Harvardx Capstone MovieLens

Laura Butgereit

08/05/2021

Contents

List of Figures

List of Tables

1 Introduction

Consumer behavior, especially consumer decision making, has been of interest to researchers for ages. Some of the earliest documented research into consumer behavior was in 1713, when Swiss professor Nicolas Bernoulli investigated how much money a person would be willing to spend to play a simple game of chance involving tossing a coin (Plous, 1993). But even earlier than that, in the 15th century, Europeans began importing spices from the Far East (Solomon et al, 2012). How did those traders decide which spices to import? How did they know that consumers would like one spice and not like another spice?

In more recent times with the advent of large datasets, it has become possible to predict consumer behavior by looking at past behavior which is documented in these datasets. Gartner defines *predictive behavior analysis* as the use of techniques such as data mining, data visualization, algorithm clustering and neural networks in order to find patterns or trends in data. These patterns or trends could be used to identify products which customers are likely to buy next (Gartner, 2021).

The research described in this paper uses a dataset of movie information and viewer ratings of that movie. The ratings range from zero to five and embody the consumers' review of the movie. Was the movie good? Then the consumer may give it a five. Was the movie terrible? Then the consumer may give it a one. The assignment is to predict the rating given various criteria. Success or failure is measured by root mean squared error (and not by accuracy measurements).

This research uses a historical dataset of movie ratings and attempts to predict the way consumers may review movies in future.

This project was part of a Capstone project in Harvardx's *Professional Certificate in Data Science*.

Section 2 describes the computational environment where this research was executed. This section also itemizes any modifications that might have needed to be done in order to process the large data files.

Section 3 describes issues which were encountered in downloading the required data files. This section provides various strategies which were utilized to handle these issues. Section 4 provides information on how the training data and validation data was extracted.

Section 5 of this paper describes the initial exploration into the *edx* dataset. This section includes numerical values and some histograms of distributions. After the initial exploration, the training dataset was reshaped to expose more information. This reshaping is described in Section 6.

The bulk of the investigation into the dataset is described in Section 7. This section includes investigations into the relationships between various pieces of data (such as the age of a movie) and the ratings that they collected.

Section 8 describes the various investigations into possible models to be able to predict the ratings given to a movie. There were a number of different strategies which were investigated. Each model is described along with the RMSE (root mean square error) for that model. A summary of the various RMSEs for each model is provided in Sub-Section 8.10. The results of the model on the validation set is described in Sub-Section 8.12.

Conclusions can be found in Section 9.

Before continuing with this document, the author wishes to share the wise words of Donald Knuth (1984). In his journal paper entitled *Literate Programming* he states:

> Let us change our traditional attitude to the construction of [computer] programs: Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.
>
> The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style. Such an author, with thesaurus in hand, chooses the names of variables carefully and explains what each variable means. He or she strives for a program that is comprehensible because its concepts have been introduced in an order that is best for human understanding...

The author eschews variables such as *mu* and *y_hat* (which may be familiar to data science practitioners and statisticians), in favour of more descriptive variables such as *overall_average* and *predicted_rating*. Although this practice may be in conflict with code examples in (Irizarry, 2021), the author is employed in the IT industry and is attending this course with the view of integrating R (running in an automated fashion started by Java programs) into a Spring Boot environment. As such the practices of good naming conventions in source code are well ingrained.

2 Compute Environment and Memory issues

The analyses presented in the paper were done on a Dell XPS 13 9380 13.3" Core i7-8565U Notebook. The notebook has 8GB memory and a Core i7-8565U processor. The notebook was running Ubuntu 20.04.2 LTS with R 4.0.3 installed.

2.1 Adding more swap space

There were memory issues and an additional 4G swap space was configured on the notebook with the following Linux commands executed as superuser:

```
fallocate -l 4G /harvardx_swapspace
chmod 600 /harvardx_swapspace
mkswap /harvardx_swapspace
swapon /harvardx_swapspace
```

In order to have this swapspace available automatically when the notebook boots up an entry needed to be made in the file

```
/etc/fstab
```

which looks like

```
/harvardx_swapfile none swap sw 0 0
```

2.2 Modifying *swappiness*

The *swappiness* value defines how aggressively the Linux kernel swaps memory pages to swap devices. The value ranges between zero and one hundred (Ljubuncic, 2015). A higher *swappiness* value implies a stronger preference toward swapping more readily. A lower value implies not swapping (Love, 2013). The value can be configured on this file

```
/etc/sysctl.conf
```

The entry looks like

```
vm.swappiness=10
```

This change was made to stop Linux swapping excessively.

2.3 Request extra memory when executing R

In order to successfully build and execute, the scripts were run in R and not Rstudio with the following command

```
R -max-mem-size=7000M --vanilla < Butgereit_script.R  > output.txt
```

3 Downloading, unzipping, and validating data

The author lives in a rural area of Africa where download speeds are slow and connections often break. For that reason, the code supplied by the Edx website for downloading data was modified slightly to include a *timeout* parameter. *The code reviewer may need to increase the parameter depending on his or her local circumstances.* The code then attempted to download the required zip file, store the file locally, and then to check the size of the downloaded file to verify that the entire data file was downloaded. If the download and size check did not succeed, error messages are printed and the script is stopped. If the download and size check did succeed, only then is the data unzipped.

3.1 Set up pseudo-constants

First a handful of pseudo-constants were set up

```
url <- "http://files.grouplens.org/datasets/movielens/ml-10m.zip"
zip_file_name <- "ml-10m.zip"
zip_file_size <- 65566137
download_timeout <- 600 #seconds == 10 minutes
ratings_dat_name <- "ml-10M100K/ratings.dat"
ratings_dat_size <- 265105635
movies_dat_name <- "ml-10M100K/movies.dat"
movies_dat_size <- 522197
```

3.2 Download zip file and check sizes

The script was broken up into sections where the sizes of all the downloaded files were tested before the next step was executed

```
#
# test to see if the file exists, if it does not exist, download it
# OR if the file is not the correct size, download it
#
# NB:  short circuited logical operator
#
if ((!file.exists(zip_file_name) ||
    (file.info(zip_file_name)$size) != zip_file_size)  ) {
    options(timeout = download_timeout)
    download.file(url, zip_file_name)
}

#
# check to see if the entire fire was downloaded.  If not, give
# manual instructions to the operator on what to do
#
if ( !file.exists(zip_file_name) ||
     (file.info(zip_file_name)$size != zip_file_size) ) {
  print("The file was not properly downloaded.  I suggest you")
  print("download it through your browser.  Point your browser to ")
  print(url)
  print("and download it to the same directory where you started rstudio")
  stop()
}

print("zip file download ok")
```

```
## [1] "zip file download ok"
```

3.3 Unzip file and check sizes

```
#
# if the ratings file does not exist OR is not the right size OR
# if the movies file does not exist OR is not the right size
# then unzip the file (it obviously exists at this point or the
# script would have stopped
#
if ( !file.exists(ratings_dat_name) ||
     (file.info(ratings_dat_name)$size != ratings_dat_size) ||
     !file.exists(movies_dat_name) ||
     (file.info(movies_dat_name)$size != movies_dat_size) ) {
  unzip(zip_file_name)
}

#
# check to see if they have unzipped ok and are the correct size
#
if ( !file.exists(ratings_dat_name) ||
     (file.info(ratings_dat_name)$size != ratings_dat_size) ||
     !file.exists(movies_dat_name) ||
     (file.info(movies_dat_name)$size != movies_dat_size) ) {
  print("The files did not unzip properly.")
  print("Do you have space on your workstation?")
  print("Or are there permissions issues?")
  stop()
}

print("data files unzipped ok")
```

```
## [1] "data files unzipped ok"
```

4 Extracting and saving the *edx* and *validation* datasets

A script was provided to generate two datasets from the downloaded files: *edx* and *validation*. In order to keep all R code on one file, additional pseudo-constants were set up to facilitate storing these two datasets on intermediate R objects

4.1 Additional pseudo-constants

```
#
# set up common pseudo-constants
#
edx_rda <- "edx.rda"
validation_rda <- "validation.rda"
```

4.2 Build *edx* and *validation* datasets

The following code rebuilds the objects when necessary

NB: The code does not check the modification dates on the downloaded files. If the *edx* and *validation* intermediary R objects need to be rebuilt, then they must be removed from the file system manually and the script must be re-executed.

```
#
# the true part of the if statement is the original code
# from edx website
#
if ( !file.exists(edx_rda) || !file.exists(validation_rda) ) {
  print("building the edx and validation data frames")
  #
  # read ratings - this is
  ratings <- fread(text = gsub("::", "\t", readLines(ratings_dat_name)),
                   col.names = c("userId", "movieId", "rating", "timestamp"))


  movies <- str_split_fixed(readLines(movies_dat_name), "\\::", 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")

  # 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(ratings, movies, test_index, temp, movielens, removed)

  print(paste("edx nrow ", nrow(edx), sep=" "))
  print(paste("validation nrow ", nrow(validation), sep=" "))
  save(edx, file = edx_rda)
  save(validation,file = validation_rda)
  print("saved edx.rda and validation.rda")
}
```

NB: It is important to note that besides saving the *validation* dataset to the intermediate R data file, no other interaction was done with the *validation* dataset until the time the final model was to be tested as documented in Sub-Section 8.12

5 Initial Analysis of *edx* Dataset

Initial investigation in the *edx* dataset provides the following information.

5.1 Rows and columns in *edx* dataset

Number of rows in *edx* dataset

```
nrow(edx)
```

```
## [1] 9000055
```

Number of columns in *edx* dataset

```
ncol(edx)
```

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

5.2 Structure of *edx* dataset

The structure of the *edx* dataset is

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
##  - attr(*, ".internal.selfref")=<externalptr>
```

And a sample of the data looks like

```
head(edx, 3)
```

```
##    userId movieId rating timestamp           title
## 1:      1     122      5 838985046 Boomerang (1992)
## 2:      1     185      5 838983525  Net, The (1995)
## 3:      1     292      5 838983421  Outbreak (1995)
##                          genres
## 1:               Comedy|Romance
## 2:        Action|Crime|Thriller
## 3: Action|Drama|Sci-Fi|Thriller
```

5.3 Total number of movies

The total number of movies which were reviewed is

```
nrow(edx %>% select(movieId) %>% unique)
```

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

However, as can be seen in Figure 1 the number of reviews which a movie collected from users (also known as reviewers) spanned a wide range of values.

```
reviews_per_movie <- edx %>% group_by(movieId) %>%
  summarize(number_of_reviews = n())
histogram_graph(x = reviews_per_movie$number_of_reviews,
```

```
                data= reviews_per_movie, bins=20) +
  xlab("Reviews per Movie")
```

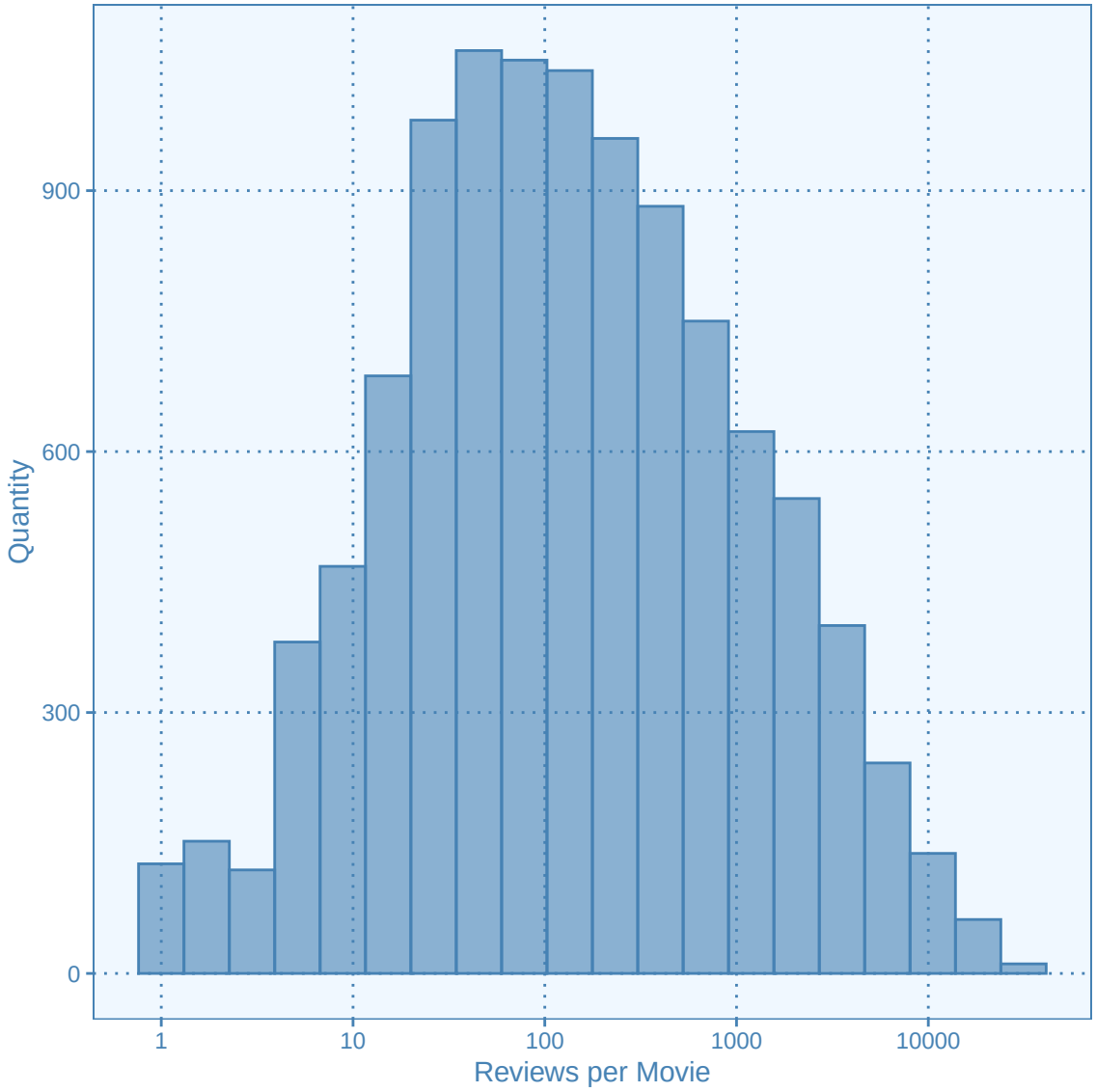

Figure 1: Reviews per movie

5.4 Total number of reviewers or users

The total number of people who have reviewed movies is

```
nrow(edx %>% select(userId) %>% unique)
```

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

The reviewers were not equally busy, however. Figure 2 shows the distribution of the number of reviews submitted per user or per reviewer.

