Capstone: MovieLens Project

Eden Ósk Eyjólfsdóttir 2025

This project develops a movie recommendation system utilizing the 10M version of the MovieLens dataset. It incorporates movie and user effects alongside 'title' and 'timestamp' variables to extract release years and calculate a new rate timestamp indicator. These elements are combined into a final model, regularized with a tuning parameter of 0.5 selected through cross-validation. The model achieves a residual mean squared error (RMSE) of approximately 0.8647 on the Validation Set, surpassing the project's target.

Keywords: MovieLens, recommendation system, EDA, capstone, R Markdown

1 Introduction

MovieLens Recommendation System Project

Introduction

MovieLens is an online platform operated by GroupLens, a research lab at the University of Minnesota. It serves as a collaborative movie recommendation system that leverages machine learning tools to analyze users' movie ratings and reviews, generating personalized recommendations based on their preferences or taste profiles (Harper & Konstan, 2015).

This project utilizes the **MovieLens 10M dataset**, which includes:

- **10,000,054 ratings** and **95,580 tags** applied to **10,681 movies** by **71,567 users**.
- Users were randomly selected, ensuring each had rated at least 20 movies.
- No demographic data is included; users are identified solely by unique IDs.
- The dataset is organized into three files: `movies.dat`, `ratings.dat`, and `tags.dat`, along with scripts for generating subsets to support five-fold cross-validation of rating predictions.

The project's objective is to develop a predictive model that minimizes the loss function by incorporating the following key effects:

- **Movie Effect:** Predict ratings based on evaluations from other users.
- **User Effect:** Factor in a user's typical rating behavior.
- **Release Year Effect:** Group movies by their release year to account for their yearly competition.

- **Rate Timestamp Effect:** Measure the time elapsed between a movie's release and its rating to reflect its gained popularity over time.

The model is further refined using regularization with a tuning parameter chosen through cross-validation, penalizing large estimates formed from small sample sizes (Irizarry, 2020). A residual mean squared error (RMSE) is used as the performance metric, representing the average prediction error.

---

2. Introduction

1. **Data Extraction**
   The dataset was downloaded using the code provided in the course. After extraction, the training set (`edx`) was prepared by ensuring it contained only relevant user-movie pairs.
2. **Exploratory Data Analysis (EDA)**
   The training set was analyzed using descriptive statistics and visualizations (e.g., histograms) to gain insights into rating distributions and user/movie behaviors.
3. **Feature Engineering**
   The dataset was enhanced by extracting release year from movie titles and calculating the rate timestamp (years between a movie's release and its rating). These variables were incorporated as additional predictors in the model.
4. **Model Development**
   The model was built step-by-step, incorporating the movie, user, release year, and rate timestamp effects. Regularization was applied by tuning the parameter λλ to minimize the RMSE. Each iteration was benchmarked against the project goal of **RMSE < 0.8649**.

2.1 Data Extraction

The extraction process followed the guidelines provided in the **Course/Capstone Project: All Learners/Project Overview: MovieLens** section of the **PH125.9x - Capstone: MovieLens Project** course.

This structured approach ensures a robust model capable of accurately predicting movie ratings while identifying areas for future improvement.

```r
# Import libraries
rm(list=ls())

if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ggplot2))
  install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(knitr))
  install.packages("knitr", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
```

```r
library(data.table)
library(ggplot2)
library(knitr)

# Get from URL or download file in Documents
zip_file <- "~/ml-10m.zip"
if(file.exists(zip_file)) {dl <- zip_file} else {
  dl <- tempfile()
  download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip",
                dl)
}
```

2.2. Data Description

The `edx` training set consists of **9,000,055 rows** and **6 columns**, with the following variables:

1. **userId**: A unique identifier for the user providing the rating.
2. **movieId**: A unique identifier for the movie being rated.

3. **title**: The title of the movie.
4. **genres**: The genre(s) associated with the movie.
5. **timestamp**: The date and time when the rating was recorded.
6. **rating**: The rating provided by the user, ranging from **0.5** to **5.0 stars**.

```
dim(edx)
```

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

```
head(edx) %>% as.tibble()
```

```
## # A tibble: 6 x 6
##   userId movieId rating timestamp title                  genres
##    <int>   <dbl>  <dbl>     <int> <chr>                  <chr>
## 1      1     122      5 838985046 Boomerang (1992)       Comedy|Romance
## 2      1     185      5 838983525 Net, The (1995)        Action|Crime|Thriller
## 3      1     292      5 838983421 Outbreak (1995)        Action|Drama|Sci-Fi|T~
## 4      1     316      5 838983392 Stargate (1994)        Action|Adventure|Sci-~
## 5      1     329      5 838983392 Star Trek: Generations~ Action|Adventure|Dram~
## 6      1     355      5 838984474 Flintstones, The (1994) Children|Comedy|Fanta~
```

We can analyze the train set using some descriptive statistics.

```
summary(edx)
```

