Movie recommender system with R

Nevil Abraham Elias

03/07/2021

Abstract

This report was prepared as part of the capstone project for HarvardX's Data Science Professional Certificate Program.

Contents

1 Introduction

Recommender systems are specialised systems or algorithms that suggest relevant items or content to users. Examples of such systems can be seen in e-commerce and online streaming services, such as YouTube and Amazon. The goal of recommender systems is to help users discover desired products, based on their preferences and previous interactions, and estimate the quality of a new product.

In this document, we create a movie recommender system based on the movielens dataset, using the prediction techniques learned in the HarvardX Data Science Professional Certificate Program.

This document is structured as follows:
Chapter 1 - Introduction - Dataset, goal of the project and key steps involved.
Chapter 2 - Methods/Analysis - Data cleaning, exploration, visualisation.
Chapter 3 - Results - Modelling performance and results.
Chapter 4 - Conclusion - Summary, limitations and future works.

1.1 MovieLens Dataset

The Movielens 10M dataset contains 10 million ratings made by 69878 users on 10677 movies. It is the subset of a larger 27 million dataset obtained from the Movielens website. The website is run by GroupLens, a research lab at the university of Minnesota.

1.2 Goal

The goal of this project is to create a movie recommender system that accurately predicts the rating given by a user for a movie. This involves the use of a metric to evaluate the model. Typically used metrics for machine learning model evaluation include Classification Accuracy, Area Under Curve (AUC), Root Mean Squared Error etc. For this project, RMSE is used as the evaluation metric.

Thus, the project aims at creating a recommender system with an **RMSE lower than 0.8649**.

Root Mean Squared Error is the square root of the average squared distance between the actual outcomes and the predicted values.

The RMSE is defined as:

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

where $y_{u,i}$ is the observed value for observation i and $\hat{y}_{u,i}$ is the predicted value.

We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1

1.3 Process

Key steps involved in this project include:

1. Project Understanding: understanding the project's goals and creating a workflow of key steps.
2. Data Preparation: downloading, importing and preparing the dataset for analysis.
3. Data Exploration: to gain insights from the data and identify key features or predictors.
4. Data Preprocessing: involves cleaning and transforming the data, such as feature selection, scaling, removing unnecessary information, etc.

5. Modelling Methods: researching and selecting modelling approaches that work best for the type of dataset.
6. Data Modelling: Creating, training and testing the selected models, including any fine tuning of parameters.
7. Model Evaluation: Evaluating the results and model's performance.
8. Communication: Present the final results along with any limitations or recommendations for future work.

For any machine learning project, it is essential that the model performs well for both the available data and the real-world data. To ensure this, the dataset is initially split into two, a training set and a validation set. Steps 3 through 7 are performed on the training set(by further subdividing into train and test sets) to select a model. This model is then trained on the entire training set and evaluated using the validation set.

2 Methods/Analysis

2.1 Data Preparation

As previously mentioned this step covers the set of activites from downloading to making the dataset ready for analysis. We start off by loading the necessary libraries.

```
# Load libraries
library(tidyverse)
library(ggthemes)
library(data.table)
library(knitr) #A General-Purpose Package for Dynamic Report Generation in R
library(kableExtra)
library(lubridate)
library(tinytex)
library(latexpdf)
# set global options
options(timeout=10000, digits=5)
```

We download and import the original dataset using the URL link where it is stored and then prepare the dataset for analysis by first splitting the original dataset into two parts: edx which contains 90% of the dataset and validation (final hold-out test set) which is 10% of the dataset.

```
#Source file
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")

# Splitting the data
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

We further split the edx dataset into two parts: 90% for the training set, which we use to create and train our models, and 10% for the testing set, which we use to test our models.

```
# Split data into training and test sets - test set will be 10% of edx
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
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)
rm(test_index, temp, removed)
```

2.2 Data Exploration

In this section, Exploratory Data Analyses are performed on the dataset to understand the data select features for modelling.

```
#Initial exploration
class(edx)

# [1] "data.table" "data.frame"

str(edx)

# Classes 'data.table' and 'data.frame':    9000055 obs. of  6 variables:
#  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
#  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
#  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
#  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 838
#  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
#  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
#  - attr(*, ".internal.selfref")=<externalptr>
```

```
head(edx, 10)
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy
1	356	5	838983653	Forrest Gump (1994)	Comedy|Drama|Romance|War
1	362	5	838984885	Jungle Book, The (1994)	Adventure|Children|Romance
1	364	5	838983707	Lion King, The (1994)	Adventure|Animation|Children|Dr
1	370	5	838984596	Naked Gun 33 1/3: The Final Insult (1994)	Action|Comedy

From this initial exploration, we learn that the dataset consists of 9000055 rows and 6 columns. Users and movies are uniquely identified using the respective columns *userId* and *movieId*. The column *title* shows the name of a movie with its release year and *genres* lists all the assosciated genres of the movie. The users rating for a movie and the time of that rating are provided by the columns *rating* and *timestamp* respectively. The rows appear to be ordered by *userId* and then *movieId*.

Now we move on to detailed exploration of each columns.

2.2.1 rating

As previously mentioned, this column contains the user rating for a particular movie. This is the variable that our recommender system tries to predict.

```
unique(edx$rating)
```

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

```
table(edx$rating)
```

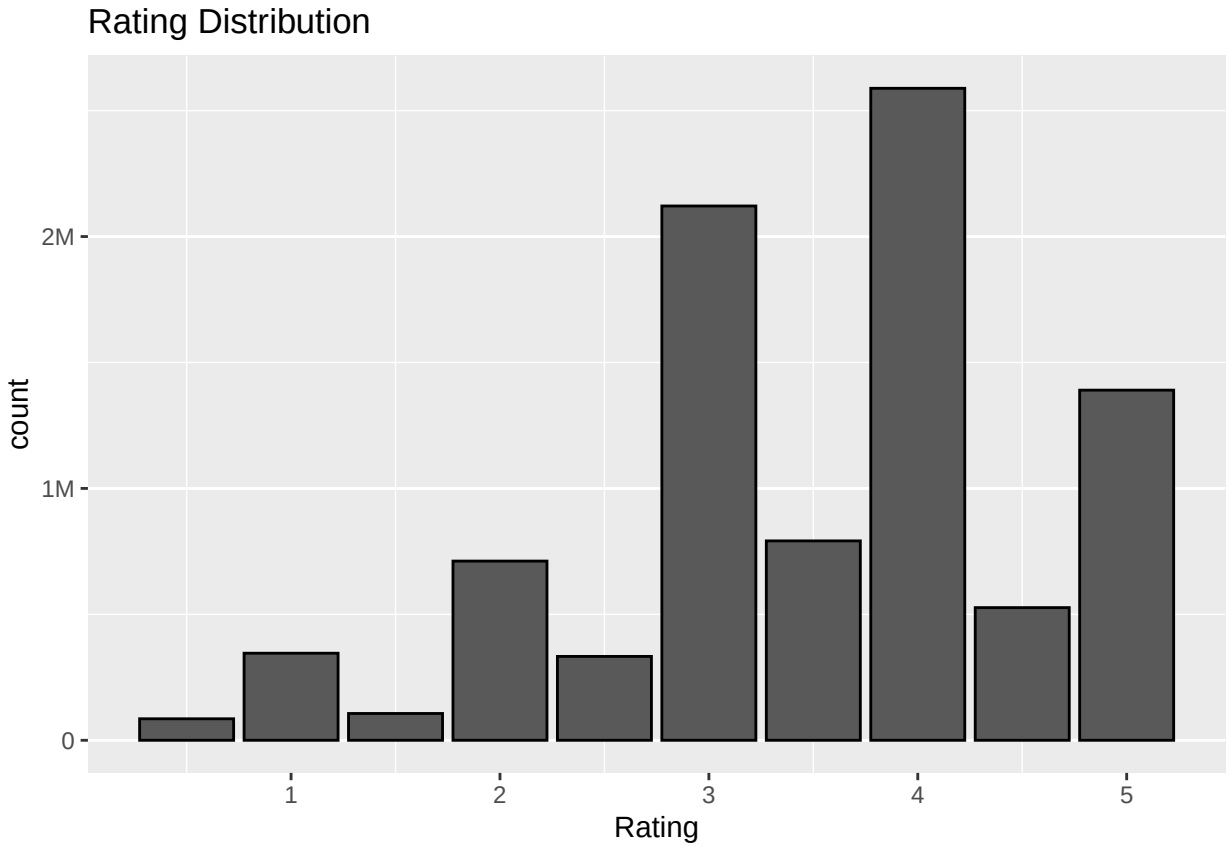
```
#
#     0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
#   85374  345679  106426  711422  333010 2121240  791624 2588430  526736 1390114
```

```
summary(edx$rating)
```

```
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#    0.50    3.00    4.00    3.51    4.00    5.00
```

Now let us plot the distribution of *rating*. Since this is a discrete variable, we will use a bar chart to visualize the distribution.

```
#Plotting rating distribution
edx %>% ggplot(aes(rating)) +
geom_bar(col = "black") +
ggtitle("Rating Distribution") +
xlab("Rating") +
scale_y_continuous(breaks = seq(0,3*10^6,10^6),
                   labels=c("0","1M","2M","3M"))+
theme(panel.grid.major.x = element_blank())
```



| From the plot we can observe two things:- 1. Most movie ratings are positive (i.e ratings are greater than the median rating value 2.5) 2. Whole star ratings are more common than half star ratings.

```
# Comparing rating types
edx %>%
  mutate(rating_type = if_else(rating > 2.5, "postitive",
                                               "negative")) %>%
  group_by(rating_type) %>%
  count()

# # A tibble: 2 x 2
# # Groups:   rating_type [2]
#   rating_type       n
#   <chr>         <int>
# 1 negative    1581911
# 2 postitive   7418144

#Comparing half star and whole star ratings
edx %>%
  mutate(rating_star = if_else(!rating %%1, "whole_star",
                                             "half_star")) %>%
  group_by(rating_star) %>%
  count()

# # A tibble: 2 x 2
# # Groups:   rating_star [2]
```

```
#   rating_star       n
#   <chr>         <int>
# 1 half_star   1843170
# 2 whole_star  7156885
```

2.2.2 userId

userId uniquely identifies a user. Here we are trying to explore user related data like the number of movies rated by a user and average rating of a user

```
# No. of unique users
n_distinct(edx$userId)
```

```
# [1] 69878
```

```
# Create a data frame for user data
user_df <- edx %>%
              group_by(userId) %>%
              summarise(n = n(), user_avg = mean(rating))
head(user_df)
```