```
reviews_per_user <- edx %>% group_by(userId) %>%
  summarize(number_of_reviews = n())
```

```
histogram_graph(x = reviews_per_user$number_of_reviews,
                data= reviews_per_user, bins=20) +
  xlab("Reviews per Reviewer(User)")
```

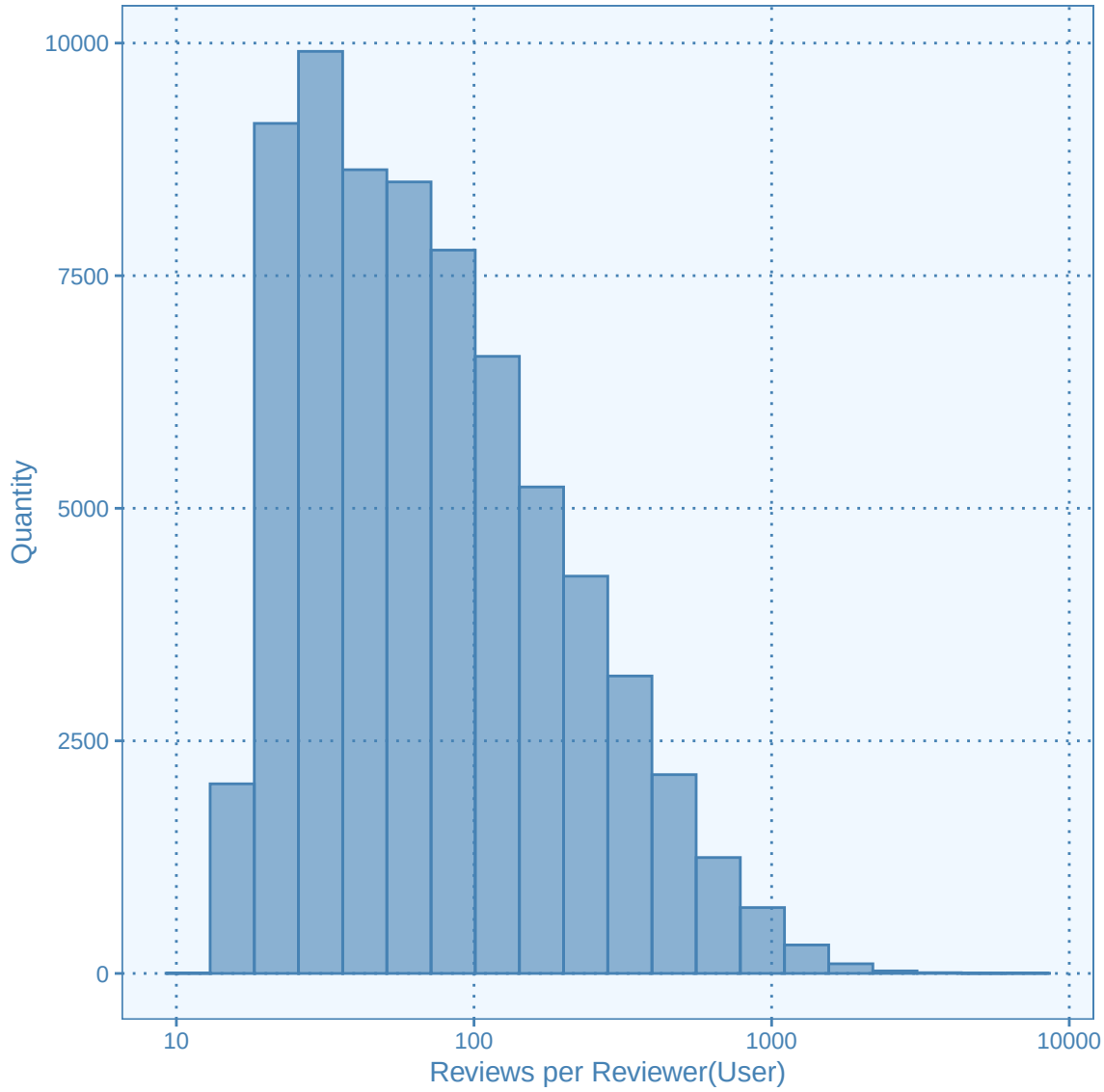

Figure 2: Reviews per user

5.5 Rating distribution

The actual ratings are a number between zero and five. The distribution of the rating values across all entries in the *edx* dataset can be seen in Figure 3.

```
ratings <- edx %>% group_by(rating) %>%
  summarize(number_by_rating = n())
options(scipen=10000000)
bar_graph(x = ratings$rating, y=ratings$number_by_rating,
                data= ratings) +
  xlab("Rating") +
```

```
  ylab("Total Number of Ratings") +
  scale_x_continuous(limits=c(0, 5.5), breaks=seq(0.0, 5.5, 0.5),
                     labels=c("0.0", "0.5", "1.0", "1.5", "2.0", "2.5",
                              "3.0", "3.5", "4.0", "4.5", "5.0", ""))
```

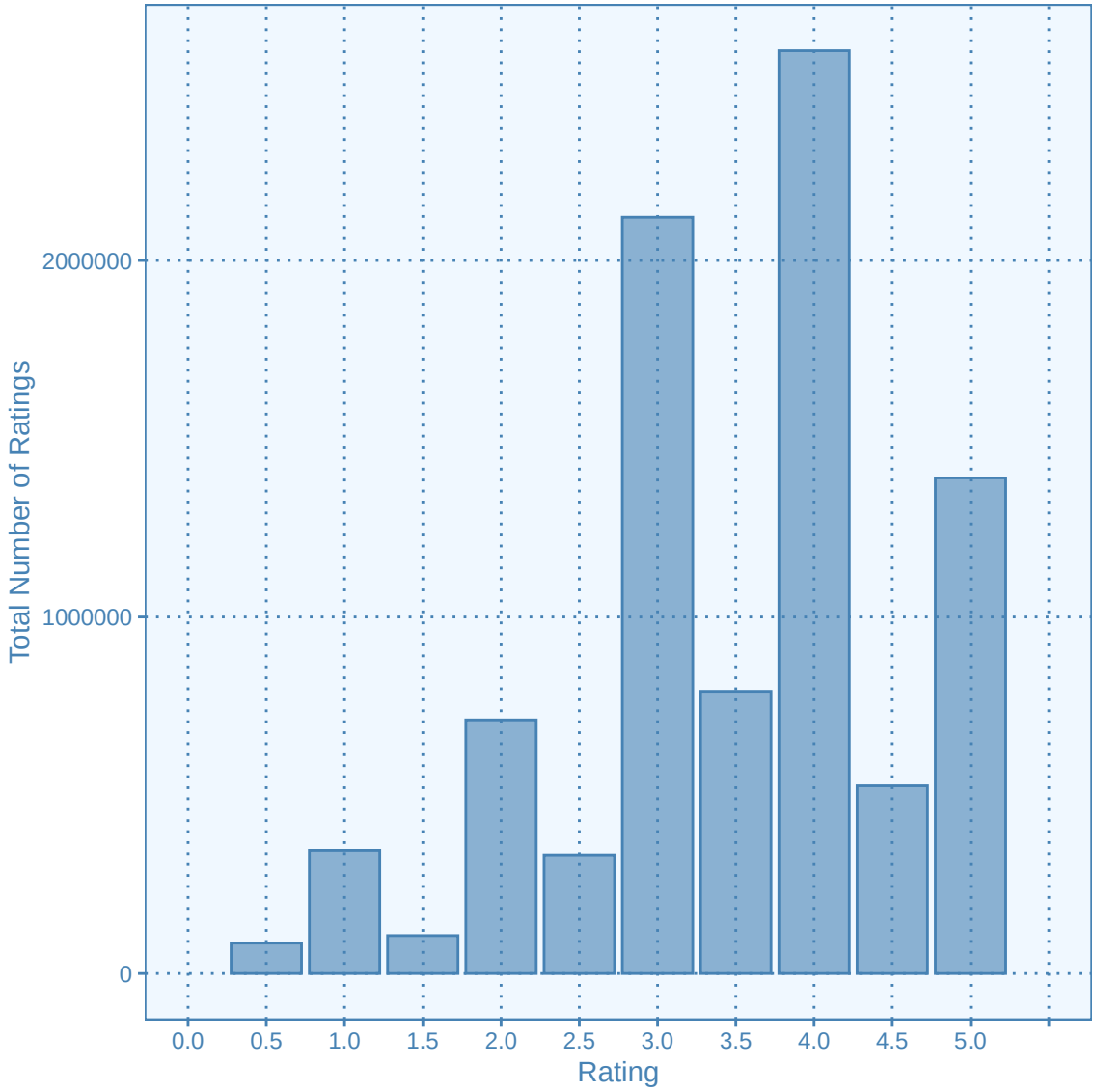

Figure 3: Rating distribution

It can be clearly seen from Figure 3 that people generally rate movies as whole integers. However approximately 20% of the ratings are given with fractional ratings. This could cause issues with predictions.

5.6 Different genres

The movies are described with a collection of genres. The maximum number of genres on any one movie in the *edx* dataset is

```
max(str_count(edx$genres, "\\|")) + 1
```

```
## [1] 8
```

The names of the genres provided in the *edx* dataset are

```
max_genres_per_movie <- max(str_count(edx$genres, "\\|")) + 1
genre_title_vector <- as.character(1:max_genres_per_movie)
genre_names <- edx %>%  select(genres) %>% unique %>%
  separate(genres, genre_title_vector, sep="\\|") %>%
  gather(to_remove, genre, genre_title_vector) %>% filter(!is.na(genre)) %>%
  select(-to_remove) %>% unique %>% filter( !grepl("no genres listed", genre) )
```

```
## Warning: Expected 8 pieces. Missing pieces filled with `NA` in 796 rows [1, 2,
## 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(genre_title_vector)` instead of `genre_title_vector` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
genre_names
```

```
##          genre
## 1       Comedy
## 2       Action
## 3     Children
## 4    Adventure
## 5    Animation
## 6        Drama
## 7        Crime
## 8       Sci-Fi
## 9       Horror
## 10    Thriller
## 11   Film-Noir
## 12     Mystery
## 13     Western
## 14 Documentary
## 15     Romance
## 16     Fantasy
## 17     Musical
## 18         War
## 19        IMAX
```

```
genre_names_rda <- "genre_names.rda"
save(genre_names, file=genre_names_rda)
```

There are varying quantities of movies in each genre which can be seen in Figure 4

```
#load(genre_names_rda)
t <- 1:length(genre_names)
df <- data.frame(genre = genre_names, total = t)

for(i in 1:nrow(genre_names)) {
  genre <- genre_names[i,]
  df$total[i] <- edx %>% select(genres) %>%
    filter( grepl(genre, genres)) %>% count
}

options(scipen=10000000)
bar_graph(data=df, x=reorder(df$genre, -as.numeric(df$total)),
```

```
        y=as.numeric(df$total)) +
ylab("Total Number of Movies") +
xlab("Genre (Ordered by Descending Number of Movies)") +
scale_x_discrete() +
theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```

