Capstone Project - MovieLens

Fidan Gasim

11/28/2020

Contents

Preface

This report was prepared as part of the capstone project for HarvardX's Data Science Professional Certificate Program[1].

1. Introduction

1.1 Overview

With the rise of digital platforms such as YouTube, Amazon, Netflix, Spotify, recommendation engines have become more popular and sophisticated than ever before. The purpose of recommendation engines is to recommend relevant items (eg. movies, music, food, etc.) to users.

Generally, recommendations systems are special types of machine learning algorithms that provide "relevant" suggestions to users. Effective recommendation systems increase customer retention and satisfaction. When executed efficiently, they can generate large revenue streams and give companies a competitive edge in their industry.

A famous example is when Netflix organized a challenge (called the "Netflix Prize")[2] in 2006 with the goal of producing a recommender system that improved its own algorithm by at least 10%. The winners received a million dollar prize.

Recommendation systems typically use ratings that users have given items to make specific recommendations. Companies, like Amazon, are able to collect huge datasets that can be used to predict what rating a particular user will give a specific item. Items for which a high rating is predicted for a given user are then recommended to that user[3]. Other indicators such as time spent on a particular webpage, comments on particular items, visited web pages, links shared with family or friends, product category, and any other interactions the user has with the platform or website can be used as predictors.

For the purposes of this project, we will be creating a movie recommendation system using the MovieLens dataset, using methods and tools that have been learned during HarvardX's Data Science Professional Certificate program[4].

1.2 MovieLens Dataset

The MovieLens dataset was collected and made available by GroupLens Research, a research lab based at the University of Minnesota[5]. The rating data is collected through the MovieLens website[6], which helps users find movies to watch based on the ratings they give to other movies, amongst other predictors.

The complete MovieLens dataset[7] consists of 27 million ratings. For the purposes of this project, we will be working with a subset of this dataset. Specifically, we will be working with the MovieLens 10M Dataset[8], which includes 10 million movie ratings applied to 10,000 movies by 72,000 users.

[1] https://www.edx.org/professional-certificate/harvardx-data-science
[2] https://www.netflixprize.com/
[3] https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems
[4] https://www.edx.org/professional-certificate/harvardx-data-science
[5] https://grouplens.org/about/what-is-grouplens/
[6] https://movielens.org/
[7] https://grouplens.org/datasets/movielens/latest/
[8] https://grouplens.org/datasets/movielens/10m/

1.3 Model Evaluation

When working on any data science project, it is important to evaluate the performance of the machine learning model and how it generalizes to new, unseen data. The model evaluation involves comparing the predicted values with the actual outcomes. In other words, we need to examine whether the model actually works and how well we can trust its predictions[9].

Evaluating a model's performance involves using a hold-out test set (data not seen by the model), which provides an "unbiased estimate of learning performance"[10]. Typically, data that is used to build the model is not used to evaluate it. Using the training set for both building and testing the algorithm can lead to *overfitting*, which is when the model memorizes the entire training set and does not adapt well to new data.

There are many types of metrics, such as Classification Accuracy, Confusion Matrix, Area Under Curve (AUC), and F-Measure, that are used to evaluate machine learning algorithms. For this project we will be using a loss function, specifically RMSE (Root Mean Squared Error), to evaluate the performance of our models. The Netflix challenge used the typical error loss: they decided on a winner based on the residual mean squared error (RMSE) on a test set.

The loss function measures the difference between predicted values and actual outcomes. In other words, when the user consistently selects the predicted movie, the error is zero and the algorithm is perfect. The goal of this project is to create a recommendation 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, it means our typical error is larger than one star, which is not optimal[11].

1.4 Process Outline

Besides data collection, the other main steps in a machine learning 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 explore any assumptions before modeling. Any necessary data transformations as well as key features/predictors would be identified in this step.
4. Data Preprocessing: involves cleaning and transforming the data, such as feature selection, scaling, removing unnecessary information, etc.
5. Modeling Methods: researching and selecting modeling approaches that work best for the type of dataset.
6. Data Modeling: Creating, training and testing the selected models, including any fine tuning of parameters.
7. Model Evaluation: Evaluating the results and model's performance.

[9] https://heartbeat.fritz.ai/introduction-to-machine-learning-model-evaluation-fa859e1b2d7f
[10] https://heartbeat.fritz.ai/introduction-to-machine-learning-model-evaluation-fa859e1b2d7f
[11] https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems

8. Communication: Present the final results along with any limitations or recommendations for future work.

We start off this project by downloading and importing the dataset from MovieLens. We then split the dataset into two subsets: edx and validation. The edx subset is further split into training and testing sets in order to design and test our algorithm. RMSE (Root Mean Squared Error) will be used to evaluate how close our predictions are to the true values in the validation set. When the model reaches the target RMSE using the training and testing sets, it is then trained on the edx dataset and tested using the validation set. The validation set is the final hold-out test set and is not used for training, developing or selecting the machine learning algorithm. We predict movie ratings in the validation set as if they were unknown.

In the following section, we will analyze the data to look for trends/effects and any relationships between the features and outcomes. We will produce charts, tables, and summary statistics to show these relationships and trends. The insights gained during the data exploration process will help us build our machine learning model.

In order to build an effective recommender system, we need to identify the most important features that will help predict what rating any user will give any movie. First, we will use a model based approach. We start by building the simplest model: we predict the same rating for all movies regardless of user (using just the mean of the observed values). We then add the movie and user effects to the linear model, which improves the RMSE. Lastly, we penalize estimates coming from small samples through a process called *regularization*. This achieves the target RMSE.

We create and evaluate another machine learning approach called *matrix factorization*, using the R recosystem package. This produces an even lower RMSE than regularization.

2. Data Exploration and Preprocessing

2.1 Data Preparation

In this section, we start off by loading the necessary libraries we'll be using to analyze our data and train our models.

```
# Load libraries
library(tidyverse)
library(tidyr)
library(dplyr)
library(ggplot2)
library(caret)
library(data.table)
library(knitr) #A General-Purpose Package for Dynamic Report Generation in R
library(kableExtra) #to build common complex tables and manipulate table styles
library(lubridate)
library(Matrix.utils) #Data.frame-Like Operations on Sparse and Dense Matrix Objects.
library(DT) #provides an R interface to the JavaScript library DataTables.
library(RColorBrewer) #Provides color schemes for maps (and other graphics)
library(ggthemes) #Some extra themes, geoms, and scales for 'ggplot2'.
library(scales)
library(recosystem) #Recommender System using Matrix Factorization
library(purrr)
library(devtools)
library(ggrepel)
library(splitstackshape)
```

```
library(tinytex)
library(latexpdf)

# set global options
options(timeout=10000, digits=10)
```

We download and import the original dataset using the URL link where it is stored.

We 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.

Once the target RMSE is achieved using the test set, we then train the final model on the entire edx dataset and test it using the validation set. This approach is referred to as *cross-validation*.

```r
# 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 Visualization

Before we start creating our machine learning models, we analyze the data to understand its nature and structure. We want to have a better understanding of the distribution of ratings and capture any significant relationships between predictors and outcomes, effects, and so on. This step will help us in building better models.

```r
class(edx)
```

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

```r
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|Adv
#  - attr(*, ".internal.selfref")=<externalptr>
```

```r
dim(edx)
```

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

```r
head(edx)
```

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

From this initial summary, we learn that the edx dataset is a data frame consisting of 9,000,055 rows and 6 columns. The type of data in each column is as follows:

- userId – integer
- movieId – numeric
- rating – numeric
- timestamp - integer
- title - character
- genres – character

We learn the content of the dataset by taking a glimpse at the first few rows of edx. We can see that the dataset is in tidy format, i.e. each row is an observation and the column names correspond to the features. The rating column is the desired outcome. The user information is stored in the userId column and the movie information is stored in the movieId, title, and genre columns. Each movie is tagged with one or more genres. The rating date and time is stored in seconds and can be found in the timestamp column.

In the following sections, we will take a more in-depth look at each of the features as well as the outcome.

We start off by analyzing the rating column which is the desired outcome.

2.2.1 Ratings

Users can rate movies on a scale of 0.5 to 5, with 10 possible ratings.

We first count the number (quantity) of each rating, which gives us an idea about the distribution of the ratings:

```
# Count the number of ratings per rating
edx %>%
  group_by(rating) %>%
  summarize(n=n())
```

rating	n
0.5	85374
1.0	345679
1.5	106426
2.0	711422
2.5	333010
3.0	2121240
3.5	791624
4.0	2588430
4.5	526736
5.0	1390114

We then plot the distribution of the ratings.

```
# Plot - Distribution of ratings
edx %>%
  group_by(rating) %>%
  summarize(n=n()) %>%
  ggplot(aes(x=rating, y=n))+
  geom_bar(stat="identity", color="white")+
  theme(axis.text.x =element_text(angle = 45, hjust = 1)) +
  labs(x="Rating", y="Count", title="Distribution of Ratings by Count")+
```

```
  scale_y_continuous(breaks = c(0,1000000,2000000,3000000),
                     labels=c("0","1M","2M","3M"))+
  theme_economist()
```

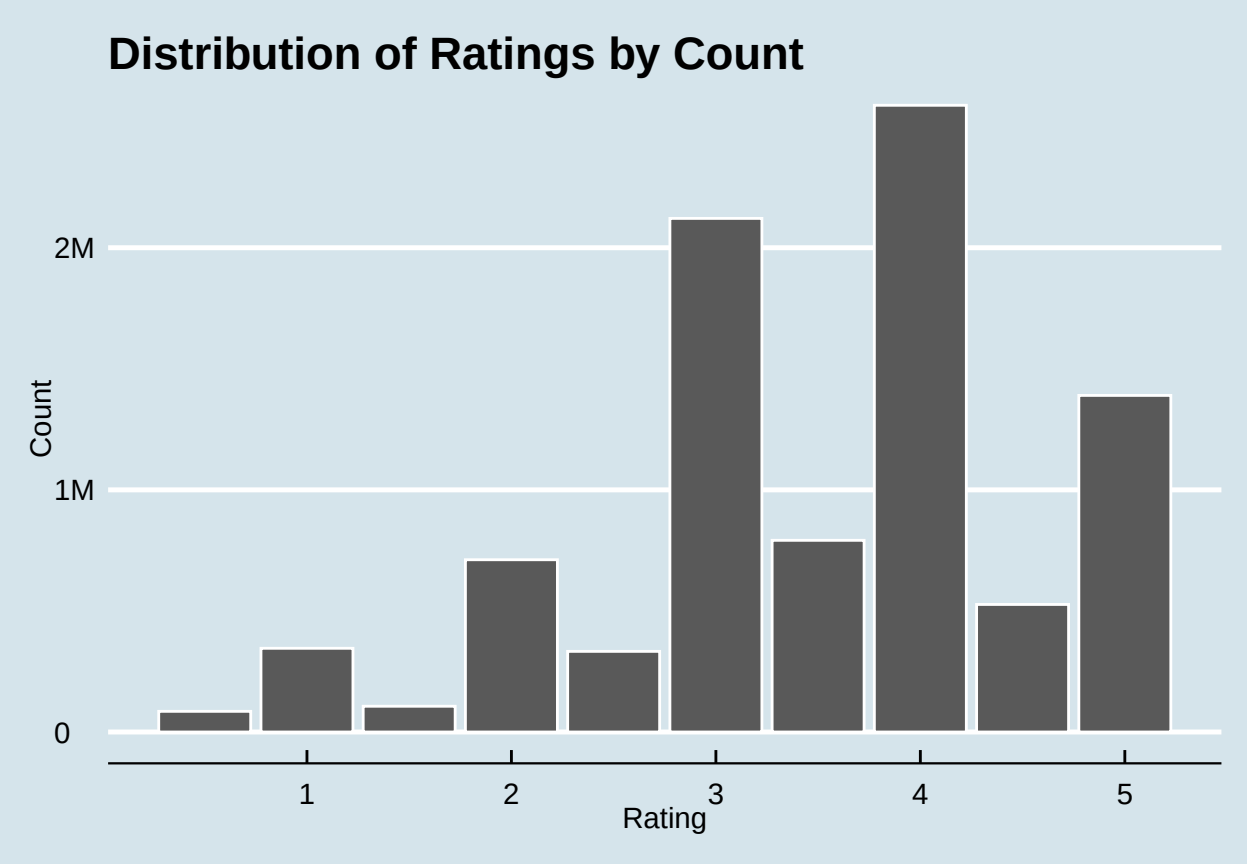


Figure 1: Distribution of Ratings

From the plot, we can see the following:

- Whole star ratings are more common than half star ratings.
- Ratings of 4 and 3 are the most common, accounting for over 50% of the ratings.

2.2.2 Movies

We start analyzing movie information by first counting the number of movies in the edx dataset.