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

Notice that there are 69,878 and 10,677 different users and movies, respectively.

```
summarize(edx,
          unique_users = n_distinct(userId),
          unique_movies = n_distinct(movieId))
```

unique_users unique_movies

1 69878 10677

Here we can see the top 50 best ranking movies, and how many votes each one received.

```
edx %>% group_by(movieId) %>%
  summarise(stars = mean(rating),
            votes = n()) %>%
  left_join(edx, by = "movieId") %>%
  select(movieId, title, votes, stars) %>%
  unique() %>%
  filter(title != "NA") %>%
  arrange(desc(stars)) %>%
  slice(1:50) %>%
  mutate(ranking = 1:n()) %>%
  select(ranking, everything()) %>%
  kable()
```

ranking	movieId	title	votes	stars
20	56015	Kansas City Confidential (1952)	1	4.500000
21	58185	Please Vote for Me (2007)	1	4.500000
22	60336	Bad Blood (Mauvais sang) (1986)	1	4.500000
23	60990	End of Summer, The (Kohayagawa-ke no aki) (1961)	3	4.500000
24	61695	Ladrones (2007)	1	4.500000
25	63179	Tokyo! (2008)	1	4.500000
26	64418	Man Named Pearl, A (2006)	1	4.500000
27	318	Shawshank Redemption, The (1994)	28015	4.455131
28	32792	Red Desert, The (Deserto rosso, Il) (1964)	6	4.416667
29	858	Godfather, The (1972)	17747	4.415366
30	32657	Man Who Planted Trees, The (Homme qui plantait des arbres, L') (1987)	5	4.400000
31	50	Usual Suspects, The (1995)	21648	4.365854
32	527	Schindler's List (1993)	23193	4.363493
33	31309	Rocco and His Brothers (Rocco e i suoi fratelli) (1960)	3	4.333333
34	58808	Last Hangman, The (2005)	3	4.333333
35	912	Casablanca (1942)	11232	4.320424
36	904	Rear Window (1954)	7935	4.318651
37	922	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	2922	4.315880
38	1212	Third Man, The (1949)	2967	4.311426
39	3435	Double Indemnity (1944)	2154	4.310817
40	1178	Paths of Glory (1957)	1571	4.308721
41	2019	Seven Samurai (Shichinin no samurai) (1954)	5190	4.306744
42	1221	Godfather: Part II, The (1974)	11920	4.301971
43	58559	Dark Knight, The (2008)	2353	4.297068
44	750	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)	10627	4.295333
45	1193	One Flew Over the Cuckoo's Nest (1975)	13014	4.293261
46	44555	Lives of Others, The (Das Leben der Anderen) (2006)	1108	4.291065
47	3030	Yojimbo (1961)	1528	4.281741
48	1148	Wallace & Gromit: The Wrong Trousers (1993)	7167	4.275429
49	745	Wallace & Gromit: A Close Shave (1995)	5690	4.275308
50	1260	M (1931)	1926	4.274662

