Capstone project

Sarah Tomori

07/12/2019

1. Overview

1.1 Introduction

The objective of this project is to create an algorithm that best predicts the ratings of movies by Netflix users. For this purpose, the MovieLens 10M data is used, which involves 10 million ratings of movies.

Besides predicting movie ratings, a recommendation system is meant for predicting the preference of a user in various other scenarios as well such as clothing, books, music, social media accounts to follow and more. Regardless of which category the recommendation system falls in, it is generally meant to improve the user experience and in many cases enhance sales or the general use of the respective product or item by increasing user interaction.

Recommendation systems are used by well-known companies such as Amazon, Netflix, YouTube, Facebook and many more. It's important to note that it does require a large amount of data and the right set of AI skills in order to build an optimal, personalised algorithm. A number of different recommendation systems exist, here among:

- Content-based filtering. Recommends products with similarities of other products.
- Collaborative filtering. Makes its predictions based on what might be of interest to the person based on the preferences of others - if person A and person B like the same product, then that could suggest that if person A likes another product, so would person B.
- Hybrid filtering. With hybrid filtering, a combination of different recommendation systems is used.
- Knowledge based filtering. This type of recommendation system suggests items based on retained information on the needs and wants of the user. This knowledge is used to relate these needs of the user to a sufficient product.
- Utility-based recommender system. Here, recommendations are made based on the utility of each object for users.
- Demographic based recommender system. This recommendation system builds recommendations based on demographic data on the users. Categories are created based on these details on this demographic data and thus used to recommend a product, movie, book or something else.

1.2 Purpose of the project

In this project, different machine learning algorithms will be tested to discover, which one does the best job at predicting Netflix' movie ratings. For this purpose, the dataset is first split into a training- and validation set, which consists of 10% of the dataset. In order to test the intermediary models, the training set (the so called edx set) is split further into a training- and test set, where the test set is 10% of the original training set.

In order to determine the best model, RMSE will be used as the determining parameter. The model that produces the lowest value of RMSE will be the final model and rerun on the validation set, which corresponds to the data that hasn't yet been touched by our model prior to this.

2. Method and analysis

In this section, the MovieLens dataset is explored with the help of different visualisation tools and data cleansing. This is an important step prior to our analysis, as it helps us understand the structure of the data and minimize the risk of disturbances that could bias results when training the model to be used for predicting movie predictions.

2.1 Preparing the data

The necessary packages are installed and libraries are loaded.

As mentioned, we use the MovieLens 10M data, which contains 10 million ratings. As we need to retain 10% of the data for the final prediction model. Thus, the data needs to be split into a training set (the edx set) and a validation set. Note that in many instances in the real world, the definition of a validation set is referred to as the test set and is what is used to run the final model. In this project, the test set will, however, be a subset of the training set created and used to test the intermediary models. This will be addressed later, in the analysis section of the project. Now, the training- and validation sets are created.

```
################################
# Create edx set, validation set
################################
# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse

## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang

## -- Attaching packages --------------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.1     v purrr   0.3.2
## v tibble  2.1.2     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
```

```
##
##     transpose
```

The next step is to download the MovieLens dataset.

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

# MovieLens dataset is downloaded.
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
title = as.character(title),
genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

Validation set is created, which consists of 10% of the MovieLens data.

```
# Validation set is created, which consists of 10% of the MovieLens data.
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

We need to make sure that userId and movieId in the validation set are present in the edX set as well.

```
# We make sure that userId and movieId in the validation set are present in the edX set as well.
# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

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

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

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

2.2 Data and exploration

2.2.1 Overview

We are going to explore what kind of class our dataset falls into. Based in the class() function in RStudio, this is shown below. We can see that the data is a data frame.

2.2.2 Data exploration

```
#Check class of data
class(edx)
```

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

When using the glimpse() function, it shows that there are 6 variables (features) in the edX dataset and 9,000,055 observations. The outcome variable, y, which we are trying to predict is the rating. Each rating represents the movie rating by one user for the respective movie.

```
#Get brief overview of the data
glimpse(edx)
```

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|D...
```

Below a summary of the edx dataset is presented.

```
#See a summary of the data
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title               genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Based on this summary, it shows that the minimum rating is 0.5 and the maximum is 5.

Next, we show that there are 10 different unique values of movie ratings.

```
#Show unique values of movie ratings
unique(edx$rating)
```

```
##  [1] 5.0 3.0 2.0 4.0 4.5 3.5 1.0 1.5 2.5 0.5
```

```
#Show number of users, movies and genres
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId), n_genres = n_distinct(g
```

```
##   n_users n_movies n_genres
## 1   69878    10677      797
```

```
#Check whether there are any missing values
anyNA(edx)
```

```
## [1] FALSE
```

2.3 Data cleaning

Based on the column containing the movie title, which we saw in the previous section, the release year needs to be separated from the title. The year that the movie was released could be of relevance. This is done below:

```
#Separate release year of the movie from the movie title
edx <- edx %>% mutate(releaseyear = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"), regex
```

To confirm that release year has actually been separated from the title column, a brief overview of the edx dataset is again printed:

```
#Confirm release year has been removed
head(edx)
```

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

It is necessary to convert timestamp to a date-format, so this feature can be used to calculate the age of the movie.

```
#Convert timestamp to date-format
edx <- edx %>% mutate(timestamp = as.POSIXct(timestamp, origin = "1970-01-01", tz = "GMT"))

edx$timestamp <- format(edx$timestamp, "%Y")
```

It now shows that timestamp has been converted into numerical values of the year instead.

```
#Show that this has been done
head(edx)
```

```
##   userId movieId rating timestamp                   title
## 1      1     122      5      1996               Boomerang
## 2      1     185      5      1996                Net, The
## 3      1     292      5      1996                Outbreak
## 4      1     316      5      1996                Stargate
## 5      1     329      5      1996 Star Trek: Generations
## 6      1     355      5      1996        Flintstones, The
##                           genres releaseyear
## 1                 Comedy|Romance        1992
## 2          Action|Crime|Thriller        1995
## 3  Action|Drama|Sci-Fi|Thriller        1995
## 4        Action|Adventure|Sci-Fi        1994
## 5 Action|Adventure|Drama|Sci-Fi        1994
## 6       Children|Comedy|Fantasy        1994
```

The timestamp variable is renamed and is now called “ratingyear”.

```
#Rename the timestamp variable
names(edx)[4] <- "ratingyear"
names(edx)
```

```
## [1] "userId"      "movieId"     "rating"      "ratingyear"  "title"
## [6] "genres"      "releaseyear"
```

The release year is separated from the movie title in the validation set as well.

```
#Separate release year from movie title in the validation set
validation <- validation %>% mutate(releaseyear = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}
```

The timestamp variable that contains the code for the date of the rating is also renamed to “ratingyear”.

```
#Rename timestamp variable with year of rating to rating year
names(validation)[4] <- "ratingyear"
```

The validation set is equally changed so timestamp is converted to the year of the rating and the year of release is extracted from the movie title.

```
#Convert timestamp variable in the validation set to date format
validation <- validation %>% mutate(ratingyear = as.POSIXct(ratingyear, origin = "1970-01-01", tz = "GM
validation$ratingyear <- format(validation$ratingyear, "%Y")
```

The two datasets are compared. The edx dataset:

```
#Show both datasets again for comparability
head(edx)
```

```
##   userId movieId rating ratingyear                   title
## 1      1     122      5       1996               Boomerang
## 2      1     185      5       1996                Net, The
## 3      1     292      5       1996                Outbreak
## 4      1     316      5       1996                Stargate
## 5      1     329      5       1996 Star Trek: Generations
## 6      1     355      5       1996        Flintstones, The
##                          genres releaseyear
## 1                Comedy|Romance        1992
## 2         Action|Crime|Thriller        1995
## 3  Action|Drama|Sci-Fi|Thriller        1995
## 4       Action|Adventure|Sci-Fi        1994
## 5 Action|Adventure|Drama|Sci-Fi        1994
## 6       Children|Comedy|Fantasy        1994
```

The validation set:

```
head(validation)
```

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

We can see that the training- and the validation sets now contain similar variables.

In order to create a feature with the age of the movie and the time between the release year and the rating year of the movie, we do so in the next step. As the rating year is a character value, it needs to be converted to a numeric one.

```
#Create age of movie feature
edx <- edx %>% mutate(movieage = 2019 - releaseyear, release_rating_range = as.numeric(ratingyear) - rel
```

We can see how it looks below:

```
#Show data
head(edx)
```

```
##   userId movieId rating ratingyear                   title
## 1      1     122      5       1996               Boomerang
## 2      1     185      5       1996                Net, The
## 3      1     292      5       1996                Outbreak
## 4      1     316      5       1996                Stargate
## 5      1     329      5       1996 Star Trek: Generations
## 6      1     355      5       1996        Flintstones, The
##                          genres releaseyear movieage release_rating_range
## 1                Comedy|Romance        1992       27                    4
## 2         Action|Crime|Thriller        1995       24                    1
## 3  Action|Drama|Sci-Fi|Thriller        1995       24                    1
## 4       Action|Adventure|Sci-Fi        1994       25                    2
## 5 Action|Adventure|Drama|Sci-Fi        1994       25                    2
## 6       Children|Comedy|Fantasy        1994       25                    2
```

The same is done for the validation year. Due to space, the output of the identical procedure of the validation set will not be continuosly printed but the code can be found below and in the script:

```
#Repeat for the validation set
validation <- validation %>% mutate(movieage = 2019 - releaseyear, release_rating_range = as.numeric(ra
```

```
#Show the data
head(validation)
```

```
##   userId movieId rating ratingyear
## 1      1     231      5       1996
## 2      1     480      5       1996
## 3      1     586      5       1996
## 4      2     151      3       1997
## 5      2     858      2       1997
## 6      2    1544      3       1997
##                                              title
## 1                                    Dumb & Dumber
## 2                                    Jurassic Park
## 3                                       Home Alone
## 4                                          Rob Roy
## 5                                   Godfather, The
## 6 Lost World: Jurassic Park, The (Jurassic Park 2)
##                                  genres releaseyear movieage
## 1                                Comedy        1994       25
## 2        Action|Adventure|Sci-Fi|Thriller        1993       26
## 3                       Children|Comedy        1990       29
## 4                Action|Drama|Romance|War        1995       24
## 5                           Crime|Drama        1972       47
## 6 Action|Adventure|Horror|Sci-Fi|Thriller        1997       22
##   release_rating_range
## 1                    2
## 2                    3
## 3                    6
## 4                    2
## 5                   25
## 6                    0
```

