# HarvardX PH125.9x Capstone - Movielens

Aditya Shah

## Overview/executive summary

The goal of this project is to train a machine learning algorithm or model that helps predict the rating a particular movie may receive from a particular viewer.

The model is trained using data from the Movielens dataset which contains ratings for individual movies by individual viewers (or users). Each record in the dataset is a rating for a particular movie given by a particular user.

This project is part of the PH125.9x Capstone course of the HarvardX Data Science Professional Certificate program and is based on the Netflix recommendation system case study covered in the course material for the PH125.8x Machine Learning course. As a result, some of the modeling approaches and code are based on the course material and this has been highlighted where possible in this project report.

## Introduction

Using the code provided in the course material for this project, the Movielens dataset is downloaded and split into the edx and validation datasets using a 90:10 split ratio.

### Downloading and preparing the Movielens dataset:

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

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

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
```

### Splitting Movielens into edx and validation datasets:

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

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

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

The edx dataset will be prepared and used for data exploration and analysis in Section 1.

In Section 2 the edx dataset will be split into training and test datasets using a 90:10 split ratio. The models will be trained using the training dataset and the best performing model will be tested on the validation dataset. The root mean squared error (RMSE) of the models on the validation set will be used to judge their performance.

The minimum RMSE requirement for this project is 0.9 when the final model is tested on the validation dataset. The ideal RMSE is less than 0.86490.

The result summary of the modeling techniques is provided in Section 3.

Section 4 contains the concluding remarks.

# 1. Data exploration and analysis

Section 1.1 covers the initial exploration and preparation of the edx dataset.

Section 1.2 explores the variables in the dataset in more depth.

## 1.1 Initial exploration and manipulation (preparing the dataset)

A quick look at the dataset shows that there are just over 9 million ratings with 6 variables containing details such as user ID, movie ID, rating, movie title, genres and the date timestamp:

```
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

### Converting ‘timestamp’ to date format and extracting the year

The ‘timestamp’ data is converted to the date format and the year of the rating is extracted and stored in ‘rated_year’:

```
edx <- edx %>%
  mutate(date = as_datetime(timestamp),
         rated_year = year(as_datetime(timestamp)))
```

### Extracting movie release year

The year the movie was released is extracted from its title and stored in ‘release_year’:

```
edx <- edx %>%
  mutate(release_year = str_extract(title, pattern = "(\\(\\d{4}\\))"), # includes brackets
         release_year = str_extract(release_year, pattern = "(\\d{4})"), # remove brackets
         release_year = as.numeric(release_year))
```

### Time between movie release and rating

The time difference between a movie being released and its rating is calculated and stored in ‘diff_year’:

```
edx <- edx %>%
  mutate(diff_year = rated_year - release_year)
```

The rows with negative values in ‘diff_year’ are removed:

```
# see range of differences
edx %>%
  arrange(desc(diff_year)) %>%
  select(diff_year)
```

```
##          diff_year
##       1:        93
##       2:        93
##       3:        93
##       4:        93
##       5:        93
##      ---
## 9000051:        -1
## 9000052:        -1
## 9000053:        -2
## 9000054:        -2
## 9000055:        -2
```

```
# remove rows with negative differences
edx <- edx %>%
  filter(diff_year >= 0)
```

### Weekday

The day of the week that the movie was rated on is extracted and stored in ‘weekday’:

```
edx <- edx %>%
  mutate(weekday = weekdays(date))
```

#### Remove ‘timestamp’

‘timestamp’ is removed to reduce the size of the dataset as it is no longer required:

```
edx <- edx %>%
  select(-timestamp)
```

## 1.2 Exploring data

### 1.2.1 Summary statistics

#### Ratings

```
edx %>%
  select(rating, rated_year, release_year, diff_year) %>%
  summary
```

```
##      rating        rated_year    release_year    diff_year    
##  Min.   :0.500   Min.   :1995   Min.   :1915   Min.   : 0.00  
##  1st Qu.:3.000   1st Qu.:2000   1st Qu.:1987   1st Qu.: 2.00  
##  Median :4.000   Median :2002   Median :1994   Median : 7.00  
##  Mean   :3.512   Mean   :2002   Mean   :1990   Mean   :11.98  
##  3rd Qu.:4.000   3rd Qu.:2005   3rd Qu.:1998   3rd Qu.:16.00  
##  Max.   :5.000   Max.   :2009   Max.   :2008   Max.   :93.00  
```

The mean rating is 3.5 and the ratings range between 0.5-5. The earliest rating is from 1995 and go up to 2009. The earliest movie rated was released in 1915 and the latest one in 2008. The range of time difference between a movie being released and its rating is 0-93 years; the average is 12 years.

#### Users and Movies

```
total_users <- n_distinct(edx$userId)
total_movies <- n_distinct(edx$movieId)
total_no_ratings <- nrow(edx)

avg_ratings_per_user <- nrow(edx)/n_distinct(edx$userId)
```

There are 69878 users and 10677 movies in the dataset. The average number of ratings per user is 128.79.

```
total_no_ratings/(total_users*total_movies)
```

```
## [1] 0.01206277
```

The number of ratings in the dataset are only 1% of the total number of possible ratings (i.e., if all the users in the dataset watched and rated all the movies in the dataset).

This highlights the purpose of this project: to develop an algorithm that can help predict the potential rating values for the other 99% of possible ratings.

#### Ratings per user/movie

```
# range of no. of ratings per user
ratings_per_user <- edx %>%
  group_by(userId) %>%
  summarise(n = n())

range(ratings_per_user$n)
```

```
## [1]   10 6616
```

```
# mean no. of ratings per user
avg_ratings_per_user <- nrow(edx)/n_distinct(edx$userId)
```

Users rate between 10-6616 movies with an average of 129 ratings per user.

```
# range of no. of ratings per movie
ratings_per_movie <- edx %>%
  group_by(movieId) %>%
  summarise(n = n())

range(ratings_per_movie$n)
```

```
## [1]     1 31362
```

```
# mean no. of ratings per movie
avg_ratings_per_movie <- nrow(edx)/n_distinct(edx$movieId)
```

Movies get between 1-31362 ratings with the average being 843 ratings.

#### Genres (combinations)

```
n_distinct(edx$genres)
```

```
## [1] 797
```

‘genres’ contains all the genres that apply to the movie. A movie can have multiple genres applied to it. There are 797 different genre combinations (a combination of all the individual genres applied to a movie).

## 1.2.2 Further analysis

This section looks at the ratings in the dataset and their relationship with the users, movies, genres, weekday and time difference variables.

### 1.2.2.1 Ratings

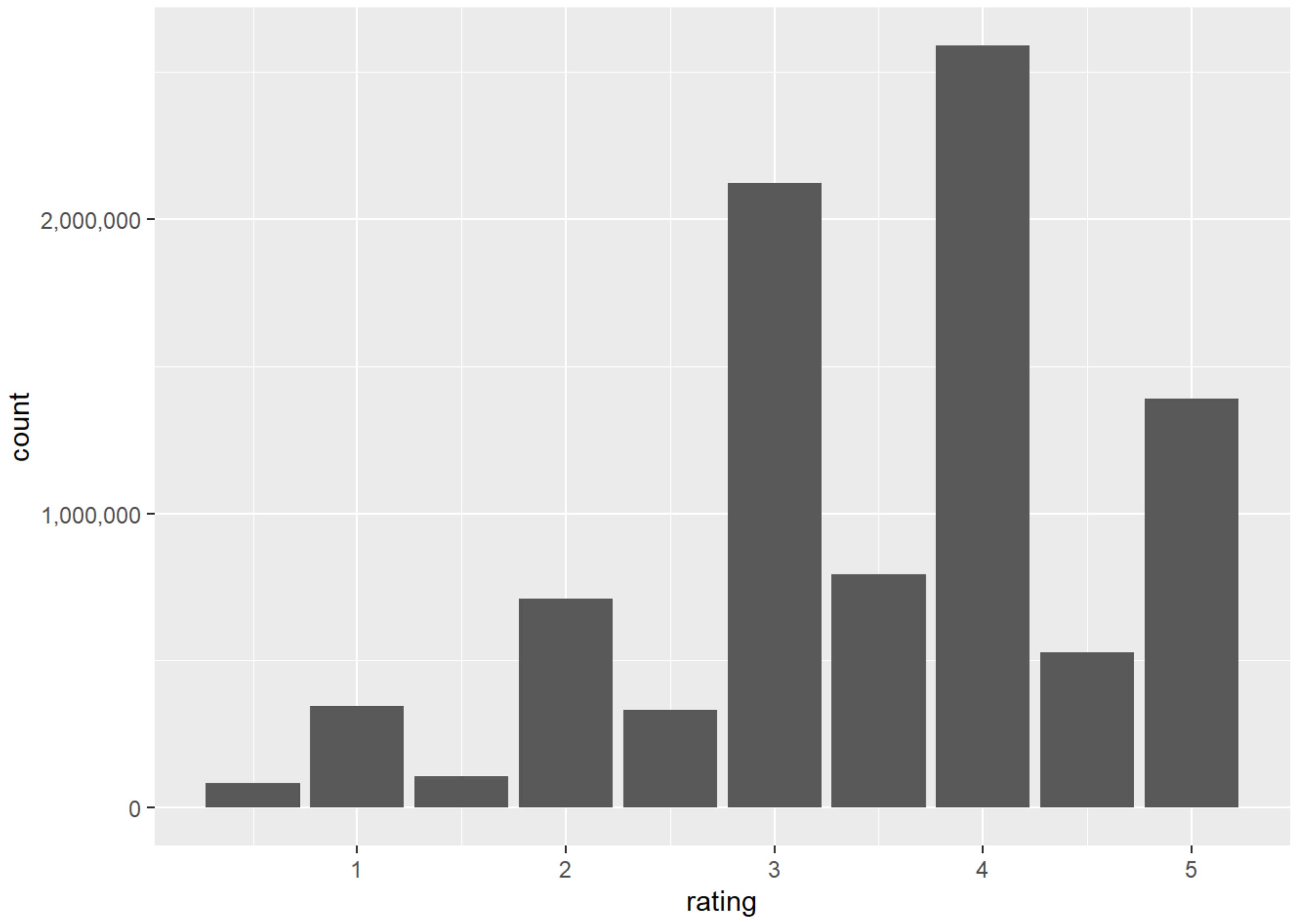
Integer values are most likely and most ratings seem to be 3 or above (3 and 4 are the most common ratings).

It appears that most users seem to give a favorable rating to most of the movies they watch.

```r
# ratings table count
edx %>%
  group_by(rating) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588399
##  2    3   2121185
##  3    5   1390077
##  4    3.5  791624
##  5    2    711402
##  6    4.5  526736
##  7    1    345649
##  8    2.5  333008
##  9    1.5  106426
## 10    0.5   85374
```

```r
# visualising the ratings table
edx %>%
  group_by(rating) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  scale_y_continuous(labels = comma) +
  geom_col()
```

