Recommendation System - MovieLens Project

Luiz Guilherme Gomes Fregona

June, 2021

Contents

1. Introduction

In 2009, Netflix opened a public machine learning competition for the best recommendation system to predict user ratings for movies. In that competition, Netflix provided a dataset of 100,480,507 ratings that 480,189 users gave to 17,770 movies. For this project, we will be also creating a movie recommendation system by using a small subset of the original data known as the "10M version of the MovieLens dataset". The data is open source and can be found at the following website: https://grouplens.org/datasets/movielens/10m/

1.1. Problem Statement

- Three major characteristics must be taken into account while dealing with the present problem:
  - The first one is related to the dataset's nature, each outcome has a different set of predictors. Different users rate different numbers of movies and rate different movies as well. This is not usually found in machine learning problems.
  - The MovieLens dataset also reveals a myriad of biases, which are commonly presented in a movie review. Some bias has its origin at a personal level and usually are related to the background of each person. It means that reaching perfect accuracy would be impossible. However, major patterns can be found by studying major bias, and its effect on the overall accuracy of our model.
  - And finally, due to the extensive number of rows, it is computationally expensive for most home computers to calculate complex algorithms. They will crackdown before being able to perform most machine learning algorithms.

1.2. Objetives

Considering all the issues pointed out in the previous topic, the goal of this project is to develop an algorithm that can predict ratings for movie i by user u efficiently, that takes into consideration most major bias and uses the whole data set as predictors for each rating estimate.

2. Pipeline

- The main steps in a data science project can change substantially between projects, and it is always wise to define our goals and sources of information before setting it in stone. The pipeline decided for this project includes the following 6 steps:
  - 1º Data preparation I: download, wrangle and store the data set(s) as a .rda file to be processed and analyzed.
  - 2º Data analysis: explore the data set to find any structure or meaningful relationship between features.
  - 3º Data preparation II: prepare the final data set for modeling. In this step it is included:
    - data cleaning: get rid of unnecessary features for modeling.
    - data wrangling: format and create new data sets for training and testing our models as well as for tuning parameters.
  - 4º Modeling: create the model, test, and validate it.
  - 5º Data analysis II: explore the model residuals to understand if further analysis is necessary.
  - 6º Modeling II: create new models based on information obtained in the previous step.
  - 7º Reporting: organize the finding in a clear .pdf and .html files.

3. Data Preparation I

In this section, we download and prepare the dataset for use in the data analysis step. The raw data were download from http://files.grouplens.org/datasets/movielens/ml-10m.zip, then separated into two objects: "ratings" and "movies". Afterward, they were put together again using the "movieId" column from rating as reference. The new object "movielens" was then split into two parts, the training set called *edx* and the final evaluation set called *validation*.

3.1 Libraries

The following libraries were used in this project:

```
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(stringr)
library(ggrepel)
library(kableExtra)
library(recosystem)
```

3.2 Raw Data Loading

```
# MovieLens 10M dataset:
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

```
#Rating dataset
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
```

```
#Movies dataset
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 dataset - only movieIds from "ratings" were considered
movielens <- left_join(ratings, movies, by = "movieId")
```

3.3 Data Wrangling

The following code generates the train and final hold-out test set. To do so, we randomly split the movielens set into two sets where 90% of it went to the training set (*edx*) and 10% to the evaluation set (*validation*). The validation set was used with only one purpose, to evaluate our final recommendation system.

```
# 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)
```

```
#Segregate movielens dataset into edx and temp datasets
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

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

```
# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

```
#Remove useless datasets and variables previously created
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

All code up to this point was provided by "HarvardX - Data Science: Capstone Project". It was not made by the author of this project.

3.4. Intermediate Dataset

In order to make the further analysis fast between days of work, the *edx* and *validation* set were saved as .rda files.

```
#Saving edx and validation set as .rda files
save(edx, file = "rda/edx.rda")
save(validation, file = "rda/validation.rda")
```

For data analysis, each time a day of work started, the *edx* dataset was loaded back to the system.

```
load("rda/edx.rda")
```

4. Data Analysis

Before doing any advanced analysis, we need to understand how the data set is structured, what are predictors we are going to use, and the relationship between them. This information will help us build a better model.

4.1. General properties

In this section, we got a grasp on the general characteristics of our data set, such as the size, classes of predictors, data set format, number of unique values per predictors, etc.

```
#Data set Dimensions
dim(edx)
```

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

The edx data set is composed by 6 columns and 9000055 observations.

```
#Data set Structure
str(edx, vec.len = 2)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 ...
##  $ rating   : num  5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

- The six predictors are, respectively:
  - userId:
    - class: integer
    - definition: user identification
  - movieId:
    - class: numeric
    - definition: movie identification
  - rating:
    - class: numeric
    - definition: rating scores
  - timestamp:
    - class: integer
    - definition: seconds
  - title:
    - class: character
    - definition: title of movies followed by its release date
  - genres:
    - class: character
    - definition: combination of multiple genres related to the movie

```r
#Small sample of observations
head(edx) %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T)
```

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

The data set is organized in a tidy format, which means that each observation represents one rating of one movie by one user. Also, the predictor "timestamp" represents seconds since midnight UTC of January 1, 1970.

```r
edx %>%
  summarise(users = n_distinct(userId),
            movies = n_distinct(movieId),
            rating = n_distinct(rating)) %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position"), font_size = 12) %>%
  row_spec(row = 0, bold = T)
```

users	movies	rating
69878	10677	10

There are 69878 unique values of userIds, 10 different rating scores, and 10677 distinct movieIds. By doing a simple multiplication between the number of unique users and movies, we can see that not all users rated all movies.

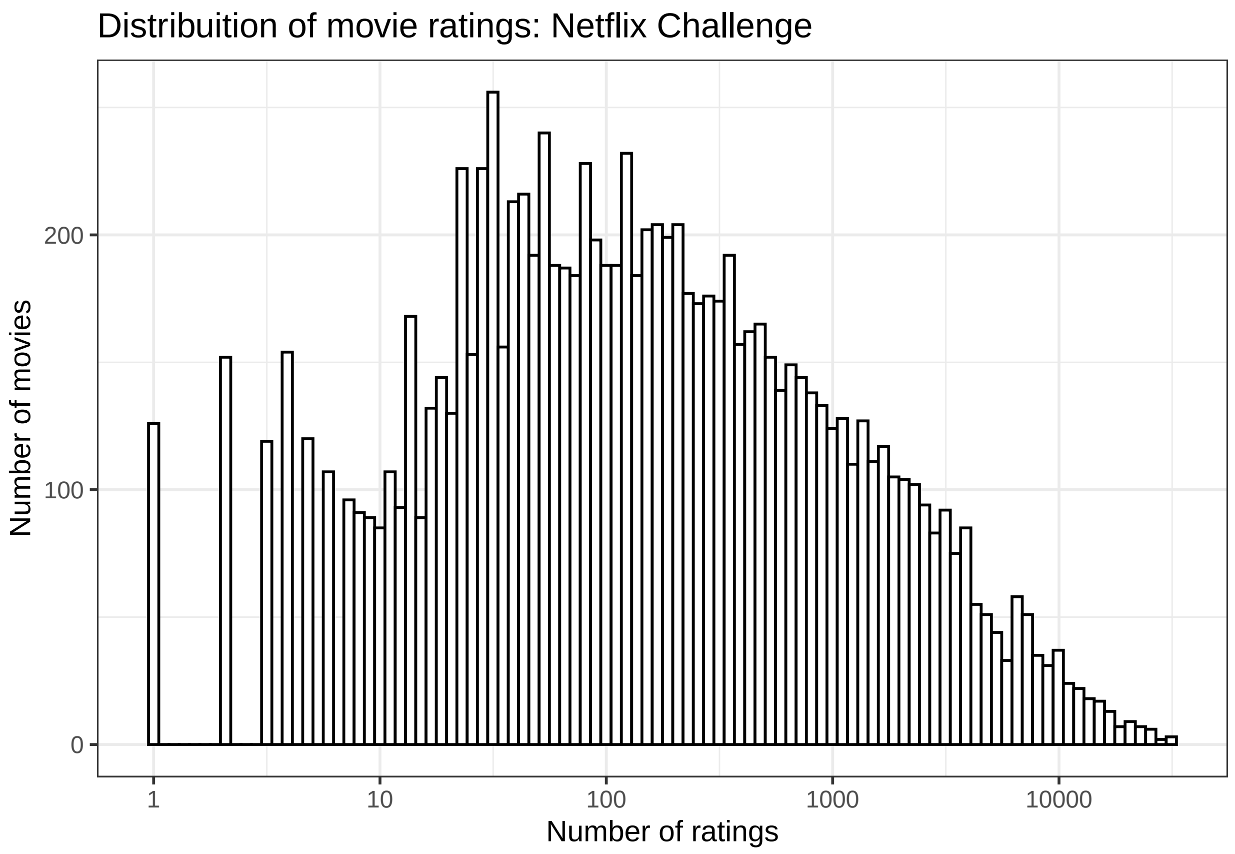
4.2. Data Exploration

The data exploration step had as its main objective to understand the variability of each predictor. With this in mind, we focused on building histograms as our first step and then moved to advanced concepts, such as confidence intervals of averages, when looking for strong evidence.

4.2.1. Movies

The movie predictor distribution is displayed right below:

```r
#Number of times distinct movies were rated by users
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 100, color = "black", fill = "white") +
  scale_x_log10() +
  ylab("Number of movies") +
  xlab("Number of ratings") +
  ggtitle("Distribuition of movie ratings: Netflix Challenge") +
  theme_bw()
```



It is clear by the figure above that some movies are more rated than others. It is obvious since there are blockbusters movies, which are watched by millions, and unsuccessful movies, not seen by a few.