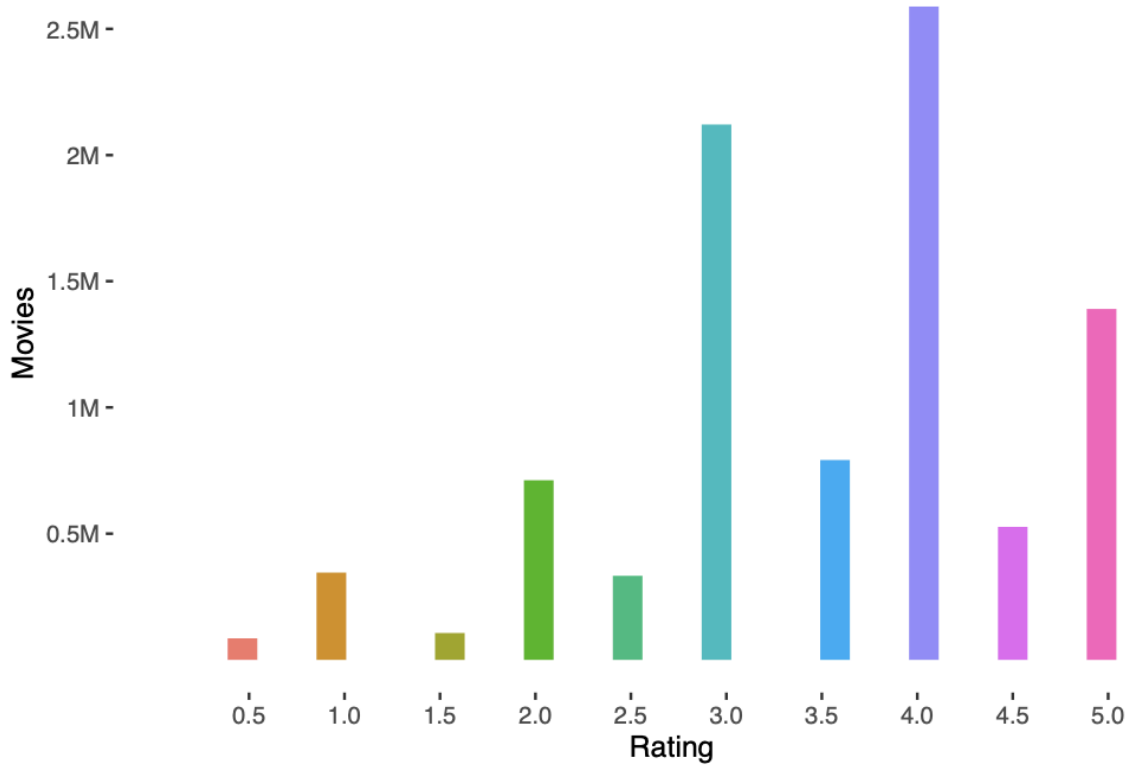
2.3 Data visualization

The most popular ratings are 4.0 and 3.0, while half stars ratings are less common.

```
# Rating histogram
y_cuts <- c(0.5, 1.0, 1.5, 2.0, 2.5)
edx %>%
  ggplot(aes(rating, fill = cut(rating, 100))) +
  geom_histogram(bins = 30, show.legend = F) +
  theme_update() +
  labs(x = "Rating", y = "Movies") +
```

```
  ggtitle("Figure 2.1: Rating distribution per movies") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white")) +
  scale_x_discrete(limits = c(seq(min(edx$rating),
                                  max(edx$rating),
                                  max(edx$rating)/n_distinct(edx$rating))))+
  scale_y_continuous(labels = paste0(y_cuts, "M"),
                     breaks = y_cuts * 10^6)
```

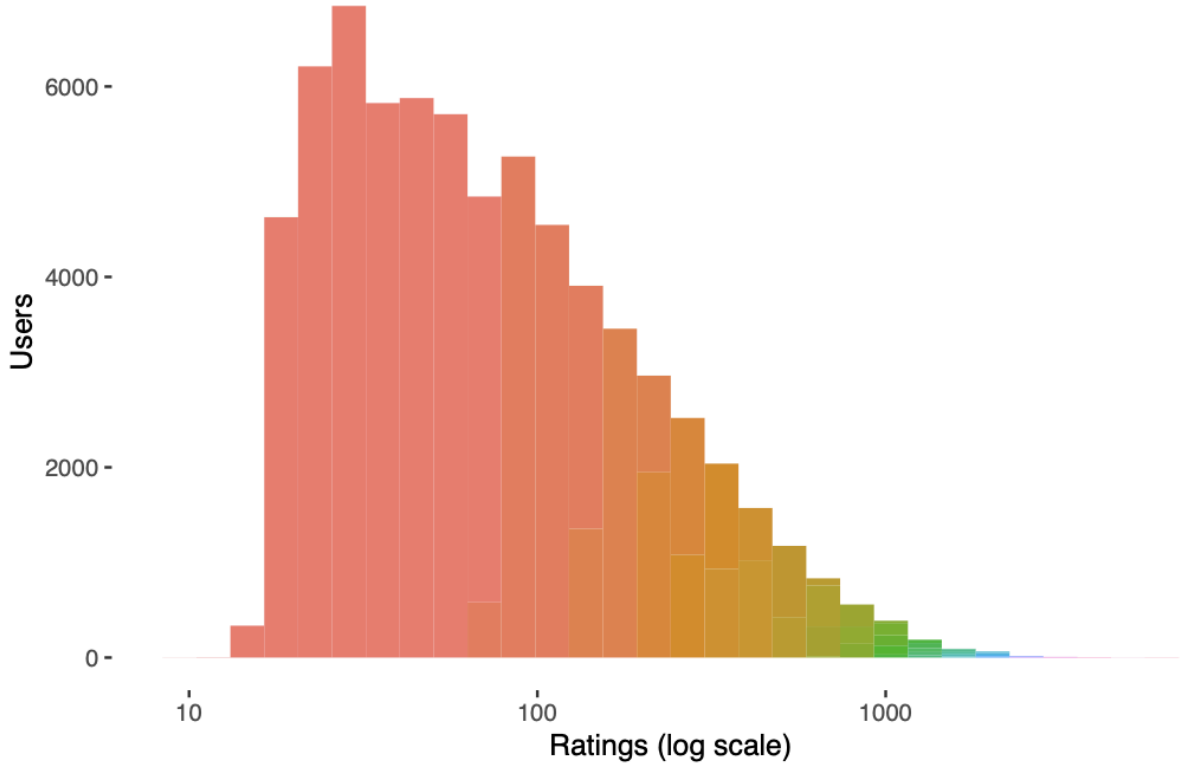
Figure 2.1: Rating distribution per movies



Users usually rate less than 100 different movies, and very few more than 1000.

```
# Ratings per users histogram
edx %>% count(userId) %>%
  ggplot(aes(n, fill = cut(n, 100))) +
  geom_histogram(bins = 30, show.legend = F) +
  theme_update() +
  labs(x = "Ratings (log scale)", y = "Users") +
  ggtitle("Figure 2.2: Ratings per users") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white")) +
  scale_x_log10()
```