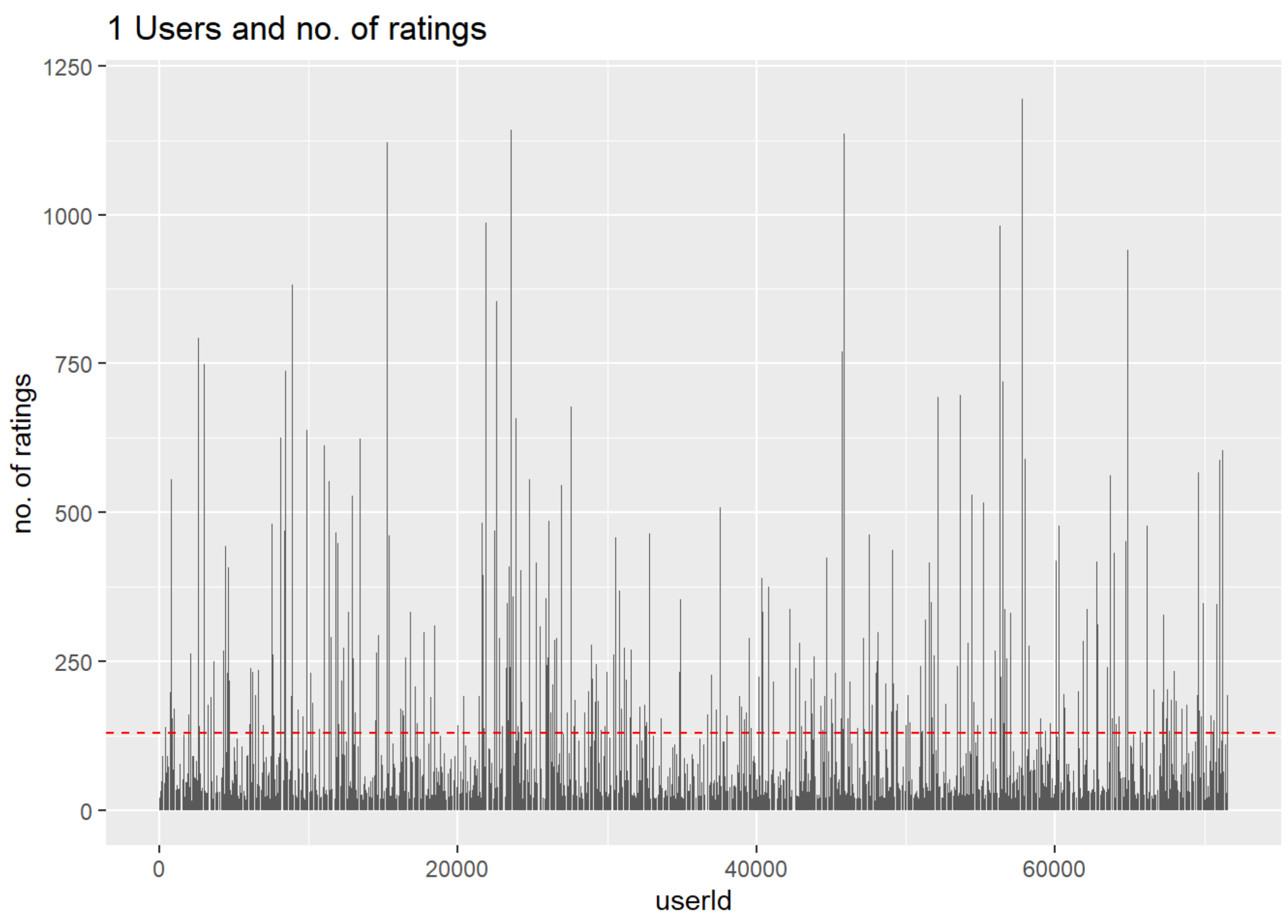


### 1.2.2.2 Ratings & Users

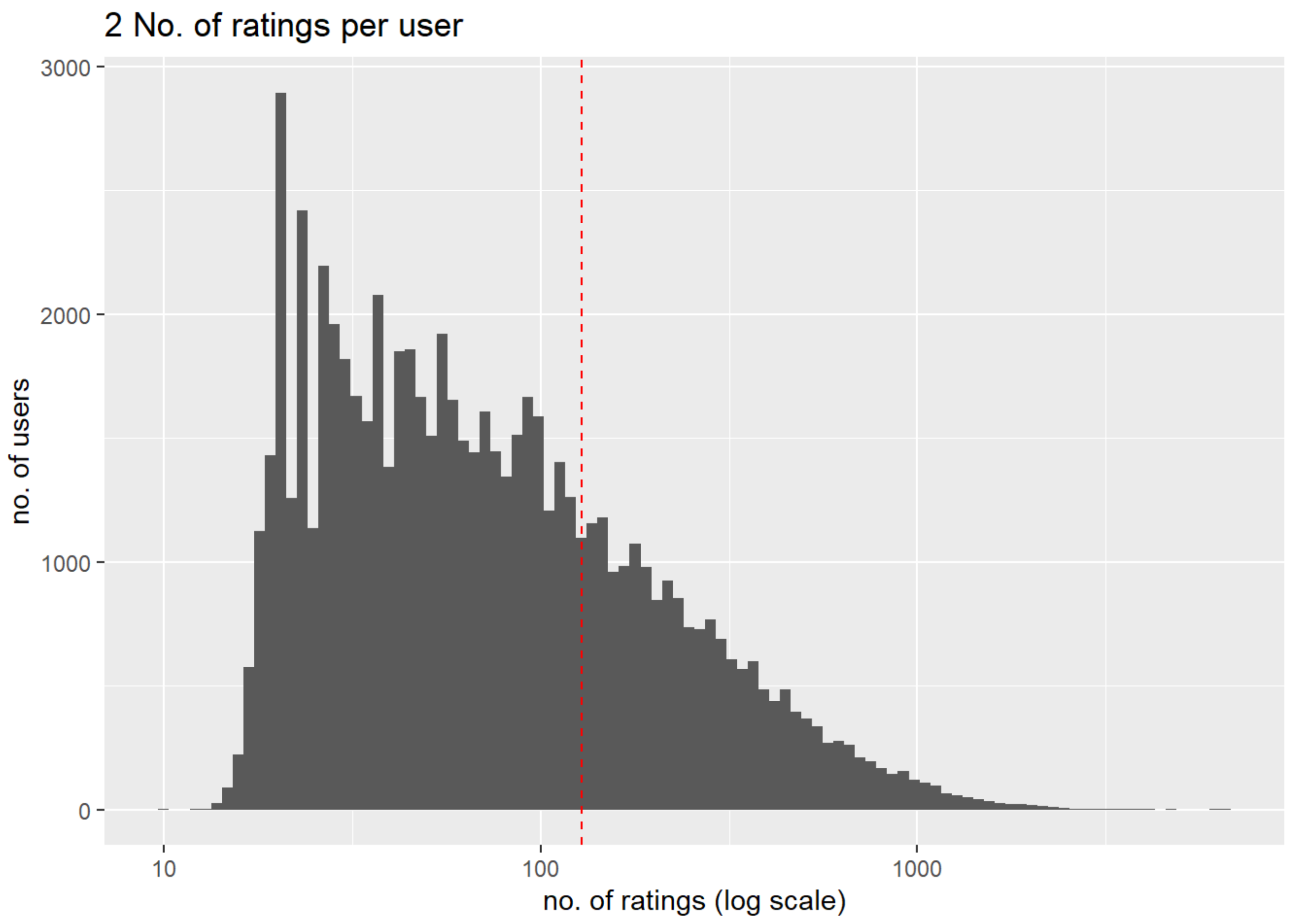
There seems to be a lot of variability in the number of ratings given by a user. For most users the number of ratings is below the overall average of 129 ratings per user (the red dashed line in figures 1 & 2). It seems that a smaller proportion of prolific users or dedicated cinephiles who rate a comparatively large number of movies are pushing the overall average of number of ratings up.

The fact that there doesn't seem to be a strong link between number of ratings by a user and their average rating suggests user effect (figure 3; overall mean rating represented by the blue dashed line).

```
# no. of ratings by user ID
edx %>%
  group_by(userId) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = userId, y = count)) +
  geom_col() +
  ylim(0, 1200) +
  labs(title = "1 Users and no. of ratings",
       y = "no. of ratings") +
  geom_hline(yintercept = avg_ratings_per_user, color="red", linetype="dashed") # avg no. rat
ings/user line
```



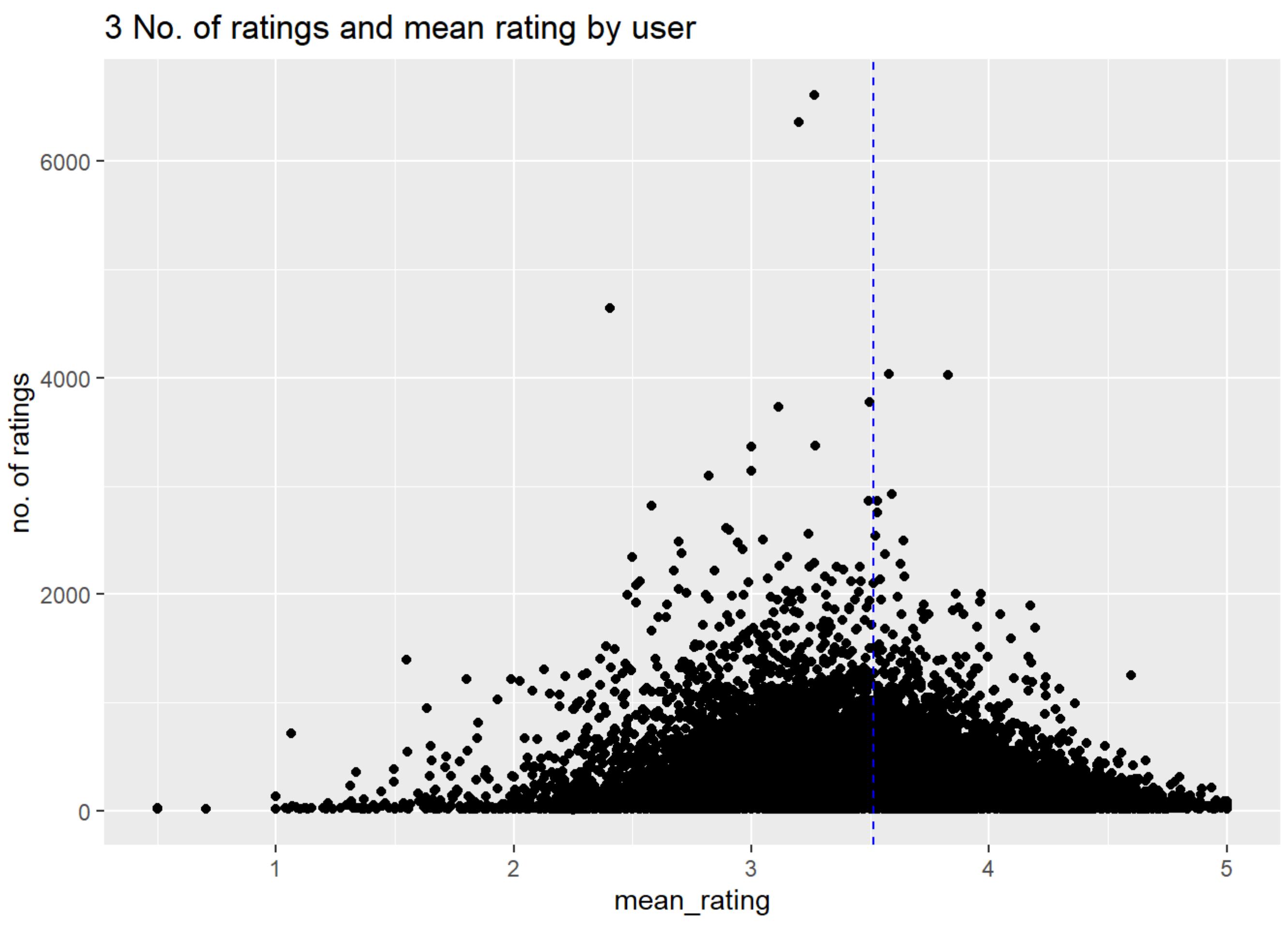
```
# histogram of no. of ratings per user
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 100) +
  scale_x_log10() +
  labs(title = "2 No. of ratings per user",
       x = "no. of ratings (log scale)",
       y = "no. of users")+
  geom_vline(xintercept = avg_ratings_per_user, color="red", linetype="dashed") # avg no. rat
ings/user line
```



```
# top 10 users by number of ratings
edx %>%
  group_by(userId) %>%
  summarise(count = n()) %>%
  arrange(desc(count)) %>%
  top_n(10)
```

```
## # A tibble: 10 x 2
##    userId count
##     <int> <int>
##  1  59269  6616
##  2  67385  6360
##  3  14463  4647
##  4  68259  4036
##  5  27468  4023
##  6  19635  3771
##  7   3817  3733
##  8  63134  3371
##  9  58357  3361
## 10  27584  3142
```

```
# mean rating
avg_rating <- mean(edx$rating)

# no. of ratings and mean rating by user
edx %>%
  group_by(userId) %>%
  summarise(count = n(),
            mean_rating  = mean(rating)) %>%
  ggplot(aes(x = mean_rating, y = count)) +
  geom_point() +
  labs(title = "3 No. of ratings and mean rating by user",
       y = "no. of ratings") +
  geom_vline(xintercept = avg_rating, color="blue", linetype="dashed")
```