```r
#Top 5 most rated movies
edx %>%
  group_by(title) %>%
  summarise(Titles = unique(title),
            Number_of_ratings = n()) %>%
  select(Titles, Number_of_ratings) %>%
  arrange(desc(Number_of_ratings)) %>%
  top_n(5) %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = T) %>%
  column_spec(1, width = "8cm") %>%
  row_spec(row = 0, bold = T)
```

Titles	**Number_of_ratings**
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015

```r
#Top 5 least rated movies
edx %>%
  group_by(title) %>%
  summarise(Titles = unique(title),
            Number_of_ratings = n()) %>%
  select(Titles, Number_of_ratings) %>%
  arrange(desc(Number_of_ratings)) %>%
  slice_tail(n=5) %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = T) %>%
  column_spec(1, width = "12cm") %>%
  row_spec(row = 0, bold = T)
```

Titles	**Number_of_ratings**
When Time Ran Out... (a.k.a. The Day the World Ended) (1980)	1
Where A Good Man Goes (Joi gin a long) (1999)	1
Won’t Anybody Listen? (2000)	1
Young Unknowns, The (2000)	1
Zona Zamfirova (2002)	1

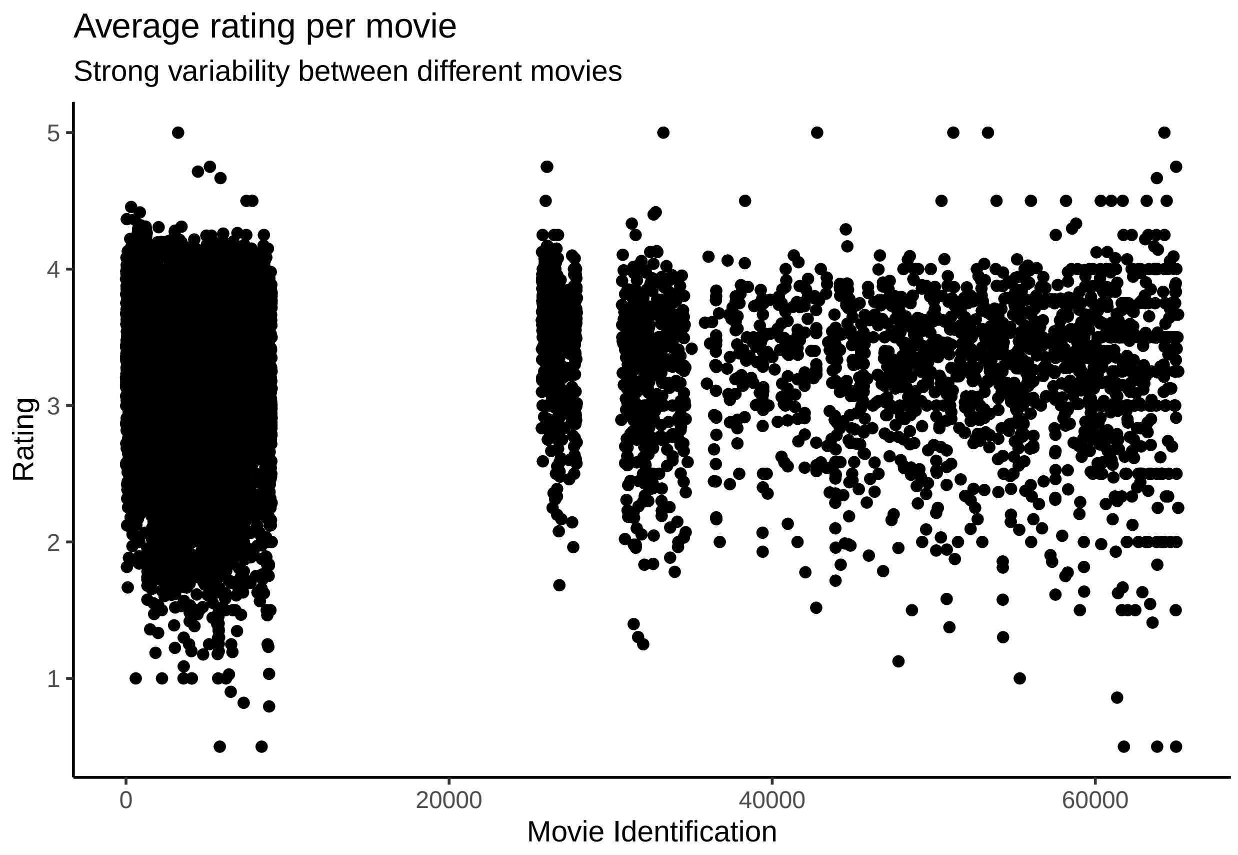
The top 5 movies were rated more than 28.000 times, while the top least rated movies just once.

```r
#Proportion of users who rated a specific movie
edx %>%
  count(movieId) %>%
  mutate(Proportion_of_users = case_when(
    n <= 700 ~ "less than 1%",
    n >= 5000 ~ "more than 10%",
    TRUE ~ "between 1% and 10%"
  )) %>%
  group_by(Proportion_of_users) %>%
  summarise(Number_of_ratings = n()) %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position"), full_width = T) %>%
  column_spec(1, width = "12cm") %>%
  row_spec(row = 0, bold = T)
```

Proportion_of_users	**Number_of_ratings**
between 1% and 10%	1955
less than 1%	8303
more than 10%	419

Most of the films are not rated by more than 1% of all users, and just a few get to be rated by more than 10% of all users.

```r
edx %>%
  group_by(movieId) %>%
  summarise(avg = mean(rating)) %>%
  ggplot(aes(movieId, avg)) +
  geom_point() +
  theme_classic() +
  ylab("Rating") +
  xlab("Movie Identification") +
  ggtitle("Average rating per movie",
          subtitle = "Strong variability between different movies")
```

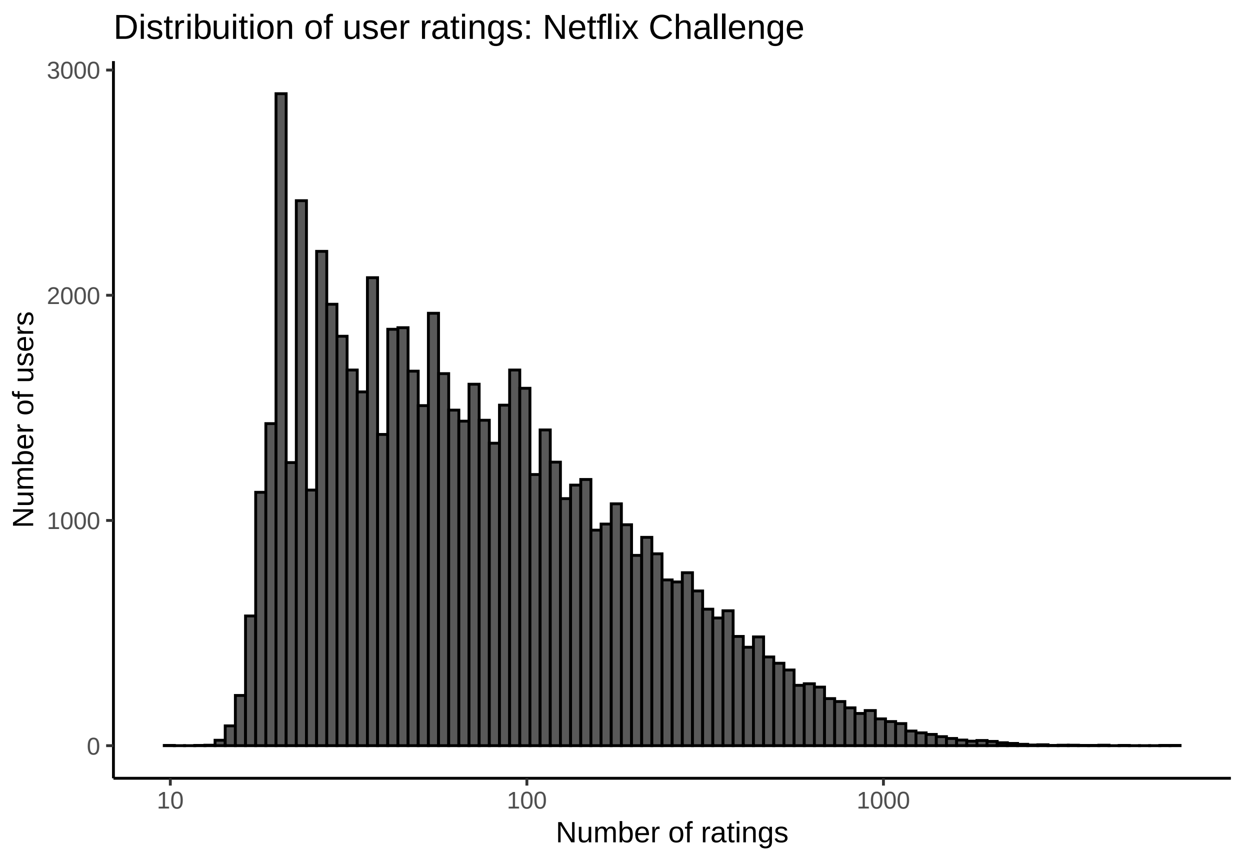


We can also see that the average rating of each movie varies substantially and that ratings between 2 and 4 are prevalent. That's an important characteristic in our data set and needs to be addressed to make better rating predictions.

4.2.2. Users

Up to 70.000 users were accounted for in this data set. The distribution of user ratings shows us that some users are more active than others. We can also see that the majority of users have rated less than 100 times. The top 5 users had rated more than 4.000 times, while the top least active users less than 15 times.

```r
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 100, color = "black") +
  scale_x_log10() +
  ylab("Number of users") +
  xlab("Number of ratings") +
  ggtitle("Distribuition of user ratings: Netflix Challenge") +
  theme_classic()
```