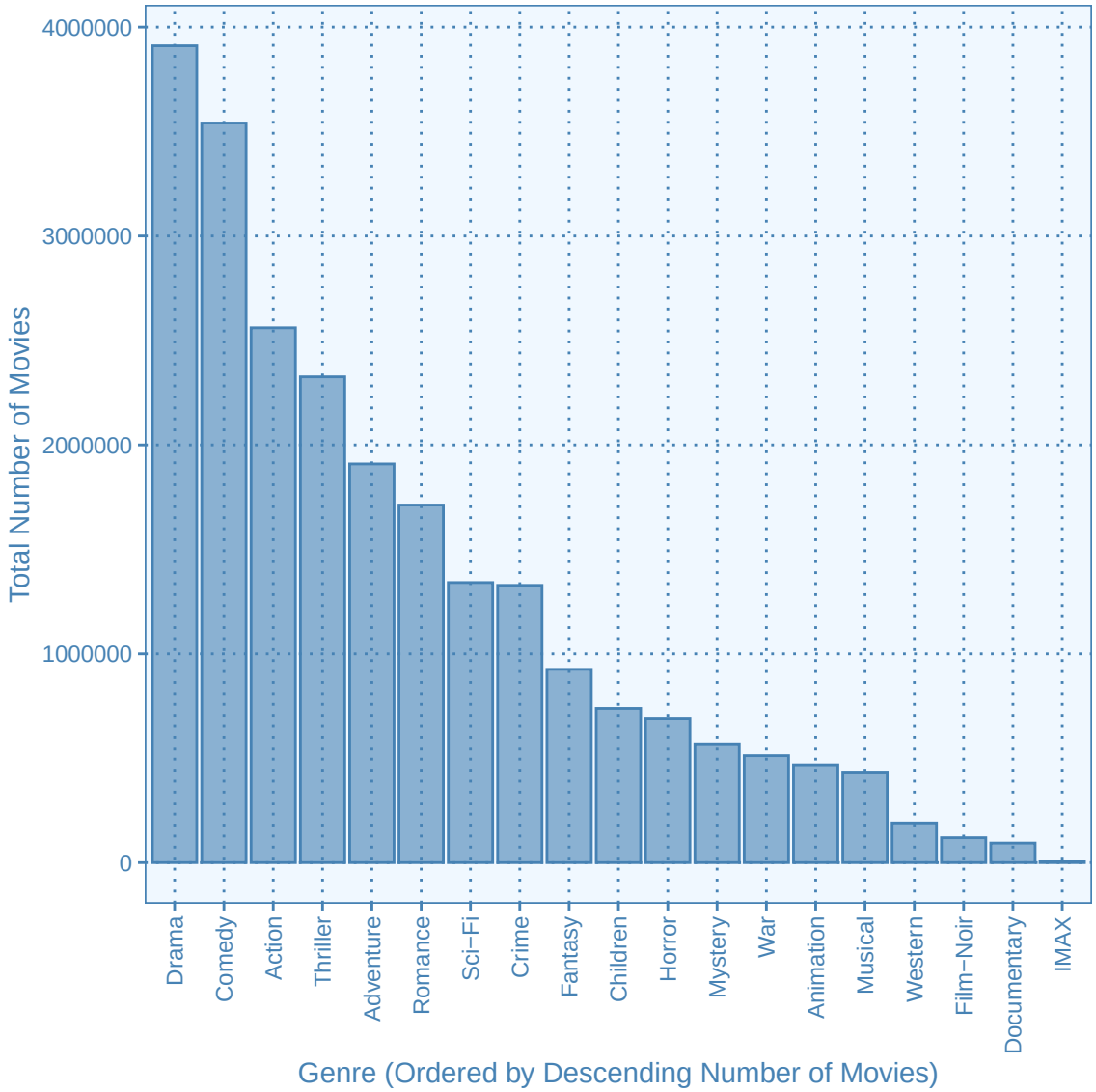

Figure 4: Total number of movies per genre

NB: It is important to note at this time that if the *validation* dataset contains movies which have more types of genres or contains movies which have a larger number of genres, the generated model may not work properly.

6 Reshaping Datasets

The initial investigation into the structure of the *edx* dataset shows two interesting colums.

- There is a release year of the movie embedded inside the *title*
- There is a *timestamp* of the review

The year that the movie was released can be extracted from the *title* field using regular expressions. The *timestamp* of the review can be converted into a number of fields including the year of the review, the month of the review, the day of the month of the review, and the day of the week of the review by using methods on the *lubridate* library.

By subtracting the year of the movie's release from the year of the review, the age of the movie at the time the review was made could be calculated.

These date-type pieces of information could give insight into the data. This will be described in more detail in Section 7.

6.1 Function to *reshape* data as necessary

However, at this point, we will only describe the *reshape* function which was written to reshape the data into the required format:

```
reshape <- function (dataset) {
  print(paste("reshaping nrow=", nrow(dataset), sep=" "))
  tmp <- dataset %>%
    mutate(review_date = as.Date(as.POSIXct(as.numeric(timestamp), origin="1970-01-1"))) %>%
    mutate(
           review_year = year(review_date),
           review_month = month(review_date),
           review_day = day(review_date),
           review_week_day = wday(review_date)) %>%
    select(-timestamp) %>%
    extract(title, c("short_title", "release_year"), regex="(^.*) \\((\\d\\d\\d\\d)\\)") %>%
    mutate(release_year = as.numeric(release_year)) %>%
    mutate(span_years = as.numeric(review_year) - as.numeric(release_year))
  print("date extracted from title and timestamp split into smaller pieces of data");
  rm(dataset)
  print("removed original dataset to free up memory.  Reshaped data looks like")
  print(head(tmp, 3))
  tmp
}
```

The *edx* dataset is transformed using this function and the newly reshaped dataset is stored as an intermediate R object.

6.2 Conditional code to *reshape* various datasets

```
#
# pseudo-constant defined at top of script
#
edx_reshaped_rda <- "edx_reshaped.rda"

#
# reshape the edx data when necessary
#
if ( !file.exists(edx_reshaped_rda)) {
```

```
   print("reshaping edx data" )
   load(edx_rda)
   edx_reshaped <- reshape(edx)
   rm(edx)
   save(edx_reshaped, file = edx_reshaped_rda)
   rm(edx_reshaped)
   print(paste("reshaped edx data saved on", edx_reshaped_rda, sep=" "))
}
```

The reshaped *edx_reshaped* dataset now looks like

```
str(edx_reshaped)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  12 variables:
##  $ userId         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId        : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating         : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ short_title    : chr  "Boomerang" "Net, The" "Outbreak" "Stargate" ...
##  $ release_year   : num  1992 1995 1995 1994 1994 ...
##  $ genres         : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Ac
##  $ review_date    : Date, format: "1996-08-02" "1996-08-02" ...
##  $ review_year    : num  1996 1996 1996 1996 1996 ...
##  $ review_month   : num  8 8 8 8 8 8 8 8 8 8 ...
##  $ review_day     : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ review_week_day: num  6 6 6 6 6 6 6 6 6 6 ...
##  $ span_years     : num  4 1 1 2 2 2 2 2 2 2 ...
```

A sample of the data looks like

```
head(edx_reshaped, 3)
```

```
##    userId movieId rating short_title release_year                      genres
## 1:      1     122      5   Boomerang         1992               Comedy|Romance
## 2:      1     185      5    Net, The         1995        Action|Crime|Thriller
## 3:      1     292      5    Outbreak         1995 Action|Drama|Sci-Fi|Thriller
##    review_date review_year review_month review_day review_week_day span_years
## 1:  1996-08-02        1996            8          2               6          4
## 2:  1996-08-02        1996            8          2               6          1
## 3:  1996-08-02        1996            8          2               6          1
```

7 Exploratory Analysis

7.1 Release year vs rating

The author was curious if there was a relationship between the year that a movie was released and the average rating that the movie received.

The Figure 5 displays the year of the release of a movie on the horizontal axis and the mean (also known as the average) of the ratings of all movies released during that year.

```
release_year_vs_rating <- edx_reshaped %>% group_by(release_year) %>%
    summarize(mean_rating_by_release_year = mean(rating))
rating_graph(data = release_year_vs_rating,
             x=release_year_vs_rating$release_year,
             y=release_year_vs_rating$mean_rating_by_release_year) +
  ggtitle("Year of Release vs Average Rating") +
  xlab("Year of Release")
```

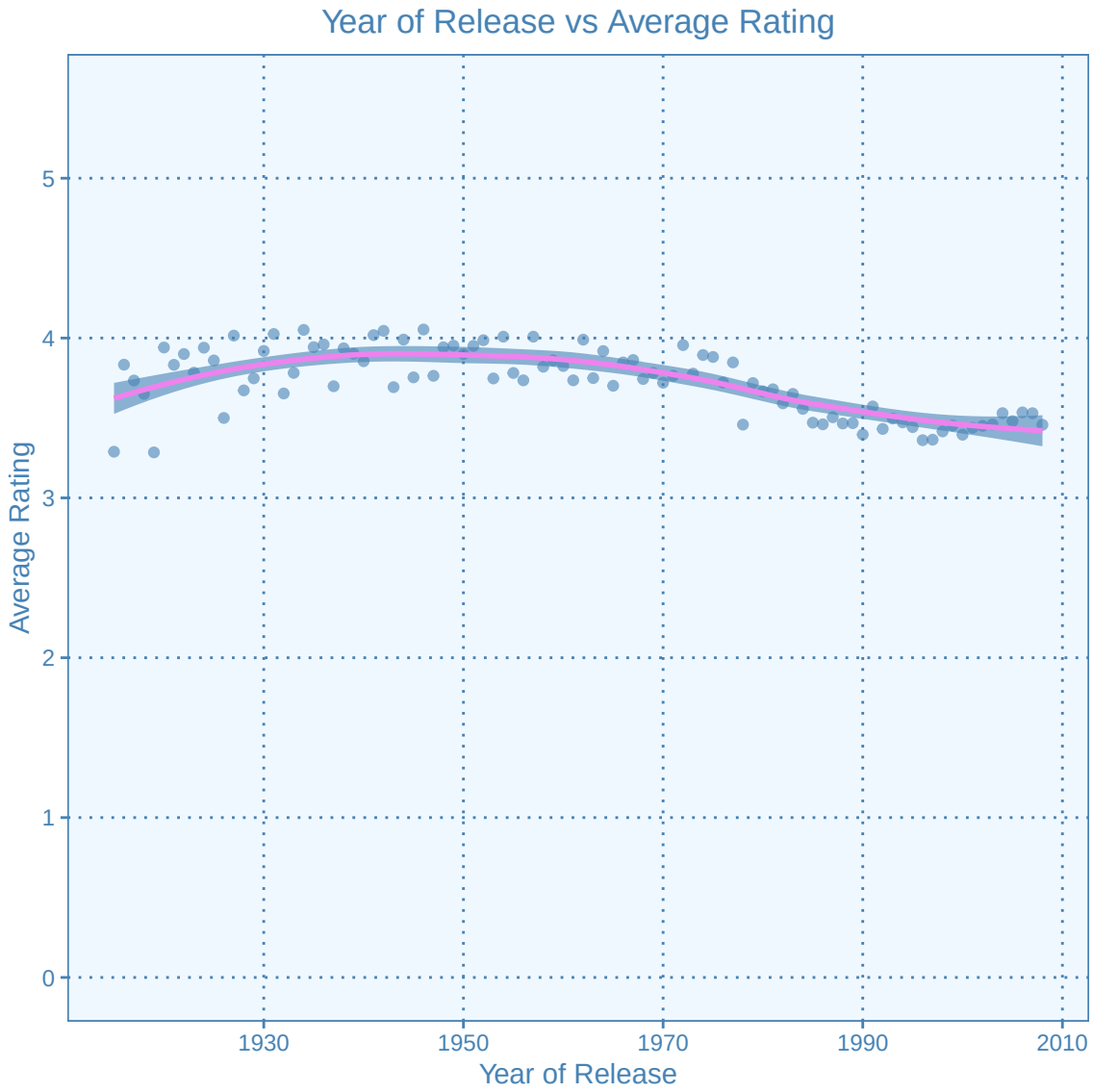

Figure 5: Release year vs average rating

On the rating scale of one to five, there is no significant change in the average ratings given the year of the release of a movie.

