HavardX PH125.9x Data Science Capstone MovieLens Project Report

Ravi Jha

12/01/2019

Contents

1. Introduction

1.1 Overview

This report is prepared to fulfill the completion requirement of the **HavardX PH125.9x Data Science Capstone** course. In this project, a movie recommendation system is created using the provided MovieLens dataset. A recommendation system is a subclass of an information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Recommendation systems are utilized heavily by leading companies such as Netflix and Amazon in a variety of areas including movies, music, news, books, search queries, social tags, and products in general.

1.2 Motivation

Data science has become an indispensable part of human life in today's society, as it solves real-life problems faced by every one of us. A strong recommendation system was of such importance that in 2006, Netflix offered a million-dollar prize to anyone who could improve the effectiveness of its recommendation system by 10%.

1.3 Project Goal

The goal of this project is to train a machine learning algorithm that predicts movie ratings from 0.5 to 5 stars, using the inputs of a provided edx dataset to predict movie ratings in a validation set.

Root Mean Square Error or *RMSE* is one of the widely used measures of the differences between values predicted by a model and the values observed. RMSE is a measure of accuracy, to compare forecasting errors of different models for a particular dataset, a lower RMSE is better than a higher one. The effect of each error on RMSE is proportional to the size of the squared error; thus larger errors have a disproportionately large effect on RMSE. Consequently, RMSE is sensitive to outliers.

RMSE with an expected value of **0.8649** or lower returned by testing the algorithm on the validation set is used as a benchmark to evaluate the model's performance. Following is the function that computes the RMSE for vectors of ratings and their corresponding predictors:

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

1.4 Key steps performed

Following are the key steps performed in this project:

- Importing the dataset and setup
- Spliting datasets
- Data wrangling
- Exploratory data analysis and visualization
- Creating and tuning predective models
- Evaluating models based on validation dataset
- Reporting results

2. Data and setup

The first step in the whole process is to import and setup data including splitting it in appropriate sub-datasets.

2.1 MovieLens DataSet

Let's start with the provided dataset, as per course instruction, the *10M version of MovieLens dataset* collected by GroupLens Research is used. This data set can be found and downloaded here:

- MovieLens 10M dataset: https://grouplens.org/datasets/movielens/10m/

2.2 Data Import and Initial Setup

To assist with our data import and initial setup, several packages from *CRAN* is utilized and loaded. These will be automatically downloaded and installed during code execution.

```
##########################
# to import and setup data
##########################

# Install requested packages if not found
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(dplyr))
  install.packages("dplyr")
if(!require(tidyr))
  install.packages("tidyr")


# Load libraries
library(tidyverse)
library(caret)
library(hexbin)
library(lubridate)
library(knitr)
library(kableExtra)

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

datafile <- "MovieLens.RData"

# Check if datafile is already downloaded
if(!file.exists("MovieLens.RData"))
{
  temp <- tempfile()
  fileURL <- "http://files.grouplens.org/datasets/movielens/ml-10m.zip"

  destfile_ratings <- "./ml-10M100K/ratings.dat"
  destfile_movies <- "./ml-10M100K/movies.dat"
```

```r
  # Execute only if destination data files - ratings.dat and movies.dat
  # are not found on the current directory folder
  if(!file.exists(destfile_ratings) && !file.exists(destfile_movies))
    {
      # Download file from url to temp file
      download.file(fileURL, temp)
      print("Downloading file to a temporary file")

      # Files to be extracted from the zip
      unzip_files= c("ml-10M100K/ratings.dat", "ml-10M100K/movies.dat")

      # Unzip files - ratings.data and movies.dat
      unzip(temp, files=unzip_files)
      print("Unzipping temporary file to destination files")

      # Remove temp file
      unlink(temp)
  }


  # create dataset object from .dat files in split into defined columns
  ratings <- fread(text = gsub("::", "\t", readLines(destfile_ratings)),
              col.names = c("userId", "movieId", "rating", "timestamp"))

  movies <- str_split_fixed(readLines(destfile_movies), "\\::", 3)
            colnames(movies) <- c("movieId", "title", "genres")

  # convert columns using mutate function
  movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[movieId],
            title = as.character(title), genres = as.character(genres))

  # join the both datasets into a new dataset by movieId
  movielens <- left_join(ratings, movies, by = "movieId")

  # Save movielens object to datafile
  save(movielens, file = datafile)

  # Remove unused objects from the memory
  rm(ratings, movies, temp)

  } else {
      # Load the datafile if it already exists
      load(datafile)
}
```

2.3 Spliting dataset: Training and Validation Sets

As per the project guidelines, the dataset will first be split into *edx* and *validation* set (10%), and the edx set will then be further split into a *training* and *test* set with the test set being 10%. The algorithm is developed using the edx set (train_set & test_set). For a final test of the algorithm, we predict movie ratings in the validation set as if they were unknown. The splitting scheme is represented as follows:

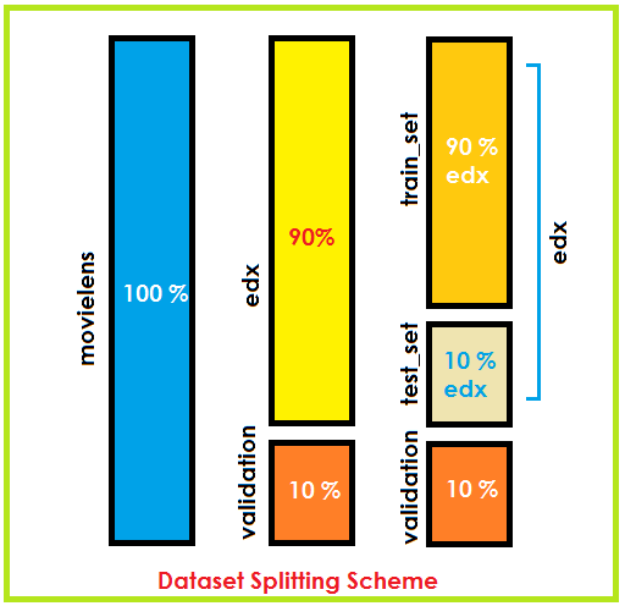


Figure 1: Dataset Spliting

The following code is used to generate the required datasets.

```
##############################################################################
# to split datasets into edx, validation, train_set, and test_set
##############################################################################

set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

# Partition the data set into edx and validation dataset.
# The Validation set will be 10% of MovieLens data
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)

# Remove unused objects from the memory
rm(test_index, movielens, removed)

# Now, further split the edx into train_set and test_set
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating,
                                  times = 1, p = 0.1, list = FALSE)
```

```r
train_set <- edx[-test_index,]
temp <- edx[test_index,]

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

# Add rows removed from test_set back into train_set
removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

# Remove unused objects from the memory
rm(removed, temp, test_index)
```

3. Data Wrangling

The next step is to understand the data through a brief data review and then do data wrangling as per the necessity.

3.1 Initial exploration

Before proceeding further with analysis, examining the data is important. The first step in it is to understand the format and the contents by looking at the few rows.

```
# Display first few rows to understand the data structure
head(train_set, 5)%>%

# to format table with theme
kable("latex", booktabs = T) %>%
kable_styling(latex_options = c("striped", "hover", "condensed", "scale_down"))%>%
row_spec(0, bold = T)
```

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

Unique Movies, Users, and Genres The dataset contains ~ 10,700 unique Movies, ~70,000 Users, and ~800 concatanated Genres.

```
# To display unique values for movies, users, and genres columns
train_set %>%
summarise(uniq_movies = n_distinct(movieId),
          uniq_users = n_distinct(userId),
          uniq_genres = n_distinct(genres))%>%

# to format table with theme
kable("latex", booktabs = T) %>%
kable_styling(latex_options = c("striped", "hover", "condensed"))%>%
row_spec(0, bold = T)
```

uniq_movies	uniq_users	uniq_genres
10677	69878	797

```
# Summary of the train_set dataset
summary(train_set)%>%

kable("latex", booktabs = T) %>%
kable_styling(latex_options = c("striped", "hover", "condensed", "scale_down"))%>%
row_spec(0, bold = T)
```

userId	movieId	rating	timestamp	title	genres
Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:8100065	Length:8100065
1st Qu.:18127	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character
Median :35732	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
Mean :35870	Mean : 4120	Mean :3.512	Mean :1.033e+09	NA	NA
3rd Qu.:53607	3rd Qu.: 3624	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

```
# to calculate ratings Mean mu
mu <- mean(train_set$rating)
```

Summary and Average Rating Mean Rating (mu): 3.5124556

Ratings There are 10 different rating scores, lowest is 0.5 and highest is 5.0.

```
# to list different rating scores
uniq_ratings <- unique(train_set$rating)
uniq_ratings <- sort(uniq_ratings)
uniq_ratings
```

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

Missing Data There are no missing values in the dataset as there are *0* columns with at least one NA.

```
  # To find NA values in the dataset
  anyNA(train_set)
```

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

Factorizing Variables In the dataset, columns `userId` and `movieId` contains numeric data and treated as numeric/integer by R. These columns can be converted to factor as this could be used as categorical data; however, in the current scenario it makes more sense to leave it as numeric to keep later computation easier e.g. summarize() function and other numeric calculations can be done easily.

	userId	movieId
Class	integer	numeric

Label encoding There are no columns with content as ordinal labels, so label encoding is not required in the dataset.

3.2 Tidying data

Based on the initial understanding further data analysis is required to identify data cleaning necessities. Following are the steps to tidy the data:

Extracting Movie Release Year The `title` column includes movie title and year the movie was released. This can be extracted as `year_released` in a separate column using the following code:

```
# Extract year_released with brackets
year_released <- str_extract(train_set$title, "\\(\\d{4}\\)$")
# Remove brackets
year_released <- str_remove_all(year_released, "[()]")
```

```
# Save year as numeric
train_set$year_released <- as.numeric(year_released)

# to display first few rows with newly added columns
head(train_set, 5)%>%

kable("latex", booktabs = T) %>%
kable_styling(latex_options = c("striped", "hover", "condensed", "scale_down")) %>%
row_spec(0, angle = 45, bold = T)
```

	userId	movieId	rating	timestamp	title	genres	year_released
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance	1992
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1995
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	1994
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1994
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	1994

Date Rated and Year Rated - Converting 'timestamp' to date Next, the `timestamp` column contains a UNIX timestamp (the number of seconds since January 1, 1970), which can be converted to easy to understand date and extracted as a `date_rated` separate column.

Further, `year_rated` can be extracted into a separate column to aid the analysis, using the following code:

```
# Convert timestamp to date_rated and extrat year as year_rated columns
train_set <- train_set %>% mutate(date_rated = as_datetime(timestamp),
                      year_rated = year(as_datetime(timestamp)))

# create a temporary dataset and remove timestamps column for displaying
train_set_dr <- subset(train_set, select = -c(timestamp) )

# to re-check the first few lines of the dataset
head(train_set_dr, 5)%>%

# to display as a formatted table with the theme
kable("latex", booktabs = T) %>%
kable_styling(latex_options = c("striped", "hover", "condensed", "scale_down")) %>%
row_spec(0, angle = 45, bold = T) %>%
footnote(general = "Column 'timestamp' is removed to accomodate
                    table width on the page.")
```

userId	movieId	rating	title	genres	year_released	date_rated	year_rated
1	122	5	Boomerang (1992)	Comedy|Romance	1992	1996-08-02 11:24:06	1996
1	292	5	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1995	1996-08-02 10:57:01	1996
1	316	5	Stargate (1994)	Action|Adventure|Sci-Fi	1994	1996-08-02 10:56:32	1996
1	329	5	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1994	1996-08-02 10:56:32	1996
1	355	5	Flintstones, The (1994)	Children|Comedy|Fantasy	1994	1996-08-02 11:14:34	1996

Note:
makecell[l]Column 'timestamp' is removed to accomodate table width on the page.

Movie Current Age and Age at the time of Review Other columns such as `movie_age` (the current age of the movie) and `age_at_review` (age of the movie when it was rated) can be helpful for the analysis. Following is the code to add it in the dataset:

```
# create movie_age by substracing current year to the year movie was released
# create age_at_review by substrating year the movie was rated and it was released
train_set <- train_set %>% mutate(movie_age = year(Sys.Date())-year_released,
                      age_at_review = year_rated - year_released)

# create a temporary dataset from training dataset
train_set_ma <- train_set

# Removing columns timestamp and date_rated from temporary dataset train_set_ma
train_set_ma <- subset(train_set_ma, select = -c(timestamp, date_rated) )

# to display first few lines of the dataset after changes
head(train_set_ma, 5)%>%

kable("latex", booktabs = T) %>%
kable_styling(latex_options = c("striped", "hover", "condensed", "scale_down"))  %>%
row_spec(0, angle = 45, bold = T) %>%
# to add foot note to the table
footnote(general = "Columns 'timestamp' and 'date_rated'
                    are removed to accomodate table width on the page.")
```

userId	movieId	rating	title	genres	year_released	year_rated	movie_age	age_at_review
1	122	5	Boomerang (1992)	Comedy|Romance	1992	1996	27	4
1	292	5	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1995	1996	24	1
1	316	5	Stargate (1994)	Action|Adventure|Sci-Fi	1994	1996	25	2
1	329	5	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1994	1996	25	2
1	355	5	Flintstones, The (1994)	Children|Comedy|Fantasy	1994	1996	25	2

Note:
makecell[l]Columns 'timestamp' and 'date_rated' are removed to accomodate table width on the page.