```
#Top 5 most active users
edx %>%
  group_by(userId) %>%
  summarise(User_Identification = unique(userId),
            Number_of_ratings = n()) %>%
  select(User_Identification, Number_of_ratings) %>%
  arrange(desc(Number_of_ratings)) %>%
  top_n(5) %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T)
```

User_Identification	**Number_of_ratings**
59269	6616
67385	6360
14463	4648
68259	4036
27468	4023

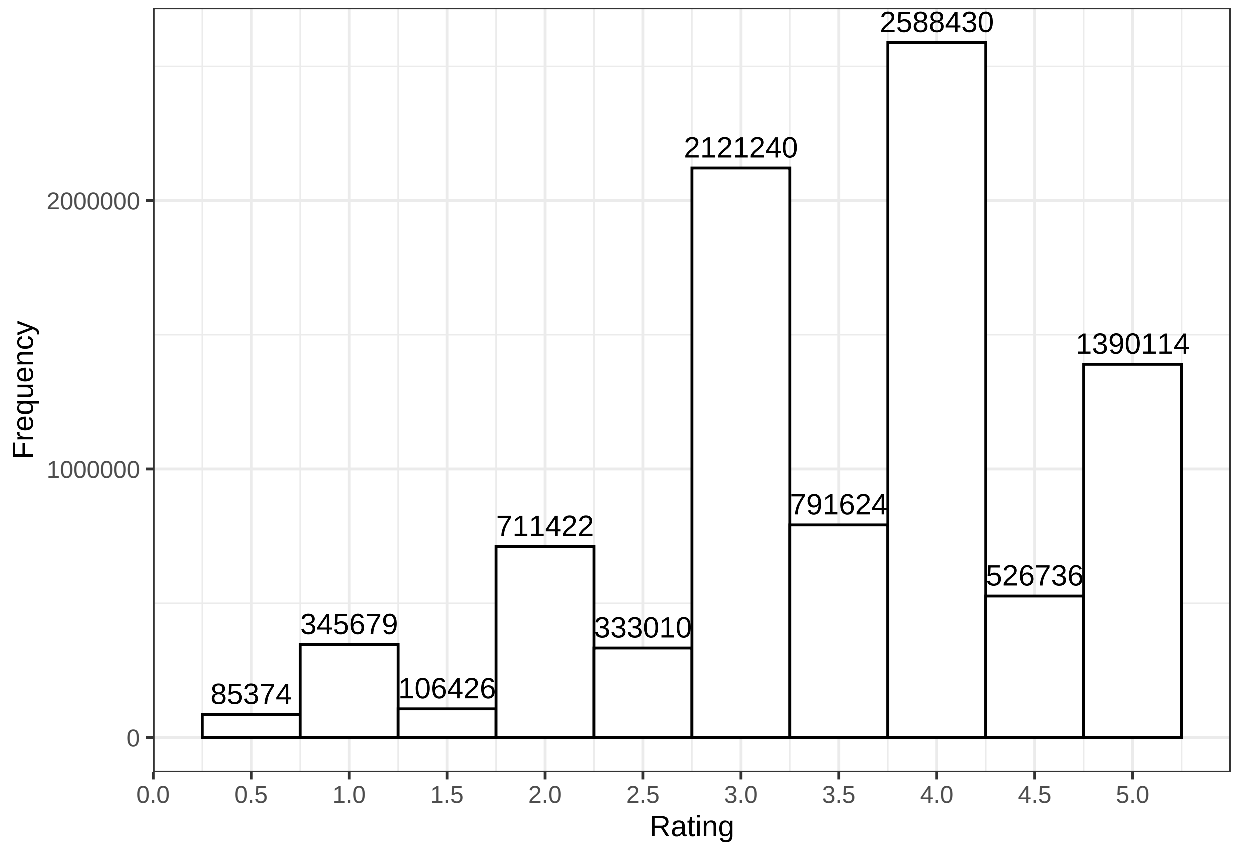
```
#Top 5 least active users
edx %>%
  group_by(userId) %>%
  summarise(User_Identification = unique(userId),
            Number_of_ratings = n()) %>%
  arrange(desc(Number_of_ratings)) %>%
  select(User_Identification, Number_of_ratings) %>%
  arrange(Number_of_ratings) %>%
  slice(1:5) %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T)
```

User_Identification	**Number_of_ratings**
62516	10
22170	12
15719	13
50608	13
901	14

4.2.3. Rating

Netflix provided 10 distinct values for their users to rate movies, varying from 0.5 to 5.0. The distribution of rating values showed 2 interesting insights: First, most movies got rounded value ratings, and on average Netflix, users were not demanding regarding the quality of movies, ranking movies usually above 3.0.

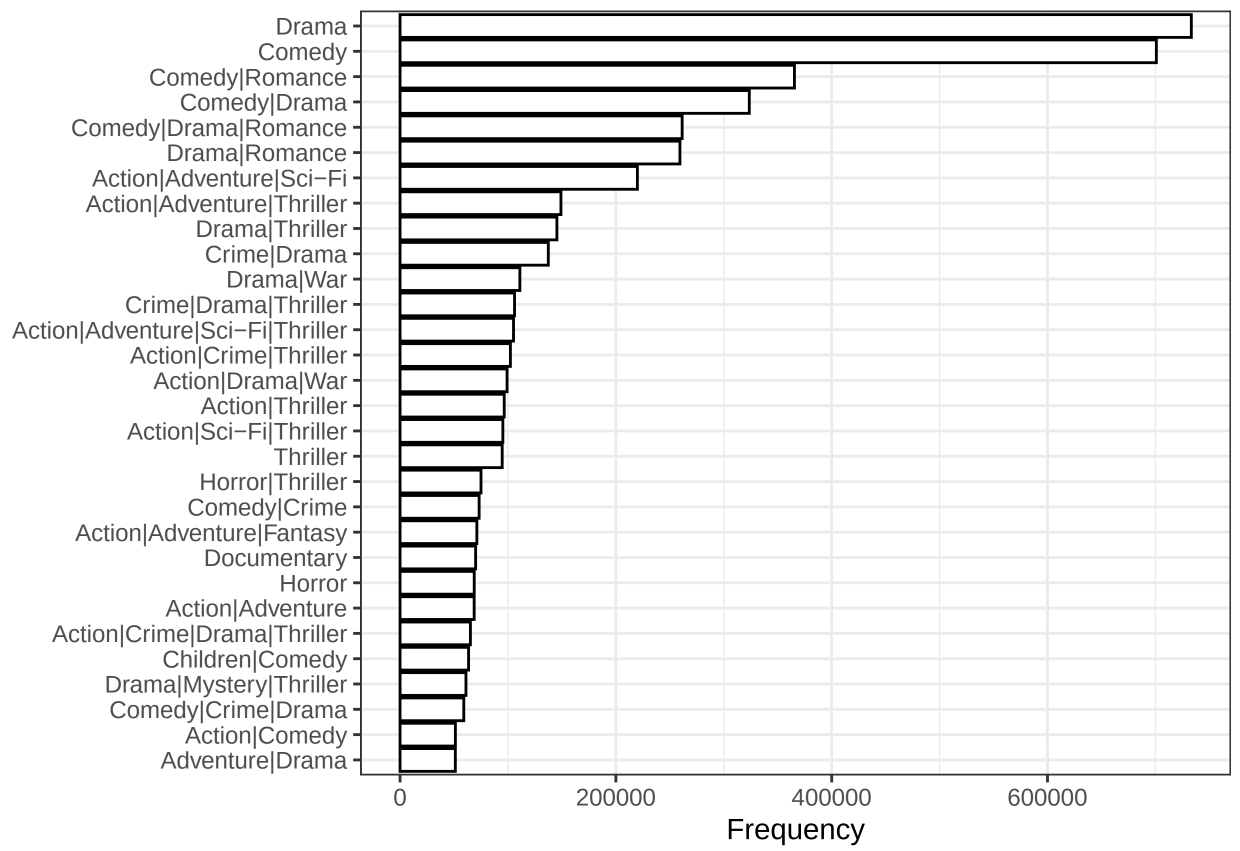
```
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, colour = "black", fill = "white") +
  scale_x_continuous(breaks = seq(0, 5, by=0.5)) +
  stat_bin(binwidth = 0.5, aes(label = ..count..), vjust = -0.5, geom = "text") +
  ylab("Frequency") +
  xlab("Rating") +
  theme_bw()
```



4.2.4 Genres

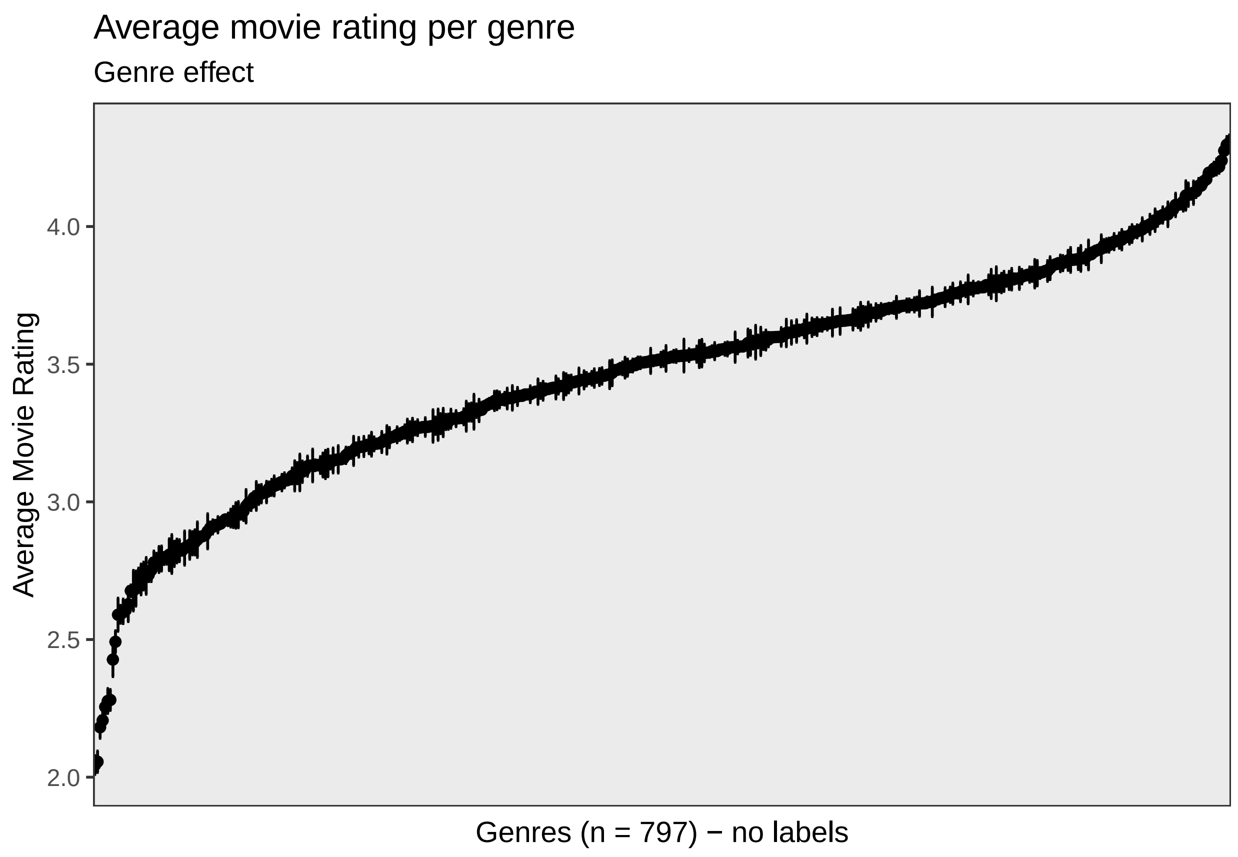
The “genres” predictor shows a strong variability in the number of movie ratings, and rating values as well. The plot below of the top 30 most rated genres indicates such numerical variability. Here, drama and comedy were the most rated genre with over 700.000 rates. The least rated genre is not presented in the following plot, but it was rated only twice.

```r
#Distribution of movie ratings per genre - top 30 most rated genres
edx %>%
  count(genres) %>%
  arrange(desc(n)) %>%
  slice_head(n=30) %>%
  ggplot(aes(reorder(genres, n, sum), n)) +
  geom_bar(stat = "identity", colour = "black", fill = "white") +
  coord_flip() +
  theme_bw() +
  ylab("Frequency") +
  theme(axis.title.y = element_blank())
```



The average values of movie ratings per genre were calculated, and it can be seen in the following plot. It was considered only genres that have been rated more than 1000 times. The scatter plot undoubtedly shows variability in rating values between genres.