```
# Number of movies in edx dataset
n_distinct(edx$movieId)
```

```
# [1] 10677
```

We learn that there are 10,677 movies and by intuition we know that some movies are rated more than others, for example blockbuster films are seen by more users and therefore rated more often than smaller independent films that are seen by fewer users.

```
# Number of ratings per movie
edx %>%
  group_by(movieId, title) %>%
  summarise(n=n())%>%
  head()
```

movieId	title	n
1	Toy Story (1995)	23790
2	Jumanji (1995)	10779
3	Grumpier Old Men (1995)	7028
4	Waiting to Exhale (1995)	1577
5	Father of the Bride Part II (1995)	6400
6	Heat (1995)	12346

```
# Movies with the highest number of ratings (popular movies)
edx %>%
  group_by(movieId, title) %>%
  summarise(n=n())%>%
  arrange(desc(n)) %>%
  head()
```

movieId	title	n
296	Pulp Fiction (1994)	31362
356	Forrest Gump (1994)	31079
593	Silence of the Lambs, The (1991)	30382
480	Jurassic Park (1993)	29360
318	Shawshank Redemption, The (1994)	28015
110	Braveheart (1995)	26212

```
# Average number of ratings per movie
avgratings<-edx %>%
  group_by(movieId, title) %>%
  summarise(n=n())

mean(avgratings$n)
```

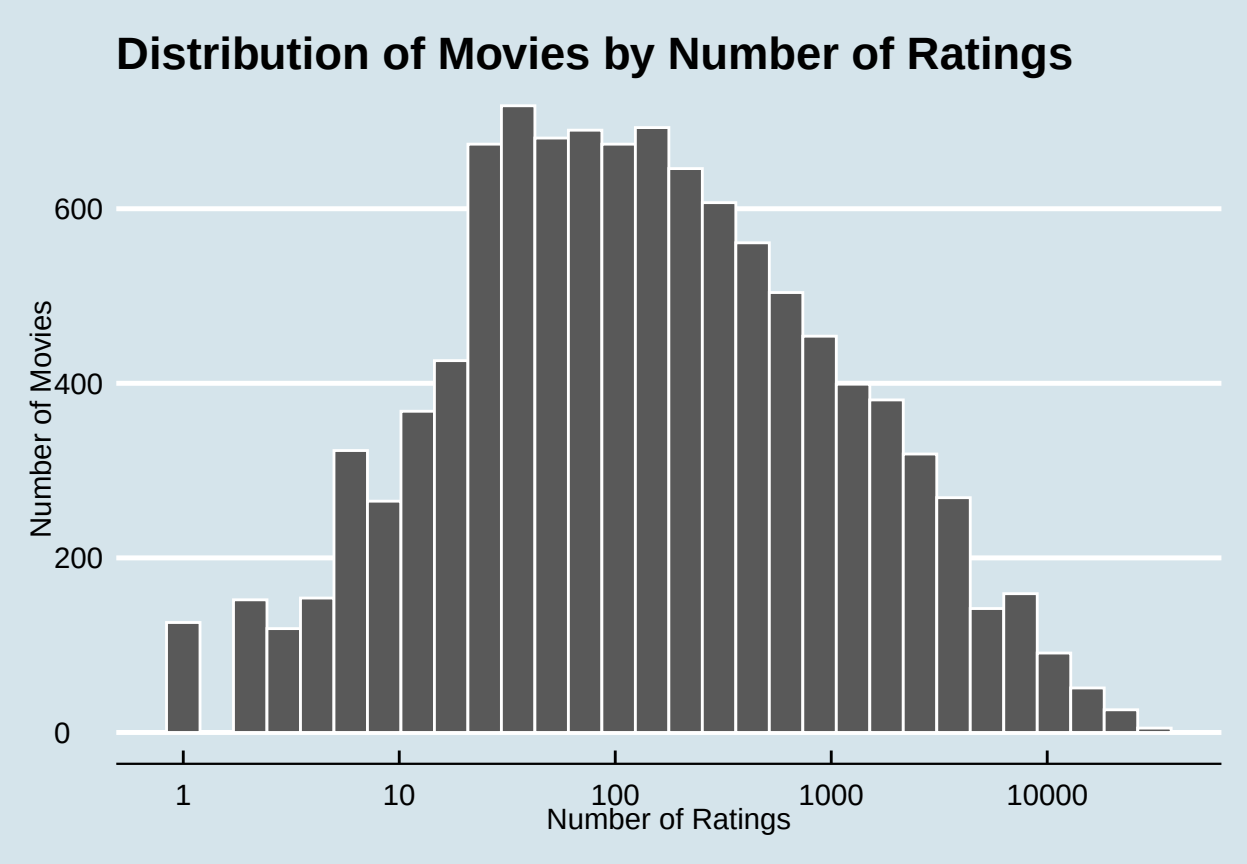
```
# [1] 842.9385595
```

As expected, movies which received the most number of ratings are popular films or blockbusters.

The average number of ratings received per film is 843.

Next we plot the distribution of movies by the number of ratings.

```
# Plot - Distribution of movies by number of ratings
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "white") +
  scale_x_log10() +
  labs(x="Number of Ratings", y="Number of Movies",
       title="Distribution of Movies by Number of Ratings")+
  theme_economist()
```



The plot confirms our intuition that some movies get rated more often than others, highlighting the prevalence of movie effects, which we'll explore further when building and training our model.

The following section provides some additional information about the movies in the dataset by analyzing the release year found in the movie title column.

```
# Create movies dataframe - Extract year from movie title
movies_df <- edx%>%
  mutate(title = str_trim(title)) %>%
  extract(title, c("temp_title", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F) %>%
  mutate(year = if_else(str_length(year) > 4, as.integer(str_split(year, "-", simplify = T)[1]),
                        as.integer(year))) %>%
  mutate(title = if_else(is.na(temp_title), title, temp_title)) %>%
  select(-temp_title)  %>%
  mutate(genres = if_else(genres == "(no genres listed)", `is.na<-`(genres), genres))
head(movies_df)
```

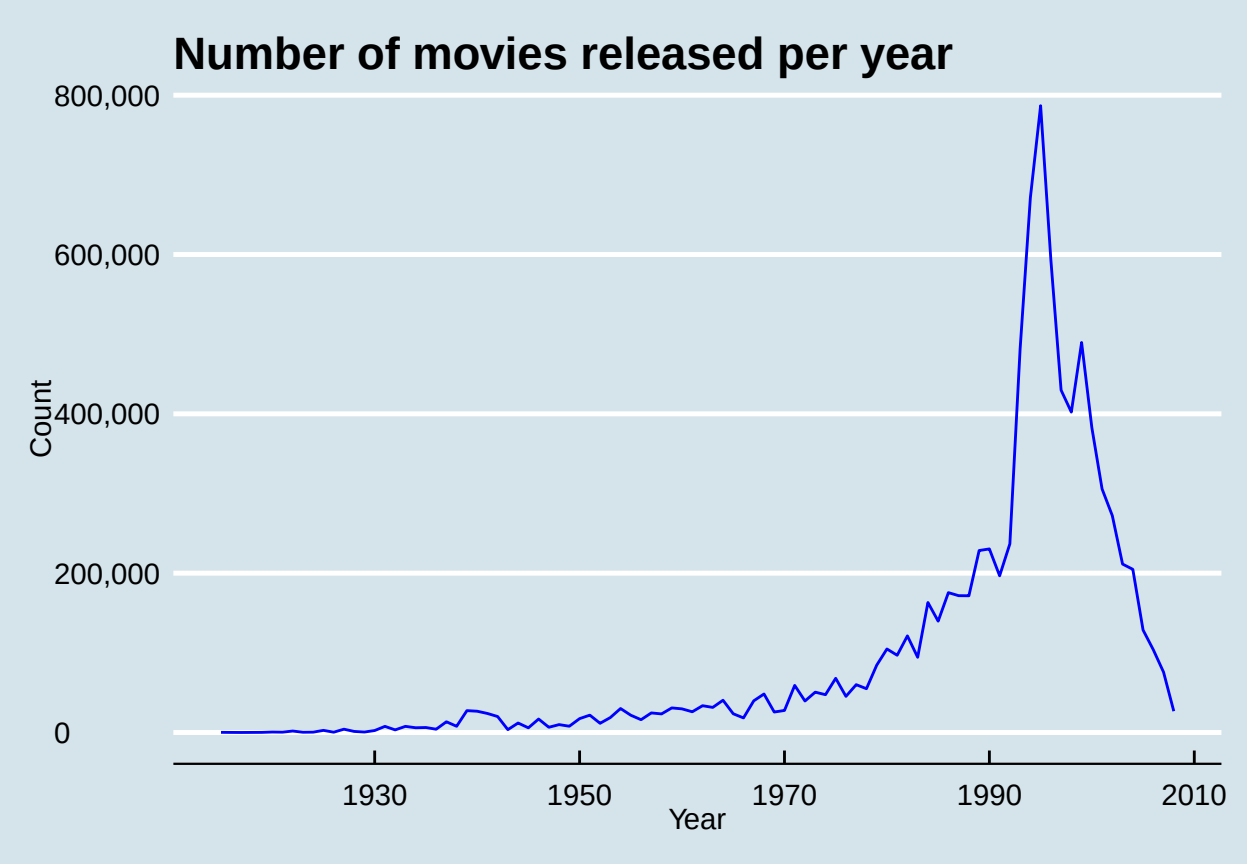
userId	movieId	rating	timestamp	title	year	genres
1	122	5	838985046	Boomerang	1992	Comedy|Romance
1	185	5	838983525	Net, The	1995	Action|Crime|Thriller
1	292	5	838983421	Outbreak	1995	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate	1994	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations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

```r
# Number of movies released per year
movies_per_year <- movies_df %>%
  na.omit() %>%
  select(movieId, year) %>%
  group_by(year) %>%
  summarise(count = n())  %>%
  arrange(year)

# Fill missing years
movies_per_year <- movies_per_year %>%
  complete(year = full_seq(year, 1), fill = list(count = 0))
print(movies_per_year)
```

```
# # A tibble: 94 x 2
#     year count
#    <dbl> <dbl>
#  1  1915   180
#  2  1916    84
#  3  1917    32
#  4  1918    73
#  5  1919   158
#  6  1920   575
#  7  1921   406
#  8  1922  1825
#  9  1923   316
# 10  1924   457
# # ... with 84 more rows
```

