Movielens Recommender System

Aditi Gupta

2024-10-29

Introduction

Netflix, currently the most popular streaming service in the world, launched a competition in 2006 promising $1 million in prize money for the winning team. The objective of the challenge was to create an algorithm that outperformed Netflix's baseline algorithm, Cine match, by 10%. The "Netflix Prize" contest highlighted the significance and financial benefits of research focused on improving existing recommendation systems, with companies such as social media firm Twitter hosting similar competitions of their own.

In the HarvardX: PH125.9x Data Science: Capstone course, an objective comparable to the "Netflix Prize" contest is presented to participants, albeit with the use of a different dataset. Rather than the datasets provided by Netflix, the 10M version of the MovieLens dataset provided by the research lab GroupLens will be used. As the name suggests, the dataset contains 10 million movie ratings, as well as 100,000 tag applications applied to 10,000 movies by 72,000 users.

Goal of the Project

The MovieLens Project is designed to encompass the various machine learning and computational concepts learned from previous courses in the HarvardX Professional Data Science Certificate program. Using the MovieLens dataset provided, the goal of the project is to train an algorithm with a RMSE score less than 0.86490.

The Root Mean Square Error, or RMSE, is the value used to assess the algorithm's performance. Generally, a lower RMSE score indicates better performance, and vice versa.

The following function computes the RMSE based on predicted and observed values:

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

Project Stages

The MovieLens project consisted of below stages, and as such, the report has been divided into these sections.

1) Create edx and final_holdout_test sets
2) Looking at data sets structure, data type, and data.
3) Pre-processing data
4) Exploratory Data Analysis
5) Modeling
6) Model evaluation and final model fitting on final_holdout_test dataset.
7) Conclusion
8) Future work
9) Thanks!
10) References

1. Create edx and final_holdout_test sets

I used the following code to generate the datasets. And develop the algorithm using the edx set. For a final test of the final algorithm, which will predict movie ratings in the final_holdout_test set as if they were unknown. RMSE will be used to evaluate how close it's predictions are to the true values in the final_holdout_test set.

```
############################################################
# Step 1: Create edx and final_holdout_test sets
############################################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tibble)) install.packages("tibble", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(rlang)) install.packages("rlang", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(corrplot)) install.packages("corrplot", repos = "http://cran.us.r-project.org")
if(!require(gplots)) install.packages("gplots", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(tibble)
library(caret)
library(rlang)
library(ggplot2)
library(dplyr)
library(corrplot)
library(gplots)

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

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"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

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

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))

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))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
#Warning message:
#In set.seed(1, sample.kind = "Rounding") :
#  non-uniform 'Rounding' sampler used
# 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,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out 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)

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

#####################################
# End of Provided Code
#####################################
```

2. Looking at data sets structure, data type, and data.

In this stage I was looking for missing values, columns, rows, summary of datasets "edx" and "final_holdout_test".

There were no missing values in either edx or final_holdout_test, ensuring data completeness for all columns. edx has 9 million observations, while final_holdout_test contains 1 million observations, giving ample data for training and validation.

```
# Check any missing value
anyNA(edx)
```

2.1 Checking for Missing Values

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

```
anyNA(final_holdout_test)
```

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

```
sapply(edx, function(x) sum(is.na(x)))
```

2.2 Counting the number of missing values in each column

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

```
sapply(final_holdout_test, function(x) sum(is.na(x)))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

```
sapply(edx, class)
```

2.3 Checking column names with data types

```
##      userId     movieId      rating   timestamp       title      genres
##   "integer"   "integer"   "numeric"   "integer" "character" "character"
```

```
sapply(final_holdout_test, class)
```

```
##      userId     movieId      rating   timestamp       title      genres
##   "integer"   "integer"   "numeric"   "integer" "character" "character"
```

```
dim(edx)
```

2.4 Listing the amount of observations and variables

```
## [1] 9000055       6
```

```
dim(final_holdout_test)
```

```
## [1] 999999      6
```

```
head(edx)
```

2.5 Looking at the first few rows and stats like mean, median, min, max, 1 and 3rd Qu.of numeric columns.

```
##   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
```

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
head(final_holdout_test)
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                      title
## 1                       Dumb & Dumber (1994)
## 2                       Jurassic Park (1993)
## 3                          Home Alone (1990)
```

```
## 4                                       Rob Roy (1995)
## 5                                  Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                    genres
## 1                                  Comedy
## 2       Action|Adventure|Sci-Fi|Thriller
## 3                          Children|Comedy
## 4               Action|Drama|Romance|War
## 5                            Crime|Drama
## 6 Action|Adventure|Horror|Sci-Fi|Thriller
```

```
summary(final_holdout_test)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:999999      Length:999999
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

3. Pre-processing data

```
edx <- edx %>% mutate(year_rated = year(as_datetime(timestamp)))
```

3.1 Converting time stamp to year rated and add it to edx

```
unique(edx$year_rated)
```

3.2 Double checking any invalid value after conversion of time stamp

```
##  [1] 1996 1997 2006 2005 2001 2003 1999 2000 2002 1998 2007 2008 2004 2009 1995
```

```
edx <- edx %>% mutate(year_released = as.numeric(str_sub(title,-5,-2)))
```

3.3 Extracting the year released from title and add it to edx

```r
unique(edx$year_released)
```

3.4 Double checking unique values of year released, looking for any invalid values

```
##  [1] 1992 1995 1994 1993 1991 1937 1970 1977 1990 1996 1971 1983 1989 1974 1967
## [16] 1984 1997 1985 2000 1982 2002 2003 2004 2005 1976 1972 1958 1942 1939 1941
## [31] 1950 1951 1979 1955 1962 1960 1980 1988 1975 1987 1981 1969 1998 1999 1986
## [46] 2001 1952 1959 1946 1940 1954 1949 1973 1948 1933 1931 1963 1922 1943 1944
## [61] 1957 1953 1965 1978 1964 1968 1966 1961 1956 1935 1938 2006 2007 1932 2008
## [76] 1934 1945 1947 1927 1925 1930 1936 1929 1923 1928 1921 1926 1915 1924 1916
## [91] 1917 1918 1920 1919
```

```r
edx <- edx %>%
  mutate(ages = pmax(as.numeric(year_rated) - as.numeric(year_released), 0))
```

3.5 Calculating the movie age when movie was rated and adding it to edx

```r
unique(edx$ages)
```

3.6 Double checking any invalid value of ages

```
##  [1]  4  1  2  3  5 59 26 20  7 14 11 10 12 16 31 38 21 13  8  9  6 17 24 15 25
## [26] 39 55 58 56 47 46 18 42 35 37 23 30 29 32  0 51 45 44 62 64 57 63 49 41 54
## [51] 43 19 70 72 36 34 40 81 28 52 60 22 50 53 27 33 66 67 61 69 68 48 65 74 75
## [76] 71 77 73 79 76 83 78 85 80 84 91 82 90 89 88 86 87 93 92
```

```r
edx$ages[edx$ages == -1 | edx$ages == -2] <- 0
```

3.7 Replacing the values -1 and -2 with 0

```r
non_finite_ages <- edx %>%
  filter(!is.finite(ages))
```

3.8 Filtering out and displaying rows with non-finite values in the 'ages' column

```
unique(edx$ages)
```

3.9 Verifying if the replacement worked

```
##  [1]  4  1  2  3  5 59 26 20  7 14 11 10 12 16 31 38 21 13  8  9  6 17 24 15 25
## [26] 39 55 58 56 47 46 18 42 35 37 23 30 29 32  0 51 45 44 62 64 57 63 49 41 54
## [51] 43 19 70 72 36 34 40 81 28 52 60 22 50 53 27 33 66 67 61 69 68 48 65 74 75
## [76] 71 77 73 79 76 83 78 85 80 84 91 82 90 89 88 86 87 93 92
```