Another step is to separate genres from each other, as many of the movies consist of multiple genres.

```
#Create a edx dataset that includes the split genres
edx_gsplit  <- edx  %>% separate_rows(genres, sep = "\\|")

#Do the same for the validation set
val_gsplit <- validation %>% separate_rows(genres, sep = "\\|")
```

```
#Create dataset with separated individual genres and relevant variables
genre_data <- edx %>% separate_rows(genres, sep = "\\|") %>% group_by(genres) %>% summarize(n_ratings =
genre_data
```

```
## # A tibble: 20 x 5
##    genres             n_ratings ratings_mean n_users n_movies
##    <chr>                  <int>        <dbl>   <int>    <int>
##  1 (no genres listed)         7         3.64       7        1
##  2 Action               2560545         3.42   69607     1473
##  3 Adventure            1908892         3.49   69521     1025
##  4 Animation             467168         3.60   59018      286
##  5 Children              737994         3.42   64059      528
##  6 Comedy               3540930         3.44   69864     3703
##  7 Crime                1327715         3.67   68691     1117
##  8 Documentary            93066         3.78   24295      481
##  9 Drama                3910127         3.67   69866     5336
## 10 Fantasy               925637         3.50   66833      543
## 11 Film-Noir             118541         4.01   31270      148
## 12 Horror                691485         3.27   60695     1013
## 13 IMAX                    8181         3.77    6393       29
## 14 Musical               433080         3.56   58918      436
## 15 Mystery               568332         3.68   61845      509
## 16 Romance              1712100         3.55   69530     1685
## 17 Sci-Fi               1341183         3.40   68469      754
## 18 Thriller             2325899         3.51   69567     1705
## 19 War                   511147         3.78   64892      510
## 20 Western                 189394         3.56   47648      275
```

```
#Do the same for the validation set
genre_data_val <- validation %>% separate_rows(genres, sep = "\\|") %>% group_by(genres) %>% summarize(
```

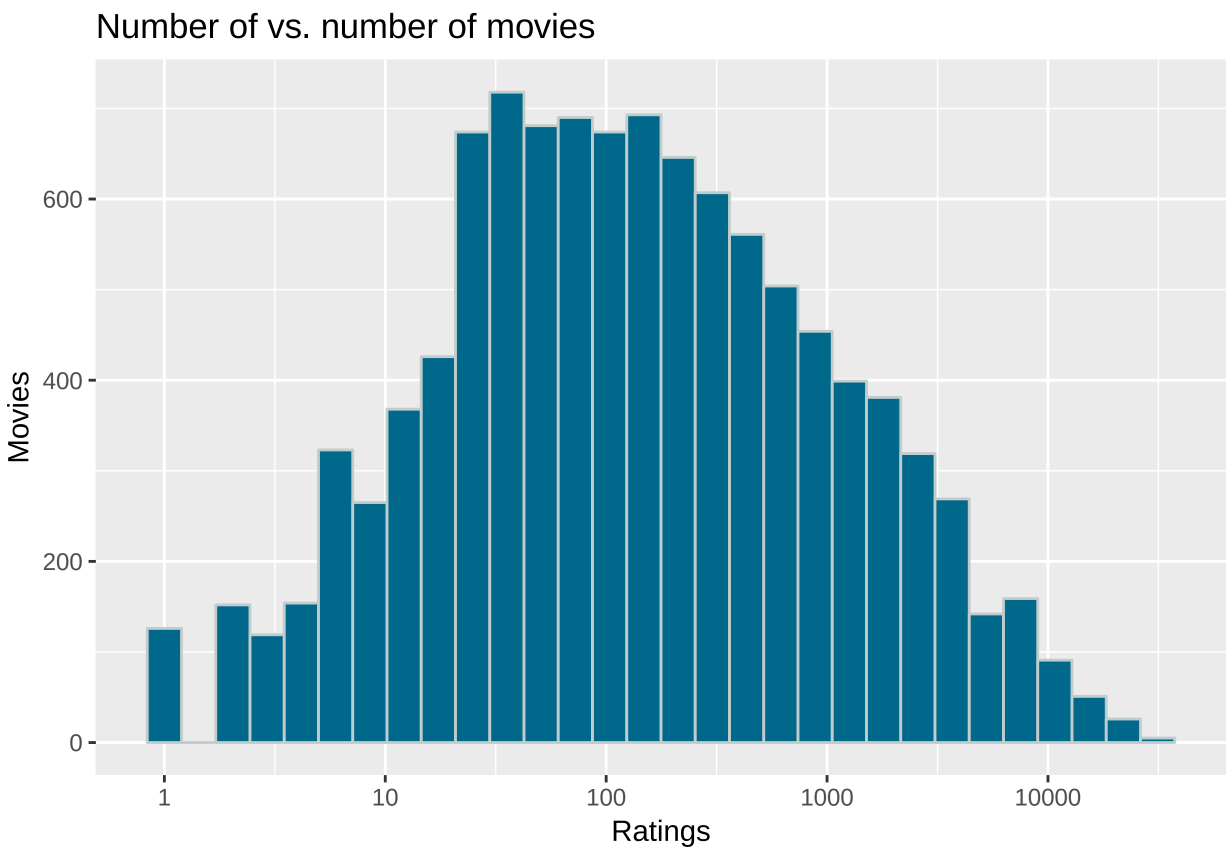
2.4 Visualisation of the data

In this section, a visualisation of the data is presented with the purpose of getting a better comprehension of the data structure.

```
#Visualisation of the data
```

2.4.1 Number of movie ratings

```
#Distribution of ratings in the training set
edx %>% count(movieId) %>%
ggplot(aes(n)) + geom_histogram(fill = "deepskyblue4", color = "azure3", bins = 30) +
scale_x_log10() + ggtitle("Number of vs. number of movies") + xlab("Ratings") + ylab("Movies")
```

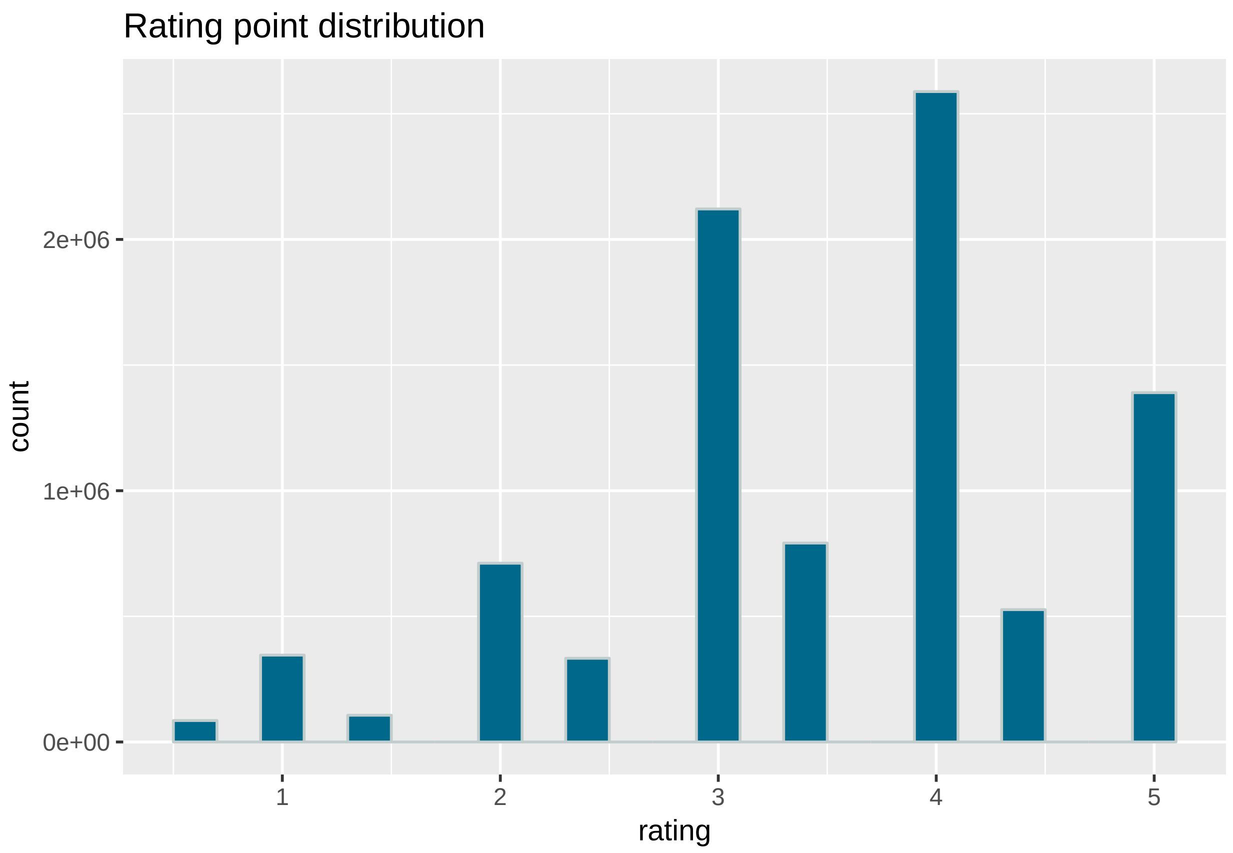


The above histogram shows how the movie ratings are distributed in the training set. Based on this, we can see that some movies have many ratings (over 600) while others have only been rated a few times. It is important to bear this in mind when training the model, as movies who were only rated a few times are likely to predict unreliable results.

2.4.2 Overview of distribution of the single ratings

Now, in order to get an idea of how each of the rating points are distributed in the dataset, we create another bar chart.

```
#Check distributions of ratings in the training set
edx %>% ggplot(aes(rating)) + geom_histogram(bins = 30, binwidth=0.2, color = "azure3", fill = "deepsky
ggtitle("Rating point distribution")
```



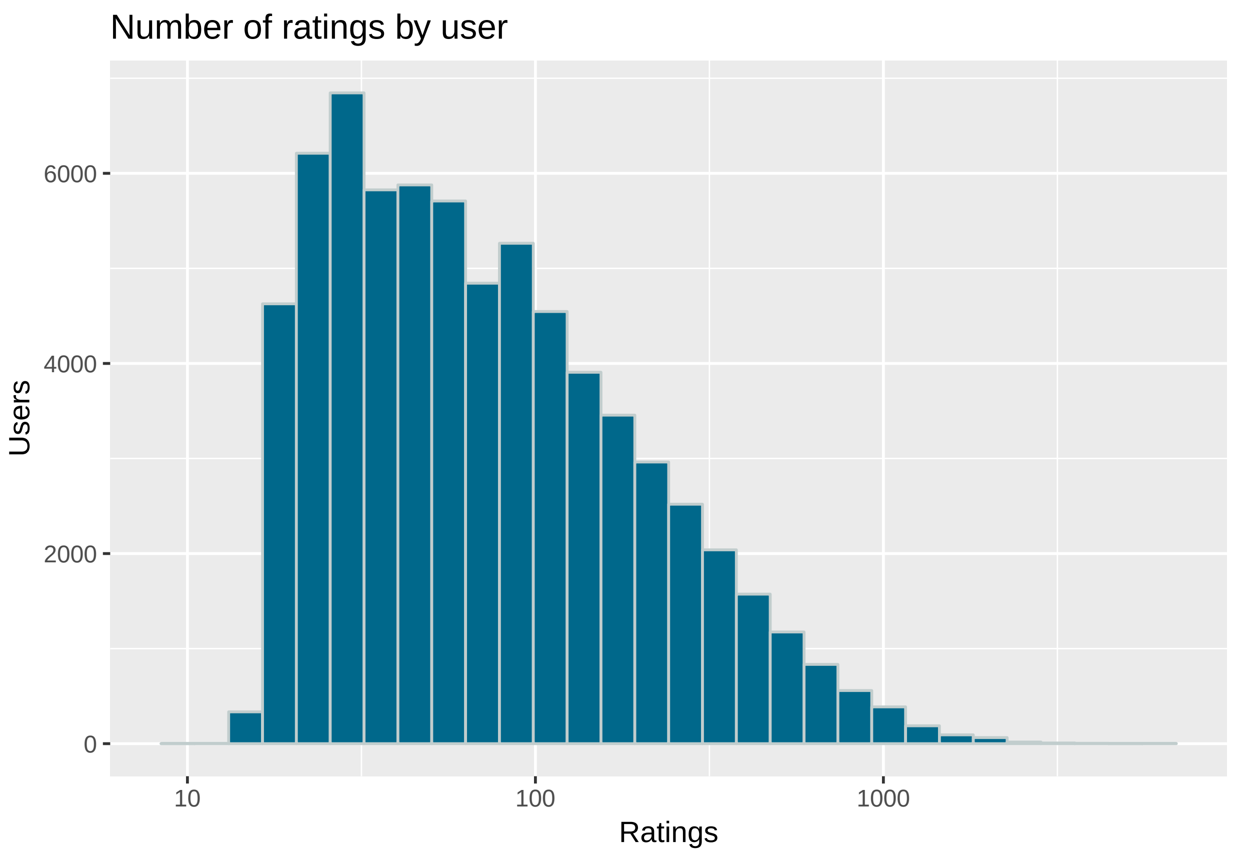
The distribution shows that 4 is the most common rating, followed by 3.5. The least common rating is 0.5 followed by 1.5. This indicates that users are typically quite generous with their ratings. It's important to note that the individual user could also have different standards as regards to what they consider a great movie; to some users, a 4-rating is considered just alright, whereas another user might rate a movie 4 only if it's extremely great. Meanwhile, this type of user would give 5-rating, only if it's really outstanding and/or perfect in their eyes.