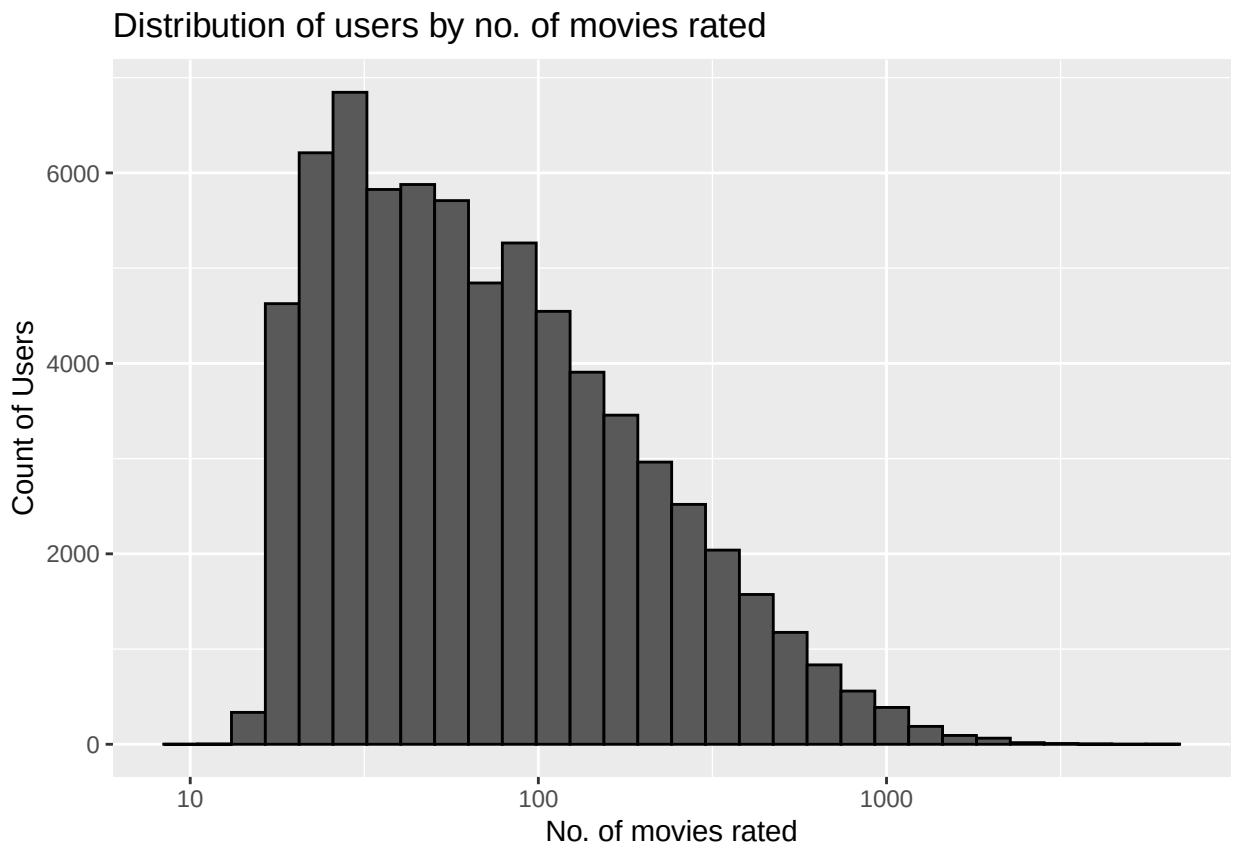
```
# # A tibble: 6 x 3
#   userId     n user_avg
#    <int> <int>    <dbl>
# 1      1    19     5
# 2      2    17     3.29
# 3      3    31     3.94
# 4      4    35     4.06
# 5      5    74     3.92
# 6      6    39     3.95
```

```
summary(user_df[,c("n","user_avg")])
```

```
#        n           user_avg
#  Min.   :  10   Min.   :0.50
#  1st Qu.:  32   1st Qu.:3.36
#  Median :  62   Median :3.63
#  Mean   : 129   Mean   :3.61
#  3rd Qu.: 141   3rd Qu.:3.90
#  Max.   :6616   Max.   :5.00
```

From the summary of the number of movies rated (n), we can intuitively guess that the distribution is log skewed. To confirm this we will plot the distribution.

```
user_df %>%
  ggplot(aes(n)) +
  geom_histogram(col = "black") +
  scale_x_log10() +
  ggtitle("Distribution of users by no. of movies rated") +
  xlab("No. of movies rated") +
  ylab("Count of Users")
```



| This plot verifies our assumption.

We previously observed that most ratings are positive ratings. The mean rating of users will help us identify the user biases.

```
# Plotting distribution of user mean
user_df %>%
  ggplot(aes(user_avg)) +
  geom_histogram(bins = 30,col = "black") +
  geom_vline(xintercept = mean(edx$rating), col = "yellow") +
  ggtitle("Distribution of users by mean rating of users") +
  ylab("Count of users")
```

The distribution shows what is known as the user bias or user effect. Bias explains the variation in ratings of a movie by two users who equally liked or disliked the movie.

One last thing worth exploring is how users rate in terms of stars, ie half-star or whole star. This can be obtained as follows

```
# Finding distribution of users by star type
edx %>%
  mutate(rating_star = ifelse(!rating %%1,
                              "whole_star", "half_star")) %>%
  group_by(rating_star) %>%
  summarise(n_users = n_distinct(userId))

# # A tibble: 2 x 2
```

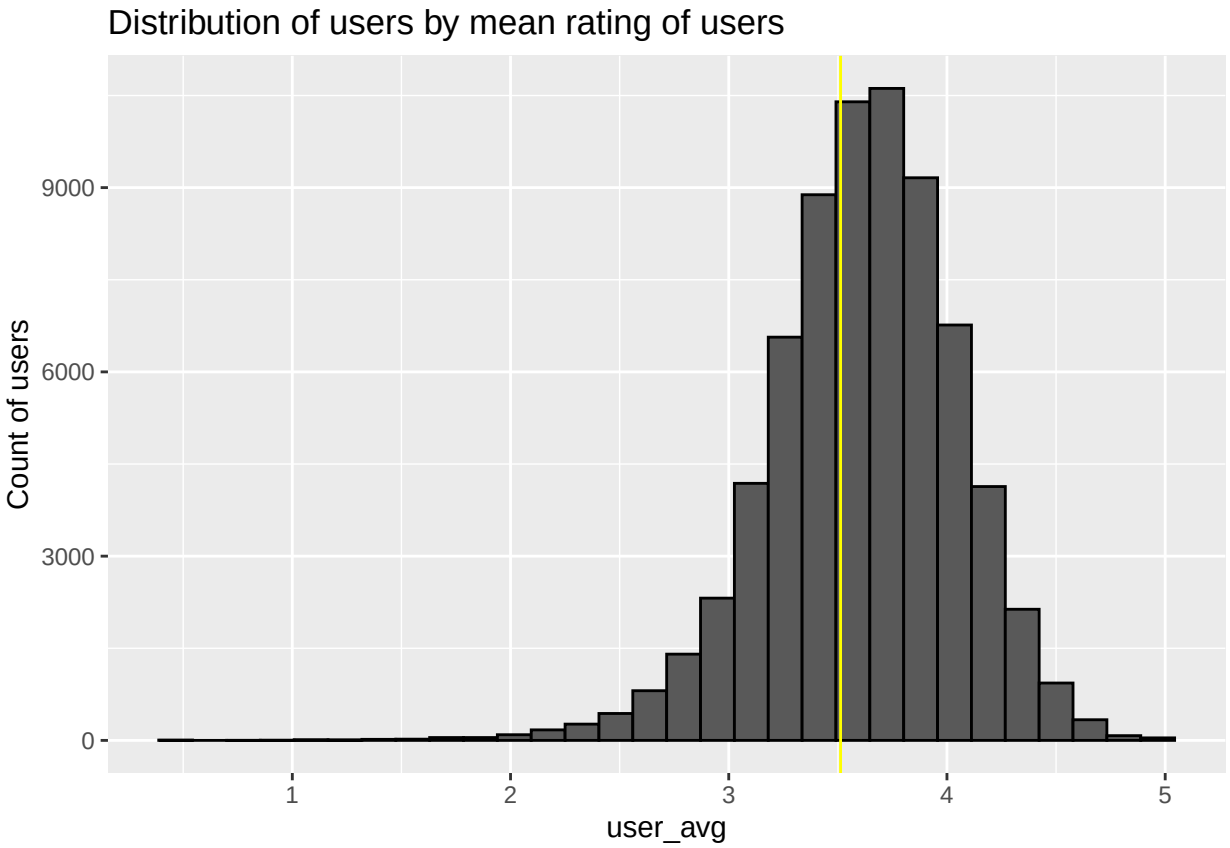


Figure 1: Distribution of user mean

```
#   rating_star n_users
#   <chr>         <int>
# 1 half_star     25115
# 2 whole_star    69872
```

This tells that only a little more than one third of the users rate in half stars. All other users rate in whole star or integers.

2.2.3 movieId

movieId uniquely identifies the movies in the dataset. For this column we perform analyses similar too *userId*. Let us first create a separate data frame for movie data. Since the column *title* is also related to movies, we'll add that column too.

```
# Creating a movie data frame
movie_df <- edx %>%
  group_by(movieId,title) %>%
  summarise(n = n(), movie_avg = mean(rating),
                           se = sd(rating)/sqrt(n))
head(movie_df,10)
```

```
# # A tibble: 10 x 5
# # Groups:   movieId [10]
#    movieId title                                n movie_avg      se
#      <int> <chr>                              <int>     <dbl>   <dbl>
#  1       1 Toy Story (1995)                   23790      3.93 0.00582
#  2       2 Jumanji (1995)                     10779      3.21 0.00916
#  3       3 Grumpier Old Men (1995)             7028      3.15 0.0119
#  4       4 Waiting to Exhale (1995)            1577      2.86 0.0275
#  5       5 Father of the Bride Part II (1995)  6400      3.07 0.0120
#  6       6 Heat (1995)                        12346      3.82 0.00797
#  7       7 Sabrina (1995)                      7259      3.36 0.0112
#  8       8 Tom and Huck (1995)                  821      3.13 0.0342
#  9       9 Sudden Death (1995)                 2278      3.00 0.0203
# 10      10 GoldenEye (1995)                   15187      3.43 0.00701
```

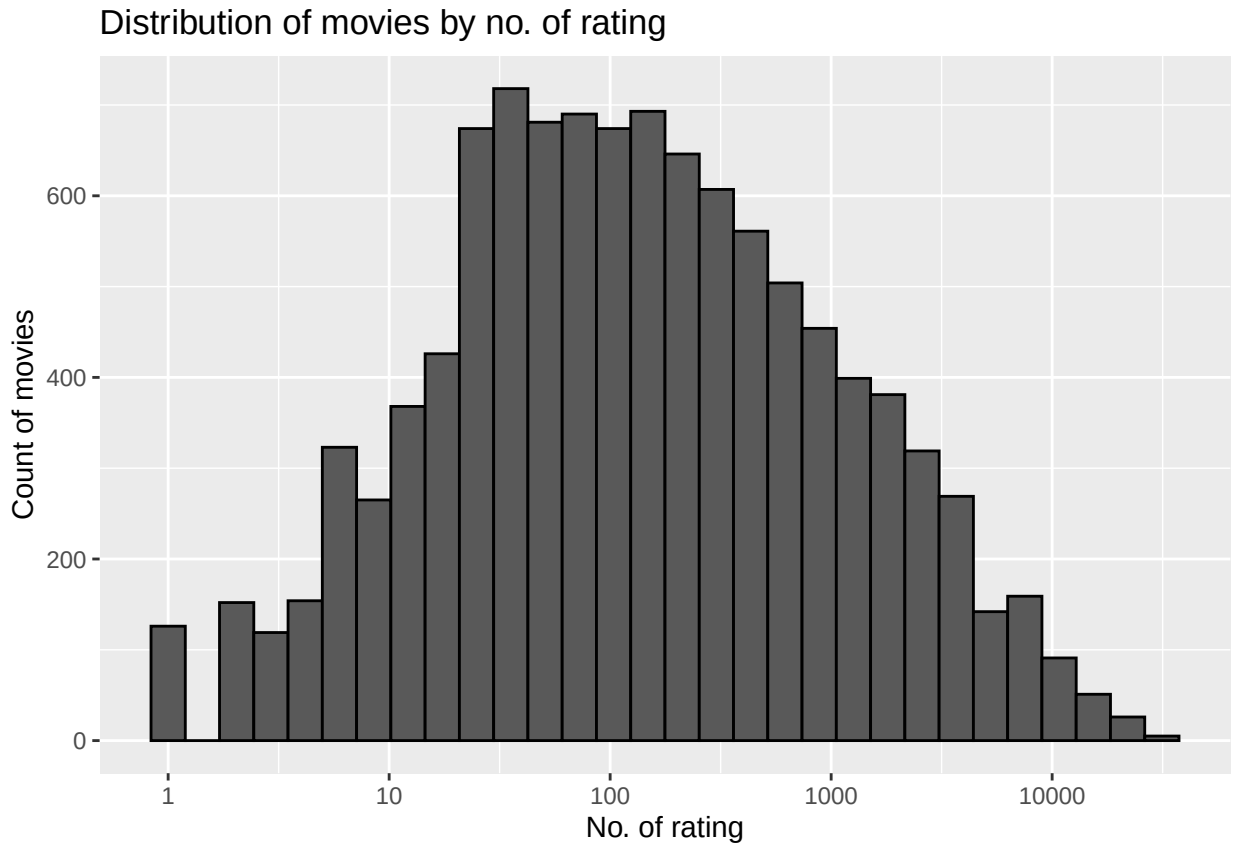
```
summary(movie_df[c("n", "movie_avg", "se")])
```

```
#        n           movie_avg          se
#  Min.   :    1   Min.   :0.50   Min.   :0.00
#  1st Qu.:   30   1st Qu.:2.84   1st Qu.:0.04
#  Median :  122   Median :3.27   Median :0.08
#  Mean   :  843   Mean   :3.19   Mean   :0.12
#  3rd Qu.:  565   3rd Qu.:3.61   3rd Qu.:0.16
#  Max.   :31362   Max.   :5.00   Max.   :1.75
#                                 NA's   :126
```