```r
# Plot - Number of movies released per year
movies_per_year %>%
  ggplot(aes(x = year, y = count)) +
  geom_line(color="blue")+
  labs(x="Year",y="Count", title="Number of movies released per year")+
  scale_y_continuous(breaks = c(0,200000,400000,600000,800000),
                     labels=c("0","200,000","400,000","600,000","800,000"))+
  theme_economist()
```



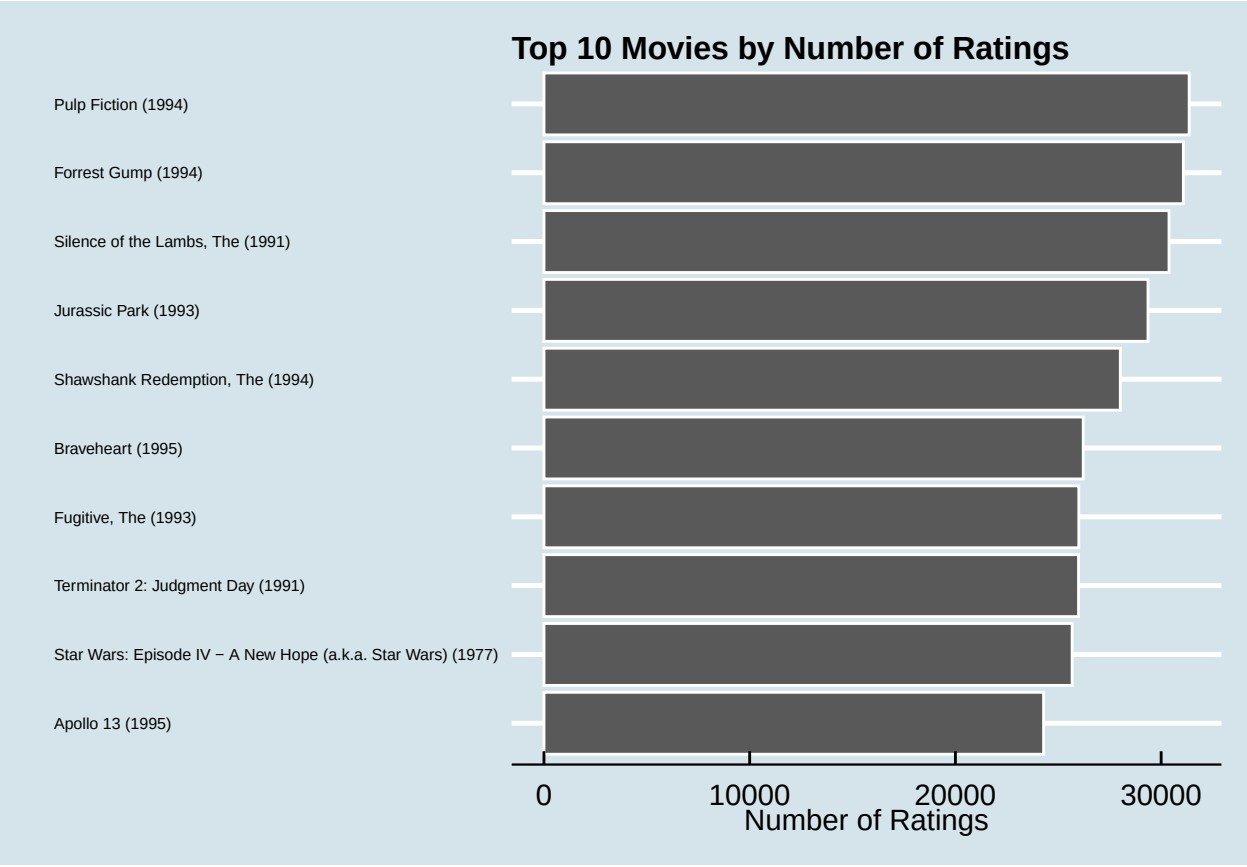
2.2.3 Titles

Related to the section above, the title column gives us further information about the movies in the edx dataset. Here we look at the top 10 movie titles by the number of ratings.

```
# Top 10 movie titles by number of ratings
edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  top_n(10,count) %>%
  arrange(desc(count))
```

title	count
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015
Braveheart (1995)	26212
Fugitive, The (1993)	25998
Terminator 2: Judgment Day (1991)	25984
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
Apollo 13 (1995)	24284

```
# Plot top 10 movies by number of ratings
edx %>%
  group_by(title) %>%
  summarize(count=n()) %>%
  top_n(10,count) %>%
  arrange(desc(count)) %>%
  ggplot(aes(x=reorder(title, count), y=count)) +
  geom_bar(stat='identity', color="white") +
  coord_flip()+
  labs(x="", y="Number of Ratings",
       title="Top 10 Movies by Number of Ratings")+
  theme_economist()+
  theme(axis.text.y = element_text(size=6),
        plot.title = element_text(size=12))
```



As shown previously, movies that received more ratings tend to be popular critics' choices or blockbusters.

2.2.4 Users

We start analyzing user information by first counting the number of distinct users in the edx dataset.

```
# Number of users in edx dataset
n_distinct(edx$userId)
```

```
# [1] 69878
```

There are 69,878 unique users in the edx dataset and we suspect that some users are more active than others (i.e. some users rate more movies than others).

```
# Number of movies rated per user
edx %>%
  group_by(userId) %>%
  summarise(n=n())%>%
  head()
```

userId	n
1	19
2	17
3	31
4	35
5	74
6	39

```
# Some users rate very few movies
edx %>%
  group_by(userId) %>%
  summarise(n=n())%>%
  arrange(n) %>%
  head()
```

userId	n
62516	10
22170	12
15719	13
50608	13
901	14
1833	14

```
# Some users rate many movies
edx %>%
  group_by(userId) %>%
  summarise(n=n())%>%
  arrange(desc(n)) %>%
  head()
```

userId	n
59269	6616
67385	6360
14463	4648
68259	4036
27468	4023
19635	3771

```
# Average number of movies rated per user

userratings<-edx %>%
  group_by(userId) %>%
  summarise(n=n())

mean(userratings$n)
```

```
# [1] 128.7966885
```

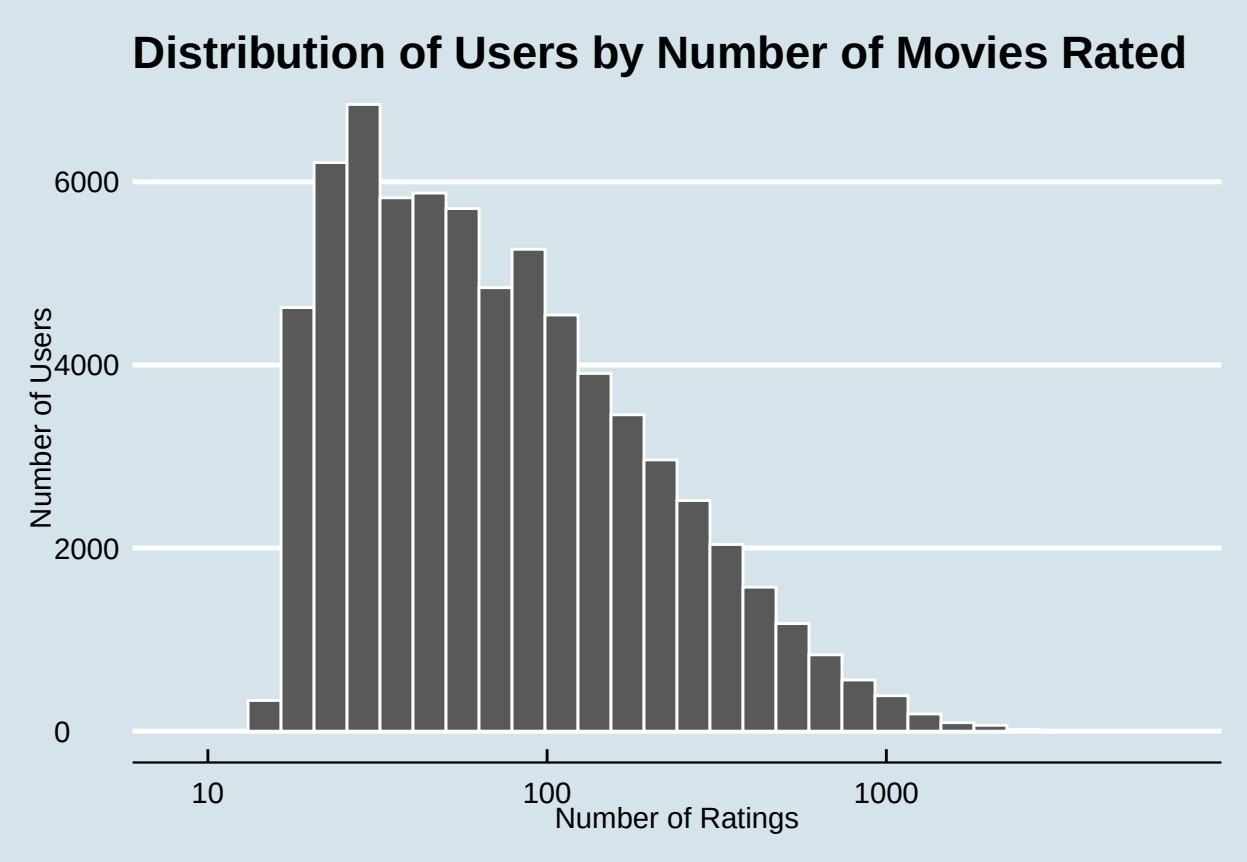
```
# Percentage of users who rated less than 20 movies

mean(userratings$n<20)
```

```
# [1] 0.04965797533
```

We can see that 5% of users rated less than 20 movies and some users rated over 3,000 films. The average number of movies rated per user is around 129.

```
# Distribution of Users by the number of movies they rated
edx %>%
  dplyr::count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "white") +
  scale_x_log10() +
  labs(x="Number of Ratings", y="Number of Users",
       title="Distribution of Users by Number of Movies Rated")+
  theme_economist()
```

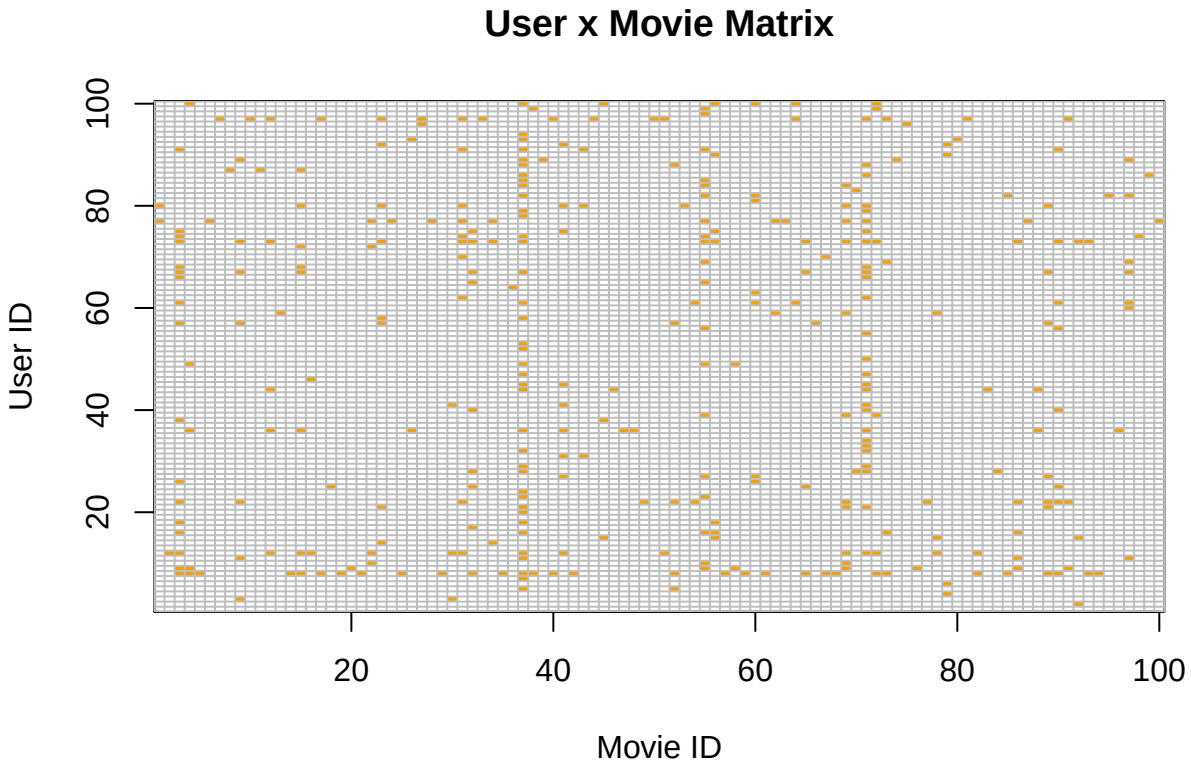


From the plot we can see that the distribution is right skewed.

Next, we look at the heatmap of users vs movies.

