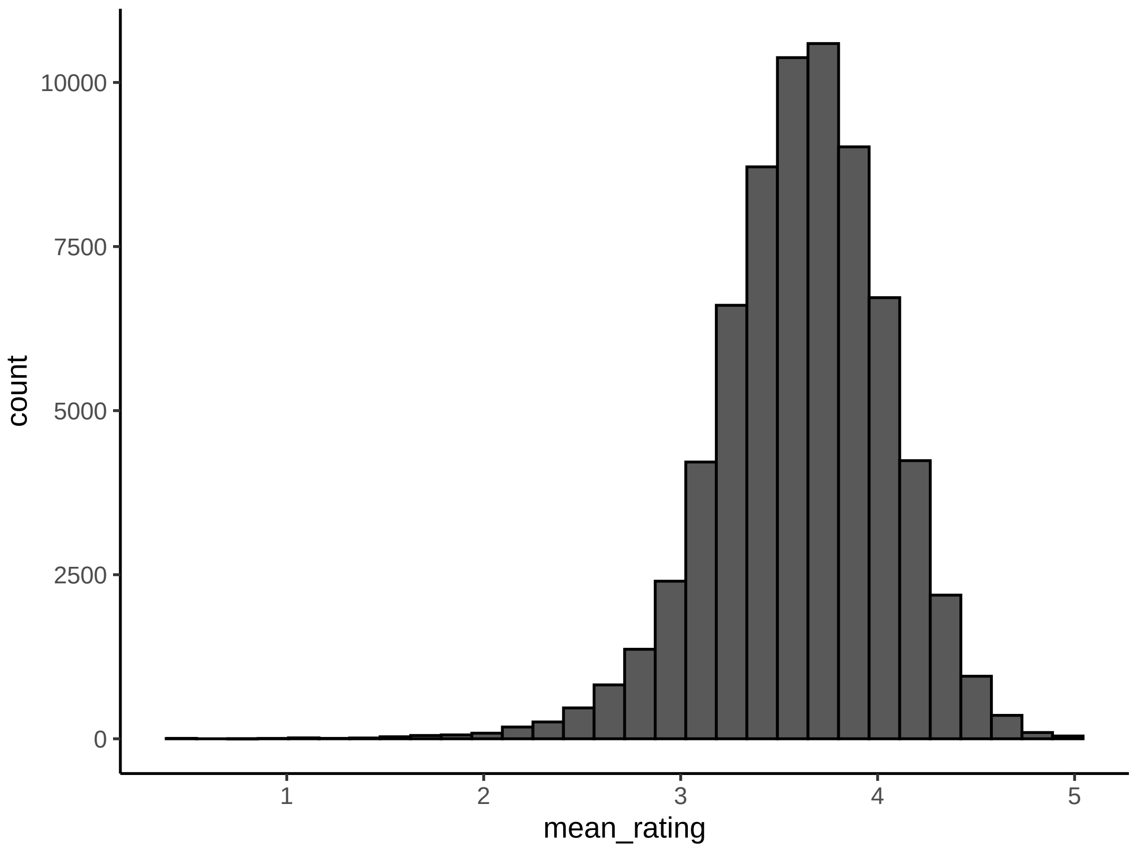
method	RMSE
Random value (2.5)	1.4658

method	RMSE
Using the average	1.0590
Movie Effect Model	0.9427

In a similar way, we can make use of the fact that not all users are the same. Some give high ratings, others give lower ratings.

To confirm, we group the train_set dataset by userId and compute the mean rating given by each user. We will use only those mean ratings where the users have given more than 100 ratings, to ensure that we employ this modification only for those users that have rated often enough to allow us to predict their tendencies. We plot a histogram using the ggplot2 library.

```r
### Improving by incorporating the fact that some users give great
# reviews, some do not. User effect model.
train_set %>%
  group_by(userId) %>%
  summarize(mean_rating = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(mean_rating)) +
  geom_histogram(bins = 30, color = "black") +
  theme_classic()
```



We can observe that most users give an average rating of around 3.5, but there is a significant variation in the ratings given by different users. In particular, there are many users that give very high and very low ratings.

We can use the average rating by a user as the expectation of the rating given by that user. But to do that we must first differentiate between the movie effect and the user effect. This can be done by subtracting the movie effect first, before computing the user effect.

To deploy this recommendation system, we join the movie_avgs dataset created earlier to the train_set dataset. In this way, for every movieId in the train_set dataset, we now have the average rating. We now group this file by userId and compute the variable b_u as the average of rating minus average rating and the average moving rating. In this way, for every userId, we have the value of difference from mean, while adjusting for the movie effect. This information is stored in user_avgs dataset.

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

To predict the movie ratings for the test_set dataset, we join the movie_avgs dataset by movieId and the user_avgs dataset by userId. The value of mu + b_i + b_u will then predict the movie rating for every movie in the test_set dataset while considering the movie effect and the user effect concomitantly. We store the predicted ratings in predicted_ratings dataset.