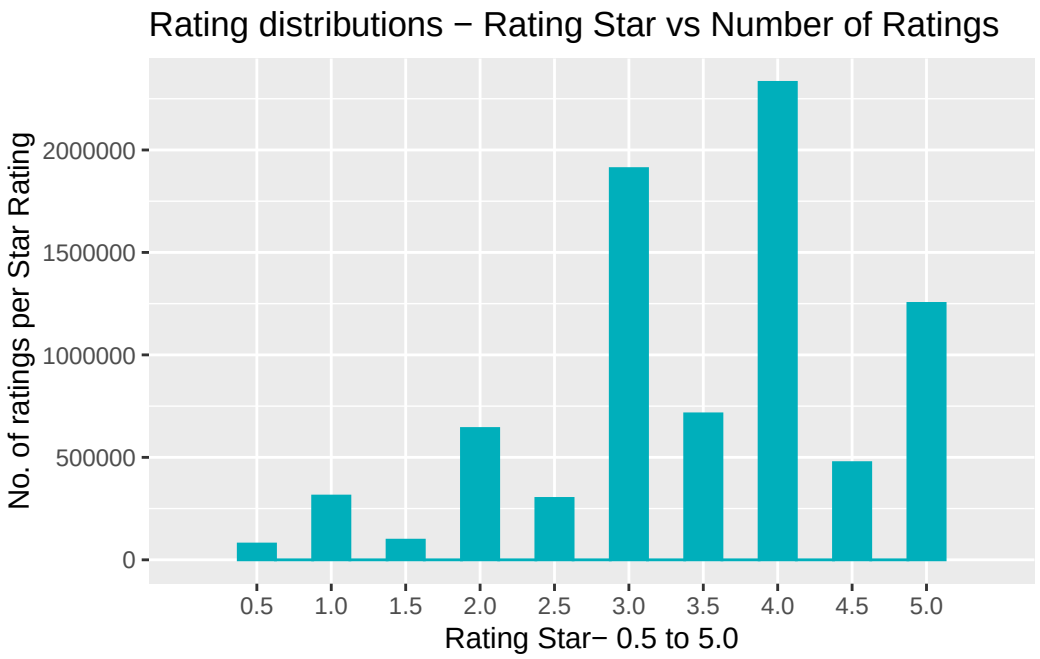
4. Exploratory Data Analysis and Visualization

The next step is to perform exploratory data analysis on the tidy data by utilizing visualization techniques.

Ratings Distribution The chart shows that users have a preference to rate movies rather higher than lower. 4.0 is the most common rating, followed by 3.0 and 5.0. 0.5 is the least preferable rating and there is no movie with a 0.0 rating.

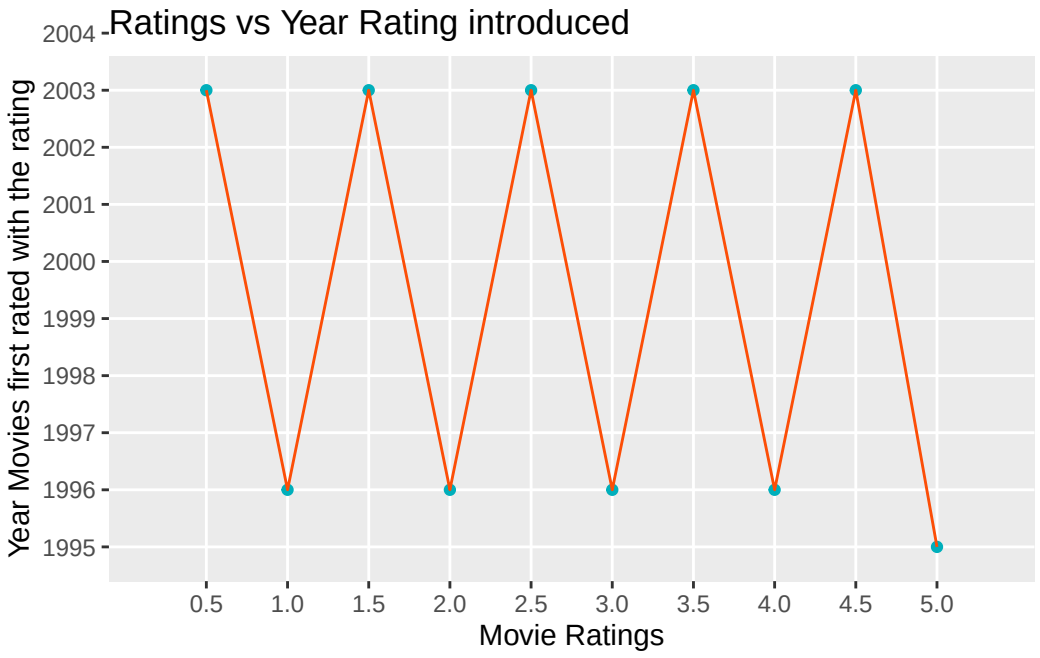
The difference in the median and mean noted in the *Summary and Average Ratings* section above also supports the negative skewness slope of the distribution towards higher ratings.

```
# to plot rating against ratings count
train_set %>%
# map rating variable to aesthetics of ggplot function
ggplot(aes(rating)) +

# to create a histogram
geom_histogram(fill = "#00AFBB", binwidth = 0.25, color = "#00AFBB") +

# scale axis to custom range and intervels
scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +

# to provide a title, x, and y axis labels
xlab("Rating Star- 0.5 to 5.0") +
ylab("No. of ratings per Star Rating") +
ggtitle("Rating distributions - Rating Star vs Number of Ratings ")
```



Half-star Ratings There are fewer *half-star ratings* than *whole-star ratings*. This may be due to the factors involving users' rating preference and also the fact that the half-star ratings are introduced in 2003. The below graph shows in which year the movies with the ratings (0.5- 5) were first got rated as shown below.

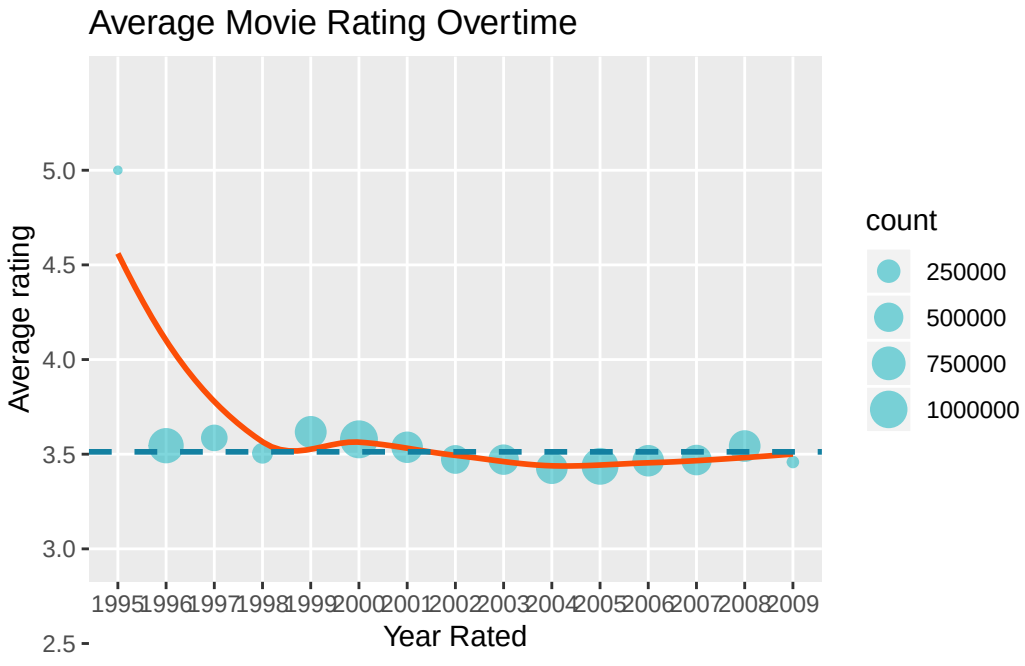
```
# plot to determine when the half-star raintgs started
train_set %>%

group_by(rating) %>%

summarise(min_year = min(year_rated), count =n()) %>%
# to map year_rated and mean rating variables to aesthetics of ggplot function
ggplot(aes(x = rating, y = min_year)) +

# to plot scattered data using geom_point
geom_point(alpha = 2, col = "#00AFBB") +
# to draw line to connect data points
geom_line(colour = "#FC4E07", size = .5) +

# scale axis to custom range and intervels
scale_x_discrete(limits = c(seq(0.5,5,0.5)))  +
scale_y_discrete(limits = c(seq(1995,2009,1)))  +

# to provide a title, x, and y axis labels
labs(title = "Ratings vs Year Rating introduced",
     x = "Movie Ratings", y = "Year Movies first rated with the rating")
```



This does not appear to affect the rating distribution. However, this may need more careful analysis later while modeling.

Average Movie Rating variation over years Let's check how the average movie rating has changed over time. As shown in the graph below the majority of averages are within the vicinity of the overall average rating, with higher average ratings between 1996 to 2001.

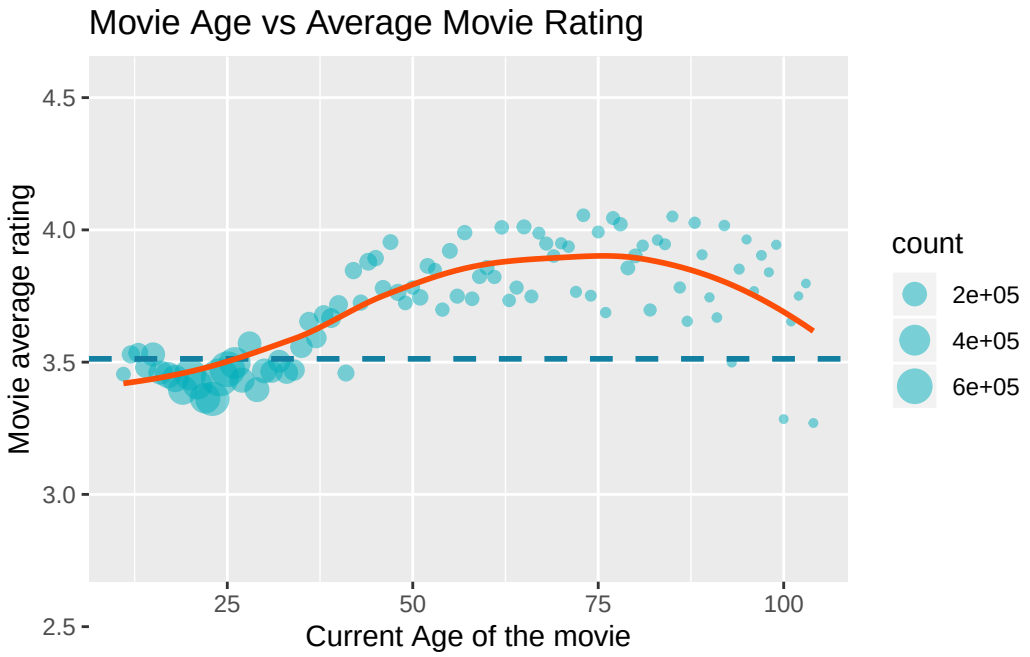
```
train_set %>%

# to plot graph for mean rating for individual year movies were rated
group_by(year_rated) %>%
summarise(avg_rating = mean(rating), count = n()) %>%

# to map year_rated and mean rating variables to aesthetics of ggplot function
ggplot(aes(x = year_rated, y = avg_rating)) +

# to plot scattered data using geom_point
geom_point(aes(size = count), alpha = 0.5, col = "#00AFBB") +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to draw a horizontal line incepting y-axis at the mean rating
geom_hline(yintercept = mu, linetype = "dashed", color="#1380A1", size=1) +

# scale axis to custom range and intervels
scale_x_discrete(limits = c(seq(1995,2009,1)))  +
scale_y_discrete(limits = c(seq(0.5,5,0.5)))  +

# to provide a title, x, and y axis labels
xlab("Year Rated") +
ylab("Average rating") +
labs(title = "Average Movie Rating Overtime")
```