7.2 Review year vs rating

The Figure 6 compares the year of review (the year the reviewer or user wrote the review and not the year the movie was released) and the average rating of all movies reviewed during that year

```
review_year_vs_rating <- edx_reshaped %>% group_by(review_year) %>%
    summarize(mean_rating_by_review_year = mean(rating))
rating_graph(data=review_year_vs_rating,
             x=review_year_vs_rating$review_year,
             y=review_year_vs_rating$mean_rating_by_review_year) +
  ggtitle("Year of Review vs Average Rating") +
  xlab("Year of Review")
```

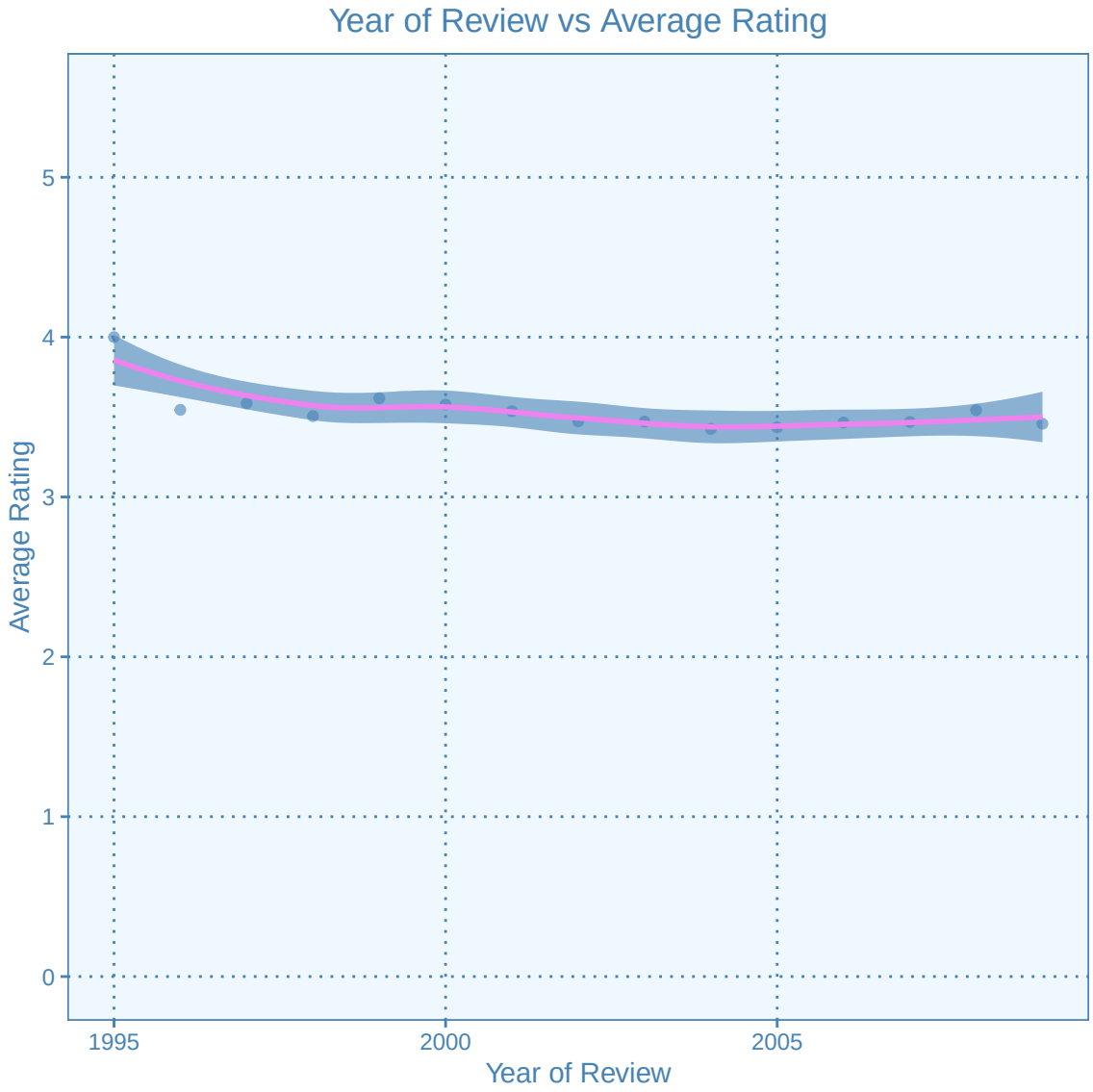

Figure 6: Year of review vs average rating

Again, on a rating scale of one to five, there is no significant changes in the average rating when observed through the year the movie was reviewed.

Comparing Figure 6 and Figure 5, it can be seen that the reviews only started in 1995 whilst releases of the movies start in 1915. This may be important in future analysis.

7.3 Age of movie vs rating

The age of a movie at the time of review can be calculated by subtracting the year the movie was released from the year the review was written. This may be important. Movies which may have *flopped* when they were initially released may have had a resurgence of positive ratings after a period of time.

Figure 7 shows the relationship between the age of a movie in years when it was rated and the average rating it was given.

```
age_of_movie_when_rated <- edx_reshaped %>% group_by(span_years) %>%
  summarize(mean_rating_by_age = mean(rating))
rating_graph(data=age_of_movie_when_rated,
             x=age_of_movie_when_rated$span_years,
             y=age_of_movie_when_rated$mean_rating_by_age) +
  ggtitle("Age of Movie when Rated vs Average Rating") +
  xlab("Age of Movie when Rated in Years")
```

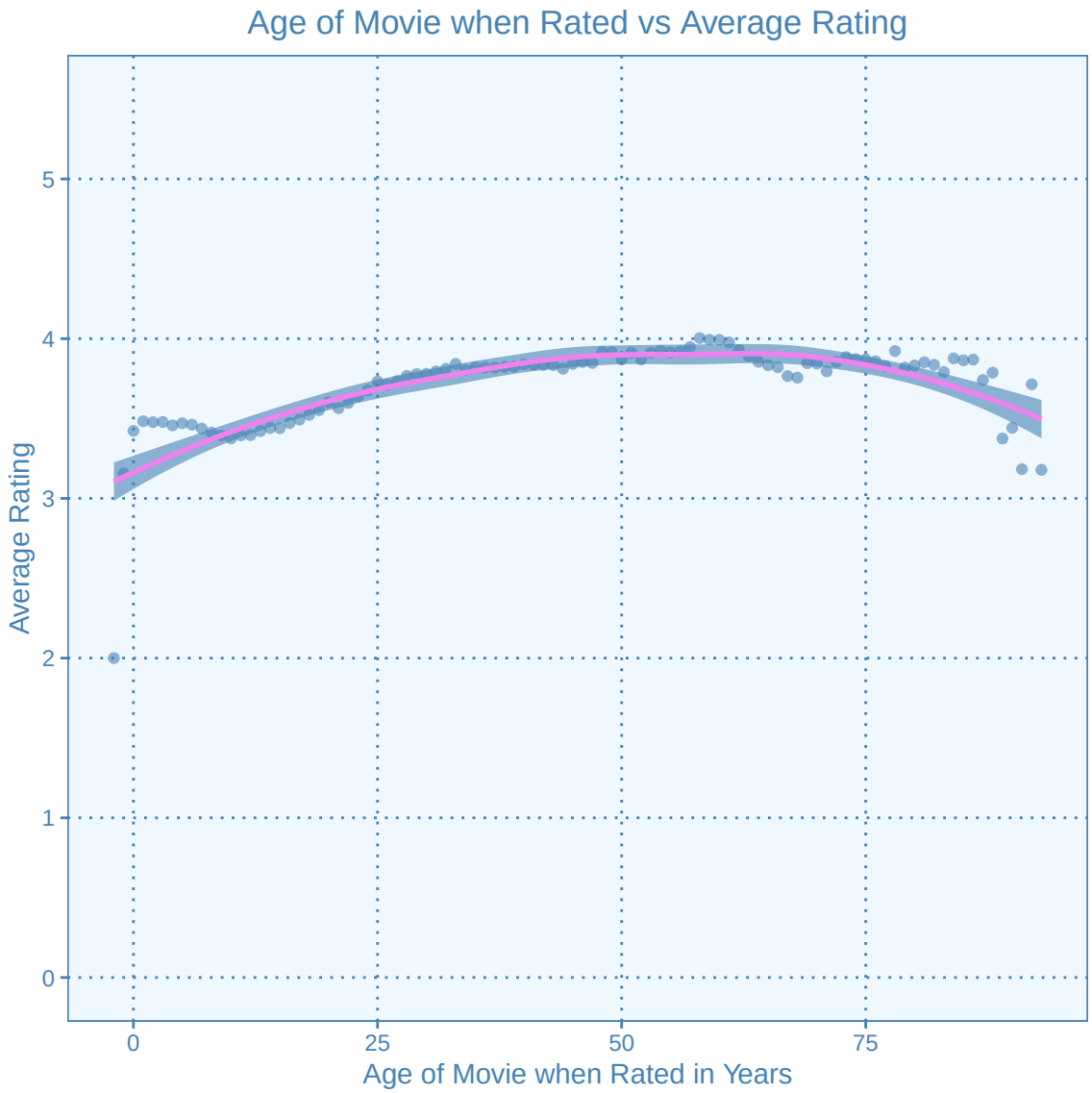

Figure 7: Age of movie vs average rating

Again, on a rating scale of one to five, there is no significant change in the average rating values of movies when compared to the age of the movie at the time it was reviewed.

7.4 Specific genre and year vs rating

The mores of a country change over time. The author wondered specifically about War movies and Science Fiction movies with respect to certain periods of time. Considering War movies, for example, it is possible that in the period of time when there is a *good war* (such as the 1940s during and immediately after World War II) that ratings for war movies might be higher than when there is an unpopular war (such as the period from approximately 1965 to 1975 during the Vietnam War). Although the actual reviews only started in 1995, the time period right after World War II and the Vietnam War could not be tested. However, there is a possibility that War movies might have received a higher than normal rating right after September 11, 2001. Figure 8 shows the average rating of War movies through the years.

```
war_movies_by_years <- edx_reshaped %>% filter(str_detect(genres, "War") ) %>%
  group_by(review_year) %>%
  summarize(war_movies_mean_rating_by_year = mean(rating))
rating_graph(data=war_movies_by_years,
             x=war_movies_by_years$review_year,
             y=war_movies_by_years$war_movies_mean_rating_by_year) +
  ggtitle("War Movies by Year of Review vs Average Rating") +
  xlab("War Movies Review Year")
```

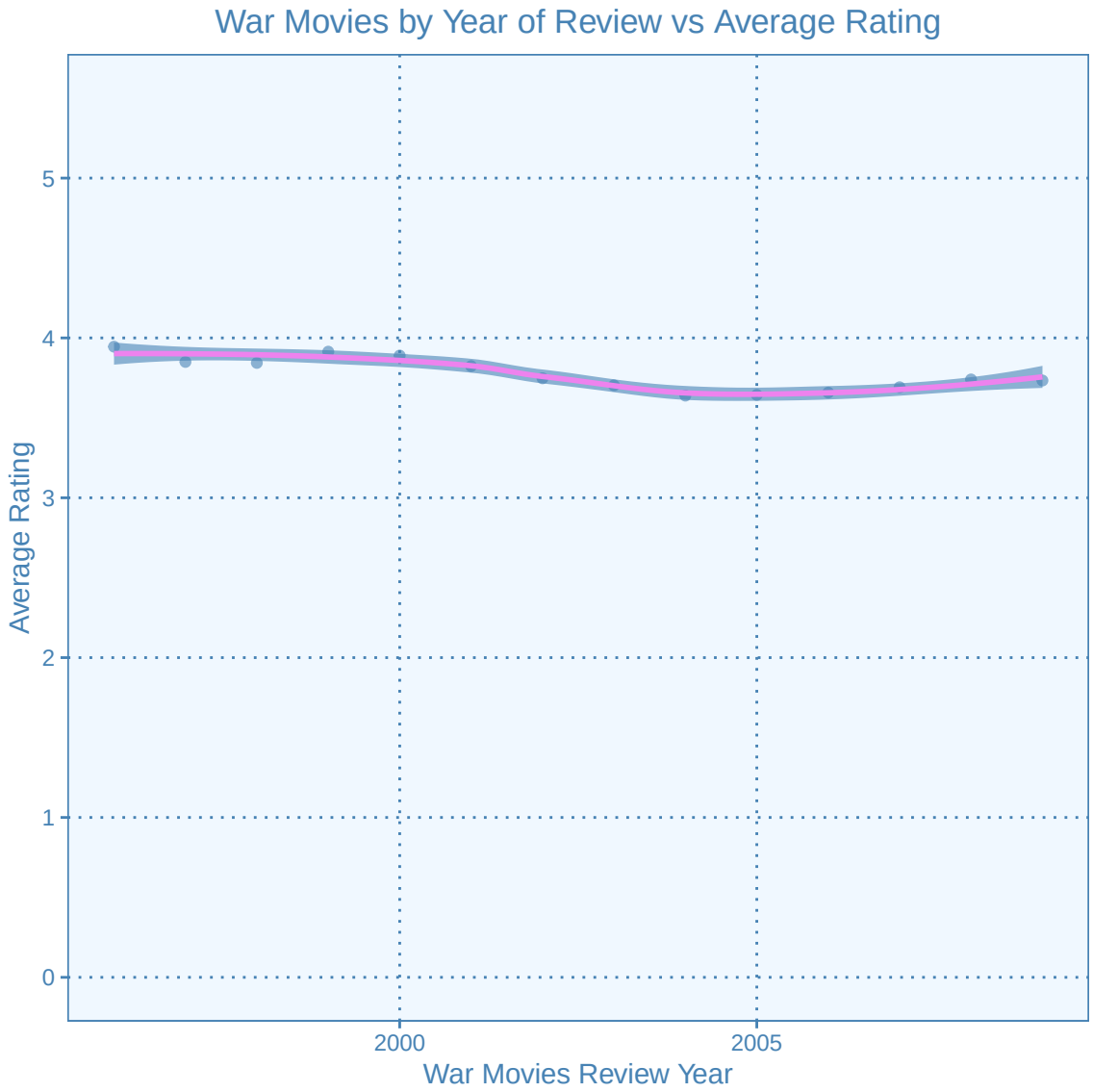

Figure 8: War movies by year vs average rating

Again, on a rating scale of one to five, Figure 8 shows that an event such as September 11, 2001, did not substantially influence ratings of War movies.