To test how good this recommendation system predicts the ratings of the test_set dataset, we compute the RMSE value using the RMSE function created earlier. We store the results in the data frame named rmse_results.

```
predicted_ratings <- test_set %>%
  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, test_set$rating)
rmse_results <- bind_rows(rmse_results,
  data_frame(method="Movie + User Effects Model",
  RMSE = model_2_rmse ))
```

```r
rmse_results %>% knitr::kable()
```

method	RMSE
Random value (2.5)	1.4658
Using the average	1.0590
Movie Effect Model	0.9427
Movie + User Effects Model	0.8646

We find that the RMSE value reduces a lot. We now repeat the process to create a new model, but now also including the genre effect together with the movie effect and the user effect. To test how good this recommendation system predicts the ratings of the test_set dataset, we compute the RMSE value using the RMSE function created earlier. We store the results in the data frame named rmse_results.

```r
### Movie + User + Genre effect model
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))


predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

model_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
  data_frame(method="Movie + User + Genre Effects",
  RMSE = model_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Random value (2.5)	1.4658
Using the average	1.0590
Movie Effect Model	0.9427
Movie + User Effects Model	0.8646
Movie + User + Genre Effects	0.8643

There is a slight reduction in RMSE value. This is expected because most of the genre effect has already been accounted for by the movie effect. Thus, the improvements over the previous model are small.

We now repeat the process to create a new model, but now also including the time effect together with the movie effect, the user effect, and the genre effect. To test how good this recommendation system predicts the ratings of the test_set dataset, we compute the RMSE value using the RMSE function created earlier. We store the results in the data frame named rmse_results.

```
### Movie + User + Genre + Time effect model
time_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(b_d = mean(rating - mu - b_i - b_u - b_g))


predicted_ratings <- test_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(time_avgs, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
  pull(pred)


model_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
  data_frame(method="Movie + User + Genre + Time Effects",
  RMSE = model_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Random value (2.5)	1.4658
Using the average	1.0590
Movie Effect Model	0.9427
Movie + User Effects Model	0.8646
Movie + User + Genre Effects	0.8643
Movie + User + Genre + Time Effects	0.8642

We observe that the RMSE value has reduced a bit further. Yet another approach is to employ regularisation. Regularisation is a technique to prevent the machine learning model from overfitting by adding some extra information to it. This constrains the total variability of the effect sizes. One way is through penalised least squares. We use a penalty term, lambda, that becomes negligible when there are very large number of samples, but becomes prominent when the number of samples is small. The equation for bias becomes:

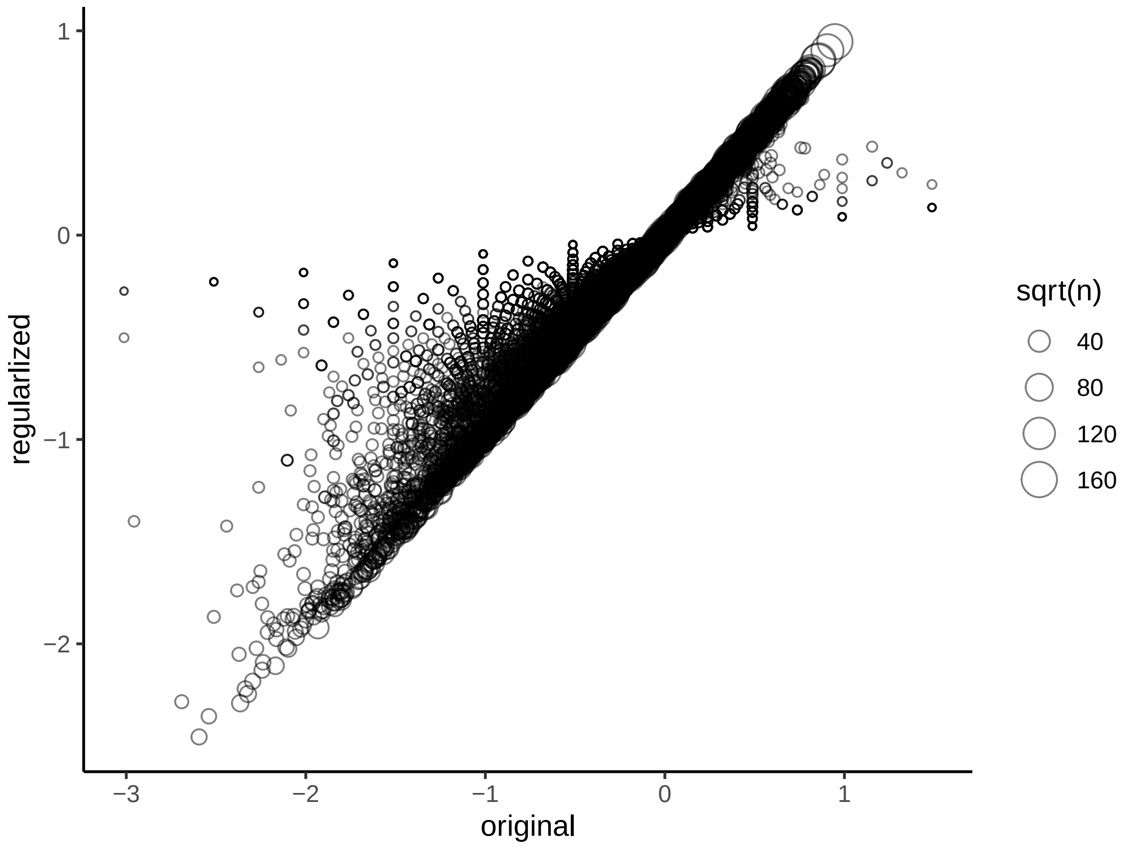
b_i = sum(rating - mu)/(n()+lambda)

When n() » lambda, lambda effectively becomes ignored in the equation. When n() « lambda, lambda becomes very prominent. We save the results in a dataset named movie_reg_avgs.

```
# Regularisation with constant lambda
lambda <- 10 # Test lambda
mu <- mean(train_set$rating)
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

To view the impact of regularisation, we can plot the regularised biases against original biases.