```
# Did same changes of the data pre-processing for final_holdout_test set
final_holdout_test <- final_holdout_test %>% mutate(year_rated = year(as_datetime(timestamp)))
unique(final_holdout_test$year_rated)
```

3.10 Did same changes of the data pre-processing for final_holdout_test set

```
##  [1] 1996 1997 2006 2005 2001 2003 1999 2000 2002 1998 2007 2008 2004 2009 1995
```

```
final_holdout_test <- final_holdout_test %>% mutate(year_released = as.numeric(str_sub(title,-5,-2)))
unique(final_holdout_test$year_released)
```

```
##  [1] 1994 1993 1990 1995 1972 1997 2001 1989 1991 1939 1982 1971 1979 1977 2000
## [16] 1969 1954 1942 1944 1983 1984 1998 1980 1956 2002 1950 1996 1992 1951 1985
## [31] 1988 1999 1981 1987 2003 2004 1978 1973 1974 2005 1986 1931 1958 1964 1960
## [46] 1962 1976 1937 1940 1952 1968 1967 1963 1965 2006 1957 1953 1975 1936 1946
## [61] 1949 1961 1970 1941 1922 1955 1959 1932 1923 1929 1966 1934 1947 2007 1948
## [76] 1933 2008 1927 1935 1945 1928 1930 1921 1938 1943 1925 1926 1915 1920 1918
## [91] 1919 1924 1916 1917
```

```
final_holdout_test <- final_holdout_test %>% mutate(ages = as.numeric(year_rated)-as.numeric(year_releas
# Replace the values -1 and -2 with 0
final_holdout_test$ages[final_holdout_test$ages == -1 | final_holdout_test$ages == -2] <- 0

# Verify if the replacement worked
unique(final_holdout_test$ages)
```

```
##  [1]  2  3  6 25  0 16  4  1  8 58 15 26 18 24 34 49 61 59 20 19  5 23 47 53 10
## [26]  9 11 12 13 22 54 14  7 17 29 28 72 45 39 43 60 57 30 27 64 40 41 33 31 35
## [51] 46 37 50 38 52 67 65 84 55 66 62 51 44 74 83 77 36 32 21 63 48 56 73 42 71
## [76] 68 86 76 69 82 70 75 78 92 81 80 89 87 79 88 85 90 93 91
```

```
head(edx)
head(final_holdout_test)
```

3.11 After changes, looking at the datasets.

```r
# Counting unique userId and movieId in the edx dataset
unique_users_edx <- length(unique(edx$userId))
unique_movies_edx <- length(unique(edx$movieId))

# Counting unique userId and movieId in the final_holdout_test dataset
unique_users_holdout <- length(unique(final_holdout_test$userId))
unique_movies_holdout <- length(unique(final_holdout_test$movieId))

# Printing the counts
cat("Unique userId in edx:", unique_users_edx, "\n")
```

3.12 Counting userId and movieId in edx and final_holdout_test datasets.

```
## Unique userId in edx: 69878
```

```r
cat("Unique movieId in edx:", unique_movies_edx, "\n")
```

```
## Unique movieId in edx: 10677
```

```r
cat("Unique userId in final_holdout_test:", unique_users_holdout, "\n")
```

```
## Unique userId in final_holdout_test: 68534
```

```r
cat("Unique movieId in final_holdout_test:", unique_movies_holdout, "\n")
```

```
## Unique movieId in final_holdout_test: 9809
```

4. Exploratory Data Analysis (EDA)

```r
# Summary statistics for the edx and final_holdout_test datasets
summary(edx)
```

4.1 Summary Statistics

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres            year_rated   year_released
##  Length:9000055     Length:9000055     Min.   :1995   Min.   :1915
##  Class :character   Class :character   1st Qu.:2000   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :2002   Median :1994
##                                        Mean   :2002   Mean   :1990
```

```
##                                          3rd Qu.:2005   3rd Qu.:1998
##                                          Max.   :2009   Max.   :2008
##       ages
##  Min.   : 0.00
##  1st Qu.: 2.00
##  Median : 7.00
##  Mean   :11.98
##  3rd Qu.:16.00
##  Max.   :93.00
```

```
summary(final_holdout_test)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres            year_rated   year_released
##  Length:999999      Length:999999      Min.   :1995   Min.   :1915
##  Class :character   Class :character   1st Qu.:1999   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :2002   Median :1994
##                                        Mean   :2002   Mean   :1990
##                                        3rd Qu.:2005   3rd Qu.:1998
##                                        Max.   :2009   Max.   :2008
##       ages
##  Min.   : 0
##  1st Qu.: 2
##  Median : 7
##  Mean   :12
##  3rd Qu.:16
##  Max.   :93
```

4.2 Slicing and dicing to look at top rating stats group by UserId, MovieId, and ages Insights: Popular movies like Pulp Fiction and Forrest Gump have high average ratings and significant rating counts, indicating their classic status. The most active users rated thousands of movies, but their average ratings vary, showing diverse rating behaviors.

```
# Table of 5 movies rated most (sorted by number of ratings in descending order)
top_movies <- edx %>%
  group_by(movieId, title) %>%
  summarize(n_rating = n(),
            avg_rating = mean(rating),
            max_rating = max(rating),
            min_rating = min(rating),
            .groups = 'drop') %>%
  arrange(desc(n_rating)) %>%
  distinct(movieId, .keep_all = TRUE) %>%
  slice(1:5)

# Display the table
```

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

movieId	title	n_rating	avg_rating	max_rating	min_rating
296	Pulp Fiction (1994)	31362	4.154789	5	0.5
356	Forrest Gump (1994)	31079	4.012822	5	0.5
593	Silence of the Lambs, The (1991)	30382	4.204101	5	0.5
480	Jurassic Park (1993)	29360	3.663522	5	0.5
318	Shawshank Redemption, The (1994)	28015	4.455131	5	0.5

```
# Table of 5 users who rated the most movies (sorted by number of ratings in descending order)
top_users <- edx %>%
  group_by(userId) %>%
  summarize(n_ratings = n(),
            avg_rating = mean(rating),
            max_rating = max(rating),
            min_rating = min(rating),
            .groups = 'drop') %>%
  arrange(desc(n_ratings)) %>%
  slice(1:5)

# Display the table
top_users %>%
  knitr::kable()
```

