Movielens - HarvardX Capstone Project

Yuko Hayakawa

2024-03-20

Introduction

This project is the Capstone Final course of the HarvardX Professional in Data Science. We will explore and visually examine the MovieLens data set. The dataset ,titled "MovieLens", was designed to generate personalized film recommendations. We will create training and test sets and develop a machine learning model to predict the movie ratings of the validation set to achieve RMSE (a root mean square error) of less than 0.8649. RMSE is the standard deviation of the prediction error and is a measure of the difference between observed and predicted. It is a common metric for measuring the effectiveness of machine learning models. First we will examine the "MovieLens" 10M Dataset for the purposes of this project.

Method and Analysis

First, we will begin by preparing the data and loading it from the GroupLens Website. And the data will be split int datasets named "edx" and "final_holdout_test". Data cleaning will be performed, NAs will be removed, data will be characterized, and the characteristics will be visualized and captured visually. Data insight will be used to identify biases that may reduce the predictive accuracy of the model. Various effects will be considered as features to improve the model, and the final RMSE will be less than 0.8649.

Data Preparation

Install and Require Packages

```r
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(readr)) install.packages("readr")
```

```r
if(!require(dplyr)) install.packages("dplyr")
if(!require(tidyr)) install.packages("tidyr")
if(!require(stringr)) install.packages("stringr")
if(!require(ggplot2)) install.packages("ggplot2")
if(!require(gridExtra)) install.packages("gridExtra")
if(!require(dslabs)) install.packages("dslabs")
if(!require(data.table)) install.packages("data.table")
if(!require(ggrepel)) install.packages("ggrepel")
if(!require(ggthemes)) install.packages("ggthemes")
if(!require(ggthemes)) install.packages("tinytex")

library(tidyverse)
library(caret)
library(readr)
library(dplyr)
library(tidyr)
library(stringr)
library(ggplot2)
library(gridExtra)
library(dslabs)
library(data.table)
library(ggrepel)
library(ggthemes)
library(tinytex)
```

Download Data set

All Data for the MovieLens Data set will be obtained from the following sources:

- https://grouplens.org/datasets/movielens/10m/
- http://files.grouplens.org/datasets/movielens/ml-10m.zip

```r
# Download Data Sets
options(timeout = 120)
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
```

```r
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)
```

Create Data Frames

We will create a data frame from the downloaded data set.

```r
# Rating Data
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

# Movie Data
movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

# Join these 2 Data
movielens <- left_join(ratings, movies, by = "movieId")
```

Create Test Set for Validation

We must partition training and test sets.

```r
# Test set for validation will be 10% of MovieLens data
# Set Seed to 1
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

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

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

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

# Remove unnecessary files
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Exploratory Analysis

Confirm the Data Overview

See the head of data

```r
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

```
## 7        Children|Comedy|Fantasy
```

Confirm the missing value NA

```r
anyNA(edx)
```

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

See the number of rows and columns

```r
glimpse(edx)
```

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

We can confirm that there are 9,000,055 Rows and 6 Columns. userID, movieId, rating, timestamp, title and genres

See unique numbers of each columns

```r
edx %>%
  summarize(unique_users = length(unique(userId)),
            unique_movies = length(unique(movieId)),
            unique_genres = length(unique(genres)))
```

```
##   unique_users unique_movies unique_genres
## 1        69878         10677           797
```

There are 69,878 unique userIDs, 10,677 unizue movieIds, and 797 unique combinations of genres.

See the rating

```r
sort(unique(edx$rating))
```

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

There are 0 to 5 points for every 0.5points.

Correlation

```
edx %>%
  select(-title, -genres) %>%
  mutate(
    userId = as.numeric(userId),
    movieId = as.numeric(movieId),
    timestamp = as.numeric(timestamp)
  ) %>%
  cor()
```

```
##                userId      movieId       rating   timestamp
## userId    1.000000000  0.004934086  0.002313643  0.01514928
## movieId   0.004934086  1.000000000 -0.006535696  0.37392821
## rating    0.002313643 -0.006535696  1.000000000 -0.03473968
## timestamp 0.015149277  0.373928209 -0.034739685  1.00000000
```

Rating shows that there is a release, timasamp and so on. It might be considered that the elapsed time since the release of the film and the timing of viewing and rating affect the Rating.