2.4.3 Number of ratings by user

It is also important to look at the number of ratings respective to the number of users, as some users might only have rated very few movies. Similarly to movies with a few ratings, this is also something that might disrupt results.

To get an idea of this, a histogram with the distribution of the number of users in the dataset is shown next.

```
#Number of ratings by user
edx %>% count(userId) %>% ggplot(aes(n)) + geom_histogram(bins = 30, fill = "deepskyblue4", color = "azu
```



The histogram above gives an overview of the distribution of the movie ratings given by users. We see that some users have rated many movies, while others have only rated a few. This could lead to user bias, since it is hard to rely on a user's ratings if they have only rated one or two movies, for example. Another disturbance could be if a user is either overly pessimistic or extremely optimistic, meaning that they rate all their movies to be excellent (a lot of 5-ratings) or very poor (a lot of 1-ratings).

For this reason, regularization can be a way of penalizing the movie- and user effects to control the variability caused by these factors. This will be addressed later in the project.

2.4.4 Average user rating

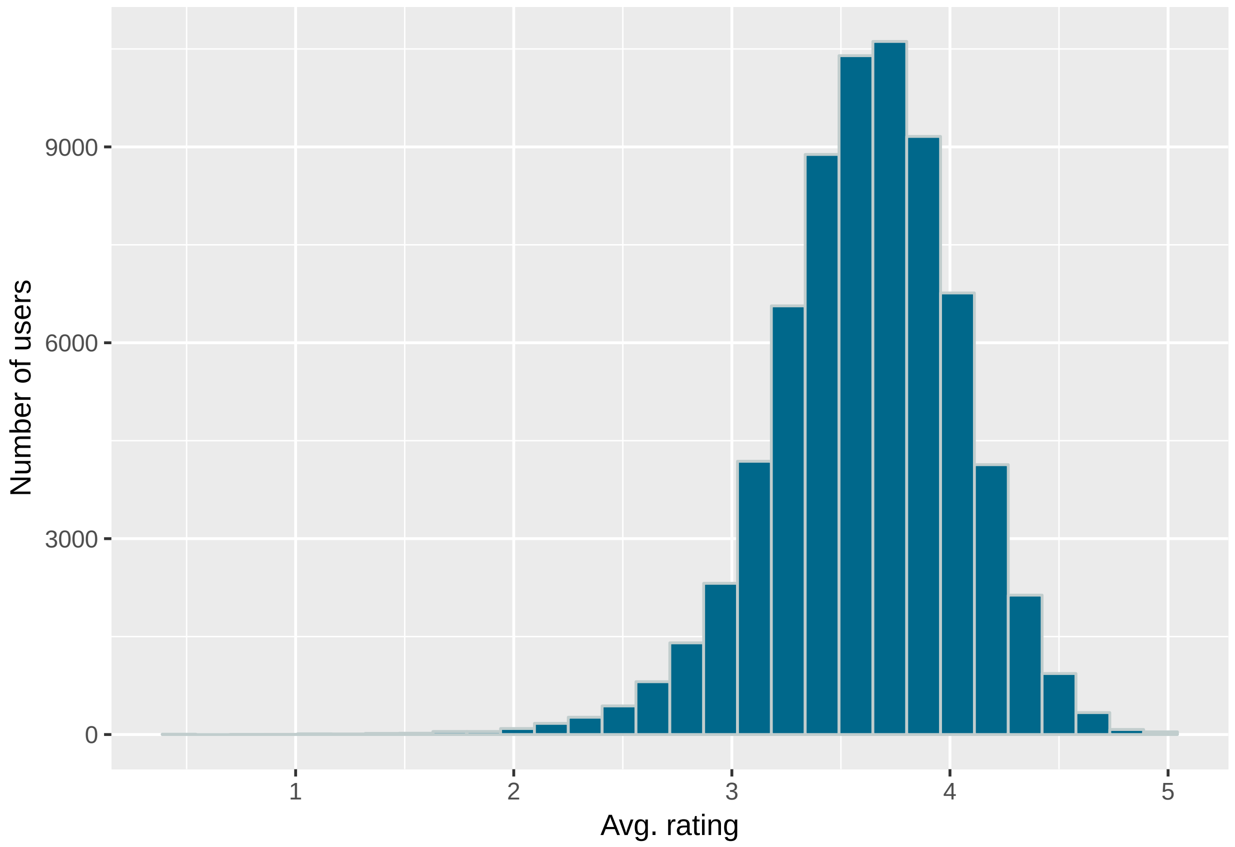
Next we create a column that shows the users' average rating.

```
#Create column with average user rating
edx <- edx %>% group_by(userId) %>% mutate(avg_user_rating = mean(rating))
head(edx)
```

```
## # A tibble: 6 x 10
## # Groups:   userId [1]
##   userId movieId rating ratingyear title genres releaseyear movieage
##    <int>    <dbl>  <dbl> <chr>      <chr> <chr>        <dbl>    <dbl>
## 1      1     122      5 1996       "Boo~ Comed~        1992       27
## 2      1     185      5 1996       "Net~ Actio~        1995       24
## 3      1     292      5 1996       "Out~ Actio~        1995       24
## 4      1     316      5 1996       "Sta~ Actio~        1994       25
## 5      1     329      5 1996       "Sta~ Actio~        1994       25
## 6      1     355      5 1996       "Fli~ Child~        1994       25
## # ... with 2 more variables: release_rating_range <dbl>,
## #   avg_user_rating <dbl>
```

Let's look at the distribution through a histogram of the users' average movie ratings.

```
#Show distribution of the average rating of users
edx %>% group_by(userId) %>% summarize(b_u = mean(rating)) %>% ggplot(aes(b_u)) +
geom_histogram(fill = "deepskyblue4", color = "azure3", bins = 30) +
 xlab("Avg. rating") + ylab("Number of users")
```



As we learned earlier, most users tend to give movies average ratings and above.

2.4.5 Average movie rating

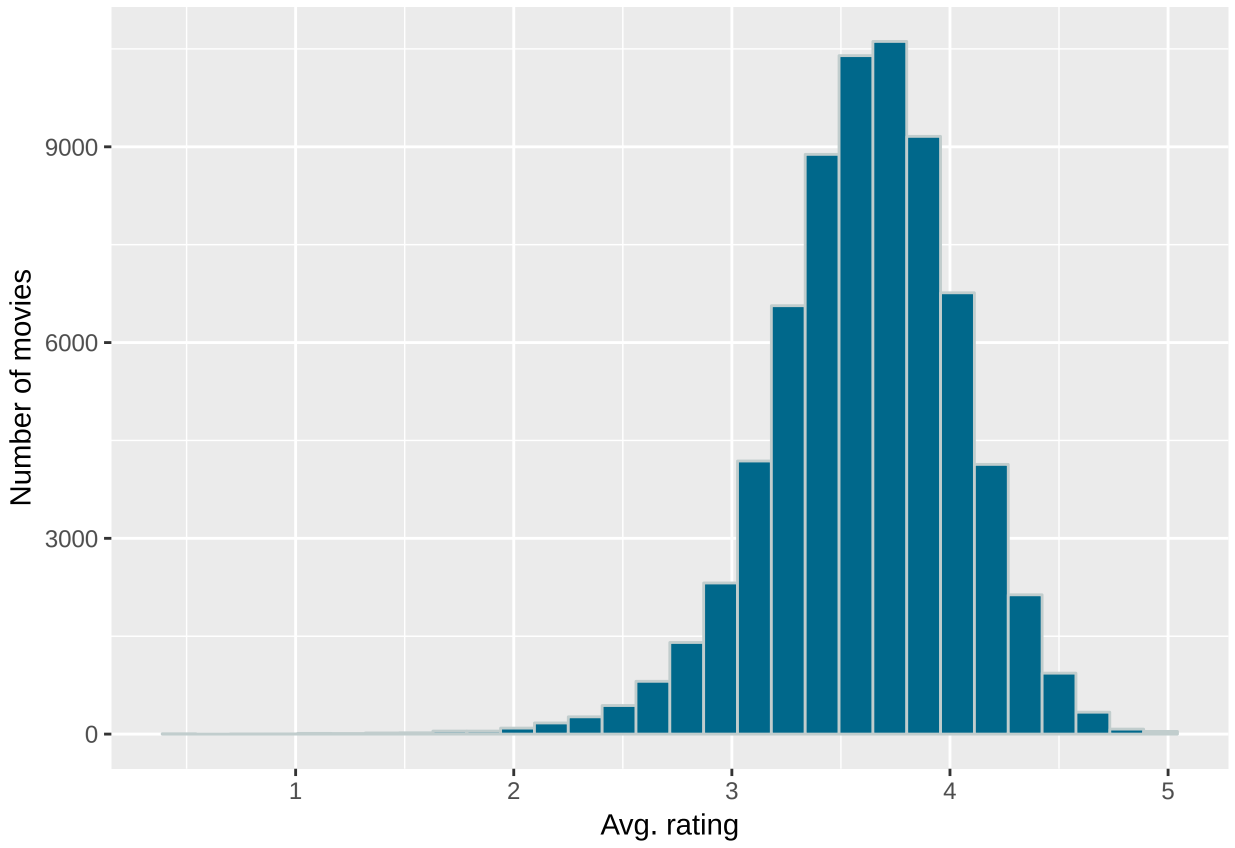
The average rating for each movie is computed as well.

```
#Create column with average movie rating
edx <- edx %>% group_by(movieId) %>% mutate(avg_movie_rating = mean(rating))
head(edx)
```

```
## # A tibble: 6 x 11
## # Groups:   movieId [6]
##   userId movieId rating ratingyear title genres releaseyear movieage
##    <int>   <dbl>  <dbl> <chr>      <chr> <chr>        <dbl>    <dbl>
## 1      1     122      5 1996       "Boo~ Comed~        1992       27
## 2      1     185      5 1996       "Net~ Actio~        1995       24
## 3      1     292      5 1996       "Out~ Actio~        1995       24
## 4      1     316      5 1996       "Sta~ Actio~        1994       25
## 5      1     329      5 1996       "Sta~ Actio~        1994       25
## 6      1     355      5 1996       "Fli~ Child~        1994       25
## # ... with 3 more variables: release_rating_range <dbl>,
## #   avg_user_rating <dbl>, avg_movie_rating <dbl>
```