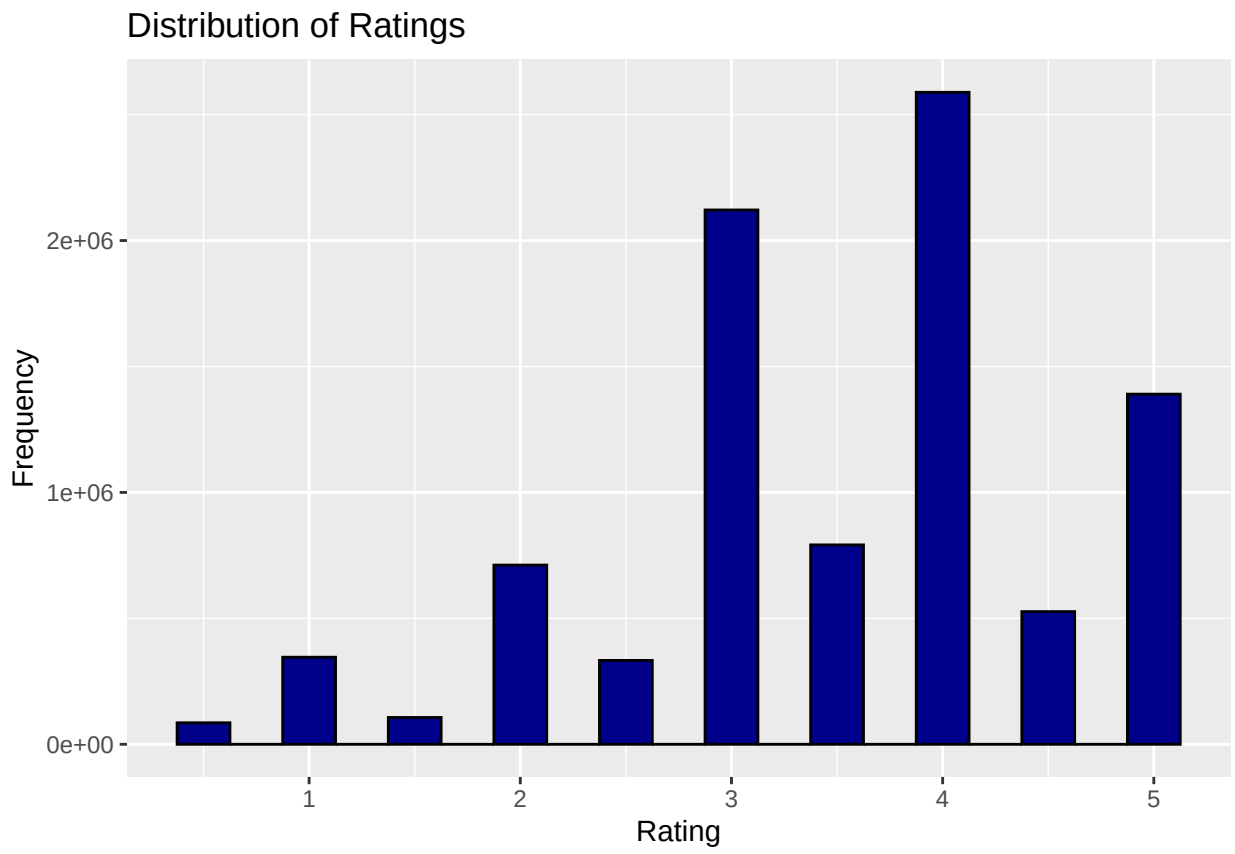
userId	n_ratings	avg_rating	max_rating	min_rating
59269	6616	3.264586	5	0.5
67385	6360	3.197720	5	1.0
14463	4648	2.403615	5	1.0
68259	4036	3.576933	5	0.5
27468	4023	3.826871	5	0.5

```
# Table of 20 ages of movie who rated the most movies (sorted by number of ratings in descending order)
top_users <- edx %>%
  group_by(ages) %>%
  summarize(n_ratings = n(),
            avg_rating = mean(rating),
            max_rating = max(rating),
            min_rating = min(rating),
            .groups = 'drop') %>%
  arrange(desc(n_ratings)) %>%
  slice(1:20)

# Display the table
top_users %>%
  knitr::kable()
```

ages	n_ratings	avg_rating	max_rating	min_rating
1	1068070	3.483180	5	0.5
2	853680	3.477037	5	0.5
3	647650	3.478478	5	0.5
4	473660	3.457079	5	0.5
5	436378	3.470467	5	0.5
6	410159	3.461274	5	0.5
0	381090	3.422455	5	0.5
7	354368	3.436889	5	0.5
8	320713	3.412401	5	0.5
9	292203	3.393785	5	0.5
10	281351	3.374975	5	0.5
11	264241	3.394863	5	0.5
12	244377	3.396113	5	0.5
13	238584	3.421478	5	0.5
14	222441	3.441270	5	0.5
15	194625	3.439586	5	0.5
16	170496	3.471680	5	0.5
17	154513	3.493729	5	0.5
18	144901	3.523116	5	0.5
19	133861	3.552080	5	0.5

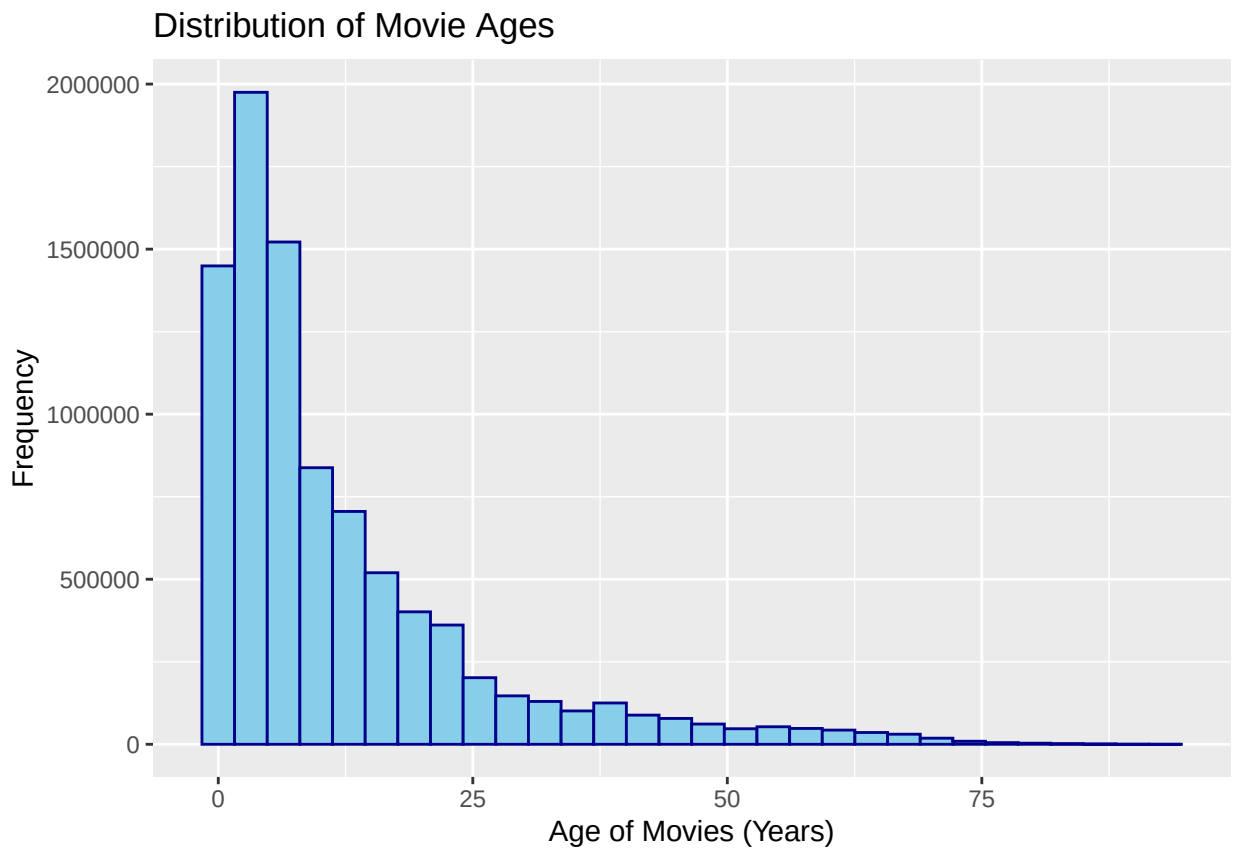
4.3 Ratings Distribution Insights: Ratings are skewed towards higher values, with 4 being the most common, suggesting users generally rate movies positively.

```
# Plotting distribution of ratings
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.25, fill = "blue4", color = "black") +
  ggtitle("Distribution of Ratings") +
  xlab("Rating") +
  ylab("Frequency")
```