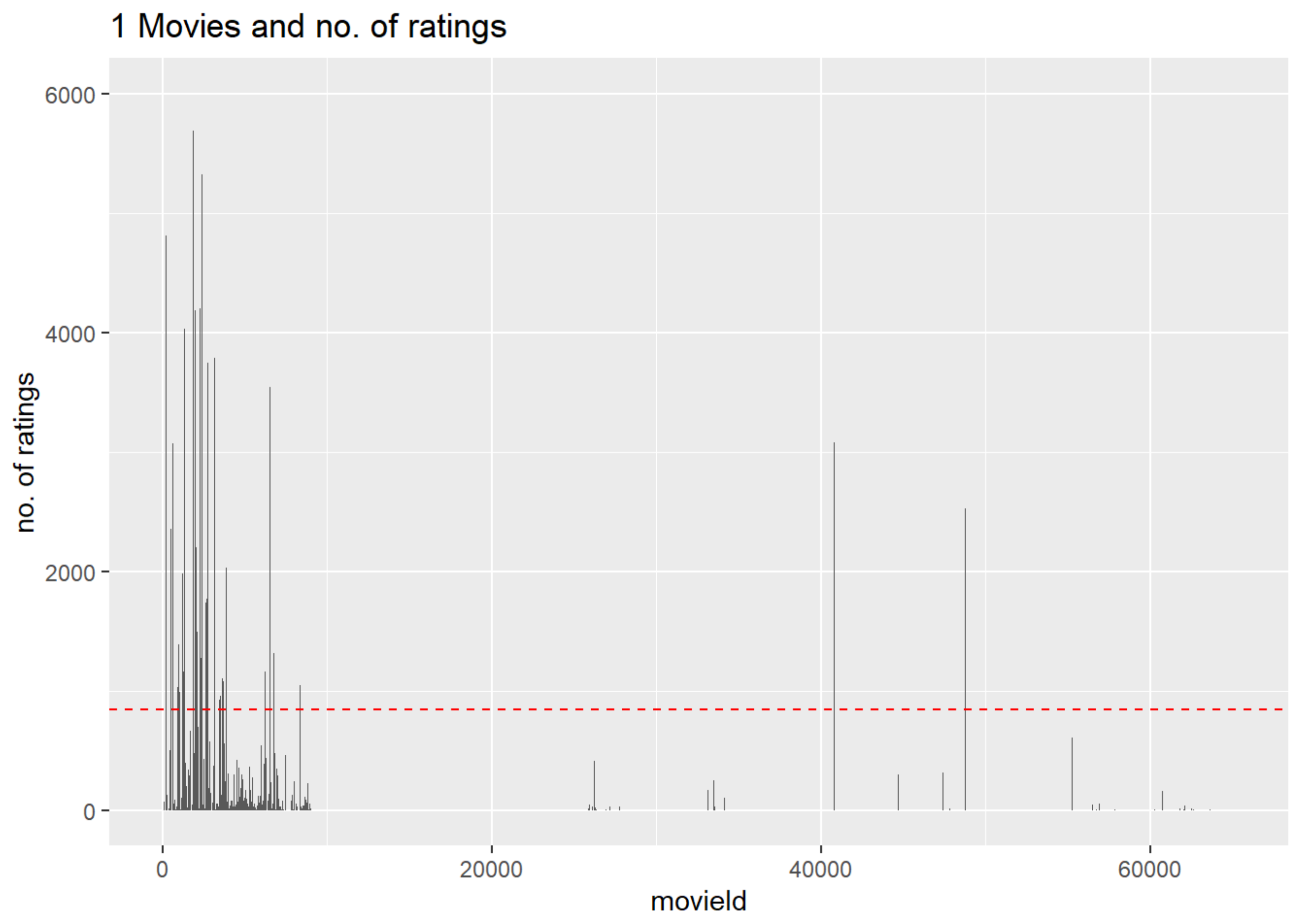
### 1.2.2.3 Ratings & Movies

There seems to be a lot of variability in the number of ratings a movie has. For most movies the number of ratings they have received is below the average of 843 ratings per movie (the red dashed line in figures 1 & 2).

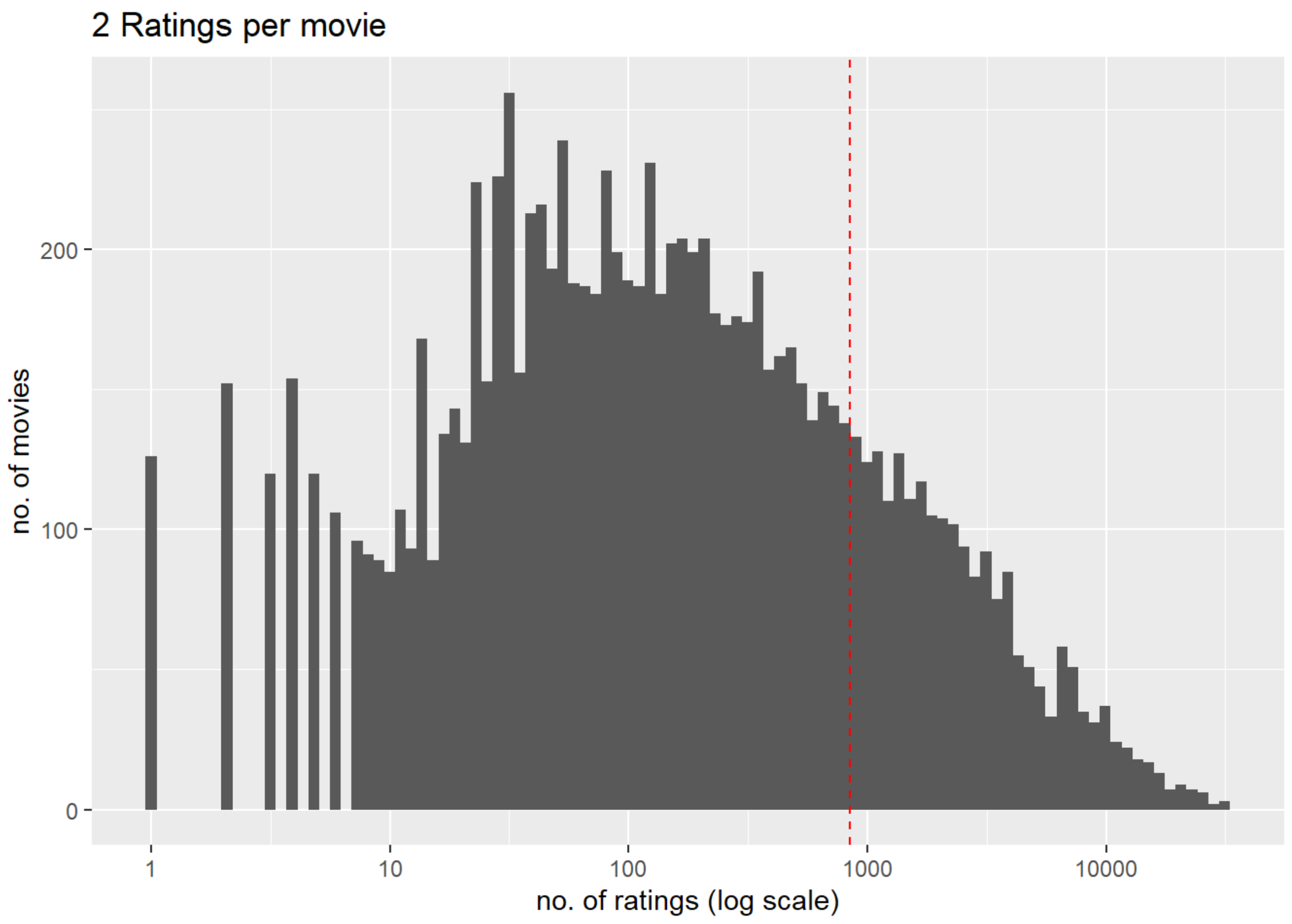
It seems that a small proportion of movies have comparatively received a substantially larger number of ratings which is pushing the average up. All of the top 10 movies by number of ratings are blockbusters.

There doesn't seem to be any strong link between the number of ratings a movie has received and their average rating which suggests movie effect (figure 3; the blue dashed line represents the overall mean rating).

```r
# no. of ratings by movie ID
edx %>%
  group_by(movieId) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = movieId, y = count)) +
  geom_col() +
  ylim(0, 6000) +
  labs(title = "1 Movies and no. of ratings",
       y = "no. of ratings") +
  geom_hline(yintercept = avg_ratings_per_movie, color="red", linetype="dashed") # avg no. ra
tings/movie line
```



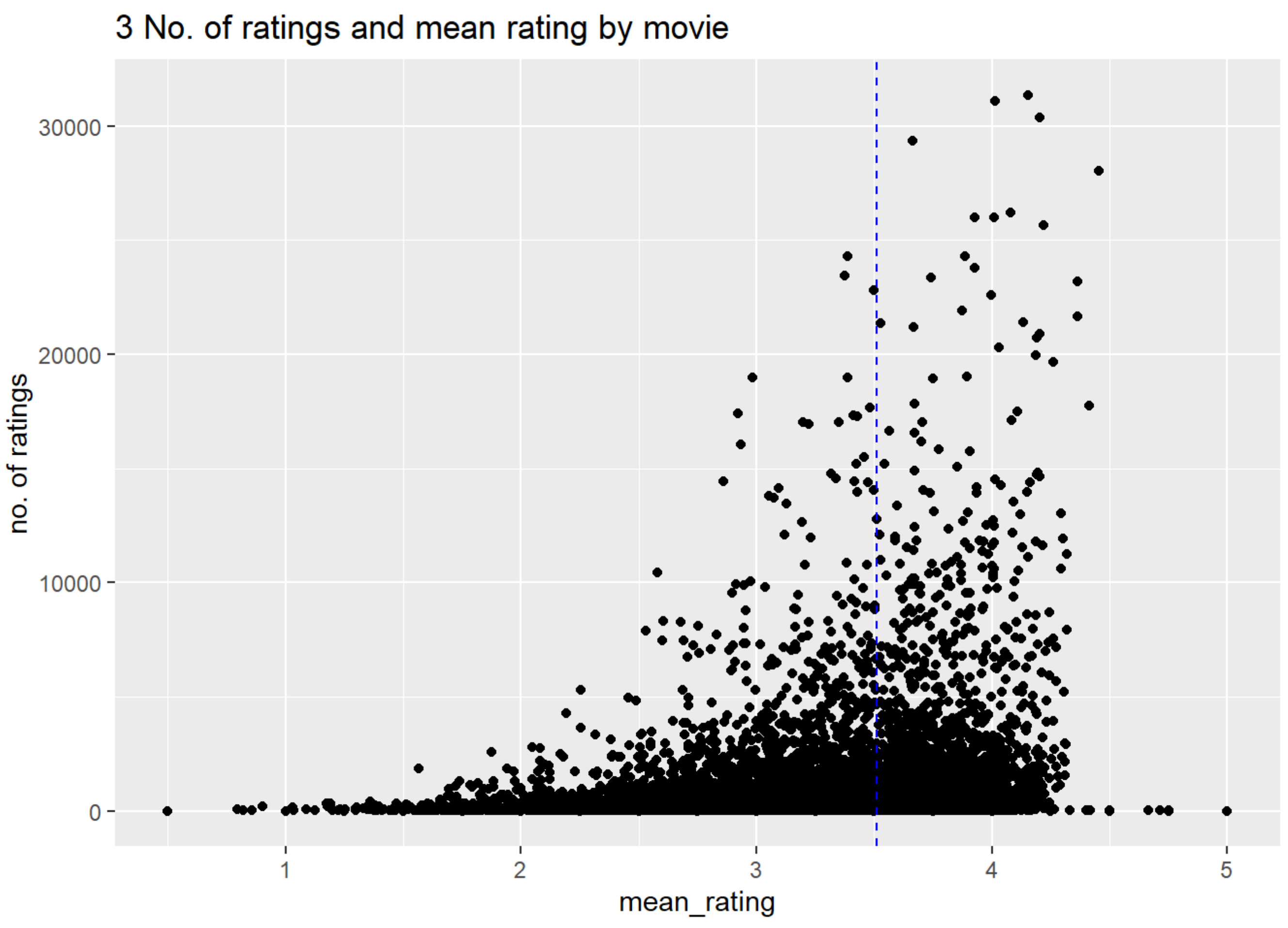
```
# histogram of no. of ratings per movie
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 100) +
  scale_x_log10() +
  labs(title = "2 Ratings per movie",
       x = "no. of ratings (log scale)",
       y = "no. of movies") +
  geom_vline(xintercept = avg_ratings_per_movie, color="red", linetype="dashed") # avg no. ra
tings/movie line
```



```
# top 10 movies by number of ratings
edx %>%
  group_by(movieId) %>%
  summarise(n = n(),
            title = title[1]) %>%
  top_n(10, n) %>%
  arrange(desc(n))
```

```
## # A tibble: 10 x 3
##    movieId     n title
##      <dbl> <int> <chr>
##  1     296 31362 Pulp Fiction (1994)
##  2     356 31079 Forrest Gump (1994)
##  3     593 30382 Silence of the Lambs, The (1991)
##  4     480 29360 Jurassic Park (1993)
##  5     318 28015 Shawshank Redemption, The (1994)
##  6     110 26212 Braveheart (1995)
##  7     457 25998 Fugitive, The (1993)
##  8     589 25984 Terminator 2: Judgment Day (1991)
##  9     260 25672 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
## 10     150 24284 Apollo 13 (1995)
```

```
# no. of ratings and mean rating by movie
edx %>%
  group_by(movieId) %>%
  summarise(count = n(),
            mean_rating  = mean(rating)) %>%
  ggplot(aes(x = mean_rating, y = count)) +
  geom_point() +
  labs(title = "3 No. of ratings and mean rating by movie",
       y = "no. of ratings") +
  geom_vline(xintercept = avg_rating, color="blue", linetype="dashed")
```



### 1.2.2.4 Ratings & Weekdays

There doesn’t seem to be much variability with regards to the number of ratings given on a particular weekday (figure 1).