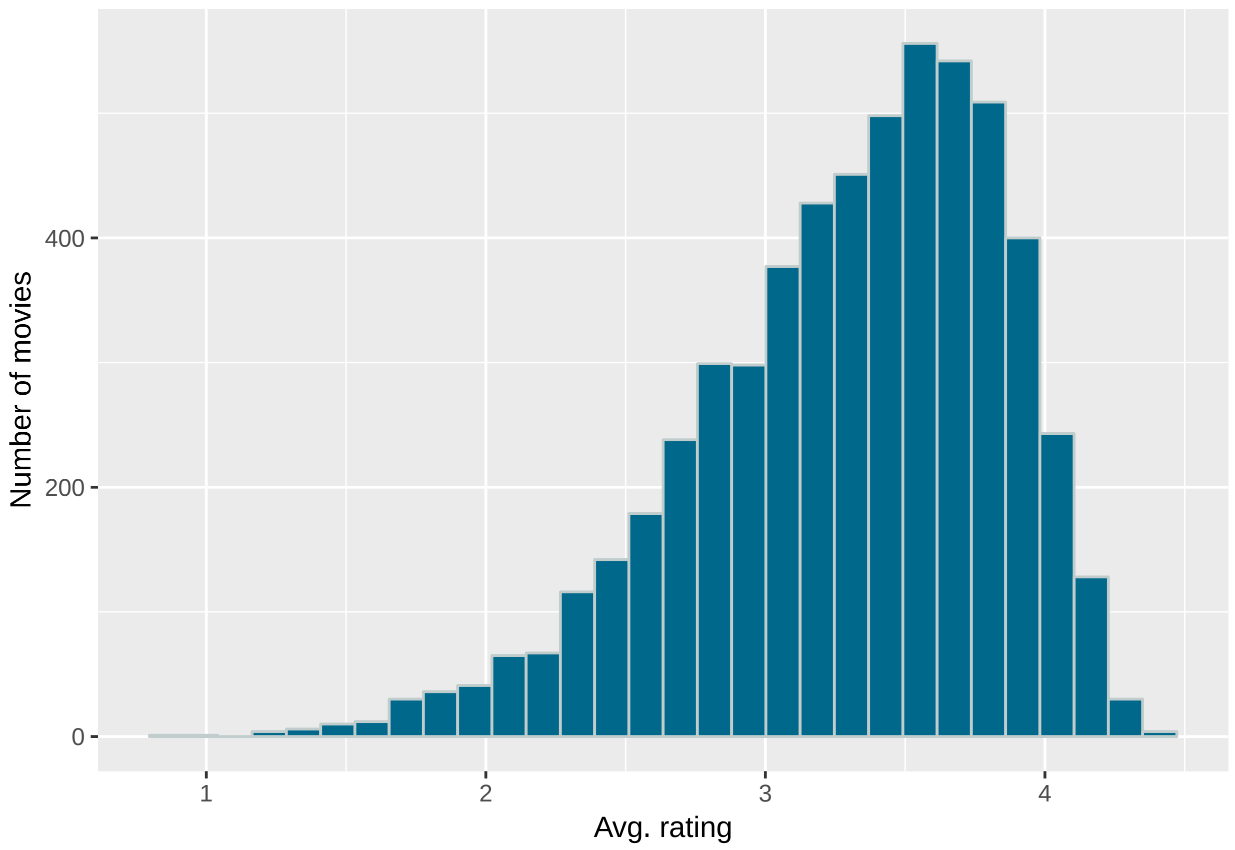
We can now show the average movie rating to see how the average ratings are distributed across movies.

```
#Give an overview of average movie rating
edx %>% group_by(userId) %>% summarize(b_u = mean(rating)) %>% ggplot(aes(b_u)) +
geom_histogram(fill = "deepskyblue4", color = "azure3", bins = 30) +
xlab("Avg. rating") + ylab("Number of movies")
```



Then, we can try to show the movies that were rated at least 100 times.

```
#Give an overview of average movie rating based on those that were rated more than a 100 times
edx %>% group_by(movieId) %>% filter(n() >= 100) %>% summarize(b_u = mean(rating)) %>% ggplot(aes(b_u))
geom_histogram(fill = "deepskyblue4", color = "azure3", bins = 30) +
xlab("Avg. rating") + ylab("Number of movies")
```



2.4.6 Genre average ratings

First, we install and load the packages needed for this section.

```
install.packages("tm", repos="http://R-Forge.R-project.org") # for text mining
```

```
## Warning: unable to access index for repository http://R-Forge.R-project.org/bin/macosx/el-capitan/co
##   cannot open URL 'http://R-Forge.R-project.org/bin/macosx/el-capitan/contrib/3.6/PACKAGES'

## Package which is only available in source form, and may need
##   compilation of C/C++/Fortran: 'tm'

## installing the source package 'tm'
```

```
install.packages("SnowballC", repos="http://R-Forge.R-project.org") # for text stemming
```

```
## Warning: package 'SnowballC' is not available (for R version 3.6.0)

## Warning: unable to access index for repository http://R-Forge.R-project.org/bin/macosx/el-capitan/co
##   cannot open URL 'http://R-Forge.R-project.org/bin/macosx/el-capitan/contrib/3.6/PACKAGES'
```

```
install.packages("wordcloud", repos="http://cran.r-project.org") # word-cloud generator
```

```
##
## The downloaded binary packages are in
##  /var/folders/7h/p5n7jmv11zj7k9r8gvhtr18h0000gn/T//Rtmph1qhge/downloaded_packages
```

```
install.packages("RColorBrewer", repos="http://cran.r-project.org") # color palettes
```

```
##
## The downloaded binary packages are in
##  /var/folders/7h/p5n7jmv11zj7k9r8gvhtr18h0000gn/T//Rtmph1qhge/downloaded_packages
```

```r
#Load
library("tm")
```

```
## Loading required package: NLP

##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate
```

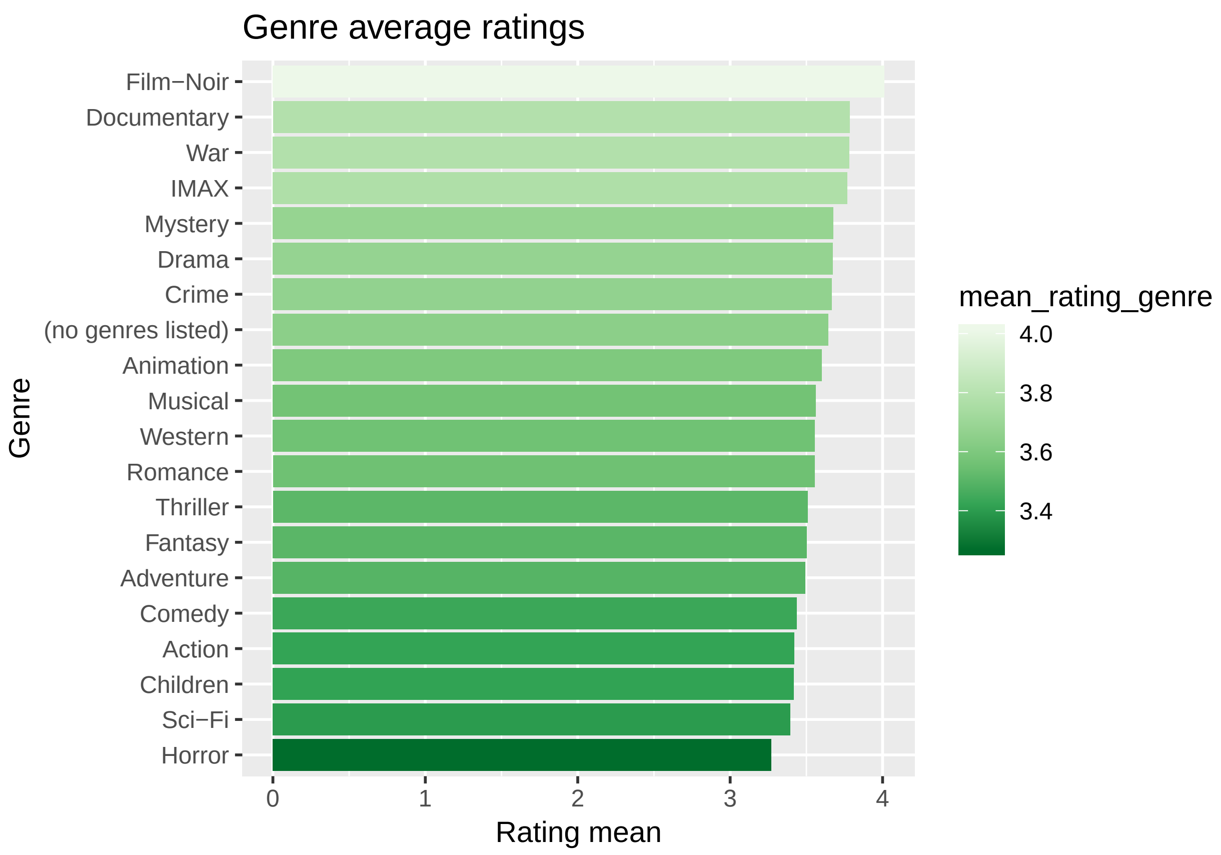
```r
library("SnowballC")
library("wordcloud")
```

```
## Loading required package: RColorBrewer
```

```r
library("RColorBrewer")
```