Figure 2.2: Ratings per users

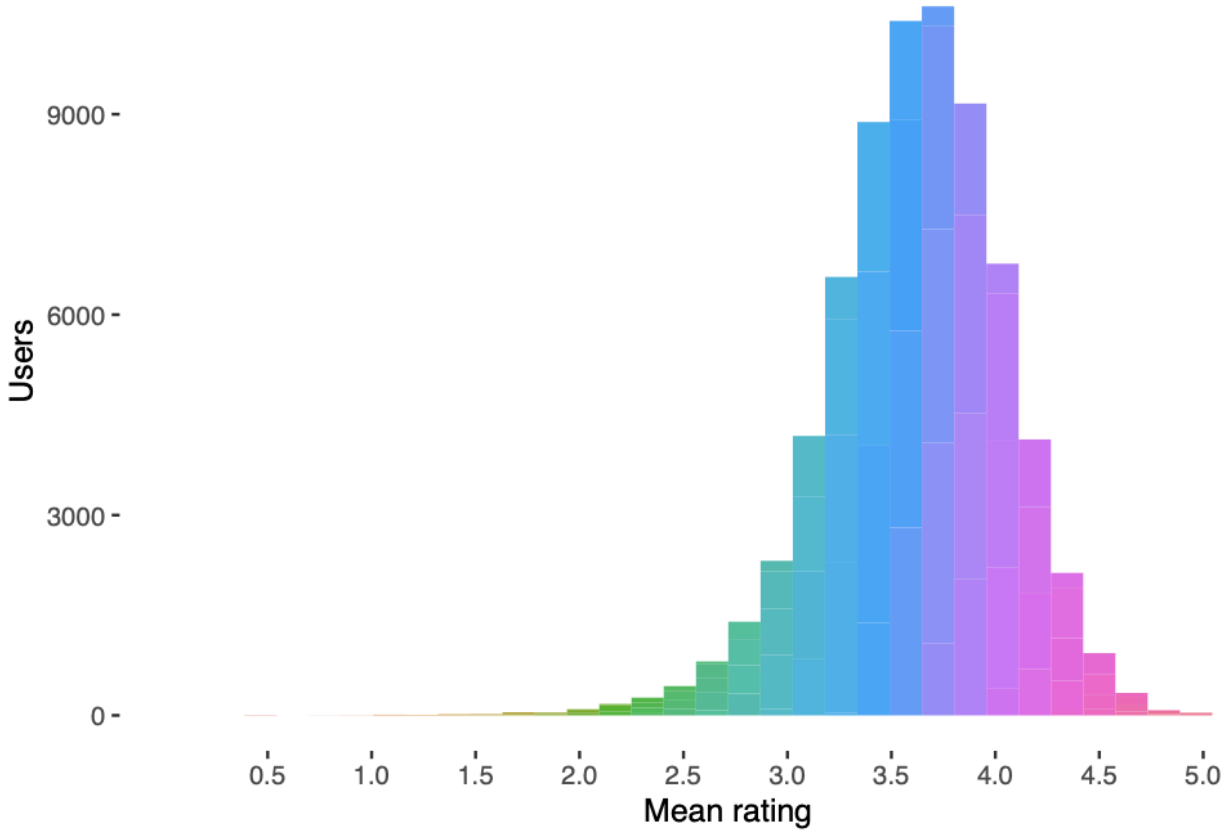


Mean rating is between 3.5 and 4.0.

```r
# Mean rating per users histogram
edx %>% group_by(userId) %>%
  summarise(m = mean(rating)) %>%
  ggplot(aes(m, fill = cut(m, 100))) +
  geom_histogram(bins = 30, show.legend = F) +
  theme_update() +
  labs(x = "Mean rating", y = "Users") +
  ggtitle("Figure 2.3: Mean rating per users") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white")) +
  scale_x_discrete(limits = c(seq(min(edx$rating),
                                  max(edx$rating),
                                  max(edx$rating)/n_distinct(edx$rating))))
```

Warning: Continuous limits supplied to discrete scale.
Did you mean `limits = factor(...)` or `scale_*_continuous()`?