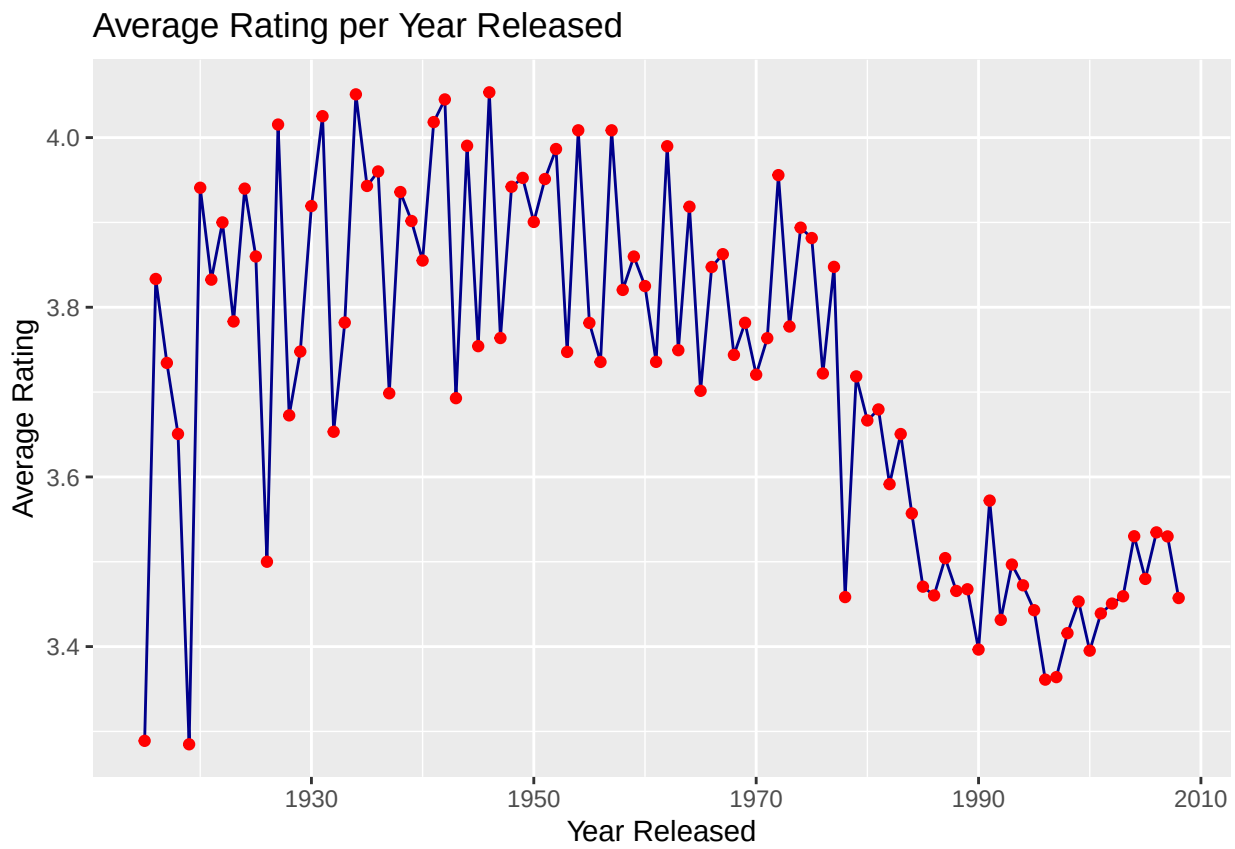
4.4 Movie Age Distribution Insights: Most ratings are for newer movies (0-10 years old), with older films receiving progressively fewer ratings. This indicates a bias towards recent releases.

```
# Plotting distribution of movie ages
edx %>%
  ggplot(aes(ages)) +
  geom_histogram(bins = 30, fill = "skyblue", color = "blue4") +
  ggtitle("Distribution of Movie Ages") +
  xlab("Age of Movies (Years)") +
  ylab("Frequency")
```



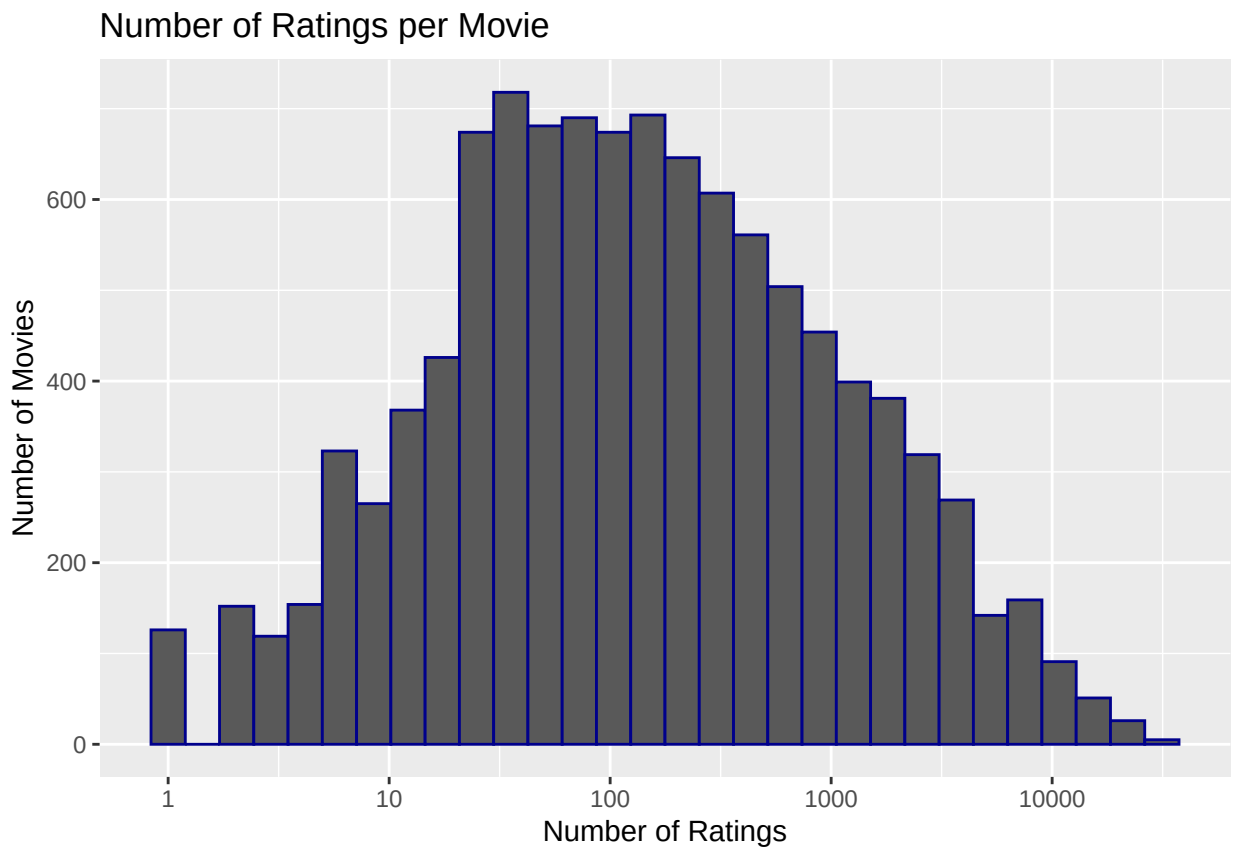
4.5 Average Rating per Year Released Insights: Movies from the late 1970s show a dip in average ratings, while older movies tend to have higher average ratings, likely because classics have endured over time.

```
# Line chart of average rating per year released
edx %>%
  group_by(year_released) %>%
  summarize(avg_rating = mean(rating, na.rm = TRUE)) %>%
  ggplot(aes(x = year_released, y = avg_rating)) +
  geom_line(color = "blue4") +
  geom_point(color = "red") +
  labs(title = "Average Rating per Year Released",
       x = "Year Released",
       y = "Average Rating")
```