Confirm the Detail of Each Items

Timestamp

See the head of timestamp

Use the as.POSIXct function to convert timestamp to a date-time object.

```
# cleaning
edx <- edx %>%
  mutate(timestamp = as.POSIXct(timestamp, origin = "1970-01-01",
                                tx = "EST"))
edx$timestamp <- format(edx$timestamp, "%Y")
names(edx)[names(edx) == "timestampe"] <- "RatingYear"
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5      1996              Boomerang (1992)
## 2      1     185      5      1996               Net, The (1995)
## 4      1     292      5      1996               Outbreak (1995)
```

```
## 5       1      316       5       1996                Stargate (1994)
## 6       1      329       5       1996 Star Trek: Generations (1994)
## 7       1      355       5       1996        Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

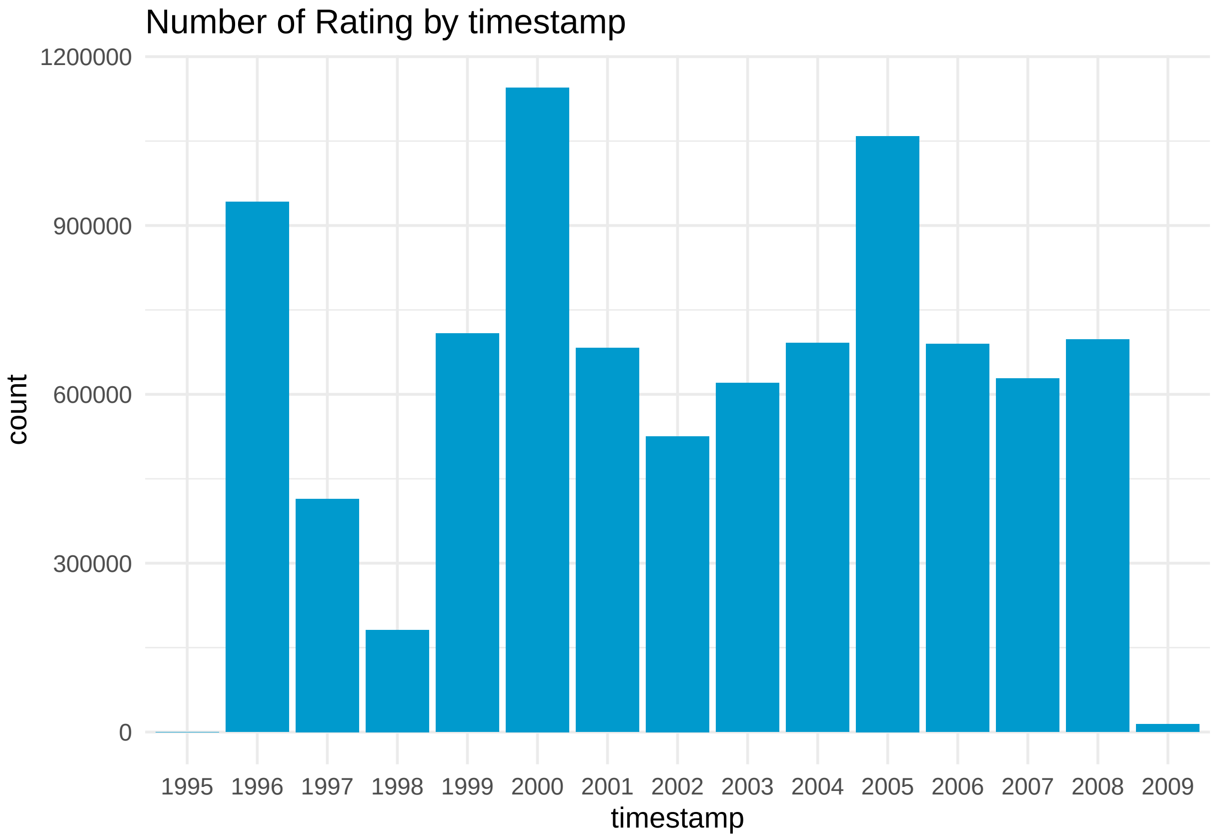
See the range of timestamp

```r
# confirm the range
range(edx$timestamp)
```

```
## [1] "1995" "2009"
```

See the number of timestamp

```r
# Numbers of Rating by year
timestamp <- edx %>%
  group_by(timestamp) %>%
  summarize(count = n())

timestamp %>%
  ggplot(aes(timestamp, count)) +
  geom_bar(stat = "identity", fill = "deepskyblue3") +
  labs(
    title = "Number of Rating by timestamp",
    ylab = "number of Rating"
  ) +
  theme_minimal()
```



TIMESTAMP is from 1995 to 2009, and it was found that the most submissions were made in 2000, 2005, and 1996, in that order.

See the number of timestamp by title and timestamp

```
edx %>%
  group_by(title, timestamp) %>%
  summarize(n = n()) %>%
  arrange(-n)
```

```
## # A tibble: 76,050 x 3
## # Groups:   title [10,676]
##    title                        timestamp     n
##    <chr>                        <chr>     <int>
##  1 Batman (1989)                1996      12015
##  2 Dances with Wolves (1990)    1996      11521
##  3 Apollo 13 (1995)             1996      11389
##  4 Pulp Fiction (1994)          1996      10921
##  5 Fugitive, The (1993)         1996      10895
##  6 True Lies (1994)             1996      10834
```

```
##   7 Forrest Gump (1994)       1996        9983
##   8 Batman Forever (1995)     1996        9905
##   9 Aladdin (1992)            1996        9851
##  10 Jurassic Park (1993)      1996        9769
## # i 76,040 more rows
```

The most common submission was made by Batman in 1996.

Users

Confirm the summary

```r
# Number of rating per userId
edx %>%
  group_by(userId) %>%
  summarise(n = length(userId), avg_rating = mean(rating)) %>%
  summarise(min_n = min(n), max_n = max(n), avg_n = mean(n), median_n = median(n))
```

```
## # A tibble: 1 x 4
##   min_n max_n avg_n median_n
##   <int> <int> <dbl>    <dbl>
## 1    10  6616  129.       62
```

The number of posts per User is 10~166 with a Mean of 128.7967 and a Median of 62.

Movies

Confirm the "Release Year"

```r
# Mutate a column "Release Year" from title and confirm the range
# Release Year
edx <- edx %>%
  mutate(edx, release = str_sub(title, -5, -2))
range(edx$release)
```

```
## [1] "1915" "2008"
```

The data is for films released between 1915~2008.

See the number and average of rating by title

```r
# remove release year from title
edx$title <- str_sub(edx$title, 1, -8)

# Summary of Rating by Movie
edx %>%
  group_by(title) %>%
  summarise(n = n(), avg_rating = mean(rating)) %>%
  arrange(-n)
```

```
## # A tibble: 10,407 x 3
##    title                                          n avg_rating
##    <chr>                                      <int>      <dbl>
##  1 Pulp Fiction                               31362       4.15
##  2 Forrest Gump                               31079       4.01
##  3 Silence of the Lambs, The                  30382       4.20
##  4 Jurassic Park                              29360       3.66
##  5 Shawshank Redemption, The                  28015       4.46
##  6 Braveheart                                 26212       4.08
##  7 Fugitive, The                              26020       4.01
##  8 Terminator 2: Judgment Day                 25984       3.93
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) 25672       4.22
## 10 Batman                                     24585       3.38
## # i 10,397 more rows
```

Confirm the number of rating

```r
# Summary of Rating by Movie
edx %>%
  group_by(title) %>%
  summarise(n = n()) %>%
  summarise(min_n = min(n), max_n = max(n), avg_n = mean(n), median_n = median(n))
```

```
## # A tibble: 1 x 4
##   min_n max_n avg_n median_n
##   <int> <int> <dbl>    <int>
## 1     1 31362  865.      124
```

The number of ratings per film ranged from 1 to 31,362, with a Mean of 864.8 and a Median of 124.

See the summary of average of rating per title