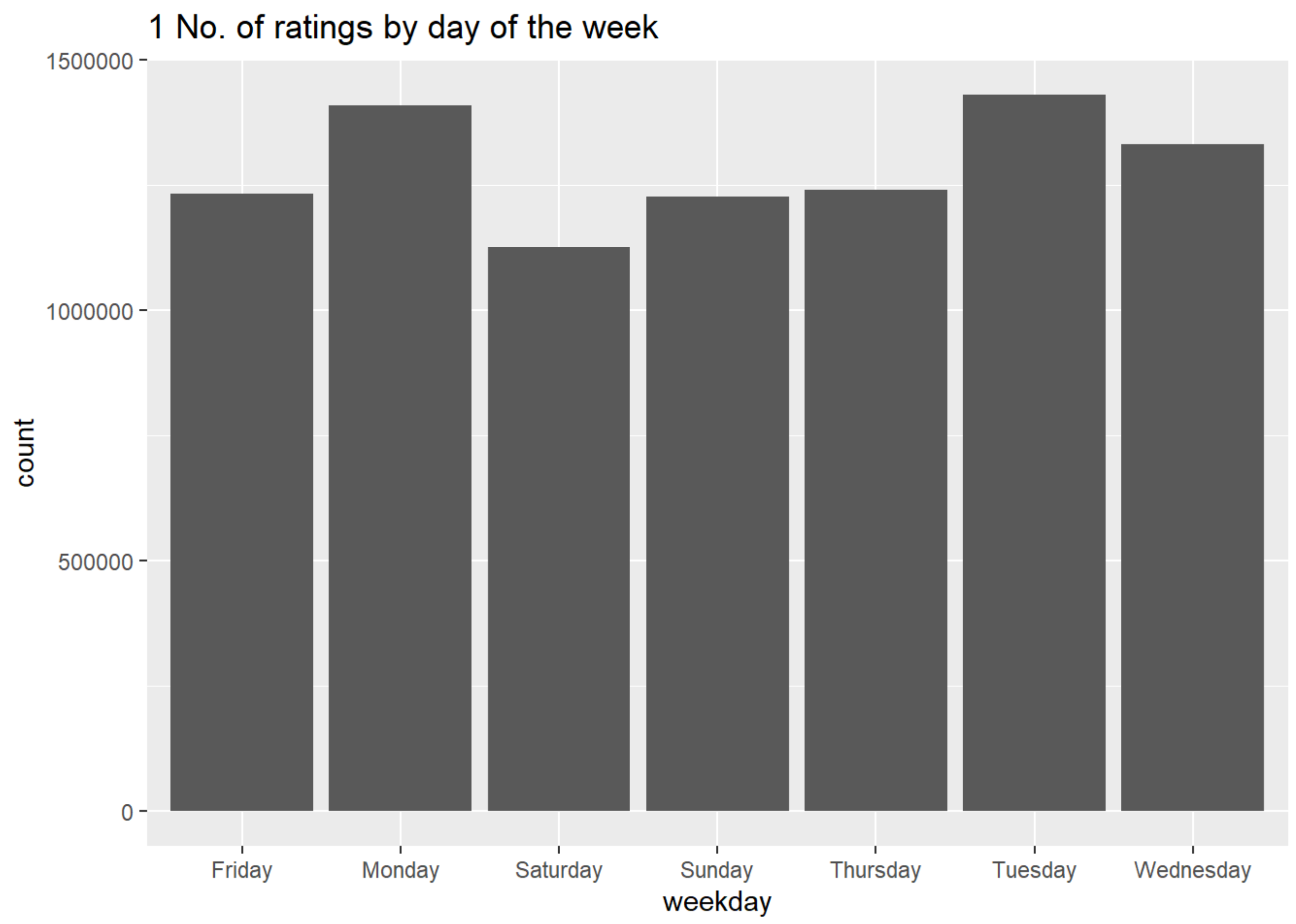
Having said that, there are slightly more ratings given on Mondays or Tuesdays than other weekdays.

There is no real variability in the mean rating across the weekdays (figure 2; the blue dashed line represents the overall mean rating).

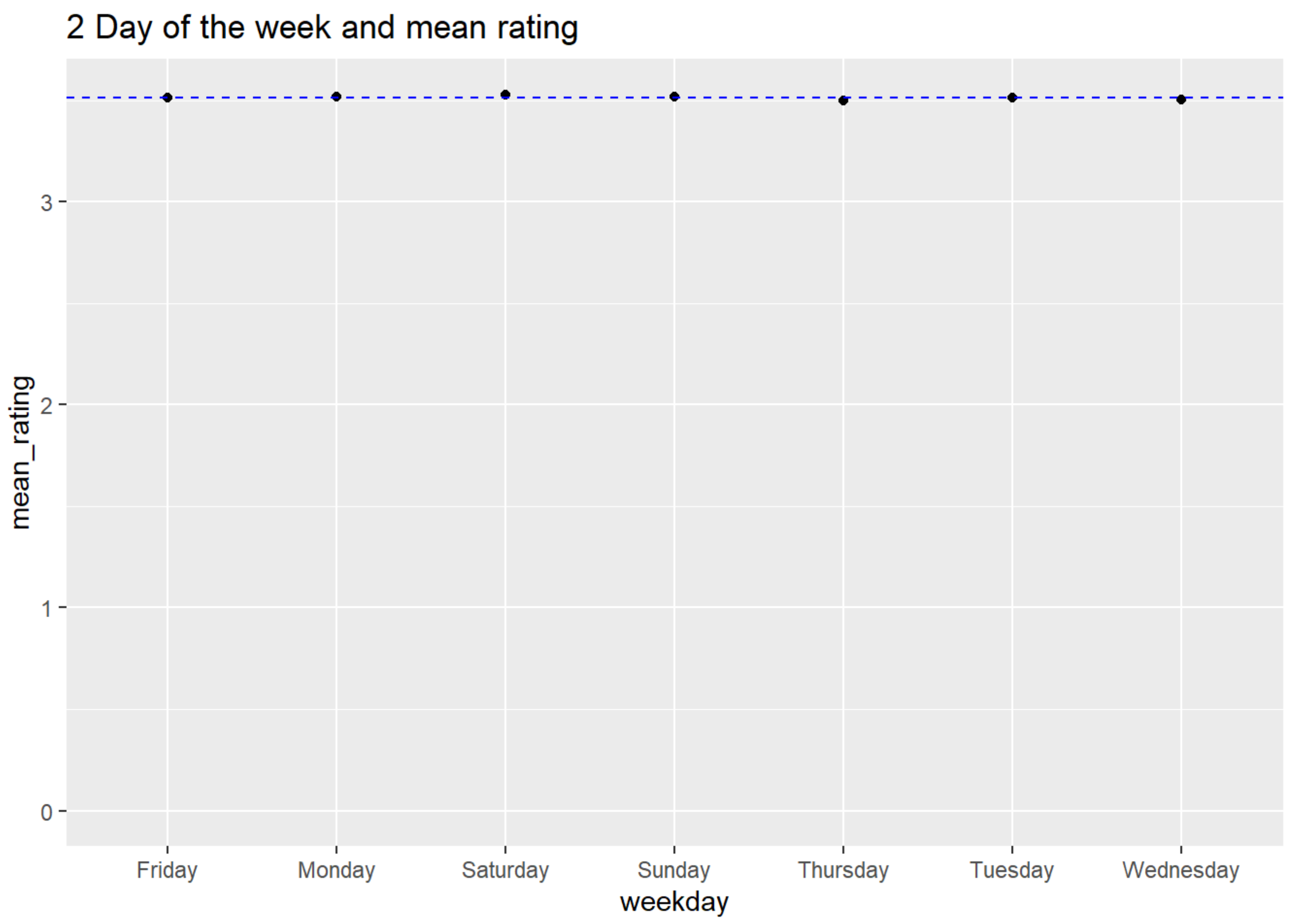
```
# table of no. of ratings by weekday
edx %>%
  group_by(weekday) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 7 x 2
##   weekday     count
##   <chr>       <int>
## 1 Tuesday   1431710
## 2 Monday    1410131
## 3 Wednesday 1332481
## 4 Thursday  1241454
## 5 Friday    1232225
## 6 Sunday    1226457
## 7 Saturday  1125422
```

```
# visualising no. of ratings by weekday
edx %>%
  group_by(weekday) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = weekday, y = count)) +
  geom_col() +
  labs(title = "1 No. of ratings by day of the week")
```



```
# weekday and mean rating
edx %>%
  group_by(weekday) %>%
  summarise(mean_rating = mean(rating)) %>%
  ggplot(aes(x = weekday, y = mean_rating)) +
  geom_point() +
  expand_limits(y = 0) +
  labs(title = "2 Day of the week and mean rating") +
  geom_hline(yintercept = avg_rating, color="blue", linetype="dashed")
```



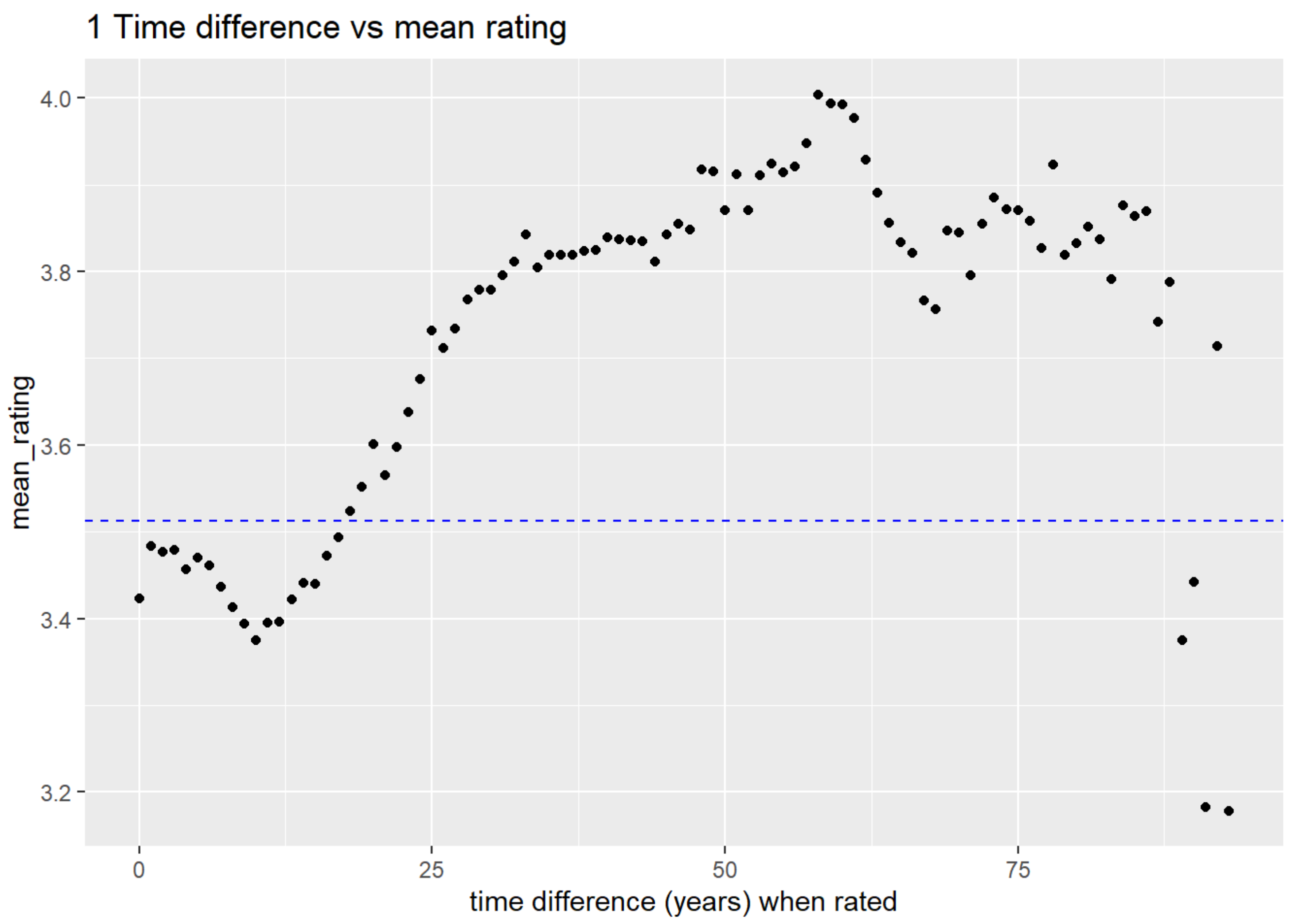
### 1.2.2.5 Ratings & Time difference when rated

Movies rated soon after release seem to get a much higher number of ratings and this seems to decrease quickly as the time between release and rating increases (figure 2).