```
# Making plot of regularised estimates vs. original least square
# estimates
data_frame(original = movie_avgs$b_i,
           regularlized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularlized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5) +
  theme_classic()
```



We observe that for movies with a large number of ratings (depicted by larger circles), the regularised biases are the same as the original biases (they lie on the y = x line). But for the movies with a small number of ratings (depicted by smaller circles), the regularised estimates are less extremist - they tend towards the y = 0 line.

To predict the ratings in the test_set dataset, we join the test_set dataset with the movie_reg_avgs by movieId, and then make the prediction as the sum of the average, mu with the estimate of the regularised bias, b_i.

To test how good this recommendation system predicts the ratings of the test_set dataset, we compute the RMSE value using the RMSE function created earlier. We store the results in the data frame named rmse_results.

```
### Regularised movie effect model, lambda = 10
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
  data_frame(method="Reg. Movie Effect Model with lambda=10",
  RMSE = model_3_rmse ))
rmse_results %>% knitr::kable()
```

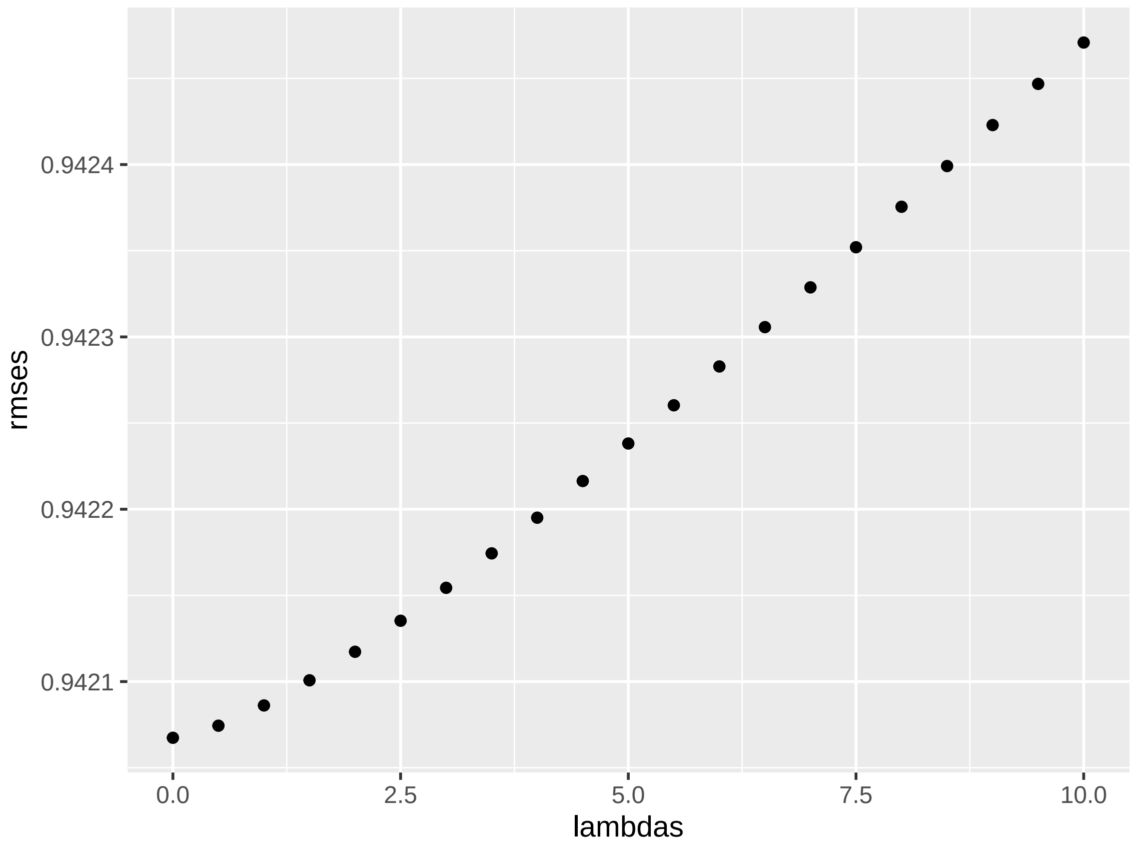
method	RMSE
Random value (2.5)	1.4658

method	RMSE
Using the average	1.0590
Movie Effect Model	0.9427
Movie + User Effects Model	0.8646
Movie + User + Genre Effects	0.8643
Movie + User + Genre + Time Effects	0.8642
Reg. Movie Effect Model with lambda=10	0.9428

In the previous technique, we had chosen the penalty parameter, lambda, randomly. But we can also choose a more optimal lambda. We do this by computing RMSEs with each of a sequence of lambdas, and choosing the lambda that gives the lowest value of RMSE. Since we should not touch the test_set dataset, and the train_set dataset is very large, we construct a new dataset cv_set for cross-validation. This is a small partition of the train_set dataset, constructed randomly for cross-validation purposes. The optimal value of lambda will be used for further computations.

