Movie Lens Recommendation System

Gayrol Taylor

1/10/2020

```
# required packages for our project
if(!require(kableExtra)) install.packages('kableExtra',
repos = 'http://cran.us.r-project.org')
```

```
## Loading required package: kableExtra
```

```
## Warning: package 'kableExtra' was built under R version 3.6.2
```

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

```
## Loading required package: dataCompareR
```

```
## Warning: package 'dataCompareR' was built under R version 3.6.2
```

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

```
## Loading required package: tidyverse
```

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

```
## v ggplot2 3.2.0     v purrr   0.3.2
## v tibble  2.1.3     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::group_rows() masks kableExtra::group_rows()
## 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
```

```
# Loading all needed libraries

library(kableExtra)
library(dataCompareR)
library(tidyverse)
library(caret)
library(data.table)
library(stringr)
library(ggplot2)

#############################################################
# Create edx set, validation set, and submission file
#############################################################

# Note: this process could take a couple of minutes

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

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

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- read.table(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 will be 10% of MovieLens data

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

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

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set

removed <- anti_join(temp, validation)
```

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

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

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Executive Summary

This project is aimed at creating a movie recommendation system using the MovieLens dataset (supplied by the HarvardX: PH125.9x Data Science: Capstone course). In so doing, it shows how useful data science and machine learning can be and the vast potential regarding data analysis for real world decision making.

The movielens dataset has more than 10 million ratings, divided in nine million for training and one million for validation. Each rating comes with a userId, movieId, rating, timestamp, title and genre. Within the training dataset are approximately 70,000 users and approximately 11,000 different movies spaning 20 genres. These genres include: Comedy, Romance, Action, Crime, Thriller, Drama, Sci-Fi and more.

This report will present an overview of the dataset, data analysis and conclusion. Whilst the exercise was used to test the accuracy of the algorithm using the Root Mean Square Error (RMSE), it was also used to determine a RMSE lower than 0.8775.

Root Mean Squared Error (RMSE) Formula:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$$

The **Regularized Movie+User Model** proved to be capable of reaching a RMSE of **0.8628015*.

Data Analysis

As a preparatory step, the edx and validation datafarmes will be saved as R objects. This prevents needing to reload the data.

```
# Save our data as R objects
save(edx, file = 'edx.RData')
save(validation, file = 'validation.RData')
```

```
# The data is then accessed using the load function
load('edx.RData')
load('validation.RData')
```

Data Overview

Sample of the data in **edx** dataset:

```
as_tibble(edx) %>%
slice(1:5) %>%
knitr::kable()
```

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

Sample of the data in the **validation** dataset:

```
as_tibble(validation) %>%
slice(1:5) %>%
knitr::kable()
```

userId	movieId	rating	timestamp	title	genres
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	480	5	838983653	Jurassic Park (1993)	Action|Adventure|Sci-Fi|Thriller
1	586	5	838984068	Home Alone (1990)	Children|Comedy
2	151	3	868246450	Rob Roy (1995)	Action|Drama|Romance|War
2	858	2	868245645	Godfather, The (1972)	Crime|Drama

edx data frame has 9000055 rows and 6 variables. **validation** data frame has 999999 rows and 6 variables.

```
library(dataCompareR)
comp_edx_val <- rCompare(edx, validation)
```

```
## Running rCompare...
```

```
## CAUTION - There are 60000324 elements across both data frames.dataCompareR may take a little
longer than usual for large data sizes.
```

```
comp_summ <- summary(comp_edx_val)
```

```
## dataCompareR is generating the summary...
```

```
comp_summ[c('datasetSummary', 'ncolInAOnly', 'ncolInBOnly', 'ncolCommon', 'rowsInAOnly', 'rowsIn
BOnly', 'nrowCommon')]
```

```
## $datasetSummary
##   Dataset Name Number of Rows Number of Columns
## 1          edx        9000055                 6
## 2   validation         999999                 6
##
## $ncolInAOnly
## [1] 0
##
## $ncolInBOnly
## [1] 0
##
## $ncolCommon
## [1] 6
##
## $rowsInAOnly
##   indices_removed
## 1         6683124
## 2         5710663
## 3         6143442
## 4         3175153
## 5         6157829
##
## $rowsInBOnly
## [1] indices_removed
## <0 rows> (or 0-length row.names)
##
## $nrowCommon
## [1] 999999
```

Duplication Checks The number of distinct users, movies and genres are:

```
# Distinct users, movies, genres
dist_col <- edx %>%
summarize(distinct_users = n_distinct(userId),
            distinct_movies = n_distinct(movieId),
            distinct_genres = n_distinct(genres))
knitr::kable(dist_col)
```

distinct_users	distinct_movies	distinct_genres
69878	10677	797

Data Sorting

Let’s tidy the data by arranging the **title**, **timestamp** and **genres** columns. This is needed because the timestamp is still in a coded format, the title is grouped with the release year and the genre has multiple entries.

The new arrangement will be as follows: 1. **userId** converted from a class of **integer** to **factor** 2. **movieId** converted from a class of **integer** to **factor** 3. New column created **premier year** for the movie year (year extracted from the title) 4. The class of **genres** will be changed to **factor** 5. **Timestamp** changed to **rate_year**

See the transformed tables below:

```
tidydf <- function(df){
  df$genres <- as.factor(df$genres) #Convert genres to factor
  df$timestamp <- as.Date(as.POSIXct(df$timestamp, origin='1970-01-01'))
  #Convert timestamp
  names(df)[names(df) == 'timestamp'] <- 'rate_year' # Rename column timestamp to rate_year
  df <- df %>%
    mutate(title = str_trim(title), rate_year = year(rate_year)) %>%  #Mutate title and rate_yea
r
    extract(title, c('title', 'premier_year'), regex = '(.*)\\s\\((\\d+)\\)', convert = TRUE)
#Separate title from year
return(df)
}
# Transform our dataframes
edx <- tidydf(edx)
validation <- tidydf(validation)
```

```
as_tibble(edx)
```

```
## # A tibble: 9,000,055 x 7
##    userId movieId rating rate_year title        premier_year genres
##     <int>   <dbl>  <dbl>     <int> <chr>               <int> <fct>
##  1      1     122      5      1996 Boomerang            1992 Comedy|Romance
##  2      1     185      5      1996 Net, The             1995 Action|Crime|~
##  3      1     292      5      1996 Outbreak             1995 Action|Drama|~
##  4      1     316      5      1996 Stargate             1994 Action|Advent~
##  5      1     329      5      1996 Star Trek: ~         1994 Action|Advent~
##  6      1     355      5      1996 Flintstones~         1994 Children|Come~
##  7      1     356      5      1996 Forrest Gump         1994 Comedy|Drama|~
##  8      1     362      5      1996 Jungle Book~         1994 Adventure|Chi~
##  9      1     364      5      1996 Lion King, ~         1994 Adventure|Ani~
## 10      1     370      5      1996 Naked Gun 3~         1994 Action|Comedy
## # ... with 9,000,045 more rows
```

```
as_tibble(validation)
```

```
## # A tibble: 999,999 x 7
##    userId movieId rating rate_year title          premier_year genres
##     <int>   <dbl>  <dbl>     <int> <chr>                 <int> <fct>
##  1      1     231    5        1996 Dumb & Dumber          1994 Comedy
##  2      1     480    5        1996 Jurassic Park          1993 Action|Adve~
##  3      1     586    5        1996 Home Alone             1990 Children|Co~
##  4      2     151    3        1997 Rob Roy                1995 Action|Dram~
##  5      2     858    2        1997 Godfather, The         1972 Crime|Drama
##  6      2    1544    3        1997 Lost World: J~         1997 Action|Adve~
##  7      3     590    3.5      2006 Dances with W~         1990 Adventure|D~
##  8      3    4995    4.5      2005 Beautiful Min~         2001 Drama|Myste~
##  9      4      34    5        1996 Babe                   1995 Children|Co~
## 10      4     432    3        1996 City Slickers~         1994 Adventure|C~
## # ... with 999,989 more rows
```

Missing Value Analysis

```
# Check edx dataframe for missing values
edx_na <- edx %>%
filter(is.na(title) | is.na(year))
```

```
## Warning in is.na(year): is.na() applied to non-(list or vector) of type
## 'closure'
```

```
glimpse(edx_na)
```

```
## Observations: 0
## Variables: 7
## $ userId       <int>
## $ movieId      <dbl>
## $ rating       <dbl>
## $ rate_year    <int>
## $ title        <chr>
## $ premier_year <int>
## $ genres       <fct>
```

```
# Check validation dataframe for missing values
validation_na <- validation %>%
filter(is.na(title) | is.na(year))
```

```
## Warning in is.na(year): is.na() applied to non-(list or vector) of type
## 'closure'
```

```
glimpse(validation_na)
```

```
## Observations: 0
## Variables: 7
## $ userId       <int>
## $ movieId      <dbl>
## $ rating       <dbl>
## $ rate_year    <int>
## $ title        <chr>
## $ premier_year <int>
## $ genres       <fct>
```

No missing value was found in the edx dataset. The dataset contains 10,677 unique movies, 69,878 unique users, and 797 unique combinations of genres with a mean movie rating of ~3.5 out of 5.

Movie Ratings Analysis

The relationship between the type of movie ratings and its frequency can help us to further understand the data. A series of tests will therefore be done on the **edx** dataframe.