The distribution of n can be visualized as follows

```
movie_df %>%
  ggplot(aes(n)) +
```

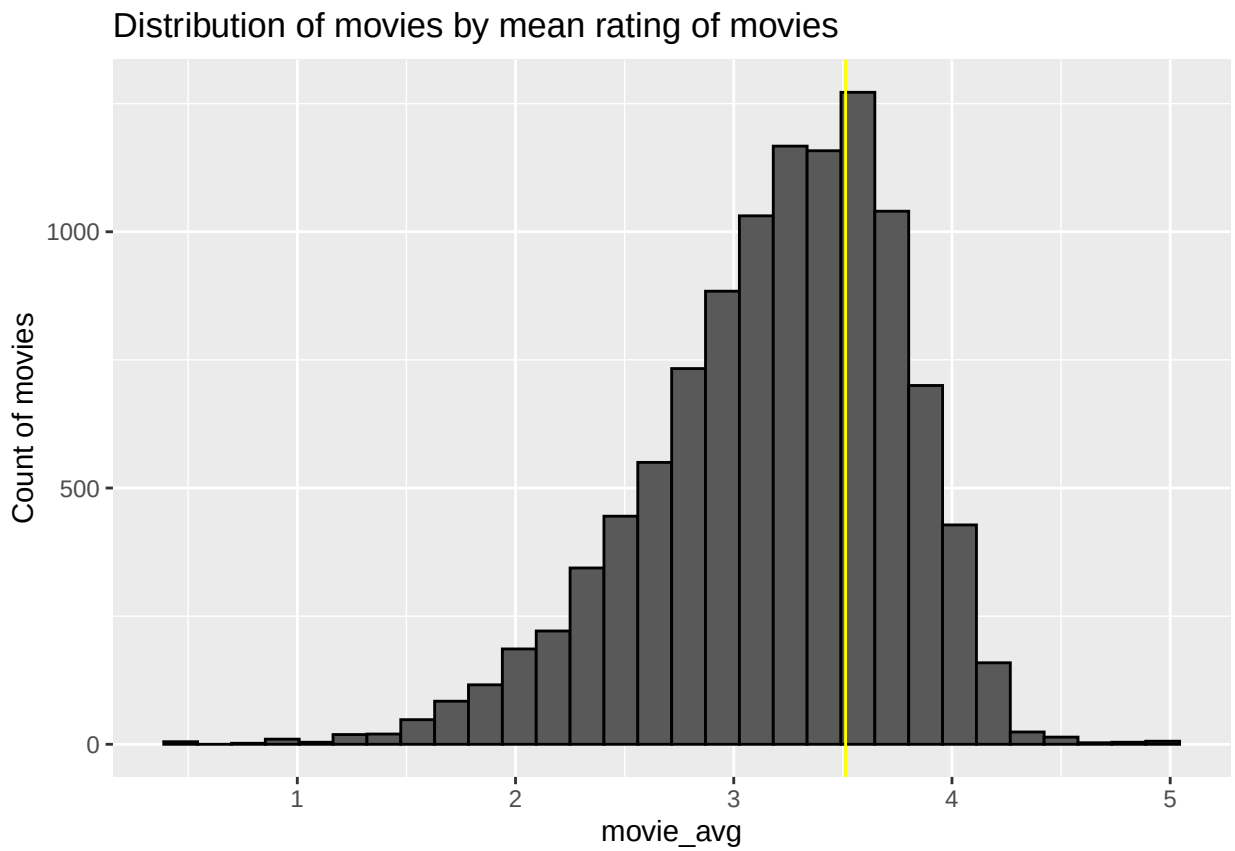
```
  geom_histogram(col = "black") +
  scale_x_log10() +
  ggtitle("Distribution of movies by no. of rating") +
  xlab("No. of rating") +
  ylab("Count of movies")
```



| As before, the distribution is log skewed.

Now let's visualize the average rating of movies.

```
# Plotting movie distribution by no. of rating
movie_df %>%
  ggplot(aes(movie_avg)) +
  geom_histogram(bins = 30,col = "black") +
  geom_vline(xintercept = mean(edx$rating), col = "yellow") +
  ggtitle("Distribution of movies by mean rating of movies") +
  ylab("Count of movies")
```



| This plot describes what is called as movie effect or movie bias. This explains why some movies tend to be rated higher than other movies. Good storyline, direction, performance etc are some of the factors which contribute towards positive movie effect while the lack of these contribute towards negative movie effect. Some good or poor movies tend to have zero or neutral effect because of their polarising nature. These effects will be explored later in the *title* section.

2.2.4 title

This column contains the names of all movies and their release years. We can utilise the dataframe created in the last section for the analysis of this column. Let us first separate the release year into a new column.

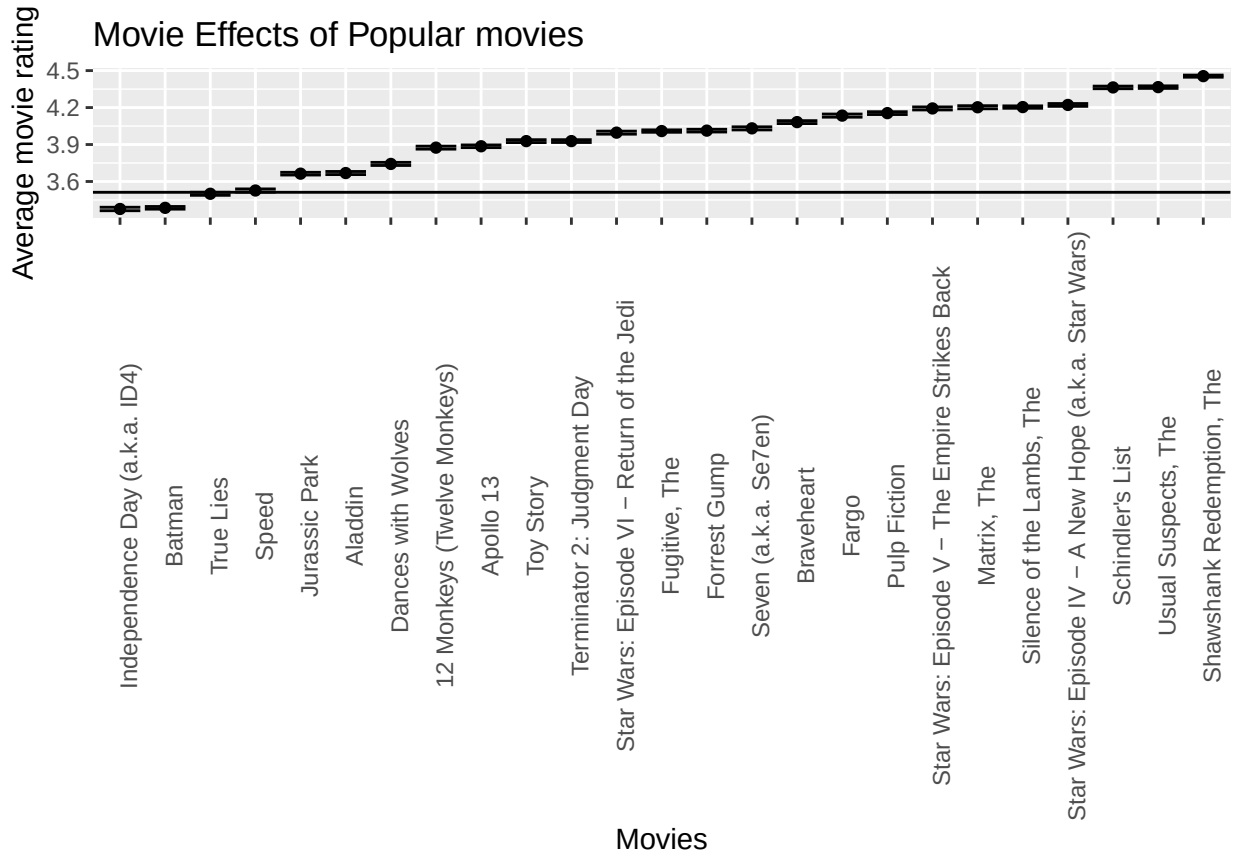
```
#Creating a separate column for release year
movie_df <- movie_df %>%
  extract("title",c("title","release_year"),
                                "(.*) \\((\\d{4})\\)$") %>%
  mutate(release_year = as.integer(release_year))
head(movie_df,10)
```

```
# # A tibble: 10 x 6
# # Groups:   movieId [10]
#    movieId title                     release_year     n movie_avg      se
#      <int> <chr>                            <int> <int>     <dbl>   <dbl>
#  1       1 Toy Story                         1995 23790      3.93 0.00582
#  2       2 Jumanji                           1995 10779      3.21 0.00916
#  3       3 Grumpier Old Men                  1995  7028      3.15 0.0119
#  4       4 Waiting to Exhale                 1995  1577      2.86 0.0275
```

```
#  5       5 Father of the Bride Part II         1995  6400      3.07 0.0120
#  6       6 Heat                                1995 12346      3.82 0.00797
#  7       7 Sabrina                             1995  7259      3.36 0.0112
#  8       8 Tom and Huck                        1995   821      3.13 0.0342
#  9       9 Sudden Death                        1995  2278      3.00 0.0203
# 10      10 GoldenEye                           1995 15187      3.43 0.00701
```