Figure 2.3: Mean rating per users

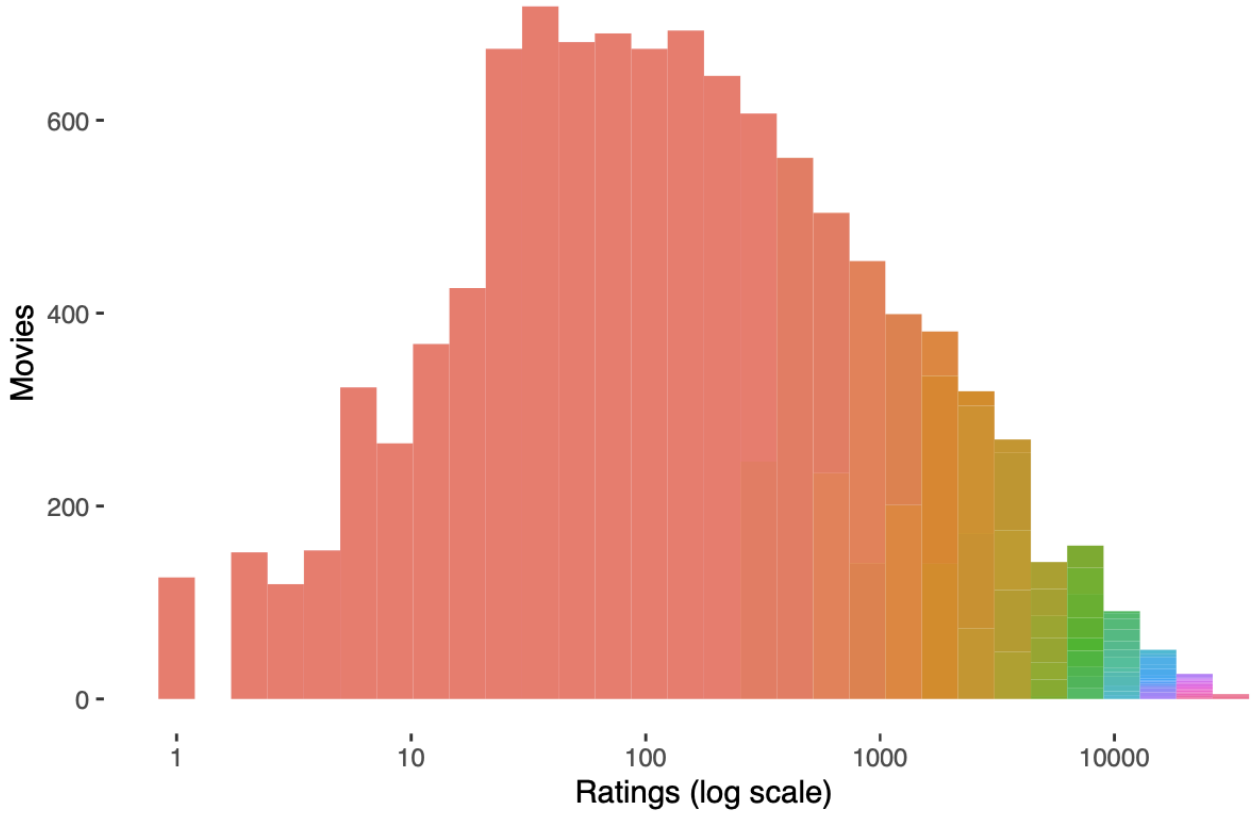


On the other hand, less than ten movies have received less than ten ratings. But also less than

400 have received more than 1000 ratings.

```
# Ratings per movies histogram
edx %>% count(movieId) %>%
  ggplot(aes(n, fill = cut(n, 100))) +
  geom_histogram(bins = 30, show.legend = F) +
  theme_update() +
  labs(x = "Ratings (log scale)", y = "Movies") +
  ggtitle("Figure 2.4: Ratings per movies") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white")) +
  scale_x_log10()
```