```r
# make a Data Frame of Average of Rating by Movie
avg_rating <- edx %>%
  group_by(title) %>%
  summarise(n = n(), avg_rating = round(mean(rating),2)) %>%
  arrange(avg_rating)
summary(avg_rating$avg_rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   2.840   3.270   3.191   3.610   5.000
```

In the evaluation, the mean is 3.191 and the median is 3.27.

Genres

Genres are tied to multiple genres and cannot necessarily be categorized into unique genres. First, let's review the types and combinations of genres.

Combination Multiple Genres

See the number and average of rating by each combination Genres

```r
# Summary
genres_mix <- edx %>%
  group_by(genres) %>%
  summarise(count = n(), ave_rating = mean(rating)) %>%
  arrange(-count)
genres_mix
```

```
## # A tibble: 797 x 3
##    genres                         count ave_rating
##    <chr>                          <int>      <dbl>
##  1 Drama                         733296       3.71
##  2 Comedy                        700889       3.24
##  3 Comedy|Romance                365468       3.41
##  4 Comedy|Drama                  323637       3.60
##  5 Comedy|Drama|Romance          261425       3.65
##  6 Drama|Romance                 259355       3.61
##  7 Action|Adventure|Sci-Fi       219938       3.51
##  8 Action|Adventure|Thriller 149091       3.43
##  9 Drama|Thriller                145373       3.45
```

```
## 10 Crime|Drama               137387       3.95
## # i 787 more rows
```

There were 797 combinations of genres, indicating that many drama and comedy related films were submitted.

Unique Genres

We found that there are 797 possible combinations of genres. Therefore, we will divide each row into individual genres in a new data frame to analyze more detail.

See the number and average of rating by each unique Genres

```r
# Separate Genres
genres_unique <- edx %>%
  separate_rows(genres, sep = "\\|")

# Numbers of each unique Genres
genres_unique %>%
  group_by(genres) %>%
  summarise(count = n(), ave_rating = mean(rating), med_rating = median(rating)) %>%
  arrange(-count)
```

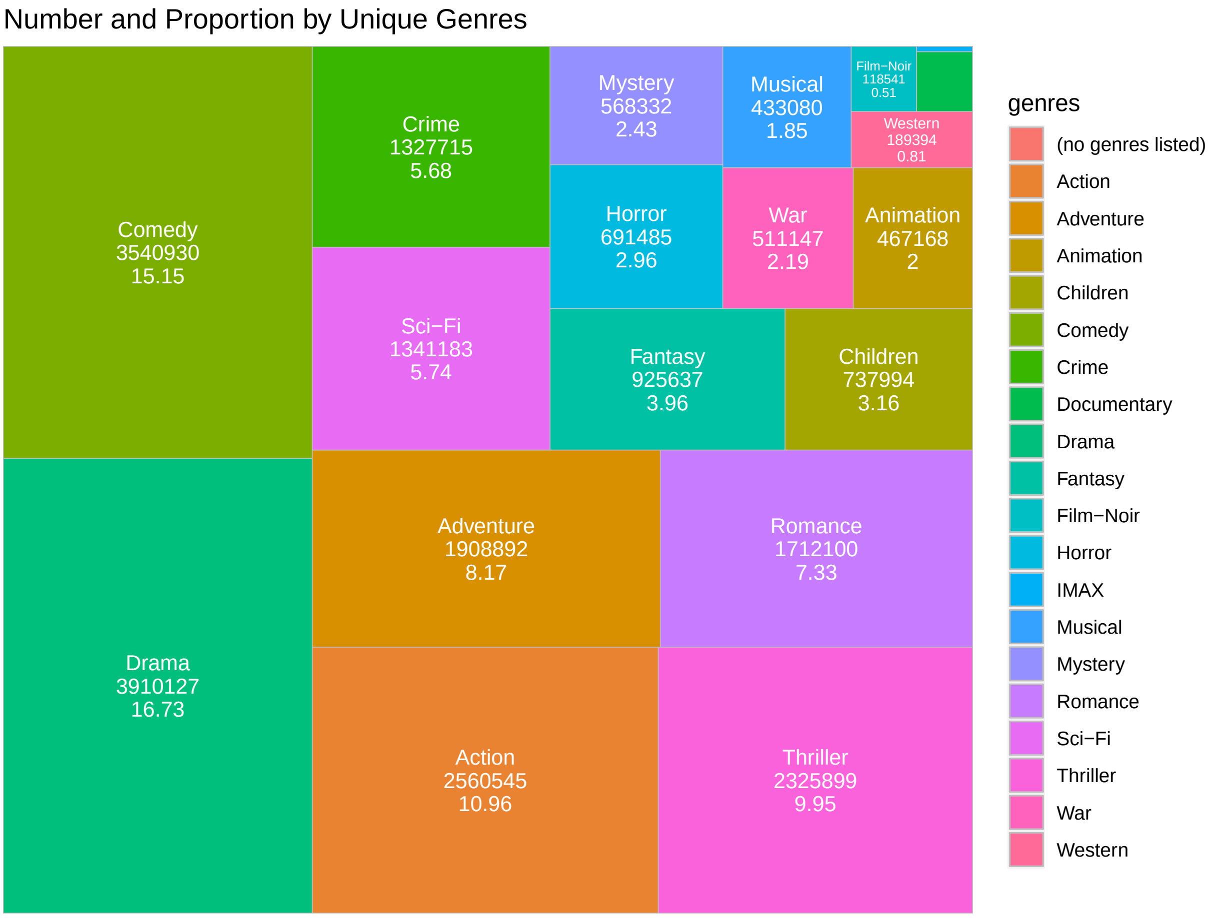
```
## # A tibble: 20 x 4
##    genres               count ave_rating med_rating
##    <chr>                <int>      <dbl>      <dbl>
##  1 Drama              3910127       3.67        4
##  2 Comedy             3540930       3.44        3.5
##  3 Action             2560545       3.42        3.5
##  4 Thriller           2325899       3.51        3.5
##  5 Adventure          1908892       3.49        3.5
##  6 Romance            1712100       3.55        4
##  7 Sci-Fi             1341183       3.40        3.5
##  8 Crime              1327715       3.67        4
##  9 Fantasy             925637       3.50        3.5
## 10 Children            737994       3.42        3.5
## 11 Horror              691485       3.27        3.5
## 12 Mystery             568332       3.68        4
## 13 War                 511147       3.78        4
## 14 Animation           467168       3.60        4
## 15 Musical             433080       3.56        4
```