We'll start here with the movie titles. First, let's find out the most popular movies.

```
movie_df %>%
  dplyr::filter(n>20000) %>%
  ggplot(aes(x = reorder(title,movie_avg),y = movie_avg,
             ymin = movie_avg - 2*se,
             ymax = movie_avg + 2*se))+
  geom_point() +
  geom_errorbar() +
  geom_hline(yintercept = mean(edx$rating)) +
  ggtitle("Movie Effects of Popular movies") +
  xlab("Movies") +
  ylab("Average movie rating") +
  theme(axis.text.x = element_text(angle = 90))
```




```
#ordering movie_df by no.of ratings
movie_df <- movie_df %>% arrange(-n)
head(movie_df,10)
```

```
# # A tibble: 10 x 6
# # Groups:   movieId [10]
#    movieId title                              release_year     n movie_avg      se
#      <int> <chr>                                     <int> <int>     <dbl>   <dbl>
#  1     296 Pulp Fiction                             1994 31362      4.15 0.00567
#  2     356 Forrest Gump                             1994 31079      4.01 0.00551
#  3     593 Silence of the Lambs, The                1991 30382      4.20 0.00482
#  4     480 Jurassic Park                            1993 29360      3.66 0.00547
#  5     318 Shawshank Redemption, The                1994 28015      4.46 0.00428
#  6     110 Braveheart                               1995 26212      4.08 0.00588
#  7     457 Fugitive, The                            1993 25998      4.01 0.00482
#  8     589 Terminator 2: Judgment Day               1991 25984      3.93 0.00562
#  9     260 Star Wars: Episode IV - A New H~         1977 25672      4.22 0.00570
# 10     150 Apollo 13                                1995 24284      3.89 0.00547
```

```
#Plotting the top 10 popular movies
movie_df %>% head(10) %>%
  ggplot(aes(n,reorder(title,n))) +
  geom_col() +
  ggtitle("Most popular movies") +
  xlab("No. of ratings") +
  ylab("Movie Title")
```

Now we will move on to the analyses of the release years.

```
n_distinct(movie_df$release_year)
```

```
# [1] 94
```

```
summary(movie_df$release_year)
```

```
#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#    1915    1979    1994    1987    2001    2008
```

It is clear that each year has atleast one movie release and with passing years more movies have been released. Let's visualise this.

```
movie_df %>% ggplot(aes(release_year)) +
  geom_density(col = "blue") +
  geom_histogram(binwidth = 3,col="black")+
  ggtitle("Distribution of movies by release year") +
  xlab("Release Year") +
  ylab("No. of movies")
```

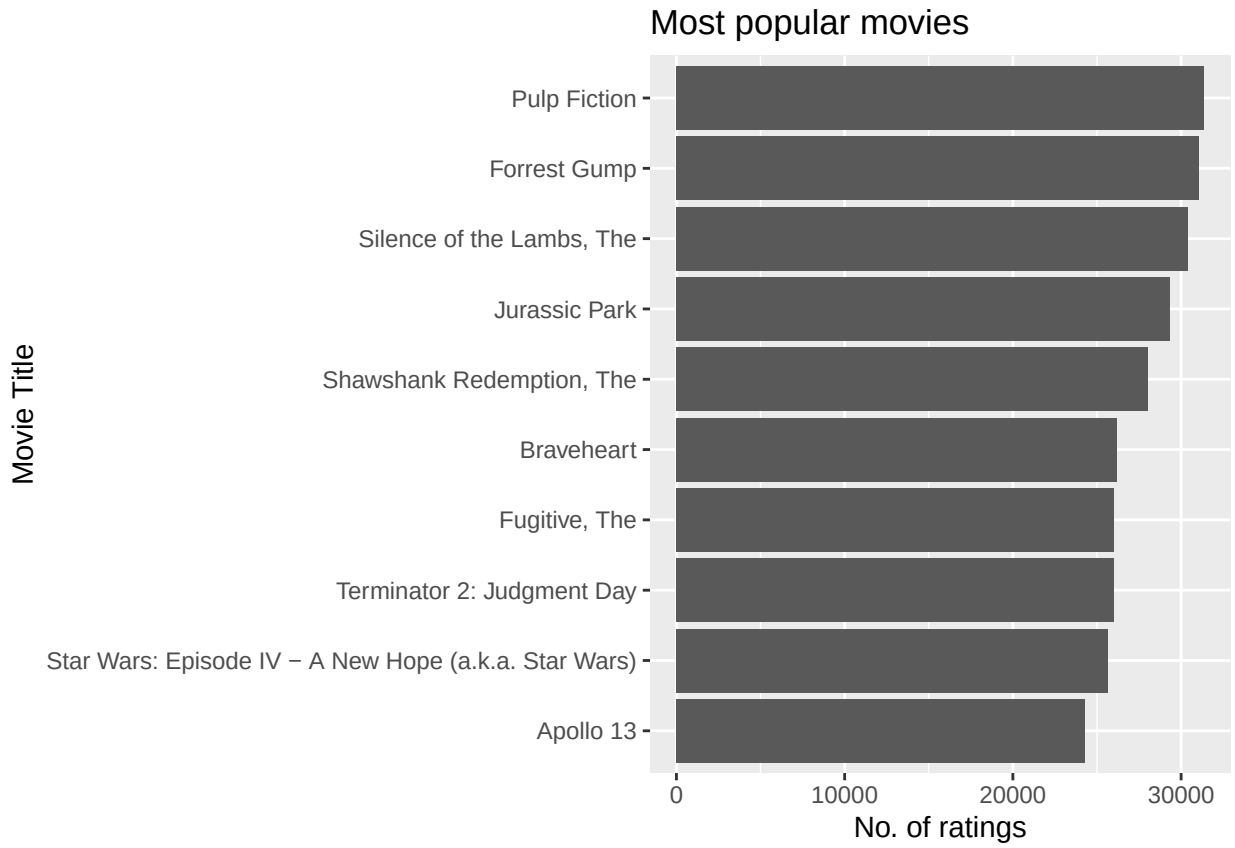
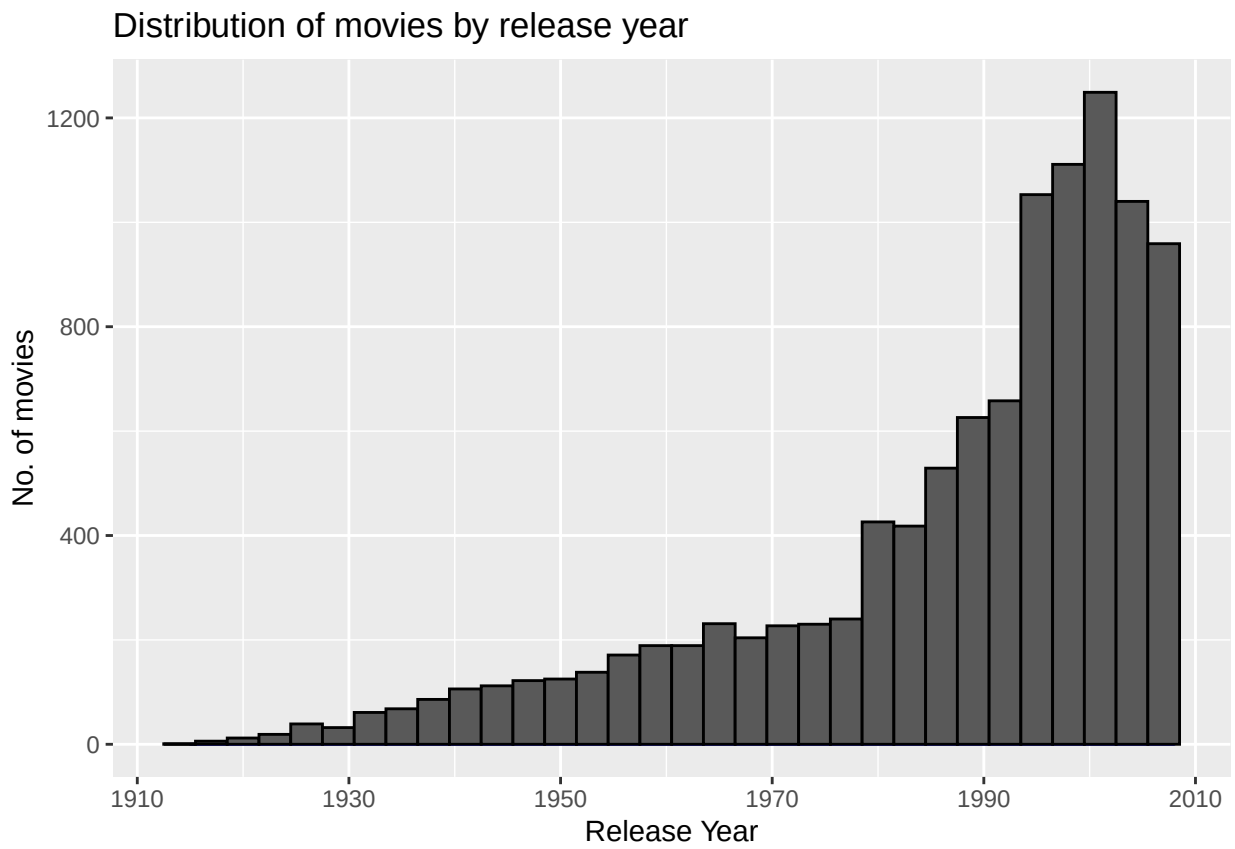


Figure 2: Most popular movies



genres | *genres* list all the assosciated genres of a movie. Let's start our analysis.

```
#Total no.of genre combinations
n_distinct(edx$genres)
```

```
# [1] 797
```

```
#Determining the no. of individual genres
unique(edx$genres) %>% #returns the subset that does not
  str_subset(pattern = "\\|",negate = T) #contain the symbol |
```

```
#  [1] "Comedy"             "Drama"              "Thriller"
#  [4] "Western"            "Horror"             "Documentary"
#  [7] "Action"             "Romance"            "Sci-Fi"
# [10] "Children"           "Adventure"          "Animation"
# [13] "Musical"            "Film-Noir"          "Crime"
# [16] "War"                "Mystery"            "Fantasy"
# [19] "IMAX"               "(no genres listed)"
```