```r
#Average movie ratings per genre
edx %>%
  group_by(genres) %>%
  summarise(n = n(),
            avg = mean(rating),
            se = sd(rating)/sqrt(n)) %>%
  filter(n >= 1000) %>%
  ggplot(aes(x = reorder(genres, avg), y = avg,
             ymin = avg - qnorm(0.975)*se, ymax = avg + qnorm(0.975)*se)) +
  geom_point() +
  geom_errorbar() +
  theme_bw() +
  theme(axis.ticks.x = element_blank()) +
  theme(axis.text.x = element_blank()) +
  ggtitle("Average movie rating per genre", subtitle = "Genre effect") +
  ylab("Average Movie Rating") +
  xlab("Genres (n = 797) - no labels")
```



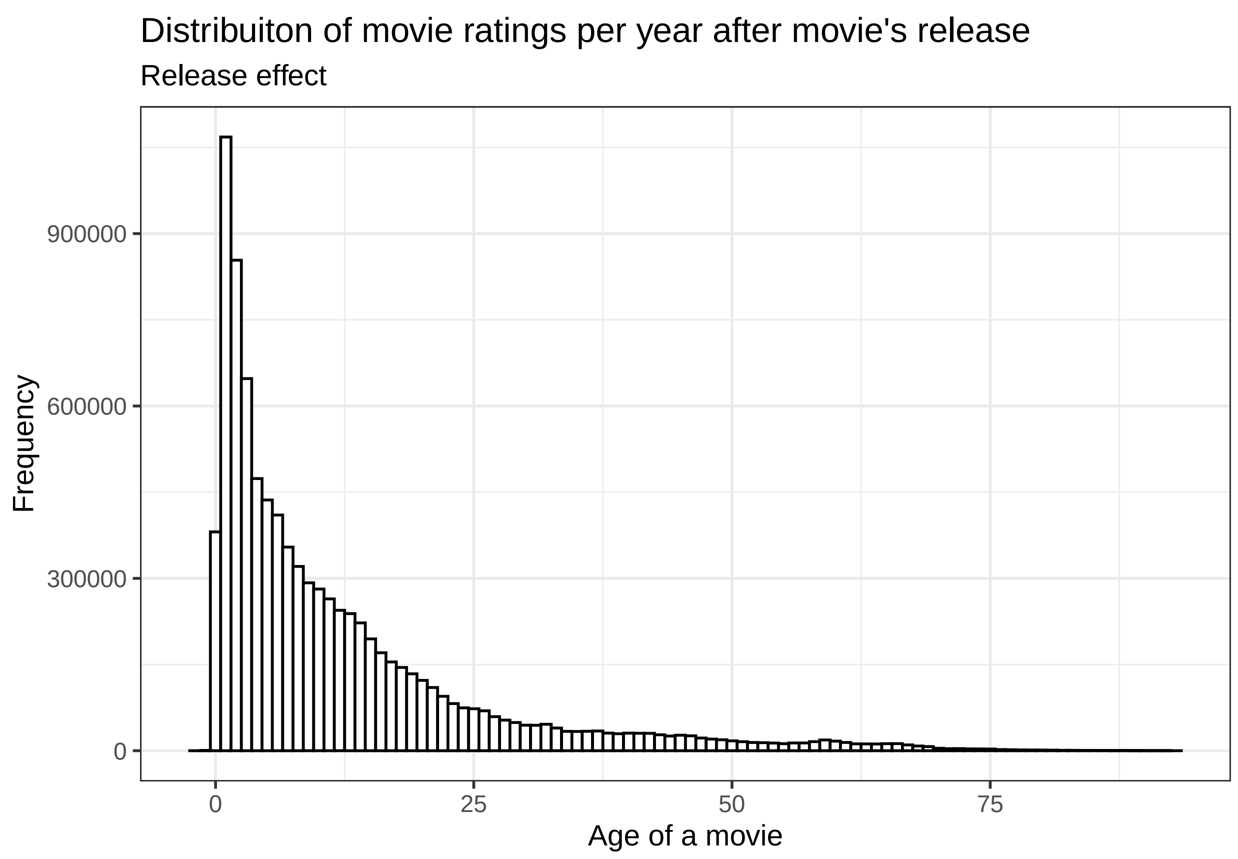
4.2.5 Timestamp

Before any analysis, the "timestamp" feature needs to be converted from seconds to date in order to facilitate visualization and comprehension of our data. We also found interesting to explore 2 bias commonly seen in movie reviews, the "Nostalgia bias", a psychological feature that human beings tend to give higher ratings to old movies, on average, compared to new ones, and the "Release effect". To do that, we subtract the release year of a movie by the time it was rated by a user.

```
##Calculating the difference between the released year of a movie,
#and the time it was rated by an user.
edx_t <- edx %>%
  mutate(rating_year = year(as_datetime(timestamp)),
         release_year = as.numeric(str_sub(title, -5, -2))) %>%
  mutate(timediff = as.numeric(rating_year) - release_year) %>%
  select(userId, movieId, rating, timediff, title, genres)
```

The distribution shows clearly the "Release effect", where right after a movie release there is a huge number of users rating movies.

```
#Distribution of movie ratings per year after movie's release
edx_t %>%
  ggplot(aes(timediff)) +
  geom_histogram(binwidth = 1, colour = "black", fill = "white") +
  theme_bw() +
  ylab("Frequency") +
  xlab("Age of a movie") +
  ggtitle("Distribuiton of movie ratings per year after movie's release",
          subtitle = "Release effect")
```



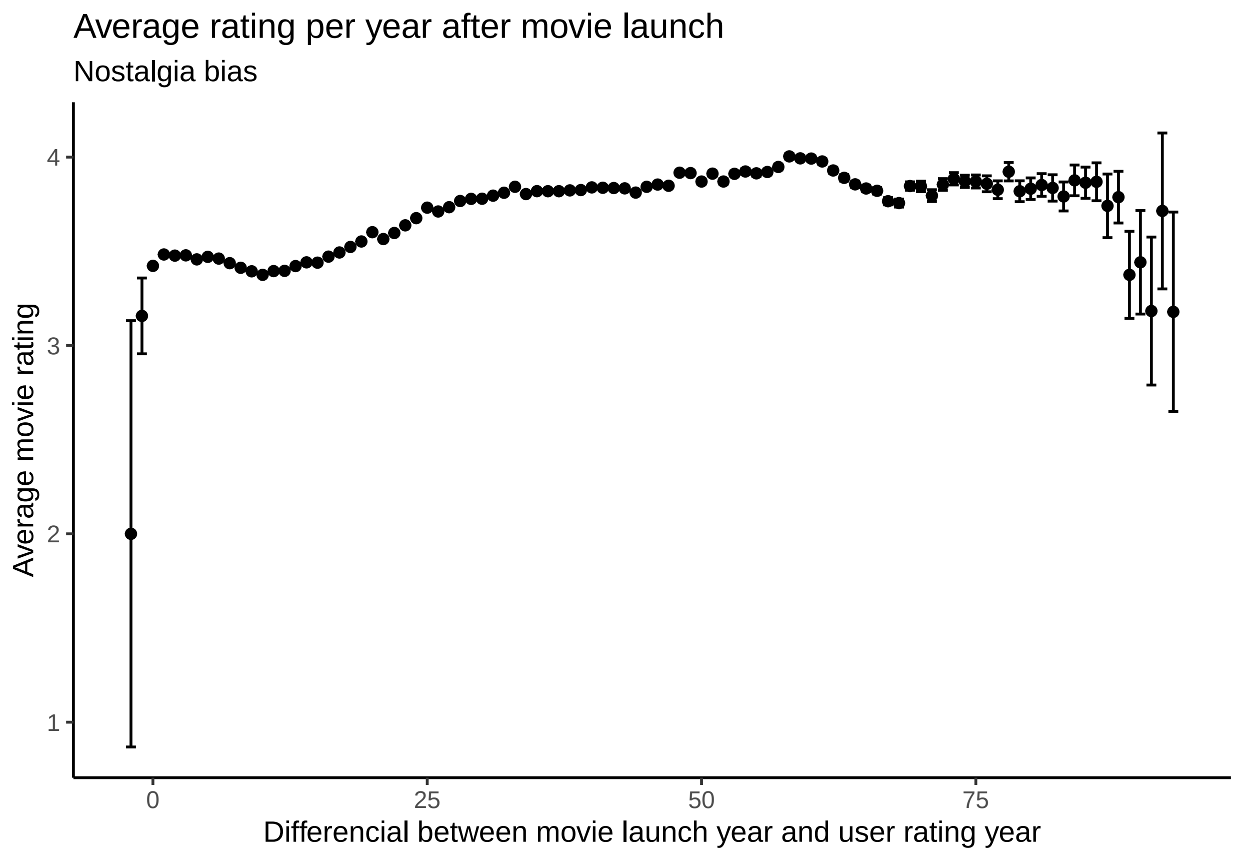
The table below indicates once again this pattern, where more than 12% of all ratings among 93 years were given before the end of the movies' first year on the screen.

```
#Ratio of ratings throughout years after movie release
edx_t %>%
  count(timediff) %>%
  mutate(rating_year = case_when(
    timediff == 1 ~ "release year",
    timediff <= 5 ~ "less than 5 years",
    TRUE ~ "more than 5 years"
    ), total = sum(n)) %>%
  group_by(rating_year) %>%
  summarise(ratio = unique(sum(n)/total)*100) %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T)
```

rating_year	**ratio**
less than 5 years	31.02712
more than 5 years	57.10551
release year	11.86737

The chart below shows the "Nostalgia bias". As we can see, movies with more than 25 years receive better ratings.

```
#Nostalgia effect
edx_t %>%
  group_by(timediff) %>%
  summarise(n = n(),
            avg = mean(rating),
            se = sd(rating)/sqrt(n)) %>%
  ggplot(aes(timediff, y = avg, ymin = avg - qnorm(0.975)*se,
             ymax = avg + qnorm(0.975)*se)) +
  geom_point() +
  geom_errorbar() +
  theme_classic() +
  ylab("Average movie rating") +
  xlab("Differencial between movie launch year and user rating year") +
  ggtitle("Average rating per year after movie launch", subtitle = "Nostalgia bias")
```



While modelling we put together both effects and considered as one, we called it the “Time effect”.

5. Data Preparation II

The *edx* set was again randomly divided into train and test sets, to train and compare models. The training set has got 90% and the test set 10% of all observations. Also, we created a new variable called *timediff*, which is the difference in time between the movie release year and the time it was rated by a user. The new data sets were then saved as .rda files for easy access to the data between days of work.

```r
#Turn the "timestamp" feature into "diff_years" predictor by subtracting the release
#year of a movie by the user's rating year.
edx_modelling <- edx %>%
  mutate(rating_year = year(as_datetime(timestamp)),
         release_year = as.numeric(str_sub(title, -5, -2))) %>%
  mutate(timediff = as.numeric(rating_year) - release_year) %>%
  select(userId, movieId, rating, timediff, title, genres)
```

```r
#Separate the edx_modelling set into training and cross validation set
set.seed(2, sample.kind="Rounding")
test_index <- createDataPartition(edx_modelling$rating, times = 1, p = 0.1,
                                  list = FALSE)
train_temp <- edx_modelling[-test_index, ]
temp <- edx_modelling[test_index, ]
```

```r
#To make sure we don't include users and movies in the cv set that do not appear
#in the training set.
test_set <- temp %>%
  semi_join(train_temp, by = "userId") %>%
  semi_join(train_temp, by = "movieId")
```

```r
#Add removed rows from the test_set into the train_set
removed <- anti_join(temp, test_set)
train_temp <- rbind(train_temp, removed)
```

For the purpose of tuning lambda before regularizing our model, we split our *train_set* one more time. The split followed the same orientations explained previously.