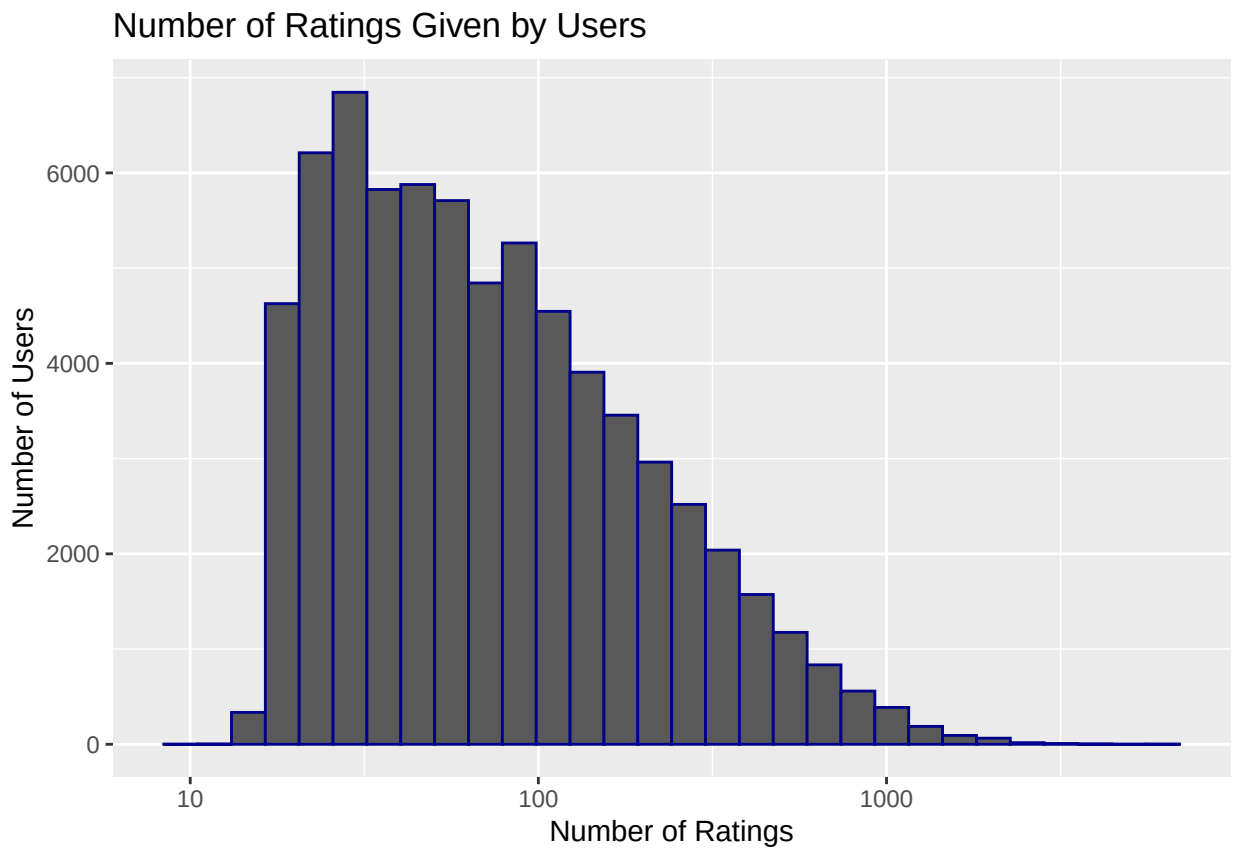
4.6 Number of Ratings per Movie Insights: A few popular movies receive thousands of ratings, while most movies have under 100, reflecting the "long tail" of movie popularity.

```
# Plotting number of ratings per movie
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "blue4") +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Number of Movies") +
  ggtitle("Number of Ratings per Movie")
```



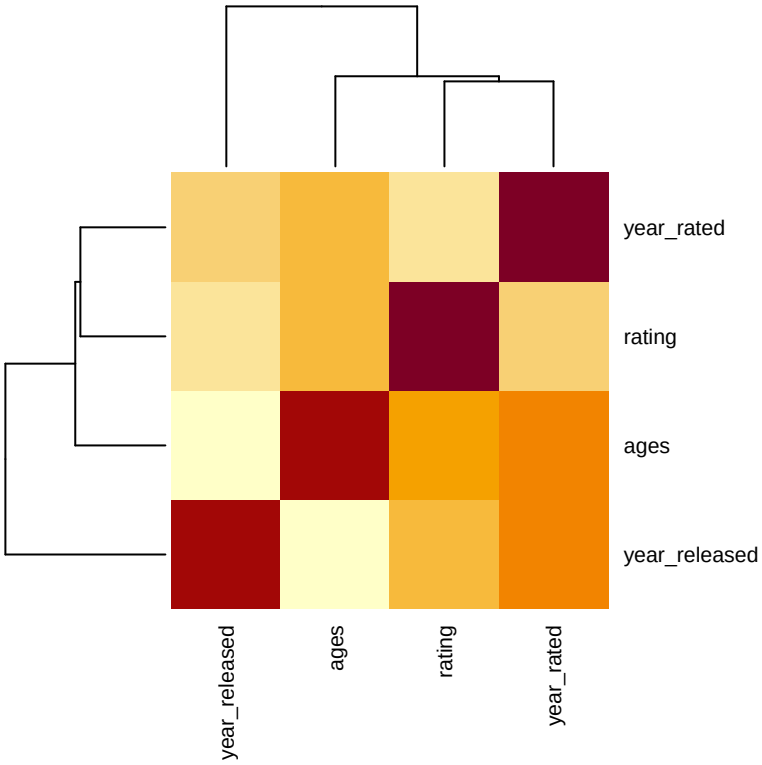
4.7 Number of Ratings Given by Users Insights: Most users rate fewer than 100 movies, but a few highly active users rate thousands, highlighting varied user engagement.

```
# Plotting number of ratings given by users
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "blue4") +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Number of Users") +
  ggtitle("Number of Ratings Given by Users")
```



4.8 Correlation Analysis Insights: There's a weak negative correlation between rating and movie age, suggesting older movies have slightly higher average ratings.