```
## 16 Western                189394        3.56         4
## 17 Film-Noir              118541        4.01         4
## 18 Documentary             93066        3.78         4
## 19 IMAX                     8181        3.77         4
## 20 (no genres listed)          7        3.64         3.5
```

We can see there are 20 unique genres.

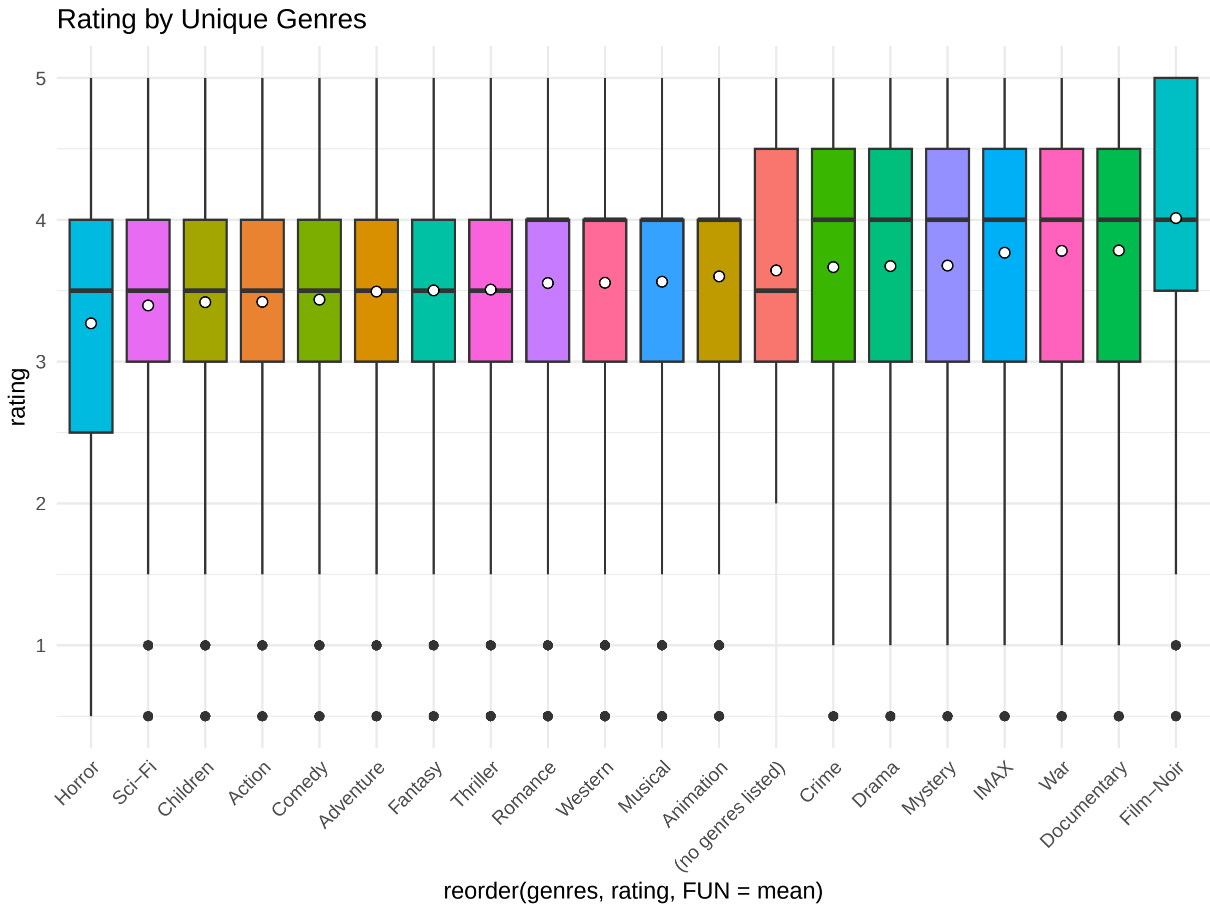
Make a Tree-map of unique genres

```r
# Make a Data Frame of summarized unique Genres
genres_unique_1 <- genres_unique %>%
  group_by(genres) %>%
  summarise(n = n(), avg_rating = mean(rating), median_rating = median(rating))

# Mutate the percentage of unique genres
genres_unique_1 <- genres_unique_1 %>%
  mutate(percent = n / sum(n) * 100) %>%
  arrange(-n)

if(!require(ggthemes)) install.packages("treemapify")
library(treemapify)

genres_unique_1 %>%
  ggplot(aes(area = n, fill = genres,
         label = paste(genres, n, round(percent, 2), sep = "\n"))) +
  geom_treemap() +
  labs(
    title = "Number and Proportion by Unique Genres"
  ) +
  geom_treemap_text(colour = "white",
                    place = "centre",
                    size = 10)
```



Make a Box plot by unique genres