```r
set.seed(10, sample.kind="Rounding")

#Split train_set into training_set and cv_set
test_index <- createDataPartition(train_temp$rating, times = 1, p = 0.1,
                                  list = FALSE)
train_set <- train_temp[-test_index, ]
temp <- train_temp[test_index, ]
```

```r
#To make sure we don't include users and movies in the cv_set_tune set
#that do not appear in the training set.
cv_set <- temp %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by = "movieId")
```

```r
#Add removed rows from the cv_set_tune into the train_set
removed <- anti_join(temp, cv_set)
train_set <- rbind(train_set, removed)
```

```r
#Save files as new .rda files for modelling
save(train_set, file = "rda/train_set.rda")
save(test_set, file = "rda/test_set.rda")
save(cv_set, file = "rda/cv_set.rda")
```

6. Modeling I

The methodology used for modeling was a model-based approach. The baseline model assumed that all variation in the data is due to random variation + independent sampling error. The value that minimizes the RMSE based on this model is the average.

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

- Where:
  - $Y_{u,i}$ is the rating for movie i by user u
  - u is the "true" rating for all movie - the overall rating average
  - ϵ independent error sampled from the same distribution centered at 0

According to our data exploration results, the data sets' variability is not only because of random variation. As we saw previously, different movies are rated differently, some users are more evaluative than others, movies of distinct genres do not receive the same rating, and old movies are better rated than new ones. Those effects were confirmed by data and added to our model.

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

- Where:
  - b_i is the average rating for movie i
  - b_u is the average rating for user u
  - b_g is the average rating for movie g
  - b_t is the average rating for movie t

6.1. Evaluation Metric

The quality of the model will be measured by the root mean squared error (RMSE). The RMSE compares the predicted value with the actual outcome and measures the difference between them. As a rule of thumb, the lower the RMSE number is the better. The root mean squared error is defined as follows:

$$RMSE = \sqrt{\frac{\sum_{u,i}(y_{u,i} - yhat_{u,i})^2}{N}}$$

- Where:
  - y_i is the rating by movie i by user u
  - $yhat_i$ is the estimated rating by movie i by user u
  - N is the total number of ratings

The code below does exactly what the formula above has shown us. A function as created in order to use it multiple times throughout this work.

```
#Loss Function - RSME
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings-predicted_ratings)^2))
}
```

There are other ways to evaluate machine learning algorithms, but that go beyond the scope of this project.

6.2. Baseline Model - Random Variation

The baseline model is just the overall average of all ratings in the training set. The result from this model is far from ideal but gives us a starting point from which we can improve on.

```
#Simplest Model -> Total Variability due to random variation only
mu_hat <- mean(train_set$rating)

#Model Accuracy
naive_rmse <- RMSE(test_set$rating, mu_hat)

#Keep track of RMSE results from different models
rmse_results <- tibble(Method = "Just the average", RMSE = naive_rmse)

#Models RMSE results
rmse_results %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "14cm")
```

Method	**RMSE**
Just the average	1.06031

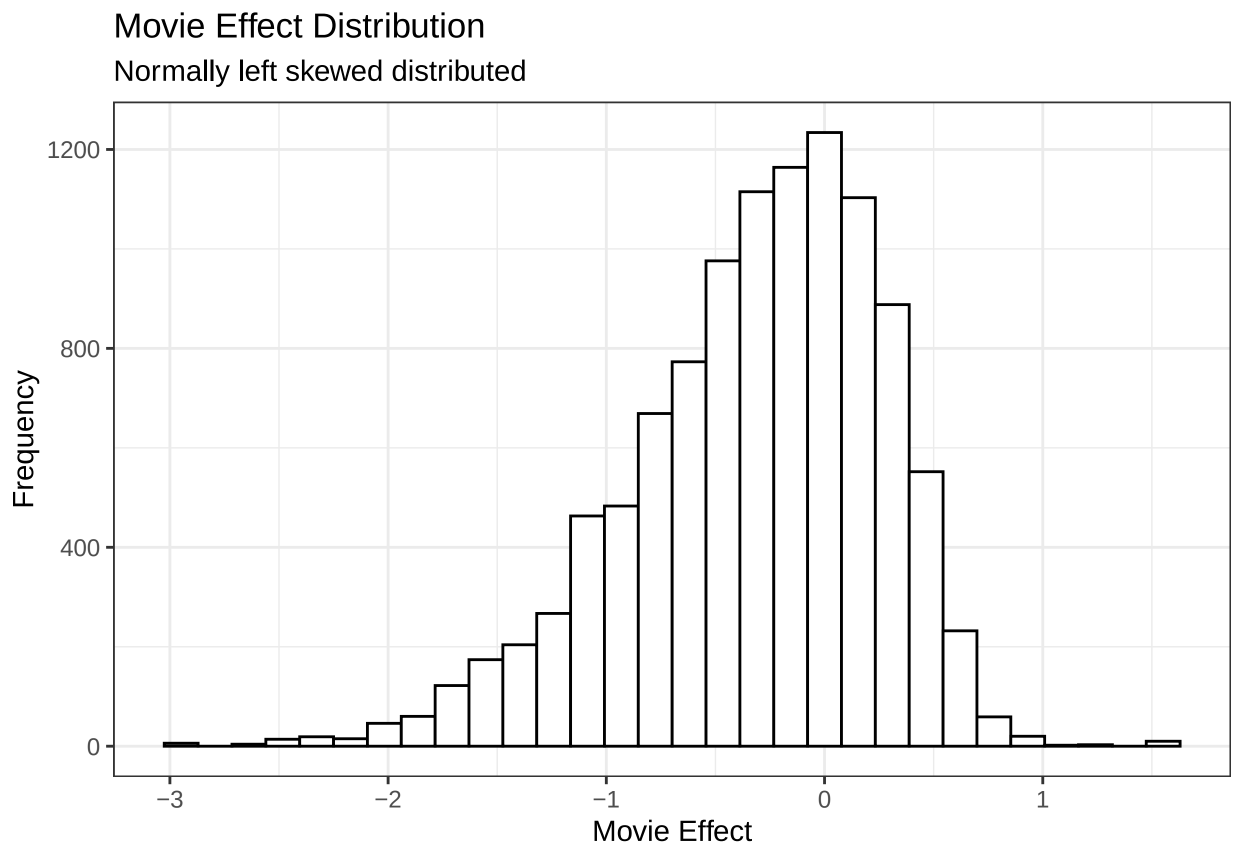
Model 2 - Random variation + Movie Effect

Here, we included the movie effect into our model. The final model is defined as following:

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

From its distribution, we can see that the movie effect is almost perfectly normal distributed.

```r
#Calculate the average
mu <- mean(train_set$rating)

#Movie Bias
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))

#Distribution of b_i
movie_avgs %>%
  ggplot(aes(b_i)) +
  geom_histogram(color = "black", fill = "white") +
  theme_bw() +
  xlab("Movie Effect") +
  ylab("Frequency") +
  ggtitle("Movie Effect Distribution", subtitle = "Normally left skewed distributed")
```



```r
#Predicted rating - b_i + mu
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  summarise(pred = mu + b_i)
```

The RMSE improved considerably after including it into our model.

```r
#Model accuracy
movie_rmse <- RMSE(test_set$rating, predicted_ratings$pred)

#Add model 2 results to tracking table
rmse_results <- rbind(rmse_results,
                      tibble(Method = "Movie Effect",
                             RMSE = movie_rmse))
#Models RMSE results
rmse_results %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "14cm")
```

Method	**RMSE**
Just the average	1.060310
Movie Effect	0.943726

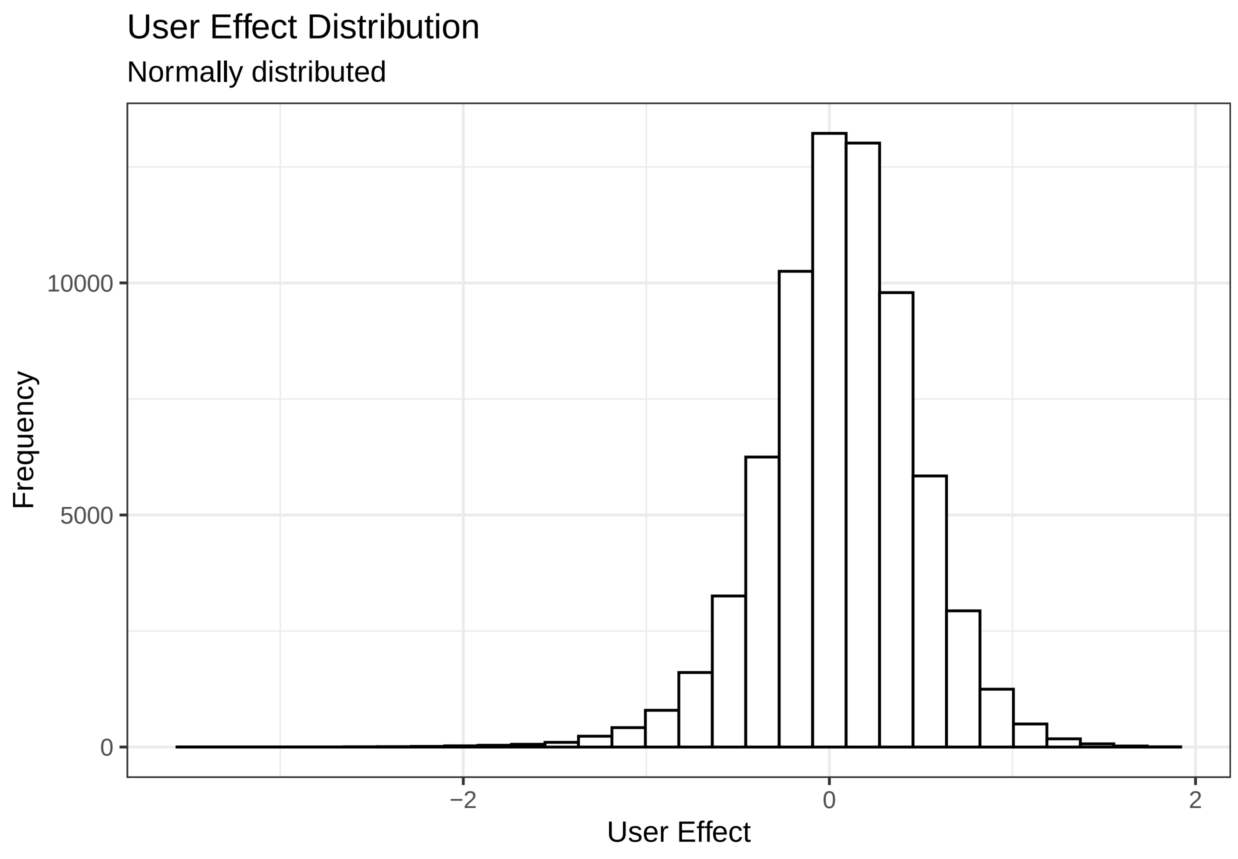
Model 3 - Random variation + Movie Effect + User Effect

Here, we included the user effect into our previous model. The final model is defined as following:

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

From its distribution, we can see that the user effect is normally distributed.