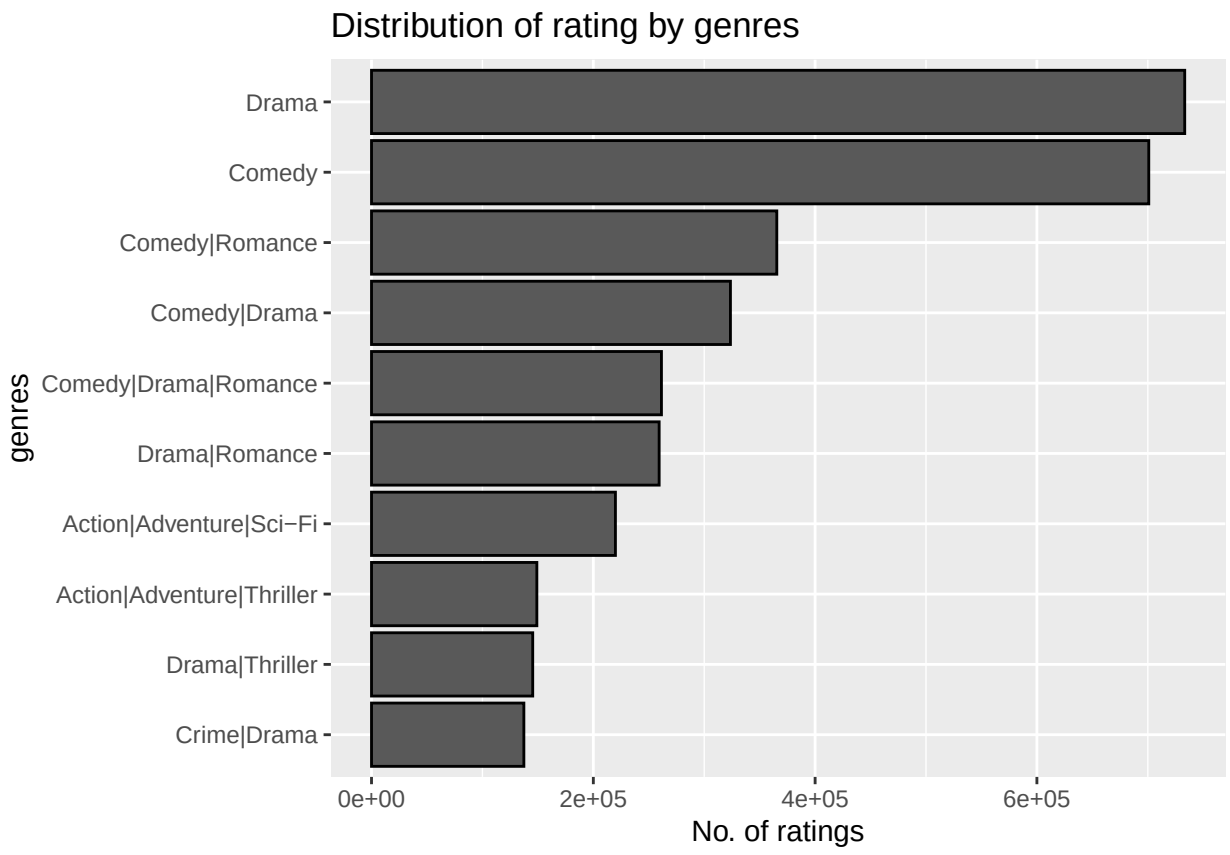
We observe that there are 19 named and one unnamed genres and 796 combinations of one or more of the named genres in the dataset. Let's further examine the unnamed genre to see if we can trivially reject it.

```
edx %>%
  dplyr::filter(genres %like% "^\\(")
```

```
#    userId movieId rating  timestamp                title             genres
#     <int>   <int>  <num>      <int>               <char>             <char>
# 1:   7701    8606    5.0 1190806786 Pull My Daisy (1958) (no genres listed)
# 2:  10680    8606    4.5 1171170472 Pull My Daisy (1958) (no genres listed)
# 3:  29097    8606    2.0 1089648625 Pull My Daisy (1958) (no genres listed)
# 4:  46142    8606    3.5 1226518191 Pull My Daisy (1958) (no genres listed)
# 5:  57696    8606    4.5 1230588636 Pull My Daisy (1958) (no genres listed)
# 6:  64411    8606    3.5 1096732843 Pull My Daisy (1958) (no genres listed)
# 7:  67385    8606    2.5 1188277325 Pull My Daisy (1958) (no genres listed)
```

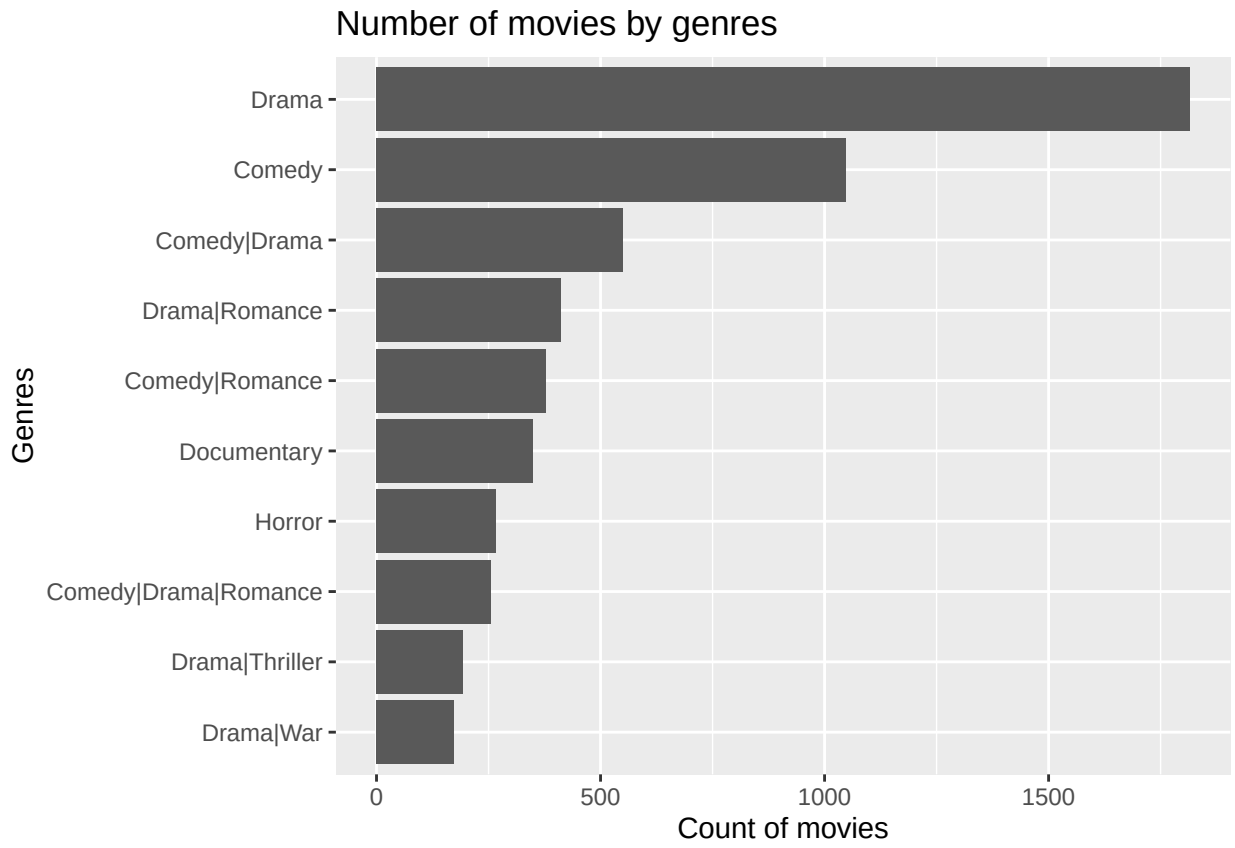
Since this is a very small subset we can trivially reject this unnamed genre and focus only on named genres. We'll first create a data frame that only contains movies and genres.

```
# Create a data frame for genres
genre_df <- edx %>%
  dplyr::filter(!genres %like% "\\(") %>% #filter unnamed genre
  group_by(genres) %>%
  summarise(N = n(), n_movies = n_distinct(movieId),
            avg = N/n_movies )
genre_df %>%
  slice_max(order_by = N, n = 10) %>%
  ggplot(aes(N,reorder(genres,N))) +
  geom_col(col = "black") +
  ggtitle("Distribution of rating by genres") +
  xlab("No. of ratings") +
  ylab("genres")
```



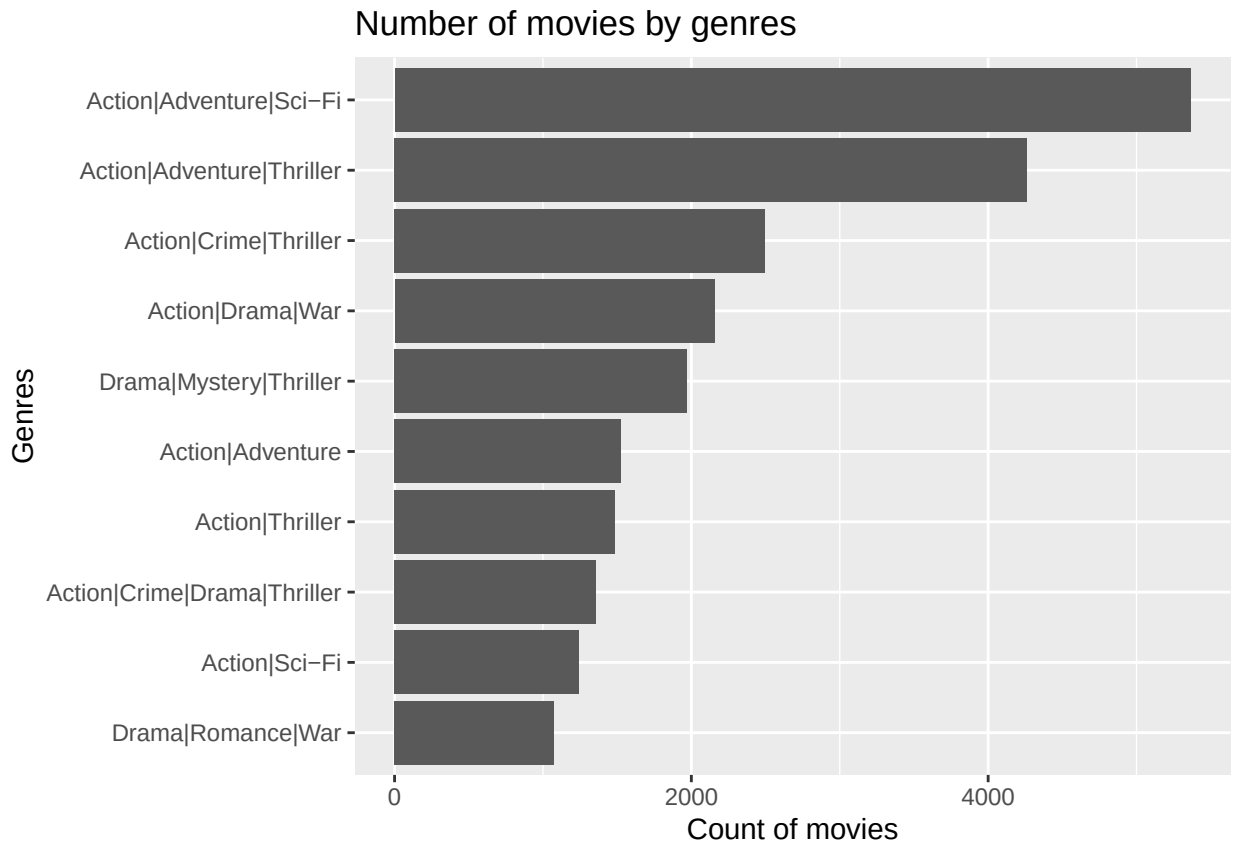
Now let's look at the genres that have the most number of movies

```
genre_df %>%
  slice_max(n_movies,n = 10) %>%
    ggplot(aes(n_movies,reorder(genres,n_movies))) +
    geom_col() +
    ggtitle("Number of movies by genres") +
    xlab("Count of movies") +
    ylab("Genres")
```