```
### Regularised movie effect model
# Choosing optimal lambda, the penalty parameter
# CV (cross-validation) set will be 10% of train_set data
set.seed(1) # if using R 4.0 or later,
# use `set.seed(1, sample.kind="Rounding")`
test_index <- createDataPartition(y = train_set$rating,
                                  times = 1, p = 0.1, list = FALSE)
cv_set <- train_set[test_index,]
```

```r
# Optimal lambda from a sequence: get lowest rmse
lambdas <- seq(0, 10, 0.5)

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

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

  predicted_ratings <- cv_set %>%
    left_join(movie_avgs, by='movieId') %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

  return(RMSE(predicted_ratings, cv_set$rating))
})

qplot(lambdas, rmses)
```



```
l = lambdas[which.min(rmses)]
```

Finally, we compute the predictions based on this recommender system. To test how good this recommendation system predicts the ratings of the test_set dataset, we compute the RMSE value using the RMSE function created earlier. We store the results in the data frame named rmse_results.

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


movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))
```

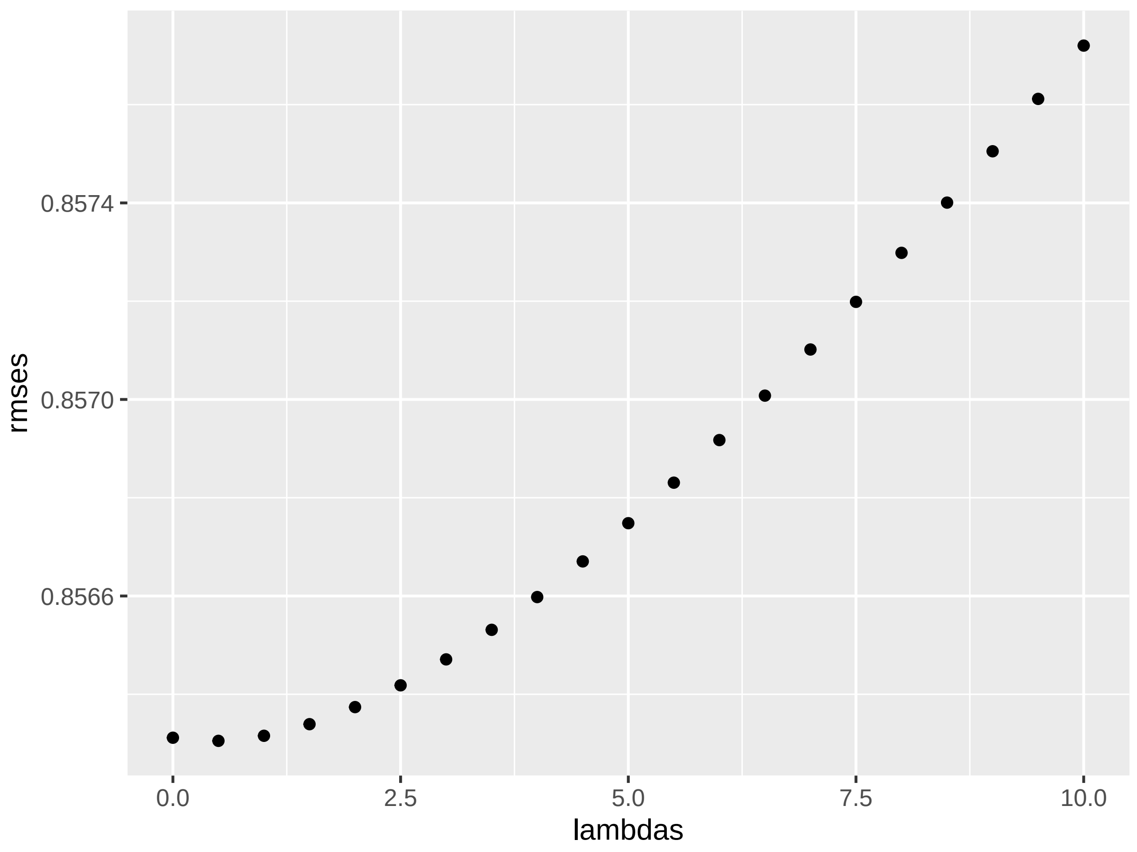
```r
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

model_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
  data_frame(method="Reg. Movie Effect",
  RMSE = model_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Random value (2.5)	1.4658
Using the average	1.0590
Movie Effect Model	0.9427
Movie + User Effects Model	0.8646
Movie + User + Genre Effects	0.8643
Movie + User + Genre + Time Effects	0.8642
Reg. Movie Effect Model with lambda=10	0.9428
Reg. Movie Effect	0.9427

Since the value of optimal lambda is small, there is no effect of regularisation on RMSE when compared to the movie effect without regularisation. This is expected since we have observed that with a value of lambda = 10, the RMSE value actually increases.

Let us do the regularisation for both the movie effect and the user effect. As before, we take a sequence of lambdas to select the one that gives the least RMSE value.

```
### Regularised movie + user effect model
# Optimal lambda from a sequence: get lowest rmse
lambdas <- seq(0, 10, 0.5)