```r
# Heatmap of users vs movies
users <- sample(unique(edx$userId), 100)

edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>%
  select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movie ID", ylab="User ID")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
title("User x Movie Matrix")
```



We can see that the user x movie matrix is sparse, with many empty cells. The goal of our recommendation system is to predict the ratings in the empty cells.

2.2.5 Timestamp

The timestamp column documents the date and time of each rating in seconds. We can use the lubridate package to convert the data into a more readable format – date (mm/dd/yyyy) and time (HH:MM:SS). We will analyze the timestamp column to check whether there any temporal effects on ratings.

```r
# Convert timestamp to more readable format
timestamp_df<-edx%>%
```

```
  mutate(timestamp = as_datetime(timestamp))
head(timestamp_df)
```

userId	movieId	rating	timestamp	title	genres
1	122	5	1996-08-02 11:24:06	Boomerang (1992)	Comedy|Romance
1	185	5	1996-08-02 10:58:45	Net, The (1995)	Action|Crime|Thriller
1	292	5	1996-08-02 10:57:01	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	1996-08-02 10:56:32	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	1996-08-02 10:56:32	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	1996-08-02 11:14:34	Flintstones, The (1994)	Children|Comedy|Fantasy

```
# Rating period covered in dataset
start_year<-lubridate::year(min(timestamp_df$timestamp))
start_year
```

```
# [1] 1995
```

```
end_year<-lubridate::year(max(timestamp_df$timestamp))
end_year
```

```
# [1] 2009
```

```
rating_period<-tibble(`Start Date` = start_year,
                      `End Date` = end_year) %>%
               mutate(Period = end_year-start_year)
rating_period
```

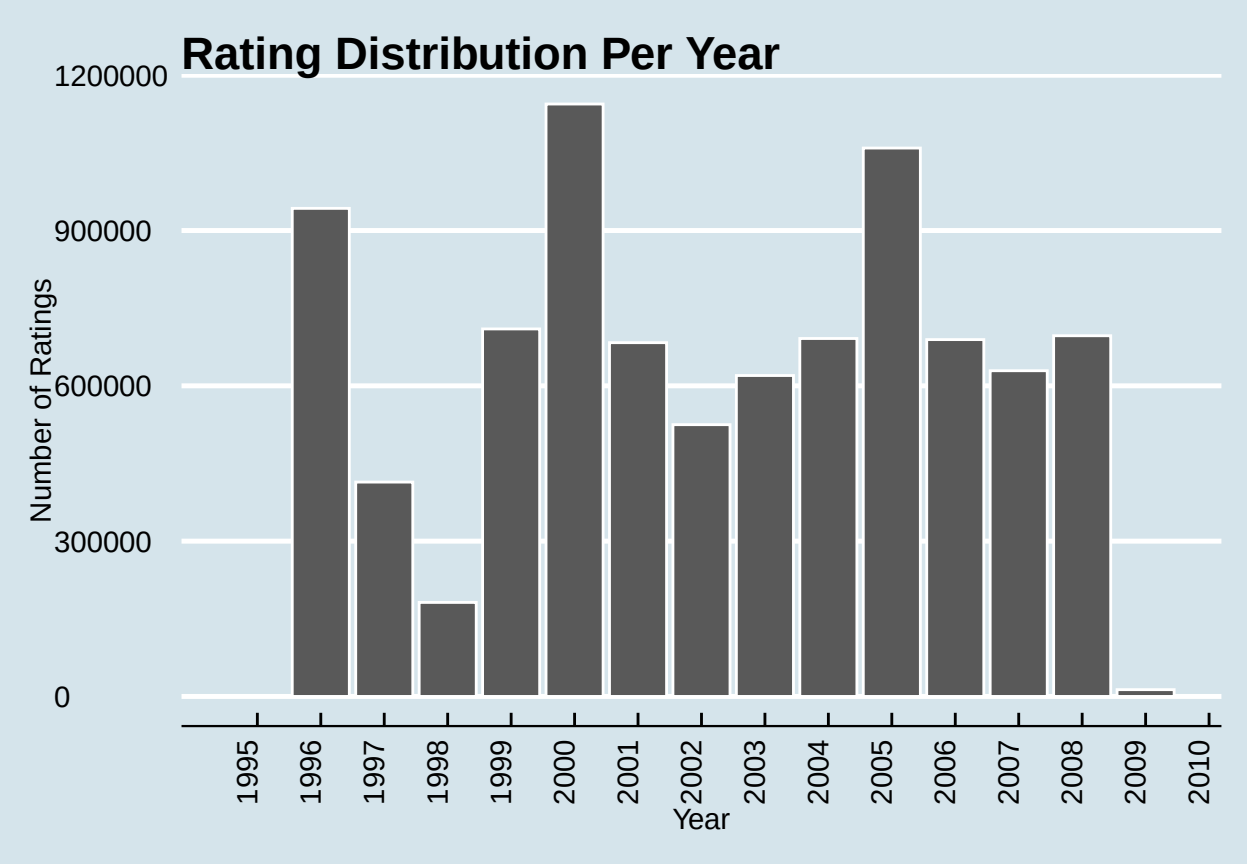
Start Date	End Date	Period
1995	2009	14

The total rating period (since the first recorded rating to the last recorded) of the dataset is approximately 14 years.

We can look at the distribution of ratings per year to check whether the number of ratings changes over time.

```
# Plot Rating Distribution Per Year (number of ratings per year)

edx %>% mutate(year = lubridate::year(as_datetime(timestamp))) %>%
  ggplot(aes(x=year)) +
  geom_bar(color = "white") +
  scale_x_continuous(breaks=1995:2010)+
  labs(x="Year", y="Number of Ratings",
       title="Rating Distribution Per Year")+
  theme_economist()+
  theme(axis.text.x = element_text(angle = 90))
```



We can see from the plot that the number of ratings has fluctuated over the years. This could be due to record-breaking blockbuster films that were released in certain years.

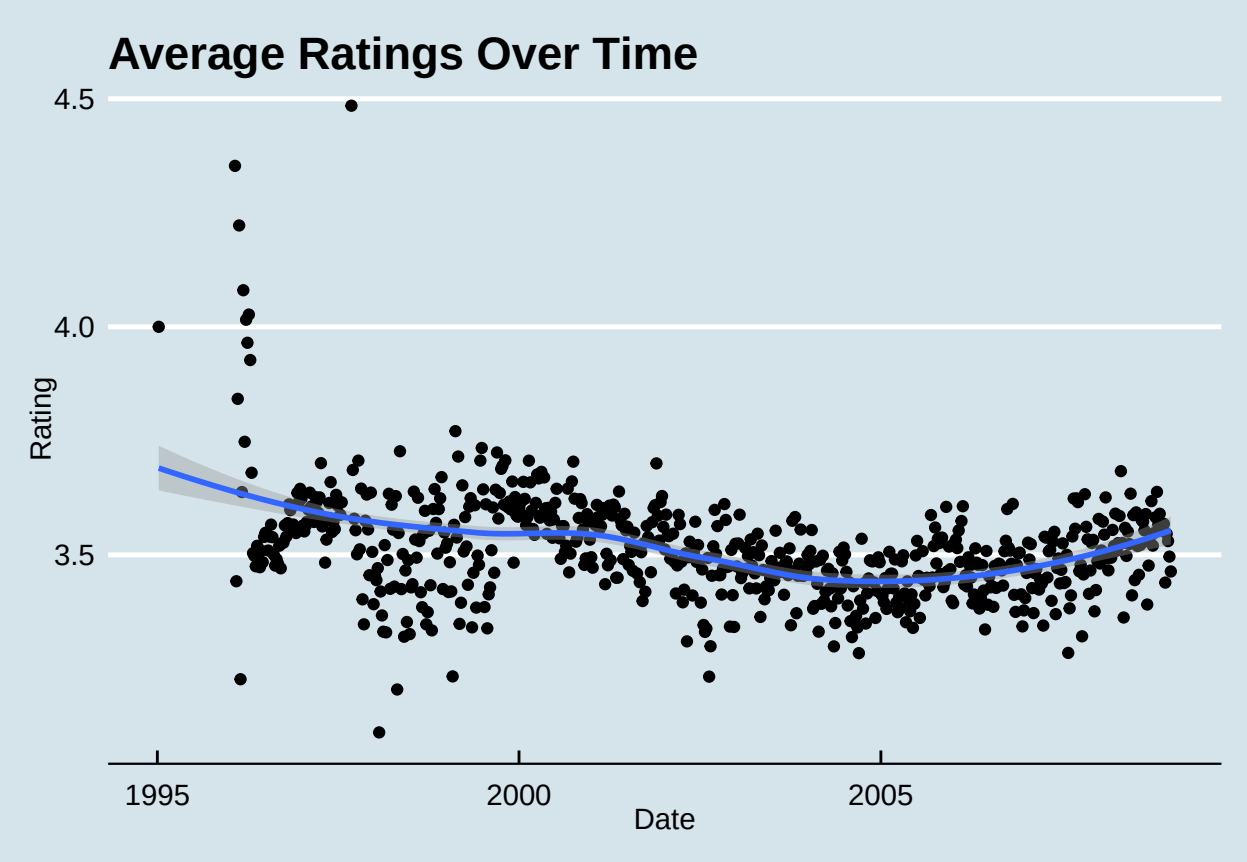
```
# list of dates with the most ratings - Blockbuster/ Popular films
edx %>% mutate(date = date(as_datetime(timestamp))) %>%
  group_by(date, title) %>%
  summarise(count = n()) %>%
  arrange(-count) %>%
  head(10)
```

date	title	count
1998-05-22	Chasing Amy (1997)	322
2000-11-20	American Beauty (1999)	277
1999-12-11	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	254
1999-12-11	Star Wars: Episode V - The Empire Strikes Back (1980)	251
1999-12-11	Star Wars: Episode VI - Return of the Jedi (1983)	241
2005-03-22	Lord of the Rings: The Two Towers, The (2002)	239
2005-03-22	Lord of the Rings: The Fellowship of the Ring, The (2001)	227
2000-11-20	Terminator 2: Judgment Day (1991)	221
1999-12-11	Matrix, The (1999)	210
2000-11-20	Jurassic Park (1993)	201

When we look at the list of dates with the most ratings, we can see that these in fact coincide with very popular/ blockbuster releases.

We can also look at average ratings over time and see whether there are any notable trends.

```
# Plot average ratings over time
edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating))%>%
  ggplot(aes(date,rating))+
  geom_point()+
  geom_smooth()+
  labs(x="Date", y="Rating", title="Average Ratings Over Time")+
  theme_economist()
```



There seems to be some evidence of a time effect although it is not a strong effect.

Finally, we look at the best films of each decade, based on their weighted rating. The weighted rating function sets the minimum number of ratings to 250 in order to exclude movies with very few ratings.