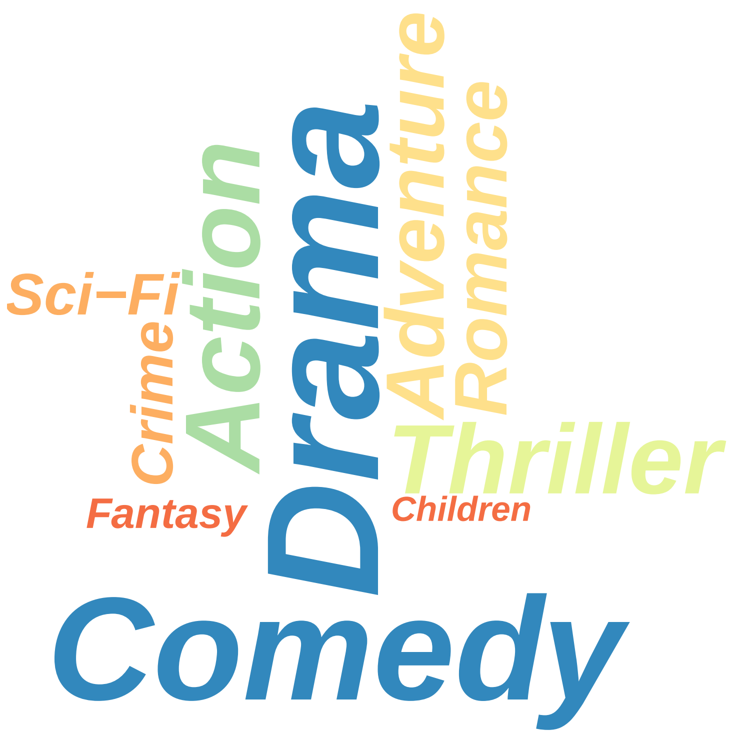
Since genres have been separated, it will be possible to see how the average genre ratings look like. This can also give an idea of whether some genres tend to be favoured over others.

```r
#Find the most popular genres by their average rating
genre_averages <- edx_gsplit %>%
group_by(genres) %>%
summarize(mean_rating_genre = mean(rating)) %>%
arrange(-mean_rating_genre)

#Show them in a histogram
genre_averages %>% ggplot(aes(reorder(genres, mean_rating_genre), mean_rating_genre, fill = mean_rating
geom_bar(stat = "identity") + coord_flip() +
scale_fill_distiller(palette = "Greens") + labs(y = "Rating mean", x = "Genre") +
ggtitle("Genre average ratings")
```



A word cloud was also constructed to see which genres have been rated the most.

```
#Show the genres that were rated the most times
wordcloud(words = genre_data$genres, freq = genre_data$n_ratings,
min.freq = 10, max.words = 10, random.order = FALSE, random.color = FALSE,
rot.per = 0.35, scale = c(5, 0.2), font = 4, colors = brewer.pal(8,"Spectral"),
main = "Most frequently rated genres")
```

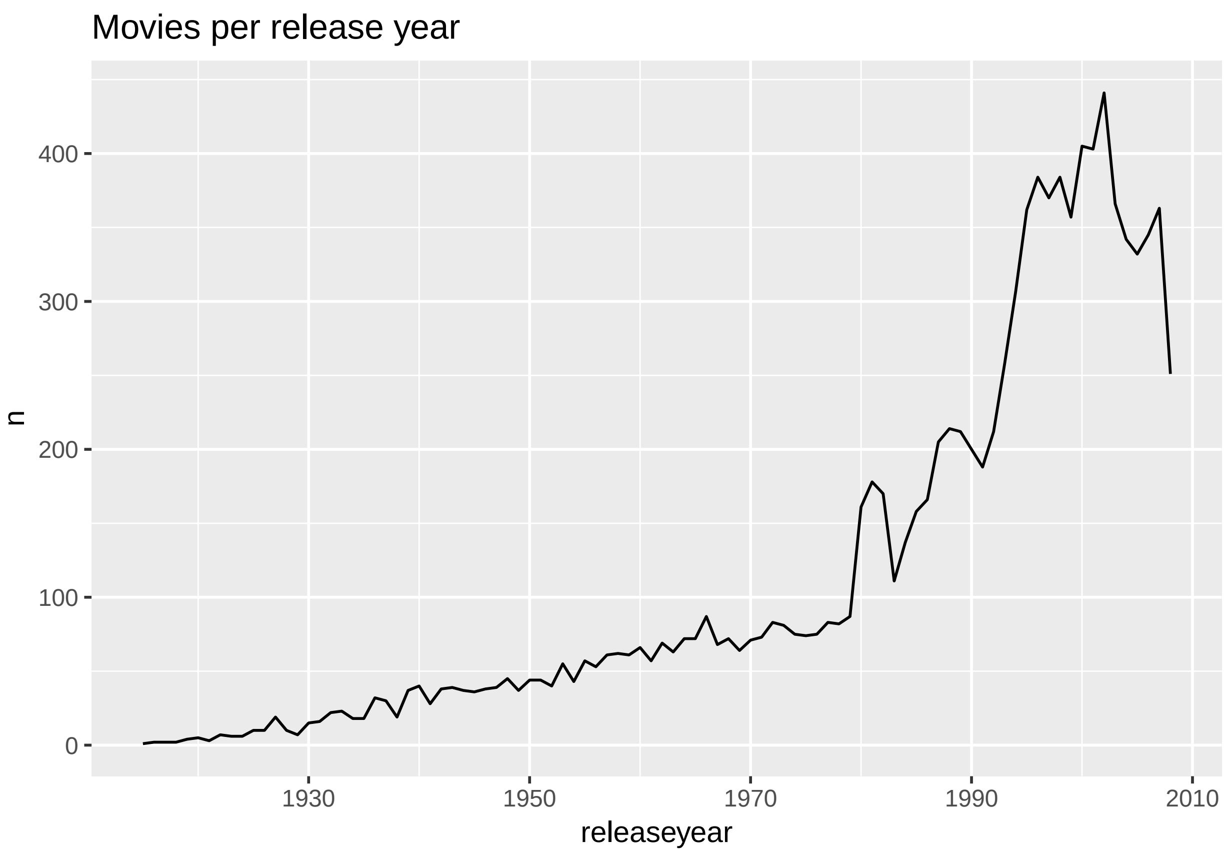


This tells us that movies that fall in the genre called drama tends to be rated more frequently, followed by action, adventure and romance, among others.

2.4.7 Number of movies released per year

Next we will look at the development in the number of movies released over time.

```
#Movies per release year
edx %>% group_by(releaseyear) %>% summarize(n = n_distinct(movieId)) %>%
ggplot(aes(releaseyear, n)) + geom_line() + ggtitle("Movies per release year")
```



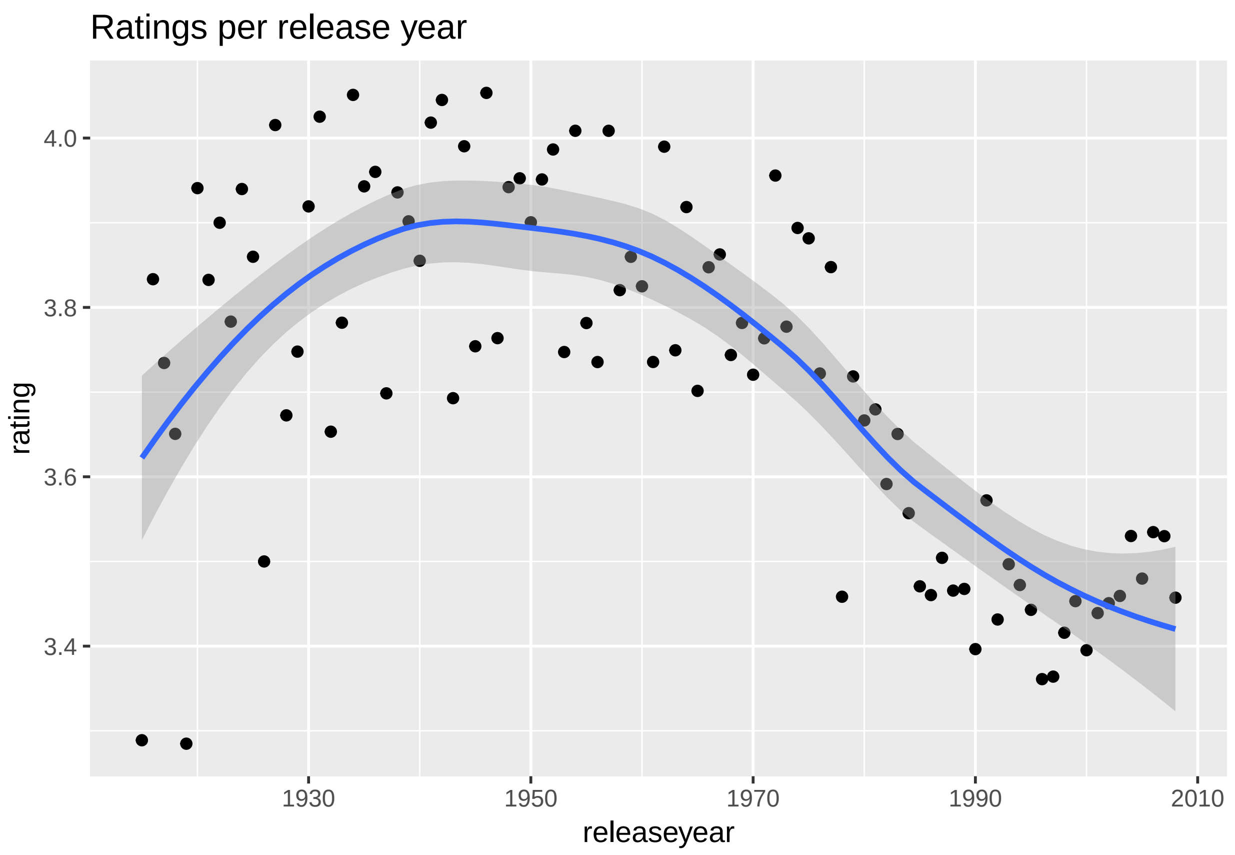
In the graph above, it shows that the release of new movies grew rapidly during the 1990's and thereafter. The data is only until 2008, which explains the extreme drop in the line showing released movies after this period.

2.4.8 Average movie rating per release year

It's also relevant to consider, whether or not the time that a movie was released could have an impact on how users rate the movie. For that reason, we compute the average movie rating per release year of the movies.

```
#Average rating per release year
edx %>% group_by(releaseyear) %>% summarize(rating = mean(rating)) %>%
ggplot(aes(releaseyear, rating)) + geom_point() + geom_smooth() + ggtitle("Ratings per release year")
```

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



The graph shows that movies released between the 1940's and 1950 had a higher average rating than movies released both before and after this period. There is a quite continuous decreasing tendency in the average rating for movies that were released in the late 1940's and after 1950. This indicates that the "old school" movies tend to be favoured over newer movies. However, it could also be due to the larger number of movies being released in recent years, leading to a wider range of movie choices and thus, a greater variety in their popularity among users e.g. due to users becoming pickier.

2.4.9 Overview of top 20 most rated movies

Now, we are going to find an overview of the top 20 most rated movies. This is shown below.

```
#Get top 20 most rated movies
most_rated_movies <- edx %>% group_by(title) %>% summarize(count = n()) %>% top_n(20, count) %>% arrang
most_rated_movies
```

```
## # A tibble: 20 x 2
##    title                                                      count
##    <chr>                                                      <int>
##  1 "Pulp Fiction "                                            31362
##  2 "Forrest Gump "                                            31079
##  3 "Silence of the Lambs, The "                               30382
##  4 "Jurassic Park "                                           29360
##  5 "Shawshank Redemption, The "                               28015
##  6 "Braveheart "                                              26212
##  7 "Fugitive, The "                                           26020
##  8 "Terminator 2: Judgment Day "                              25984
##  9 "Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) " 25672
## 10 "Batman "                                                  24585
## 11 "Apollo 13 "                                              24284
## 12 "Toy Story "                                              23790
## 13 "Independence Day (a.k.a. ID4) "                          23449
## 14 "Dances with Wolves "                                     23367
## 15 "Schindler's List "                                       23193
## 16 "True Lies "                                              22823
## 17 "Star Wars: Episode VI - Return of the Jedi "             22584
## 18 "12 Monkeys (Twelve Monkeys) "                            21891
## 19 "Usual Suspects, The "                                    21648
## 20 "Fargo "                                                  21395
```

We can see that the most rated movies, such as Pulp Fiction, Forrest Gump and The Silence of the Lambs have been rated over 30,000 times.