```r
#User bias
user_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))

#Distribution of b_u
user_avgs %>%
  ggplot(aes(b_u)) +
  geom_histogram(color = "black", fill = "white") +
  theme_bw() +
  xlab("User Effect") +
  ylab("Frequency") +
  ggtitle("User Effect Distribution", subtitle = "Normally distributed")
```



```r
#Predicted rating - mu + b_i + b_u
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  summarise(pred = mu + b_i + b_u)
```

The RMSE also improved considerably after including the user effect into our model. Both effects have shown a strong predictive power.

```r
#Model accuracy
user_rmse <- RMSE(test_set$rating, predicted_ratings$pred)

#Add model 3 results to tracking table
rmse_results <- rbind(rmse_results,
                      tibble(Method = "Movie + User Effects",
                             RMSE = user_rmse))

#Models RMSE results
rmse_results %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "14cm")
```

Method	**RMSE**
Just the average	1.0603099
Movie Effect	0.9437260
Movie + User Effects	0.8669568

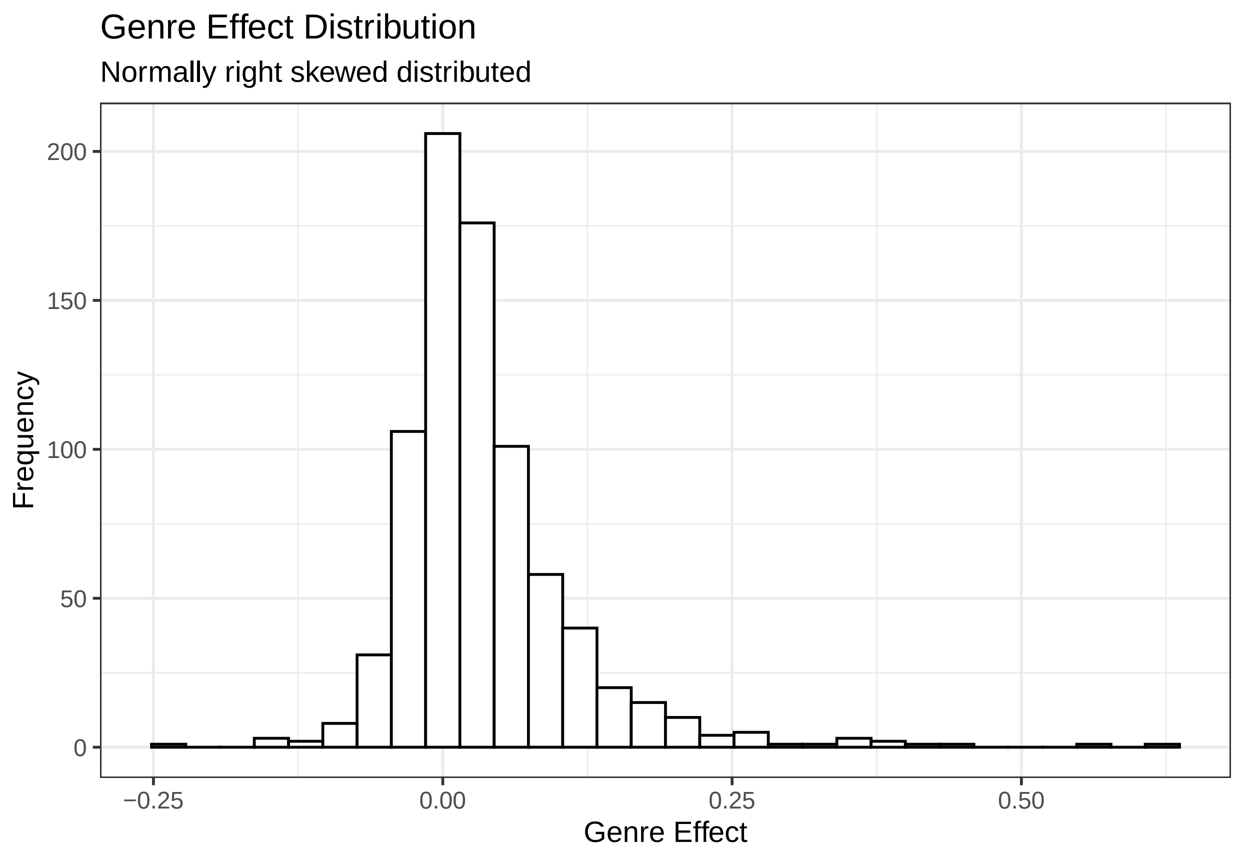
Model 4 - Random variation + Movie Effect + User Effect + Genre Effect

Here, we included the genre effect into our model 3. The current model is defined as following:

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

From its distribution, we can see that the genre effect is normally right skewed distributed

```r
#Genre Bias
genre_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarise(b_g = mean(rating - mu - b_i - b_u))

#Distribution of b_g
genre_avgs %>%
  ggplot(aes(b_g)) +
  geom_histogram(color = "black", fill = "white") +
  theme_bw() +
  xlab("Genre Effect") +
  ylab("Frequency") +
  ggtitle("Genre Effect Distribution", subtitle = "Normally right skewed distributed")
```



```r
#Predicted rating -  mu + b_i + b_u + b_g
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  summarise(pred = mu + b_i + b_u + b_g)
```

Even though we found a fair amount of variability while exploring the genre predictor, the RMSE did not improve considerably after including the genre effect into our model. Further analysis will be done, and a new model with the genre predictor sub-divided into individual genres may be necessary.

```r
#Model accuracy
genre_rmse <- RMSE(test_set$rating, predicted_ratings$pred)

#Add model 4 results to tracking table
rmse_results <- rbind(rmse_results,
                      tibble(Method = "Movie + User + Genre Effects",
                             RMSE = genre_rmse))
#Models RMSE results
rmse_results %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "15cm")
```

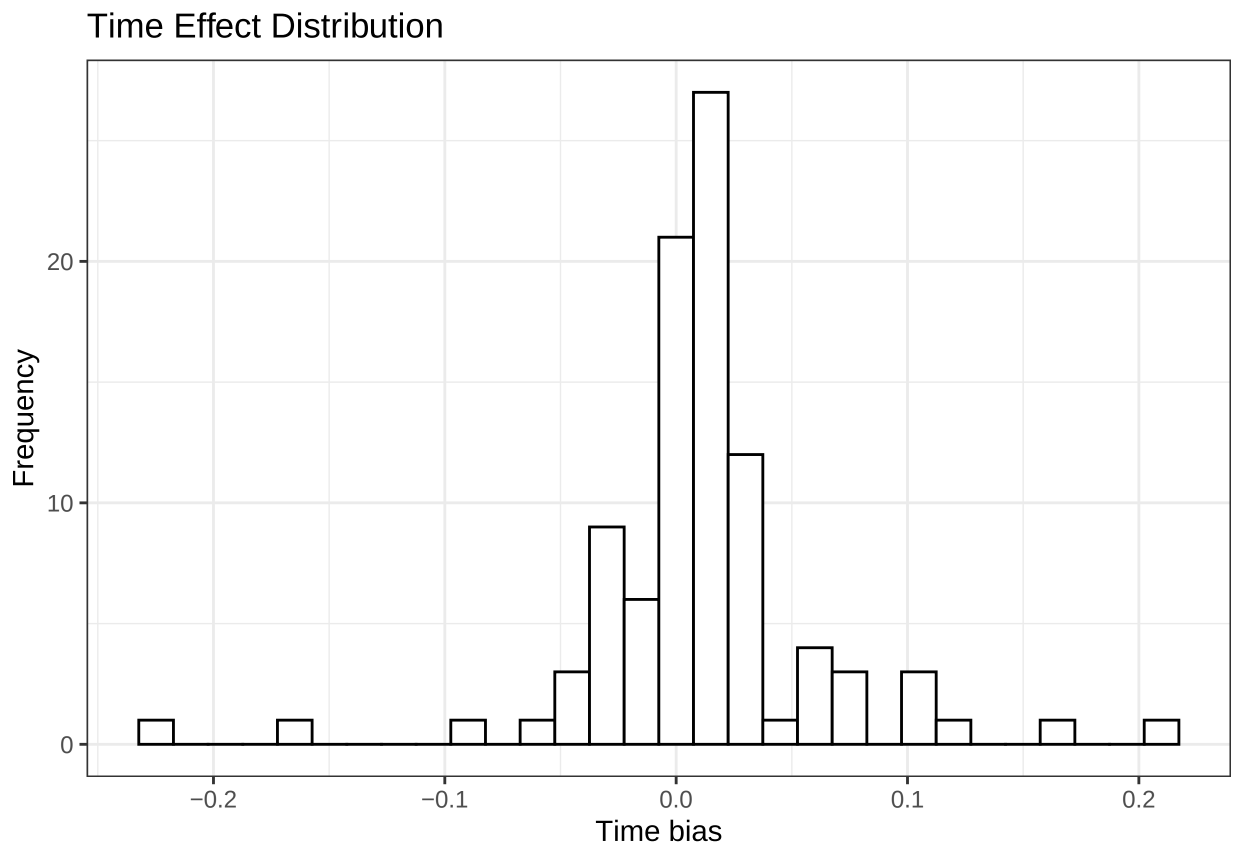
Method	**RMSE**
Just the average	1.0603099
Movie Effect	0.9437260
Movie + User Effects	0.8669568
Movie + User + Genre Effects	0.8666389

Model 5 - Random variation + Movie Effect + User Effect + Genre Effect + Time Effect

Here, we included the time ("nostalgia") effect into our model 4. The current model is defined as following:

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

```r
#Time bias
time_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  group_by(timediff) %>%
  summarise(b_t = mean(rating - mu - b_i - b_u - b_g))

#Distribution of b_t
time_avgs %>%
  ggplot(aes(b_t)) +
  geom_histogram(color = "black", fill = "white") +
  theme_bw() +
  xlab("Time bias") +
  ylab("Frequency") +
  ggtitle("Time Effect Distribution")
```



```
#Predicted rating -  mu + b_i + b_u + b_g + b_t
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(time_avgs, by = "timediff") %>%
  summarise(pred = mu + b_i + b_u + b_g + b_t)
```

Although we found a fair amount of variability while exploring the time predictor, the RMSE did not improve considerably after including the time effect into our model as well. Further analysis will be done, and a new model with both time predictor, timestamp, and release date separately, may be necessary.