```
# average rating per movie
avg_rating <- movies_df%>%
  na.omit() %>%
  select(title, rating, year) %>%
  group_by(title, year) %>%
  summarise(count = n(), mean = mean(rating)) %>%
  ungroup() %>%
  arrange(desc(mean))
head(avg_rating)
```

title	year	count	mean
Blue Light, The (Das Blaue Licht)	1932	1	5
Fighting Elegy (Kenka erejii)	1966	1	5
Hellhounds on My Trail	1999	1	5
Satan's Tango (Sátántangó)	1994	2	5
Shadows of Forgotten Ancestors	1964	1	5
Sun Alley (Sonnenallee)	1999	1	5

```
# Calculate Weighted Rating

# MR = average rating for the movie = Mean Rating
# C = number of ratings for the movie = Count
# Min = minimum count required to be listed in the Top 250
# OMR = the mean rating across all movies = Overall Mean Rating

weighted_rating <- function(MR, C, Min, OMR) {
  return (C/(C+Min))*MR + (Min/(C+Min))*OMR
}

avg_rating <- avg_rating %>%
  mutate(wr = weighted_rating(mean, count, 500, mean(mean))) %>%
  arrange(desc(wr)) %>%
  select(title, year, count, mean, wr)
head(avg_rating)
```

title	year	count	mean	wr
Pulp Fiction	1994	31362	4.154789235	0.9843073253
Forrest Gump	1994	31079	4.012822163	0.9841666931
Silence of the Lambs, The	1991	30382	4.204101113	0.9838093388
Jurassic Park	1993	29360	3.663521798	0.9832551909
Shawshank Redemption, The	1994	28015	4.455131180	0.9824653691
Braveheart	1995	26212	4.081851824	0.9812818209

```
# best movie of every decade based on score
avg_rating %>%
  mutate(decade = year  %/% 10 * 10) %>%
  arrange(year, desc(wr)) %>%
  group_by(decade) %>%
  summarise(title = first(title), wr = first(wr),
            mean = first(mean), count = first(count))
```

decade	title	wr	mean	count
1910	Birth of a Nation, The	0.2647058824	3.288888889	180
1920	Cabinet of Dr. Caligari, The (Das Cabinet des Dr. Caligari.)	0.5206136146	3.980662983	543
1930	All Quiet on the Western Front	0.7198879552	4.030739300	1285
1940	Pinocchio	0.9274521184	3.529020651	6392
1950	Cinderella	0.8940902351	3.603529969	4221
1960	Psycho	0.9494643218	4.091281669	9394
1970	M*A*S*H (a.k.a. MASH)	0.9315349856	3.922975158	6803
1980	Star Wars: Episode V - The Empire Strikes Back	0.9764473126	4.192918134	20729
1990	Dances with Wolves	0.9790505719	3.742628493	23367
2000	Gladiator	0.9653523664	3.936975091	13931

As expected, these are all well-known movies.

2.2.6 Genres

According to our intuition, some genres are more popular than others and therefore receive a higher number of ratings. Furthermore, movies in certain genres also receive higher average ratings due to critics' reviews. The following analysis will verify whether our intuition is indeed correct.

We will do this by creating a separate data frame which contains each genre on a separate row, instead of in combinations (e.g. "Comedy" instead of "Action|Comedy|Thriller").

```
# number of movies per genre
edx %>% group_by(genres) %>%
  summarise(n=n()) %>%
  head()
```

genres	n
(no genres listed)	7
Action	24482
Action|Adventure	68688
Action|Adventure|Animation|Children|Comedy	7467
Action|Adventure|Animation|Children|Comedy|Fantasy	187
Action|Adventure|Animation|Children|Comedy|IMAX	66

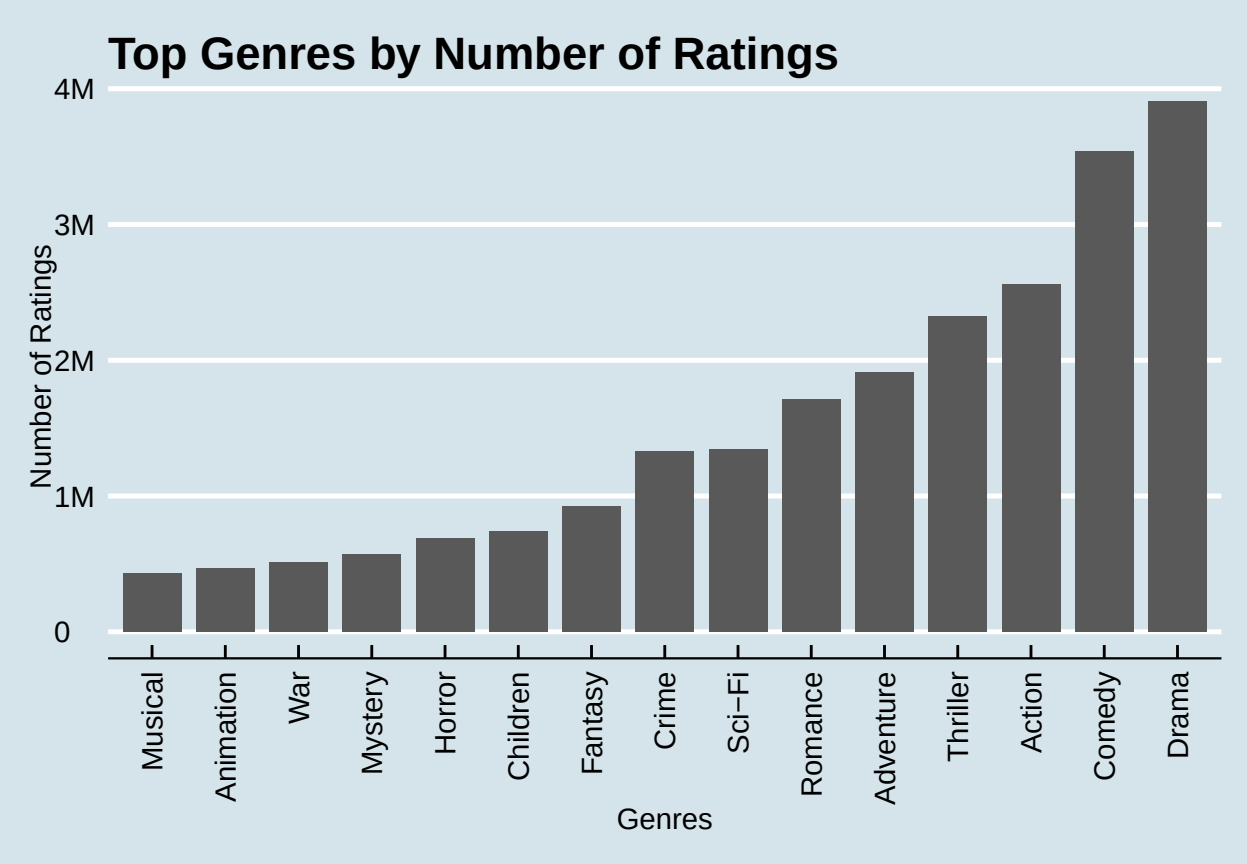
```
# split genres column into separate rows
genre_df <- movies_df %>% separate_rows(genres, sep = "\\|")
head(genre_df)
```

userId	movieId	rating	timestamp	title	year	genres
1	122	5	838985046	Boomerang	1992	Comedy
1	122	5	838985046	Boomerang	1992	Romance
1	185	5	838983525	Net, The	1995	Action
1	185	5	838983525	Net, The	1995	Crime
1	185	5	838983525	Net, The	1995	Thriller
1	292	5	838983421	Outbreak	1995	Action

```
# Top 15 genres by number of ratings
genre_df%>%group_by(genres)%>%
  summarize(count=n())%>%
  top_n(15,count) %>%
  arrange(desc(count))
```

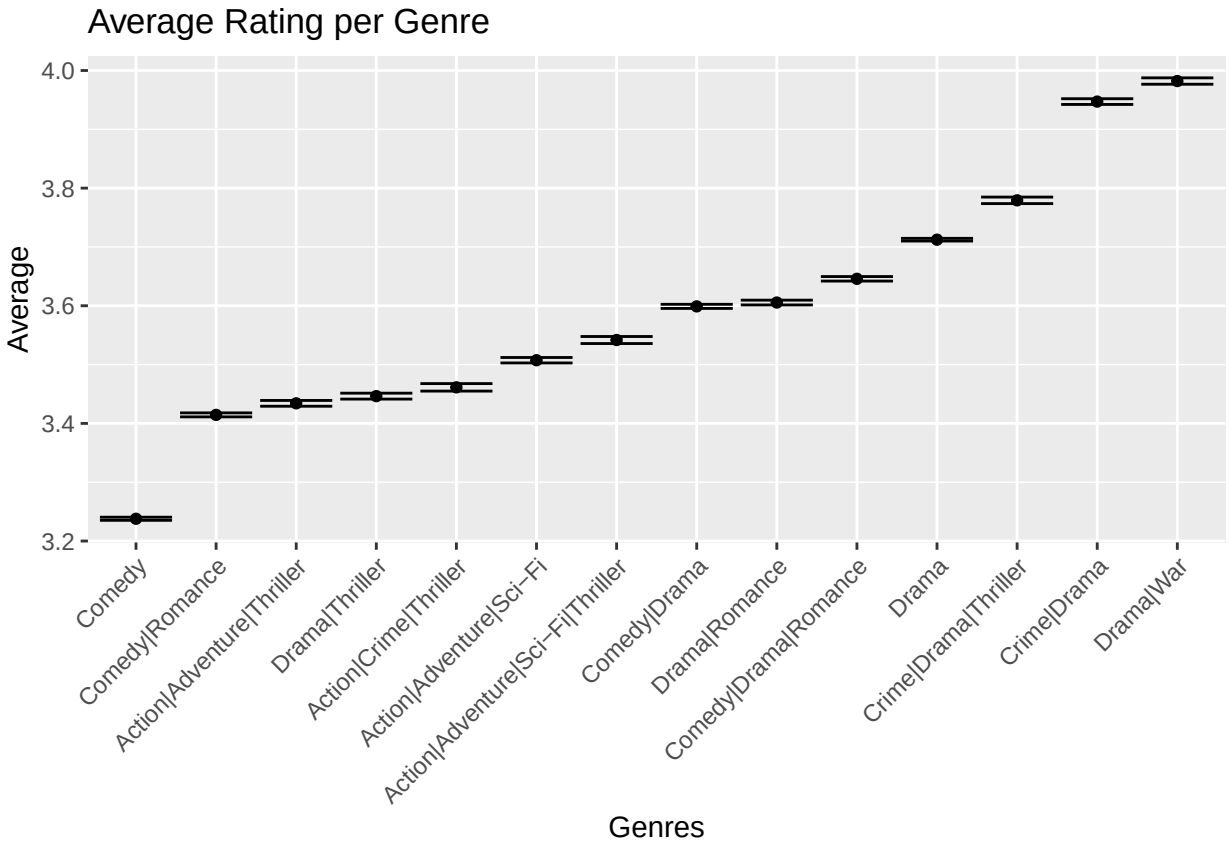
genres	count
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994
Horror	691485
Mystery	568332
War	511147
Animation	467168
Musical	433080

```
# Plot - Ratings Distribution by Genre (number of ratings per genre)
genre_df%>%group_by(genres)%>%
  summarize(count=n())%>%
  top_n(15,count) %>%
  ggplot()+
  geom_bar(aes(x=reorder((factor(genres)),count), y=count),
           stat="identity", width=0.8) +
  labs(x="Genres", y="Number of Ratings",
       title="Top Genres by Number of Ratings") +
  scale_y_continuous(breaks = c(0,1000000,2000000,3000000,4000000),
                     labels=c("0","1M","2M","3M","4M"))+
  theme_economist()+
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



We can in fact see that some genres received a higher number of ratings than others, with Drama, Comedy, and Action being the most popular.

```
# Plot - Average rating per genre
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 100000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(x="Genres",y="Average", title="Average Rating per Genre")
```