```
genres_unique %>%
  ggplot(aes(x = reorder(genres, rating, FUN=mean), y = rating, fill = genres)) +
  geom_boxplot() +
  theme_minimal() +
  stat_summary(fun = "mean", geom = "point", shape = 21, size = 2., fill = "white") +
  labs(title = "Rating by Unique Genres") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none")
```



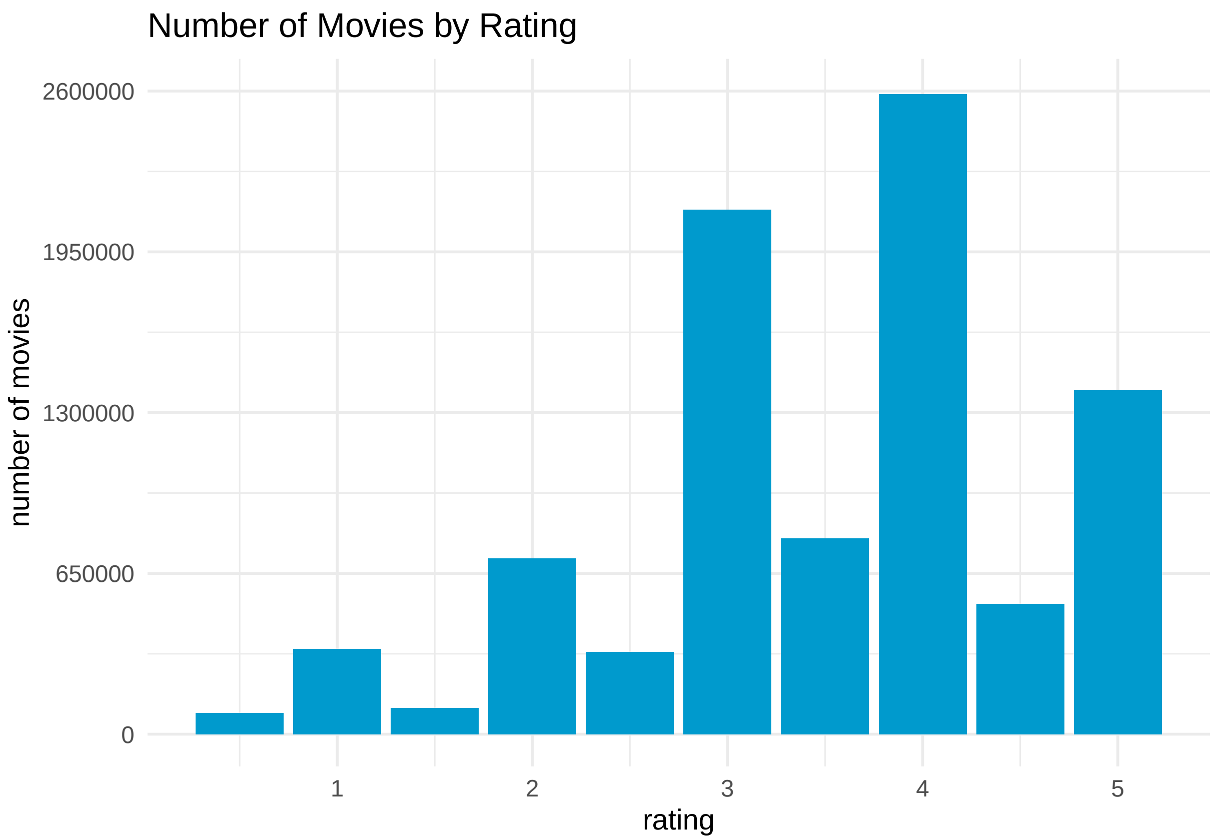
Sorting by Average Rating makes it easier to understand, with “Film-Noir” having the highest rating, followed by “Documentary”, “War”, and “IMA”.

Rating

See the number of movie by rating

```
# Summarize per Rating
Rating <- edx %>%
  group_by(rating) %>%
  summarize(n = n())

# Make a Bar-plot of number of Rating
Rating  %>%
  ggplot(aes(rating, n)) +
  geom_bar(stat = "identity", fill = "deepskyblue3") +
  theme_minimal() +
  labs(
```

```r
    title = "Number of Movies by Rating",
    x = "rating",
    y = "number of movies") +
  scale_y_continuous(breaks = seq(0, 2600000, length = 5), limits = c(0,2600000)) +
  theme_minimal()
```



Many of the evaluations were submitted to 4pt and 3pt, and there were few submissions to .5 at any point.

See the number and average of Rating by Users

```r
edx %>%
  group_by(userId) %>%
  summarise(count = length(userId), avg_rating = mean(rating)) %>%
  arrange(-count)
```

```
## # A tibble: 69,878 x 3
##    userId count avg_rating
##     <int> <int>      <dbl>
##  1  59269  6616       3.26
```

```
##   2  67385  6360        3.20
##   3  14463  4648        2.40
##   4  68259  4036        3.58
##   5  27468  4023        3.83
##   6  19635  3771        3.50
##   7   3817  3733        3.11
##   8  63134  3371        3.27
##   9  58357  3361        3.00
##  10  27584  3142        3.00
## # i 69,868 more rows
```

Movie Age

We will examine the number of years that have elapsed since a film was released.

Confirm the Release Year

```r
# See the numbers and average of Rating per release years
release_avg_rating <- edx %>%
  group_by(release) %>%
  summarise(count = n(), avg_rating = mean(rating))