```
#Model accuracy
time_rmse <- RMSE(test_set$rating, predicted_ratings$pred)

#Add model 5 results to tracking table
rmse_results <- rbind(rmse_results,
                      tibble(Method = "User + Movie + Genre + Time Effects",
                             RMSE = time_rmse))

#Models RMSE results
rmse_results %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "14cm")
```

Method	**RMSE**
Just the average	1.0603099
Movie Effect	0.9437260
Movie + User Effects	0.8669568
Movie + User + Genre Effects	0.8666389
User + Movie + Genre + Time Effects	0.8662351

Model 6 - Random variation + Movie Effect + User Effect + Genre Effect + Time Effect + Regularization

The model-based approach provided a good estimation for the ratings so far. However, there is the main issue that has not been addressed. As we can see, we have considered averages as a representative number for samples of ratings from each movie, user, genre, and time regardless of how many observations each sample may have. It may result in predictions with a large estimated error in small sample sizes.

In order to address this issue, the use of a factor (λ) that penalizes small sample sizes and has little or no impact otherwise is required. This process is called regularization, and in this case, is defined as follows:

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

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

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

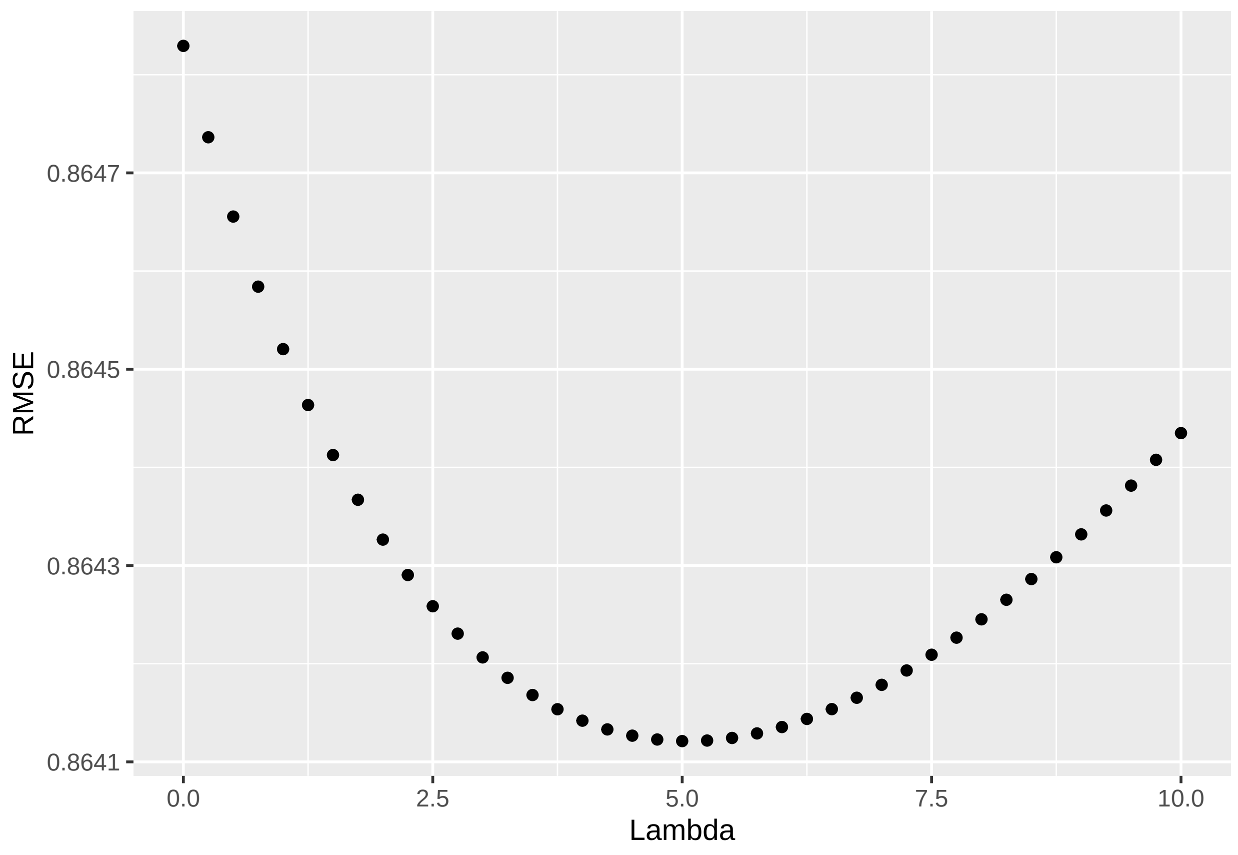
$$\hat{b}_t = \frac{1}{n_t + \lambda} \sum_{i=1}^{n_t} (y_{u,i} - \hat{b}_i - \hat{b}_u - \hat{b}_g - \hat{\mu})$$

The tuning parameter λ is usually estimated through simulations in which several values of λ are tested and the one that minimizes the RMSE is picked.

```r
#Finding the optimal number of lambda based on rmse results
lambdas <- seq(0,10,0.25)

rmse_all <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

  #Regularized movie effect
  b_i <- train_set %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(l+n()))

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

  #Regularized genre effect
  b_g <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - mu - b_i - b_u)/(l+n()))
  #Regularized time effect
  b_t <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(timediff) %>%
    summarise(b_t = sum(rating - mu - b_i - b_u - b_g)/(l+n()))

  #Predicted values
  predicted_rating <- cv_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_t, by = "timediff") %>%
    summarise(pred = mu + b_i + b_u + b_g + b_t) %>%
    .$pred

  #Accuracy
  return(RMSE(predicted_rating, cv_set$rating))

})
```

The λ that minimizes the RMSE is 5.

```
#RSME X lambdas
qplot(lambdas,rmse_all,ylab = "RMSE",xlab = "Lambda")
```



```r
#Optimal Lambda
lambda <- lambdas[which.min(rmse_all)]
```

With the optimal λ at hand, we can now estimate the values of all effects, and recalculate the model predictions.

```r
#Recalculate predictions with optimal lambda

#Regularized movie effect
b_i <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(lambda+n()))

#Regularized user effect
b_u <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating - mu - b_i)/(lambda+n()))

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

#Regularized time effect
b_t <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(timediff) %>%
  summarise(b_t = sum(rating - mu - b_i - b_u - b_g)/(lambda+n()))

#Predictions
predicted_rating <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_t, by = "timediff") %>%
  summarise(pred = mu + b_i + b_u + b_g + b_t) %>%
  .$pred
```

The regularization did not improve the RSME much. The final RSME is 0.865.

```r
#Accuracy
reg_rmse <- RMSE(test_set$rating, predicted_rating)

#Add model 6 results to tracking table
rmse_results <- rbind(rmse_results,
                      tibble(Method = "User + Movie + Genre + Time Effects +
                             Regularization",
                             RMSE = reg_rmse))
#Models RMSE results
```

```
rmse_results %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "14cm")
```

Method	**RMSE**
Just the average	1.0603099
Movie Effect	0.9437260
Movie + User Effects	0.8669568
Movie + User + Genre Effects	0.8666389
User + Movie + Genre + Time Effects	0.8662351
User + Movie + Genre + Time Effects + Regularization	0.8655429

For further data exploration, new data sets including all effects were built.

```
#Add effects into Training and Test Set
reg_train_set <- train_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_t, by = "timediff")

reg_test_set <- test_set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_t, by = "timediff")
```

Data Analysis II

The main goal in this second exploration step is to observe if there are remaining any important patterns that were not covered until now. To do that, we will study the residuals of the linear model developed, and see if there is still a correlation between different movies or users.

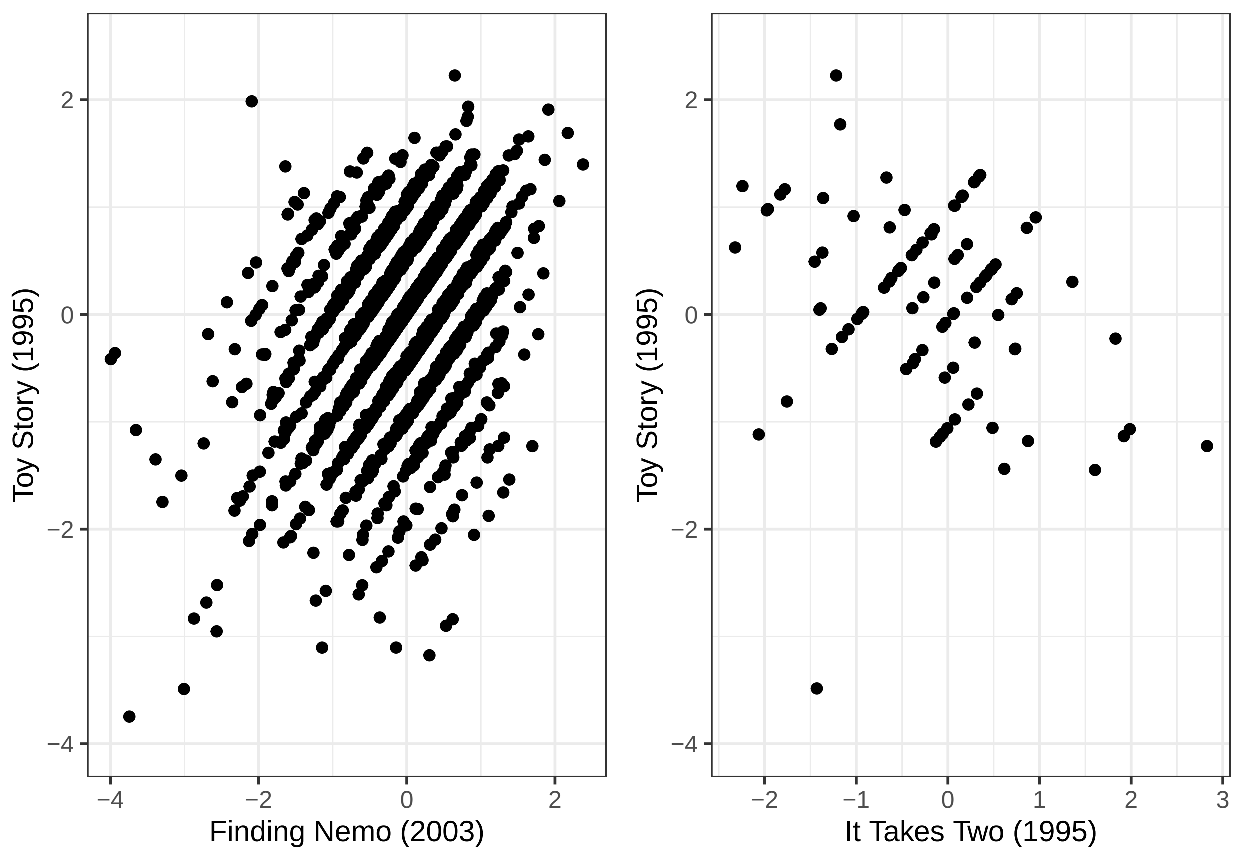
```r
#Data wrangling - Convert data.frame reg_train_set into matrix Users X Movies

#In order to have a better performance, we filtered movies with less than 100 rating,
#and users that rated less than 100 times out of the data set
reg_exp_set <- reg_train_set %>%
  group_by(movieId) %>%
  filter(n() >= 100) %>%
  ungroup() %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  ungroup()

#Calculate model residuals
residuals_set <- reg_exp_set %>%
  mutate(residual = rating - mu - b_i - b_u - b_g - b_t) %>%
  select(movieId, userId, residual)

#MovieId X UserId residual matrix
y <- residuals_set %>%
  spread(movieId, residual) %>%
  as.matrix()

#Name Colunms and Rows for better interpretation
rownames(y) <- y[, 1]
y <- y[ ,-1]

movie_titles <- reg_train_set %>%
  select(movieId, title) %>%
  distinct()
colnames(y) <- with(movie_titles, title[match(colnames(y), movieId)])
```

As we can see, many movies are highly correlated with each other. It could indicate the presence of structure in the residuals.