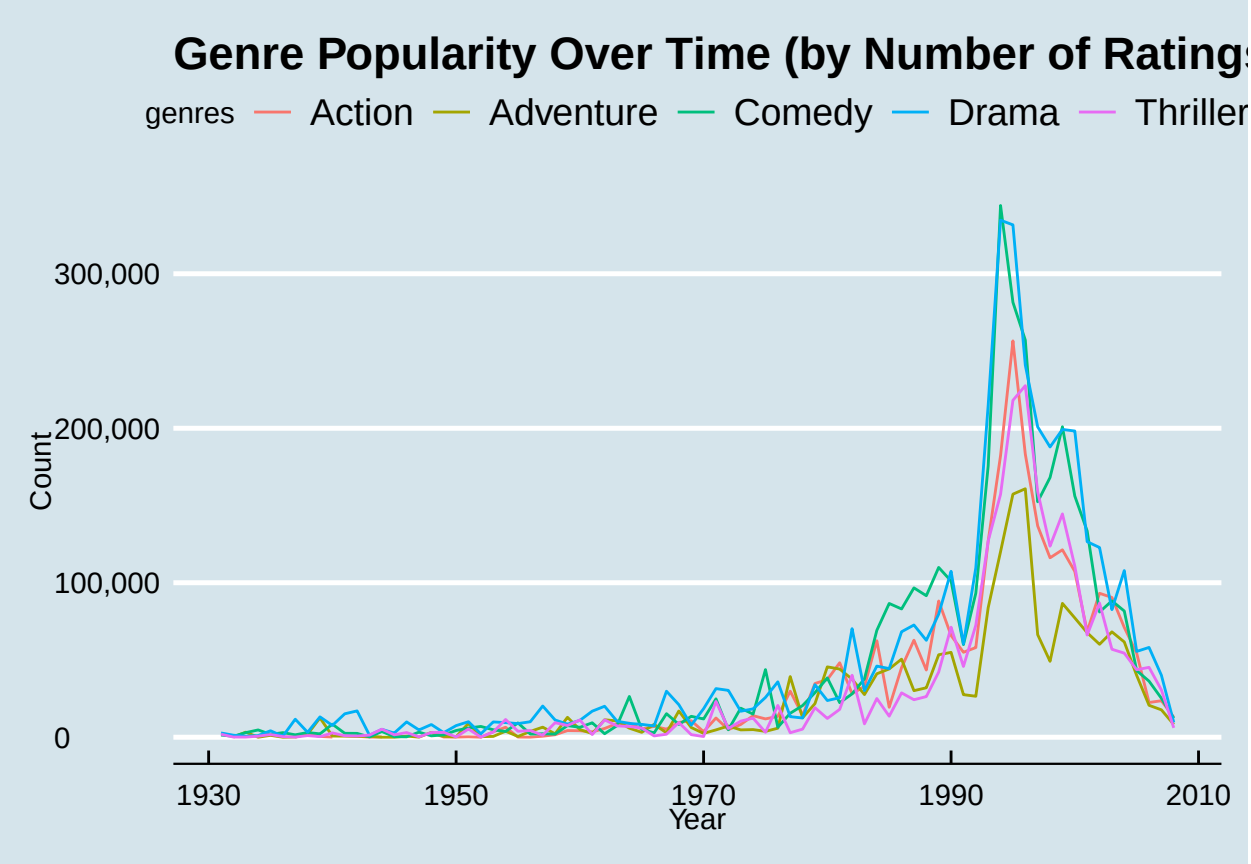


From the plot above we can also see that some genres and genre combinations tend to be rated higher than others, which shows strong evidence of a genre effect.

Since we have timestamp data, we can also analyze genre popularity and genre performance over time. Specifically, we will select the 5 most popular genres (for readability purposes) and check whether their popularity (number of ratings) and their average ratings have changed over the years.

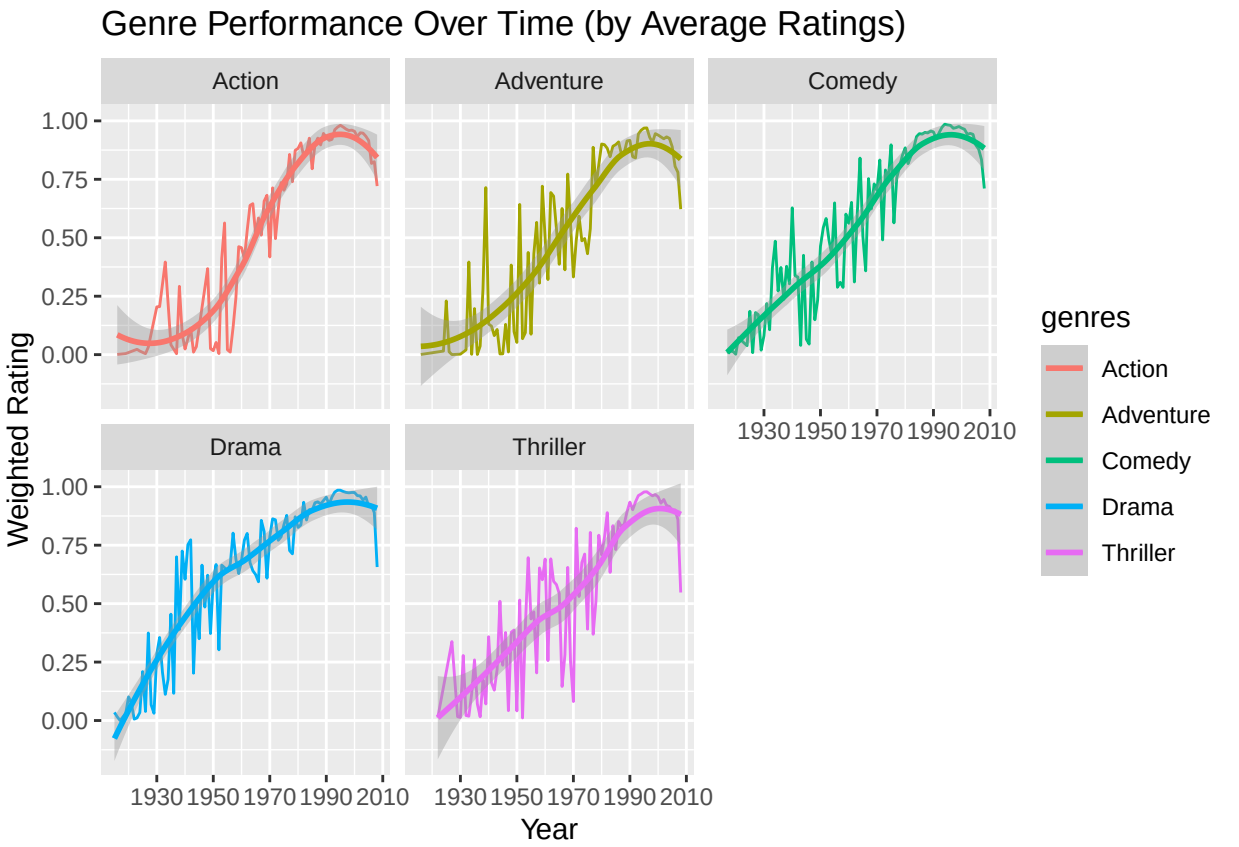
```
# Genre popularity over time
genre_popularity <- genre_df %>%
  na.omit() %>%
  select(movieId, year, genres) %>%
  mutate(genres = as.factor(genres)) %>%
  group_by(year, genres) %>%
  summarise(count = n()) %>%
  complete(year = full_seq(year, 1), genres, fill = list(number = 0))

# Plot - genre popularity over time (select five for readability)
genre_popularity %>%
  filter(year > 1930) %>%
  filter(genres %in% c("Drama", "Comedy", "Action", "Thriller", "Adventure")) %>%
  ggplot(aes(x = year, y = count)) +
  geom_line(aes(color=genres)) +
  scale_fill_brewer(palette = "Paired")+
  scale_y_continuous(breaks = c(0,100000,200000,300000,400000),
                     labels=c("0","100,000","200,000","300,000","400,000"))+
  labs(x="Year",y="Count", title="Genre Popularity Over Time (by Number of Ratings)")+
  theme_economist()
```




```
# Genre performance over time (as per user ratings)
genre_ratings <- genre_df %>%
  na.omit() %>%
  select(movieId, year, genres, rating) %>%
  mutate(decade = year  %/% 10 * 10) %>%
  group_by(year, genres) %>%
  summarise(count = n(), avg_rating = mean(rating)) %>%
  ungroup() %>%
  mutate(wr = weighted_rating(mean, count, 5000, mean(mean))) %>%
  arrange(year)

# Plot - Genre performance over time (as per user ratings)
genre_ratings %>%
  filter(genres %in% c("Drama", "Comedy", "Action", "Thriller",
                       "Adventure")) %>%
  ggplot(aes(x = year, y = wr)) +
  geom_line(aes(group=genres, color=genres)) +
  geom_smooth(aes(group=genres, color=genres)) +
  facet_wrap(~genres)+
  labs(x="Year", y="Weighted Rating",
       title="Genre Performance Over Time (by Average Ratings)")
```



Once again we can see that there seems to be a strong genre effect.

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. As we've seen through our data exploration process, several features (users, movies, genre, etc.) can be used to predict the rating a user will give a particular movie.

However, including many predictors increases the complexity of our models and therefore require more computer resources. Therefore, for the purposes of this project, we will focus only on the movie and user effects.

```
# Data Cleaning - Select the most relevant features
train_set<-train_set%>%select(userId, movieId, rating, title)
test_set<-test_set%>%select(userId, movieId, rating, title)
```

3. Modeling Methods

3.1 Linear Model - Naive Approach

We will start by building the simplest recommendation system: predicting the same rating for all movies, regardless of the user. We can use a model based approach to find out what this number should be. A model that assumes that all users will give the same rating to all movies with all the differences explained by random variation would be 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. According to statistical theory, the average minimizes the RMSE, so the initial prediction is the average of all observed ratings. Any other value would increase the RMSE so this is a good starting point, but we can definitely do better with other models.

3.2 Linear Model with Movie & User Effects

Movie Effect

We know from experience that different movies are rated differently as some are more popular than others and user preference varies. This intuition was confirmed during our data analysis. We call this the movie effect, or bias, and denote it as b_i and add it to our model:

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

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

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

User Effect

As we saw from our data analysis, similar to the movie effect, there is also variability from user to user in how they rate movies. For instance, some users are more cranky than others and tend to give lower ratings than the average. We can calculate this user effect, or user bias, as follows:

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

And add it to our model to improve our predictions:

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

Although we detected evidence of a genre effect during our data analysis, for the purposes of this project, we will not be including this in our model. As mentioned previously, including too many predictors increases our model's complexity.

3.3 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:

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

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

This is the formula we will be minimizing:

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

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.

3.4 Matrix Factorization

Matrix Factorization is a powerful tool often used in machine learning. It was popularized by the winners of the Netflix challenge[12]. It is related to factor analysis, singular value decomposition (SVD), and principal component analysis (PCA).

As outlined previously, our linear model accounts for movie effects through b_i and user effects through b_u as follows:

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

However, it doesn't taken into account that groups of movies have similar rating patterns as do groups of users. This is an important source of variation that we can calculate by analyzing the residuals:

$$r_{u,i} = y_{u,i} - \hat{b}_i - \hat{b}_u$$

We will convert the data into a matrix so that each user is in a row, each movie in a column, and the rating $y_{u,i}$ is the cell in row u and column i. This is an example of a **user x movie** matrix:

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5
User 1	?	3	2	?	5
User 2	3	4	?	?	2
User 3	5	?	2	?	3
User 4	4	?	?	3	?