2.4.10 Overview of the 20 least rated movies

Next step is to get an overview of the top 20 least rated movies. This is shown below:

```
#Get 20 least rated movies
least_rated_movies <- edx %>% group_by(title) %>% summarize(count = n()) %>% top_n(-20, count) %>% arran
least_rated_movies
```

```
## # A tibble: 20 x 2
##    title                                                                 count
##    <chr>                                                                 <int>
##  1 "1, 2, 3, Sun (Un, deuz, trois, soleil) "                                 1
##  2 "100 Feet "                                                               1
##  3 "4 "                                                                      1
##  4 "Accused (Anklaget) "                                                     1
##  5 "Ace of Hearts "                                                          1
##  6 "Ace of Hearts, The "                                                     1
##  7 "Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio ~        1
##  8 "Africa addio "                                                           1
##  9 "Aleksandra "                                                             1
## 10 "Bad Blood (Mauvais sang) "                                               1
## 11 "Battle of Russia, The (Why We Fight, 5) "                                1
## 12 "Bellissima "                                                             1
## 13 "Big Fella "                                                              1
## 14 "Black Tights (1-2-3-4 ou Les Collants noirs) "                           1
## 15 "Blind Shaft (Mang jing) "                                                1
## 16 "Blue Light, The (Das Blaue Licht) "                                      1
## 17 "Borderline "                                                             1
## 18 "Brothers of the Head "                                                   1
## 19 "Chapayev "                                                               1
## 20 "Cold Sweat (De la part des copains) "                                    1
```

The least rated movies are all the movies that were rated only once. In fact, there are 125 movies that have been rated this few times. We can see this below:

```
#Get all least rated movies
all_least_rated <- edx %>% group_by(title) %>% summarize(count = n()) %>% top_n(-20, count) %>% arrange
all_least_rated
```

```
## # A tibble: 125 x 2
##    title                                                                 count
##    <chr>                                                                 <int>
##  1 "1, 2, 3, Sun (Un, deuz, trois, soleil) "                                 1
##  2 "100 Feet "                                                        1
##  3 "4 "                                                               1
##  4 "Accused (Anklaget) "                                              1
##  5 "Ace of Hearts "                                                   1
##  6 "Ace of Hearts, The "                                              1
##  7 "Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio ~     1
##  8 "Africa addio "                                                    1
##  9 "Aleksandra "                                                      1
## 10 "Bad Blood (Mauvais sang) "                                        1
## # ... with 115 more rows
```

2.4.11 Overview of the top 20 best movies based on average ratings

Looking at the top 20 movies based on average rating, we see which movies have been rated to be the best.

```
#Get top 20 movies based on average ratings
top_movies <- edx %>% group_by(title) %>% summarize(rating = mean(rating), count = n()) %>% top_n(20, ra
top_movies
```

```
## # A tibble: 20 x 3
##    title                                                       rating count
##    <chr>                                                        <dbl> <int>
##  1 "Blue Light, The (Das Blaue Licht) "                          5        1
##  2 "Fighting Elegy (Kenka erejii) "                              5        1
##  3 "Hellhounds on My Trail "                                     5        1
##  4 "Satan's Tango (Sátántangó) "                                 5        2
##  5 "Shadows of Forgotten Ancestors "                             5        1
##  6 "Sun Alley (Sonnenallee) "                                    5        1
##  7 "Constantine's Sword "                                        4.75     2
##  8 "Human Condition II, The (Ningen no joken II) "               4.75     4
##  9 "Human Condition III, The (Ningen no joken III) "             4.75     4
## 10 "Who's Singin' Over There? (a.k.a. Who Sings Over There) (~   4.75     4
## 11 "Class, The (Entre les Murs) "                                4.67     3
## 12 "I'm Starting From Three (Ricomincio da Tre) "                4.67     3
## 13 "Bad Blood (Mauvais sang) "                                   4.5      1
## 14 "Demon Lover Diary "                                          4.5      1
## 15 "End of Summer, The (Kohayagawa-ke no aki) "                  4.5      3
## 16 "Kansas City Confidential "                                   4.5      1
## 17 "Ladrones "                                                   4.5      1
## 18 "Life of Oharu, The (Saikaku ichidai onna) "                  4.5      3
## 19 "Man Named Pearl, A "                                         4.5      1
## 20 "Mickey "                                                     4.5      1
```

It also shows, based on the data, that a lot of the top rated movies based on average rating have in fact only been rated very few times, if not only once. This indicates potential noise in the data and can disrupt our results when we conduct the model.

2.4.12 Overview of the top 20 most rated movies

Below, we can see that Pulp Fiction, Forest Gump, The Silence of the Lamb, Jurassic Park and The Shawnshank Redemption are among the most rated movies.

```
#Get top 20 most rated movies
most_rated_movies <- edx %>% group_by(title) %>% summarize(count = n()) %>% top_n(20, count) %>% arrang
most_rated_movies
```

```
## # A tibble: 20 x 2
##    title                                                   count
##    <chr>                                                   <int>
##  1 "Pulp Fiction "                                         31362
##  2 "Forrest Gump "                                         31079
##  3 "Silence of the Lambs, The "                            30382
##  4 "Jurassic Park "                                        29360
##  5 "Shawshank Redemption, The "                            28015
##  6 "Braveheart "                                           26212
##  7 "Fugitive, The "                                        26020
##  8 "Terminator 2: Judgment Day "                           25984
##  9 "Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) " 25672
## 10 "Batman "                                               24585
## 11 "Apollo 13 "                                            24284
## 12 "Toy Story "                                            23790
## 13 "Independence Day (a.k.a. ID4) "                        23449
## 14 "Dances with Wolves "                                   23367
## 15 "Schindler's List "                                     23193
## 16 "True Lies "                                            22823
## 17 "Star Wars: Episode VI - Return of the Jedi "           22584
## 18 "12 Monkeys (Twelve Monkeys) "                          21891
## 19 "Usual Suspects, The "                                  21648
## 20 "Fargo "                                                21395
```

2.4.13 Overview of the 20 worst movies based on average ratings

We can also look at the 20 movies that have the lowest average ratings.

```r
#Get 20 worst movies based on average ratings
bottom_movies <- edx %>% group_by(title) %>% summarize(rating = mean(rating), count = n()) %>% top_n(-2
bottom_movies
```

```
## # A tibble: 20 x 3
##    title                                                   rating count
##    <chr>                                                    <dbl> <int>
##  1 "Accused (Anklaget) "                                    0.5       1
##  2 "Besotted "                                              0.5       2
##  3 "Confessions of a Superhero "                            0.5       1
##  4 "Hi-Line, The "                                          0.5       1
##  5 "War of the Worlds 2: The Next Wave "                    0.5       2
##  6 "SuperBabies: Baby Geniuses 2 "                          0.795    56
##  7 "Hip Hop Witch, Da "                                     0.821    14
##  8 "Disaster Movie "                                        0.859    32
##  9 "From Justin to Kelly "                                  0.902   199
## 10 "Criminals "                                             1         2
## 11 "Dischord "                                              1         1
## 12 "Dog Run "                                               1         1
## 13 "Monkey's Tale, A (Les Château des singes) "             1         1
## 14 "Mountain Eagle, The "                                   1         2
## 15 "Relative Strangers "                                    1         1
## 16 "Stacy's Knights "                                       1         1
## 17 "When Time Ran Out... (a.k.a. The Day the World Ended) " 1         1
## 18 "Pokémon Heroes "                                        1.03    137
## 19 "Roller Boogie "                                         1.03     15
## 20 "Carnosaur 3: Primal Species "                           1.09     68
```

The 20 worst rated movies equally consist of some movies that were only rated once or very few times.

However, there are also some that have been rated over 50 and even 100 times. Regardless, it is important to bear in mind the movie effect in our analysis to make sure that overtraining is prevented.

3. Data analysis

Before continuing with our analysis, the edx dataset is further split into test and training sets. The test set will be 10% of the data. The purpose of this is to use the test set to test our intermediary models before deciding on the final model to run on the validation set. The training set is split further into tests and training sets. Test set will be 10% of the data.

```
#Split edx data further in to test and training sets
# Test set will be 10% of current edx data
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index2 <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index2,]
temp2 <- edx[test_index2,]
```

It is made sure that the variables in the test set are in the training set as well.

```
# Make sure userId and movieId in test set are also in training set
test_set <- temp2 %>%
semi_join(train_set, by = "movieId") %>%
semi_join(train_set, by = "userId")

# Add rows removed from test set back into training set
removed <- anti_join(temp2, test_set)
```

```
## Joining, by = c("userId", "movieId", "rating", "ratingyear", "title", "genres", "releaseyear", "movi
```

```
train_set <- rbind(train_set, removed)
rm(test_index2, temp2, removed)
```

As stated in the introduction, we will use RMSE, also known as the error loss function, from which we can determine which model that minimizes the residual mean squared error.

Here, N is defined as the number of observations (combinations of user/movie ratings and the sum all of these). The RMSE in our case denotes the error between our predicted rating and the true rating. Thus, the lower this value, the better the model should be at predicting movie ratings from a dataset it hasn't seen before.

We can now create a function that calculates RMSE.

```
#Define the function that calculates RMSE
RMSE <- function(true_ratings, predicted_ratings){
sqrt(mean((true_ratings - predicted_ratings)^2, na.rm=T))
}
```

3.1 Models

Next step is to test out different types of model and find out which one performs the best.

3.1.1 Basic model with average movie ratings

The most basic of model is the one where the same rating is predicted regardless of which user rated it. In this case, RMSE would be the average rating of all the movies in the dataset. This function looks as follows:

Yu,i = u + Eu,i

with Eu,i being the independent errors centered around 0 and with u being the true movie rating for all the movies. Thus, with this model the value of RMSE that minimizes the RMSE is the least square estimate of u and the average of all ratings.

```
#Get mu_hat with the simplest model
mu_hat <- mean(train_set$rating)
mu_hat
```

```
## [1] 3.512456
```

When trying to test this on our test set, we get a slightly different value of RMSE.

```
#Predict the known ratings with mu_hat
naive_rmse <- RMSE(test_set$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060054
```

A results table is created to save all of the coming results:

```
#Create the results table
rmse_results <- tibble(method = "Simple average model", RMSE = naive_rmse)
```

We can now continue to test another range of models in order to determine, which one predicts the best RMSE.

3.1.2 Model with movie effect

As addressed earlier, there may be variation in how movies are rated - e.g. some movies by default get better ratings than others, so it is necessary to take this "bias" into consideration.

The next model to be tested is the one where we include movie effect in the formula.