The overall average rating is shown by a dashed horizontal line in the graph

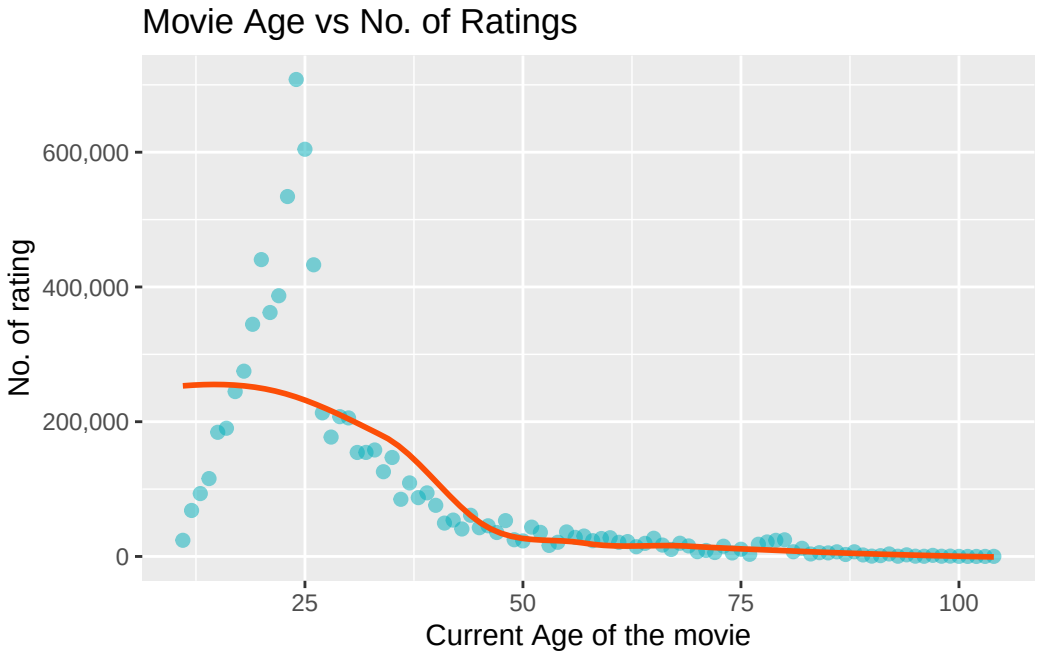
Movie current age vs its average rating The below chart depicts the relationship of the movie's current age to the movies' average rating. Rating varies with the movie age. It is evident that the average rating is higher for the older movies up to 90 years old, then the rating droped. Also, the average rating is highest for movies with age around 75 years; however, movies with the current age around 25 received have higher rating count.

```
train_set %>%

# to plot graph for mean rating for individual year movies were rated
group_by(movie_age) %>%
summarize(count = n(), avg_rating_by_age = mean(rating)) %>%

# to map movie_age and mean rating by movie age variables to aesthetics of ggplot function
ggplot(aes(movie_age, avg_rating_by_age)) +

# to plot scattered data using geom_point
geom_point(aes(size = count), alpha = 0.5, col = "#00AFBB") +

# to draw a horizontal line incepting y-axis at the mean rating
geom_hline(yintercept = mu, linetype = "dashed", color="#1380A1", size=1) +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# scale axis to custom range and intervels
scale_y_discrete(limits = c(seq(0.5,5,0.5)))  +

# to provide a title, x, and y axis labels
xlab("Current Age of the movie") +
ylab("Movie average rating") +
ggtitle("Movie Age vs Average Movie Rating")
```



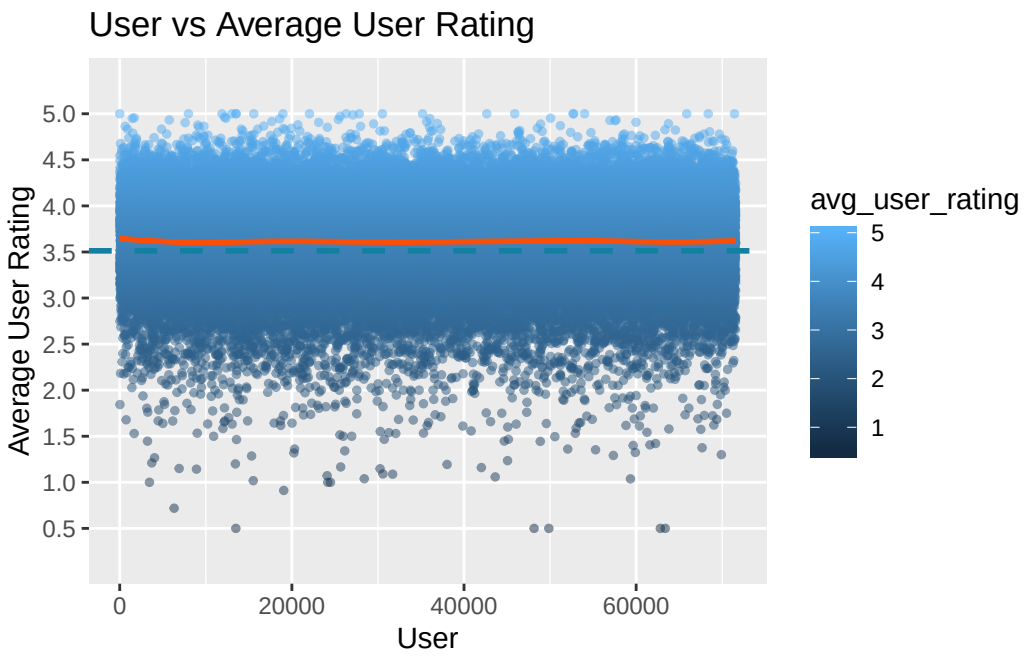
Movie current age vs No. of ratings received It can be clearly seen from the below graph that the movies with age between 15-25 years have received the highest number of ratings. It makes sense as it correlates with when the rating system was introduces. On the other hand, the movies with more than 50 years of age the number of ratings has declined significantly.

```
# to plot Movie current age vs No. of ratings received
train_set %>%

# to plot graph for mean rating for individual year movies were rated
group_by(movie_age) %>%
summarize(count = n()) %>%

# to map movie_age and count variables to aesthetics of ggplot function
ggplot(aes(movie_age, count)) +

# to plot scattered data using geom_point
geom_point(alpha = 0.5, col = "#00AFBB", size = 2) +

# scale axis to show non-scientific values (without exponential)
scale_y_continuous(labels = scales::comma) +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("Current Age of the movie") +
ylab("No. of rating") +
ggtitle("Movie Age vs No. of Ratings")
```



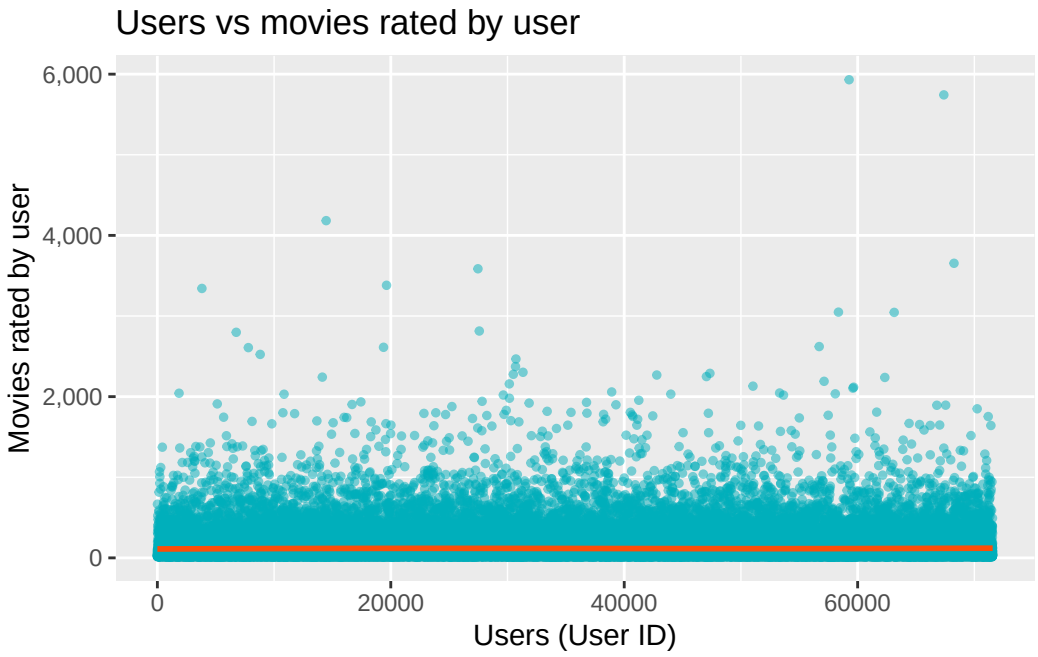
Users vs average rating given by the user From the chart below it can be observed that the majority of the average ratings by users are consistent between ~2.5 and ~4.5.

```
# userId vs average movie rating
train_set %>%

# to plot graph for mean rating for individual user
group_by(userId) %>%
summarize(avg_user_rating = mean(rating)) %>%

# to map movie_age and mean rating by movie age variables to aesthetics of ggplot function
ggplot(aes(userId, avg_user_rating, color = avg_user_rating)) +

# to plot scattered data using geom_point
geom_point(alpha = 0.5, size = 1) +

# to draw a horizontal line incepting y-axis at the mean rating
geom_hline(yintercept = mu, linetype = "dashed", color="#1380A1", size=1) +

# scale axis to custom range and intervels
scale_y_discrete(limits = c(seq(0.5,5,0.5)))  +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("User") +
ylab("Average User Rating") +
ggtitle("User vs Average User Rating")
```



Users vs No. of movies rated by user Furthermore, the majority of users rated a maximum of ~500 movies, and there are very limited users who rated more than 3,000 movies.

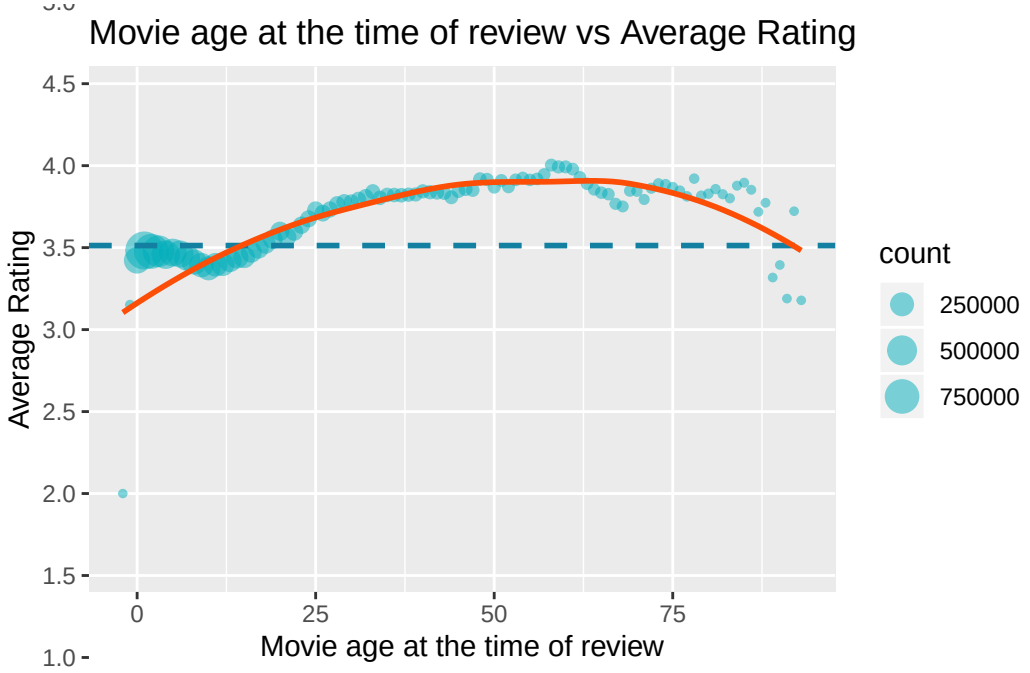
```
# to plot a graph userId against average movie rating
train_set %>%

# to plot graph for count of movies rated for individual userId
group_by(userId) %>%
summarize(count = n()) %>%

# to map userId and count variables to aesthetics of ggplot function
ggplot(aes(userId, count)) +

# to plot scattered data using geom_point
geom_point(alpha = 0.5, col = "#00AFBB", size = 1) +

# scale axis to show non-scientific values (without exponential)
scale_y_continuous(labels = scales::comma) +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("Users (User ID)") +
ylab("Movies rated by user") +
ggtitle("Users vs movies rated by user")
```



Movie Age at the time of review vs Average Rating The below graph shows how does a rating change related to the movie's age at the time of the review (denoted by `age_at_review`). It can be seen that the movies with the age between 40 and 70 year's of age at the time of review received higher average rating. Also, the average rating for movies increased intiailly before dropping after 70 year of age.