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

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

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

  predicted_ratings <- cv_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  return(RMSE(predicted_ratings, cv_set$rating))
})

qplot(lambdas, rmses)
```



```
l = lambdas[which.min(rmses)]

mu <- mean(train_set$rating)

movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))
```

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

predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

model_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
  data_frame(method="Reg. Movie + User Effect",
  RMSE = model_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Random value (2.5)	1.4658
Using the average	1.0590
Movie Effect Model	0.9427
Movie + User Effects Model	0.8646
Movie + User + Genre Effects	0.8643

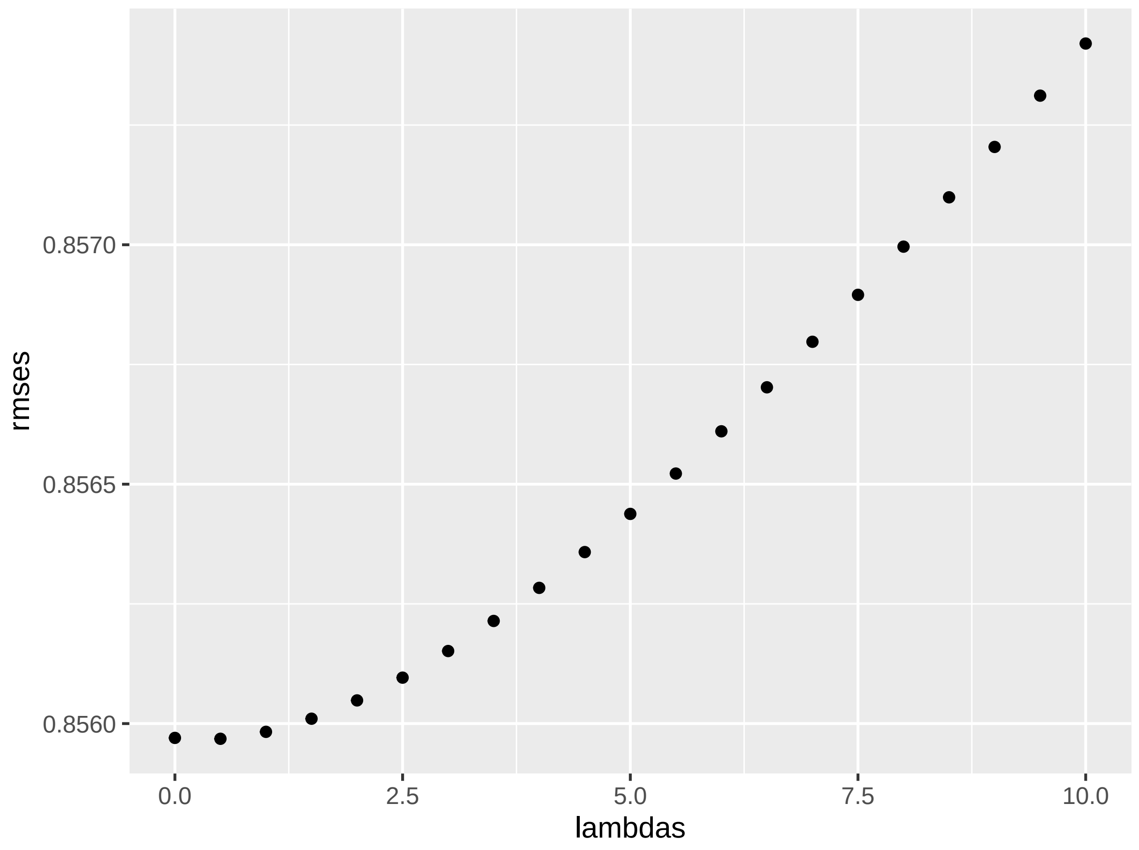
method	RMSE
Movie + User + Genre + Time Effects	0.8642
Reg. Movie Effect Model with lambda=10	0.9428
Reg. Movie Effect	0.9427
Reg. Movie + User Effect	0.8645

We observe that the value of lambda = 0.5 gives the lowest value of RMSE. We use this value of lambda to make the predictions about the test_set dataset. We store the results in the data frame named rmse_results.

We observe that there is a distinct improvement in the RMSE value over the regularisation on just the movie effect.

Let us do the regularisation for the movie effect, the user effect, and the genre effect. As before, we take a sequence of lambdas to select the one that gives the least RMSE value.

```
### Regularised movie + user + genre effect model
# Optimal lambda from a sequence: get lowest rmse
lambdas <- seq(0, 10, 0.5)

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

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

  genre_avgs <- train_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))

  predicted_ratings <- cv_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(genre_avgs, by='genres') %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

  return(RMSE(predicted_ratings, cv_set$rating))
})

qplot(lambdas, rmses)
```



```
l = lambdas[which.min(rmses)]


mu <- mean(train_set$rating)


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


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

```
genre_reg_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))

predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  left_join(genre_reg_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

model_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
  data_frame(method="Reg. Movie + User + Genre Effect",
  RMSE = model_rmse))
rmse_results %>% knitr::kable()
```