```r
#Correlation
corr <- round(cor(y, use = "pairwise.complete"), 2)

#Organize the correlation results in a table
corr %>%
  as.table %>%                                  # Start from the correlation matrix
  as.data.frame() %>%                           # Turn into 3-column table
  subset(Var1 != Var2 & abs(Freq) > 0.6) %>%    # Take only results highly correlated
  sample_n(20) %>%                              # Randomly choosing pair of movies
  arrange(Freq) %>%                             # Sort it
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "6cm")
```

Var1	**Var2**	**Freq**
Return to Never Land (2002)	When the Cat's Away (Chacun cherche son chat) (1996)	-1.00
Live Nude Girls (1995)	Knife in the Water (Nóz w wodzie) (1962)	-1.00
Switchblade Sisters (1975)	Thoroughly Modern Millie (1967)	-1.00
At Close Range (1986)	Bushwhacked (1995)	-1.00
Air Bud (1997)	Blow-Up (Blowup) (1966)	-0.78
Date Movie (2006)	Robin and Marian (1976)	-0.68
Communion (1989)	Suburban Commando (1991)	0.62
Little Women (1933)	Homegrown (1998)	0.63
Promise, The (La Promesse) (1996)	Shall We Dance? (2004)	0.66
Burnt Offerings (1976)	Family Guy Presents: Stewie Griffin - The Untold Story (2005)	0.67
Twisted (2004)	Radio (2003)	0.68
Mummy, The (1932)	Street Kings (2008)	0.75
Gun Shy (2000)	Double Impact (1991)	0.80
Drowning by Numbers (1988)	Stop! Or My Mom Will Shoot (1992)	0.80
Computer Wore Tennis Shoes, The (1969)	Return to Never Land (2002)	0.84
Tremors (1990)	Picture Bride (Bijo photo) (1994)	0.87
House Arrest (1996)	Mr. Bean's Holiday (2007)	0.97
Hoax, The (2007)	Faraway, So Close (In weiter Ferne, so nah!) (1993)	1.00
How I Won the War (1967)	Eight Below (2006)	1.00
Mighty Joe Young (1949)	Luther (2003)	1.00

```r
#Positive correlation
g1 <- as.data.frame(y) %>%
  ggplot(aes(x = `Finding Nemo (2003)`, y = `Toy Story (1995)`)) +
  geom_point() +
  theme_bw() +
  xlab("Finding Nemo (2003)") +
  ylab("Toy Story (1995)")

#Negative correlation
g2 <- as.data.frame(y) %>%
  ggplot(aes(x = `It Takes Two (1995)`, y = `Toy Story (1995)`)) +
  geom_point() +
  theme_bw() +
  xlab("It Takes Two (1995)") +
  ylab("Toy Story (1995)")

gridExtra::grid.arrange(g1,g2,ncol = 2)
```



If all variations in the data were due to movie to movie, user to user, genre to genre, and time to time differences, after we fitted the model, all observations should be independent of each other. But, it doesn't happen in the residuals. Many movies are highly correlated to each other.

Modeling II

Based on the results exposed in the residuals matrix, we could see that there are still important sources of variability that were not covered in our previous model. Some structures are closely related to similar movie groups having similar rating patterns. Here, we will try matrix factorization, which addresses this issue by attributing different factors to distinct patterns found in the data.

In a more technical view, a matrix factorization is an approach, which approximates the matrix R_{mxn} by the product of two matrices of lower dimension P_{kxm} and Q_{kxn}, $R \approx P' \times Q$, where m represent the users, and n the movies. A typical solution for P and Q is given by the optimization problem described in https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html.

Model 7 - Random variation + Genre Effect + Time Effect + Movie Effect + User Effect + Regularization + Parallel Matrix Factorization

Here, we included the residuals into the model. The current model is defined as following:

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

Where,

$$r_{u,i} = p_{u,1}q_{1,i} + p_{u,2}q_{2,i} + \cdots + p_{u,m}q_{m,i} = P'Q$$

```r
#Calculate residuals, and build train, cv and test sets
mf_train <- reg_train_set %>%
  mutate(residual = rating - mu - b_i - b_u - b_g - b_t)

mf_test <- reg_test_set %>%
  mutate(residual = rating - mu - b_i - b_u - b_g - b_t)
```

The residuals were estimated through matrix factorization by the recosystem package.

- The usage of the recosystem is summarize below:
  - Create a model object (a Reference Class object in R) by calling Reco().
  - Call the $tune() method to select best tuning parameters along a set of candidate values.
  - Train the model by calling the $train() method.
  - Export the model via $output().
  - Use the $predict() method to compute predicted values.

```r
#In recosystem, a DataSource class object is used as input data.
#In other words, the training and the test set have to be
#converted to a DataSource object prior to modelling.
set.seed(40, sample.kind = "Rounding")
train_data <-  with(mf_train, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating     = residual))
test_data  <-  with(mf_test,  data_memory(user_index = userId,
                                           item_index = movieId,
                                           rating     = residual))
```

```r
#Create the model object
r <- Reco()
```

```r
#Define the best tuning parameters
tune <- r$tune(train_data, opts = list(dim = c(10, 20, 30), lrate = 0.1,
                                       costp_l1 = 0, costq_l1 = 0,
                                       nthread = 2, niter = 10))
```

```r
# Train the algorithm
r$train(train_data, opts = c(tune$min, nthread = 4, niter = 20))

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

```r
#Predicted_values
mf_test <- mf_test %>%
  mutate(y_hat = y_hat + mu + b_i + b_u + b_g + b_t)
```

A substantial improvement was achieve by matrix factorization. The final RSME is 0.796.

```r
#Accuracy
mf_rmse <- RMSE(mf_test$rating, mf_test$y_hat)

#Add model 6 results to tracking table
rmse_results <- rbind(rmse_results,
                      tibble(Method = "User + Movie + Genre + Time Effects
                             + Regularization + Matrix Factorization",
                             RMSE = mf_rmse))

#Models RMSE results
rmse_results %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "14cm")
```

Method	**RMSE**
Just the average	1.0603099
Movie Effect	0.9437260
Movie + User Effects	0.8669568
Movie + User + Genre Effects	0.8666389
User + Movie + Genre + Time Effects	0.8662351
User + Movie + Genre + Time Effects + Regularization	0.8655429
User + Movie + Genre + Time Effects + Regularization + Matrix Factorization	0.7965661

Model 8 - Parallel Matrix Factorization

The past several models were all based on a model-based approach. It means that all patterns were estimated by single models one by one. Here, we will let all variability be estimated by one single model, the matrix factorization.

```
set.seed(20, sample.kind = "Rounding")
train_data <-  with(mf_train, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating     = rating))
test_data  <-  with(mf_test,  data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating     = rating))
```

```
#Create the model object
r <- Reco()
```

```
#Define the best tuning parameters
tune <- r$tune(train_data, opts = list(dim = c(10, 20, 30), lrate = 0.1,
                                       costp_l1 = 0, costq_l1 = 0,
                                       nthread = 2, niter = 10))
```

```
# Train the algorithm
r$train(train_data, opts = c(tune$min, nthread = 5, niter = 25))

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

As we can see, the parallel matrix factorization algorithm did a slightly better job representing all patterns in our data than the model-based approach. It is also much more efficient.

```
#Accuracy
reco_rmse <- RMSE(mf_test$rating, y_hat)

#Add model 6 results to tracking table
rmse_results <- rbind(rmse_results,
                      tibble(Method = "Parallel Matrix Factorization",
                             RMSE = reco_rmse))

#Models RMSE results
rmse_results %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "14cm")
```

Method	RMSE
Just the average	1.0603099
Movie Effect	0.9437260
Movie + User Effects	0.8669568
Movie + User + Genre Effects	0.8666389
User + Movie + Genre + Time Effects	0.8662351
User + Movie + Genre + Time Effects + Regularization	0.8655429
User + Movie + Genre + Time Effects + Regularization + Matrix Factorization	0.7965661
Parallel Matrix Factorization	0.7899910

Final Model - Evaluation

Based on the algorithms results, we can see that the matrix factorization alone did the best job predicting user ratings in the 10M version of the MovieLens dataset. Here, we are going to use the entire data set, and perform the final hold-out test.

```
#For final evaluation, we used the edx and validation data sets.
set.seed(30, sample.kind = "Rounding")
train_data <-  with(edx, data_memory(user_index = userId,
                                     item_index = movieId,
                                     rating     = rating))
test_data  <-  with(validation,  data_memory(user_index = userId,
                                             item_index = movieId,
                                             rating     = rating))
```

```
#Create the model object
r <- Reco()
```

```
#Define the best tuning parameters
tune <- r$tune(train_data, opts = list(dim = c(10, 20, 30), lrate = 0.1,
                                       costp_l1 = 0, costq_l1 = 0,
                                       nthread = 2, niter = 10))
```

```
# Train the algorithm
r$train(train_data, opts = c(tune$min, nthread = 5, niter = 25))

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

The RSME accuracy of the best model performed in this project is 0.781.

```
#Accuracy
final_rmse <- RMSE(validation$rating, y_hat)

#Add model 6 results to tracking table
rmse_results <- rbind(rmse_results,
                      tibble(Method = "Final Model",
                             RMSE = final_rmse))

#Models RMSE results
rmse_results %>%
  kbl(booktabs = T) %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(row = 0, bold = T) %>%
  column_spec(1, width = "14cm")
```

Method	**RMSE**
Just the average	1.0603099
Movie Effect	0.9437260
Movie + User Effects	0.8669568
Movie + User + Genre Effects	0.8666389
User + Movie + Genre + Time Effects	0.8662351
User + Movie + Genre + Time Effects + Regularization	0.8655429
User + Movie + Genre + Time Effects + Regularization + Matrix Factorization	0.7965661
Parallel Matrix Factorization	0.7899910
Final Model	0.7814162

Conclusion

This project has examined a couple of predictive techniques applied to a movie review data set. Through data exploration, we have seen that all predictors appeared to present a clear variability of rating values. Those remarks were used while modeling. After evaluating each model, the biggest improvements were achieved due to the movie and user effects. Regularization did not improve considerably the RMSE, but it was performed anyway. Using this final model we achieved an RMSE of 0.865. After we fitted our model, a new analysis in the residuals was performed. During the data exploration analysis, we observed a strong correlation between distinct movies, which highlighted structures in the residuals that were not addressed so far. It led us to apply parallel matrix factorization. This technique improved significantly the accuracy of our model which was able to achieve an RMSE of 0.781.

Future work will focus on applying new algorithms present in the recommenderlab package, such as user-based and item-based collaborative filtering. Also, we did not take into account all possible arrange of predictors present in the movielens data set, and further analysis of it will be done later. As an example, the timestamp and release year could be studied separately, and the genres split into single units.