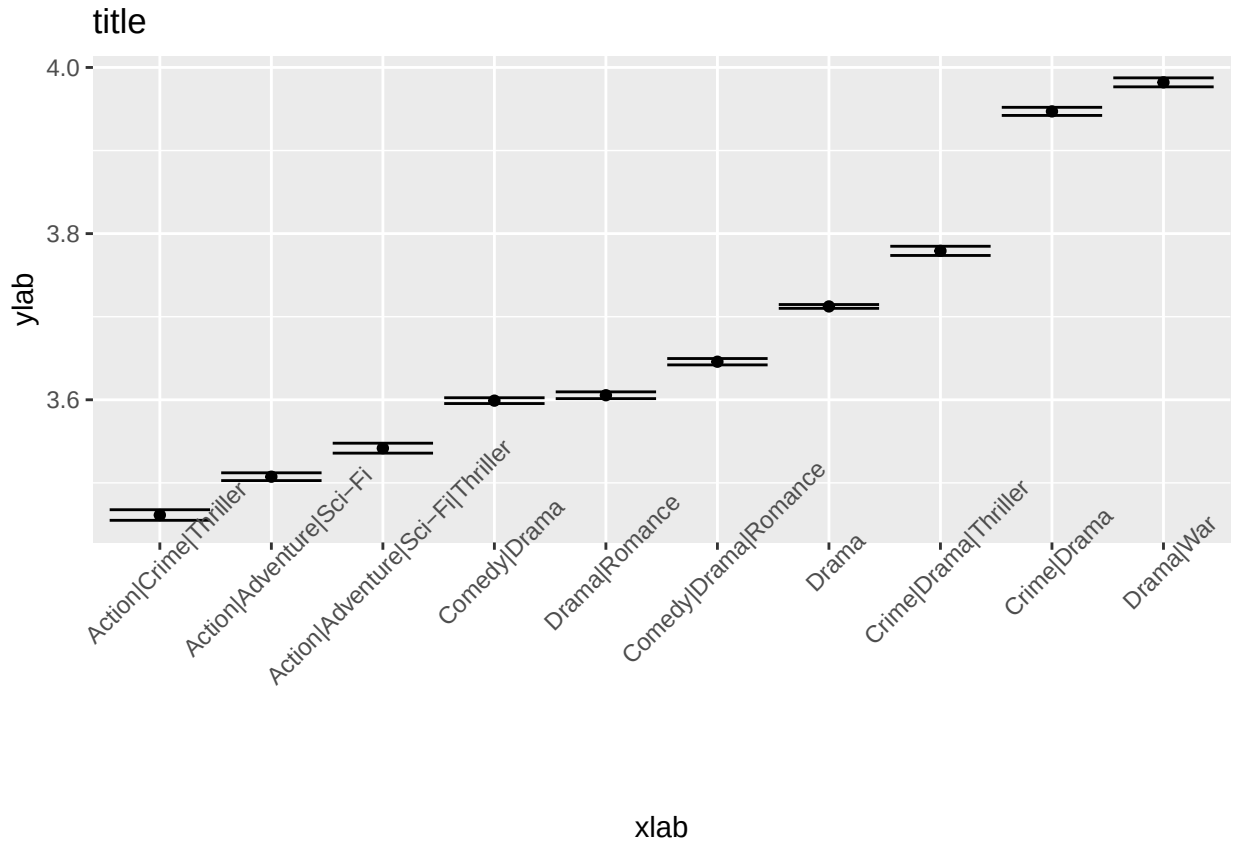
Let's also look at the average number of movies per genre.

```
genre_df %>%
  dplyr::filter(n_movies>30) %>%
  slice_max(avg,n = 10) %>%
    ggplot(aes(avg,reorder(genres,avg))) +
    geom_col() +
    ggtitle("Number of movies by genres") +
    xlab("Count of movies") +
    ylab("Genres")
```



Now we'll look at the average ratings of each genre.

```
edx %>%
  group_by(genres) %>%
  dplyr::filter(n()>10^5) %>%
  summarise(mean = mean(rating), se = sd(rating)/sqrt(n())) %>%
  slice_max(mean,n = 10) %>%
  ggplot(aes(reorder(genres,mean),mean, ymin = mean - 2*se,
                                        ymax = mean + 2*se)) +
  geom_point() +
  geom_errorbar() +
  ggtitle("title") +
  xlab("xlab") +
  ylab("ylab") +
  theme(axis.text.x = element_text(angle = 45))
```



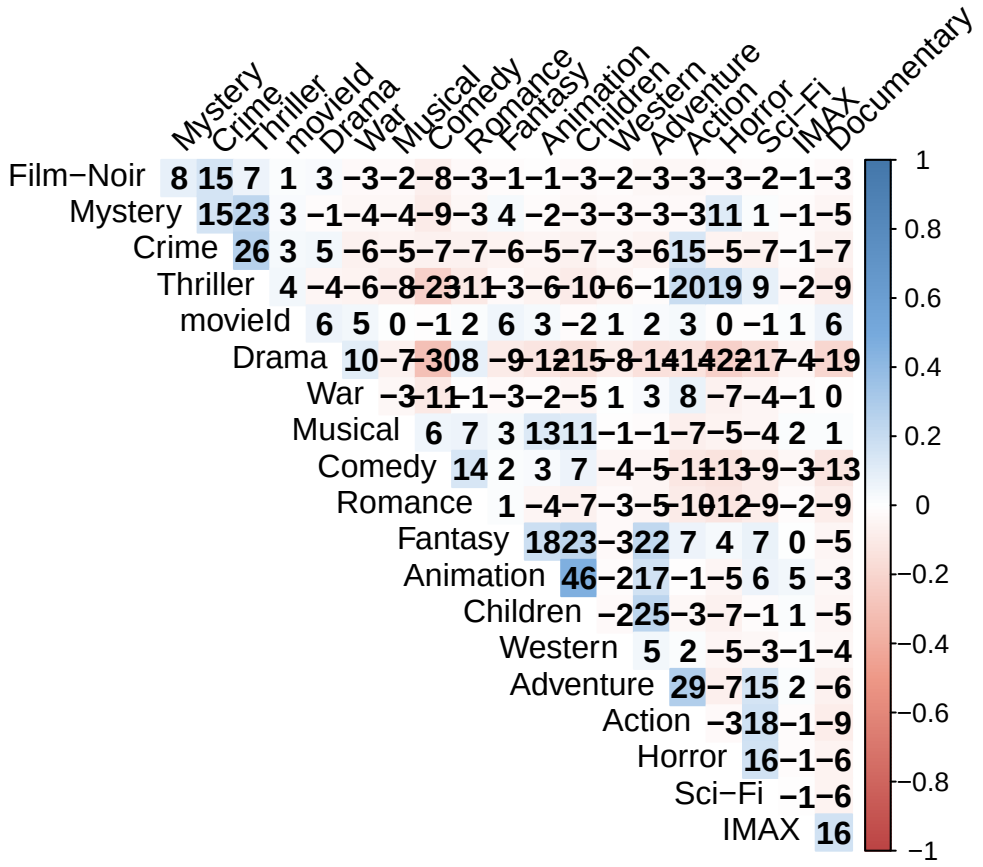
Now, let's look at the correlation between genres. We'll use a movie-genres matrix to determine the correlation.

```
#Creating the dataframe
mg_df <- edx %>%
  dplyr::filter(!genres %like% "^\\(") %>%
  group_by(movieId) %>%
  summarise(genres = genres[1])
head(mg_df,10)
```

```
# # A tibble: 10 x 2
#    movieId genres
#      <int> <chr>
#  1       1 Adventure|Animation|Children|Comedy|Fantasy
#  2       2 Adventure|Children|Fantasy
#  3       3 Comedy|Romance
#  4       4 Comedy|Drama|Romance
#  5       5 Comedy
#  6       6 Action|Crime|Thriller
#  7       7 Comedy|Romance
#  8       8 Adventure|Children
#  9       9 Action
# 10      10 Action|Adventure|Thriller
```

```
#Separating the genres to different rows
mg_df <- mg_df %>%
```

```
  separate_rows(genres,sep = "\\|")
head(mg_df,10)

# # A tibble: 10 x 2
#    movieId genres
#      <int> <chr>
#  1       1 Adventure
#  2       1 Animation
#  3       1 Children
#  4       1 Comedy
#  5       1 Fantasy
#  6       2 Adventure
#  7       2 Children
#  8       2 Fantasy
#  9       3 Comedy
# 10       3 Romance

#Creating genres matrix
mg_mat <- mg_df %>%
  mutate(genre_value = 1) %>%
  pivot_wider(movieId,names_from = genres,values_from=genre_value,
                       values_fill = 0,values_fn = mean)
#create correlation plot
col <- colorRampPalette(c("#BB4444", "#EE9988", "#FFFFFF", "#77AADD", "#4477AA"))
cor(mg_mat) %>%
  corrplot::corrplot(method = "col",
                     col=col(200),
                     type="upper",
                     order="hclust",
                     addCoef.col = "black", # Add coefficient of correlation
                     addCoefasPercent = T,  # Display coefficient in percentage
                     tl.col="black", tl.srt=45, #Text label color and rotation
                     diag=FALSE
)
```



2.2.5 timestamp

The timestamp records the time of the rating in seconds since 1970-01-01 midnight. We can convert this into a more readable format using this code.

```
time_df <- edx  %>%
  mutate(timestamp = as_datetime(timestamp))  %>%
  select(timestamp, rating)
  head(time_df)
```

```
#              timestamp rating
#                  <POSc>  <num>
# 1: 1996-08-02 11:24:06      5
# 2: 1996-08-02 10:58:45      5
# 3: 1996-08-02 10:57:01      5
# 4: 1996-08-02 10:56:32      5
# 5: 1996-08-02 10:56:32      5
# 6: 1996-08-02 11:14:34      5
```

```
  summary(time_df$timestamp)
```

```
#                  Min.               1st Qu.                Median
# "1995-01-09 11:46:49" "2000-01-01 23:11:23" "2002-10-24 21:11:58"
#                  Mean               3rd Qu.                  Max.
# "2002-09-21 13:45:07" "2005-09-15 02:21:21" "2009-01-05 05:02:16"
```

We understand from the summary that the ratings were made for a period of 14 years from 1995 to 2009. Let us plot the count of ratings for these years.

```
time_df %>% ggplot(aes(year(timestamp))) +
  geom_bar(col = "black") +
  scale_x_discrete(breaks = 1995:2010) +
  ggtitle("Yearwise distribution of Rating time") +
  xlab("Year of rating") +
  ylab("Count") +
  theme(axis.text.x = element_text(angle = 90))
```

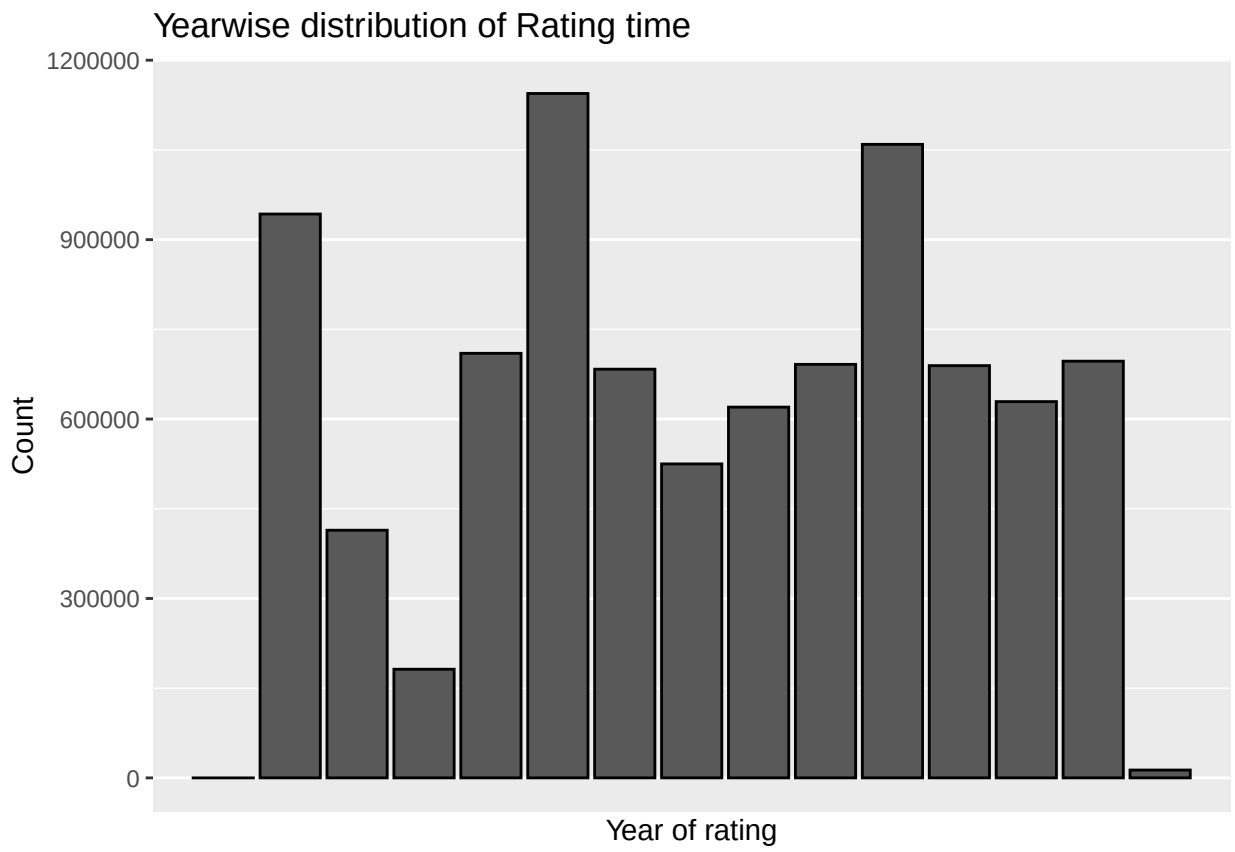


Figure 3: Yearwise distribution of time of rating

We observe that the distribution varies with time. The reason for this should be further explored. Now let's explore the relation of ratings with the time of rating

```
time_df %>%
  group_by(month = round_date(timestamp,unit = "month")) %>%
  summarize(rating = mean(rating))%>%
  ggplot(aes(month,rating))+
  geom_point()+
  geom_smooth()+
  labs(x="Time of Rating", y="Average Rating", title="Average Ratings Over Time")
```

There is thus a week effect with rating time.