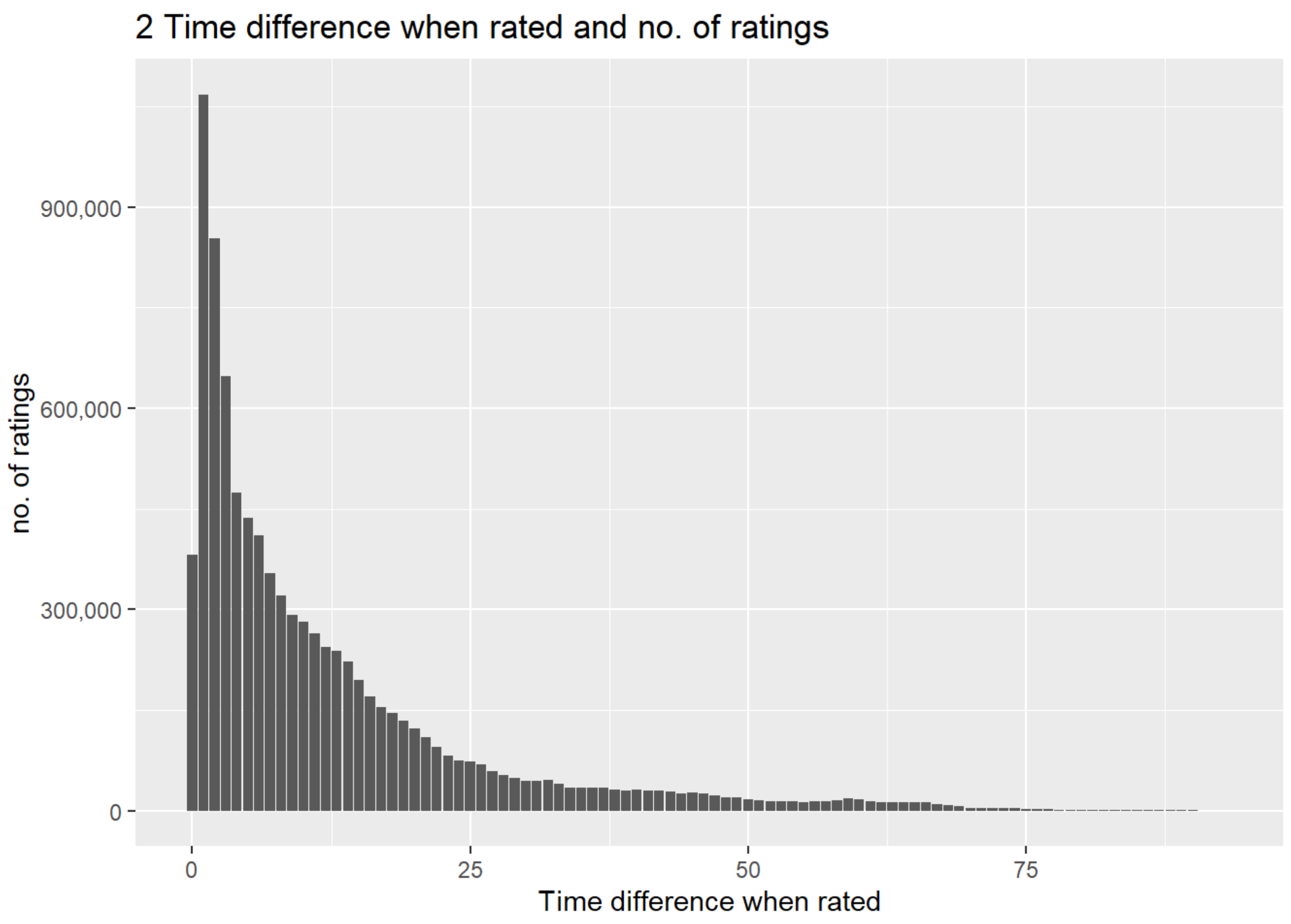
Despite a higher number of ratings, the average rating for ratings soon after a movie is released is lower than the overall average rating (blue dashed line in figure 1) and seems to improve as the time between release and rating increases.

It is possible that when movies are newer there is a lot of promotion and buzz so people watch it for fear of missing out but don't always enjoy it or doesn't live up to the hype whereas for older movies people make an informed choice based on existing reviews and seem to enjoy more what they choose to watch.

```
# time difference when rated and mean rating
edx %>%
  group_by(diff_year) %>%
  summarise(mean_rating = mean(rating)) %>%
  ggplot(aes(x = diff_year, y = mean_rating)) +
  geom_point() +
  labs(title = "1 Time difference vs mean rating",
       x = "time difference (years) when rated") +
  geom_hline(yintercept = avg_rating, color="blue", linetype="dashed")
```



```
# time difference and no. of ratings
edx %>%
  group_by(diff_year) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = diff_year, y = count)) +
  scale_y_continuous(labels = comma) +
  geom_col() +
  labs(title = "2 Time difference when rated and no. of ratings",
       x = "Time difference when rated",
       y = "no. of ratings")
```



```
edx %>%
  group_by(diff_year) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 94 x 2
##    diff_year   count
##        <dbl>   <int>
##  1         1 1068070
##  2         2  853680
##  3         3  647650
##  4         4  473660
##  5         5  436378
##  6         6  410159
##  7         0  380915
##  8         7  354368
##  9         8  320713
## 10         9  292203
## # ... with 84 more rows
```

### 1.2.2.6 Ratings & Genres (combinations)

There seems to be a lot of variability among the average ratings of the genre combinations which range between 1.45 to 4.71; most genres seem to have an average rating around 3-4 (figure 1; the overall mean rating is represented by the blue dashed line).

The number of ratings received by a genre range between 2 to 733246. There is a lot of variability in the average ratings of genres even when looking at genres with 2000 or more ratings (figure 2; the overall mean rating is represented by the blue dashed line). This suggests a strong genre effect.

Although the number of ratings and average rating for a genre doesn’t seem to go hand in hand, genres with a high number of ratings seem to have an average rating that is around the overall average rating (blue dashed line in figure 3).

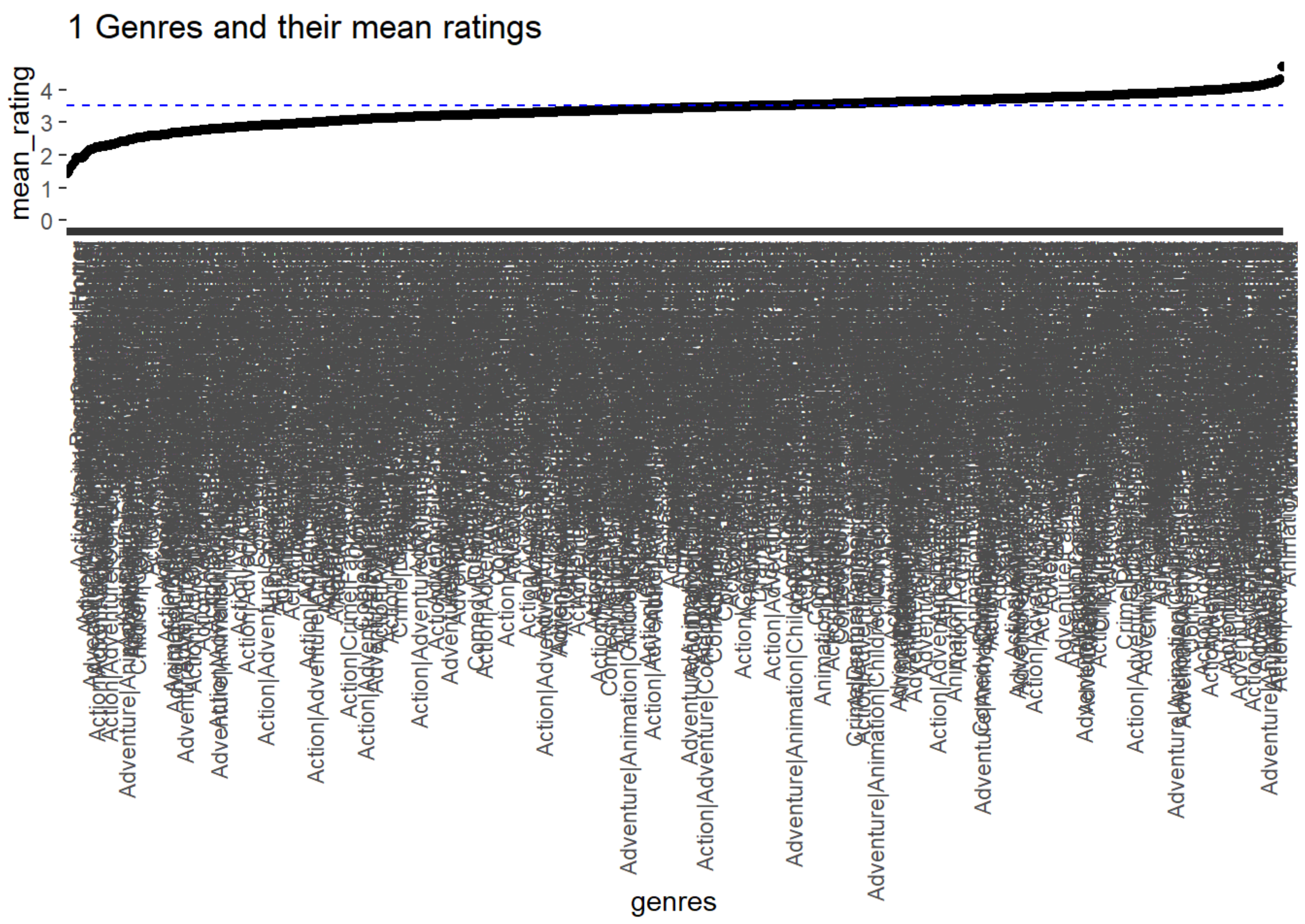
```
# no. of ratings and mean rating by genre
genre_rating <- edx %>%
  group_by(genres) %>%
  summarise(count = n(),
            mean_rating  = mean(rating))

range(genre_rating$count)
```

```
## [1]      2 733246
```

```
range(genre_rating$mean_rating)
```