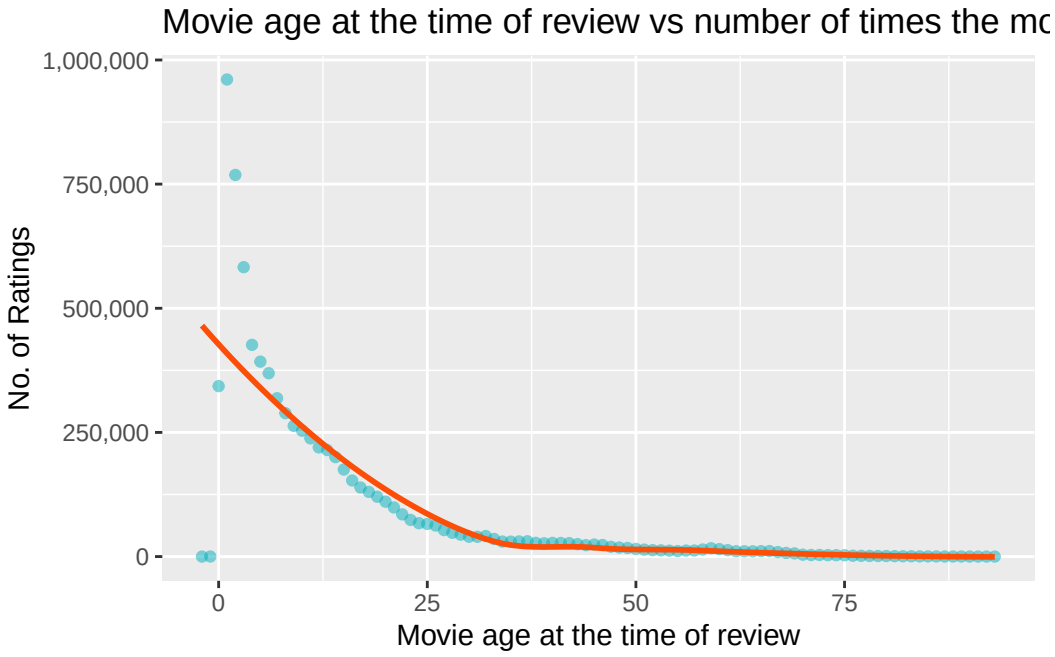
Furthermore, the data points that show reviews before the release date are either bad data points or were reviewed by critics before the movie was actually premiered.

```
# to plot a graph showing movie's age at the time of the revoew and its average rating
train_set %>%

# to plot mean rating graph for movies with age at review in years
group_by(age_at_review) %>%
summarize(count = n(), avg_by_rating_age = mean(rating)) %>%

# to map age at review and mean rating by age at review +
# variables to aesthetics of ggplot function
ggplot(aes(age_at_review, avg_by_rating_age)) +

# to plot scattered data using geom_point
geom_point(aes(size = count), alpha = 0.5, col = "#00AFBB") +

# to draw a horizontal line incepting y-axis at the mean rating
geom_hline(yintercept = mu, linetype = "dashed", color="#1380A1", size=1) +

#scale axis to custom range and intervels
scale_y_discrete(limits = c(seq(0.5,5,0.5)))  +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("Movie age at the time of review") +
ylab("Average Rating") +
ggtitle("Movie age at the time of review vs Average Rating")
```


Movie age at the time of review vs Average Rating
Average Rating
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
25
50
75
Movie age at the time of review
count
250000
500000
750000

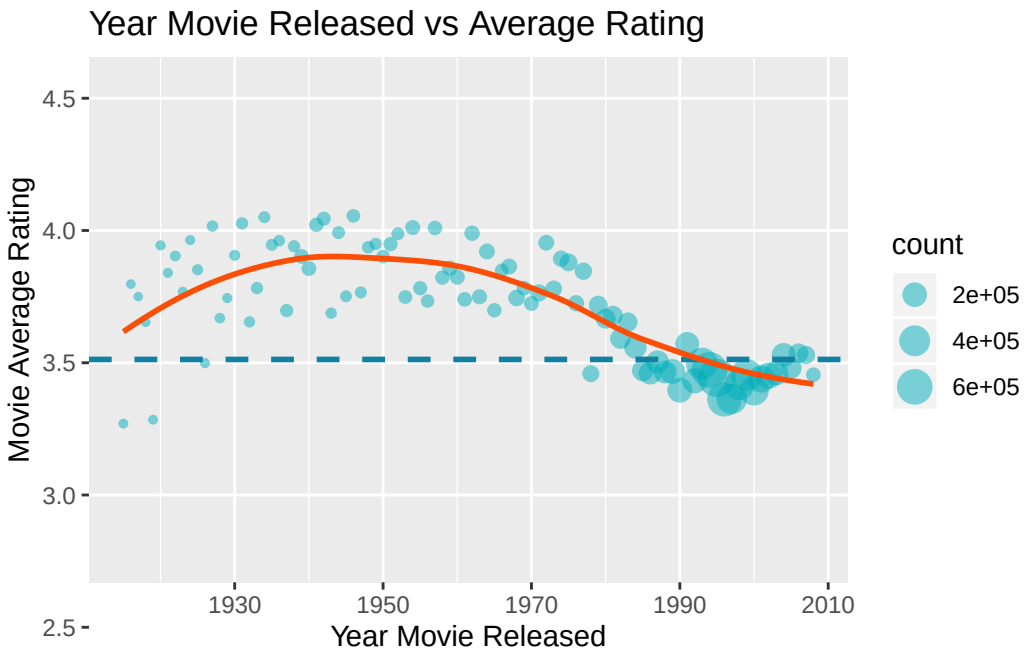
Movie Age at the time of review vs No. of Ratings It is evident from the graph that the review count is higher for movies recently premiered; as the age at the time of review increases the number of ratings plummeted significantly. There also few movies which were rated before its release, it could be because of erroneous data or critics gave the ratings before the release.

```
# to plot age at review vs rating count
train_set %>%

# to plot graph for rating count for movies' age at the time of the review
group_by(age_at_review) %>%
summarize(count = n()) %>%

# to map age at review and movie count variables to aesthetics of ggplot function
ggplot(aes(age_at_review, count)) +

# to plot scattered data using geom_point
geom_point(alpha = 0.5, col = "#00AFBB") +

# scale axis to show non-scientific values (without exponential)
scale_y_continuous(labels = scales::comma) +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("Movie age at the time of review") +
ylab("No. of Ratings") +
ggtitle("Movie age at the time of review vs number of times the movie rated")
```



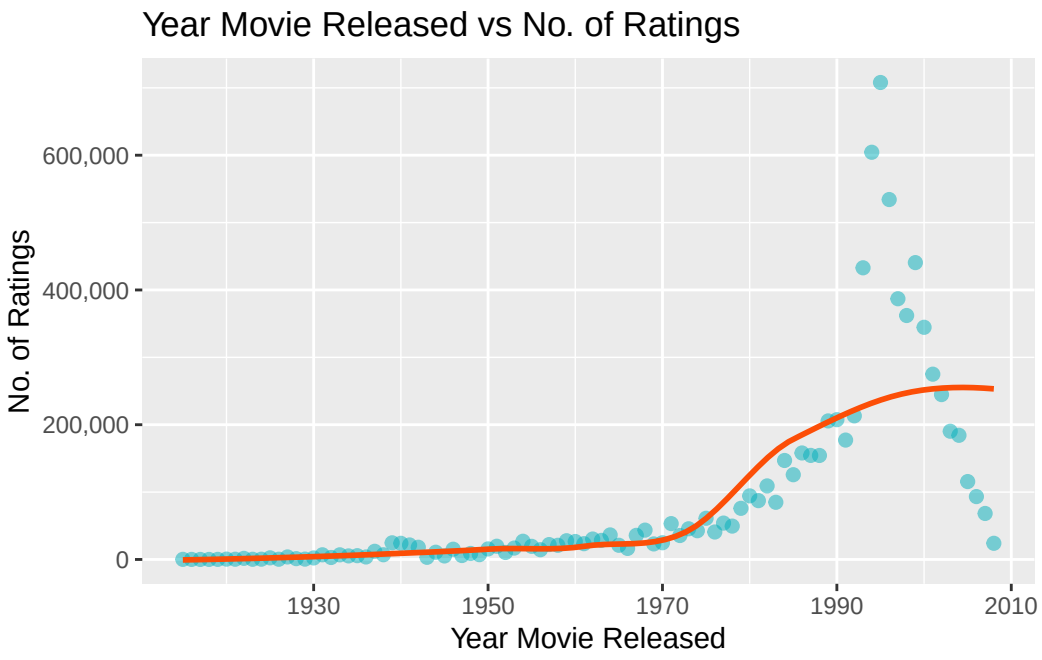
Movie Year Released vs its average rating The next observation shows that the movies premiered between 1930 and 1970 have received a higher average rating; whereas for most of the movies released after 1985 the average rating received is less than the overall rating average. It also shows that the user preferred to give higher ratings before 1985.

```
# to plot year released against average ratings
train_set %>%

# to plot graph for mean rating for individual year movies were released
group_by(year_released) %>%
summarize(count = n(), avg_by_year_rel = mean(rating)) %>%

# to map year released and mean rating by the year released +
# variables to aesthetics of ggplot function
ggplot(aes(year_released, avg_by_year_rel)) +

# to plot scattered data using geom_point
geom_point(aes(size = count), alpha = 0.5, col = "#00AFBB") +

# to draw a horizontal line incepting y-axis at the mean rating
geom_hline(yintercept = mu, linetype = "dashed", color="#1380A1", size=1) +

#scale axis to custom range and intervels
scale_y_discrete(limits = c(seq(0.5,5,0.5)))  +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("Year Movie Released") +
ylab("Movie Average Rating") +
ggtitle("Year Movie Released vs Average Rating")
```



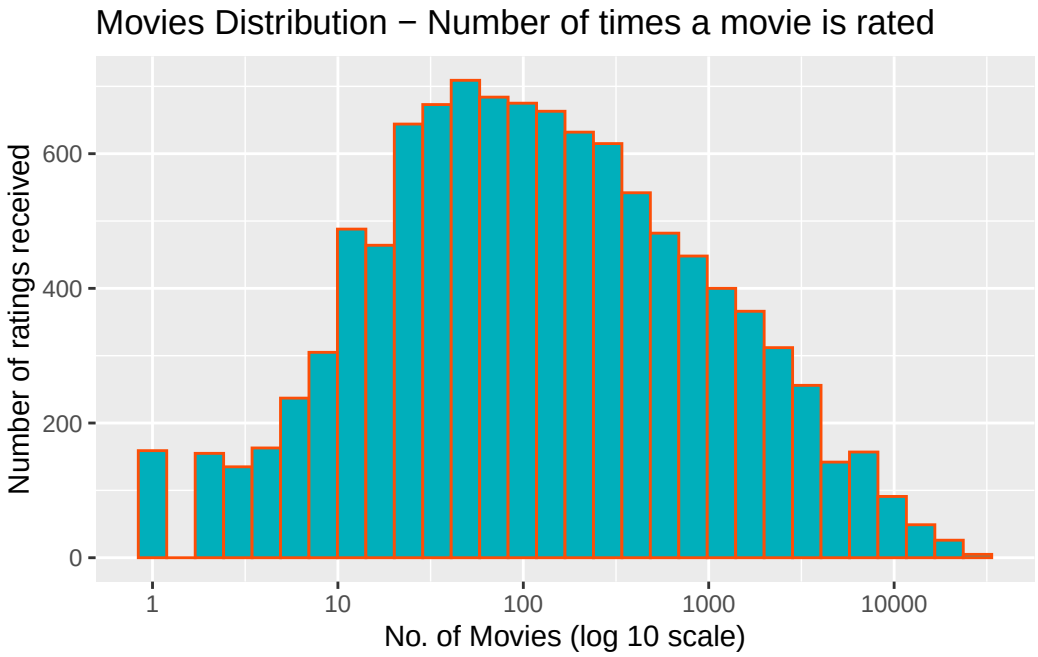
Movie Year Released vs number of ratings The other observation is that the movies released between 1990-2000 received an exceptionally higher number of ratings. In general, more recent movies get more ratings and the movies released before 1930 get few ratings. The year 2009 shows a steep decline probably attributed to the fact that this dataset does not contain the movie rating for the full year 2009.

```
# to plot year_released against count of the ratings
train_set %>%

# to plot graph for rating count against individual year movies were released
group_by(year_released) %>%
summarize(count = n()) %>%

# to map year released and movie count variables to aesthetics of ggplot function
ggplot(aes(year_released, count)) +

# to plot scattered data using geom_point
geom_point(alpha = 0.5, col = "#00AFBB", size = 2) +

# scale axis to show non-scientific values (without exponential)
scale_y_continuous(labels = scales::comma) +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("Year Movie Released") +
ylab("No. of Ratings") +
ggtitle("Year Movie Released vs No. of Ratings")
```