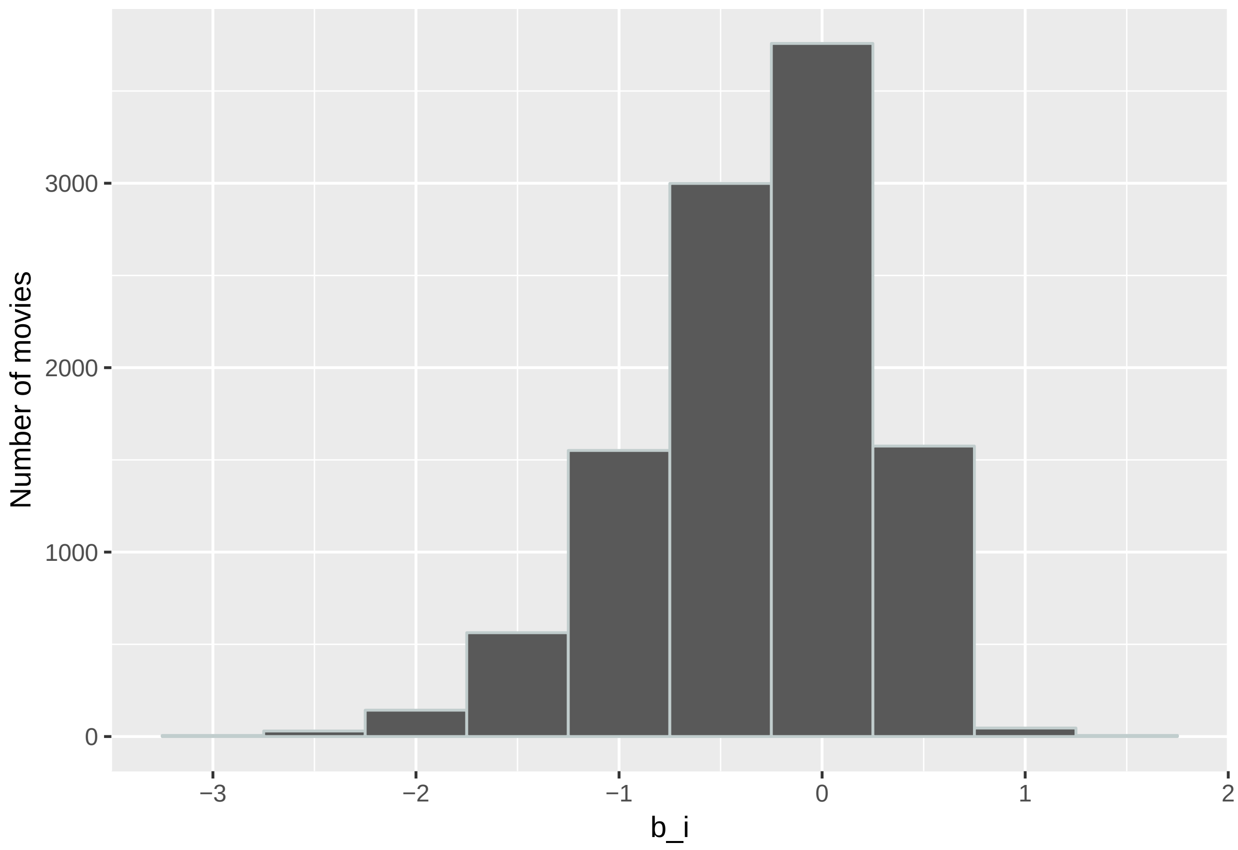
Yu,i = u + bi + Eu,i

This is an extension of the previous model, as it now includes bi, which can be considered the movie bias. Again, least squares is used to estimate the value.

```
#Penalize movie effects and adjust the mean
b_i <- train_set %>% group_by(movieId) %>%
summarize(b_i = sum(rating - mu_hat)/(n() + 1))
```

Next, we can plot the calculated estimates of the movie effect bias according to the number of movies.

```
#Save and plot the movie averages with the movie effect model
movie_effect_avgs <- train_set %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu_hat))
movie_effect_avgs %>% qplot(b_i, geom = "histogram", bins = 10, data = ., color = I("azure3"), xlab = "
```



We save the new predicted ratings.

```
#Save the new predicted ratings
predicted_ratings <- mu_hat + test_set %>% left_join(movie_effect_avgs, by='movieId') %>%
pull(b_i)
```

Now we can test and calculate the new RMSE.

```
#Calculate the RMSE for the movie effect model
movie_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
movie_effect_rmse
```

```
## [1] 0.9429615
```

And then save it with the results from the other model:

```
#Save with previous model results
rmse_results <- bind_rows(rmse_results, tibble(method="Movie effect model", RMSE = movie_effect_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   method                 RMSE
##   <chr>                 <dbl>
## 1 Simple average model 1.06
## 2 Movie effect model   0.943
```

As the RMSE has now reduced from 1.06 to 0.943, the model has improved and does a better job at predicting the movie ratings.

3.1.3 Model with movie and user effect

Besides the movie effect, another thing to consider is that users might not have the same rating patterns. While some users give most movies very high ratings, others are a lot more picky or pessimistic as discussed earlier. For that reason, the user effect should be included as well.

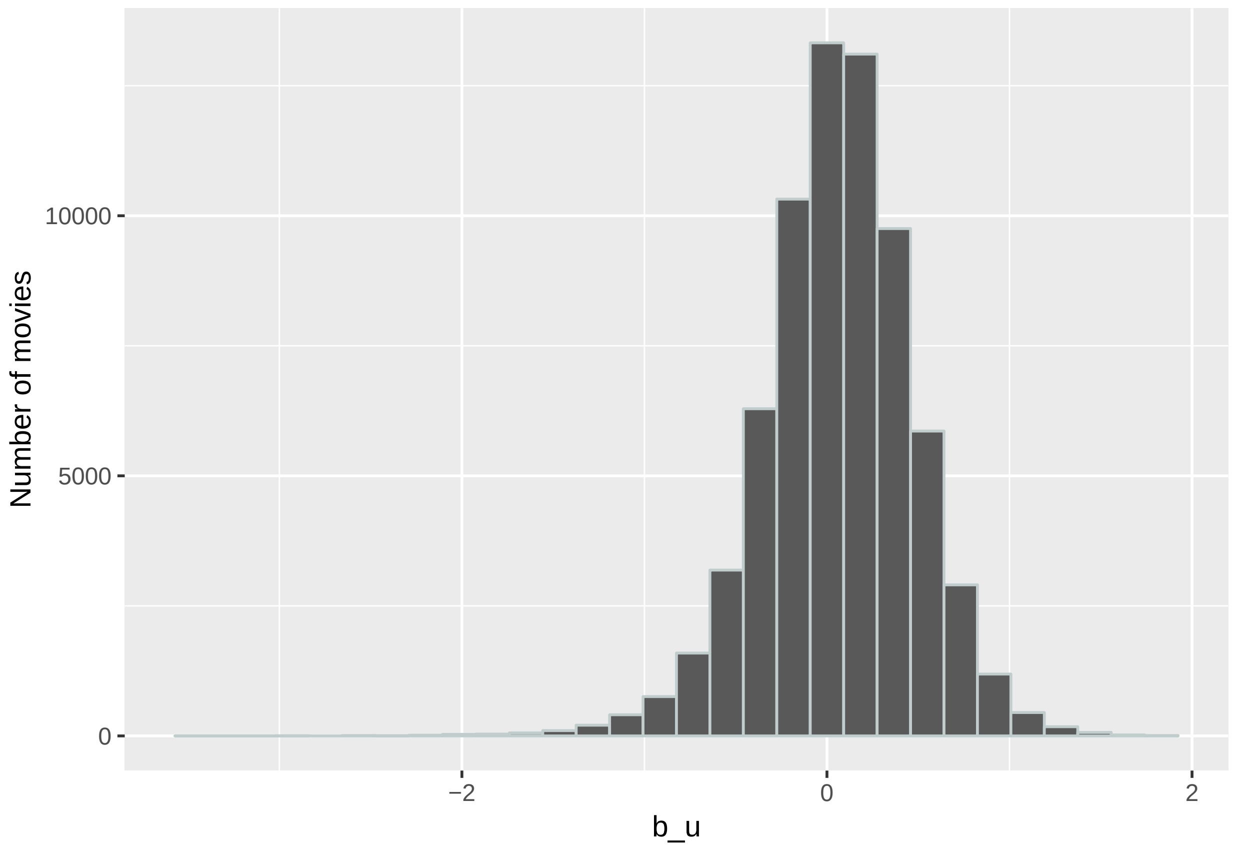
The user effect model could look as follows:

Yu,i = u + bi + bu + Eu,i

```
#Penalize user and movie effects and adjust the mean
b_u <- train_set %>% group_by(userId) %>%
summarize(b_u = sum(rating - b_i - mu_hat)/(n() + 1))
```

Next, we can save and plot the user averages with the movie and user effect model.

```
#Save and plot the user averages with the movie and user effect model
user_effect_avgs <- train_set %>% left_join(movie_effect_avgs, by='movieId') %>%
group_by(userId) %>%
summarize(b_u = mean(rating - mu_hat - b_i))
user_effect_avgs %>% qplot(b_u, geom = "histogram", bins = 30, data = ., color = I("azure3"), xlab = "b_
```



As discussed above, there could be a difference between how excited users are about the movies that they rate. This also shows on this histogram, as we can see that there is some variability in how users rate movies.

Next, the predicted values with the new model can be constructed:

```
#Calculate the new predicted ratings
predicted_ratings <- test_set %>%
left_join(movie_effect_avgs, by='movieId') %>%
left_join(user_effect_avgs, by = 'userId') %>%
```

```r
mutate(pred = mu_hat + b_i + b_u) %>%
pull(pred)
```

And we can calculate the new RMSE for the model with both the movie and user effect.

```r
#Calculate the new RMSE
user_effect_rmse <- RMSE(predicted_ratings, test_set$rating)
user_effect_rmse
```

```
## [1] 0.8646843
```

These results are saved with our previous results.

```r
#Save with the previous model results
rmse_results <- bind_rows(rmse_results,
tibble(method="Movie + User Effects Model",
RMSE = user_effect_rmse))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Simple average model       1.06
## 2 Movie effect model         0.943
## 3 Movie + User Effects Model 0.865
```

Now, we can see that the value of RMSE has further reduced, indicating that the model with both the movie and user effect in mind does an even better job at explaining movie ratings.