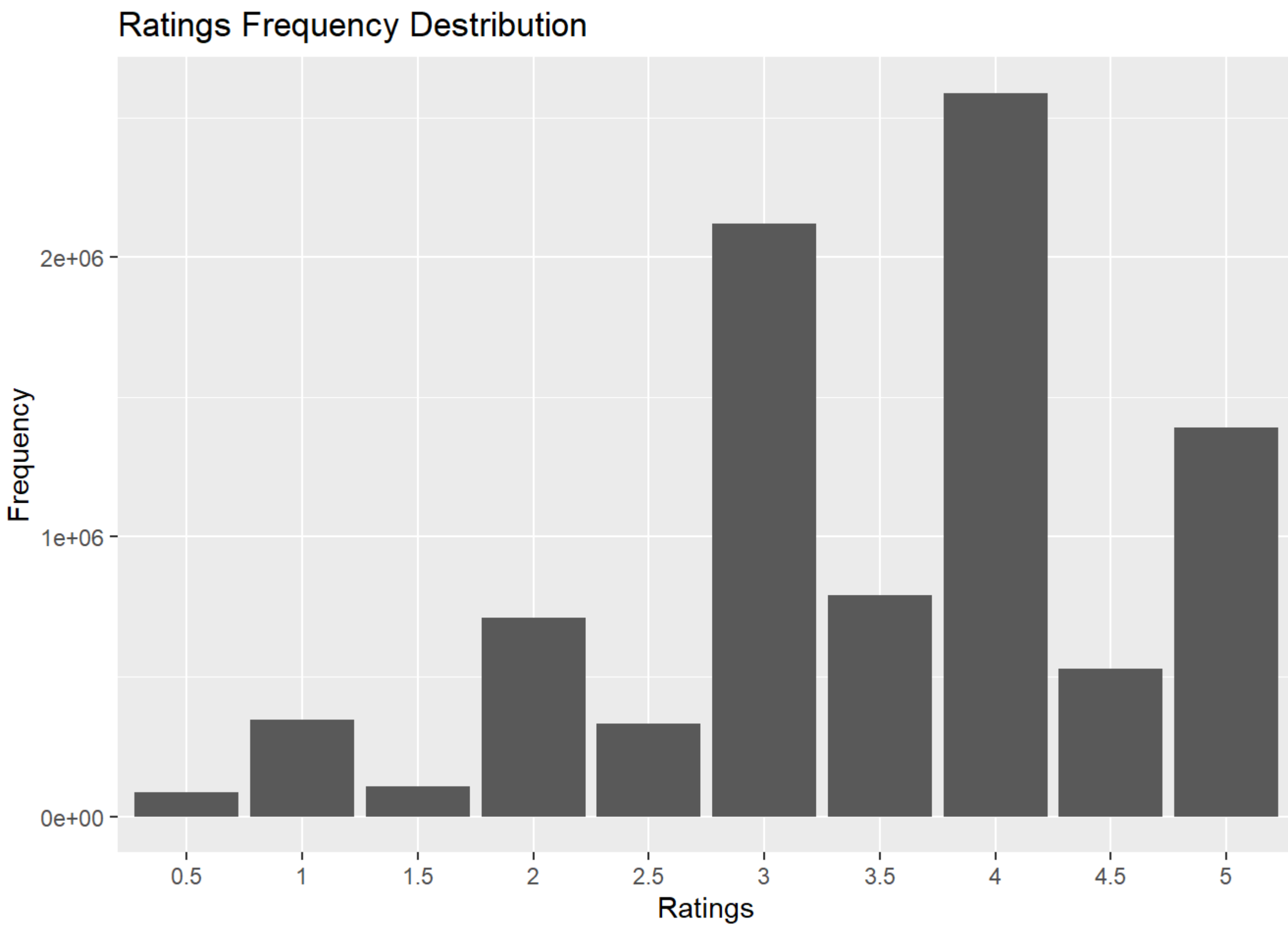
Ratings Destribution

```
# Check frequencies of ratings unique values
table_rating <- as.data.frame(table(edx$rating))
colnames(table_rating) <- c('Ratings', 'Frequency')
knitr::kable(table_rating)
```

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

Graph plot of the Ratings Destribution:

```
# Frequency plot of the ratings
table_rating %>% ggplot(aes(Ratings, Frequency)) +
geom_bar(stat = 'identity') +
labs(x='Ratings', y='Frequency') +
ggtitle('Ratings Frequency Destribution')
```



From this chart it can be seen that most ratings are between 3 and 4. As well, more users give ratings above 2.5.

Top 20 Movies (by views)

```
# Top movies by number of views
tmovies <- edx %>% select(title) %>%
group_by(title) %>%
summarize(count=n()) %>%
arrange(desc(count))
# Print top_movies
knitr::kable(head(tmovies,20))
```

title	count
Pulp Fiction	31362
Forrest Gump	31079
Silence of the Lambs, The	30382
Jurassic Park	29360
Shawshank Redemption, The	28015
Braveheart	26212
Fugitive, The	26020
Terminator 2: Judgment Day	25984

title	count
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars)	25672
Batman	24585
Apollo 13	24284
Toy Story	23790
Independence Day (a.k.a. ID4)	23449
Dances with Wolves	23367
Schindler’s List	23193
True Lies	22823
Star Wars: Episode VI - Return of the Jedi	22584
12 Monkeys (Twelve Monkeys)	21891
Usual Suspects, The	21648
Fargo	21395

The most viewed movie is **Pulp Fiction** with ratings.

Average Movie Ratings (Top 20 by Rating AVG)

```
# Top movies by rating average
rating_avg <- edx %>%
  select(title, rating) %>%
  group_by(title) %>%
  summarise(count = n(), avg = mean(rating), min = min(rating), max = max(rating)) %>%
  arrange(desc(avg))
# Print top_movies
knitr::kable(head(rating_avg,20))
```

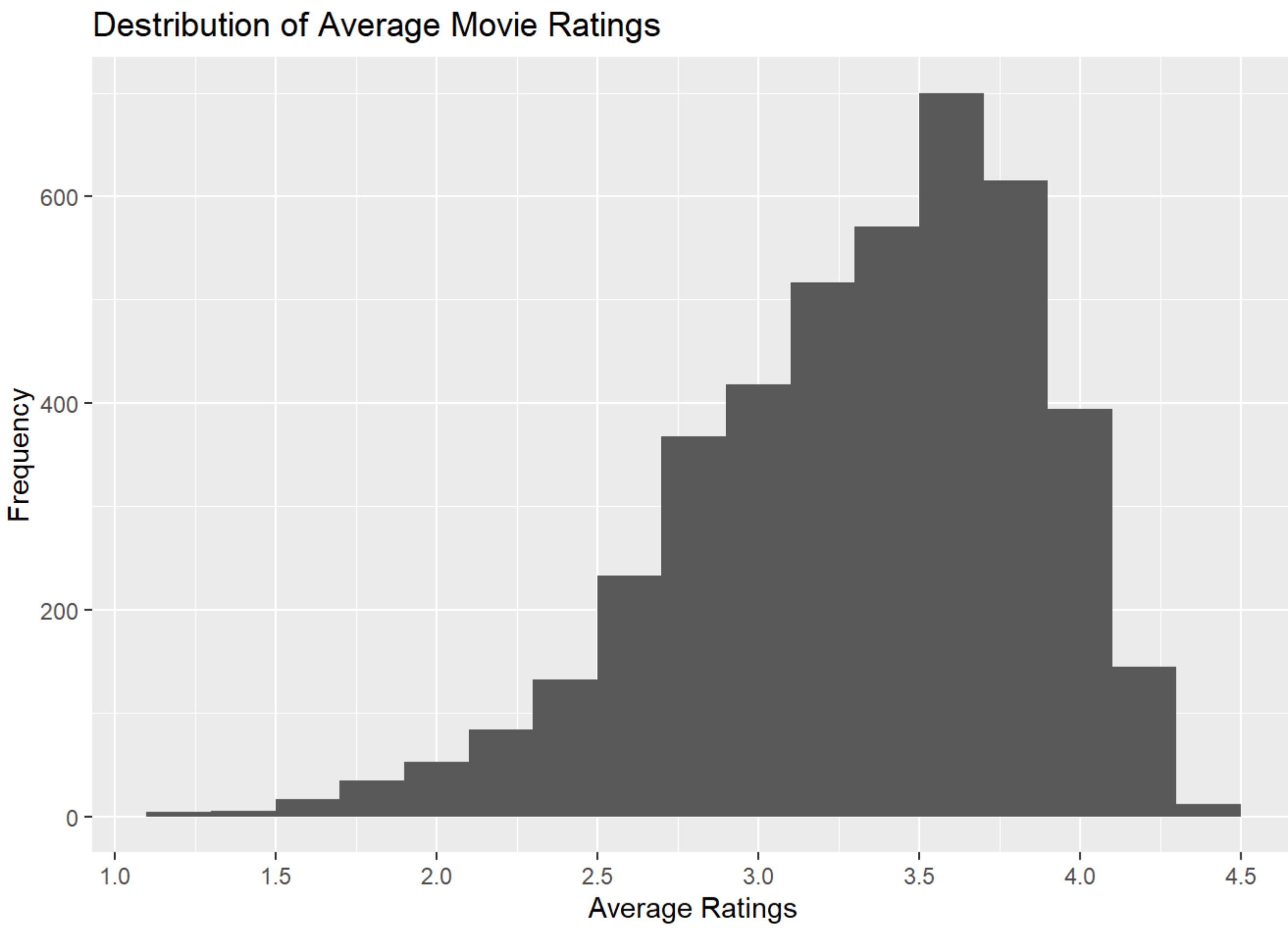
title	count	avg	min	max
Blue Light, The (Das Blaue Licht)		15.000000	5.0	5.0
Fighting Elegy (Kenka erejii)		15.000000	5.0	5.0
Hellhounds on My Trail		15.000000	5.0	5.0
Satan’s Tango (SÃ¡tÃ¡ntangÃ³)		25.000000	5.0	5.0
Shadows of Forgotten Ancestors		15.000000	5.0	5.0
Sun Alley (Sonnenallee)		15.000000	5.0	5.0
Constantine’s Sword		24.750000	4.5	5.0
Human Condition II, The (Ningen no joken II)		44.750000	4.5	5.0
Human Condition III, The (Ningen no joken III)		44.750000	4.5	5.0
Who’s Singin’ Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva)		44.750000	4.0	5.0
Class, The (Entre les Murs)		34.666667	4.0	5.0
I’m Starting From Three (Ricomincio da Tre)		34.666667	4.5	5.0
Bad Blood (Mauvais sang)		14.500000	4.5	4.5
Demon Lover Diary		14.500000	4.5	4.5
End of Summer, The (Kohayagawa-ke no aki)		34.500000	4.0	5.0
Kansas City Confidential		14.500000	4.5	4.5
Ladrones		14.500000	4.5	4.5
Life of Oharu, The (Saikaku ichidai onna)		34.500000	4.0	5.0
Man Named Pearl, A		14.500000	4.5	4.5
Mickey		14.500000	4.5	4.5