Movies Distribution It can be noticed from the below distribution that some movies are rated much often than other movies.

```
# to plot movie and the count of ratings recieved by a specific movie
train_set %>%

# to map movies count variables to aesthetics of ggplot function
count(movieId) %>%
ggplot(aes(n)) +

# to plot graph as histogram with bin size of 30
geom_histogram(fill = "#00AFBB", bins = 30, color = "#FC4E07") +

# scale axis to log 10
scale_x_log10() +

# to provide a title, x, and y axis labels
xlab("No. of Movies (log 10 scale)") +
ylab("Number of ratings received") +
ggtitle("Movies Distribution - Number of times a movie is rated")
```



Moreover, there are movies which were rated only once and with extreme ratings (rating less than 2 or more than 4), following table shows the 10 such movies:

```
# filter out obscure outliers movies which was rated once and with
# extreme ratings (rating less than 2 or more than 4)

# create a temporary dataset from training set
train_set_working <- train_set %>%

# filter outliers movie with extreme ratings
filter(rating < 2 | rating > 4) %>%

# calculate count of filtered uniqueu movies
```

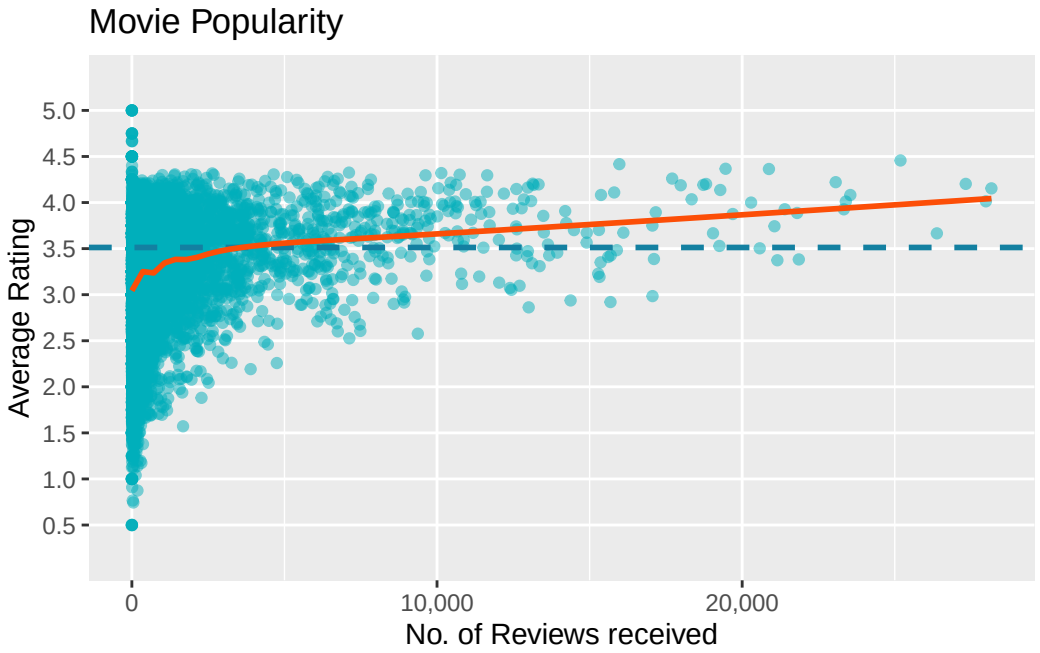
```
    group_by(title) %>%
    summarise(rating = mean(rating), count = n()) %>%

    # filter out movies rated only once
    filter(count == 1)

    # to display first 10 rows
    slice(train_set_working, 1:10)%>%

    # to apply theme to the table
    kable("latex", booktabs = T) %>%
    kable_styling(latex_options = c("striped", "hover", "condensed"))%>%
    row_spec(0, bold = T)
```

title	rating	count
100 Rifles (1969)	0.5	1
10th & Wolf (2006)	1.5	1
20,000 Leagues Under the Sea (1916)	4.5	1
24 7: Twenty Four Seven (1997)	5.0	1
29th Street (1991)	5.0	1
30 Years to Life (2001)	0.5	1
36 fillette (1988)	0.5	1
36 Quai des OrfÃ¨vres (Department 36) (2004)	1.0	1
5 Fingers (1952)	0.5	1
55 Days at Peking (1963)	4.5	1

In fact, there are **550** movies (in train_set) with extreme ratings that have been rated only once. These all seem like obscure movies and are noisy estimates that should not be trusted. Therefore, **regularisation** and a **penalty term** will be applied to the models. Regularization and panelty terms will be discussed later in this report during modeling.

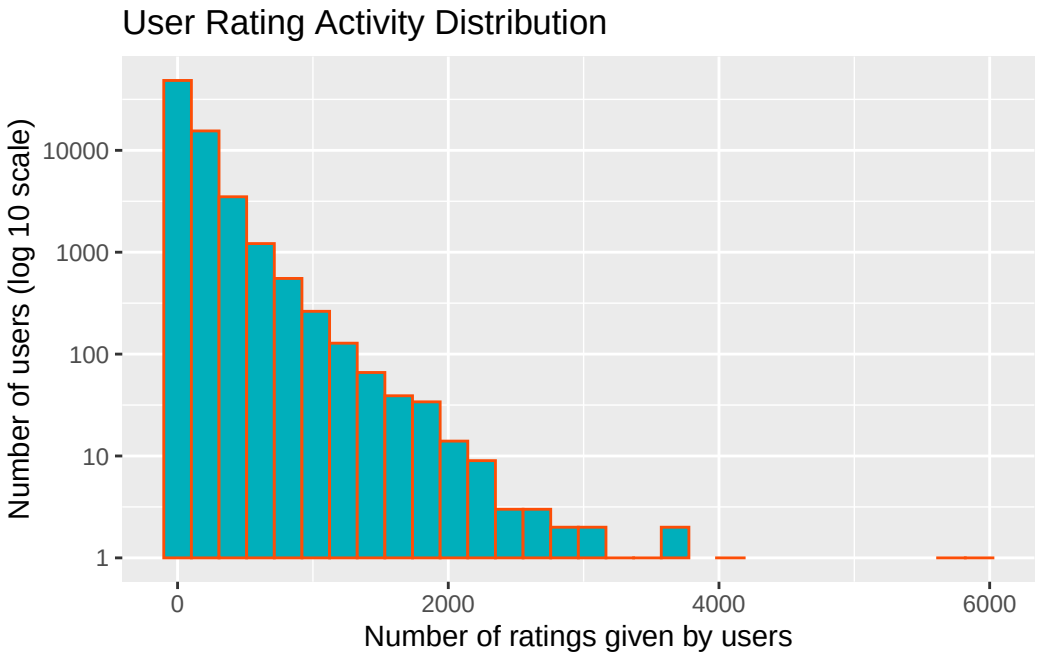
Movie Popularity Lets, now check the relationship between the number of reviews a movie received and the average rating of that movie. It can be noticed from the graph below that the popular movies received a higher rating average as well as the number of ratings. On the other hand, a general trend is of less rating average and the number of ratings received for less popular movies.

```
# to plot review count and average ratings received by a specific movie
train_set %>%

# to plot graph for mean rating and count for individual movies
group_by(movieId) %>%
summarise(count = n(), avg_rating = mean(rating)) %>%

# to map movie count and mean rating by movie variables to aesthetics of ggplot function
ggplot(aes(x = count, y = avg_rating)) +

# to plot scattered data using geom_point
geom_point(alpha = 0.5, col = "#00AFBB") +

# to draw a horizontal line incepting y-axis at the mean rating
geom_hline(yintercept = mu, linetype = "dashed", color="#1380A1", size=1) +

#scale axis to custom range and intervels
scale_x_continuous(labels = scales::comma)  +
scale_y_discrete(limits = c(seq(0.5,5,0.5)))  +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("No. of Reviews received") +
ylab("Average Rating") +
labs(title = "Movie Popularity")
```



Users Distribution It can be noticed by looking at the below user distribution graph that some users more actively rated movies than others.

```
# to plot number of ratings given by a specific user
train_set %>%
count(userId) %>%

# to map users count variables to aesthetics of ggplot function
ggplot(aes(n)) +

# to plot graph as histogram with bin size of 30
geom_histogram(fill = "#00AFBB", bins = 30, color = "#FC4E07") +

# scale axis to log 10
scale_y_log10() +

# to provide a title, x, and y axis labels
xlab("Number of ratings given by users") +
ylab("Number of users (log 10 scale)") +
ggtitle("User Rating Activity Distribution")
```



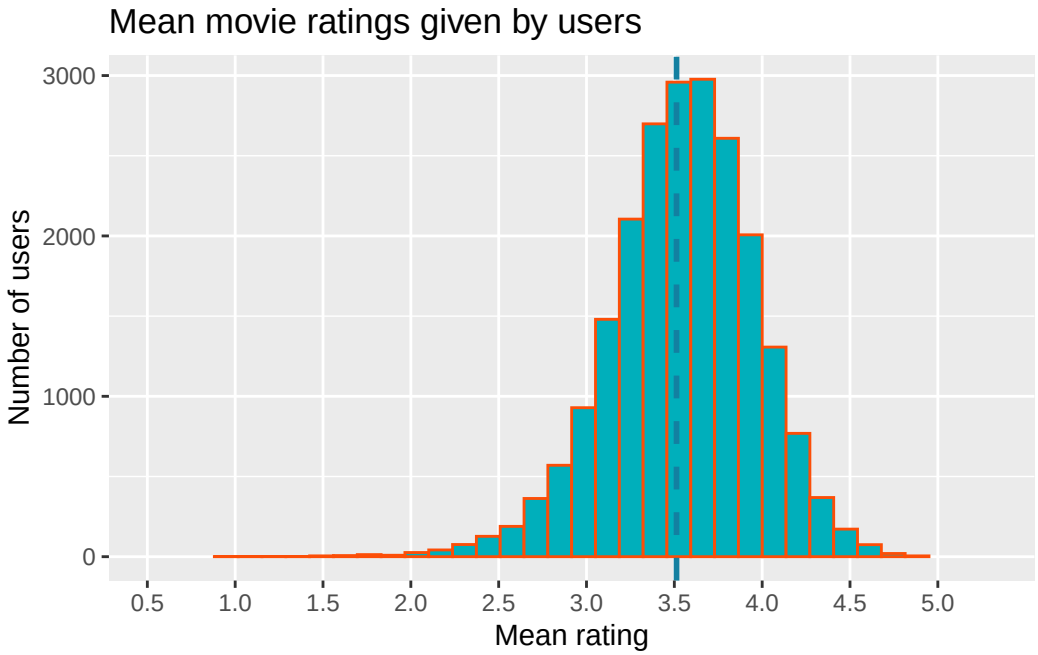
Furthermore, the below table depicts number of movies rated by most active, least active, and average user.

	Most active user rated	Least active user rated	Average user rated
Number of ratings	5931	9	116

Moreover, it can be noticed from the below graph that some users preferred to give higher ratings while others tend to give lower than average ratings. To eliminate the *effect of outlier* points only the *users who rated 100 or more ratings* are included.

```
# "Mean movie ratings given by users
train_set %>%
```

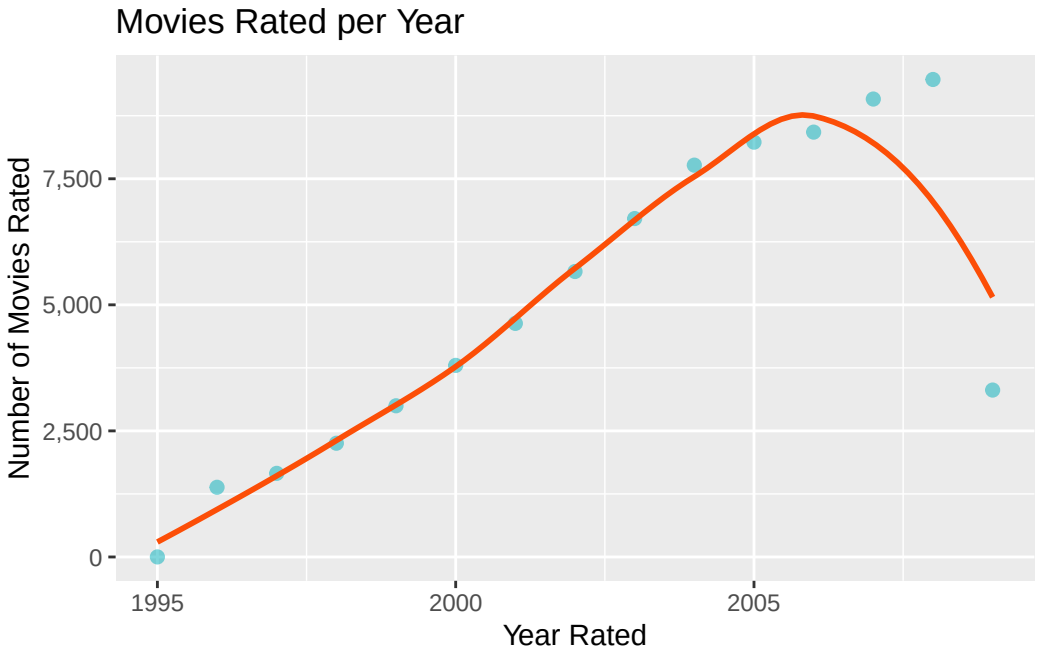
```
# to plot graph for bias for individual user who rated atleast 100 movies
group_by(userId) %>%
filter(n() >= 100) %>%
summarize(b_u = mean(rating)) %>%

# to map user bias variable to aesthetics of ggplot function
ggplot(aes(b_u)) +

# to plot histogram
geom_histogram(fill = "#00AFBB", bins = 30, color = "#FC4E07") +

# to draw a vertical line incepting x-axis at the mean rating
geom_vline(xintercept = mu, linetype = "dashed", color="#1380A1", size=1) +

# to provide a title, x, and y axis labels
xlab("Mean rating") +
ylab("Number of users") +
ggtitle("Mean movie ratings given by users") +
scale_x_discrete(limits = c(seq(0.5,5,0.5)))
```