With respect to Science Fiction movies, since 1995 there have been no real substantial science events that have grabbed the popular attention as did, for example, Apollo moon missions (late 1960s). Figure 9 shows Science Fiction movies through the years with their average ratings.

```
scifi_movies_by_years <- edx_reshaped %>% filter(str_detect(genres, "Sci-Fi") ) %>%
  group_by(review_year) %>%
  summarize(scifi_movies_mean_rating_by_year = mean(rating))
rating_graph(data=scifi_movies_by_years,
             x=scifi_movies_by_years$review_year,
             y=scifi_movies_by_years$scifi_movies_mean_rating_by_year) +
  ggtitle("Sci-Fi Movies by Year of Review vs Average Rating") +
  xlab("Sci-Fi Movies Review Year")
```

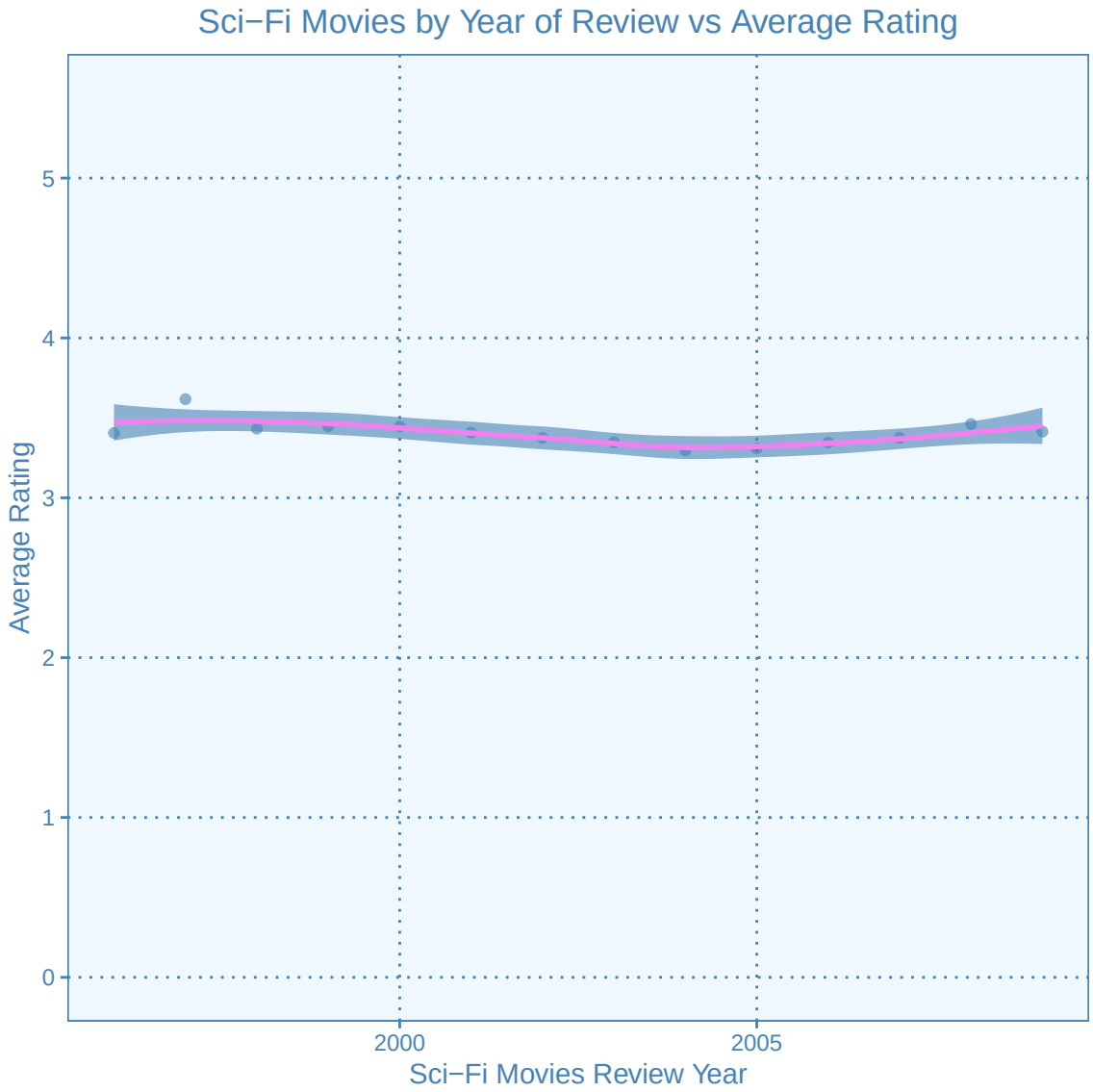

Figure 9: Sci-Fi movies by year vs average rating

Again, on a rating scale of one to five, there is no substantial change in the average ratings of science fiction movies over the years.

7.5 Specific genre and month vs rating

The author wondered if certain months might be influential in the ratings over certain genres of movies. With specific respect to Romance movies, the author wondered if the *Christmas season* or the *Summer season* might influence ratings. This is because the movie industry has two sub-genres of Summer Romance movies and Christmas Romance movies. Figure 10 shows Romance movies by month and their average ratings.

```
romance_movies_by_months <- edx_reshaped %>% filter(str_detect(genres, "Romance") ) %>%
  group_by(review_month) %>%
  summarize(romance_movies_mean_rating_by_month = mean(rating))
rating_graph(data=romance_movies_by_months,
             x=romance_movies_by_months$review_month,
             y=romance_movies_by_months$romance_movies_mean_rating_by_month) +
  ggtitle("Romance Movies by Month of Review vs Average Rating") +
  xlab("Romance Movies Review Month") +
  scale_x_continuous(limits=c(1, 12), breaks=1:12,
            labels=c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                     "Jul", "Aug", "Sep", "Oct", "Nov", "Dec") )
```

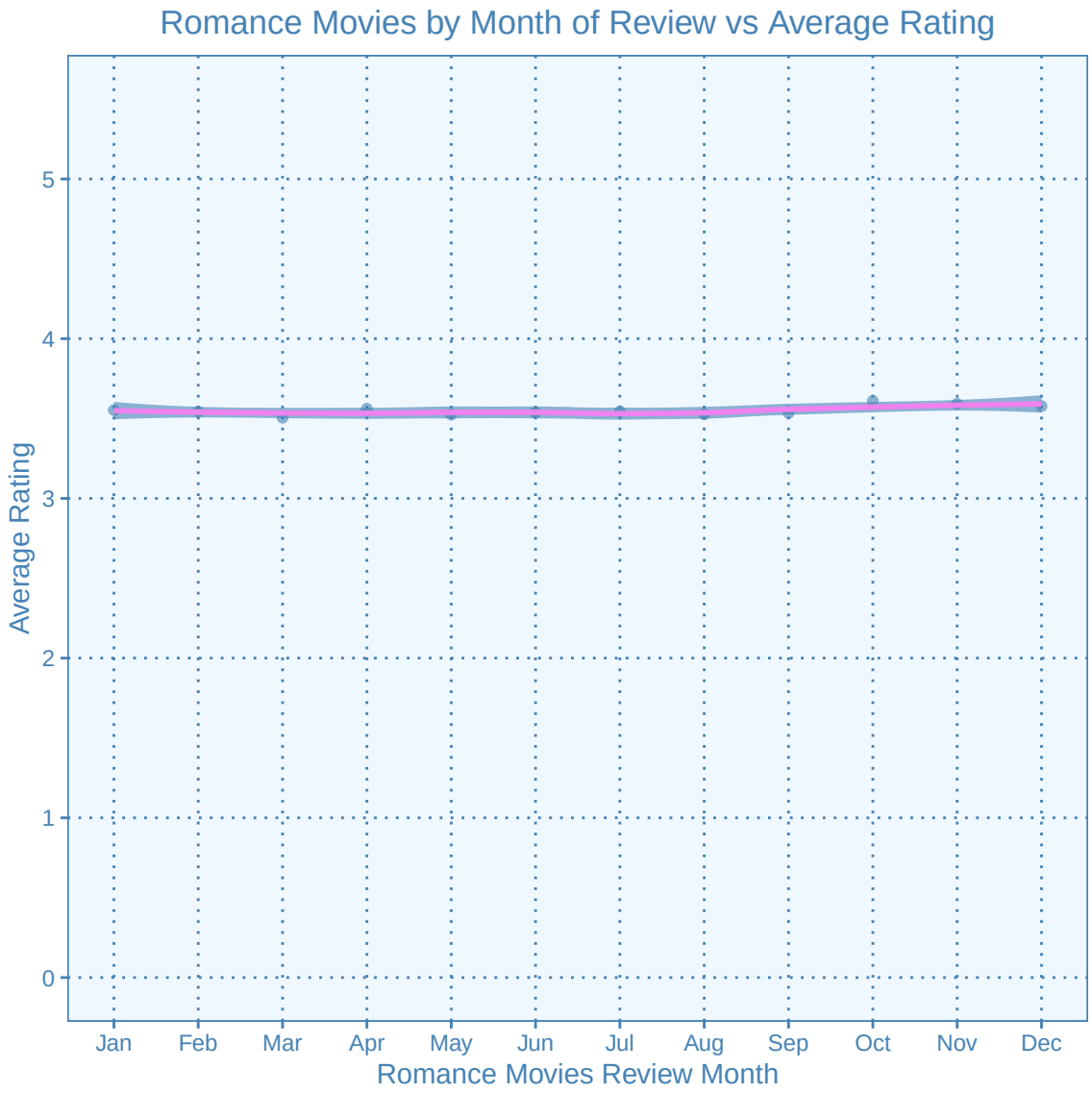

Figure 10: Romance movies by month vs average rating

Again, on a rating scale of one to five, there is no substantial influence in the rating of Romance movies due to the month of the year.

7.6 Day of the month vs rating

There is a possibility that people give more positive ratings to movies when they themselves are in a more positive state of mind. This state of mind could be influenced by personal financial issues and be related to payday. Figure 11 shows average movie ratings with respect to the day of the month.

```
movie_ratings_by_days <- edx_reshaped %>%
  group_by(review_day) %>%
  summarize(movie_rating_by_day = mean(rating))
rating_graph(data=movie_ratings_by_days,
             x=movie_ratings_by_days$review_day,
             y=movie_ratings_by_days$movie_rating_by_day) +
  geom_point() + ggtitle("Day of the Month of Review vs Average Rating") +
```

```
  xlab("Day of the Month") +
  scale_x_continuous(limits=c(1, 31), breaks=1:31)
```

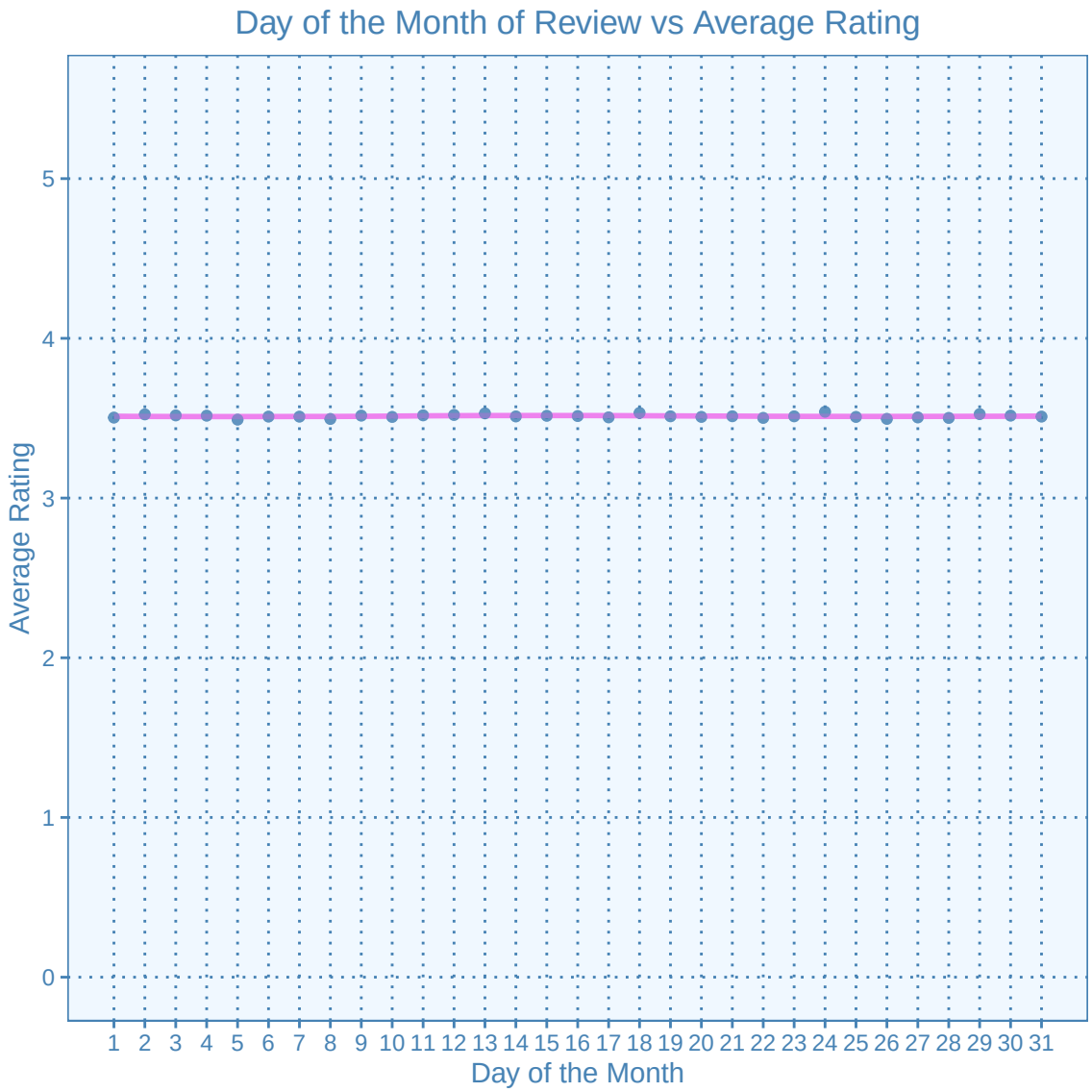

Figure 11: Day of the month vs average rating

Again, on a rating scale of one to five, there is no substantial relationship between the day of the month and the average ratings given to a movie.