The result illustrate what seems to be an anomoly where the dataset is skewed by movies that received just a few ratings but those ratings averaged high enough to put the movie in the top 20s. These are outliers and should be dealth with by excluding them from the dataset. As a safe gaurd, only movies with more than 200 ratings will be considered.

```
# Top movies by rating average
rating_avg_200 <- edx %>%
select(title, rating) %>%
group_by(title) %>%
summarise(count = n(), avg = mean(rating), min = min(rating), max = max(rating)) %>%
filter(count > 200) %>%
arrange(desc(avg))
# Print top_movies
knitr::kable(head(rating_avg_200,20))
```

title	count	avg	min	max
Shawshank Redemption, The	28015	4.455131	0.5	5
Godfather, The	17747	4.415366	0.5	5
Usual Suspects, The	21648	4.365854	0.5	5
Schindler’s List	23193	4.363493	0.5	5
Casablanca	11232	4.320424	0.5	5
Rear Window	7935	4.318651	0.5	5
Sunset Blvd. (a.k.a. Sunset Boulevard)	2922	4.315880	0.5	5
Third Man, The	2967	4.311426	0.5	5
Double Indemnity	2154	4.310817	0.5	5
Paths of Glory	1571	4.308721	0.5	5
Seven Samurai (Shichinin no samurai)	5190	4.306744	0.5	5
Godfather: Part II, The	11920	4.301971	0.5	5
Dark Knight, The	2353	4.297068	0.5	5
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb	10627	4.295333	0.5	5
One Flew Over the Cuckoo’s Nest	13014	4.293261	0.5	5
Lives of Others, The (Das Leben der Anderen)	1108	4.291065	0.5	5
Yojimbo	1528	4.281741	0.5	5
Wallace & Gromit: The Wrong Trousers	7167	4.275429	0.5	5
Wallace & Gromit: A Close Shave	5690	4.275308	0.5	5
M	1926	4.274662	0.5	5

```
rating_avg_200 %>%
  ggplot(aes(x= avg, fill = count)) +
  geom_histogram( binwidth = 0.2) +
  scale_x_continuous(breaks=seq(0, 5, by= 0.5)) +
  labs(x='Average Ratings', y='Frequency') +
  ggtitle('Destribution of Average Movie Ratings')
```

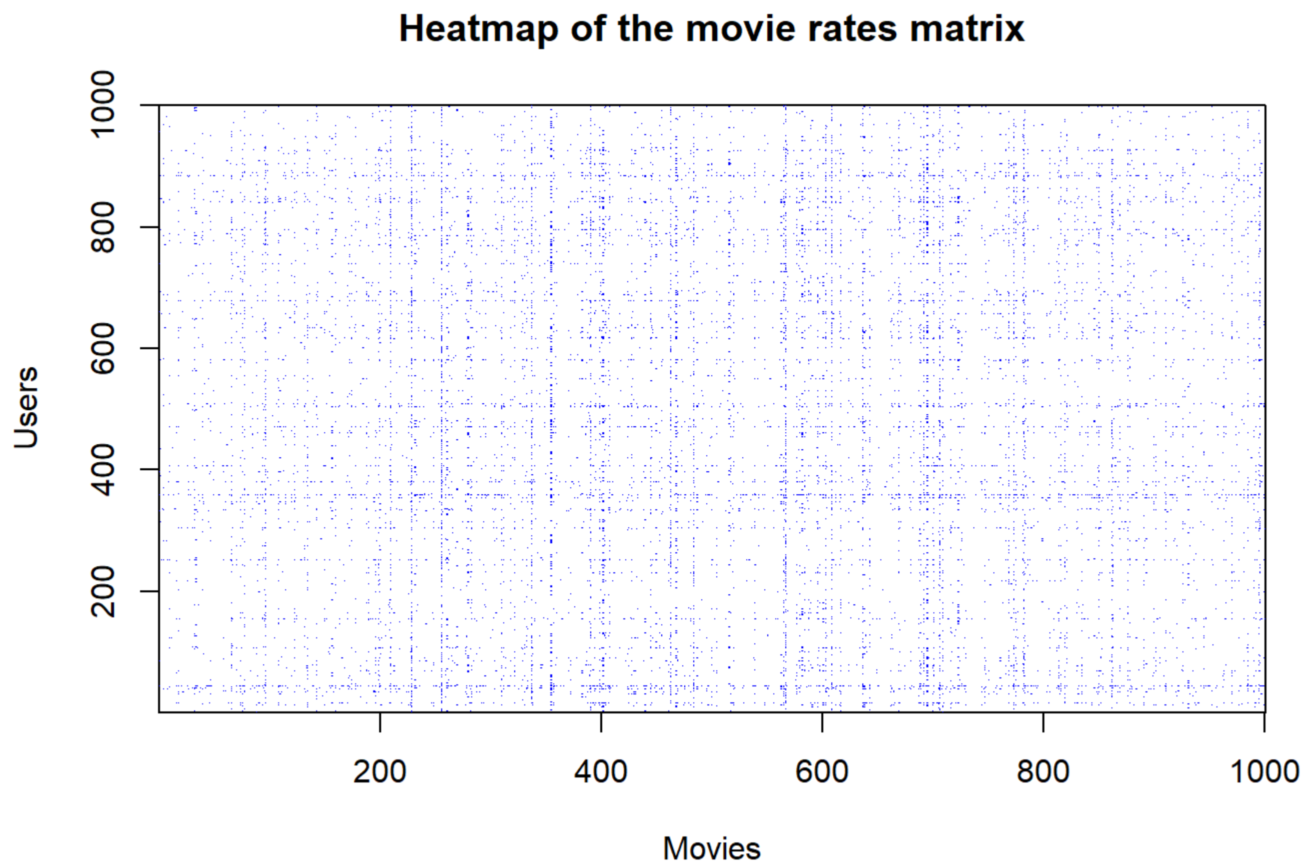


Again the largest number of rantings are between 3 and 4.

Data Heat Map

The figure below shows the matrix for a random sample of 1000 movies and 1000 users with blue indicating a user/movie combination for which we have a rating. No distinct pattern can be deduced from the plot. Since the chart is just displaying some random users and items, the next step is to visualize only the users who have seen many movies and the movies that have been seen by many users.

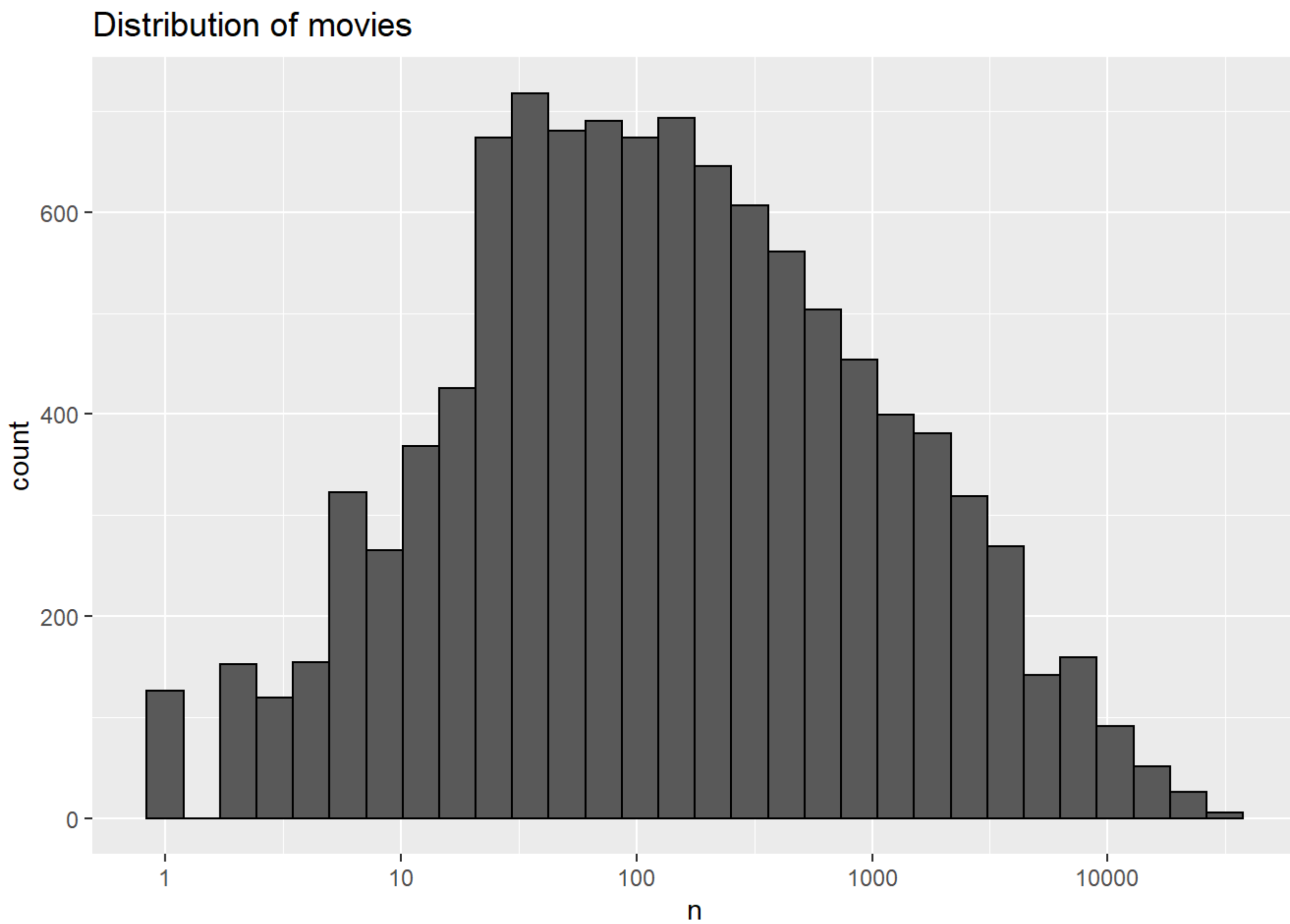
```
# We create a copy of existing edx
edx_copy <-edx
# Sample of 100 users
users <- sample(unique(edx_copy$userId), 1000)
edx_copy %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 1000)) %>%
  as.matrix() %>% t(.) %>%
  image(1:1000, 1:1000,., col = 'blue', xlab='Movies', ylab='Users', main = 'Heatmap of the movi
e rates matrix')
```



Movie to User Destribution

Some movies are rated more often than others. On the plot we can see distribution of movies based on user ratings.