The overall average rating is shown by a dashed horizontal line in the graph

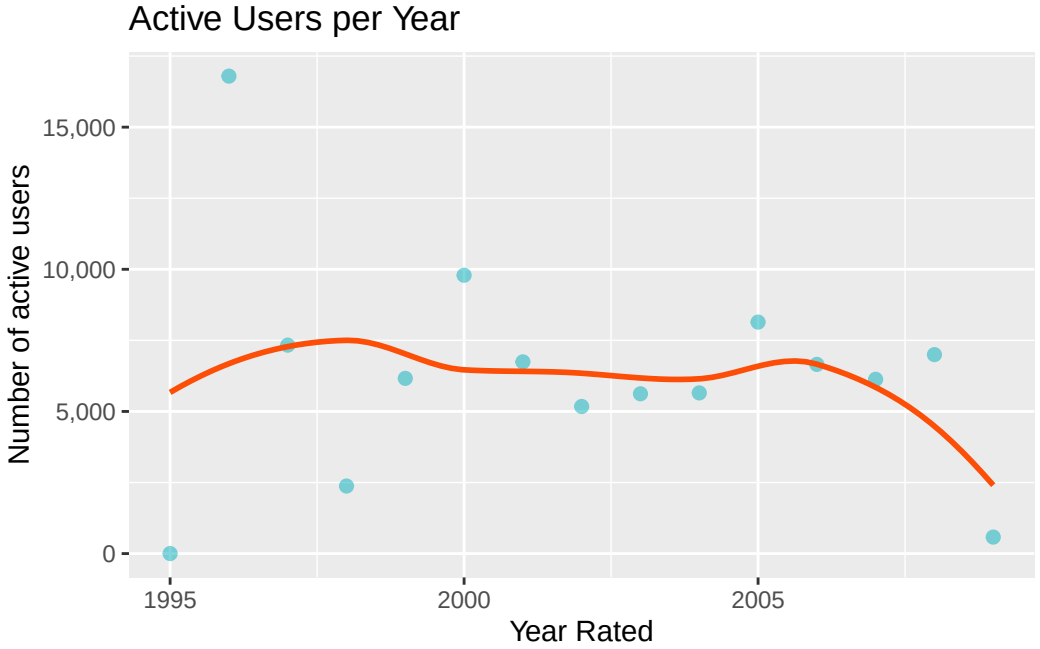
Number of movies rated per year Next, we can see that in 1995 the least number of movies got rated; whereas 2008 has highest number of movies were rated. It can also be noticed, the number of movies rated each year constantly increased from 1995 to 2008 before dropping in 2009. The sharp decline in 2009 may be due to the fact that the data is not collected for the whole year, and as it is the case with 1995 as well.

```
# to plot number of movies rated per year
train_set %>%

select(year_rated, movieId) %>%
group_by(year_rated) %>%
summarise(count = n_distinct(movieId)) %>%

# to map year rated and number of movies variables to aesthetics of ggplot function
ggplot(aes(x = year_rated, y = count)) +

# to plot scattered data using geom_point
geom_point(alpha = 0.5, col = "#00AFBB", size = 2) +

# scale axis to show non-scientific values (without exponential)
scale_y_continuous(labels = scales::comma) +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("Year Rated") +
ylab("Number of Movies Rated") +
ggtitle("Movies Rated per Year")
```



Number of active users per year As we saw in the previous graph, despite the most number of active users in the year 1996 fewer movies were rated. This implies that the movie rating system after getting introduced in 1995, in 1996 more users interested and rated movies; however, in the following years comparatively fewer users rated more movies.

```
# to plot number of active users per year
train_set %>%

# to select columns
select(year_rated, userId) %>%

# to plot specific users per year rated
group_by(year_rated) %>%
summarise(count = n_distinct(userId)) %>%

# to map year_rated and active users per year variables to aesthetics of ggplot function
ggplot(aes(x = year_rated, y = count)) +

# to plot scattered data using geom_point
geom_point(alpha = 0.5, col = "#00AFBB", size = 2) +

# scale axis to show non-scientific values (without exponential)
scale_y_continuous(labels = scales::comma) +

# to draw a smooth conditional mean line
geom_smooth(se = FALSE, colour = "#FC4E07", size = 1) +

# to provide a title, x, and y axis labels
xlab("Year Rated") +
ylab("Number of active users") +
ggtitle("Active Users per Year")
```



5. Modeling Approach

Now, with the analysis we are ready for modeling the prediction model, this section describes modeling approaches and insights gained before finalizing a target model.

5.1 Loss Function

As discussed earlier we will use *Root Mean Square Error*, or *RMSE* loss function to measure the performance of the models. Lower the RMSE better the performance of the model.

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

where N is sample size, y is the observed value, $\hat{y}$ is predicted value for u,i times and $\sum$ = summation

5.2 Model 1: Basic Rating Mean Model

Let's start with a very basic model based on the rating average. This simplest possible prediction model will predict the same rating for all movies without taking any other predictors into account. The difference in the predicted values is explained by random variation in the independent error variable.

The model can be represented as follows:

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

where μ is the mean which shows the true rating value for all movies. And, $\epsilon_{u,i}$ is the independent random error variable.

Now, lets now calculate RMSE for this model as follow:

```
      ################################
      # to find basic naive mean RMSE
      ################################
      rmse_mean <- RMSE(test_set$rating, mu)
      rmse_mean
```

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

This gives us our first model with RMSE to start with. Also, it can be noticed that this is equivalent to the standard deviation of rating distribution.

The results can be displayed in a table for easy comparison using the following code:

```
      # to display RMSE results in a tabular form
      rmse_results <- tibble(Model = "Rating Mean Naive Model",
                             Dataset = "test_set", RMSE = round(rmse_mean, digits = 5))
      rmse_results %>%

      # to apply theme to the table
      kable("latex", booktabs = T) %>%
      kable_styling(latex_options = c("striped", "hover", "condensed")) %>%

      # to group rows in various catagories
      pack_rows("Basic Prediction Model", 1, 1) %>%
      row_spec(0, bold = T) %>%
      row_spec(1:1, bold = T, color = "white", background = "#D7261E")
```

Model	Dataset	RMSE
Basic Prediction Model		
Rating Mean Naive Model	test_set	1.06005

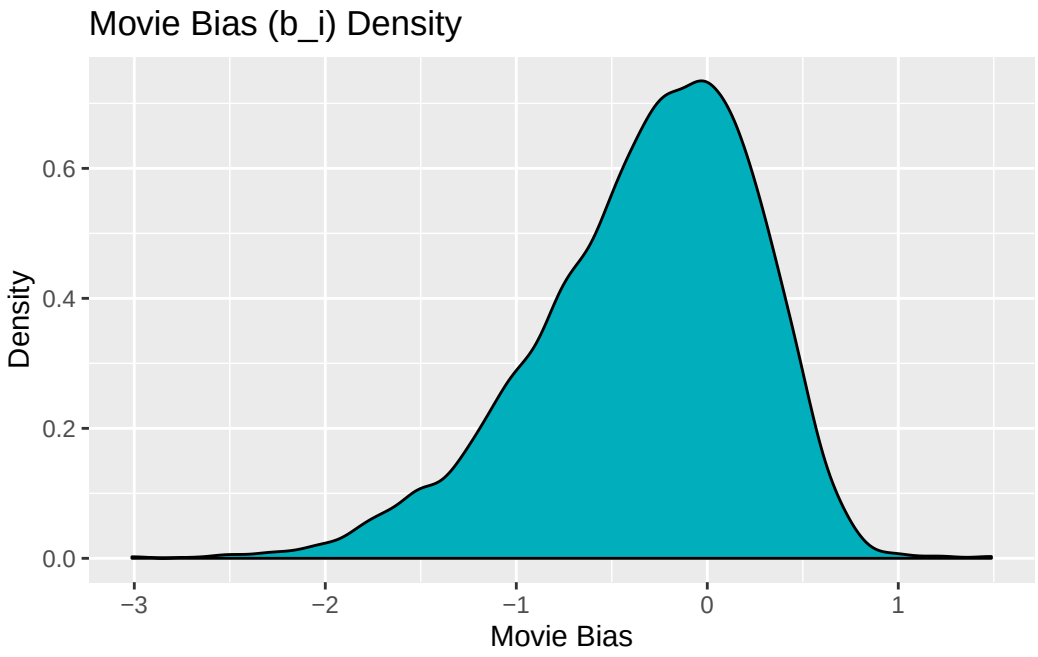
We can further refine this basic prediction model by removing outlier data points from the calculation; however, this may not yield the desired RMSE improvement. So we will move to the predictor based models by utilizing potential predicators insights gained during exploratory data analysis.

5.3 Single Predictor Model

To begin with a more complex model, we can start with a single potential predictor identified during the data exploration. Each potential predictors can be evaluated and compared with other models based on the RMSE generated.

5.3.1 Model 2: Movie Effect Model This model takes into account the effect of the movie considering that each movie is not rated equally and introduces a movie bias term to the *Basic Rating Mean Model*. The term is based on individual movie mean rating and its difference with the overall mean rating.

As it can be seen from the graph below, the left skewness implies that more movies have negative effects (rated less than the average).



The model can be denoted as follow:

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

where $Y_{u,i}$ is the predicted rating, $\epsilon_{u,i}$ is the independent error, μ the mean rating for all movies, and b_i is the bias for each movie i:

The least-squares lm() method is not being used as it is computation-intensive and slow on a large datasets, specifically computing bias for each movie. So we will use the average of the residuals.

Individual movie bias is calculated using the formula:

$$b_i = mean(rating_i - \mu)$$

Lets calcluate the RMSE and see how much it improved as compared to the last model using the following code:

```
      ##################################################
      # model taking into account the movie effects, b_i
      ##################################################

      # to find movie bias for all movies with unique movieId
      movie_bias <- train_set %>%
      group_by(movieId) %>%
      summarise(b_i = mean(rating - mu))

      # to calculate predicted ratings using mean + movie bias in test_set set for the movieId
      predicted_ratings <- mu + test_set %>%
      left_join(movie_bias, by='movieId') %>%
      pull(b_i)

      # to calculate RMSE using the RMSE function by passing predicted and true ratings
      rmse_movie <- RMSE(predicted_ratings, test_set$rating)
      rmse_movie
```

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

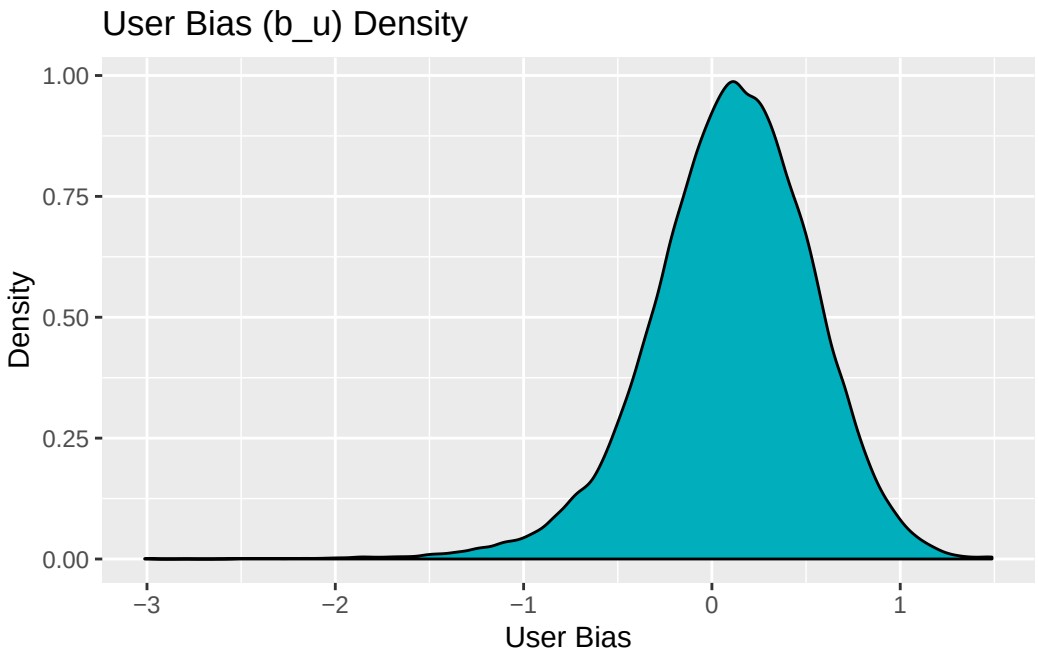
```
      # append results to the table
      rmse_results <- bind_rows(rmse_results,
                                tibble(Model="Movie Effect Model",
                                Dataset = "test_set", RMSE = round(rmse_movie, digits = 5)))
      rmse_results %>%
          kable("latex", booktabs = T) %>%
          kable_styling(latex_options = c("striped", "hover", "condensed")) %>%

          # to group rows in various catagories
          pack_rows("Basic Prediction Model", 1, 1) %>%
          pack_rows("Single Predictor Model", 2, 2) %>%
          row_spec(0, bold = T) %>%
          # to highlight the last row
          row_spec(2:2, bold = T, color = "white", background = "#D7261E")
```

Model	Dataset	RMSE
Basic Prediction Model		
Rating Mean Naive Model	test_set	1.06005
Single Predictor Model		
Movie Effect Model	**test_set**	**0.94296**

As it can be seen, the Movie Effect Model predicts the movie rating with bias b_i and mean μ and gives an improved prediction with a lower RMSE value.