7.7 Day of the week vs rating

The author thought there might a possibility that people give more positive ratings to movies when they are not stressed about work related issues (unrelated to payday related issues). There might be a relationship between *weekends* and *workdays*. Figure 12 looks at the day of the week and average ratings.

```
movie_ratings_by_day_of_week <- edx_reshaped %>%
  group_by(review_week_day) %>%
  summarize(movie_rating_by_day_of_week = mean(rating))
```

```
rating_graph(data=movie_ratings_by_day_of_week,
             x=movie_ratings_by_day_of_week$review_week_day,
             y=movie_ratings_by_day_of_week$movie_rating_by_day_of_week) +
  geom_point() + ggtitle("Day of the Week of Review vs Average Rating") +
  xlab("Day of the Week") +
  scale_x_continuous(limits=c(1, 7), breaks=1:7,
                     labels=c("Sun", "Mon", "Tue", "Wed", "Thu",
                              "Fri", "Sat"))
```

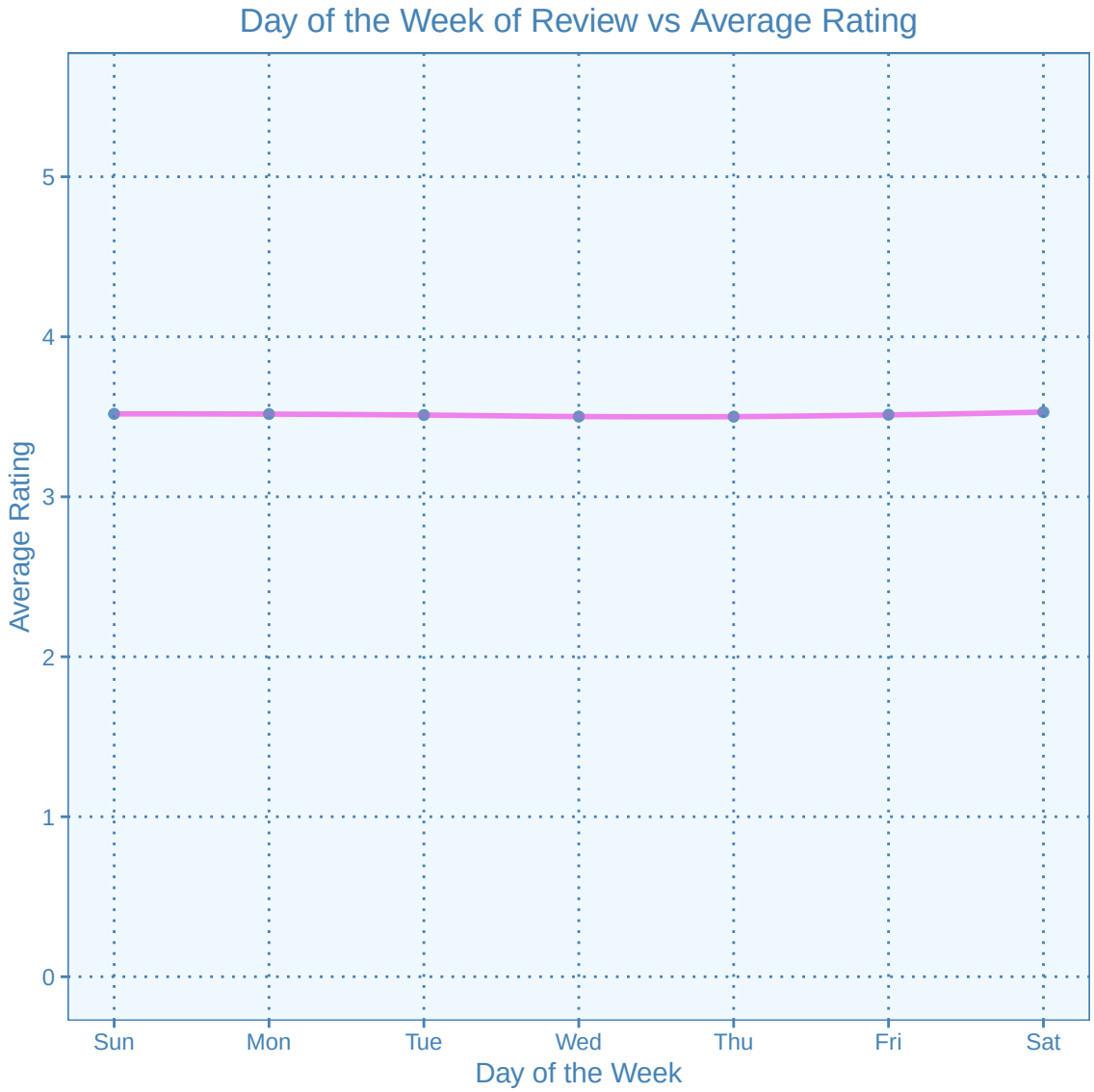

Figure 12: Day of the week vs average rating

Again, on a rating scale of one to five, there is no substantial relationship between day of the week and average ratings given to movies.

7.8 Genre vs rating

Although the author has previously looked specifically at War movies (Figure 8), Science Fiction movies (Figure 9), and Romance movies (Figure 10), Figure 13 displays average rating across all movies in all the possible genres. The horizontal axis is displayed in decreasing rating value.

```
genre_ratings <- vector(mode="numeric", length=length(genre_names))
load(edx_reshaped_rda)
for(i in 1:nrow(genre_names) ) {
  edited_genre <- genre_names[i,]
  movie_rating_by_genre <- edx_reshaped %>% filter(str_detect(genres, edited_genre) ) %>%
    summarize(genre_rating = mean(rating))
  genre_ratings[i] <- movie_rating_by_genre$genre_rating
}
data <- data.frame(genre_names, genre_ratings)


rating_graph(data=data,
             x=reorder(data$genre, -data$genre_ratings),
             y=data$genre_ratings) +
  geom_point() + ggtitle("Genre vs Average Rating") +
  xlab("Genre (Ordered by Descending Average Rating)") +
  scale_x_discrete() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```

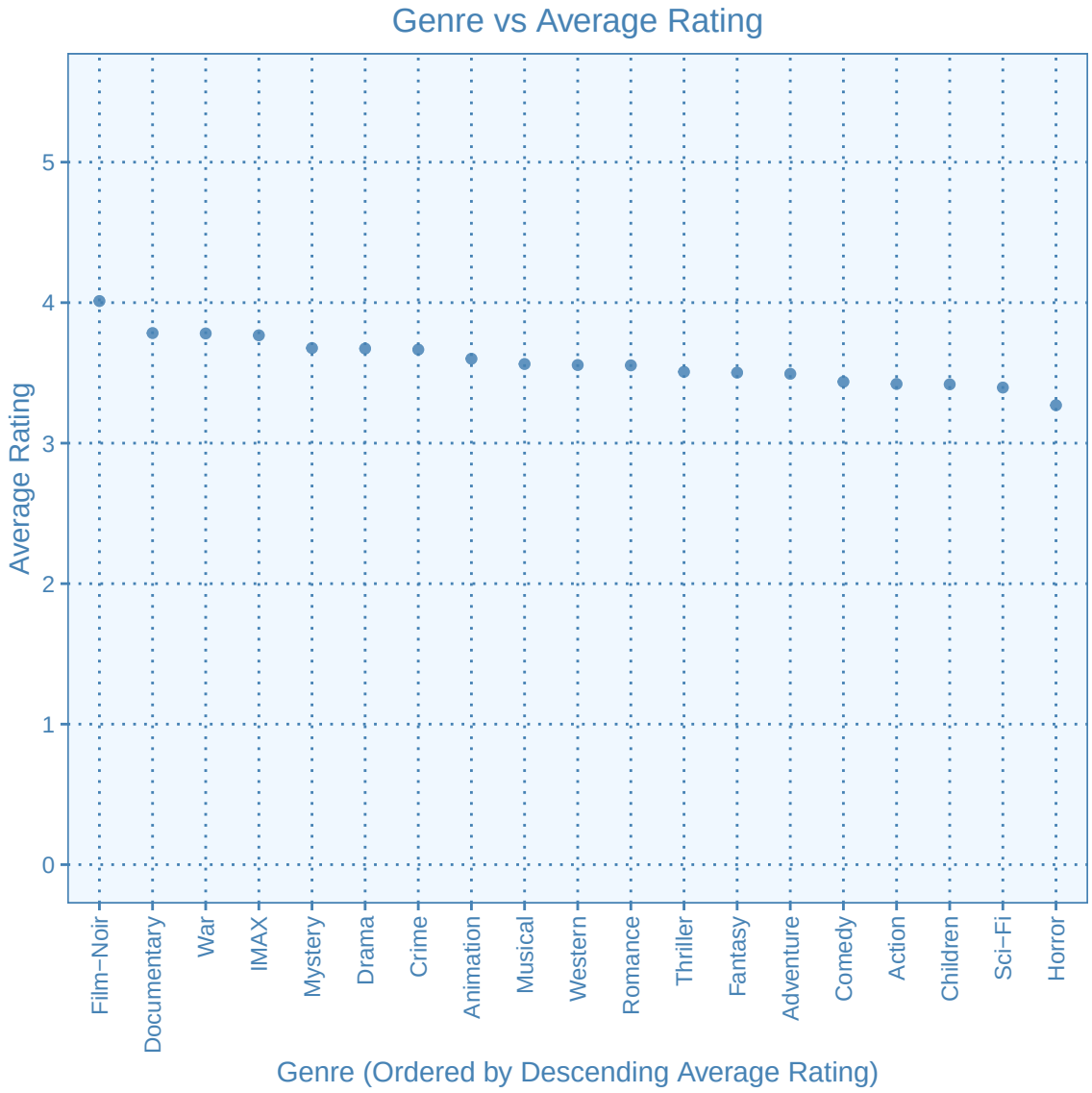

Figure 13: Genre vs average rating

Again, over a rating scale of zero to five, there is no substantial change in ratings when viewed in relation to the genre of the movie.

7.9 Reviewer vs rating

Figure 14 shows the relationship between individual reviewers or users (the horizontal axis) and the average rating given to movies by that user (the vertical axis). The horizontal axis is sorted by descending average rating values given by that user.