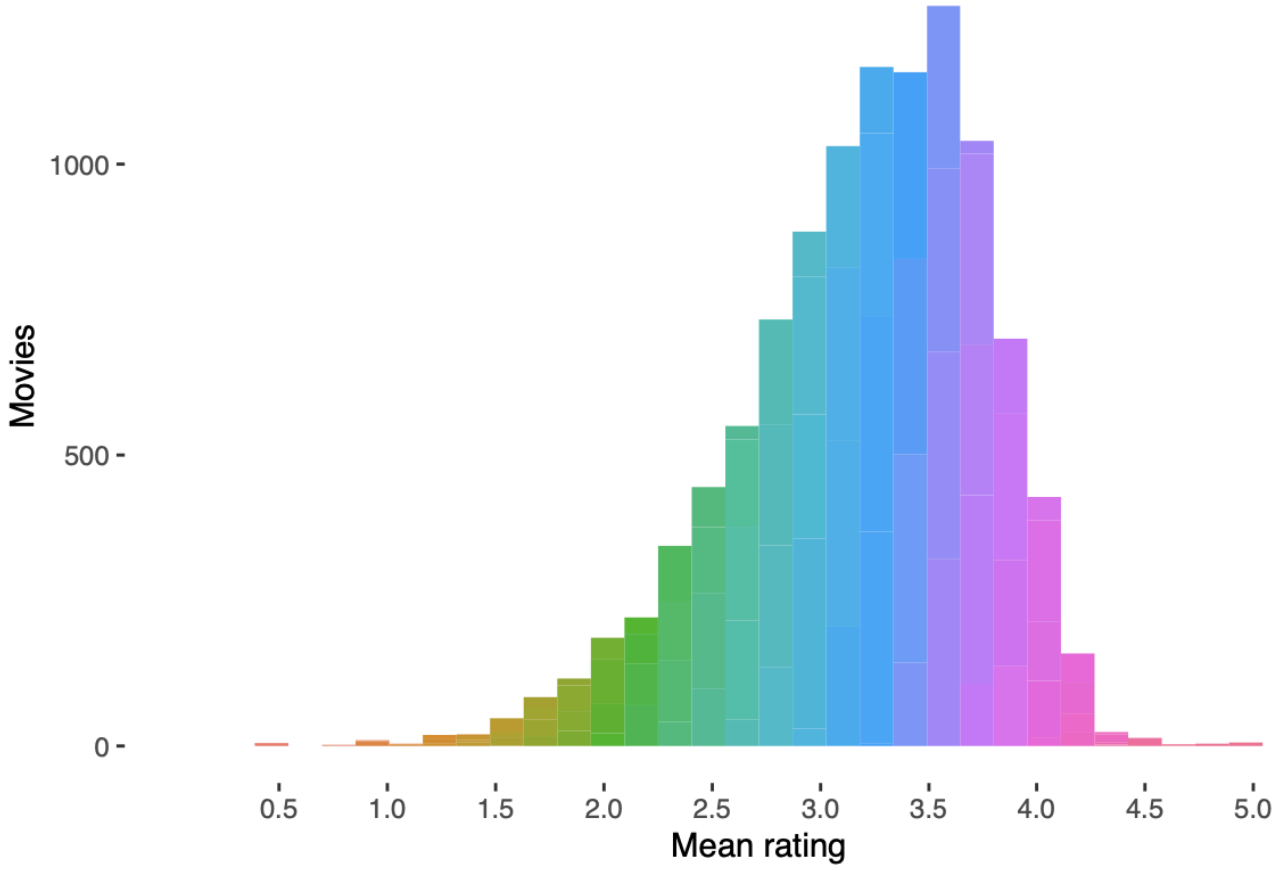
Figure 2.4: Ratings per movies



Mean rating is near 3.5.

```r
# Mean rating per movies histogram
edx %>% group_by(movieId) %>%
  summarise(m = mean(rating)) %>%
  ggplot(aes(m, fill = cut(m, 100))) +
  geom_histogram(bins = 30, show.legend = F) +
  theme_update() +
  labs(x = "Mean rating", y = "Movies") +
  ggtitle("Figure 2.5: Mean rating per movies") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white")) +
  scale_x_discrete(limits = c(seq(min(edx$rating),
                                  max(edx$rating),
                                  max(edx$rating)/n_distinct(edx$rating))))
```

Figure 2.5: Mean rating per movies



2.4 Data calculations

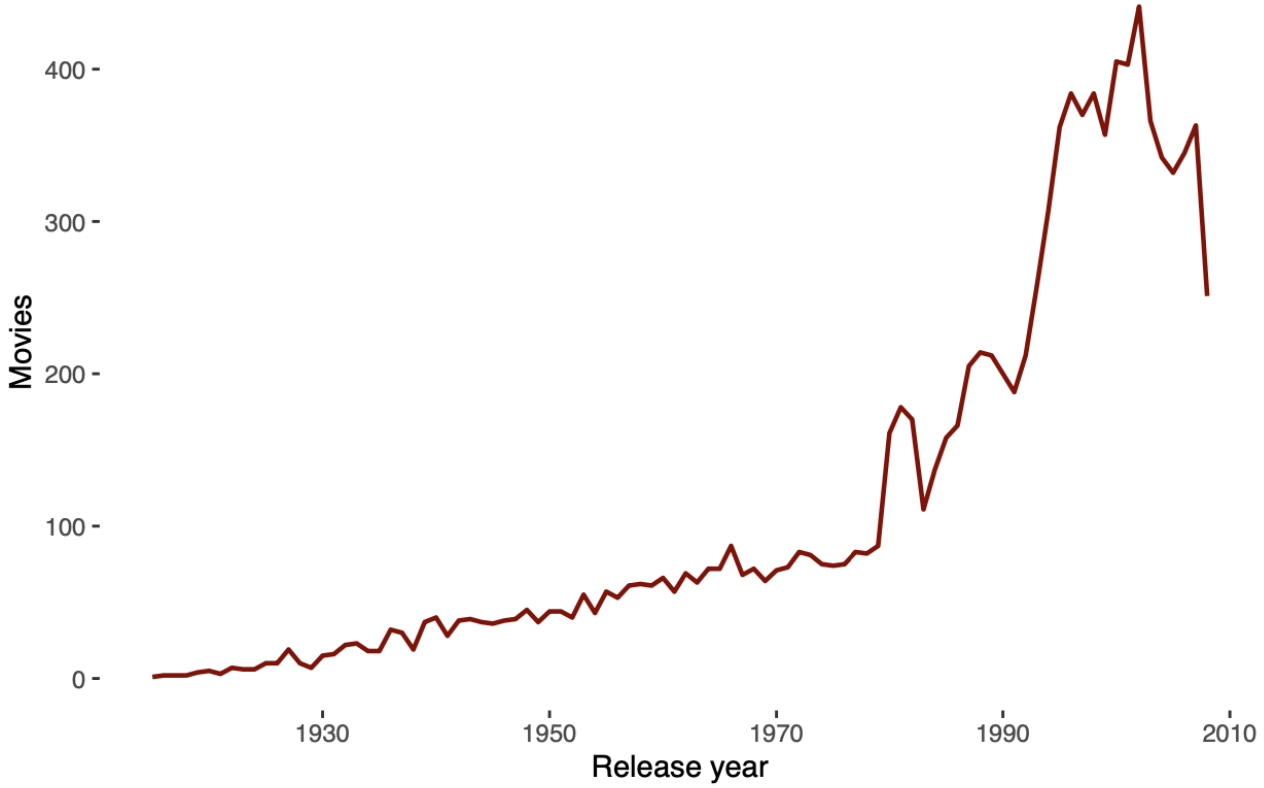
Notice that the data's timestamp can be transformed using January 1st, 1970 as initial date. We
can also extract the release year from the title, and using both years we can calculate the rate timestamp.

```
# Dates addition
edx <- edx %>%
  mutate(timestamp = as.POSIXct(timestamp,
                                origin = "1970-01-01",
                                tz = "GMT"),
         year_movie = as.numeric(substr(title,
                                        nchar(title)-4,
                                        nchar(title)-1)),
         year_rated = as.numeric(format(timestamp, "%Y")),
         rate_ts = year_rated - year_movie)
```