The general idea is to decompose the large user-movie matrix R_{mxn} into two lower dimensionality matrices P_{kxm} and Q_{kxn}, such that:

$$R \sim P'Q$$

[12] https://www.netflixprize.com/

Matrix factorization simplifies matrix operations by allowing us to perform on the decomposed matrices rather than the original matrix itself[13]. It is used to discover "latent features between two entities"[14].

The R recosystem package[15] allows us to perform parallel matrix factorization in order to decompose the rating matrix and predict the ratings.

4. Results

4.1 Model Evaluation Functions

We start off by defining our loss function – Root Mean Squared Error.

```
# 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)
```

Method	RMSE
Project Goal	0.8649

4.2 Linear Model - Naive Approach

First, we will build the simplest recommender model, which is just the average of all the ratings:

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

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

```
## [1] 3.512455641
```

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

```
## [1] 1.060053702
```

[13] https://machinelearningmastery.com/introduction-to-matrix-decompositions-for-machine-learning/
[14] https://medium.com/@paritosh_30025/recommendation-using-matrix-factorization-5223a8ee1f4
[15] https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

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

knitr::kable(rmse_results, digits=6) %>% kable_styling()
```

Method	RMSE
Project Goal	0.864900
Naive RMSE	1.060054

4.3 Linear Model with Movie Effect

Next we add the movie effect to our model:

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

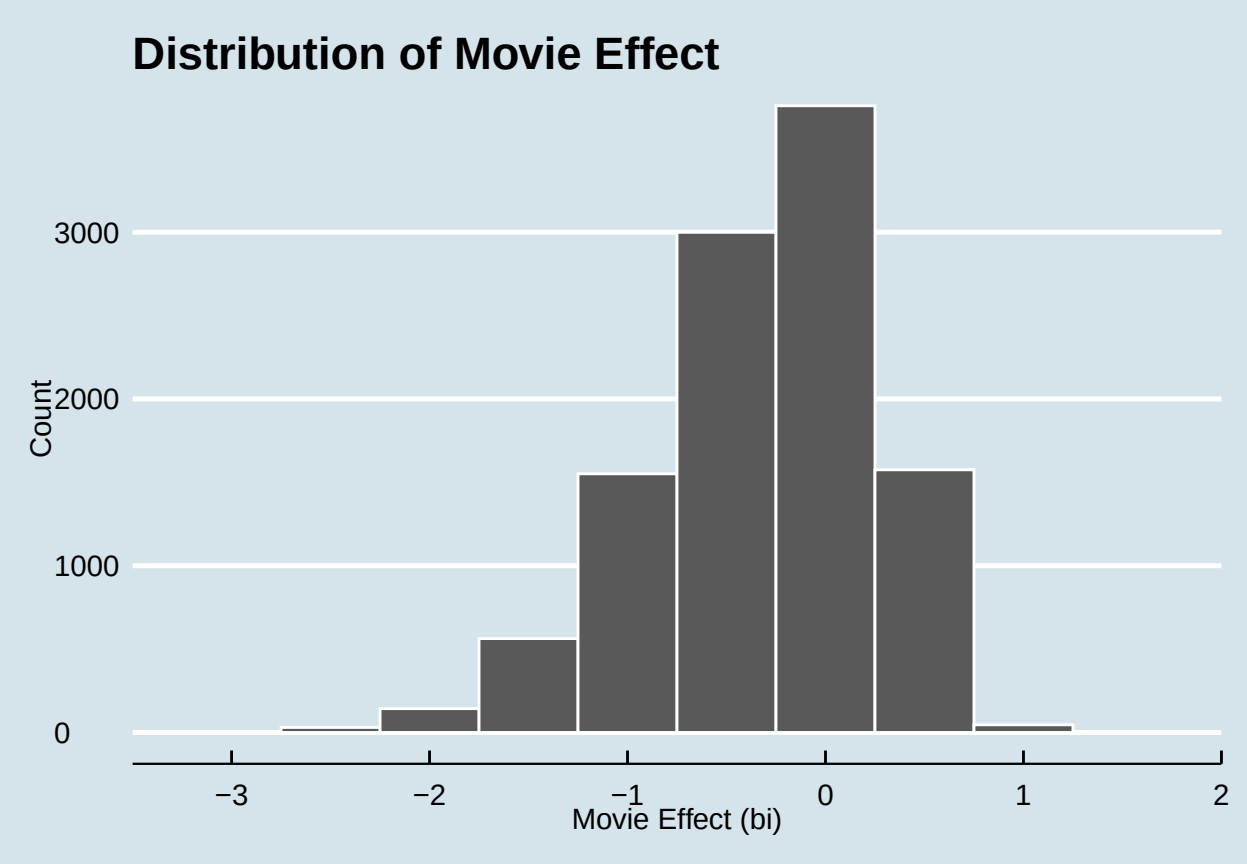
We calculate the movie effect b_i as follows:

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

movieId	b_i
1	0.4150040461
2	-0.3064057081
3	-0.3613951537
4	-0.6372808156
5	-0.4416057873
6	0.3018943367

As evidenced from the plot below, the distribution of the movie effect is left skewed.

```
# Plot - Distribution of Movie Effect (bi)
bi %>% ggplot(aes(x = b_i)) +
  geom_histogram(bins=10, col = I("white")) +
  labs(x="Movie Effect (bi)", y="Count",
       title="Distribution of Movie Effect")+
  theme_economist()
```




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

RMSE(test_set$rating, y_hat_bi)
```

```
## [1] 0.942961498
```

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

knitr::kable(rmse_results, digits=6)%>% kable_styling()
```

Method	RMSE
Project Goal	0.864900
Naive RMSE	1.060054
Movie Effect	0.942961

We can see from the results table that our RMSE has improved compared to the original model which took just the average.

4.4 Linear Model with Movie & User Effect

We now also include the user effect b_u in our model to check whether this improves our RMSE:

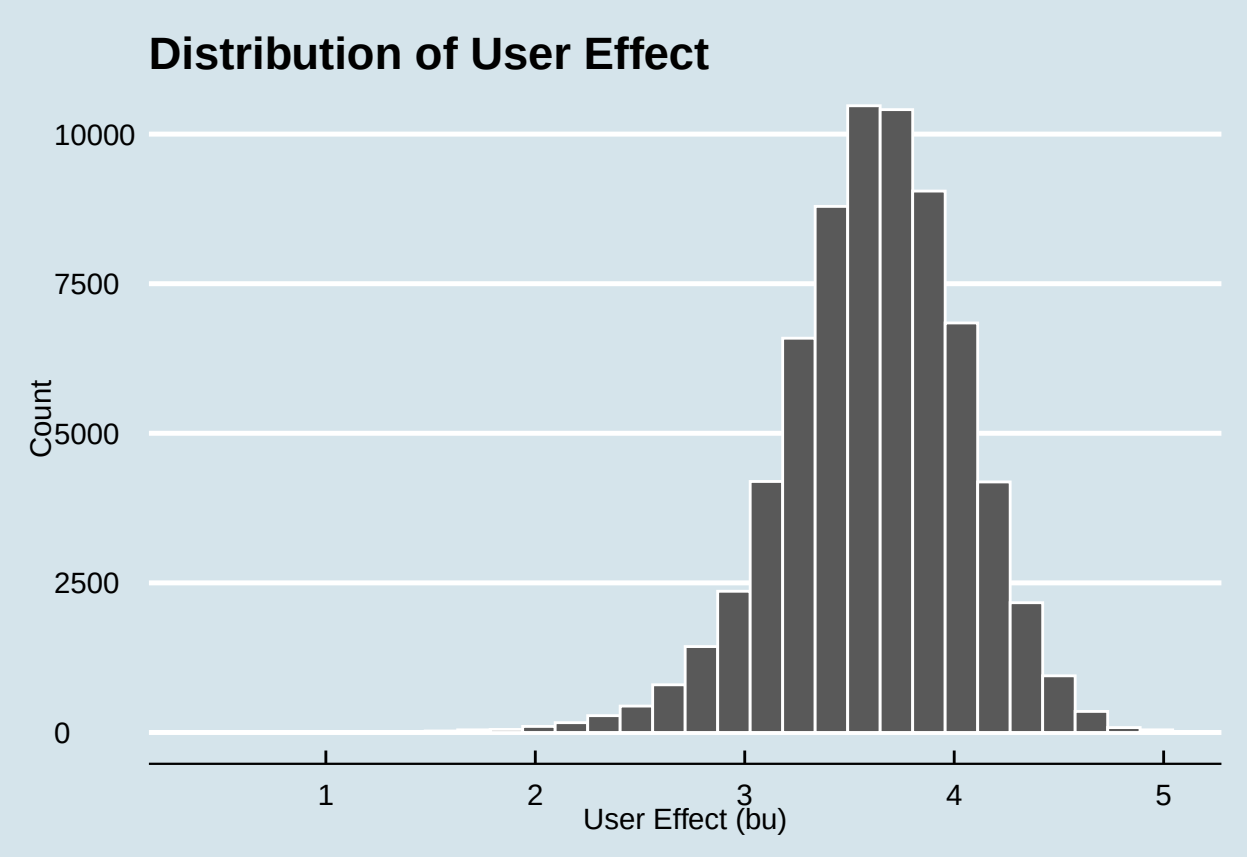
$$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

```
# Calculate user effect

bu <- train_set %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

As we can see from the plot below, the user effect is normally distributed.

```
# Plot - User Effect Distribution
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "white")+
  labs(x="User Effect (bu)", y="Count",
  title="Distribution of User Effect")+
  theme_economist()
```



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

RMSE(test_set$rating, y_hat_bu)
```

```
## [1] 0.8646842949
```

```r
# 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_bu)))
```

Method	RMSE
Project Goal	0.864900
Naive RMSE	1.060054
Movie Effect	0.942961
Movie + User Effect	0.864684

Model Evaluation

Including the user effect further minimizes the RMSE, proving that this is a better model than our initial prediction using just the mean. However, we still need to make sure the model makes sound predictions. Although there is substantial movie to movie variation, our RMSE only improved by about 11%.

We can check where we made mistakes by looking at the 10 largest errors, using only the movie effect b_i.

```r
# Evaluate Model Results

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

userId	movieId	rating	title	b_i	residual
29924	6483	5.0	From Justin to Kelly (2003)	-2.6381387009	4.125683060
10680	318	0.5	Shawshank Redemption, The (1994)	0.9441109782	-3.956566619
51845	318	0.5	Shawshank Redemption, The (1994)	0.9441109782	-3.956566619
39975	858	0.5	Godfather, The (1972)	0.9041953764	-3.916651017
49231	858	0.5	Godfather, The (1972)	0.9041953764	-3.916651017
58785	858	0.5	Godfather, The (1972)	0.9041953764	-3.916651017
70446	858	0.5	Godfather, The (1972)	0.9041953764	-3.916651017
62012	50	0.5	Usual Suspects, The (1995)	0.8540962429	-3.866551884
17017	527	0.5	Schindler's List (1993)	0.8516292373	-3.864084878
25397	527	0.5	Schindler's List (1993)	0.8516292373	-3.864084878

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 bi
bi %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  head(10)  %>%
  select(title)
```

title
Hellhounds on My Trail (1999)
Satan's Tango (Sátántangó) (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

```
# 10 worst movies according to bi
bi %>% 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 all seem to be quite obscure films.

```
# number of ratings for "10 best movies according to bi"
train_set %>% count(movieId) %>%
  left_join(bi, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  head(10) %>%
  select(n, title)
```

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

```
# number of ratings for "10 worst movies according to bi"
train_set %>% count(movieId) %>%
  left_join(bi, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  head(10) %>%
  select(n, title)
```

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

When we look at how often these movies were rated, we can see that they were rated by very few users. Smaller sample sizes produce greater uncertainty and therefore larger estimates of the movie effect become more likely.