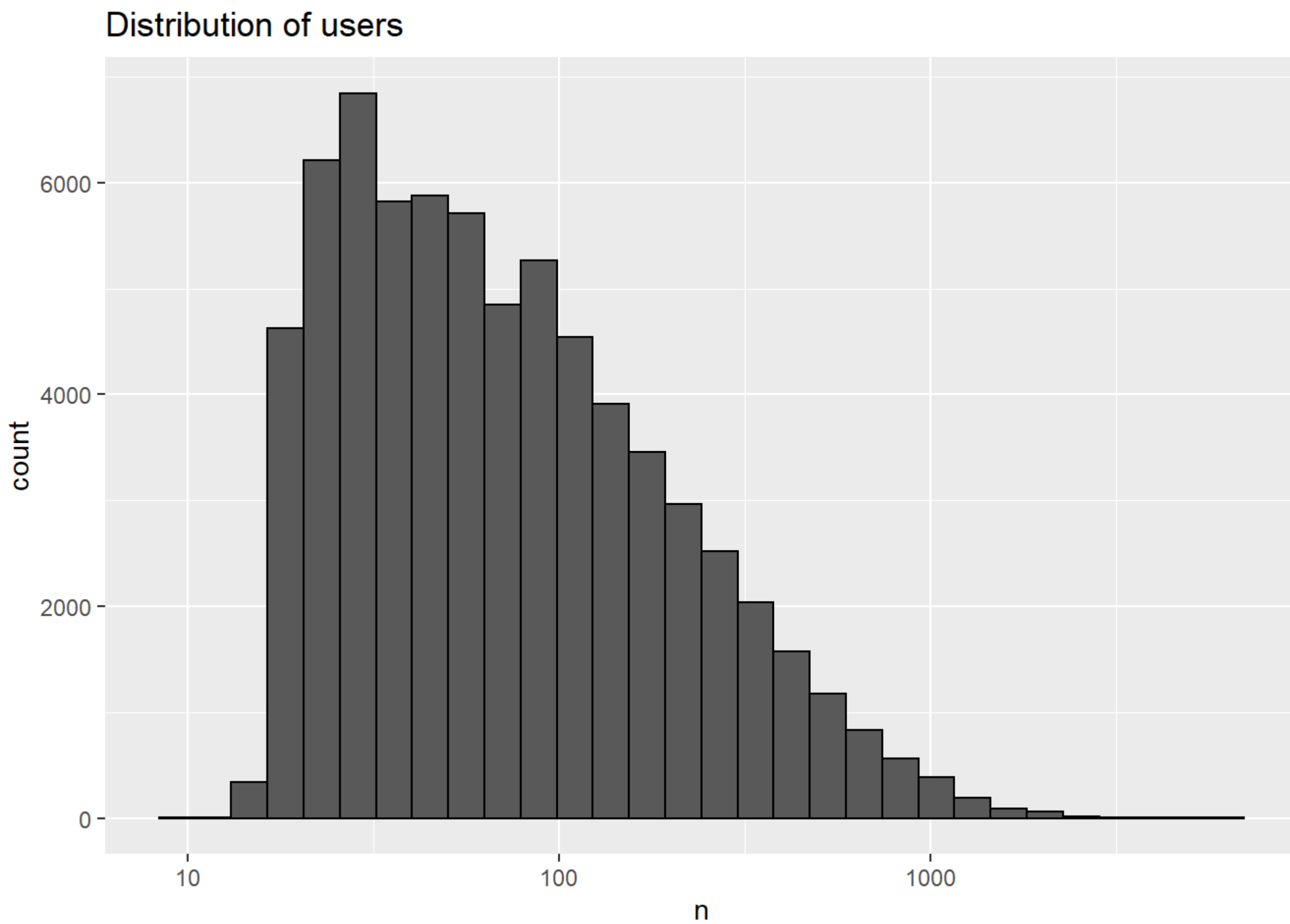
```
edx %>%  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = 'black') +
  scale_x_log10() +
  ggtitle('Distribution of movies')
```



Movie to User Activity

Some users are more active than others at rating movies:

```
edx %>%  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = 'black') +
  scale_x_log10() +
  ggtitle('Distribution of users')
```



Destribution by Genre

The top 20 genres based on number of viewers is shown below. Note that this is based on the raw genre information which in many instances is made up of multiple movie types. Lateron, the genre will be broken out into individual entries and analyzed.

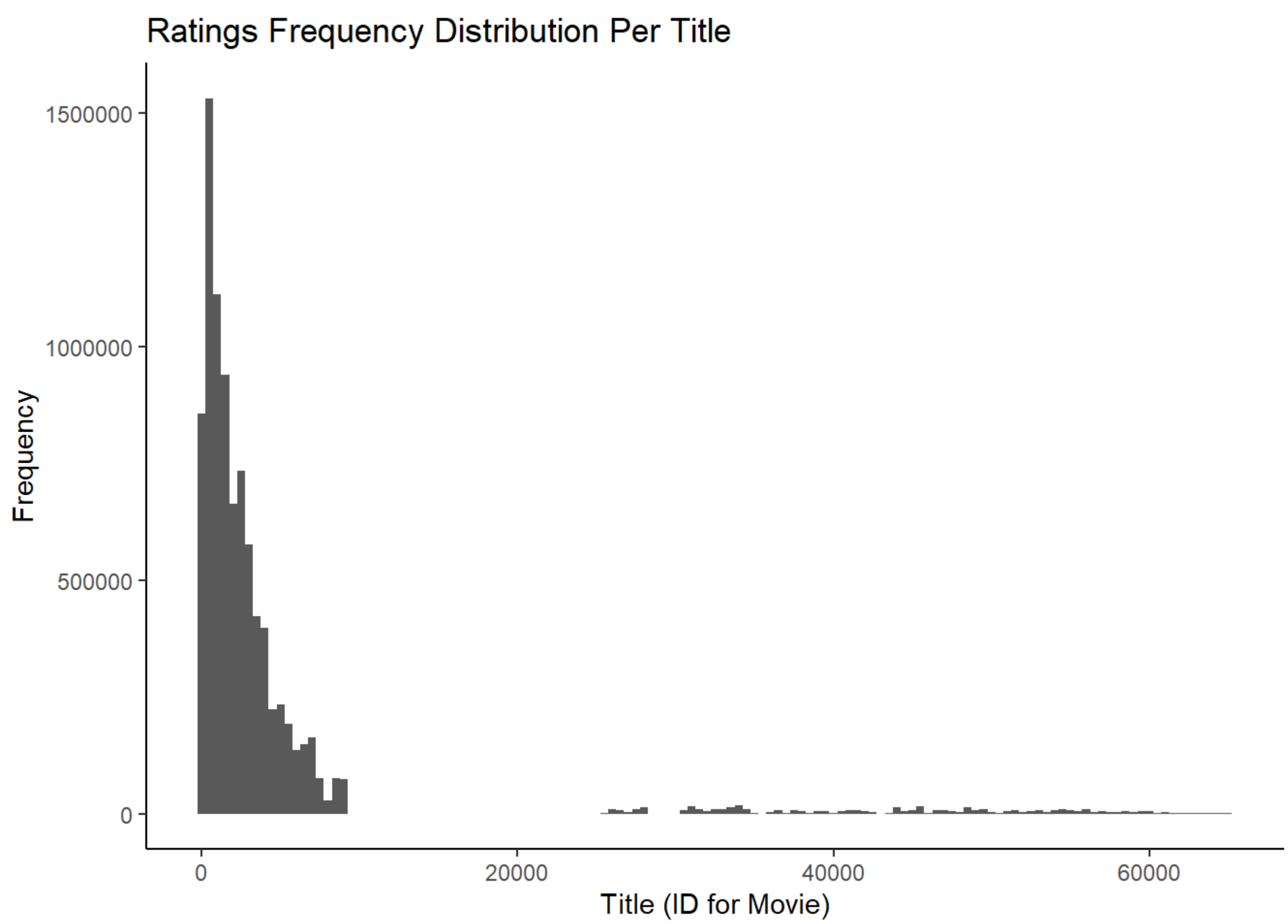
```
# Top movies by number of views
tgen <- edx %>% select(genres) %>%
group_by(genres) %>%
summarize(count=n()) %>%
arrange(desc(count))
# Print top_movies
knitr::kable(head(tgen,20))
```

genres	count
Drama	733296
Comedy	700889
Comedy|Romance	365468
Comedy|Drama	323637
Comedy|Drama|Romance	261425
Drama|Romance	259355
Action|Adventure|Sci-Fi	219938
Action|Adventure|Thriller	149091
Drama|Thriller	145373

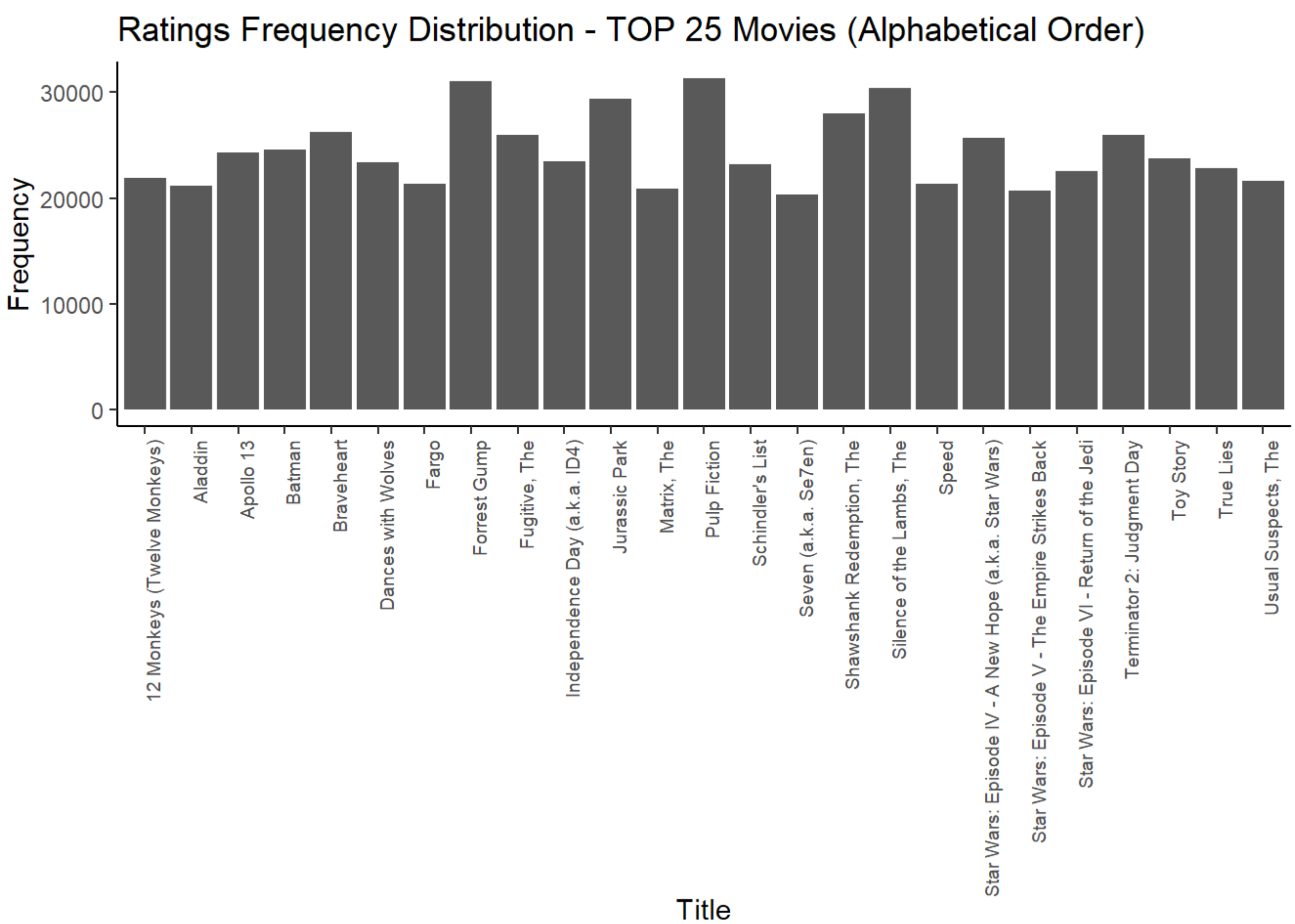
genres	count
Crime|Drama	137387
Drama|War	111029
Crime|Drama|Thriller	106101
Action|Adventure|Sci-Fi|Thriller	105144
Action|Crime|Thriller	102259
Action|Drama|War	99183
Action|Thriller	96535
Action|Sci-Fi|Thriller	95280
Thriller	94662
Horror|Thriller	75000
Comedy|Crime	73286