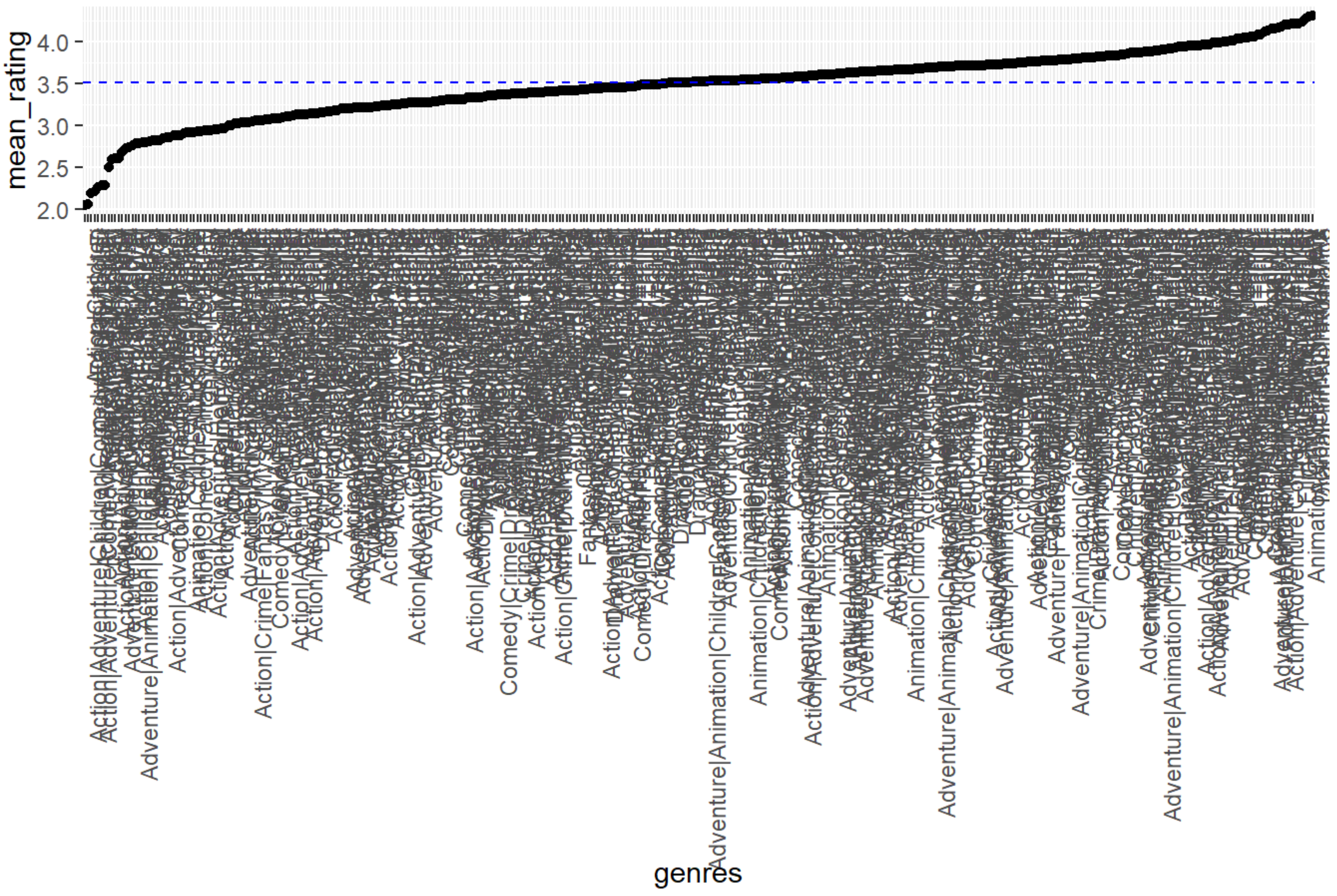
```
## [1] 1.449111 4.714286
```

```
# mean rating by genre
edx %>%
  group_by(genres) %>%
  summarise(mean_rating  = mean(rating)) %>%
  mutate(genres = reorder(genres ,mean_rating)) %>%
  ggplot(aes(x = genres, y = mean_rating)) +
  geom_point() +
  expand_limits(y = 0) +
  geom_hline(yintercept = avg_rating, color="blue", linetype="dashed") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = "1 Genres and their mean ratings")
```

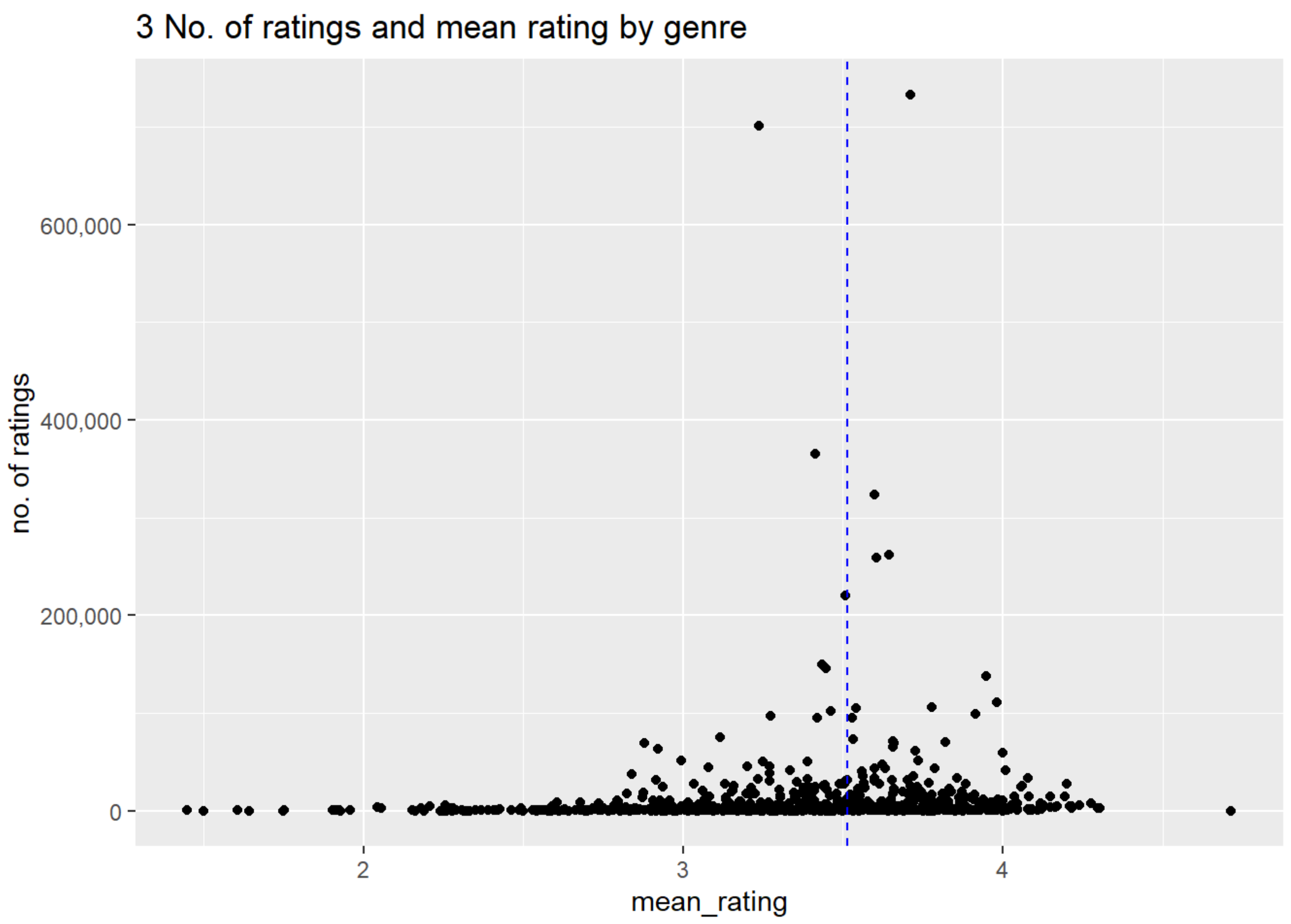


```
# only genres with 2000 or more ratings
edx %>%
  group_by(genres) %>%
  summarise(mean_rating = mean(rating), n = n()) %>%
  filter(n >= 2000) %>%
  mutate(genres = reorder(genres, mean_rating)) %>%
  ggplot(aes(x = genres, y = mean_rating)) +
  geom_point() +
  geom_hline(yintercept = avg_rating, color="blue", linetype="dashed") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = "2 Genres (with 2000 or more ratings) and their mean ratings")
```

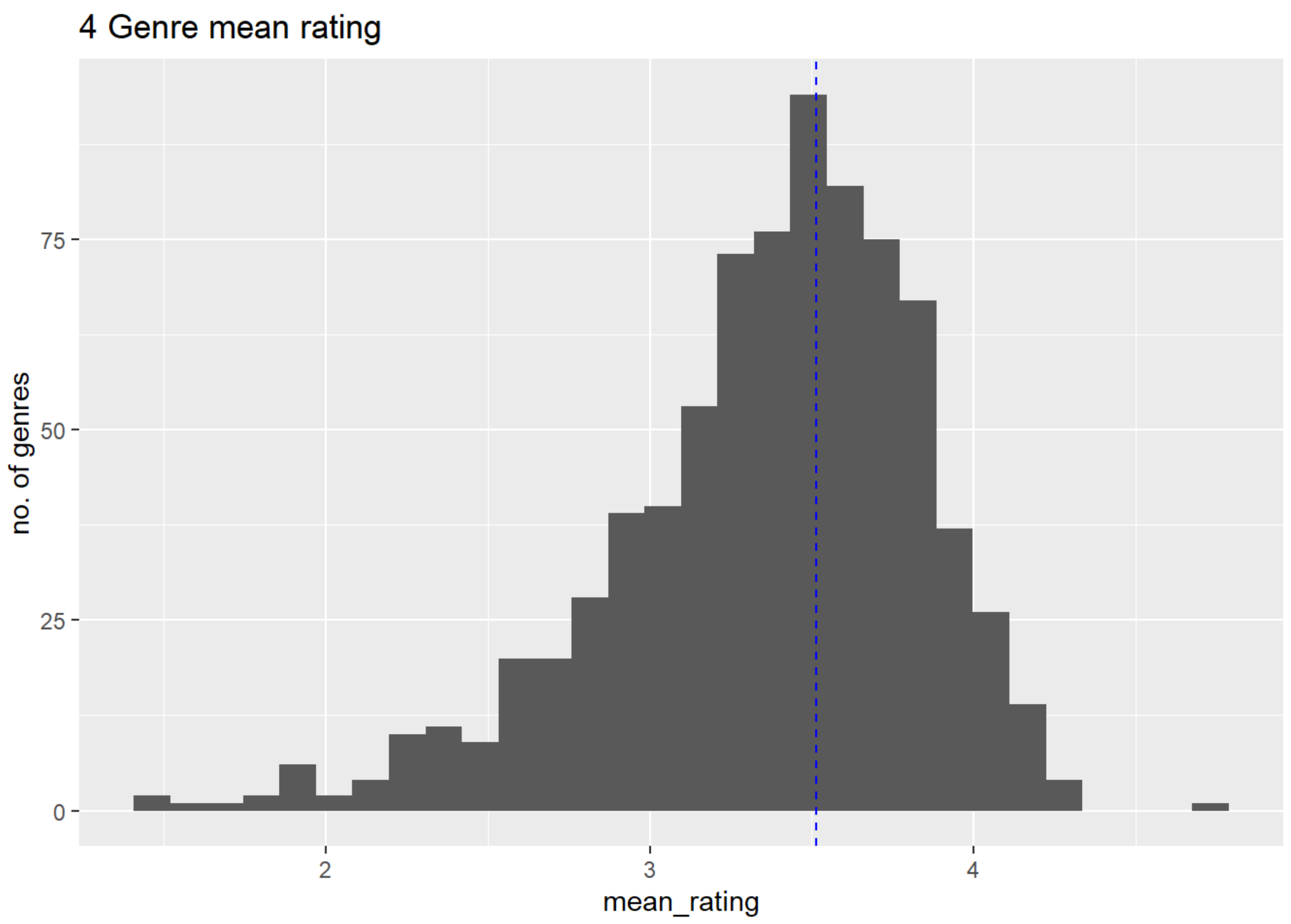
2 Genres (with 2000 or more ratings) and their mean ratings



```
# number of ratings and mean rating by genre
genre_rating %>%
  ggplot(aes(x = mean_rating, y = count)) +
  geom_point() +
  scale_y_continuous(labels = comma) +
  labs(title = "3 No. of ratings and mean rating by genre",
       y = "no. of ratings") +
  geom_vline(xintercept = avg_rating, color="blue", linetype="dashed")
```



```
genre_rating %>%
  ggplot(aes(x = mean_rating)) +
  geom_histogram() +
  labs(title = "4 Genre mean rating",
       y = "no. of genres") +
  geom_vline(xintercept = avg_rating, color="blue", linetype="dashed")
```



# 2. Modeling/Training Algorithms

### Introduction

The edx dataset will be split into training and test sets using a 90:10 ratio.

The models will be trained as per the plan below and their RMSE values will be noted in a table.

The best performing model (based on the RMSE) will be tested on the validation set.

### Modeling/Training plan

A basic model that predicts ratings based on the overall mean rating will first be trained.

Improvements to this model will be attempted by adding the effects of the movie, user, weekday, time difference (between movie release and rating) and genre on the ratings.

## 2.1 Split into training and test sets

The dataset is split into two sets - the training and the test sets. The training set will be used to train the model and the test set to test the accuracy of the model.

There is no “optimum” ratio for splitting a dataset, however, commonly used ratios for splitting a dataset into training and testing sets include 50:50, 60:40, 70:30, 80:20 and 90:10. If the training set is not sufficiently large enough, the model may as a result show a high variance during training and likewise if the test set is not sufficiently large enough the performance value may show a high variance.

As the dataset is quite large (nearly 9 million ratings), it will be split using a 90:10 ratio (90% training and 10% test) which will result in datasets with sufficient data for their respective uses.

```
# test set will be 10% of edx
set.seed(1, sample.kind="Rounding")
index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train <- edx[-index,]
temp <- edx[index,]

# Make sure userId and movieId in test set are also in train set
test <- temp %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, test)
train <- rbind(train, removed)

rm(temp, removed)
```

## 2.2 Loss function - defining the RMSE function

The accuracy of the models will be judged based on the residual mean squared error (RMSE) on the test set.