5.3.2 Model 3: User Effect Model Furthermore, lets now check the User Effect Model. This model takes into account the effect of the user on rating movies considering that each user does not rate movies

equally, some cranky user may rate a good movie with a low rating and vice versa. In general, this model based on the assumption that the user will rate a movie with an average of ratings given to other movies. This introduces a user bias term denoted by b_u. The term is based on individual user mean rating and its difference with the overall mean rating.



The model can be denoted as follow:

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

where $Y_{u,i}$ is the predicted movie rating, $\epsilon_{u,i}$ is the independent error, μ the mean rating for all movies, and b_u is the bias for each user u.

Again, because of the bad performance of the least-squares lm() method, the average of the residuals will be utilized.

Individual user bias is calculated using the formula:

$$b_u = mean(rating_u - \mu)$$

Now lets calculate RMSE for this model with the help of below code:

```
#####################################################
# model taking into account the user effects, b_u
#####################################################

  # to find user bias for all users with unique userId
  user_bias <- train_set %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu))

# to calculate predicted ratings using mean + user bias in test_set for the movieId
predicted_ratings <- test_set %>%
  left_join(user_bias, by='userId') %>%
  mutate(pred = mu + b_u) %>%
```

```
    # pull /select predicted value from the joined dataaset
    pull(pred)

  # to calculate RMSE using the RMSE function by passing predicted and true ratings
  rmse_usr <- RMSE(predicted_ratings, test_set$rating)
  rmse_usr
```

```
## [1] 0.977709
```

```
  # append results to the table
  rmse_results <- bind_rows(rmse_results,
                            tibble(Model="User Effect Model",
                            Dataset = "test_set", RMSE = round(rmse_usr, digits = 5)))
  rmse_results %>%
      kable("latex", booktabs = T) %>%
      kable_styling(latex_options = c("striped", "hover", "condensed")) %>%

      # to group rows in various catagories
      pack_rows("Basic Prediction Model", 1, 1)  %>%
      pack_rows("Single Predictor Model", 2, 3) %>%
      row_spec(0, bold = T) %>%
      # to highlight the last row
      row_spec(3:3, bold = T, color = "white", background = "#D7261E")
```

Model	Dataset	RMSE
Basic Prediction Model		
Rating Mean Naive Model	test_set	1.06005
Single Predictor Model		
Movie Effect Model	test_set	0.94296
User Effect Model	**test_set**	**0.97771**

The User Effect Model doesn't improve prediction model and it has higher RMSE value as compared with Movie effet model.

5.3.3 Model 4: Year Released Effect Model Similarly, Year Released Effect Model can be developed, it takes into account the effect of the year movie was released on rating movies considering that each movie does not get rated equally. In general, this model based on the assumption that the movie will be rated with the average of ratings given to other movies on the particular release year. This introduces a year release (premiered) bias term denoted by b_p. The model can be denoted as follow:

$$Y_{p,i} = \mu + b_p + \epsilon_{p,i}$$

where $Y_{p,i}$ is the predicted movie rating, $\epsilon_{p,i}$ is the independent error, μ the mean rating for all movies, and b_p is the bias for each year released p.

Individual year released bias is calculated using the formula:

$$b_p = mean(rating_p - \mu)$$

Now lets calculate RMSE for this model with the help of below code:

```
  ##########################################################################
  # model taking into account the year released effects, b_p (bias)
```

```
      ######################################################################

      # to find bias for all movies released in a particular year
      YRel_avgs <- train_set %>%
        group_by(year_released) %>%
        summarise(b_p = mean(rating - mu))

        # to calculate predicted ratings using mean and year_released bias in test_set set
        # year_released in not available in the test_set dataset, +
        # so extrat it with the same logic used to extrat train_set column
        predicted_ratings <- test_set %>%
        mutate(year_released=as.numeric(str_remove_all(str_extract(title,
                                                    "\\(\\d{4}\\)$"), "[()]"))) %>%

        # perform a left join by year released mutate predicted value and
        # pull it to be passsed into predicted ratings dataset
        left_join(YRel_avgs, by='year_released') %>%
        mutate(pred = mu + b_p) %>%
        pull(pred)

      # to calculate RMSE using the RMSE function by passing predicted and true ratings
      rmse_YRel <- RMSE(predicted_ratings, test_set$rating)
      rmse_YRel
```

```
## [1] 1.049283
```

```
      # append results to the table
      rmse_results <- bind_rows(rmse_results,
                                tibble(Model="Year Released Effect Model",
                                Dataset = "test_set", RMSE = round(rmse_YRel, digits = 5)))
      rmse_results %>%
          kable("latex", booktabs = T) %>%
          kable_styling(latex_options = c("striped", "hover", "condensed")) %>%

          # to group rows in various catagories
          pack_rows("Basic Prediction Model", 1, 1) %>%
          pack_rows("Single Predictor Model", 2, 4) %>%
          row_spec(0, bold = T)  %>%
          # to highlight the last row
          row_spec(4:4, bold = T, color = "white", background = "#D7261E")
```

Model	Dataset	RMSE
Basic Prediction Model		
Rating Mean Naive Model	test_set	1.06005
Single Predictor Model		
Movie Effect Model	test_set	0.94296
User Effect Model	test_set	0.97771
Year Released Effect Model	test_set	1.04928

The performance is worse than the other two single predictor models. Let's move on to the next model.

5.3.4 Model 5: Year Rated Effect Model Moreover, Year Rated Effect Model takes into account the effect of the year movie was rated on rating movies. In general, this model based on the assumption that the movie will be rated with the average of ratings given to other movies on the particular rating year. This introduces a year rated bias term denoted by b_r. The model can be denoted as follow:

$$Y_{r,i} = \mu + b_r + \epsilon_{r,i}$$

where $Y_{r,i}$ is the predicted movie rating, $\epsilon_{r,i}$ is the independent error, μ the mean rating for all movies, and b_r is the bias for each year rated r.

Individual year rated bias is calculated using the formula:

$$b_r = mean(rating_r - \mu)$$

Now lets calculate RMSE for this model with the help of below code:

```
      ##############################################################################
      # model taking into account the year a movie was rated effects, b_r (bias)
      ##############################################################################

      # to find bias for all movies released in a particular year
      YRat_avgs <- train_set %>%
        group_by(year_rated) %>%
        summarise(b_r = mean(rating - mu))

        # to calculate predicted ratings using mean in test_set for the year rated
        # year_released in not available in the test_set dataset, +
        # so extrat it with the same logic used to extrat train_set column
        predicted_ratings <- test_set %>%
        mutate(year_rated = year(as_datetime(timestamp))) %>%

        # perform a left join by year released mutate predicted value and
        # pull it to be passsed into predicted ratings dataset
        left_join(YRat_avgs, by='year_rated') %>%
        mutate(pred = mu + b_r) %>%
        pull(pred)

      # to calculate RMSE using the RMSE function by passing predicted and true ratings
      rmse_YRat <- RMSE(predicted_ratings, test_set$rating)
      rmse_YRat
```

```
## [1] 1.058391
```

```
      # append results to the table
      rmse_results <- bind_rows(rmse_results,
                                tibble(Model="Year Rated Effect Model",
                                Dataset = "test_set", RMSE = round(rmse_YRat, digits = 5)))
      rmse_results %>%
          kable("latex", booktabs = T) %>%
          kable_styling(latex_options = c("striped", "hover", "condensed")) %>%

          # to group rows in various catagories
          pack_rows("Basic Prediction Model", 1, 1) %>%
          pack_rows("Single Predictor Model", 2, 5) %>%
          row_spec(0, bold = T)  %>%
```

```
        # to highlight the last row
        row_spec(5:5, bold = T, color = "white", background = "#D7261E")
```

Model	Dataset	RMSE
Basic Prediction Model		
Rating Mean Naive Model	test_set	1.06005
Single Predictor Model		
Movie Effect Model	test_set	0.94296
User Effect Model	test_set	0.97771
Year Released Effect Model	test_set	1.04928
Year Rated Effect Model	**test_set**	**1.05839**

This model seems further degrade the performance and almost close to the basic mean model.

Due to the memory limitation on the machine *Genre Effect Model* could not be succesful, as genre category splitting increased dataset by many folds. Hence not included in the report.

So, let's move to multiple predictor models.

5.4 Multiple Predictors Model

As we have noticed, the single predictor model has shown some improvements over the basic rating average model. The multiple predicators approach combines multiple potential predicators with lower RMSE.

5.4.1 Model 6: Movie and User Effects Model Let's start with the two predictors - movie effect b_i and user effect b_u.

The next step is to incorporate the combination of both Movie b_i, and User Effects b_u, into the model.

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the mean rating for all movies, b_i is the bias for each movie i, and b_u is the bias for each user u.

Please note, least-squares lm() method is not being used as it is computation-intensive and slow on a large datasets, specifically computing bias. So alternatively, b_u is computed as the average for each movie i, as follow

$$y_{u,i} - \mu - b_i$$

Now, lets calculate the RMSE for this model using the following code:

```
      ########################################
      ## the model taking Movie and User bias
      ########################################

      # to find bias for all movies released in a particular year
      user_bias <- train_set %>%
      left_join(movie_bias, by="movieId") %>%
      group_by(userId) %>%
      summarise(b_u = mean(rating - mu - b_i))

      # to calculate predicted ratings using mean, user bias and movie bias in test_set
      predicted_ratings <- test_set %>%
```

```
    # first perform a left join by movieId then by userId, mutate predicted value
    # pull it to be passsed into predicted ratings dataset
    left_join(movie_bias, by='movieId') %>%
    left_join(user_bias, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

    # to calculate RMSE using the RMSE function by passing predicted and true ratings
    rmse_Mov_Usr <- RMSE(predicted_ratings, test_set$rating)
    rmse_Mov_Usr
```

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

The RMSE result for the model can be appended to the results table for comparison as follows:

```
    # append results to the table
    rmse_results <- bind_rows(rmse_results,
                              tibble(Model="Movie & User Effect Model",
                              Dataset = "test_set", RMSE = round(rmse_Mov_Usr, digits = 5)))
    rmse_results %>%
        kable("latex", booktabs = T) %>%
        kable_styling(latex_options = c("striped", "hover", "condensed")) %>%

        # to group rows in various catagories
        pack_rows("Basic Prediction Model", 1, 1) %>%
        pack_rows("Single Predictor Model", 2, 5)  %>%
        pack_rows("Multipe Predictors Model", 6, 6) %>%
        row_spec(0, bold = T)  %>%
        # to highlight the last row
        row_spec(6:6, bold = T, color = "white", background = "#D7261E")
```

Model	Dataset	RMSE
Basic Prediction Model		
Rating Mean Naive Model	test_set	1.06005
Single Predictor Model		
Movie Effect Model	test_set	0.94296
User Effect Model	test_set	0.97771
Year Released Effect Model	test_set	1.04928
Year Rated Effect Model	test_set	1.05839
Multipe Predictors Model		
Movie & User Effect Model	**test_set**	**0.86468**

From the result, we can see an improvement in the RSME; however, it is still higher than target RMSE. Let's check next how this can further be improved using the *regularization*.

5.4.2 Other Multiple Predictors Models Moreover, there are multiple combinations of potential predictors (as shown below) that could be tried to calculate and compare RMSE; however, to keep the scope of this report limited to the goal of achieving the RMSE, other multiple predictors models are not discussed here.