Numbers of Ratings per Movie

```
ggplot(edx, aes(movieId)) +
theme_classic()  +
geom_histogram(binwidth=500) +
labs(title = "Ratings Frequency Distribution Per Title",
     x = "Title (ID for Movie)",
     y = "Frequency")
```



```
edx %>%
   group_by(title) %>%
   summarise(count = n()) %>%
   arrange(desc(count)) %>%
   head(n=25) %>%
   ggplot(aes(title, count)) +
   theme_classic()  +
   geom_col() +
   theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 7)) +
   labs(title = "Ratings Frequency Distribution - TOP 25 Movies (Alphabetical Order)",
        x = "Title",
        y = "Frequency")
```



Genre Analysis

Rating Distribution per Genre

Due to the coupled nature in which the genres are store, we'll first uncouple each genre and then proceed to analyze the number of views and ratings.

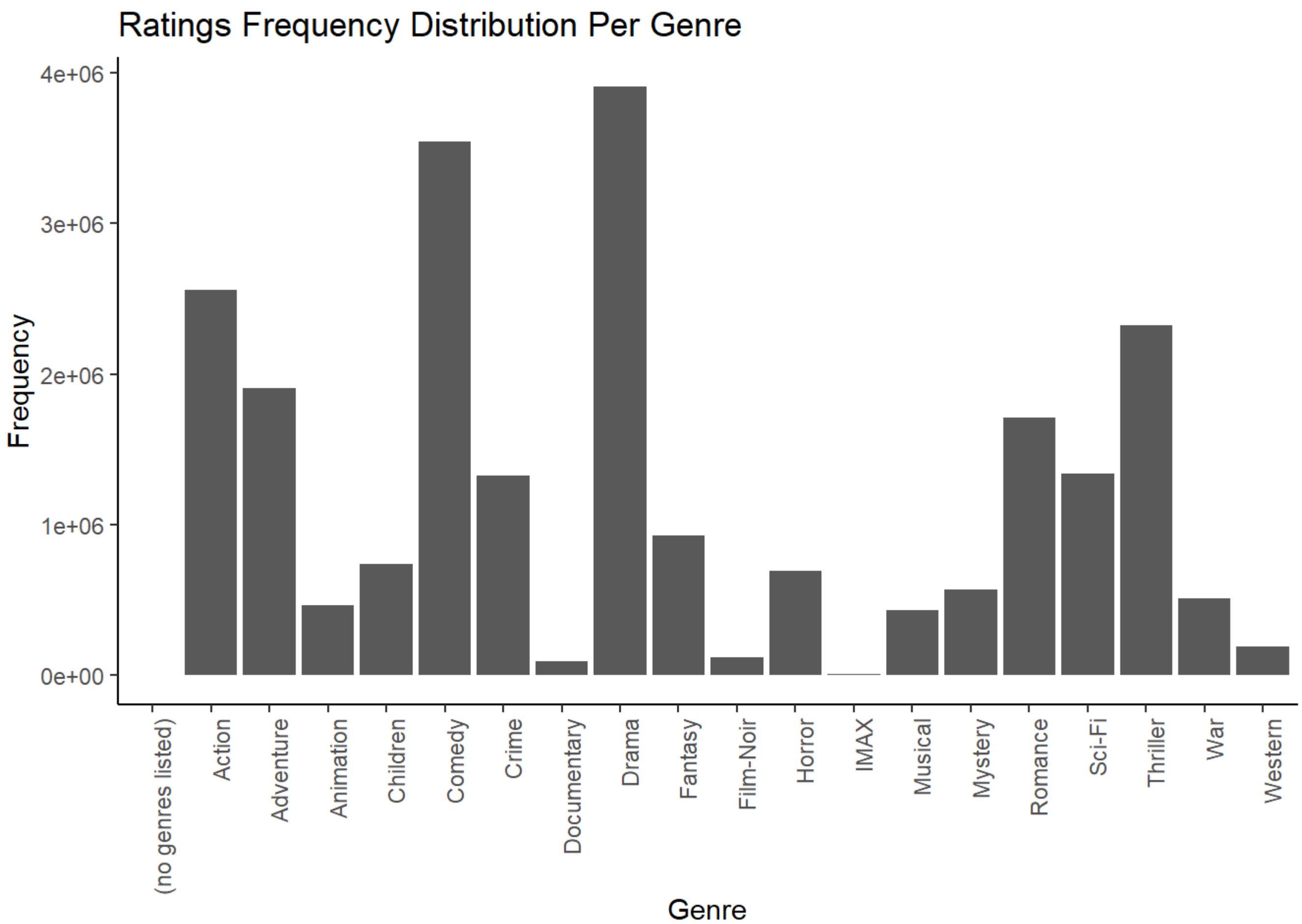
```
# Extract the genre in edx

edx <- edx %>%
   mutate(genre = fct_explicit_na(genres,
                                    na_level = "(no genres listed)")
          ) %>%
   separate_rows(genre,
                 sep = "\\|")
```

```
# Extract the genre in validation

validation <- validation %>%
   mutate(genre = fct_explicit_na(genres,
                                    na_level = "(no genres listed)")
          ) %>%
   separate_rows(genre,
                 sep = "\\|")
```

```
edx %>%
   group_by(genre) %>%
   summarise(count = n()) %>%
   ggplot(aes(genre, count)) +
   theme_classic()  +
   geom_col() +
   theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
   labs(title = "Ratings Frequency Distribution Per Genre",
        x = "Genre",
        y = "Frequency")
```



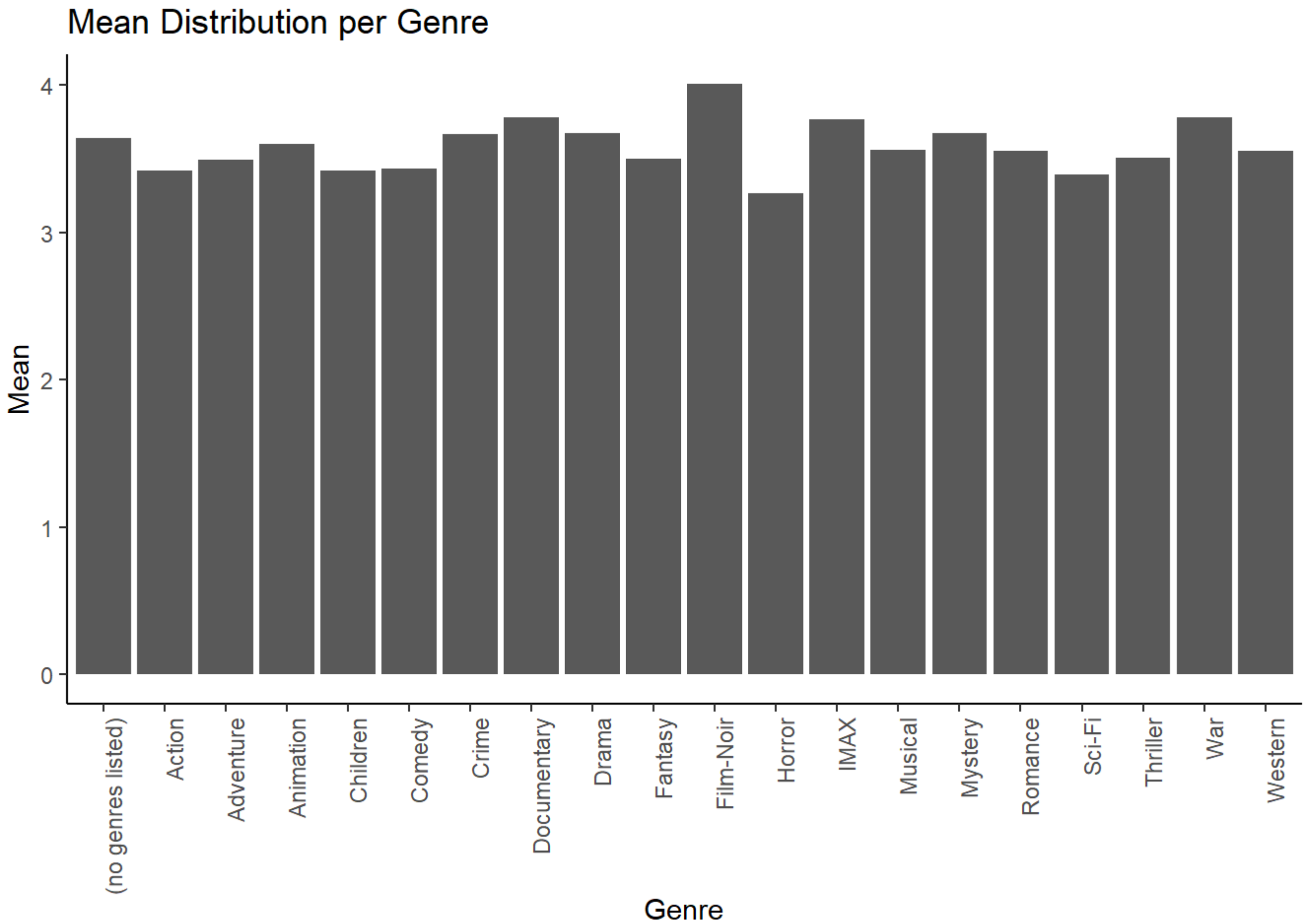

```
edx %>%
   group_by(genre) %>%
   summarise(count = n()) %>%
   arrange(desc(count)) %>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                 position = "center",
                 font_size = 10,
                 full_width = FALSE)
```

genre	count
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994

genre	count
Horror	691485
Mystery	568332
War	511147
Animation	467168
Musical	433080
Western	189394
Film-Noir	118541
Documentary	93066
IMAX	8181
(no genres listed)	7