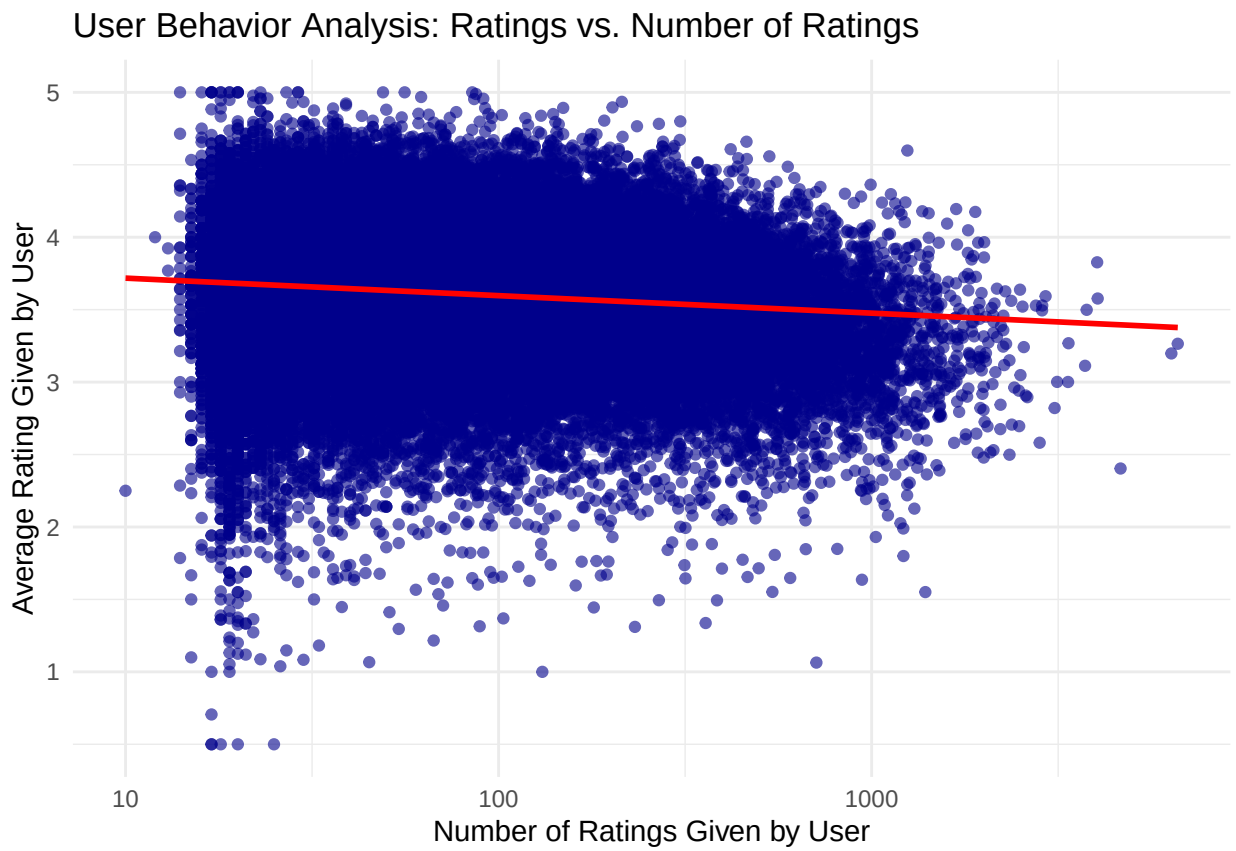
```
# Generating a correlation matrix
corr_matrix <- cor(edx %>%
                     select(rating, year_rated, year_released, ages),
                   use = "complete.obs")

# Heatmap of the correlation matrix
heatmap(corr_matrix,
          cellnote = round(corr_matrix, 2),
          notecol = "black",
          dendrogram = "none",
          trace = "none",
          margins = c(8, 8),
          cexRow = 0.8,
          cexCol = 0.8,
          key = TRUE,
          density.info = "none")
```



4.9 User Behavior Analysis

Insight: As users rate more movies, their average ratings tend to decrease, suggesting that more active users may adopt a more critical perspective over time or explore a broader range of movies, including lower-rated ones. This trend highlights how user engagement level can influence overall rating patterns, with high-frequency raters contributing to a slightly lower average rating across the data set.

```
#User Behavior Analysis: Ratings vs. Number of Ratings

# Calculate the number of ratings and average rating per user
user_behavior <- edx %>%
  group_by(userId) %>%
  summarise(num_ratings = n(), avg_rating = mean(rating))

# Create a scatter plot
ggplot(user_behavior, aes(x = num_ratings, y = avg_rating)) +
  geom_point(color = "blue4", alpha = 0.6) +
  geom_smooth(method = "lm", color = "red", se = FALSE) +
  xlab("Number of Ratings Given by User") +
  ylab("Average Rating Given by User") +
  ggtitle("User Behavior Analysis: Ratings vs. Number of Ratings") +
  scale_x_log10() +
  theme_minimal()
```



5. Modeling

We explore different models to predict user ratings.

5.1 Naive Model Insights: The Naive Model has the highest RMSE (1.06), which improves with each subsequent model.

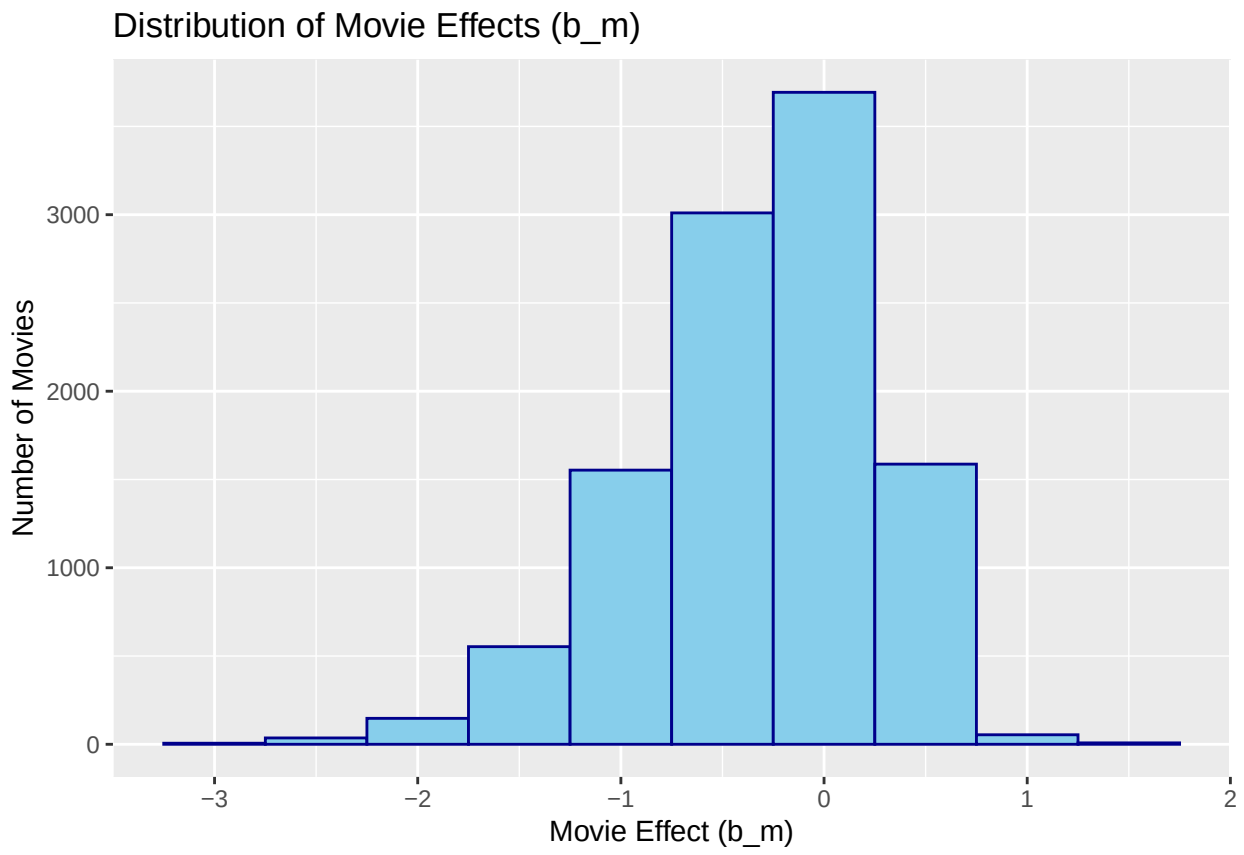
```
#First, let's split the edx dataset into training and test sets for model development. We'll reserve fi

# Split the edx data into training and test sets (80/20 split)
set.seed(1)  # For reproducibility
train_index <- createDataPartition(edx$rating, times = 1, p = 0.8, list = FALSE)
train_set <- edx[train_index, ]
test_set <- edx[-train_index, ]

# Ensure that the test set only contains users and movies from the training set
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

# Compute the mean rating (mu) from the training set
mu <- mean(train_set$rating)

# Test results based on simple prediction using the training/test split
```

```r
naive_rmse <- RMSE(test_set$rating, mu)

# Save prediction into data frame
rmse_results <- data_frame(method = "Naive Model", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.059735

5.2 Movie Effect Model Insights: This model improves upon the naive approach by accounting for each movie's specific average deviation from the overall mean rating, achieving a lower RMSE than the naive model. However, it does not yet consider user-specific preferences.