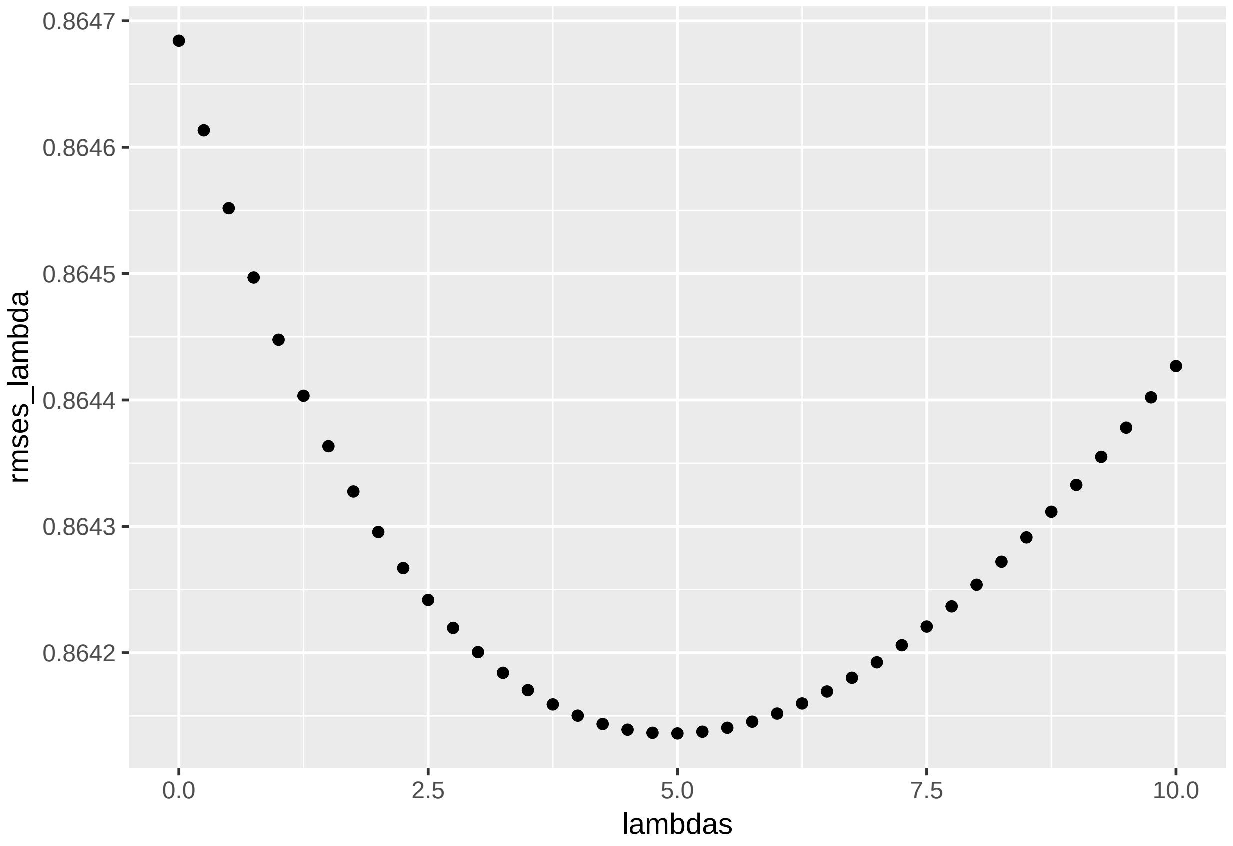
3.1.4 Regularized model with movie and user effect

The model with the movie effect proved to include some highly rated movies that were in fact only rated once or at least very few times. This causes disturbances in the data, because it is difficult to rely on movies with such few ratings - this is especially if these have very large predictions.

To reduce this noise, penalization can be used to limit the variability caused by the movie effect. Similarly, it is possible to penalize the user effect, because of the users who only rated very few movies.

First, we need to choose the tuning parameter, lamda, to determine the different values of lambda and their respective values of RMSE. We use the training set for the tuning of the optimal lambda.

```r
#Chose the tuning parameter
lambdas <- seq(0, 10, 0.25)

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

mu_hat <- mean(train_set$rating)

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

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

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

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

qplot(lambdas, rmses_lambda)
```



After having completed the tuning, the lambda that provides the lowest RMSE can be found.

```
#Find lambda that minimizes RMSE
lambda <- lambdas[which.min(rmses_lambda)]
lambda
```

```
## [1] 5
```

Knowing that 5 appears to be the value of lambda that provides the best RMSE, it is applied on the test set.

```
#Apply the optimal lambda on the test set
predicted_reg <- sapply(lambda, function(l){

mu_hat <- mean(train_set$rating)

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

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

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

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

#Show the new RMSE
predicted_reg
```

```
## [1] 0.8641362
```

This gives an RMSE of 0.8641. The result is saved with the previous rmse results.

```r
#Save rmse with remaining results
rmse_results <- bind_rows(rmse_results,
tibble(method="Regularized Movie + User Effect Model",
RMSE = predicted_reg))
rmse_results
```

```
## # A tibble: 4 x 2
##   method                                 RMSE
##   <chr>                                 <dbl>
## 1 Simple average model                  1.06
## 2 Movie effect model                    0.943
## 3 Movie + User Effects Model            0.865
## 4 Regularized Movie + User Effect Model 0.864
```

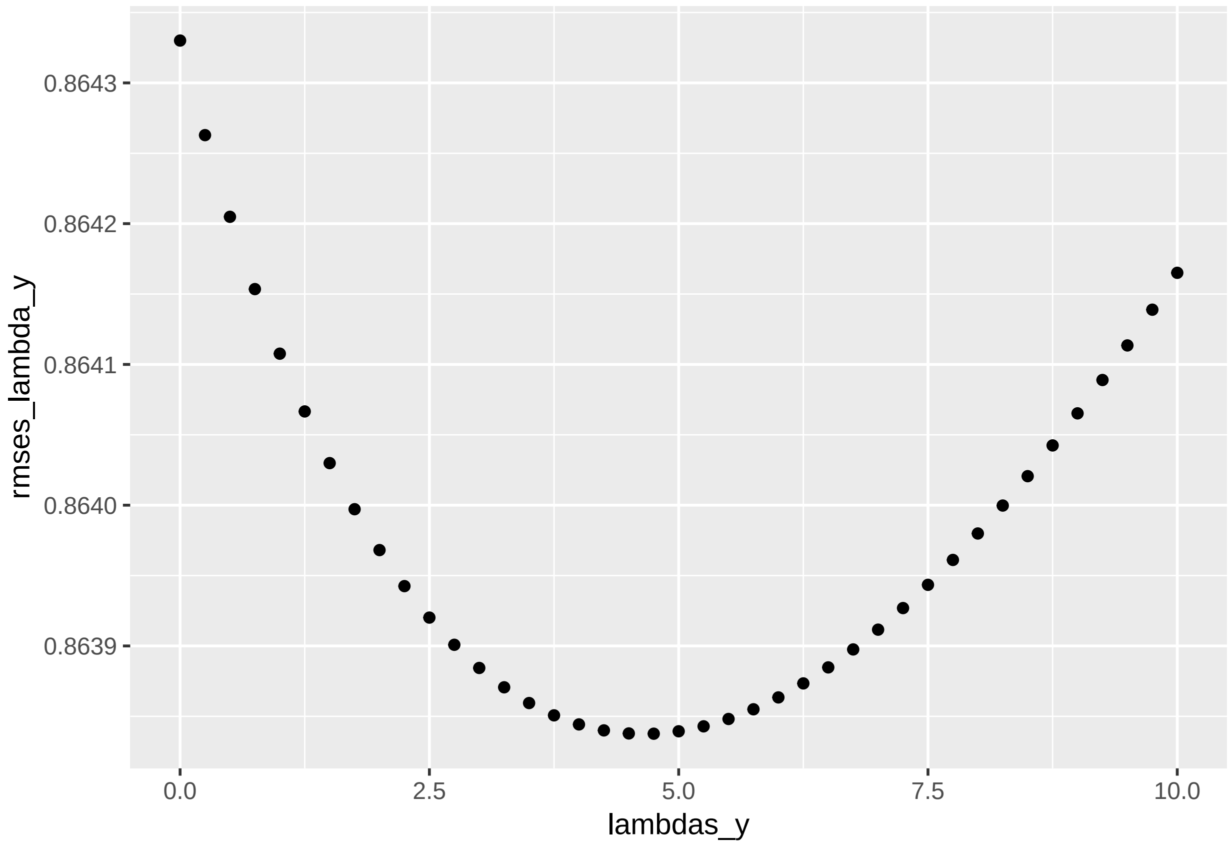
Based on this, we see that the model has improved further with an even lower value of RMSE.

3.1.5 Regularized model with movie, user and time effect

Another possibility to consider is whether the time that the movie was released could influence the movie rating. A model containing the movie's release year is now tested.

Make new regularized model with year effect added.

```r
#Chose the tuning parameter
lambdas_y <- seq(0, 10, 0.25)

rmses_lambda_y <- sapply(lambdas_y, function(l){

mu_hat <- mean(train_set$rating)

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

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

predicted_ratings <-
test_set %>%
left_join(b_i, by = "movieId") %>%
left_join(b_u, by = "userId") %>%
left_join(b_y, by = "releaseyear") %>%
mutate(pred = mu_hat + b_i + b_u + b_y) %>%
pull(pred)

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

qplot(lambdas_y, rmses_lambda_y)
```



```r
#Find lambda that minimizes RMSE
lambda_y <- lambdas_y[which.min(rmses_lambda_y)]
lambda_y
```

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

So the optimal lambda for this model, which minimizes the RMSE is 4.75. We can now apply it on the test set to see what value of RMSE it results in.

```
#Apply the optimal lambda on the test set
predicted_reg_y <- sapply(lambda_y, function(l){

mu_hat <- mean(train_set$rating)

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

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

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

predicted_ratings <-
test_set %>%
left_join(b_i, by = "movieId") %>%
left_join(b_u, by = "userId") %>%
left_join(b_y, by = "releaseyear") %>%
mutate(pred = mu_hat + b_i + b_u + b_y) %>%
pull(pred)

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

predicted_reg_y
```

```
## [1] 0.8638377
```

With an even lower value, this model leads to an RMSE of 0.863877 and is thus the best so far at predicting user's movie ratings.

Results can be saved to the previous results.

```
#Save rmse with remaining results
rmse_results <- bind_rows(rmse_results,
tibble(method="Regularized Movie + User + Year Effect Model",
RMSE = predicted_reg_y))
rmse_results
```

```
## # A tibble: 5 x 2
##   method                                          RMSE
##   <chr>                                          <dbl>
## 1 Simple average model                           1.06
## 2 Movie effect model                             0.943
## 3 Movie + User Effects Model                     0.865
## 4 Regularized Movie + User Effect Model        0.864
## 5 Regularized Movie + User + Year Effect Model 0.864
```

3.1.6 Applying the chosen final model on the validation set

Now, the best model, which is the regularized one with movie, user and year effect, is chosen. We can finally apply it on the validation set and get the final RMSE.

```
#Use the optimal model on the validation set
chosen_model_RMSE <- sapply(lambda_y, function(l){

mu_hat <- mean(edx$rating)

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

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

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

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

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

chosen_model_RMSE
```

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

4. Conclusion

Having tested a range of models, it can be concluded that the simplest of algorithms does a quite poor job at predicting movie ratings. It is highly unlikely that a user's movie preference can be determined by an average rating of all movies, as there are many more factors that can determine how the individual user rates a movie.

Based on the analysis in the previous section, we can conclude that the regularized model with the user, movie and year effect does the best job at predicting the movie ratings with a final RMSE of 0.8645223. It is slightly higher than the RMSE when using the exact same model during regularization, where the value was 0.8638377, but it still a better model than the first four we tested.

For further improvements, matrix factorization could be applied to adjust for group variation but that is beyond the scope of this project. In order to make a better algorithm, it would also have been relevant to include actors, had this data been available, as users might favor some actors over others.