```
movie_ratings_by_userId <- edx_reshaped %>%
  group_by(userId) %>%
  summarize(movie_rating_by_userId = mean(rating))
rating_graph(data=movie_ratings_by_userId,
     x=reorder(movie_ratings_by_userId$userId, -movie_ratings_by_userId$movie_rating_by_userId),
     y=movie_ratings_by_userId$movie_rating_by_userId) +
  geom_point() + ggtitle("Reviewer vs Average Rating") +
```

```
  xlab("Reviewer (Ordered by Descending Average Rating)") +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        panel.grid.major.x =  element_line(colour = "white", linetype = 3, size = 0.5),
        panel.grid.minor.x = element_blank())
```

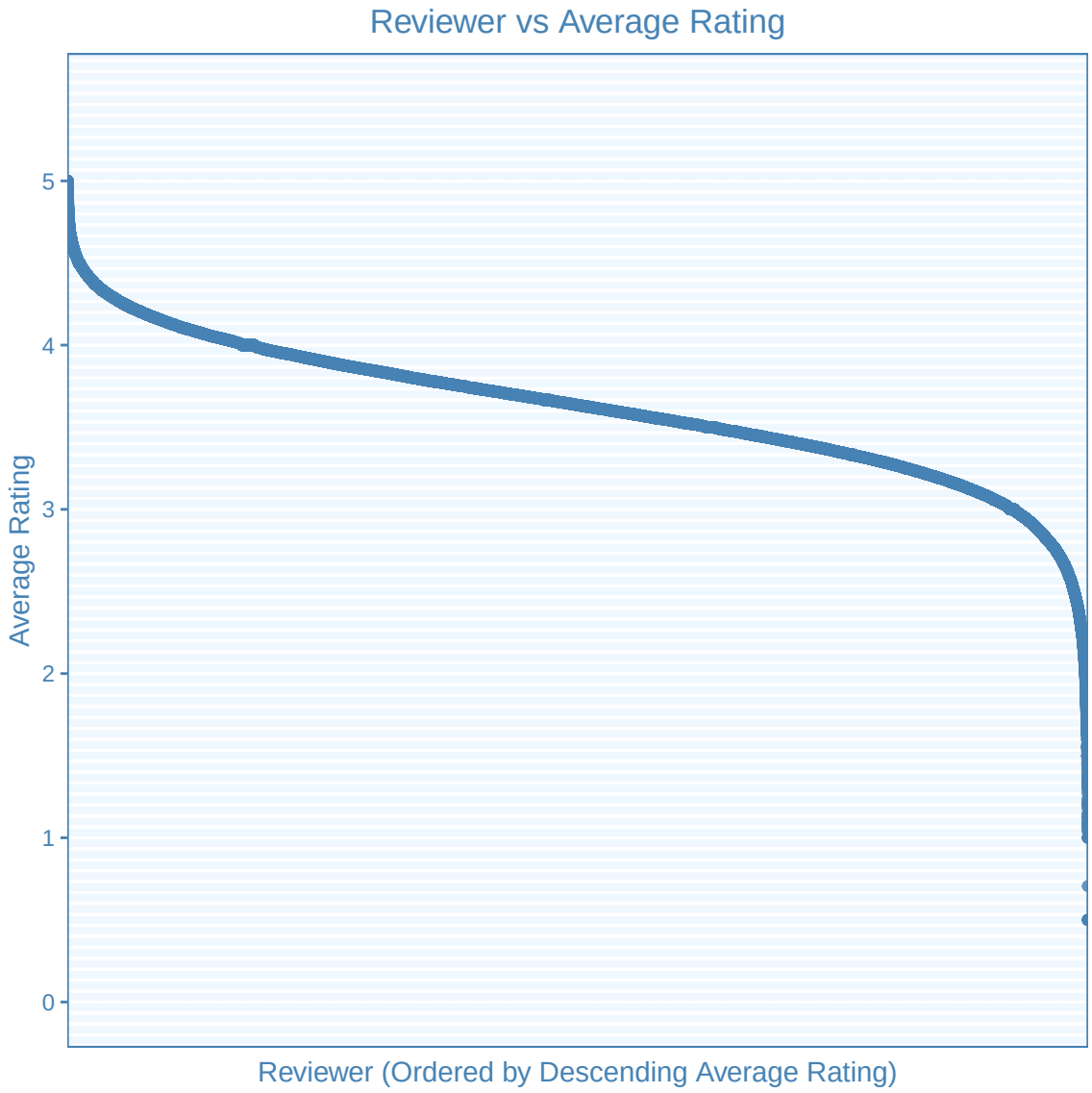

Figure 14: Reviewer (user) vs average rating

From Figure 14 it is clear that there is a relationship between the reviewer and the average rating that the reviewer gives to movies. In other words, certain users or reviewers have a tendancy to rate movies higher than other users or reviewers. This will definitely have to be investigated when the models are developed

7.10 Movie vs rating

Figure 15 shows the relationship between the individual movies themselves (displayed on the horizontal axis) and the average rating that users have given that movie (displayed on the vertical axis). The horizontal axis is sorted in descending order by the average rating that the movie received.

```r
movie_ratings_by_movieId <- edx_reshaped %>%
  group_by(movieId) %>%
  summarize(movie_rating_by_movieId = mean(rating))
rating_graph(data=movie_ratings_by_movieId,
     x=reorder(movie_ratings_by_movieId$movieId,
        -movie_ratings_by_movieId$movie_rating_by_movieId),
     y=movie_ratings_by_movieId$movie_rating_by_movieId) +
  geom_point() + ggtitle("Movie vs Average Rating") +
  xlab("Movie (Ordered by Descending Average Rating)") +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        panel.grid.major.x =  element_line(colour = "white", linetype = 3, size = 0.5),
        panel.grid.minor.x = element_blank())
```

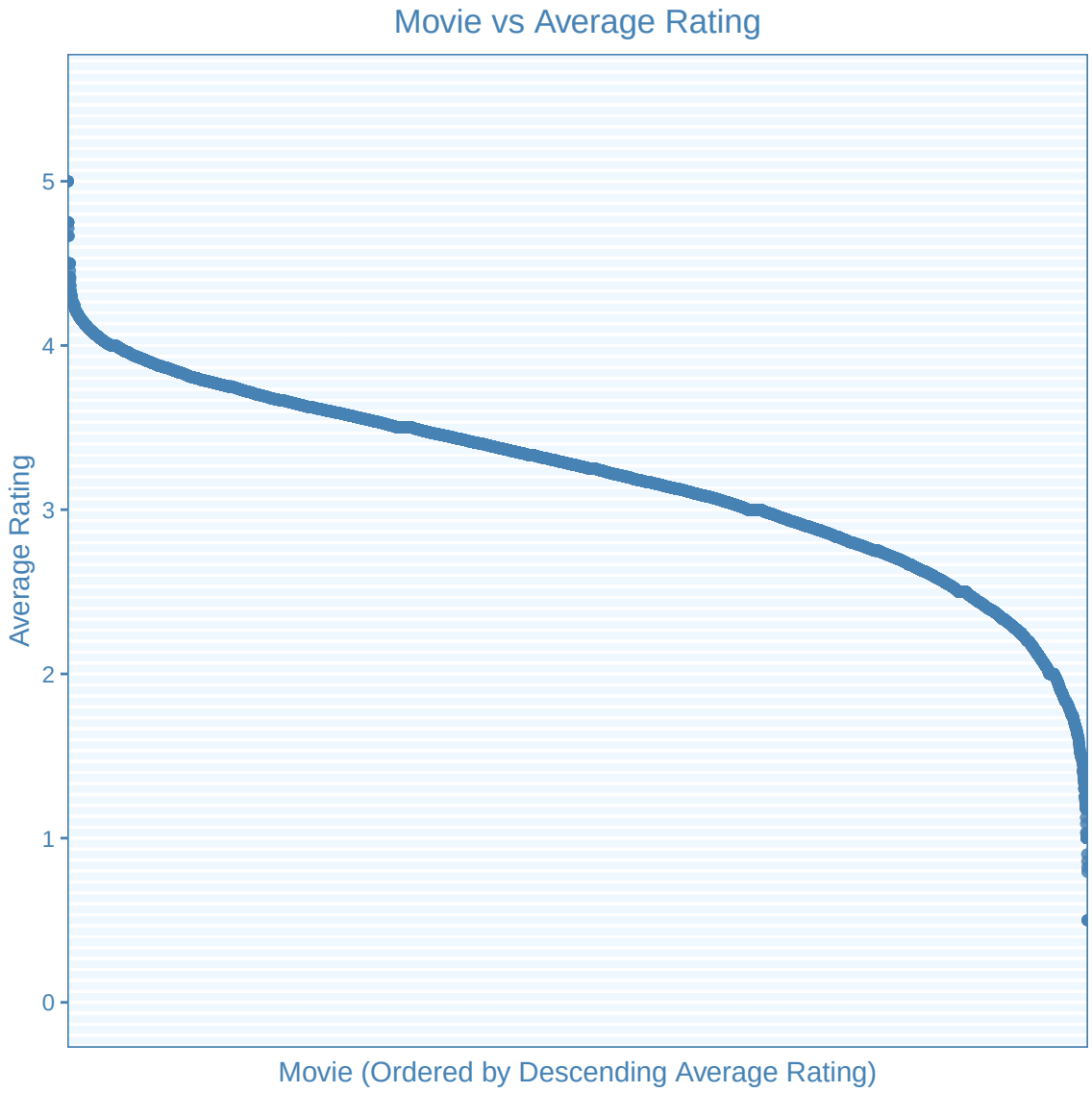

Figure 15: Movie vs average rating

From Figure 15 it is clear that there is a relationship between the movie itself and the average rating that the movie receives. This makes intuitive sense. Some movies are more popular than other movies. This fact will need to be investigated in detail when the models are developed.

7.11 Summary of exploratory analysis

From the aforegoing, it is clear that of all the columns in the reshaped dataset, it is only the reviewer (or user) and the movie itself which appear to influence the rating of a movie in any substantial way. Fields such as the year of the release of the movie or the year of the review have negligible effect on the average ratings.

However, the reviewer and the movie both have major influence. This will have to be investigated when the models are developed.

8 Model Investigations and Results

This section describes the investigation into which model could be used to predict ratings. Eight models were developed each with an improving (decreasing) RMSE (root mean square error).

From the analysis done in Section 7, it is clear that only the reviewer (identified by the *userId* column) and the movie itself (identified by the *movieId* column) affect the rating of a movie in any substantial way. The remaining columns in the dataset will be ignored.

To evaluate each model, the following function was used:

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

Each model will be reported in this section

8.1 Splitting *edx* data into *train* and *test*

The *edx* dataset was split into a training (*train*) and testing (*test*) dataset (leaving the *validation* set untouched). This was done with

```
#
# set up some pseudo constants that are used throughout
#
train_rda <- "train.rda"
test_rda <- "test.rda"
if ( !file.exists(train_rda) || !file.exists(test_rda) ) {
        load(edx_rda)
        #
        # break the edx file into a train and test set
        #
        set.seed(1, sample.kind="Rounding")
        #
        #
        test_index <- createDataPartition(edx$rating, times = 1, p = 0.25, list = FALSE)
        train <- edx[-test_index,]
        test <- edx[test_index,]
        print(paste("edx has ", nrow(edx), sep=" "))
        print(paste("train has ", nrow(train), sep=" "))
        print(paste("test has ", nrow(test), sep=" "))
        save(train, file=train_rda)
        save(test, file=test_rda)
        rm(edx, train, test)                    # clean up memory
}
```

25% of the *edx* dataset was retained as test data.

It is important to note that due to the findings which are documented in Section 7, the training set and test set do not have to be reshaped. They can be reshaped but the new columns will be ignored.

8.2 Average rating (naive approach)

For this simple model, the mean (average) across all movies in the training set was calculated.

The first attempt to generate a model to predict ratings was simply to take the mean or average of all movies on the training dataset. This value would be then applied against all elements of the test dataset and an RMSE would be calculated. Irizarry (2021) calls this model a *naive approach* and refers to the overall average

or mean as *mu*. The author of this paper as a follower of Knuth's *literate programming* paradigm, however, uses the term *overall_average* within the R source code.

```
overall_average <- mean(train$rating)
rmse <- RMSE(test$rating, overall_average)
print(rmse)
```

```
## [1] 1.060299
```

The RMSE of this first model was 1.0602987.

8.3 Average rating by user (user effect)

For this second model, the mean across movies which were rated by a specific user was calculated. Figure 14 clearly shows there is a relationship between the reviewer and the average rating that a movie received by that reviewer. This could be known as the *user effect* by (Irizarry, 2021) but will be called *user_averages* by the author of this paper. Only information about the user is used to determine this model.

When the model needs to predict a rating for this specific user, then this value is used. This could be calculated for every element in the *train* dataset and compared against the ratings in the *test* dataset. The RMSE could be calculated.

```
user_averages <- train %>% group_by(userId) %>%
                 summarize(userId = first(userId),
                           user_average = mean(rating))
model <- test %>% inner_join(user_averages, by="userId")
rmse <- RMSE(model$rating, model$user_average)
print(rmse)
```

```
## [1] 0.9787829
```

There was a slight improvement of the RMSE with the value now being 0.9787829.

8.4 Movie average (movie effect)

The third model was to do a similar process as the second model but now just compare the movie averages instead of the user averages. This could be known as the *movie effect* by (Irizarry, 2021) but will be called *movie_averages* by the author of this paper. Only information about the movie is used to determine this model.

```
movie_averages <- train %>% group_by(movieId) %>%
                  summarize(movieId = first(movieId),
                            movie_average = mean(rating))
model <- test %>% inner_join(movie_averages, by="movieId")
rmse <- RMSE(model$rating, model$movie_average)
print(rmse)
```

```
## [1] 0.9437782
```

There is again a slight improvement of the RMSE with the value now being 0.9437782.

8.5 Average of movie effect and user effect

The fourth model was to take the numerical average of movie average and the user average. This could be known as averaging the movie effect and the user effect.

```
model <- test %>% inner_join(user_averages, by="userId") %>%
                            inner_join(movie_averages, by="movieId") %>%
                            mutate(prediction = (movie_average + user_average)/2)
```

```
rmse <- RMSE(model$rating, model$prediction)
print(rmse)
```

```
## [1] 0.9131466
```

This gives us yet another slight improvement of the RMSE of 0.9131466.

8.6 Remove overall average from movie and user averages

On close inspection of the previous model, however, one sees that the overall average of the ratings is, in effect, included twice: once in the user average (or user effect) and once in the movie average (or movie effect). This model removes the influence of the overall average from the individual user averages and from the individual movie averages. It is then added back in only once.