Mean Distribution per Genre

```
edx %>%
   group_by(genre) %>%
   summarise(mean = mean(rating)) %>%
   ggplot(aes(genre, mean)) +
   theme_classic()  +
   geom_col() +
   theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
   labs(title = "Mean Distribution per Genre",
        x = "Genre",
        y = "Mean")
```




```
edx %>%
   group_by(genre) %>%
   summarise(mean = mean(rating)) %>%
   arrange(desc(mean)) %>%
   head(n=35) %>%
   kable() %>%
   kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
               position = "center",
               font_size = 10,
               full_width = FALSE)
```

genre	mean
Film-Noir	4.011625
Documentary	3.783487
War	3.780813
IMAX	3.767693
Mystery	3.677001
Drama	3.673131
Crime	3.665925
(no genres listed)	3.642857
Animation	3.600644
Musical	3.563305

genre	mean
Western	3.555918
Romance	3.553813
Thriller	3.507676
Fantasy	3.501946
Adventure	3.493544
Comedy	3.436908
Action	3.421405
Children	3.418715
Sci-Fi	3.395743
Horror	3.269815

##Model Building

Residual Mean Square Error (RMSE) is an error function. “RMSE can be interpreted as the standard deviation of the unexplained variance, and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate better fit. RMSE is a good measure of how accurately the model predicts the response, and is the most important criterion for fit if the main purpose of the model is prediction.” (Grace-Martin K: Assessing the Fit of Regression Models)

In this application RMSE measures the typical error we make when predicting the movie rating. If the RMSE error is larger than 0.8775, it means our typical error is larger than that required for this assignment and hence deem the algorithm to be a bad fit. Several models will be compared in this analysis. The formula used for RMSE is:

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

where N is the number of users, movie ratings, and the sum incorporating the total combinations.

We’ll first test models ranging from simplest to most optimal to ascertain the best model to use for the movie recommendation system.

Model 1 : Computing predicted ratings for all movies regardless of user (Naive)

Our first model assumes that the same rating for all movies and users with all the differences can be explained by random variation. Formula:

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

where $\varepsilon_{i,u}$ are the independent errors centered at 0 and μ the **true** rating for all movies.

```
# Ratings for all movies
mu_hat <- mean(edx$rating)
mu_hat
```

```
## [1] 3.527019
```

The magnitude of a typical residual can give us a sense of generally how close our estimates are.Least squares fitting procedure guarantee that the mean of the residuals is zero. Thus, it makes more sense to compute **root mean squared error (RMSE)**. We will use function **RMSE**:

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

```
# RMSE calculation
simple_model_rmse <- RMSE(validation$rating, mu_hat)
simple_model_rmse
```

```
## [1] 1.052558
```

We see that our first evaluation for RMSE is 1.0525579 a little bit higher that 1. We will continue comparing different approaches to check if we can get a lower value for RMSE.

Below we create a results table to store all RMSE values we get in different approaches:

```
rmse_values <- tibble(method = 'Simple model RMSE', RMSE = simple_model_rmse)
knitr::kable(rmse_values)
```

method	RMSE
Simple model RMSE	1.052558

Model 2 : Computing predicted ratings for all movies based on movie effects

During data exploration, we noticed that some movies are just generally rated higher than others. We will add in our previously built simple model the term b_i to represent average ranking for movie i:

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

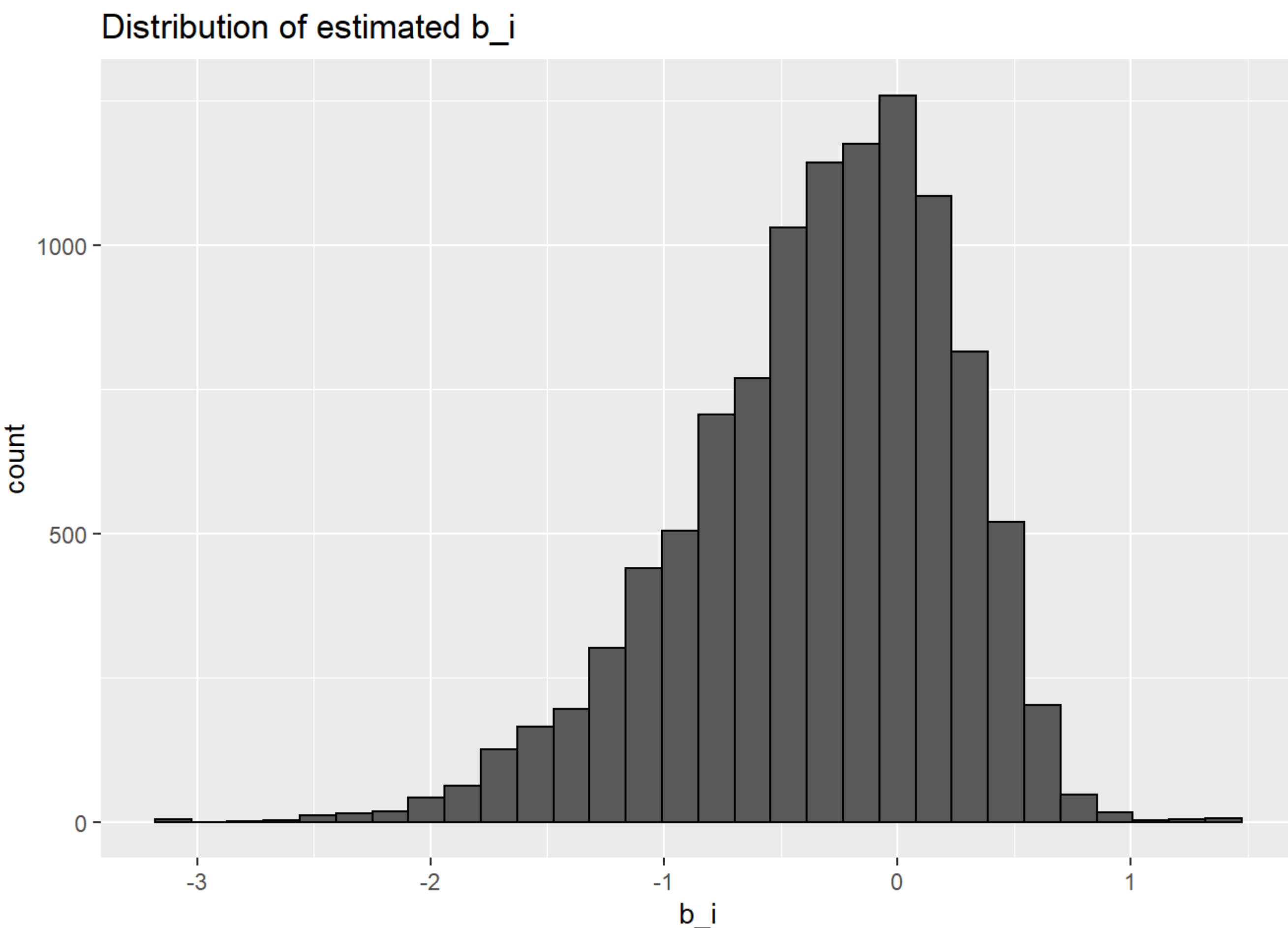
We will refer to the bs as **effects** or **bias**, movie-specific effect. We will estimate b_i-s using **least square method**. Function **lm** makes this possible, but it can be very slow so we will procced by taking Professor Rafael Irizarry's advice.

In this particular situation, we know that the least squares estimate $\hat{b}_i$ is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i. So we can compute them this way:

```
#Compute the average of all ratings of the edx set
mu <- mean(edx$rating)
#Compute b_i
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

Let's plot now the estimated b_i-s distribution:

```
#Plot b_i distribution
movie_avgs %>%
ggplot(aes(b_i)) +
geom_histogram(bins = 30, color = 'black') +
ggtitle('Distribution of estimated b_i')
```



From the plot we can see that these estimates vary substantially.