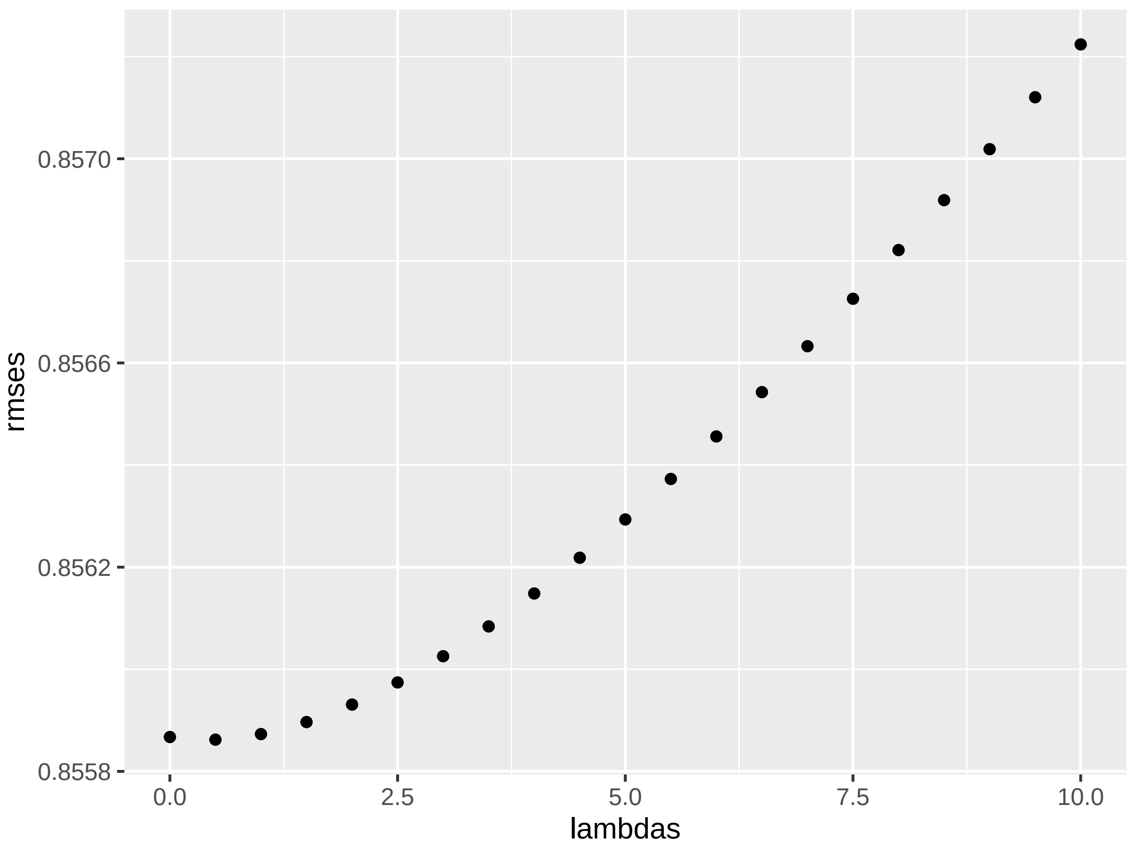
method	RMSE
Random value (2.5)	1.4658
Using the average	1.0590
Movie Effect Model	0.9427

method	RMSE
Movie + User Effects Model	0.8646
Movie + User + Genre Effects	0.8643
Movie + User + Genre + Time Effects	0.8642
Reg. Movie Effect Model with lambda=10	0.9428
Reg. Movie Effect	0.9427
Reg. Movie + User Effect	0.8645
Reg. Movie + User + Genre Effect	0.8641

We observe that there is a distinct, albeit small improvement in the RMSE value over the regularisation on just the movie + user effect.

Let us do the regularisation for the movie effect, the user effect, the genre effect, and the time effect. As before, we take a sequence of lambdas to select the one that gives the least RMSE value.

```
### Regularised movie + user + genre + time effect
### model
# Optimal lambda from a sequence: get lowest rmse
lambdas <- seq(0, 10, 0.5)

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

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

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

  genre_avgs <- train_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))

  time_avgs <- train_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(genre_avgs, by='genres') %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
    group_by(date) %>%
    summarize(b_d = sum(rating - mu - b_i - b_u - b_g)/(n()+l))

  predicted_ratings <- cv_set %>%
    mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(genre_avgs, by='genres') %>%
    left_join(time_avgs, by='date') %>%
    mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
    pull(pred)

  return(RMSE(predicted_ratings, cv_set$rating))
})

qplot(lambdas, rmses)
```



```
l = lambdas[which.min(rmses)]

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

```r
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))

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

genre_reg_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))

time_reg_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(b_d = sum(rating - mu - b_i - b_u - b_g)/(n()+l))

predicted_ratings <- test_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  left_join(genre_reg_avgs, by='genres') %>%
  left_join(time_reg_avgs, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
  pull(pred)

model_rmse <- RMSE(predicted_ratings, test_set$rating)


rmse_results <- bind_rows(rmse_results,
  data_frame(method="Reg. Movie + User + Genre + Time Effect",
  RMSE = model_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Random value (2.5)	1.4658
Using the average	1.0590
Movie Effect Model	0.9427
Movie + User Effects Model	0.8646
Movie + User + Genre Effects	0.8643
Movie + User + Genre + Time Effects	0.8642
Reg. Movie Effect Model with lambda=10	0.9428
Reg. Movie Effect	0.9427
Reg. Movie + User Effect	0.8645