```r
movie_effects <- train_set %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mu))

# Plot the distribution of the movie effect (b_m)
movie_effects %>%
  ggplot(aes(x = b_m)) +
  geom_histogram(bins = 10, color = "blue4", fill = "skyblue") +
  labs(title = "Distribution of Movie Effects (b_m)",
       x = "Movie Effect (b_m)",
       y = "Number of Movies")
```



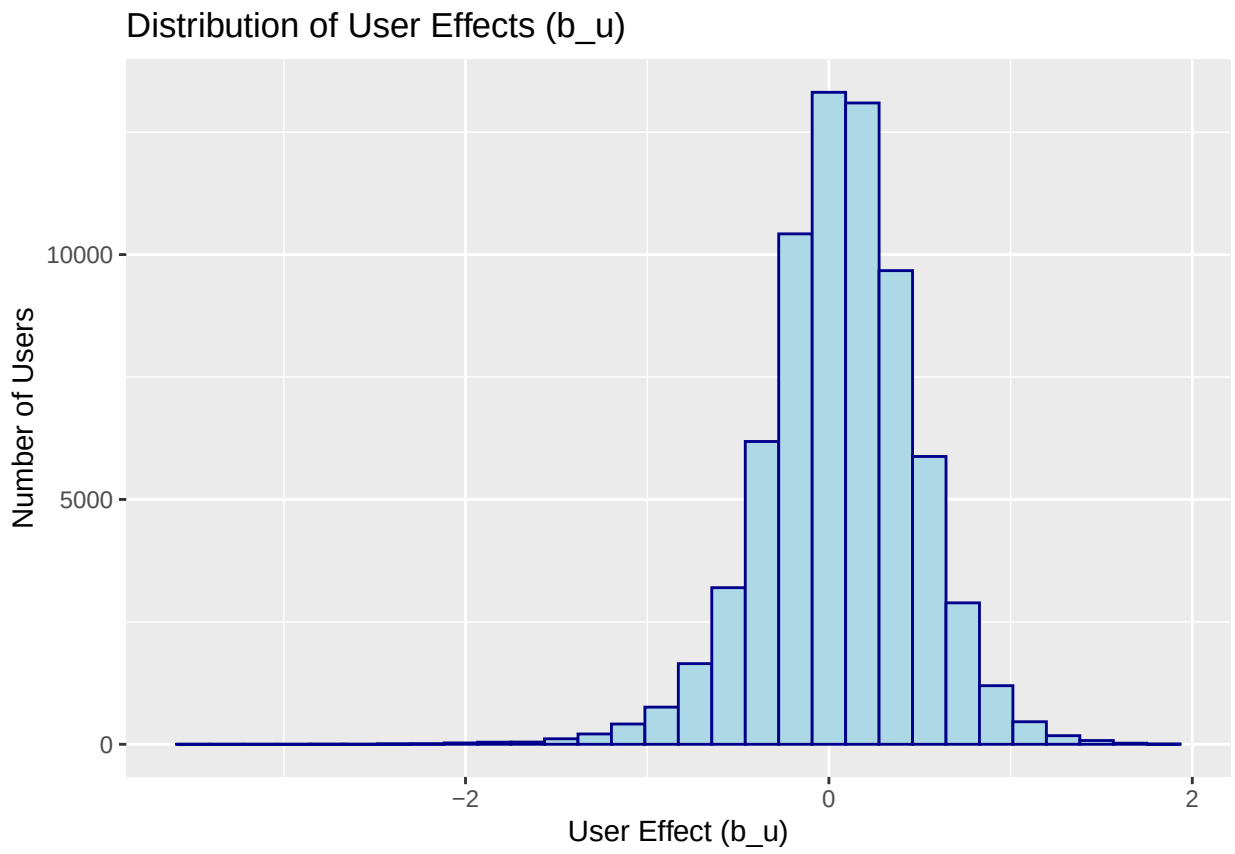
```r
# Test and save RMSE results on the test set
predicted_ratings <- test_set %>%
  left_join(movie_effects, by='movieId') %>%
  mutate(prediction = mu + b_m) %>%
  pull(prediction)

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Movie Effect Model",
                                     RMSE = model_1_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.059735
Movie Effect Model	0.943203

5.3 Movie + User Effect Model Insights: Adding user effects captures individual rating tendencies, reducing RMSE further by personalizing predictions. This model demonstrates that incorporating both movie and user influences significantly enhances predictive accuracy.

Notes: Regularization is a technique used in machine learning to prevent overfitting by adding a penalty to the model complexity, encouraging simpler models that generalize better to new data. This process balances the trade-off between training accuracy and the model's ability to make accurate predictions on unseen datasets.

```r
# User effect (b_u): Calculate the user effect from the training set
user_effects <- train_set %>%
  left_join(movie_effects, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_m))

# Plot the distribution of the user effect (b_u)
user_effects %>%
  ggplot(aes(x = b_u)) +
  geom_histogram(bins = 30, color = "blue4", fill = "lightblue") +
  labs(title = "Distribution of User Effects (b_u)",
       x = "User Effect (b_u)",
       y = "Number of Users")
```