```
user_averages <- train %>% group_by(userId) %>%
                 summarize(userId = first(userId),
                 user_average = mean(rating - overall_average))
movie_averages <- train %>% group_by(movieId) %>%
                 summarize(movieId = first(movieId),
                 movie_average = mean(rating - overall_average))
model <- test %>% inner_join(user_averages, by="userId") %>%
                 inner_join(movie_averages, by="movieId") %>%
                 mutate(prediction = movie_average + user_average + overall_average)
rmse <- RMSE(model$rating, model$prediction)
print(rmse)
```

```
## [1] 0.8861918
```

With this, the RMSE has improved again to 0.8861918

8.7 Remove the user effect from the movie effect

This model removes the user average (user effect) from the movie average (movie effect) (along with removing the overall average) from both.

```
user_averages <- train %>% group_by(userId) %>%
  summarize(userId = first(userId),
            user_average = mean(rating - overall_average))
movie_averages <- train %>% group_by(movieId) %>%
  inner_join(user_averages, by="userId") %>%
  summarize(movieId = first(movieId),
            movie_average = mean(rating - overall_average - user_average ))
model <- test %>% inner_join(user_averages, by="userId") %>%
  inner_join(movie_averages, by="movieId") %>%
  mutate(prediction = movie_average + user_average + overall_average)
rmse <- RMSE(model$rating, model$prediction)
print(rmse)
```

```
## [1] 0.8827439
```

Again, the RMSE has shown an improvement to 0.8827439.

8.8 Remove the movie effect from the user effect

This model does the reverse of the previous model and removes the movie averages (movie effect) from the user averages (user effect).

```
movie_averages <- train %>% group_by(movieId) %>%
  summarize(movieId = first(movieId),
            movie_average = mean(rating - overall_average))
user_averages <- train %>% group_by(userId) %>%
  inner_join(movie_averages, by="movieId") %>%
  summarize(userId = first(userId),
            user_average = mean(rating - overall_average - movie_average ))
model <- test %>% inner_join(user_averages, by="userId") %>%
  inner_join(movie_averages, by="movieId") %>%
  mutate(prediction = movie_average + user_average + overall_average)
rmse <- RMSE(model$rating, model$prediction)
print(rmse)
```

```
## [1] 0.8663133
```

The RMSE has improved to 0.8663133.

8.9 Cater for movies and users which had very few reviews

The ratings of movies which only have one or two reviews can abnormally influence the predictions especially if those one or two reviews are extremely high. This is also true where users have given only a few reviews. The user is still *unpredictable*.

To cater for such situations, instead of calculating the simple mean of the movie rating (minus the overall average rating) as in

```
movie_averages <- train %>% group_by(movieId) %>%
                  summarize(movieId = first(movieId),
                  movie_average = mean(rating - overall_average))
```

a tuneable parameter can be used to assist in calculating this *movie_average* as in

```
movie_averages <- train %>% group_by(movieId) %>%
                  summarize(movieId = first(movieId), movie_review_count = n(),
                    movie_average =
                    sum(rating - overall_average)/(movie_review_count + qty_constant))
```

where *qty_constant* is some constant which needs yet to be determined. This *qty_constant* is refered to as *lamda* by (Irizarry, 2021). Note that if *qty_constant* is equal to zero, then the expression

```
movie_average = sum(rating - overall_average) / (movie_review_count + qty_constant)
```

is, in fact, the mean or average as used in the previous models.

The optimal *qty_constant* can be determined iteratively by trying a sequence of values and looking for the smallest RMSE. Figure 16 illustrates this phenomenon.

```
qty_constants <- seq(2, 8, 0.1)
rmses <- sapply(qty_constants, function(qty_constant) {
      movie_averages <- train %>% group_by(movieId) %>%
                summarize(movieId = first(movieId), movie_review_count = n(),
                movie_average = sum(rating - overall_average)/
                                (movie_review_count + qty_constant))
      user_averages <- train %>% group_by(userId) %>%
            inner_join(movie_averages, by="movieId") %>%
            summarize(userId = first(userId), user_review_count = n(),
                user_average = sum(rating - overall_average - movie_average )/
                (user_review_count + qty_constant))
```

```
      model <- test %>% inner_join(user_averages, by="userId") %>%
              inner_join(movie_averages, by="movieId") %>%
             mutate(prediction = movie_average + user_average + overall_average)
      rmse <- RMSE(model$rating, model$prediction)
})
df <- data.frame(qty_constants, rmses)
ggplot(aes(qty_constants, rmses), data=df) +
  geom_point() +
  ylab("RMSE") +
  xlab("Qty Cutoff Constant (lamda)") +
  my_theme()
```

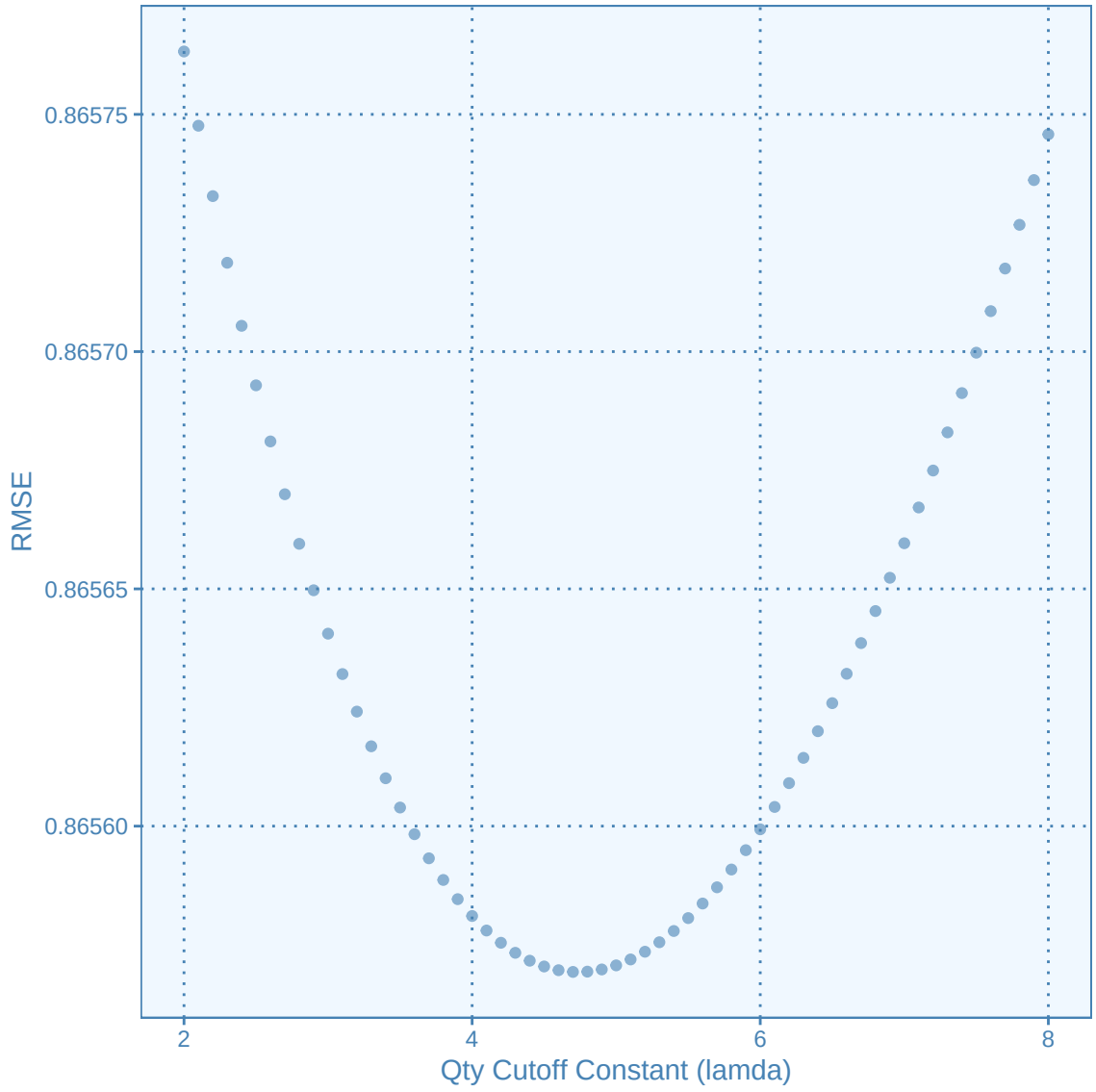

Figure 16: Finding optimal quantity constant

The minimum RMSE is 0.8655693 which occurs with *qty_constant* 4.7. This *qty_constant* value describes a

turning point where the average ratings of movies which have less than *qty_constant* number of reviews are less predictive than the average ratings of movies which have more than *qty_constant* number of reviews. This means that our model is *less sure* that the average ratings are actually predictive of the rating when we have less than *qty_constant* reviews. The same can be said about the number of reviews submitted by an individual user or reviewer. Irizarry (2021) calls this feature *regularization*.

That value can be put into the actual model and tested properly with the test data:

```
qty_constant <- qty_constants[which.min(rmses)]
movie_averages <- train %>% group_by(movieId) %>%
  summarize(movieId = first(movieId), movie_review_count = n(),
            movie_average = sum(rating - overall_average)/
              (movie_review_count + qty_constant))
user_averages <- train %>% group_by(userId) %>%
    inner_join(movie_averages, by="movieId") %>%
    summarize(userId = first(userId), user_review_count = n(),
              user_average = sum(rating - overall_average - movie_average)/
          (user_review_count + qty_constant))

model <- test %>% inner_join(user_averages, by="userId") %>%
  inner_join(movie_averages, by="movieId") %>%
  mutate(prediction = movie_average + user_average + overall_average)
rmse <- RMSE(model$rating, model$prediction)
print(rmse)
```

```
## [1] 0.8655693
```

This gives an RMSE of 0.8655693.

8.10 Summary of models and RMSEs

A summary of the RMSEs for all eight models which were still using the *train* and *test* datasets can be found in Table 1.

Table 1: Results using train and test datasets

Model	RMSE
Overall Average (Naive Approach)	1.0602987
User Averages (User Effect)	0.9787829
Movie Averages (Movie Effect)	0.9437782
Average of Averages (Average Effect)	0.9131466
Include Overall Averages Only Once	0.8861918
Remove User Average from Movie Average	0.8827439
Remove Movie Average from User Average	0.8663133
Compensate for Users and Movies with few reviews	0.8655693

8.11 Rebuild model with all of *edx* dataset

At this point, the model is operating at an acceptable level. Now the entire *edx* dataset (excluding the *validation* set) will be used to rebuild this model using the *qty_constant* value which was iteratively determined.

```
movie_averages <- edx %>% group_by(movieId) %>%
  summarize(movieId = first(movieId), movie_review_count = n(),
            movie_average = sum(rating - overall_average)/
```

```
              (movie_review_count + qty_constant))
user_averages <- edx %>% group_by(userId) %>%
    inner_join(movie_averages, by="movieId") %>%
    summarize(userId = first(userId), user_review_count = n(),
              user_average = sum(rating - overall_average - movie_average )/
            (user_review_count + qty_constant))
```

8.12 Results with *validation* dataset

That *validation* dataset is processed using this model. Note that the *validation* dataset does not need to be reshaped in anyway. After the investigations described in Section 7, it became apparent that none of the new fields created by the *reshape* function added any substantial value to the data.

```
load(validation_rda)
```

```
model <- validation %>% inner_join(user_averages, by="userId") %>%
  inner_join(movie_averages, by="movieId") %>%
  mutate(prediction = movie_average + user_average + overall_average)
rmse <- RMSE(model$rating, model$prediction)
print(rmse)
```

```
## [1] 0.8648208
```

The RMSE is 0.8648208.

9 Conclusion

This paper describes the first of two Capstone projects for the *Professional Certificate in Data Science* offered by Harvardx on the Edx learning platform.

The project is to predict ratings that a reviewer would give to a movie given a handful of different pieces of information about the movie and about the reviewer. A dataset was provided by the learning platform with instructions on how to break up the data into a training set and a validation set.

The author's code predicted the ratings with an RMSE (root mean squared error) value of 0.8648208 which is less than the cutoff value of 0.86490 to earn 25 points.

Caveats: The model developed by this research depends on information about existing movies and existing reviewers. The model is dependent on previous actions of the reviewer and previous actions of all reviewers for partiular movies. That means that this model will not be effective when a brand new movie is added to the *validation* set or when a brand new reviewer is added to the *validation* set

Acknowledgements

The author would like to thank Professor Irizarry and his team for creating a compelling and effective course. The exercises drew a person into the problem and enticed them to investigate the data. Thank you.

References

Gartner Glossary, 2021, Accessed https://www.gartner.com/en/information-technology/glossary/predictive-behavior-analysis

Irizarry, R.A., 2021, Introduction to Data Science.

Knuth, D.E., 1984. Literate programming. The Computer Journal, 27(2), pp.97-111.

Ljubuncic, I., 2015. Problem-solving in High Performance Computing: A Situational Awareness Approach with Linux. Morgan Kaufmann.

Love, R., 2013. Linux system programming: talking directly to the kernel and C library. O'Reilly Media, Inc.

Plous, S., 1993. The psychology of judgment and decision making. Mcgraw-Hill Book Company.

Solomon, M., Russell-Bennett, R. and Previte, J., 2012. Consumer behaviour. Pearson Higher Education AU.