# Mutate Movie Age
edx$release <- as.numeric(edx$release)
edx <- edx %>%
  mutate(movie_age = 2023 - release)

# Summarize Movie Age
summary(edx$movie_age)
```

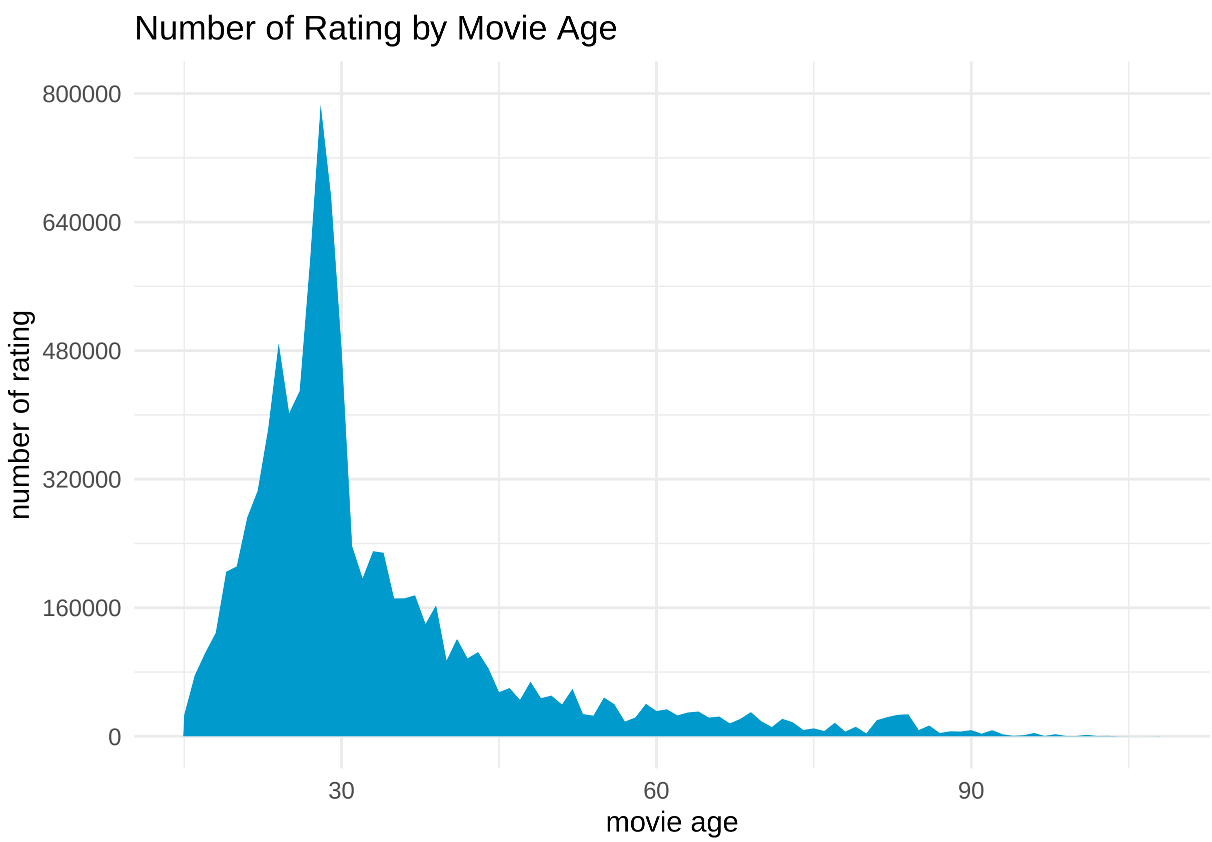
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   15.00   25.00   29.00   32.78   36.00  108.00
```

The elapsed time since release was calculated to be 15~108 years, with a mean of 32.78 years and a median of 29 years for the movie data.

See the numbers by Movie Age

```r
# Make Data Frame of number and average Rating per Movie Age
movie_age_rating <- edx %>%
  group_by(movie_age) %>%
```

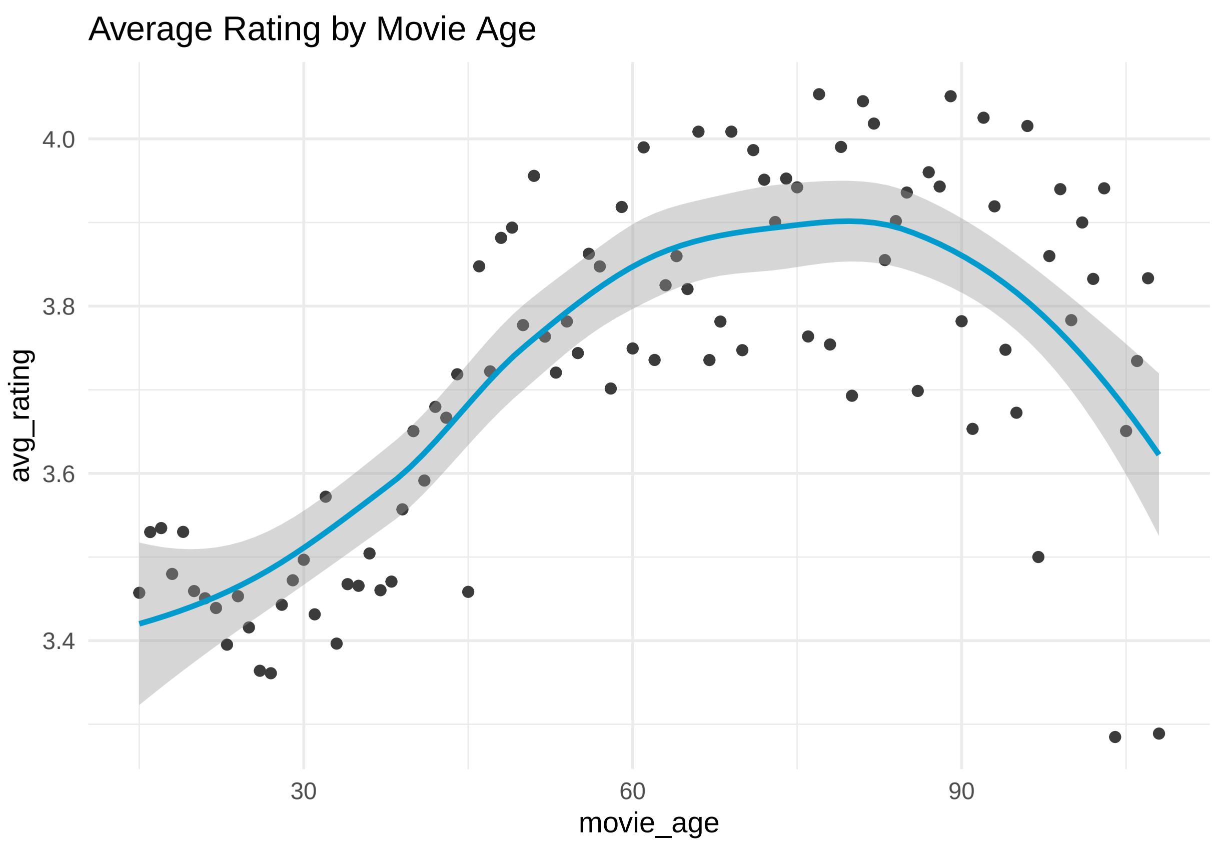
```
  summarise(count = n(), avg_rating = mean(rating))

# Make a Bar-plot of number of Movie Age
movie_age_rating %>%
  ggplot(aes(movie_age, count)) +
  geom_area(fill = "deepskyblue3") +
  labs(
    title = "Number of Rating by Movie Age",
    x = "movie age",
    y = "number of rating") +
  scale_y_continuous(breaks = seq(0, 800000, length = 6), limits = c(0,800000)) +
  theme_minimal()
```



Make a scatter-plot of average Rating per Movie Age

```
movie_age_rating %>%
  ggplot(aes(movie_age, avg_rating)) +
  geom_point(color = "grey23") +
  geom_smooth(color = "deepskyblue3") +
```

```r
  theme_minimal() +
  labs(title = "Average Rating by Movie Age")
```



We can see from this graph that there is a positive correlation between the average rating of each movie and Movie Age. We also find that older movies tend to lead to higher ratings than newer movies. However, ratings begin to decline when older than around 90 years. From these facts, it can be inferred that Movie Age has a meaningful impact on Rating.