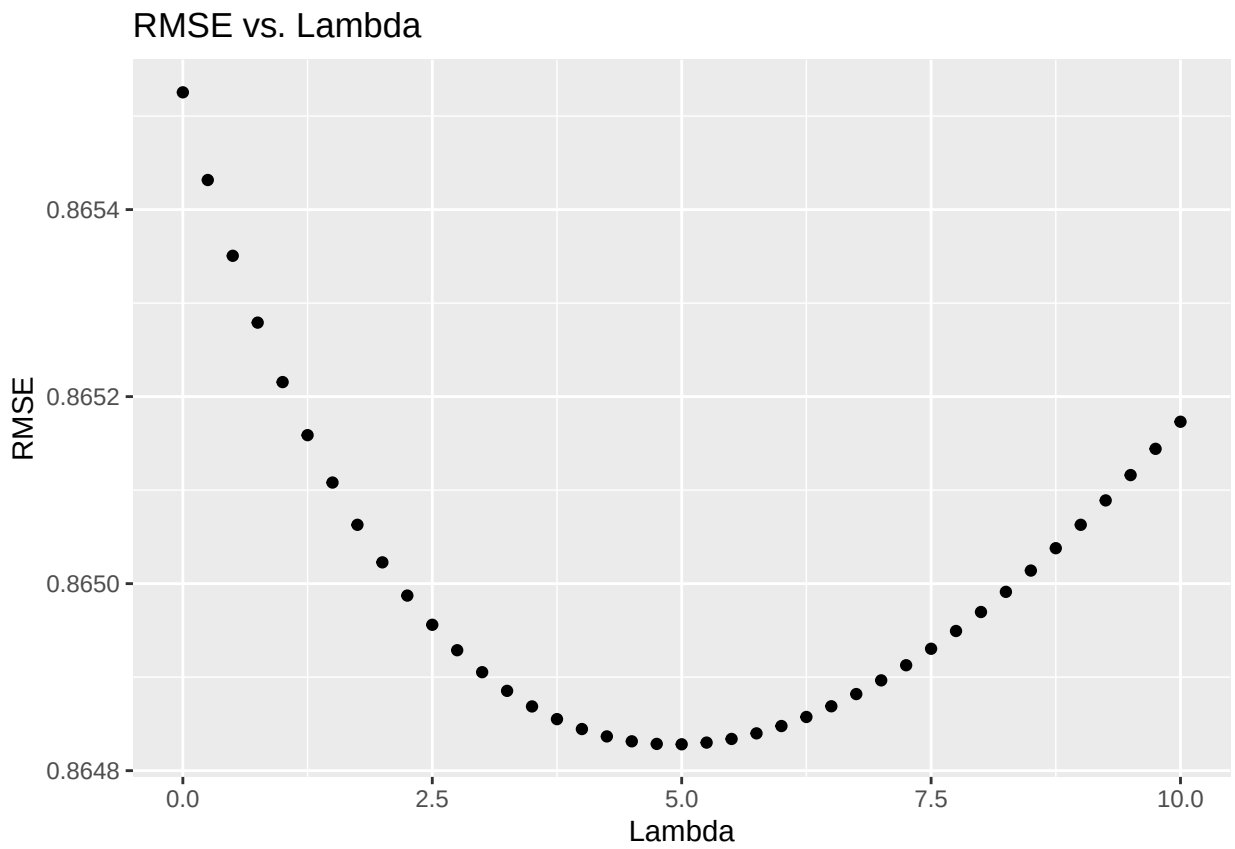



```
# Test and save RMSE results on the test set
predicted_ratings <- test_set %>%
  left_join(movie_effects, by='movieId') %>%
  left_join(user_effects, by='userId') %>%
  mutate(prediction = mu + b_m + b_u) %>%
  pull(prediction)

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Movie + User Effect Model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0597347
Movie Effect Model	0.9432030
Movie + User Effect Model	0.8655254

5.4. Regularized Movie + User Model Insights: Regularization reduces overfitting by penalizing extreme ratings, especially for movies or users with fewer ratings. This model achieves the lowest RMSE among the three, indicating that regularization improves the robustness and generalizability of the model.

```r
lambdas <- seq(0, 10, 0.25)

# Compute RMSE for each lambda
rmses <- sapply(lambdas, function(l){
  # Mean rating from the training set
  mu_reg <- mean(train_set$rating)

  # Regularized movie effect (b_m_reg)
  b_m_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m_reg = sum(rating - mu_reg) / (n() + !!l)) # Use !!l to pass lambda correctly

  # Regularized user effect (b_u_reg)
  b_u_reg <- train_set %>%
    left_join(b_m_reg, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg = sum(rating - b_m_reg - mu_reg) / (n() + !!l)) # Use !!l to pass lambda correctly

  # Predict ratings for the test set
  predicted_ratings <- test_set %>%
    left_join(b_m_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    mutate(prediction = mu_reg + b_m_reg + b_u_reg) %>%
    pull(prediction)

  # Compute RMSE
  return(RMSE(test_set$rating, predicted_ratings))
})

# Plot RMSE against lambdas
qplot(lambdas, rmses) +
  labs(title = "RMSE vs. Lambda",
       x = "Lambda",
       y = "RMSE")
```




```
# Find the optimal lambda (minimizing RMSE)
lambda_optimal <- lambdas[which.min(rmses)]
lambda_optimal
```

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

```
# Calculate RMSE for the optimal lambda
model_m_u_reg_rmse <- min(rmses)
model_m_u_reg_rmse
```

```
## [1] 0.8648282
```

```
# Save RMSE results
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Regularized Movie + User Effect Model",
                                     RMSE = model_m_u_reg_rmse))
```

Results

Here are the RMSE results for all models.

```
# RMSE Results Summary
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0597347
Movie Effect Model	0.9432030
Movie + User Effect Model	0.8655254
Regularized Movie + User Effect Model	0.8648282

6. Model evaluation and final model fitting on final_holdout_test dataset.

Insights: The final model regularized movie + user effect model, with an RMSE of 0.8648, successfully accounts for both movie and user effects, enhancing the prediction accuracy for user movie ratings.

```
#####################################
# Final Model Fitting
#####################################

# Compute the mean rating on the full edx dataset
mu_final <- mean(edx$rating)

# Regularized movie effects on the full edx dataset
b_m_final <- edx %>%
  group_by(movieId) %>%
  summarize(b_m_final = sum(rating - mu_final) / (n() + lambda_optimal))

# Regularized user effects on the full edx dataset
b_u_final <- edx %>%
  left_join(b_m_final, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u_final = sum(rating - b_m_final - mu_final) / (n() + lambda_optimal))

# Final predictions on the final_holdout_test dataset
final_predicted_ratings <- final_holdout_test %>%
  left_join(b_m_final, by = "movieId") %>%
  left_join(b_u_final, by = "userId") %>%
  mutate(final_prediction = mu_final + b_m_final + b_u_final) %>%
  pull(final_prediction)

# Calculate final RMSE using final_holdout_test
final_rmse <- RMSE(final_holdout_test$rating, final_predicted_ratings)

# Add final RMSE to results
rmse_results <- bind_rows(rmse_results,
                          data_frame(method = "Final Model (Regularization)",
                                     RMSE = final_rmse))

# Display final RMSE results
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0597347
Movie Effect Model	0.9432030
Movie + User Effect Model	0.8655254

method	RMSE
Regularized Movie + User Effect Model	0.8648282
Final Model (Regularization)	0.8648177

7. Conclusion

In this report, we explored and modeled the MovieLens dataset, applying various techniques to predict user ratings. The regularized movie + user effect model performed the best, demonstrating the importance of accounting for both movie and user preferences in recommendation systems.

8. Future work

8.1 Genre Analysis: Add genre-specific effects to improve model accuracy, as genre preferences significantly impact ratings. Analyze genre-specific trends to uncover popular genres and their rating patterns across user demographics.

8.2 Temporal Dynamics: Incorporate time-based effects to capture shifts in user preferences and movie popularity over years. Examine seasonal release effects, such as holidays vs. off-season, to refine time-based predictions.

8.3 User Segmentation: Cluster users by rating behavior and genre preferences to enhance personalization in recommendations. Address the cold-start problem by leveraging genre and demographic data for new users or movies.

8.4 Enhanced Regularization Techniques: Apply Bayesian regularization to improve predictions for users or movies with limited ratings. Experiment with automated hyper parameter tuning to optimize regularization strength and model performance.

8.5 Collaborative Filtering: Integrate collaborative filtering to enhance predictions by identifying similarities between users or items. Explore hybrid models that combine collaborative and content-based filtering for improved accuracy.

8.6 Sentiment Analysis on Reviews: Use sentiment analysis on reviews (if available) to correlate user sentiment with ratings, adding predictive insights.

9. Thanks!

I am incredibly grateful to Professor Rafael Irizarry for his dedication and expertise in the HarvardX Data Science Program, which has made this journey both inspiring and deeply rewarding. I would also like to thank my peers, whose thoughtful discussions and insights have enriched my learning experience. A special thanks goes to my husband, Abhi, and my two young sons, Kian and Evan, whose boundless energy and love have been a constant source of joy and motivation, even during the most challenging moments. Thank you all for being my foundation and my greatest inspiration.

10. References

1) I had to upgrade the R version in between while writing the code and I have used chatgpt for debugging while fitting correct version of libraries and make my code work again.
2) Also, I looked up at various github projects to find the right documentation structure for my project.
3) I have also used the knowledge from another data science bootcamp (springboard) in EDA portion to know this data better.