There are less than 100 movies per year for movies released before 1980, but more than 200 for those released after 1990.

```
edx %>%
  select(movieId, year_movie) %>%
  unique() %>%
  group_by(year_movie) %>%
  summarise(count = n()) %>%
  ggplot(aes(year_movie, count)) +
  geom_line(color = "darkred",
            size = 0.8) +
  labs(x = "Release year", y = "Movies") +
  ggtitle("Figure 2.6: Release year per movies") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white"))
```

Figure 2.6: Release year per movies



2.5 Data modeling

Now we make a naive prediction based on the average rating, and from there, we can build the model adding movie, user, release year and rate timestamp effects. Finally, we regularize these
effects to minimize the RMSE.

```
# Summary statistics
mu <- mean(edx$rating)
```

The average rating (μ= 3.5124652) is used to make a naive prediction. We obtain RMSE = 1.06,
meaning our predictions are on average more than one star far from the actual value.

```
# Average prediction
rmse_naive <- RMSE(edx$rating, mu)
rmse_results <- tibble(Model = "01. Average only",
                       RMSE = rmse_naive,
                       Goal = ifelse(rmse_naive < 0.8649, "Under", "Over"))
```

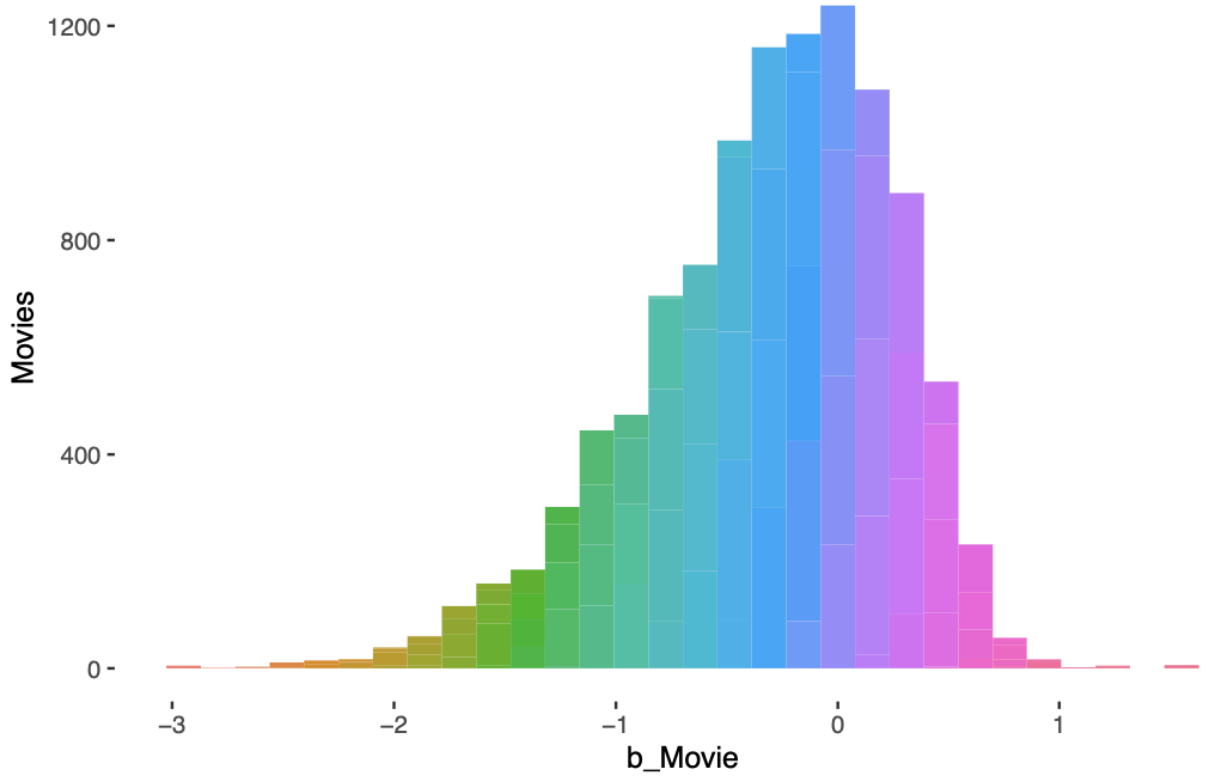
Model	RMSE	Goal
01. Average only	1.060331	Over