The rating for a movie $i$ by user $u$ is defined as $y_{u,i}$ and the predicted rating is denoted as $\hat{y}_{u,i}$.

RMSE is defined as:

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

The RMSE function will calculate the RMSE for the ratings in the test set and the predicted ratings by the models.

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

*The code for this function is based on the code in the course material.*

## 2.3 Models

### 2.3.1 Model 1: Basic (mean rating)

This model predicts the same rating for all movies based on the mean rating, regardless of the user, and any differences are explained by random variation:

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

Here, $\mu$ is the true rating and $\varepsilon_{i,u}$ is the random variation that explains the difference between the true rating and the predicted rating ($\hat{\mu}$).

*This modeling approach is based on the approach in the course material.*

```
# overall mean rating
mu_hat <- mean(train$rating)
mu_hat
```

```
## [1] 3.512495
```

```
# rmse
model_1_rmse <- RMSE(test$rating, mu_hat)
model_1_rmse
```

```
## [1] 1.060289
```

The RMSE result for this model is 1.060289. The RMSE is just over 1 which means the typical error in the rating for this model is larger than 1 star. Clearly there is room for improvement and this RMSE result will serve as a baseline and benchmark for the subsequent models.

A table containing the RMSE values for all the models is created to help keep track of the accuracy:

```
# create result table
rmse_results <- tibble(method = "1 Basic (mean rating)", RMSE = model_1_rmse)
```

### 2.3.2 Model 2: Mean + Movie

It was noted in 1.2.2.3 that there is a movie effect in the ratings whereby some movies are in general rated higher than others.

This model adds the movie effect $b_i$ to model 1:

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

It allows us to see if the average rating for a movie can help predict what rating a user might give it. To do this, the least squares estimate $\hat{b}_i$ is used which is calculated using the average of $Y_{u,i} - \hat{\mu}$.

*This modeling approach is based on the approach in the course material.*

```
# movie effect
movie_avg <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))

# model
predict_model_2 <- mu_hat + test %>%
  left_join(movie_avg, by='movieId') %>%
  pull(b_i)

# rmse
model_2_rmse <- RMSE(test$rating, predict_model_2)
model_2_rmse
```

```
## [1] 0.9438267
```

The RMSE result for this model is 0.9438267, which is an improvement on Model 1.

The RMSE results table is updated:

```
# update rmse result table
rmse_results <- rmse_results %>% add_row(method = "2 Mean + Movie", RMSE = model_2_rmse)
```

## 2.3.3 Model 3: Mean + User

It was noted in 1.2.2.2 that there is a user effect in the ratings whereby some users seem hard to please so provide a lower rating while others enjoy most movies that they watch thereby providing a higher rating.

This model adds the user effect $b_u$ to model 1:

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

It allows us to see if the average rating that a user generally provides can help predict what rating they might provide for a particular movie. To do this, the least squares estimate $\hat{b}_u$ is used which is calculated using the average of $Y_{u,i} - \hat{\mu}$.

```
# user effect
user_avg <- train %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat))

# model
predict_model_3 <- mu_hat + test %>%
  left_join(user_avg, by='userId') %>%
  pull(b_u)

# rmse
model_3_rmse <- RMSE(test$rating, predict_model_3)
model_3_rmse
```

```
## [1] 0.9772024
```

The RMSE result for this model is 0.9772024, which is an improvement on Model 1, but not by the same margin as the improvement seen for model 2 above.

The RMSE results table is updated:

```
# update rmse result table
rmse_results <- rmse_results %>% add_row(method = "3 Mean + User", RMSE = model_3_rmse)
```

## 2.3.4 Model 4: Mean + Movie + User

Models 2 and 3 show that adding the movie $b_i$ and user $b_u$ effects individually to model 1 improve the RMSE.

This model adds the user effect to model 2 which results in a model that takes both the user and movie effect into account:

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

*This modeling approach is based on the approach in the course material.*

```
# adding user effect to mean + movie
movie_user_avg <- train %>%
  left_join(movie_avg, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

# model
predict_model_4 <- test %>%
  left_join(movie_avg, by='movieId') %>%
  left_join(movie_user_avg, by='userId') %>%
  mutate(predict = mu_hat + b_i + b_u) %>%
  pull(predict)

# rmse
model_4_rmse <- RMSE(test$rating, predict_model_4)
model_4_rmse
```

```
## [1] 0.8652566
```

The RMSE result for this model is 0.8652566, which is an improvement on model 2 and closer to the target RMSE for this project.

The RMSE results table is updated:

```
# update rmse result table
rmse_results <- rmse_results %>% add_row(method = "4 Mean + Movie + User", RMSE = model_4_rms
e)
```

### 2.3.5 Model 5: Mean + Movie (regularized)

Some movies have comparatively very few ratings. When predicted ratings for movies are based on very few ratings, it can cause uncertainty and lead to larger estimates of $b_i$ which can increase the RMSE.

Regularization allows the penalization of larger estimates derived from smaller sample sizes using the penalty $\lambda$.

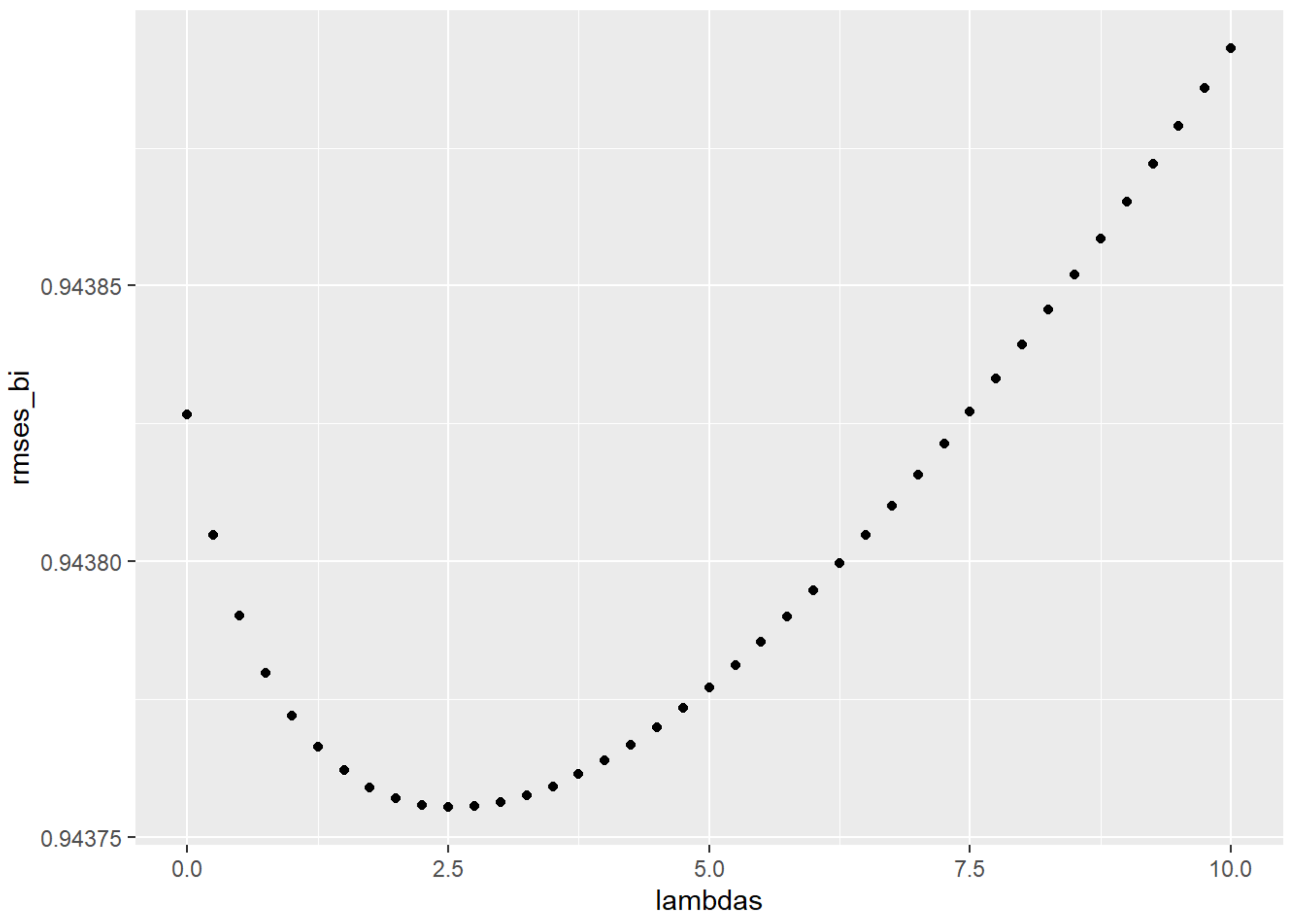
This model adds the regularized movie effect $b_i(\lambda)$ to model 1:

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

*This modeling approach is based on the approach in the course material.*

First, the optimum tuning parameter $\lambda$ is determined:

```r
# find optimum lambda

lambdas <- seq(0, 10, 0.25)

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

  mu <- mean(train$rating)

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

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

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

qplot(lambdas, rmses_bi)
```



```r
lambda_bi <- lambdas[which.min(rmses_bi)]
```

The optimum $\lambda$ is 2.5.

The regularized movie effect is then added to model 1 (mean rating):

```
# adding regularized movie effect
movie_avg_r <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda_bi), n_i = n())

# model
predict_model_5 <- test %>%
  left_join(movie_avg_r, by = "movieId") %>%
  mutate(pred = mu_hat + b_i) %>%
  pull(pred)

# rmse
model_5_rmse <- RMSE(test$rating, predict_model_5)
model_5_rmse
```

```
## [1] 0.9437553
```

The RMSE for this model is 0.9437553, which is a slight improvement on model 2 (mean + movie effect).

The RMSE result table is updated:

```
# update rmse result table
rmse_results <- rmse_results %>% add_row(method = "5 Mean + Movie(r)", RMSE = model_5_rmse)
```

### 2.3.6 Model 6: Mean + Movie (reg) + User (reg)

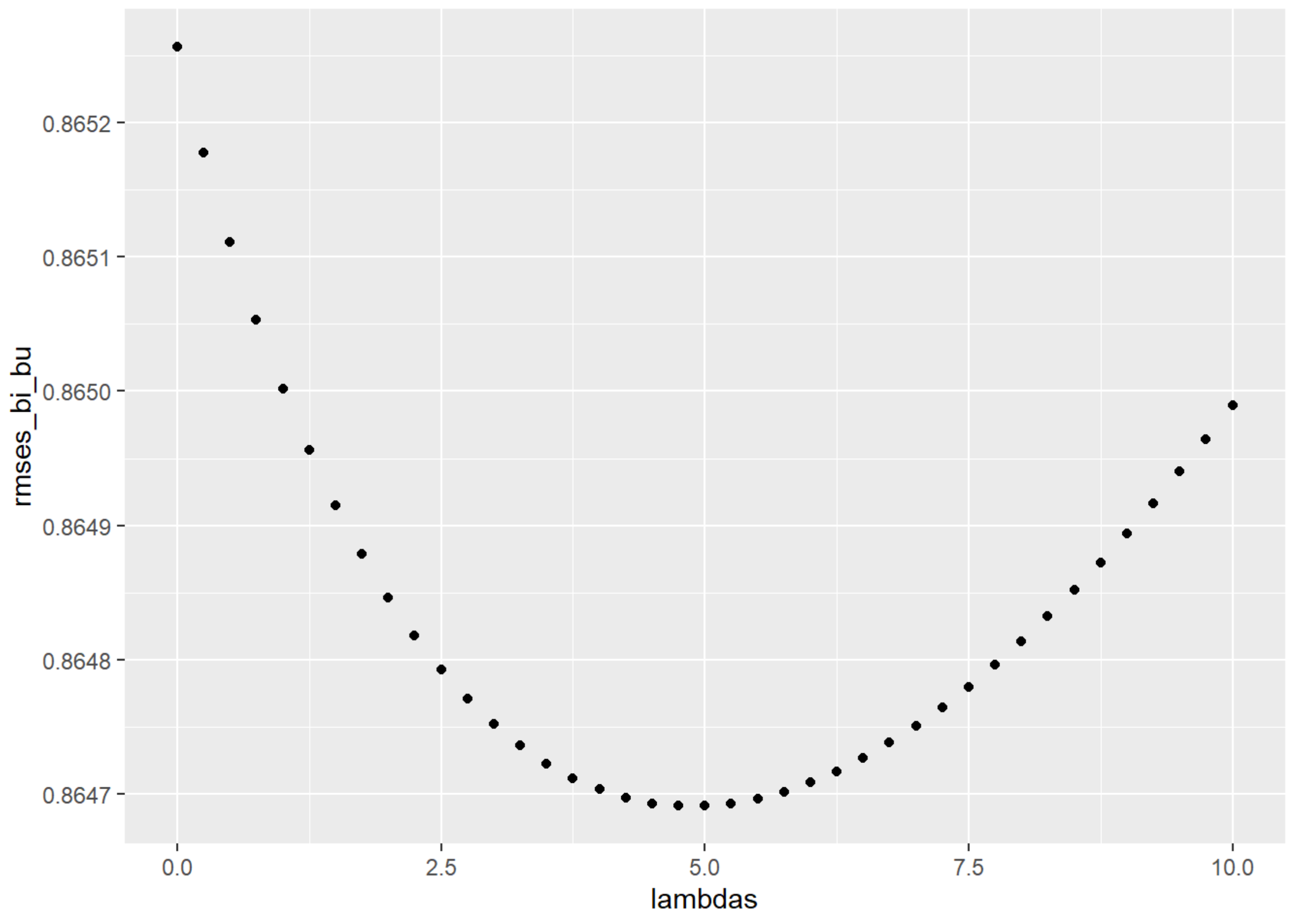
This model adds the regularized user effect $b_u(\lambda)$ to model 5:

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

*This modeling approach is based on the approach in the course material.*

The optimum tuning parameter $\lambda$ is first determined:

```r
# find optimum lambda

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

  mu <- mean(train$rating)

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

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

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

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

qplot(lambdas, rmses_bi_bu)
```



```r
lambda_bi_bu <- lambdas[which.min(rmses_bi_bu)]
```

The optimum $\lambda$ is 5.

The regularized user effect is added to model 5:

```
# adding regularized user effect to mean + movie (r)
movie_r_user_r_avg <- train %>%
  left_join(movie_avg_r, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu_hat - b_i)/(n()+lambda_bi_bu), n_i = n())

# model
predict_model_6 <- test %>%
  left_join(movie_avg_r, by='movieId') %>%
  left_join(movie_r_user_r_avg, by='userId') %>%
  mutate(predict = mu_hat + b_i + b_u) %>%
  pull(predict)

# rmse
model_6_rmse <- RMSE(test$rating, predict_model_6)
model_6_rmse
```

```
## [1] 0.8647192
```

The RMSE result is 0.8647192, which is a slight improvement on model 4.

The RMSE result table is updated:

```
# update rmse result table
rmse_results <- rmse_results %>% add_row(method = "6 Mean + Movie(r) + User(r)", RMSE = model
_6_rmse)
```

### 2.3.7 Model 7: Mean + Movie (reg) + User (reg) + Time difference when rated

It was noted in 1.2.2.5 above that when movies are rated soon after release the average rating is lower than the overall average rating and this improves as the time difference between movie release and rating increases.

This model adds the time difference effect $b_y$ to model 6:

$$Y_{u,i} = \mu + b_i(\lambda) + b_u(\lambda) + b_y + \varepsilon_{u,i}$$

```
# adding time difference effect (difference between rated year and release year) to Mean + Mo
vie (r) + User (r)
movie_age_avg <- train %>%
  left_join(movie_avg_r, by='movieId') %>%
  left_join(movie_r_user_r_avg, by='userId') %>%
  group_by(diff_year) %>%
  summarize(b_y = mean(rating - mu_hat - b_i - b_u))

# model
predict_model_7 <- test %>%
  left_join(movie_avg_r, by='movieId') %>%
  left_join(movie_r_user_r_avg, by='userId') %>%
  left_join(movie_age_avg, by='diff_year') %>%
  mutate(predict = mu_hat + b_i + b_u + b_y) %>%
  pull(predict)

# rmse
model_7_rmse <- RMSE(test$rating, predict_model_7)
model_7_rmse
```

```
## [1] 0.8641939
```

The RMSE is 0.8641939, which is a slight improvement on model 6.

The RMSE table is updated:

```
# update rmse result table
rmse_results <- rmse_results %>% add_row(method = "7 Mean + Movie(r) + User(r) + Time differe
nce", RMSE = model_7_rmse)
```

### 2.3.8 Model 8: Mean + Movie (reg) + User (reg) + Time difference + Weekday

This model adds the weekday effect $b_w$ to model 7 to see if the day of the week for the rating can help improve the RMSE:

$$Y_{u,i} = \mu + b_i(\lambda) + b_u(\lambda) + b_y + b_w + \varepsilon_{u,i}$$

```
# adding weekday effect to Mean + Movie(r) + User(r) + Time difference
weekday_avg <- train %>%
  left_join(movie_avg_r, by='movieId') %>%
  left_join(movie_r_user_r_avg, by='userId') %>%
  left_join(movie_age_avg, by='diff_year') %>%
  group_by(weekday) %>%
  summarize(b_w = mean(rating - mu_hat - b_i - b_u - b_y))

# model
predict_model_8 <- test %>%
  left_join(movie_avg_r, by='movieId') %>%
  left_join(movie_r_user_r_avg, by='userId') %>%
  left_join(movie_age_avg, by='diff_year') %>%
  left_join(weekday_avg, by="weekday") %>%
  mutate(predict = mu_hat + b_i + b_u + b_y + b_w) %>%
  pull(predict)

# rmse
model_8_rmse <- RMSE(test$rating, predict_model_8)
model_8_rmse
```

```
## [1] 0.8641942
```

The RMSE is 0.8641942, which is not an improvement. This is not surprising as a strong weekday effect was not noted in 1.2.2.4 earlier.

The RMSE table is updated:

```
# update rmse result table
rmse_results <- rmse_results %>% add_row(method = "8 Mean + Movie(r) + User(r) + Time diff +
 Weekday", RMSE = model_8_rmse)
```

### 2.3.9 Model 9: Mean + Movie (reg) + User (reg) + Time difference + Genres

It was noted in 1.2.2.6 earlier that there seems to be a strong genre (combinations) effect.

This model adds the genre effect $b_g$ to model 7:

$$Y_{u,i} = \mu + b_i(\lambda) + b_u(\lambda) + b_y + b_g + \varepsilon_{u,i}$$

```
# adding genre effect to Mean + Movie(r) + User(r) + Time diff
genre_avg <- train %>%
  left_join(movie_avg_r, by='movieId') %>%
  left_join(movie_r_user_r_avg, by='userId') %>%
  left_join(movie_age_avg, by='diff_year') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u - b_y))

# model
predict_model_9 <- test %>%
  left_join(movie_avg_r, by='movieId') %>%
  left_join(movie_r_user_r_avg, by='userId') %>%
  left_join(movie_age_avg, by='diff_year') %>%
  left_join(genre_avg, by="genres") %>%
  mutate(predict = mu_hat + b_i + b_u + b_y + b_g) %>%
  pull(predict)

# rmse
model_9_rmse <- RMSE(test$rating, predict_model_9)
model_9_rmse
```

```
## [1] 0.8638751
```

The RMSE is 0.8638751, which is an improvement on model 7.

The RMSE table is updated:

```
# update rmse result table
rmse_results <- rmse_results %>% add_row(method = "9 Mean + Movie(r) + User(r) + Time diff +
 Genre", RMSE = model_9_rmse)
```

## 2.4 Validation

The best performing model is model 9 with RMSE 0.8638751:

$$Y_{u,i} = \mu + b_i(\lambda) + b_u(\lambda) + b_y + b_g + \varepsilon_{u,i}$$

The performance of this model will be tested on the validation dataset created at the start of this project.

### 2.4.1 Prepare validation dataset

Before assessing the performance of the model it is necessary to prepare the validation dataset by following the same steps as followed in 1.1 to prepare the ‘edx’ dataset:

```
# convert timestamp to date and extract rating year
validation_prep <- validation %>%
  mutate(date = as_datetime(timestamp),
         rated_year = year(as_datetime(timestamp)))

# extract movie release year
validation_prep <- validation_prep %>%
  mutate(release_year = str_extract(title, pattern = "(\\(\\d{4}\\))"), # includes brackets
         release_year = str_extract(release_year, pattern = "(\\d{4})"), # remove brackets
         release_year = as.numeric(release_year))

# measure difference between year released and year rated
validation_prep <- validation_prep %>%
  mutate(diff_year = rated_year - release_year)

# remove rows with negative differences
validation_prep <- validation_prep %>%
  filter(diff_year >= 0)

# drop raw timestamp column
validation_prep <- validation_prep %>%
  select(-timestamp)
```

### 2.4.2 Predicted ratings for validation dataset and performance

The predicted ratings for the validation dataset are calculated using the same approach as that for model 9:

```
# predicted ratings for validation (same approach as model 9)

validate_pred_ratings <- validation_prep %>%
  left_join(movie_avg_r, by='movieId') %>%
  left_join(movie_r_user_r_avg, by='userId') %>%
  left_join(movie_age_avg, by='diff_year') %>%
  left_join(genre_avg, by="genres") %>%
  mutate(predict = mu_hat + b_i + b_u + b_y + b_g) %>%
  pull(predict)
```

The RMSE is then calculated:

```
# test
validate_pred_ratings_rmse <- RMSE(validation_prep$rating, validate_pred_ratings)
validate_pred_ratings_rmse
```

```
## [1] 0.8644241
```

The RMSE is **0.8644241**, which satisfies the ideal target RMSE (less than 0.86490) for this project.

# 3. Results

A quick summary of all the models and their RMSE values:

```
rmse_results %>%
  mutate(across(-method, num, digits = 8))
```

```
## # A tibble: 9 x 2
##   method                                                 RMSE
##   <chr>                                             <num:.8!>
## 1 1 Basic (mean rating)                            1.06028900
## 2 2 Mean + Movie                                   0.94382669
## 3 3 Mean + User                                    0.97720238
## 4 4 Mean + Movie + User                            0.86525658
## 5 5 Mean + Movie(r)                                0.94375534
## 6 6 Mean + Movie(r) + User(r)                      0.86471921
## 7 7 Mean + Movie(r) + User(r) + Time difference    0.86419393
## 8 8 Mean + Movie(r) + User(r) + Time diff + Weekday 0.86419424
## 9 9 Mean + Movie(r) + User(r) + Time diff + Genre   0.86387512
```

The best performing model is model 9:

$$Y_{u,i} = \mu + b_i(\lambda) + b_u(\lambda) + b_y + b_g + \varepsilon_{u,i}$$

This model was tested on the validation dataset and the RMSE was 0.8644241.

# 4. Conclusion

### Summary

The goal of this project was to train a machine learning algorithm that could help predict the rating a particular viewer might give to a particular movie so as to build a recommendation system that could recommed movies to a viewer that were predicted to be likely to get a favourable rating.

Nine models were trained and the best performing model took into account the overall average rating for the dataset and improved it by adding the effects of the individual viewers, movies, genres, and the time difference between movie release and rating.

### Limitations

The models for this project were trained on a subset of rating data from MovieLens (by the GroupLens lab at the University of Minnesota) that contains just over 8 million individual ratings.

Whilst models trained on this dataset might perform as expected at recommending movies to viewers registered at the MovieLens website, it is not necessarily true that they might perform equally well when implemented by streaming providers such as Netflix, Amazon Prime, Disney+, etc. This is because viewers who register and rate movies at MovieLens might not be representative of the typical Netflix or Disney+ subscriber.

Furthermore, based on the RMSE, the predicted rating of even the best performing model is likely to be only within 0.8 stars of the true rating which is a large margin considering that the ratings can only be between 0-5 stars. This can be the difference between a viewer really liking and not liking a recommended movie.

### Future work

Adding the effect of the day of the week the movie is rated was tested (model 8) but it did not improve the performance of the model.

It is possible that although the day of the week may not influence the ratings, the hour of the day or the week of the year that the rating is made on could have an effect. This information can be extracted from the ‘timestamp’ variable in the dataset and could be considered for future improvements to the model.

The concept of Matrix Factorization could also be useful in investigating and modeling the patterns in rating variations between groups of similar viewers and movies.

## Bibliography

1. Baheti, P (2022) V7 *Train Test Validation Split: How To & Best Practices [2022]* https://www.v7labs.com/blog/train-validation-test-set (https://www.v7labs.com/blog/train-validation-test-set)
2. Irizarry, R. (2022) *Introduction to Data Science* https://rafalab.github.io/dsbook/ (https://rafalab.github.io/dsbook/)