Modeling

Define RMSE Function

The formula for RMSE can be defined as follows with:

- $\bar{y}_{u_1 i}$ = the prediction of movieId by user
- $y_{u_1 i}$ = the rating of movieId i by userId
- N = the number of userId and movieId combinations and the sum of these different combinations

$$\sqrt{\frac{1}{N}\sum_{u_1 i}(\hat{y}_{u_1 i} - y_{u_1 i})^2}$$

```
# define RMSE Function
RMSE <- function(actual_rating, predicted_rating){
  sqrt(mean((actual_rating - predicted_rating)^2))
}
```

Modeling

We will try calculate RMSE with consideration of combining possible effects. To do so, assign the following:

- μ = average rating
- $\varepsilon_{u_1 i}$ = independent Errors centered at 0
- bi = MovieId effects
- bu = UserId effects
- ba = Movie Age effect

Benchmark Model_Without Effect

The Benchmark Model calculates the Normal RMSE based on the mean of the edx dataset.

The formula can be defined as follows:

$$y_{u_1 i} = \mu + \varepsilon_{u_1 i}$$

```
# without effect
edx_mu <- mean(edx$rating)
RMSE_N <- RMSE(final_holdout_test$rating, edx_mu)
RMSE_N
```

```
## [1] 1.061202
```

Movie effect Model

We will try adding the Movie effect(bi)to Benchmark.

The formula can be defined as follows:

$$y_{u_1 i} = \mu + bi + \varepsilon_{u_1 i}$$

```
# calculate bi
bi <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - edx_mu))

# predict rating
pred_bi <- edx_mu + final_holdout_test %>%
  left_join(bi, by = "movieId") %>%
  .$b_i

RMSE_1 <- RMSE(final_holdout_test$rating, pred_bi)
RMSE_1
```

```
## [1] 0.9439087
```

Movie & User effect Model

We will try adding the Movie effect(bi) and User effect(bu) to Benchmark.

The formula can be defined as follows:

$$y_{u_1 i} = \mu + bi + bu + \varepsilon_{u_1 i}$$

```
# calculate bu
bu <- edx %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - edx_mu - b_i))

# predict rating
pred_bu <- final_holdout_test %>%
  left_join(bi, by = 'movieId') %>%
  left_join(bu, by = 'userId') %>%
  mutate(pred = edx_mu + b_i + b_u) %>%
  .$pred

RMSE_2 <- RMSE(final_holdout_test$rating, pred_bu)
RMSE_2
```

```
## [1] 0.8653488
```

Movie & User & Movie_Age effect Model

We will try adding the Movie effect(bi), User effect(bu) and Movie Age effect (ba)to Benchmark.

The formula can be defined as follows:

$$y_{u_1 i} = \mu + bi + bu + ba + \varepsilon_{u_1 i}$$

```
# calculate ba
ba <- edx %>%
  left_join(bi, by = 'movieId') %>%
  left_join(bu, by = 'userId') %>%
  group_by(movie_age) %>%
  summarise(b_a = mean(rating - edx_mu - b_i - b_u))

# mutate movie_age to final_holdout_test dataset
final_holdout_test <- final_holdout_test %>%
  mutate(release = str_sub(title, -5, -2))
final_holdout_test$release <- as.numeric(final_holdout_test$release)
final_holdout_test <- final_holdout_test %>%
  mutate(movie_age = 2023 - release)

# predict rating
pred_ba <- final_holdout_test %>%
  left_join(bi, by = 'movieId') %>%
  left_join(bu, by = 'userId') %>%
  left_join(ba, by = 'movie_age') %>%
  mutate(pred = edx_mu + b_i + b_u + b_a) %>%
  .$pred

RMSE_3 <- RMSE(final_holdout_test$rating, pred_ba)
RMSE_3
```

```
## [1] 0.8650043
```

Compare RMSEs

```r
result_1 <- tibble(Model_Type = c("RMSE w/o Effect", "Movie Effect", "Movie & User Effect", "Movie,U
                   RMSE = c(RMSE_N, RMSE_1, RMSE_2, RMSE_3)) %>%
  mutate(RMSE = sprintf("%0.5f", RMSE))

result_1 %>%
  knitr::kable()
```

Model_Type	RMSE
RMSE w/o Effect	1.06120
Movie Effect	0.94391
Movie & User Effect	0.86535
Movie,User & Movie Age Effect	0.86500

Four patterns of RMSEs were checked to take into account the impact, and the highest RMSE was 0.8650, which did not achieve the target value.