```
# Predict bi
model_2_pred <- mu + validation %>%
left_join(movie_avgs, by='movieId') %>%
.$b_i
movie_effect_rmse <- RMSE(validation$rating, model_2_pred)
# Enter RMSE value in table
rmse_values <- bind_rows(rmse_values,
                          tibble(method='Movie Effect Model',
                                 RMSE = movie_effect_rmse))
knitr::kable(rmse_values)
```

method	RMSE
Simple model RMSE	1.052558
Movie Effect Model	0.941070

Our model has improved with movie effect added. Let’s try to get it better.

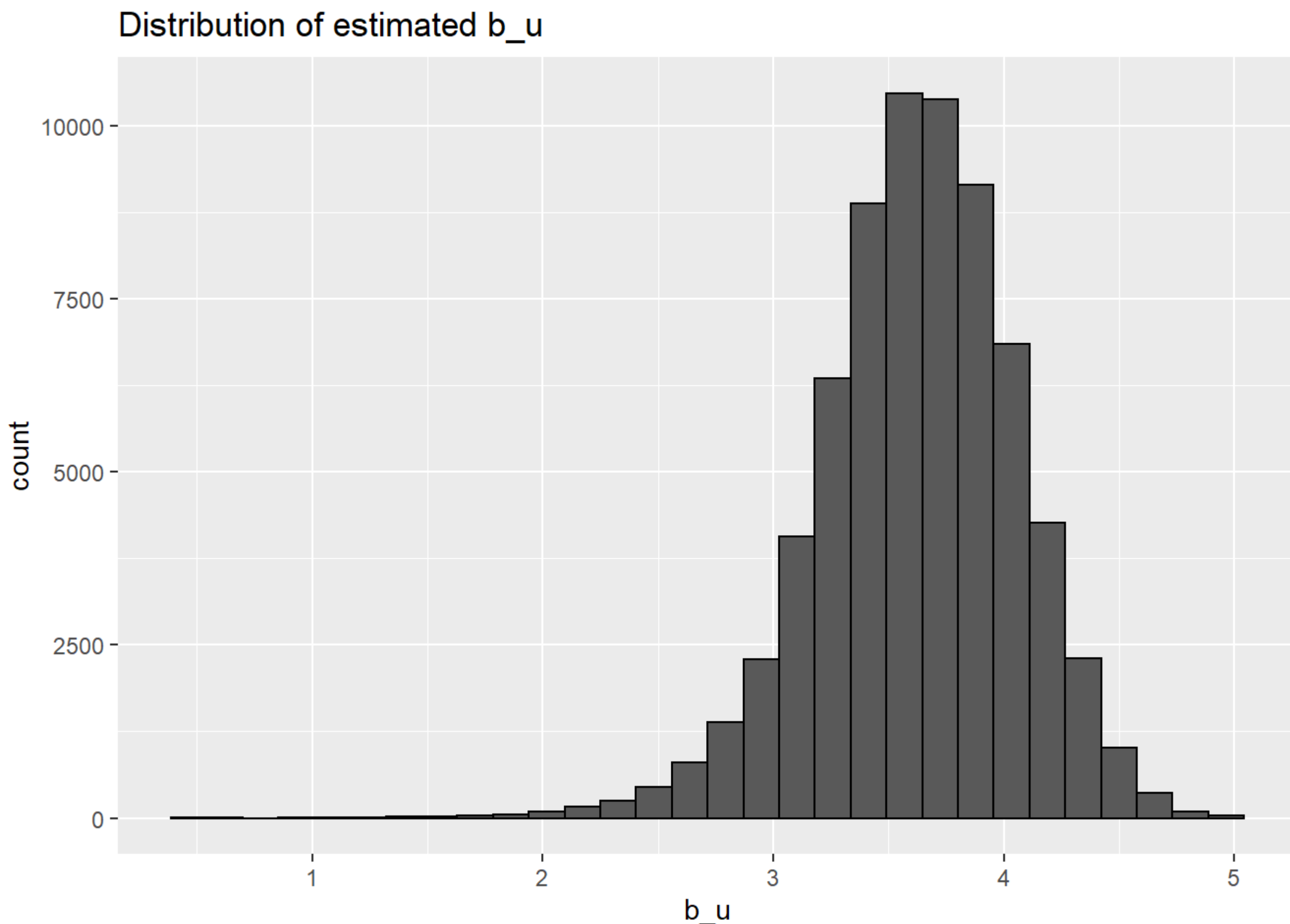
Model 3 : Computing predicted ratings for all movies based on movie and user effects

Another improvement to our model may be:

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

where b_u is a user-specific effect. Why should we think that adding user effect can lead to model improvement? Let’s compute the average rating for user u for those that have rated over 100 movies and plot the estimated b_u-s distribution:

```
# Compute average rating for user u who rated more than 100 movies
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = 'black') +
  ggtitle('Distribution of estimated b_u')
```



We notice that there is substantial variability across users as well so we proceed with model fitting. Let's compute an approximation by computing $\hat{\mu}$ and $\hat{b}_i$ and estimating $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$:

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

Now let's see if RMSE improves:

```
# Predicted ratings
model3_pred <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
movie_user_effect <- RMSE(validation$rating, model3_pred)
rmse_values <- bind_rows(rmse_values,
                         tibble(method='Movie + User Effects Model',
                                RMSE = movie_user_effect))
knitr::kable(rmse_values)
```

method	RMSE
Simple model RMSE	1.052558
Movie Effect Model	0.941070
Movie + User Effects Model	0.863366

Our RMSE is lower now but still not at the target we are looking for.

Model 4 : Computing predicted ratings for all movies based on movie and user effects and genre

Another improvement to our model may be:

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

where b_u is a user-specific effect and b_g the effect based on genre. Taking the average rating for user u for those that have rated over 100 movies and plot the estimated b_u-s distribution:

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

Now let’s see if RMSE improves even more.

```
# Ratings on validation dataset

model4_pred <- validation %>%
   left_join(movie_avgs, by='movieId') %>%
   left_join(user_avgs, by='userId') %>%
   left_join(genre_avgs, by='genre') %>%
   mutate(pred2 = mu + b_i + b_u + b_g) %>%
   pull(pred2)

# Adding the results to the results dataset
movie_user_genre_effect <- RMSE(validation$rating, model4_pred)
rmse_values <- bind_rows(rmse_values,
                         tibble(method='Movie + User + Genre Effects Model',
                                RMSE = movie_user_genre_effect))
knitr::kable(rmse_values)
```

method	RMSE
Simple model RMSE	1.0525579
Movie Effect Model	0.9410700
Movie + User Effects Model	0.8633660
Movie + User + Genre Effects Model	0.8632723

Regularization

Regularization helps to choose preferred model complexity, so that the model is better at predicting. It allows for reduced errors caused by movies with outliers that can influence the prediction and skew the error metric. The method uses a penalty term or tuning parmeter, λ, to minimise the RMSE. Modifying b_i and b_u for movies with limited ratings. Formula:

Let's explore problems in our first model, using only movie effects b_i:

```
validation %>%
left_join(movie_avgs, by='movieId') %>%
mutate(residual = rating - (mu + b_i)) %>%
arrange(desc(abs(residual))) %>%
select(title, residual) %>% slice(1:10)
```

```
##                         title  residual
## 1            PokÃ©mon Heroes  3.970803
## 2            PokÃ©mon Heroes  3.970803
## 3  Shawshank Redemption, The -3.955131
## 4  Shawshank Redemption, The -3.955131
## 5  Shawshank Redemption, The -3.955131
## 6             Godfather, The -3.915366
## 7             Godfather, The -3.915366
## 8             Godfather, The -3.915366
## 9             Godfather, The -3.915366
## 10            Godfather, The -3.915366
```

The predictions seems large, hence, we'll examine the top 10 worst and best movies based on b_i.

```
# merged database of MOvie and Title
merge_db <- edx %>%
select(movieId, title) %>%
distinct()
```

Top 10 Best Movies

Here are the 10 best movies according to our estimate and how often they were rated (based on prediction):

```
# top 10 best movies based on b_i
movie_avgs %>% left_join(merge_db, by="movieId") %>%
arrange(desc(b_i)) %>%
select(title, b_i) %>%
slice(1:10)
```

```
## # A tibble: 10 x 2
##    title                                                          b_i
##    <chr>                                                        <dbl>
##  1 Hellhounds on My Trail                                        1.47
##  2 Satan's Tango (SÃ¡tÃ¡ntangÃ³)                                  1.47
##  3 Shadows of Forgotten Ancestors                                1.47
##  4 Fighting Elegy (Kenka erejii)                                 1.47
##  5 Sun Alley (Sonnenallee)                                       1.47
##  6 Blue Light, The (Das Blaue Licht)                             1.47
##  7 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to ta~  1.22
##  8 Human Condition II, The (Ningen no joken II)                  1.22
##  9 Human Condition III, The (Ningen no joken III)                1.22
## 10 Constantine's Sword                                           1.22
```

```
validation %>% count(movieId) %>%
left_join(movie_avgs) %>%
left_join(merge_db, by="movieId") %>%
arrange(desc(b_i)) %>%
select(title, b_i, n) %>%
slice(1:10)
```

```
## Joining, by = "movieId"
```

```
## # A tibble: 10 x 3
##    title                                                        b_i     n
##    <chr>                                                      <dbl> <int>
##  1 Hellhounds on My Trail                                     1.47      1
##  2 More                                                       1.19      3
##  3 Valerie and Her Week of Wonders (Valerie a tÃ½den divu)      0.973     2
##  4 Kansas City Confidential                                   0.973     4
##  5 Shawshank Redemption, The                                  0.928  3111
##  6 Red Desert, The (Deserto rosso, Il)                        0.890     1
##  7 Godfather, The                                             0.888  4134
##  8 Man Who Planted Trees, The (Homme qui plantait des arbres, ~ 0.873     4
##  9 Usual Suspects, The                                        0.839  7167
## 10 Schindler's List                                           0.836  5168
```

Top 10 Worst Movies

Here are the 10 worst movies according to our estimate and how often they were rated (based on prediction):

```
# top 10 worse movies based on b_i
movie_avgs %>% left_join(merge_db, by="movieId") %>%
arrange(b_i) %>%
select(title, b_i) %>%
slice(1:10)
```

```
## # A tibble: 10 x 2
##    title                            b_i
##    <chr>                          <dbl>
##  1 Besotted                       -3.03
##  2 Hi-Line, The                   -3.03
##  3 Accused (Anklaget)             -3.03
##  4 Confessions of a Superhero     -3.03
##  5 War of the Worlds 2: The Next Wave -3.03
##  6 SuperBabies: Baby Geniuses 2   -2.73
##  7 Hip Hop Witch, Da              -2.71
##  8 Disaster Movie                 -2.67
##  9 From Justin to Kelly           -2.63
## 10 Criminals                      -2.53
```

```
validation %>% count(movieId) %>%
left_join(movie_avgs) %>%
left_join(merge_db, by="movieId") %>%
arrange(b_i) %>%
select(title, b_i, n) %>%
slice(1:10)
```