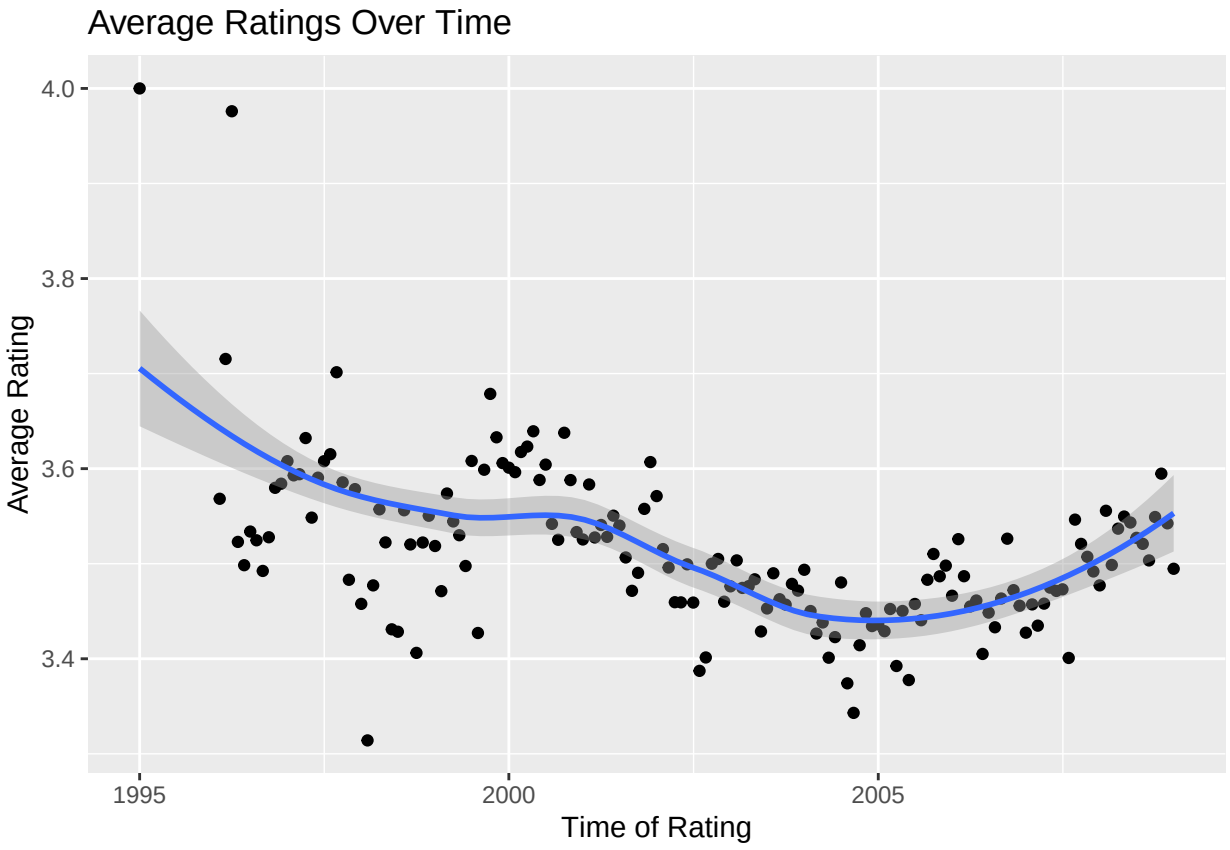


Figure 4: Average rating per month of rating

2.3 Data Selection & Cleaning

Before we can start building our models, we need to clean our data which involves selecting only the most relevant features. Since timestamp only shows a weak correlation we will discard it. The age of the movie from the title column will also not be used for modelling.

```
# Data Cleaning - Select the most relevant features
train_set<-train_set%>%select(-timestamp)
test_set<-test_set%>%select(-timestamp)
```

2.4 Modelling Approach

2.4.1 Linear Model - Naive Approach

The simplest recommender system is one which predicting the same rating for all user-movie pairs. From statistical theory we know that the mean minimises the losses. Thus the simplest model with all the differences explained by random variation can be expressed as follows:

$$\hat{Y}_{u,i} = \mu + \epsilon_{u,i}$$

Where $\hat{Y}$ is the predicted rating, μ is the mean of observed data and $\epsilon_{u,i}$ is the error distribution.

2.4.2 Linear Model with Movie & User Effects

The movie effect was vissualised in the previous section. A model that takes into account the movie effect can be expressed as follows:

$$\hat{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

| The movie bias is calculated as the average of the difference between the observed rating y and the mean μ for movie i.

$$b_i = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu})$$

| Similarly user effect is calculated as:

$$b_u = \frac{1}{N}\sum_{i=1}^{N}(y_{u,i} - b_i - \hat{\mu})$$

| A model that takes into account user and movie effects is expressed as follows:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

| Finally one that considers movie, user and genres effects is given by:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$$

2.4.3 Regularization

Although our linear model provides a good prediction of the ratings, it does not take into consideration that some movies have very few ratings and some users only rate a few movies. These small sample sizes of movies and users produce noisy estimates, which results in large errors, and can therefore increase our RMSE. This is because there is more uncertainty when we have a few users.

In order to constrain the total variability of the movie and user effects, we introduce a concept called *regularization* to our model. Regularization allows us to penalize large estimates when working with smaller sample sizes. It is similar to the *Bayesian* approach. The modified movie and user effects, with the penalty term added, can be calculated as follows:

$$\hat{b}_i = \frac{1}{n_i + \lambda} \sum_{i=1}^{N} (y_i - \hat{\mu})$$

$$\hat{b}_u = \frac{1}{n_u + \lambda} \sum_{i=1}^{N} (y_{u,i} - \hat{b}_i - \hat{\mu})$$

$$\hat{b}_g = \frac{1}{n_u + \lambda} \sum_{i=1}^{N} (y_{u,i} - \hat{b}_i - b_u - \hat{\mu})$$

This is the formula we will be minimizing:

$$\sum_{u,i} (y_{u,i} - \mu - b_i - b_u - b_g)^2 + \lambda (\sum_{i} b_i^2 + b_u^2 + b_g^2)$$

This approach gives us the desired effect: When our sample size n_i is very large, a case which will give us a stable estimate, then the penalty λ is effectively ignored since $n_i + \lambda \sim n_i$. However, when the n_i is small, then the estimate $\hat{b}_i \lambda$ shrinks towards 0. The larger the penalty λ, the more we shrink.

Since the penalty λ is a tuning parameter, we can use cross-validation to choose it by running simulations with different values of λ. We choose the λ which minimizes the RMSE the most.

3 Modelling

3.1 Model Setup

We are first going to define RMSE as our loss function.

```
# Define Root Mean Squared Error (RMSE) - loss function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

We will also create a results table that we will update with each model's RMSE in order to compare the different models.

```
# create a results table with all the RMSEs
rmse_results <- tibble(Method = "Project Goal", RMSE = 0.86490)
```

3.2 Linear Model

The prediction is the average of all ratings.

```
# Mean of observed ratings
mu <- mean(train_set$rating)
mu

# [1] 3.512456

# Naive RMSE - predict all unknown ratings with overall mean
naive_rmse <- RMSE(test_set$rating, mu)
naive_rmse

# [1] 1.060054

# Update results table with naive RMSE
rmse_results <- bind_rows(rmse_results,
                    tibble(Method = "Naive RMSE",
                           RMSE = RMSE(test_set$rating, mu)))
```

3.3 Linear Model with movie effect

Here we add the movie effects to the average rating.

```
# Calculate movie effect
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# Predict ratings with mean + bi
y_hat <- mu + test_set %>%
  left_join(b_i, by = "movieId") %>%
  .$b_i
RMSE(test_set$rating, y_hat)

# [1] 0.9429615

# Update results table with movie effect RMSE
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Movie Effect",
                                 RMSE = RMSE(test_set$rating, y_hat)))
```

We observe that the rmse has significantly improved.

3.4 Linear Model with movie and user effects

Here both the effects are added to the average rating.

```
# Calculate user effect
b_u <- train_set %>%
  left_join(b_i, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# Predict ratings with mean + bi + bu
y_hat <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
RMSE(test_set$rating, y_hat)

# [1] 0.8646843

# Update results table with movie effect RMSE
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Movie + User Effect",
                                 RMSE = RMSE(test_set$rating, y_hat)))
```

3.5 Linear model with user, movie and genre effect

Now we will add the final effect to our model.

```
# Calculate genres effect
b_g <- train_set %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        group_by(genres) %>%
        summarise(b_g = mean(rating - b_i - b_u -mu))

# Predict ratings
y_hat <- test_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(b_g, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
RMSE(test_set$rating, y_hat)

# [1] 0.8643

# Update results table with movie effect RMSE
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Movie + User + Genres Effect",
                                 RMSE = RMSE(test_set$rating, y_hat)))
```

Model Evaluation

Including the user effect further minimizes the RMSE, proving that this is a better model. However, we still need to make sure the model makes sound predictions. We can check where we made mistakes by looking at the 10 largest errors, using only the movie effect.

```
# Evaluate Model Results
# calculate biggest residuals (errors)
test_set %>%
  left_join(b_i, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  head(10)
```

userId	movieId	rating	title	genres	b_i	residual
29924	6483	5.0	From Justin to Kelly (2003)	Musical|Romance	-2.6381387	4.125683
10680	318	0.5	Shawshank Redemption, The (1994)	Drama	0.9441110	-3.956567
51845	318	0.5	Shawshank Redemption, The (1994)	Drama	0.9441110	-3.956567
39975	858	0.5	Godfather, The (1972)	Crime|Drama	0.9041954	-3.916651
49231	858	0.5	Godfather, The (1972)	Crime|Drama	0.9041954	-3.916651
58785	858	0.5	Godfather, The (1972)	Crime|Drama	0.9041954	-3.916651
70446	858	0.5	Godfather, The (1972)	Crime|Drama	0.9041954	-3.916651
62012	50	0.5	Usual Suspects, The (1995)	Crime|Mystery|Thriller	0.8540962	-3.866552
17017	527	0.5	Schindler's List (1993)	Drama|War	0.8516292	-3.864085
25397	527	0.5	Schindler's List (1993)	Drama|War	0.8516292	-3.864085

Next we look at the 10 best and 10 worst movies based on b_i.