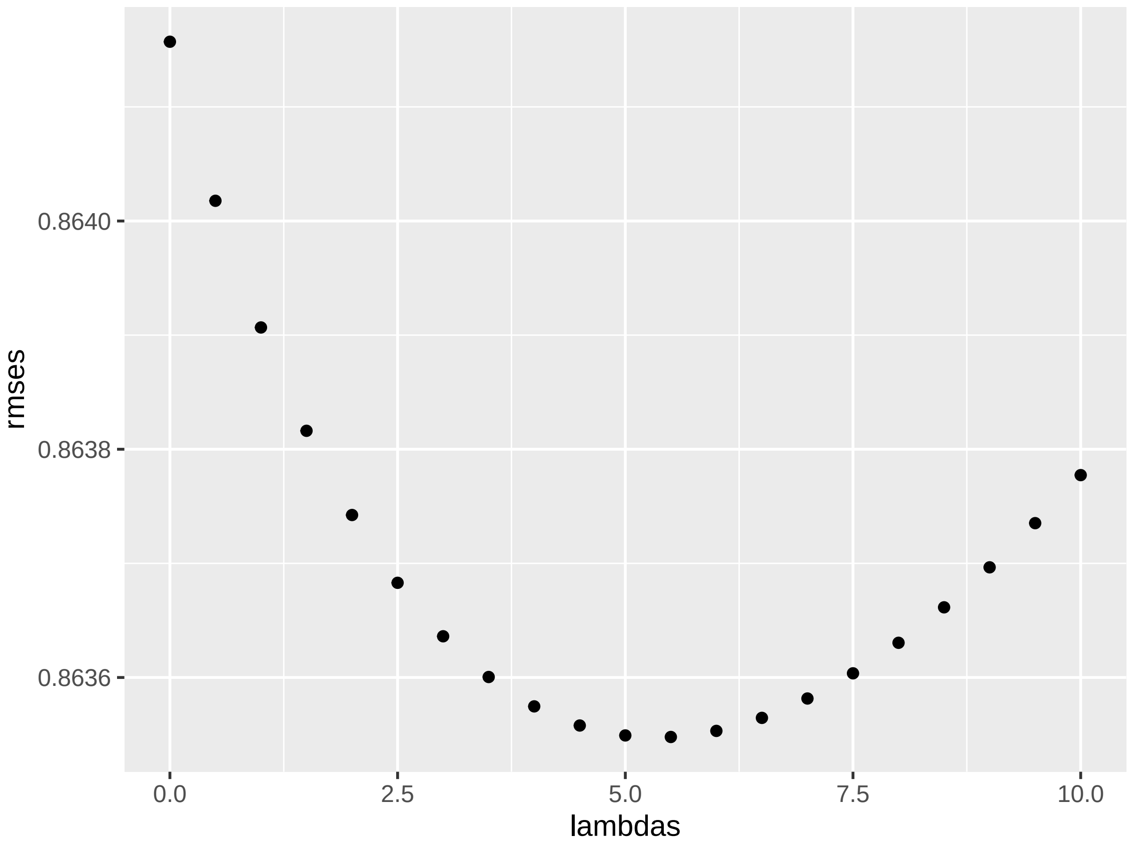
method	RMSE
Reg. Movie + User + Genre Effect	0.8641
Reg. Movie + User + Genre + Time Effect	0.8640

Validation

We note that the algorithm accounting for regularised movie + user + genre + time effects gives the lowest value of RMSE. Thus, we choose this algorithm as the algorithm of choice. This algorithm will now be validated against the validation dataset. Since the validation dataset is the out-sample now, we train our final model over the whole of the train_set dataset and optimise it using the test_set dataset.

```r
### Validation
# Optimal lambda from a sequence: get lowest rmse
# Here we use all of train_set dataset to train
# and all of test_set dataset to optimise lambda
lambdas <- seq(0, 10, 0.5)

rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

  movie_avgs <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
```

```r
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n()+1))

genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+1))

time_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(b_d = sum(rating - mu - b_i - b_u - b_g)/(n()+1))

predicted_ratings <- test_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(time_avgs, by='date') %>%
    mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```



```
l = lambdas[which.min(rmses)]

mu <- mean(train_set$rating)

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

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

genre_reg_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - b_i - b_u)/(n()+l))

time_reg_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(b_d = sum(rating - mu - b_i - b_u - b_g)/(n()+l))

predicted_ratings <- validation %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  left_join(genre_reg_avgs, by='genres') %>%
  left_join(time_reg_avgs, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
  pull(pred)

model_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- bind_rows(rmse_results,
  data_frame(method="Validation",
  RMSE = model_rmse))
```

We store the results in the data frame named rmse_results.

RMSE

The RMSE values computed till now are displayed below:

```
rmse_results %>% knitr::kable()
```

method	RMSE
Random value (2.5)	1.4658
Using the average	1.0590
Movie Effect Model	0.9427
Movie + User Effects Model	0.8646
Movie + User + Genre Effects	0.8643

method	RMSE
Movie + User + Genre + Time Effects	0.8642
Reg. Movie Effect Model with lambda=10	0.9428
Reg. Movie Effect	0.9427
Reg. Movie + User Effect	0.8645
Reg. Movie + User + Genre Effect	0.8641
Reg. Movie + User + Genre + Time Effect	0.8640
Validation	0.8647

We note that the machine learning algorithm utilising regularised Movie + User + Genre + Time effects model is able to provide quite accurate predictions. The root mean square error between the predicted and the actual ratings in the validation dataset is 0.8647, lower than the target value of 0.8649 for this assignment!

method	RMSE
Validation	0.8647

Housekeeping

We clear the memory using the rm(list=ls()) command.