4.5 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.

```
# Regularization #####

# 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)
```

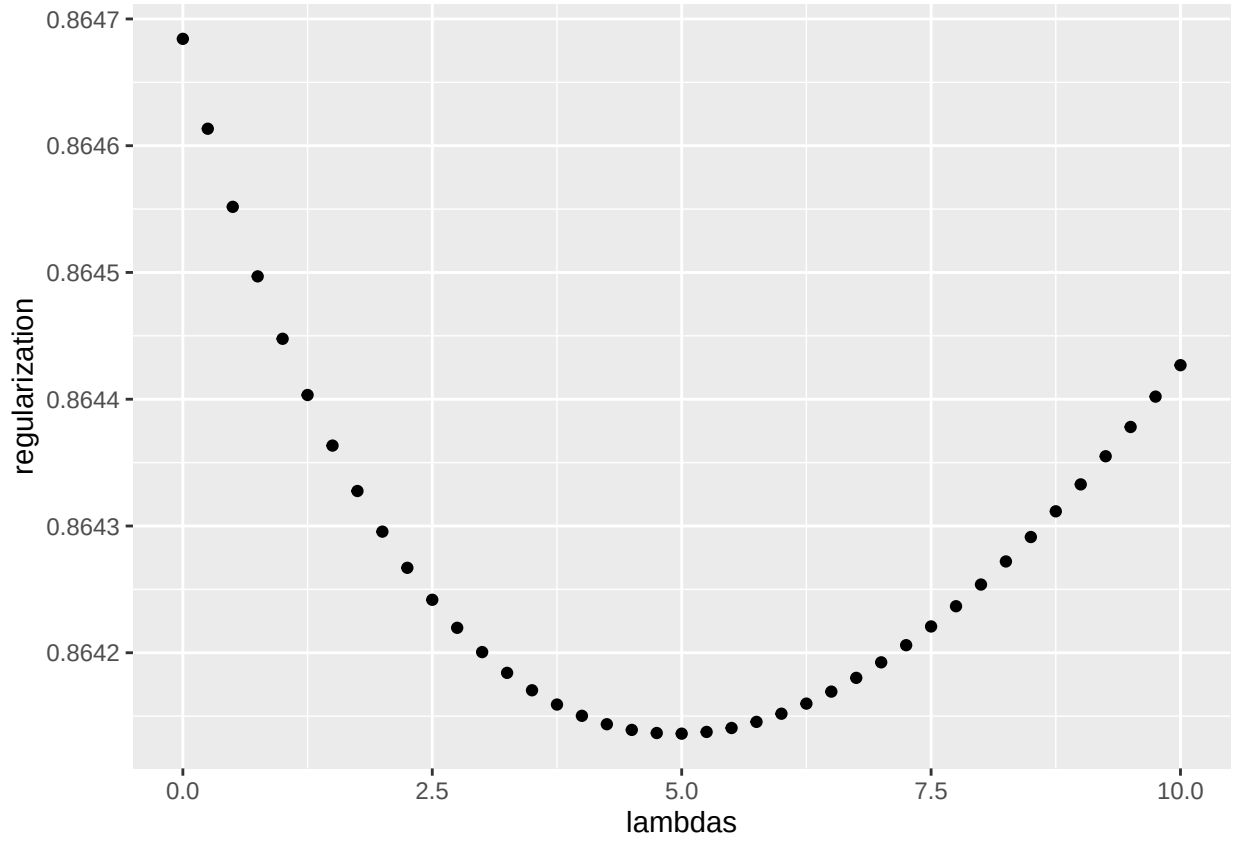
```
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

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

  predicted_ratings <-
    test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    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] 5
```

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))

# Prediction with regularized bi and bu
y_hat_reg <- test_set %>%
  left_join(bi_reg, by = "movieId") %>%
  left_join(bu_reg, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

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

Method	RMSE
Project Goal	0.864900
Naive RMSE	1.060054
Movie Effect	0.942961
Movie + User Effect	0.864684
Regularized Movie + User Effect	0.864136

Model Evaluation

Regularization of the movie and user 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") %>%
  mutate(pred = mu + b_i + b_u) %>%
  arrange(desc(pred)) %>%
  group_by(title) %>%
```

```
  select(title) %>%
  head(10)
```

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

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)
Pokémon the Movie 2000 (2000)

4.6 Matrix Factorization

As detailed previously, matrix factorization decomposes the large sparse user-movie matrix into two lower dimensional matrices. The information in our training set is in tidy format so it needs to be converted into a user-movie matrix where each row corresponds to a given user, the movies are in the columns, and the ratings are in the cells. We can achieve it using this code:

```
# Convert training set into user-movie matrix

# Example code - Code not run as it takes up too much memory)

train_matrix <- train_set %>%
```

```
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
head(train_matrix)
```

We will not run this code as it uses up too much memory and therefore can't be executed on a standard laptop. Instead we will be using the R recosystem package which provides a simple way to build a recommender system using matrix factorization.

We run the following code based on the instructions in the recosystem package vignette[16].

```
# use recosystem package instead
set.seed(123,sample.kind="Rounding") # randomized

# Convert the training and test sets into recosystem format
train_data <-  with(train_set, data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))

test_data  <-  with(test_set,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = rating))

# Create the model object
r <-  recosystem::Reco()

# Select the tuning parameters
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       lrate = c(0.01, 0.1),
                                       nthread = 4,
                                       niter = 10))

# Train the algorithm
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 10))
```

```
## iter      tr_rmse          obj
##    0       0.9826   1.1047e+07
##    1       0.8752   8.9855e+06
##    2       0.8428   8.3452e+06
##    3       0.8208   7.9566e+06
##    4       0.8045   7.6933e+06
##    5       0.7921   7.4998e+06
##    6       0.7819   7.3557e+06
##    7       0.7733   7.2418e+06
##    8       0.7659   7.1445e+06
##    9       0.7597   7.0663e+06
```

```
# Calculate the predicted values
y_hat_mf <-  r$predict(test_data, out_memory())
```

[16] https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

```r
# Update the results table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Matrix Factorization",
                                 RMSE = RMSE(test_set$rating, y_hat_mf)))
```

Method	RMSE
Project Goal	0.864900
Naive RMSE	1.060054
Movie Effect	0.942961
Movie + User Effect	0.864684
Regularized Movie + User Effect	0.864136
Matrix Factorization	0.795153

Matrix factorization substantially improves the RMSE.

4.7 Final Validation

As we can see from the results table, both the regularized model and matrix factorization achieved the goal RMSE. However, matrix factorization achieved the lowest RMSE so we will test out that model only for our final validation. We will now train on the edx set and test on the validation set. We will achieve our goal if the RMSE stays below the target of 0.8649.

Matrix Factorization

When working with the initial train and test sets, matrix factorization produced the lowest RMSE. Now we need to validate if this in fact holds when training on the edx set and testing on the validation set.

```r
# Final Validation - Matrix Factorization
set.seed(1234, sample.kind = "Rounding")

# Convert 'edx' and 'validation' sets to recosystem input format
edx_rec<- with(edx, data_memory(user_index = userId,
                                item_index = movieId,
                                rating = rating))

validation_rec<-  with(validation, data_memory(user_index = userId,
                                               item_index = movieId,
                                               rating = rating))

# Create the model object
r <-  recosystem::Reco()

# Tune the parameters
opts <-  r$tune(edx_rec, opts = list(dim = c(10, 20, 30),
                                     costp_l2 = c(0.01, 0.1),
                                     costq_l2 = c(0.01, 0.1),
                                     lrate = c(0.01, 0.1),
                                     nthread = 4,
                                     niter = 10))

# Train the model
r$train(edx_rec, opts = c(opts$min, nthread = 4, niter = 10))
```

```
## iter      tr_rmse          obj
##    0       0.9726   1.2006e+07
##    1       0.8723   9.8806e+06
##    2       0.8386   9.1721e+06
##    3       0.8162   8.7476e+06
##    4       0.8006   8.4669e+06
##    5       0.7890   8.2703e+06
##    6       0.7794   8.1177e+06
##    7       0.7713   8.0000e+06
##    8       0.7643   7.9030e+06
##    9       0.7583   7.8202e+06
```

```
# Calculate the prediction
y_hat_mf_final <-  r$predict(validation_rec, out_memory())

# Update the results table
rmse_results <- bind_rows(rmse_results,
                          tibble(Method = "Final Validation - Matrix
                                 Factorization",
                                 RMSE = RMSE(validation$rating,
                                             y_hat_mf_final)))
```

Method	RMSE
Project Goal	0.864900
Naive RMSE	1.060054
Movie Effect	0.942961
Movie + User Effect	0.864684
Regularized Movie + User Effect	0.864136
Matrix Factorization	0.795153
Final Validation - Matrix Factorization	0.791754

The final validation with matrix factorization produced an RMSE of 0.7918. It performed even better using the edx and validation sets than it did on the initial train and test sets.

Let's look at the top 10 best movies and 10 worst movies predicted using matrix factorization:

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

title
Charm's Incidents (Charms Zwischenfälle) (1996)
Shawshank Redemption, The (1994)
Rhyme & Reason (1997)
Shawshank Redemption, The (1994)
Boys Life 2 (1997)
Lord of the Rings: The Return of the King, The (2003)
Shawshank Redemption, The (1994)
Usual Suspects, The (1995)
Schindler's List (1993)
Annie Hall (1977)

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

title
Time Walker (a.k.a. Being From Another Planet) (1982)
Dead Girl, The (2006)
Inland Empire (2006)
Beast of Yucca Flats, The (1961)
Bug (2007)
Free Willy 2: The Adventure Home (1995)
What Time Is It There? (Ni neibian jidian) (2001)
Daddy Day Camp (2007)
Texas Chainsaw Massacre, The (1974)
Faces of Death 6 (1996)

5. Conclusion

We started off our project by preparing our dataset and analyzing it for any insights that might be helpful in building our recommender system. From our analysis, we saw strong evidence of a movie effect (some movies were rated more often than others and some movies received higher ratings than others). We also noticed that there was variability in user behavior (user effect) since some users rated more movies than others and some users were more critical than others.

We started off by building the simplest linear model which took just the average of all the ratings. Although this was a good starting point, it produced quite a high RMSE and did not take into account movie and user effects. We then added movie and user effects to the model which helped improve the RMSE but still remained above target. We used regularization to penalize large estimates coming from small sample sizes of movies and users. This method produced an RMSE of 0.8641, successfully achieving the project target of RMSE < 0.86490.

Our final approach involved matrix factorization using the R recosystem package, which achieved an RMSE of 0.795, surpassing the project's goal.

We chose matrix factorization for final validatiion and achieved an RMSE of 0.7918.

5.1 Limitations

We were unable to use other machine learning algorithms such as linear regression, generalized linear models, random forests or k-nearest neighbors because they would be too slow when working with a dataset this large. We would also run into memory issues if we're using standard laptops to compute them. Therefore, we focused on a model-based approach and matrix factorization, which we were able to compute.

Modern recommender systems use many features to make predictions. Once again, due to computational limitations, we used only two predictors: movie and user effects. Including other predictors, such as genre, timestamp, etc. could help overcome the issue of there being no initial movie recommendations for new or inactive users (i.e. those that don't rate movies often), however they also increase the complexity of our model.

Although matrix factorization predicts with greater accuracy, it is harder to tune and does take a long training time.

5.2 Future Work

This report focused on building recommender systems using a linear model approach and matrix factorization. There are several other widely used recommender systems which weren't explored in this report, such as content-based and memory-based collaborative filtering methods, which can be implemented using the R recommenderlab package[17]. Other approaches to building recommender systems include SlopeOne[18] and SVD Approximation[19]. Future work could include using these alternative methods to make predictions and checking their performance against the linear model (with regularization) and matrix factorization.

[17] https://cran.r-project.org/web/packages/available_packages_by_name.html
[18] https://arxiv.org/abs/cs/0702144
[19] https://cran.r-project.org/web/packages/recommenderlab/readme/README.html