Predictors: `movieId`, `userId`, `year_rated`, `year_released`, `age_at_review`, & `genre`

5.5 Regularized Model

Regularization allows penalizing large estimates that are formed using small sample sizes and skew the error metric. Regularization is a technique to constrain the total variability of the effect sizes. In general, it is used to avoid model overfitting by adding an additional penalty term in the error function.

5.5.1 Model 7: Regularized Movie and User bias Prediction Model Let's start with modifying the combined Movie and User Effects Model shown in the last section. The Regularized model uses a tuning parameter, λ, to minimize the effects of obscure movies and users on the prediction and in turn minimize the RMSE.

The least square model that we dicussed in the last section cannot limit variability of the movie and user effect sizes, so we will target to minimize the following penalized regression equation:

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

where N is sample size, b_i is movie bias, b_u is user bias, λ is tuning parameter, μ is mean

The first term is for least squares of user and movie effects and second is a penalty term which is controlled by λ (if bais gets large so the whole panelty, and negates the overall effect).

Moreover, the regularization parameter Lambda (λ) controls the tradeoff between fitting training data and model complexity by adjusting the weight of the penalty term.

- If your lambda value is too high, the model will be simple, but there is a risk of underfitting data. The model won't learn enough about the training data to make useful predictions.

- If lambda value is too low, the model will be more complex, and it runs the risk of overfitting the data. In general, the model will learn too much about the particularities of the training data, and won't be able to generalize to new data.

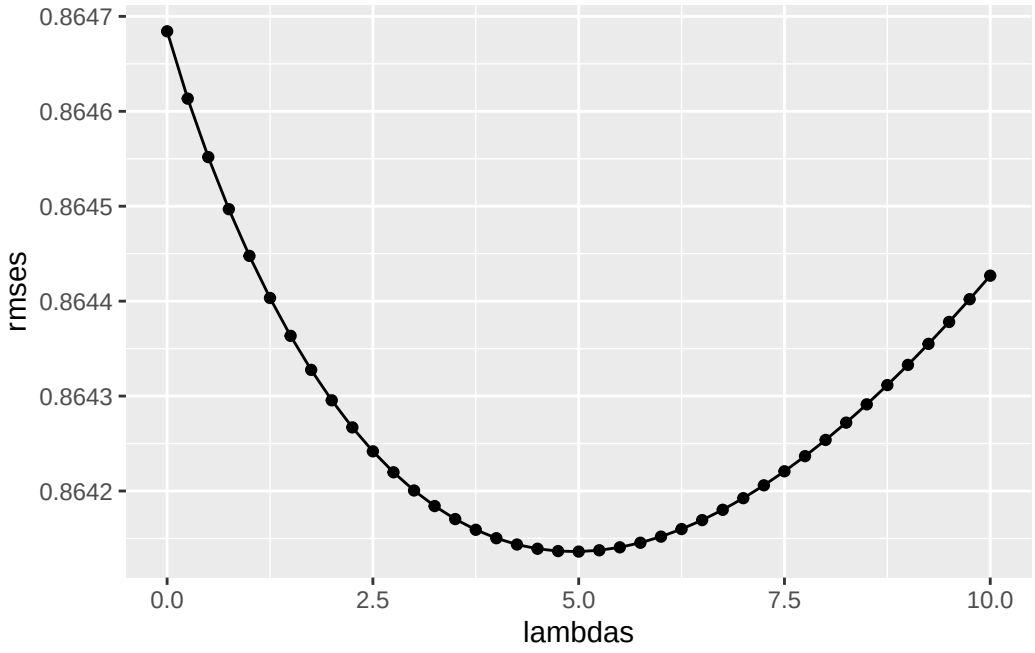
Let's calculate RMSEs for different values of lambdas:

```
##########################################################
# Predict via regularisation, movie and user effect model
##########################################################

# lambdas vetror contains values from 0 to 10 with an interverl of 0.25
lambdas <- seq(0, 10, 0.25)

# sapply apply various values of lambda from vector lambdas to the +
# function to calcuate all possible RMSE and stores in rmses
rmses <- sapply(lambdas, function(lambda){

# calcuate mean
mu <- mean(train_set$rating)

# calculate movie bias using lambda
movie_bias <- train_set %>%
group_by(movieId) %>%
summarise(b_i = sum(rating - mu)/(n()+lambda))

# calculate user bias using lambda
user_bias <- train_set %>%
left_join(movie_bias, by="movieId") %>%
group_by(userId) %>%
```

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

      # first left join test_set with movie bias by movieId and +
      # then join again with user bias using userId
      predicted_ratings <- test_set %>%
      left_join(movie_bias, by = "movieId") %>%
      left_join(user_bias, by = "userId") %>%
      mutate(pred = mu + b_i + b_u) %>%
      pull(pred)

      # returns the value of calculated RMSE on train_set set with the passed lambda argument
      return(RMSE(predicted_ratings, test_set$rating))

      })
```

Now, lets plot RMSE vs lambdas to select the optimal value of lambda which minimizes the RMSE:

```
      # plot a graph lambdas against generated rmses
      qplot(lambdas, rmses, geom=c("point", "line"))
```



Based on the plotted graph we can clearly see the value of RMSE is minimum for lambdas ~ 5, let's check the optimal lambda for which RMSE is minimum using the following code :

```
      # to find the minimum value of lambda from lambdas for which RMSE is minimum
      lambda <- lambdas[which.min(rmses)]
      lambda
```

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

The optimal lambda is: 5

Now, let's select the minimum RMSE for this model (for which the lambda is optimal), using the following code:

```
rmse_reg = min(rmses)

# append results to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="Regularized Movie & User Effect Model",
                          Dataset = paste0("test_set   |   lambda :", toString(lambda)),
                          RMSE = round(rmse_reg, digits = 5)))
rmse_results %>%
  kable("latex", booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hover", "condensed")) %>%

  # to group rows in various catagories
  pack_rows("Basic Prediction Model", 1, 1) %>%
  pack_rows("Single Predictor Model", 2, 5)  %>%
  pack_rows("Multipe Predictors Model", 6, 6) %>%
  pack_rows("Regularized Model", 7, 7) %>%
  row_spec(0, bold = T)  %>%
  # to highlight the last row
  row_spec(7:7, bold = T, color = "white", background = "#D7261E")
```

Model	Dataset	RMSE
Basic Prediction Model		
Rating Mean Naive Model	test_set	1.06005
Single Predictor Model		
Movie Effect Model	test_set	0.94296
User Effect Model	test_set	0.97771
Year Released Effect Model	test_set	1.04928
Year Rated Effect Model	test_set	1.05839
Multipe Predictors Model		
Movie & User Effect Model	test_set	0.86468
Regularized Model		
Regularized Movie & User Effect Model	**test_set | lambda :5**	**0.86414**

This model generated the desired RMSE value of 0.8641362 with an improvement over other models. Next, let's check the effectiveness of the model on the validation.

5.6 Evaluating the selected model on validation dataset

```
##############################################################################
# Predict via regularisation, movie and user effect model on validation set
##############################################################################

# lambda is the previously selected optimal lambda

# calcuate mean on edx set
mu_edx <- mean(edx$rating)

# calculate movie bias using lambda
movie_bias <- edx %>%
group_by(movieId) %>%
summarise(b_i = sum(rating - mu_edx)/(n()+lambda))
```

```r
# calculate user bias using lambda
user_bias <- validation %>%
left_join(movie_bias, by="movieId") %>%
group_by(userId) %>%
summarise(b_u = sum(rating - b_i - mu_edx)/(n()+lambda))

# true_and_predicted_ratings is a temporary dataset to store
# predicted ratings along with other edx colums
true_and_predicted_ratings <- validation %>%

# first left join validations with movie bias by movieId and then
# join again with user bias using userId
left_join(movie_bias, by = "movieId") %>%
left_join(user_bias, by = "userId") %>%
mutate(pred = mu_edx + b_i + b_u)

# to pull predicted ratings for RMSE calculation
predicted_ratings <- true_and_predicted_ratings %>%
pull(pred)

# calculate RMSE on validation set
model_reg_val <- RMSE(validation$rating, predicted_ratings)

# append results to the table
rmse_results <- bind_rows(rmse_results,
                          tibble(Model="Regularized Movie & User Effect Model",
                          Dataset = paste0("Validation Set   |   lambda :", toString(lambda)),
                          RMSE = round(model_reg_val, digits = 5)))
rmse_results %>%
  kable("latex", booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hover", "condensed")) %>%

  # to group rows in various catagories
  pack_rows("Basic Prediction Model", 1, 1) %>%
  pack_rows("Single Predictor Model", 2, 5)  %>%
  pack_rows("Multipe Predictors Model", 6, 6) %>%
  pack_rows("Regularized Model", 7, 8) %>%
  row_spec(0, bold = T) %>%
  # to highlight the last row
  row_spec(8:8, bold = T, color = "white", background = "#D7261E")
```

Model	Dataset	RMSE
Basic Prediction Model		
Rating Mean Naive Model	test_set	1.06005
Single Predictor Model		
Movie Effect Model	test_set	0.94296
User Effect Model	test_set	0.97771
Year Released Effect Model	test_set	1.04928
Year Rated Effect Model	test_set	1.05839
Multipe Predictors Model		
Movie & User Effect Model	test_set	0.86468
Regularized Model		
Regularized Movie & User Effect Model	test_set | lambda :5	0.86414
Regularized Movie & User Effect Model	**Validation Set | lambda :5**	**0.84286**

This model generated the desired RMSE value of *0.84286* on validation set. Next, we list and compare all the results in the next section.

6. Results

6.1 Modeling result comparison and performance

The result of RMSE derived from various models is listed below. As we can clearly see there is an improvement as we moved from the basic Average Rating model to the Regularized model. From all the models explored in this project, the **regularized Movie and User Effects model** seems to be the best and the finally proposed model with the lowest RMSE (highlighted in green).

```
# to display result tables with RMSE model results
rmse_results %>%
kable("latex", booktabs = T) %>%
kable_styling(latex_options = c("striped", "hover", "condensed")) %>%
pack_rows("Basic Prediction Model", 1, 1) %>%
pack_rows("Single Predictor Model", 2, 5)  %>%
pack_rows("Multipe Predictors Model", 6, 6) %>%
pack_rows("Regularized Model", 7, 8) %>%
row_spec(0, bold = T) %>%
row_spec(8:8, bold = T, color = "white", background = "#3DDB48")
```

Model	Dataset	RMSE
Basic Prediction Model		
Rating Mean Naive Model	test_set	1.06005
Single Predictor Model		
Movie Effect Model	test_set	0.94296
User Effect Model	test_set	0.97771
Year Released Effect Model	test_set	1.04928
Year Rated Effect Model	test_set	1.05839
Multipe Predictors Model		
Movie & User Effect Model	test_set	0.86468
Regularized Model		
Regularized Movie & User Effect Model	test_set | lambda :5	0.86414
Regularized Movie & User Effect Model	**Validation Set | lambda :5**	**0.84286**

7. Conclusion

7.1 Brief summary

The main aim of the project is to develop a recommendation system to predict movie ratings. The provided 10M MovieLens dataset is a real-world challenge that seasoned data scientists would have handled. From data setup, data wrangling as well as exploratory analysis and ggplot based visualization graphs to various modeling approaches, the concepts learned throughout the series of courses are implemented.

Initially, basic prediction models including Rating Mean and single predictors based models are developed, later more complex multiple predictors models are attempted and finally **regularized Movie and User Effect predictive model** is implemented. A RMSE value of **0.84286** is attained on validation set which is less than the targeted value.

7.2 Future work

The model using the Movie and User effects seems to have the best performance among the other two predictor models; however, multiple combinations of more than two potential predictors can be explored for further improvements as future work.

For example, *Genre-specific* effect on a user preference to rate a movie can be further explored. If a user rates movies with Drama genre high, the *Drama* effect for that user can be taken into account while predicting rating as the user is more likely to give higher ratings to movies with Drama as a sub-genre in combined genre Drama|Comedy and vice-versa. Such a genre-specific effect can be used along with Movie and User Effects for further improvement.

Furthermore, Other more complex modeling approaches such as XGBoost, NNET, etc could be used to further improve the performance.

7.3 Limitations

Initially, k-fold cross-validation with k=10 was planned; however, due to computing resource constraint it was not successful. So, a simpler alternative approach to partition provided edx dataset into training and test sets are utilized.

Also, due to the constraint, more sophisticated models with higher performance didn't succeed and thus not included in this report. Those would require powerful computing machines.

One such example is splitting single pipe-delimited concatenated `genres` categories into single categories and use it in genre-specific effect, mentioned in '*Future Work* section'. However, the target RMSE was achieved with the Regularized Movie and User Effect model implemented in the project.

8. References

- Data Science textbook by Rafael Irizarry
  - Chapter 34.7 Recommendation systems
  - Chapter 34.9 Regularization

9. Github Repo

- https://github.com/jha-r/Harvardx-PH125.9x-Capstone-MovieLens