```
rm(list=ls())
```

Conclusion

Summary

The objective of this project was to create a movie recommendation system using the 10M version of the MovieLens dataset, using all the tools shown throughout the courses in this series. Particularly, the aim was to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set. The goodness of the prediction was to be judged by the root mean square errors between the prediction and the actual ratings in the validation dataset. In particular, the target was to have an RMSE below 0.8649.

To this end, we divided the 10M version of the MovieLens dataset into two portions: the edx dataset, which was used to train and test various machine learning algorithms, and the validation dataset, which was to be used in the very end for the final evaluation of the exercise. The edx dataset was further divided into train_set and test_set datasets, to be used for training and testing various machine learning algorithms and procedures.

In this project, we employed the methods of

1. using a random rating as the predicted rating
2. using the average rating as the predicted rating
3. employing the movie effects, to account for the fact that some movies are more highly rated than others
4. employing the movie + user effects, to account for the fact that some movies are more highly rated than others, and some users give better ratings than others

5. employing the movie + user + genre effects, to account for the fact that some movies are more highly rated than others, some users give better ratings than others, and some genres have better ratings than others

6. employing the movie + user + genre + time effects, to account for the fact that some movies are more highly rated than others, some users give better ratings than others, some genres have better ratings than others, and movies released at different times have different ratings

7. using regularisation and penalised least squares with set penalty on movie effects

8. using regularisation and penalised least squares with a penalty decided by data on movie effects

9. using regularisation and penalised least squares with a penalty decided by data on movie and user effects

10. using regularisation and penalised least squares with a penalty decided by data on movie, user, and genre effects

11. using regularisation and penalised least squares with a penalty decided by data on movie, user, genre, and time effects

Next, using the RMSE values as an indicator of the strength of the model, we chose the regularised movie + user + genre + time effects model (as it gave the least RMSE). This selected model was then used on the validation dataset. The obtained RMSE on comparing the predictions on the validation dataset with the acutal ratings in the validation dataset was 0.8647, lower than the target RMSE of 0.8649.

In this way, the objectives of the project were successfully met.

Potential impact

Recommender systems have a huge impact due to their ability to suggest relevant items to the users. Thus, for instance, the recommender system can be used to predict the kind of rating that I would give a movie, if I watched and rated it. It is important to note here that this prediction is being given based on the characteristics of the movie and my own preferences / characteristics.

Now if we have the information about the kinds of movies that I would rate highly if I watched them, I could use this information to selectively watch those movies. Since I would rate them highly, I would also enjoy them more. In this way, the recommender system is providing me with a benefit.

On the other hand, if the streaming service or the DVD seller had this information, they could use this information to ensure that I am glued to their system / services, by selectively providing me those movies that I like a lot. This would add to their profits, and thus would be beneficial to them as well.

Thus, a potential impact of a movie recommender system is increase in the consumer and producer surplus concomitantly. By doing this, the movie recommender system increases the utility in the society. And that too at a very little cost (some time, some computing power, some electricity)! It should come as no surprise that today recommender systems are ubiquitous in their presence and popularity. We observe recommendations (and pretty good ones!) being provided by a plethora of services including Amazon, Netflix, Youtube, etc.

It could, however, also be argued that precisely because the recommender systems are able to provide very good recommendations to suit the tastes of the viewers, we observe several viewers literally glued to their screens, binge watching several movies in a stretch. This could play a role in reducing the labour supply in the economy (since it makes leisure more enjoyable cf. consumption), which could have several cascading microeconomic and macroeconomic impacts. This is something that would require further investigation.

Limitations

While the recommender systems currently account for the fact the different movies are different and different users are different, they also leave certain other potential influencers on movie ratings. For instance, Behavioural Economics suggests that choices may be influenced by the time of the day that the user is watching the movie, whether the movie is being watched alone or in the company of friends, whether food is available or not, and so on. These, and several other factors, may be incorporated in more detailed future models.

Similarly, so far we have assumed that users have fixed preferences. There are some users that love action movies and hate romance, while there are others that love romantic movies and hate thrillers. But the reality is that users do not have fixed preferences. Preferences change with time, age, and maturity. Preferences change when a users becomes bored after watching several movies of the loved genres. Preferences change when someone is in love. The present recommender system does not incorporate changing user preferences. This is something that may be incorporated in more detailed future models.

The recommender systems are also plagued by a scarcity of data. For a user that has abundantly revealed their preferences through either rating or selecting (or clicking) several different movies and watched them for different amounts of time, the recommender system will be able to come up with very good recommendations. However, for a relatively newer user on the platform, there might not be enough data available regarding their preferences to come up with a good recommendation.

Even for a user that has used the system for a very long period of time, there might not be sufficient amount of relevant data. For example, a user that likes action and thriller movies may have watched and rated hundreds of action and thriller movies, and the recommender system may be in a position to provide very good recommendations of action and thriller movies. But it is also possible that the same user may also love animes, it is just that they have not yet been exposed to animes. Thus the user has not watched and rated even a single anime. In this case, the recommender system may not suggest any anime movie. This is something that may be incorporated in more detailed future models.

Another related issue is that the data changes with time. While we have lots of data regarding movies released in the 1990s, we have comparatively much less data for a movie that was released yesterday. The recommender system must be flexible and agile enough to incorporate changes as newer movies get added to the dataset.

The recommender systems also face issues when we have items that are highly unpredictable. There are some movies that are either loved greatly, or hated greatly. If we took a mean of the ratings, the recommender system will suggest that they are average movies, while in fact some users may love (or hate) those movies to a much greater extent than any other average movie. This is something that may be incorporated in more detailed future models.

Future work

The future work can be on several lines:

1. Improving the algorithm(s): We can incorporate the behavioural aspects of consumers to improve the algorithms. In particular, we can try to incorporate the social and emotional aspects of consumers when they perform the rating and when they are to be recommended new movies.

2. Improving the availability of data: While this project worked only on the the data available in the MovieLens dataset, we can try to simultaneously incorporate the data available in multiple datasets to increase data density and redundancy.

3. Improving the 'defaults': We can work on the rating to be given to newly released movies, and on ratings to be given to genres that the consumers don't watch, and may not have been exposed to.

4. Exploring the expansion of recommender systems: Currently recommender systems have successfully been deployed for movies and products. We can explore the possibilities of their usage in other domains as well.

5. Studying the impacts of recommender systems on economic surplus and labour availability. This point has been touched upon in the Impacts section. We can study the impacts as a future work.