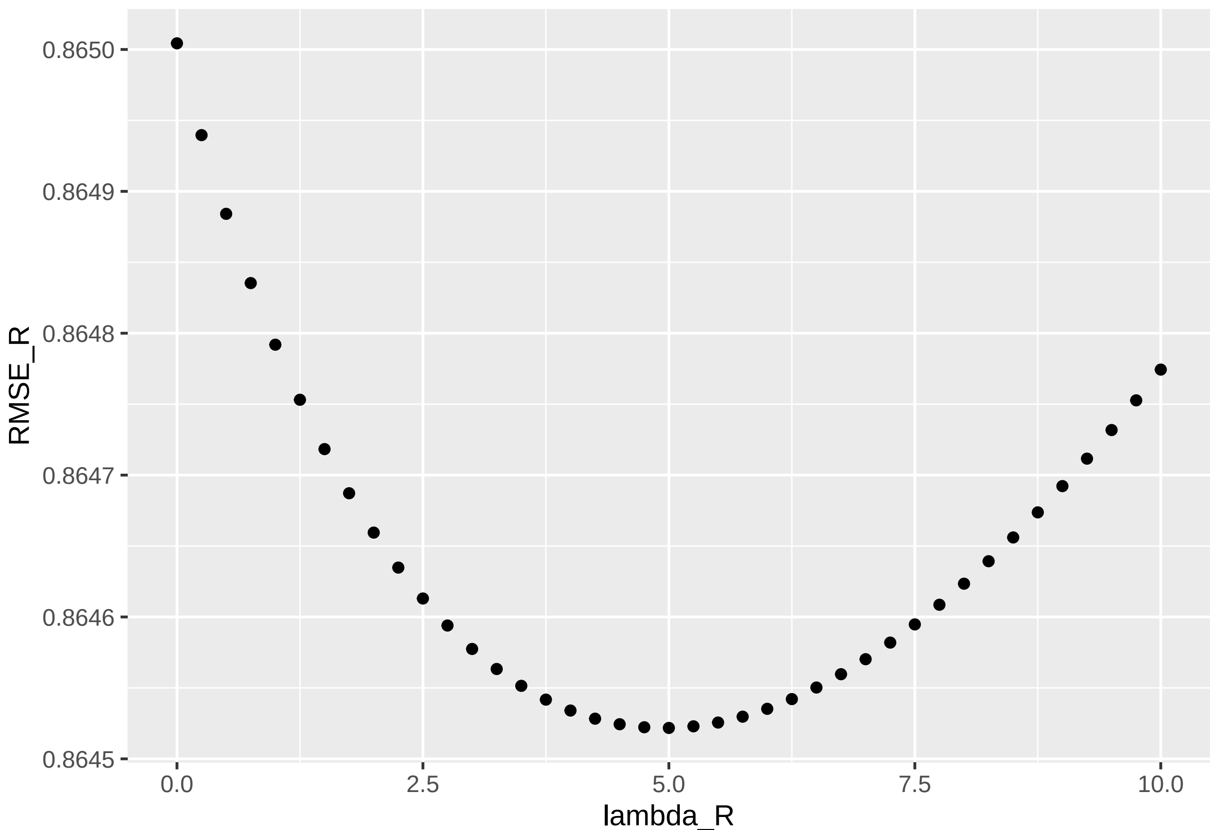
Movie & User & Movie_Age with Regularization Model

Each effects with Regularization

We will regularize with using sapply and add a tuning parameter lambda to minimize the RMSE. This fuction penalizes outliesrs from *bi*, *bu* and *baa* such as users, movies and movie_age with very few ratings to optimize the recommendation system.

```r
lambda_R <- seq(0, 10, 0.25)
RMSE_R <- sapply(lambda_R, function(i){
  edx_mu <- mean(edx$rating)

  bi_R <- edx %>%
    group_by(movieId) %>%
    summarise(b_i_R = sum(rating - edx_mu) / (n() + i))

  bu_R <- edx %>%
    left_join(bi_R, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u_R = sum(rating - edx_mu - b_i_R) / (n() + i))
```

```r
  ba_R <- edx %>%
    left_join(bi_R, by = "movieId") %>%
    left_join(bu_R, by = "userId") %>%
    group_by(movie_age) %>%
    summarise(b_a_R = sum(rating - edx_mu - b_i_R - b_u_R) / (n() + i))

  predict_rating <- final_holdout_test %>%
    left_join(bi_R, by = "movieId") %>%
    left_join(bu_R, by = "userId") %>%
    left_join(ba_R, by = "movie_age") %>%
    mutate(predict = edx_mu + b_i_R + b_u_R + b_a_R) %>%
    .$predict

  return(RMSE(predict_rating, final_holdout_test$rating))
})

qplot(lambda_R, RMSE_R)
```



```r
lambda_R_min <- lambda_R[which.min(RMSE_R)]
lambda_R_min
```

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

Minimum lambda is 5. This lambda is the best Tune for this model.

Calculate and Create Prediction

```r
bi_R <- edx %>%
    group_by(movieId) %>%
    summarise(b_i_R = sum(rating - edx_mu) / (n() + lambda_R_min))

bu_R <- edx %>%
    left_join(bi_R, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u_R = sum(rating - edx_mu - b_i_R) / (n() + lambda_R_min))

ba_R <- edx %>%
    left_join(bi_R, by = "movieId") %>%
    left_join(bu_R, by = "userId") %>%
    group_by(movie_age) %>%
    summarise(b_a_R = sum(rating - edx_mu - b_i_R - b_u_R) / (n() + lambda_R_min))

predict_rating_R <- final_holdout_test %>%
    left_join(bi_R, by = "movieId") %>%
    left_join(bu_R, by = "userId") %>%
    left_join(ba_R, by = "movie_age") %>%
    mutate(predict = edx_mu + b_i_R + b_u_R + b_a_R) %>%
    .$predict

RMSE_R <- RMSE(predict_rating_R, final_holdout_test$rating)
RMSE_R
```

```
## [1] 0.8645218
```

Compare RMSEs

```
result_2 <- tibble(Model_Type = c("RMSE w/o Effect", "Movie Effect", "Movie & User Effect", "Movie,U
                   RMSE = c(RMSE_N, RMSE_1, RMSE_2, RMSE_3, RMSE_R)) %>%
  mutate(RMSE = sprintf("%0.5f", RMSE))

result_2 %>%
  knitr::kable()
```

Model_Type	RMSE
RMSE w/o Effect	1.06120
Movie Effect	0.94391
Movie & User Effect	0.86535
Movie,User & Movie Age Effect	0.86500
Movie,User & Movie Age Effect with Regulatization	0.86452

Finally Optimization confirmed an RMSE below the target of 0.8649. The model with the lowest RMSE when applied to the test set was the regularized model with Movie , User and Movie Age effects. This was a significant improvement from the RMSE of the benchmark model used as the reference.

Conclusion

After testing five models to account for the effects of variables, the final model with regularization for user,movie,and movie age effects showed good results that is a RMSE of 0.86452. But there are biases that can be further explored to improve better the accuracy of the model. Additionally, to significantly improve the RMSE, methods such as matrix factorization can be used with the addition of genre influences. In addition, if User profile information were available, the frequency of viewing a movie and the ratio of the number of times and timing of Ratings to the number of times a movie is viewed would also be considered to derive more accurate results, taking into account the impact on Ratings.