```
# create a database of movie titles
movie_titles <- train_set %>%
  select(movieId, title) %>%
  distinct()
```

Top 10 best movies

```
# 10 best movies according to b_i
b_i %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  head(10)  %>%
  select(title)
```

title
Hellhounds on My Trail (1999)
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)
Shadows of Forgotten Ancestors (1964)
Fighting Elegy (Kenka erejii) (1966)
Sun Alley (Sonnenallee) (1999)
Blue Light, The (Das Blaue Licht) (1932)
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)
Life of Oharu, The (Saikaku ichidai onna) (1952)
Human Condition II, The (Ningen no joken II) (1959)
Human Condition III, The (Ningen no joken III) (1961)

Top 10 worst movies

```r
# 10 worst movies according to bi
b_i %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  head(10)  %>%
  select(title)
```

title
Besotted (2001)
Hi-Line, The (1999)
Accused (Anklaget) (2005)
Confessions of a Superhero (2007)
War of the Worlds 2: The Next Wave (2008)
SuperBabies: Baby Geniuses 2 (2004)
Disaster Movie (2008)
From Justin to Kelly (2003)
Hip Hop Witch, Da (2000)
Criminals (1996)

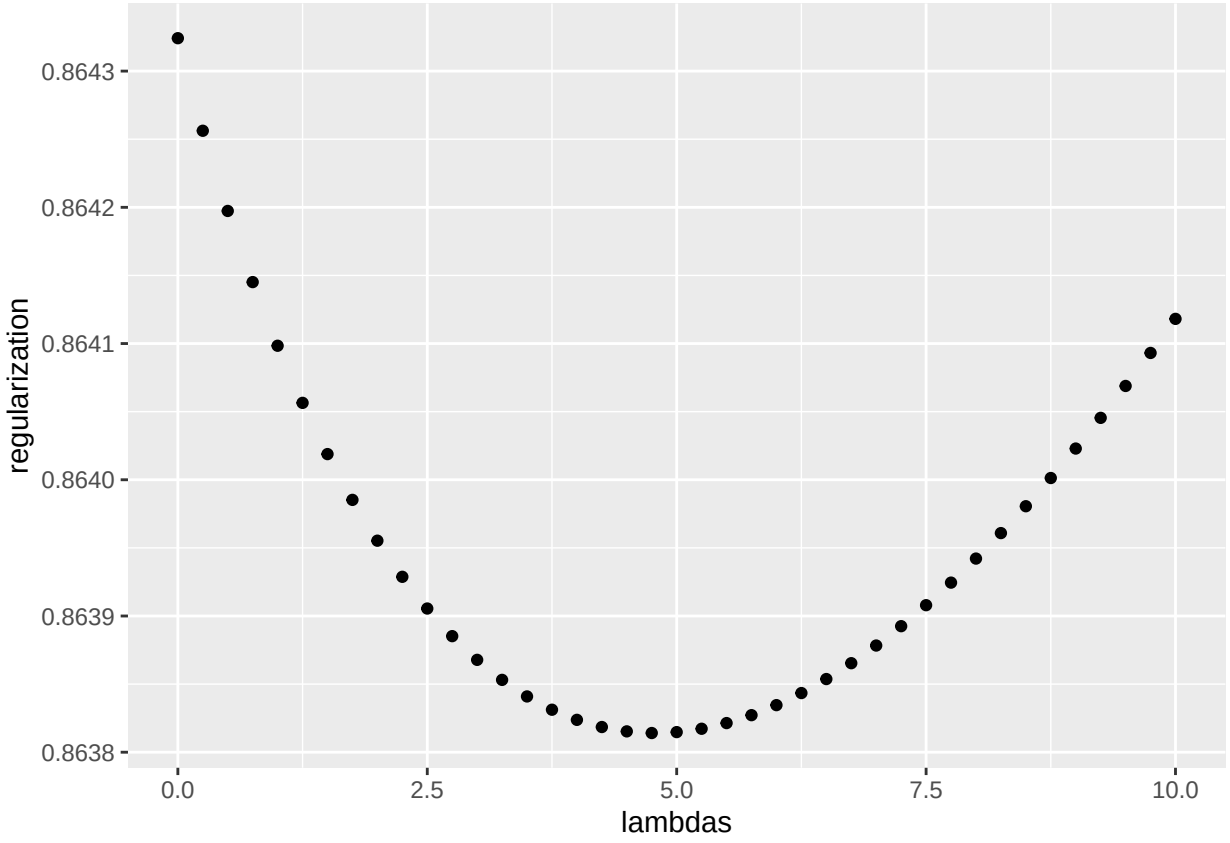
These movies are unpopular movies. Hence we realise that our model is at a disadvantage as it shows greater bias when the sample size is large. Hence we need to regularise our model.

3.6 Regularization

Due to the uncertainty created by small sample sizes, we add a penalty term λ to our model to regularize the movie and user effects. λ is a tuning parameter, which means we can use cross-validation to select the optimal value that minimizes the RMSE. We will write a regularization function to simulate several values of λ.

```r
# Define a set of lambdas to tune
lambdas <- seq(0, 10, 0.25)
# Tune the lambdas using regularization function
regularization <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

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

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

  b_g <- train_set %>%
    left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - b_i - b_u -mu)/(n()+l))

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
```

```
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    mutate(pred = mu + b_i + b_u + b_g) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})
# Plot - lambdas vs RMSE
qplot(lambdas, regularization)
```




```
# Choose the lambda which produces the lowest RMSE
lambda<- lambdas[which.min(regularization)]
lambda

# [1] 4.75
```

Now that we have found the optimal λ which minimizes the RMSE, we apply it to our model by regularizing the movie and user effects.

```
# Calculate the movie and user effects with the best lambda (parameter)
mu <- mean(train_set$rating)

# Movie effect (bi)
bi_reg <- train_set %>%
```

```
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# User effect (bu)
bu_reg <- train_set %>%
  left_join(bi_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Genres Effect
bg_reg <- train_set %>%
  left_join(b_i, by = 'movieId') %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarise(b_g = sum(rating - b_i - b_u -mu)/(n()+lambda))

# Prediction with regularized bi and bu
y_hat_reg <- test_set %>%
  left_join(bi_reg, by = "movieId") %>%
  left_join(bu_reg, by = "userId") %>%
  left_join(bg_reg, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
# Update results table with regularized movie + user effect
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Regularized Model",
                                 RMSE = RMSE(test_set$rating, y_hat_reg)))
```

Model Evaluation

Regularization of the effects did indeed improve our RMSE but does it make better predictions?

Let's look at the top 10 best movies after penalizing movie and user effects:

```
# Top 10 best movies after regularization
test_set %>%
  left_join(bi_reg, by = "movieId") %>%
  left_join(bu_reg, by = "userId") %>%
  left_join(bg_reg, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  arrange(desc(pred)) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Star Wars: Episode V - The Empire Strikes Back (1980)
Full Metal Jacket (1987)
Shawshank Redemption, The (1994)
Shawshank Redemption, The (1994)
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
Beautiful Thing (1996)
Sling Blade (1996)
Shawshank Redemption, The (1994)
Star Wars: Episode VI - Return of the Jedi (1983)
Schindler's List (1993)

These make much more sense. They are popular movies that have been rated many times.

These are the top 10 worst movies based on penalized movie and user effects:

```
# Top 10 worst movies after regularization
test_set %>%
  left_join(bi_reg, by = "movieId") %>%
  left_join(bu_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  arrange(pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Speed 2: Cruise Control (1997)
Faces of Death 2 (1981)
Driven (2001)
Free Willy 3: The Rescue (1997)
Meatballs 4 (1992)
Faces of Death: Fact or Fiction? (1999)
Gigli (2003)
Problem Child 2 (1991)
Gigli (2003)
PokÃ©mon the Movie 2000 (2000)

3.7 Final Validation

For the final validation we will use the edx dataset to train our last model. If the final RMSE on the validation set is less than .8649 we will achieve our target.

```
# Calculate the movie and user effects with the best lambda (parameter)
mu <- mean(edx$rating)

# Movie effect (bi)
bi_fin <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# User effect (bu)
bu_fin <- edx %>%
```

Method	RMSE
Project Goal	0.8649000
Naive RMSE	1.0600537
Movie Effect	0.9429615
Movie + User Effect	0.8646843
Movie + User + Genres Effect	0.8643241
Regularized Model	0.8638187
Final Model	0.8648597

```
  left_join(bi_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Genres Effect
bg_fin <- edx %>%
  left_join(b_i, by = 'movieId') %>%
  left_join(b_u, by = "userId") %>%
  group_by(genres) %>%
  summarise(b_g = sum(rating - b_i - b_u -mu)/(n()+lambda))

# Prediction with regularized bi and bu
y_hat_fin <- validation %>%
  left_join(bi_reg, by = "movieId") %>%
  left_join(bu_reg, by = "userId") %>%
  left_join(bg_reg, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
# Update results table with regularized movie + user effect
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Final Model",
                                 RMSE = RMSE(validation$rating, y_hat_fin)))
```

4 Results

```
knitr::kable(rmse_results)%>% kable_styling()
```

The final validation produces an RMSE of 0.86486 which achieves the set target.

Let's look at the top 10 best movies and 10 worst movies predicted using final model:

```
# top 10 best movies predicted by matrix factorization
validation %>%
  mutate(pred = y_hat_fin) %>%
  arrange(desc(pred)) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Usual Suspects, The (1995)
Shawshank Redemption, The (1994)
Shawshank Redemption, The (1994)
Shawshank Redemption, The (1994)
Eternal Sunshine of the Spotless Mind (2004)
Donnie Darko (2001)
Schindler's List (1993)
Schindler's List (1993)
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
Star Wars: Episode VI - Return of the Jedi (1983)

```
# top 10 worst movies predicted by matrix factorization
validation %>%
  mutate(pred = y_hat_fin) %>%
  arrange(pred) %>%
  group_by(title) %>%
  select(title) %>%
  head(10)
```

title
Battlefield Earth (2000)
Police Academy 4: Citizens on Patrol (1987)
Karate Kid Part III, The (1989)
PokÃ©mon Heroes (2003)
Kazaam (1996)
Turbo: A Power Rangers Movie (1997)
PokÃ©mon Heroes (2003)
Barney's Great Adventure (1998)
Free Willy 3: The Rescue (1997)
RoboCop 2 (1990)

5 Conclusion

The aim of this project was to create a movie-recommender system. We started by exploring the dataset, and selected the best predicting features.

We started modelling with the mean value and proceeded to add movie, user and genres effects to the mean value. The model improved its performance but for movies with small sample size, the model performed poorly. Hence we regularised the model. The regularised model achieved the target of RMSE $<$ .8649.

5.1 Limitations

1. The model doesn't exactly say if a particular user likes a movie or not. It only says whether a movie is highly rated, user rates genrously, genres is popular etc.
2. Some machine learning algorithms are computationally expensive to run in a commodity laptop and therefore were unable to test. The required amount of memory far exceeded the available in a commodity laptop, even with increased virtual memory.
3. The model works only for existing users, movies and rating values, so the algorithm must run every time a new user or movie is included, or when the rating changes. This is not an issue for small client base and a few movies, but may become a concern for large data sets. The model should consider these changes and update the predictions as information changes.

5.2 Future Work

1. Grouping genres into fewer categories based on correlation between genres. This will help examine how much a user likes a genre as opposed how much it is liked overall.
2. Deep Learning and other advanced recommendation algorithms maybe used.