```
## Joining, by = "movieId"
```

```
## # A tibble: 10 x 3
##    title                                                   b_i     n
##    <chr>                                                 <dbl> <int>
##  1 Confessions of a Superhero                            -3.03     1
##  2 War of the Worlds 2: The Next Wave                    -3.03     1
##  3 SuperBabies: Baby Geniuses 2                          -2.73     5
##  4 Disaster Movie                                        -2.67     8
##  5 From Justin to Kelly                                  -2.63    34
##  6 Criminals                                             -2.53     2
##  7 Mountain Eagle, The                                   -2.53     2
##  8 When Time Ran Out... (a.k.a. The Day the World Ended) -2.53     4
##  9 PokÃ©mon Heroes                                       -2.50    38
## 10 Roller Boogie                                         -2.49     2
```

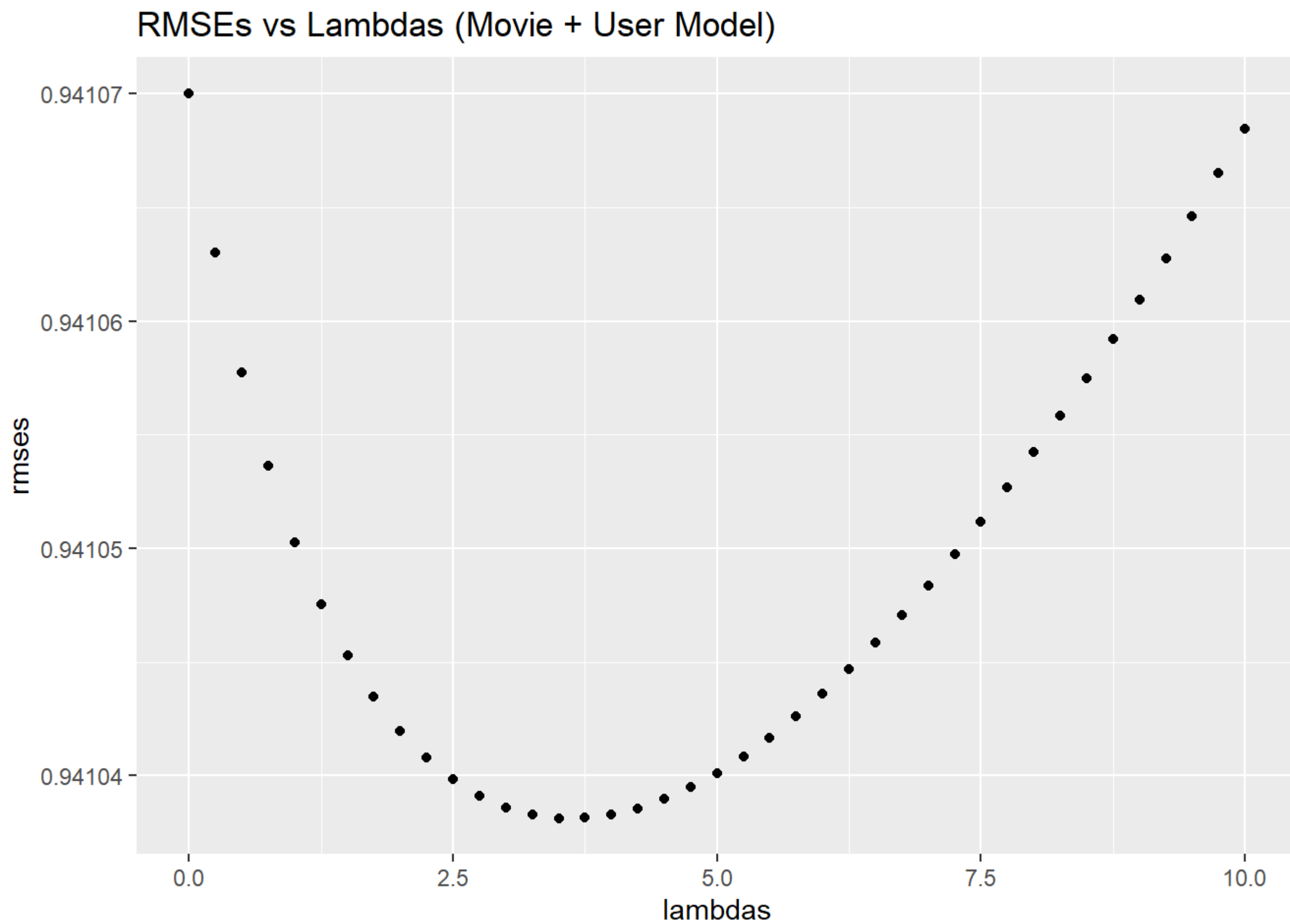
Penalized least squares

Regularization is a means of constraining the **total variability** of the effect sizes have on the prediction and is done by adding a **penalty** to the least squared equation. This **penalty** increases as b_i increases, thus leading to an optimal scenario. Using λ as a tuning parameter:

```
lambdas <- seq(0, 10, 0.25)
mu <- mean(edx$rating)
just_the_sum <- edx %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- validation %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, validation$rating))
})
```

Plotting RMSE values together with lambdas:

```
# Plot lambdas and rmse
ggplot(data.frame(lambdas = lambdas, rmses = rmses ), aes(lambdas, rmses)) +
  ggtitle('RMSEs vs Lambdas (Movie + User Model)')  +
  geom_point()
```




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

```
## [1] 3.5
```

We can use regularization for the estimate user effects as well. We are minimizing:

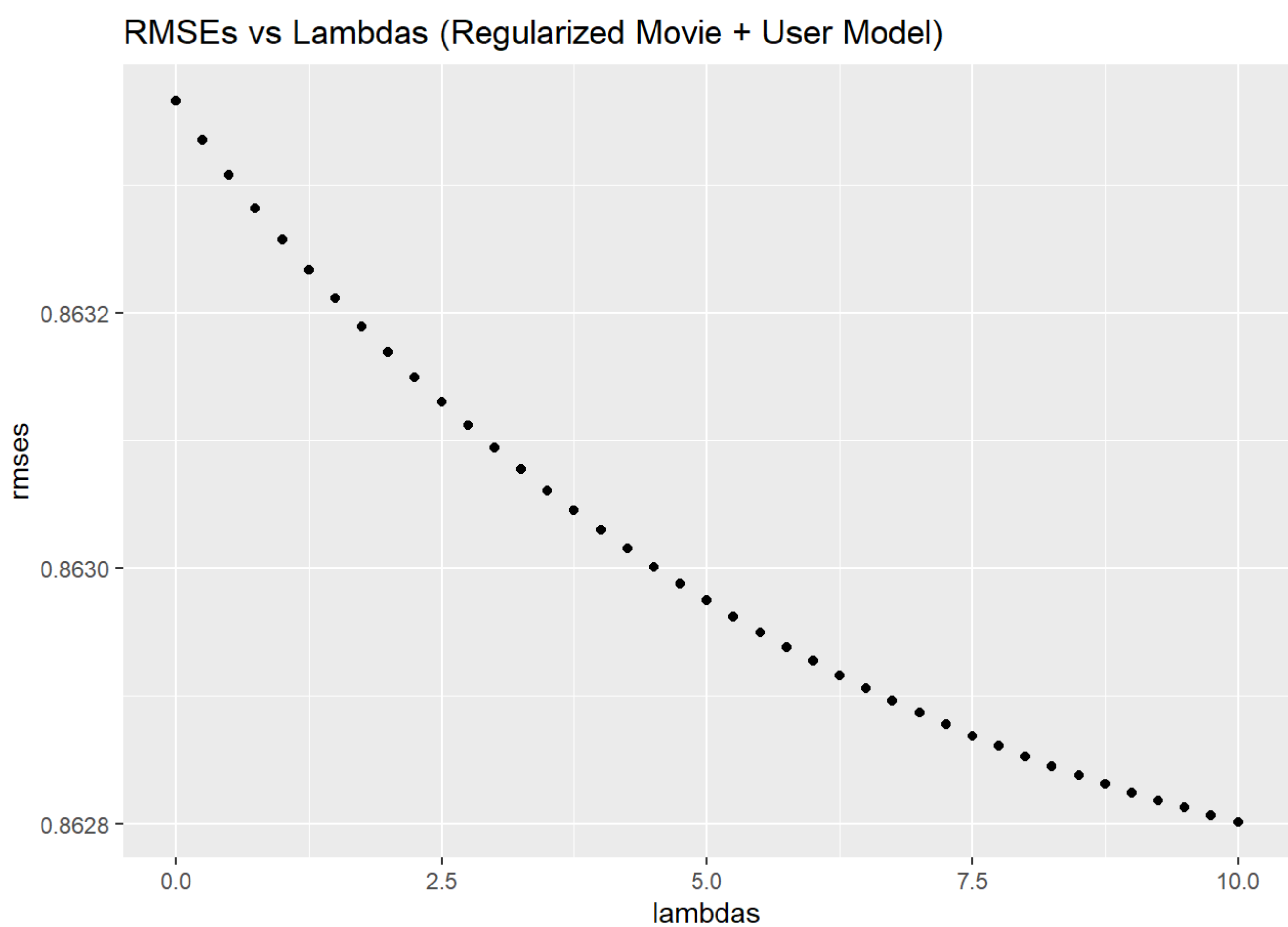
$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

The estimates that minimize this can be found similarly to what we did above. Here we use cross-validation to pick a λ:

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
mu <- mean(edx$rating)
b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

b_u <- edx %>%
    left_join(b_i, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
    predicted_ratings <-
    validation %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

return(RMSE(validation$rating, predicted_ratings))
})
ggplot(data.frame(lambdas = lambdas, rmses = rmses ), aes(lambdas, rmses)) +
  ggtitle('RMSEs vs Lambdas (Regularized Movie + User Model)') +
geom_point()
```



For the full model, the optimal λ is:

```
# Value of lambda that minimizes  RMSE
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 10
```

method	RMSE
Simple model RMSE	1.0525579
Movie Effect Model	0.9410700
Movie + User Effects Model	0.8633660
Movie + User + Genre Effects Model	0.8632723
Regularized Movie + User Effect Model	0.8628015

Conclusion

Summary table showing the RMSE values for all models:

method	RMSE
Simple model RMSE	1.0525579
Movie Effect Model	0.9410700
Movie + User Effects Model	0.8633660
Movie + User + Genre Effects Model	0.8632723
Regularized Movie + User Effect Model	0.8628015

We can see that lowest value RMSE we could achive so far is **0.8628015** which is lower than our starting goal (0.8775). **movieId** variable has a large impact on the **rmse** value. When we combined this impact with **userId** effect the **rmse** value became smaller.

Final model:

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

Appendix

1a - Initial Code privided by edX

1b - Code used in this report - MovieLens Project.R

1c - Environment

```
print("Operating System:")
```

```
## [1] "Operating System:"
```

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.0
## year           2019
## month          04
## day            26
## svn rev        76424
## language       R
## version.string R version 3.6.0 (2019-04-26)
## nickname       Planting of a Tree
```

```
print("All installed packages")
```

```
## [1] "All installed packages"
```

```
installed.packages()
```