Now we will start incorporating the movie effect, which generates a RMSE = 0.9423.

```
# Movie effect prediction
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarise(b_Movie = mean(rating - mu))

movie_avgs %>%
  ggplot(aes(b_Movie, fill = cut(b_Movie, 100))) +
  geom_histogram(bins = 30, show.legend = F) +
  theme_update() +
  labs(x = "b_Movie", y = "Movies") +
  ggtitle("Figure 3.1: Movies per computed b_Movie") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white"))
```

Figure 3.1: Movies per computed b_Movie




```
predicts_movie <- mu + edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_Movie)

rmse_model1 <- RMSE(predicts_movie, edx$rating)

rmse_results <- rmse_results %>%
  bind_rows(tibble(Model = "02. Movie Effect",
                   RMSE = rmse_model1,
                   Goal = ifelse(rmse_model1 < 0.8649, "Under", "Over")))
```

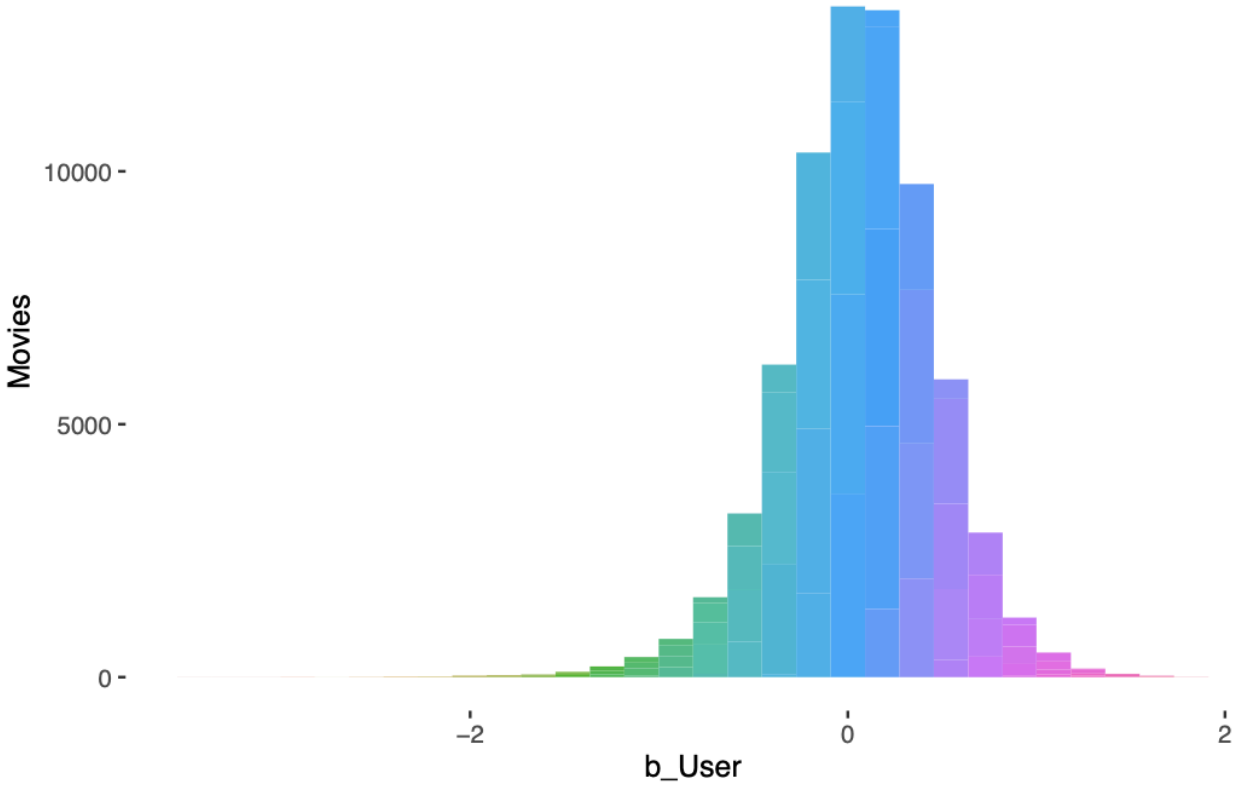
Model	RMSE	Goal
01. Average only	1.0603313	Over
02. Movie Effect	0.9423475	Over

Then we add the user effect. With this two, we get a RMSE = 0.8567, which is lower than the target.

```
# Movie and user effect prediction
user_avgs <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_User = mean(rating - b_Movie - mu))

user_avgs %>%
  ggplot(aes(b_User, fill = cut(b_User, 100))) +
  geom_histogram(bins = 30, show.legend = F) +
  theme_update() +
  labs(x = "b_User", y = "Movies") +
  ggtitle("Figure 3.2: Movies per computed b_User") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white"))
```

Figure 3.2: Movies per computed b_User




```
predicts_movie_user <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_Movie + b_User) %>%
  pull(pred)

rmse_model2 <- RMSE(predicts_movie_user,
                    edx$rating)
rmse_results <- rmse_results %>%
```

```
  bind_rows(tibble(Model = "03. +User Effect",
                   RMSE = rmse_model2,
                   Goal = ifelse(rmse_model2 < 0.8649, "Under", "Over")))
```

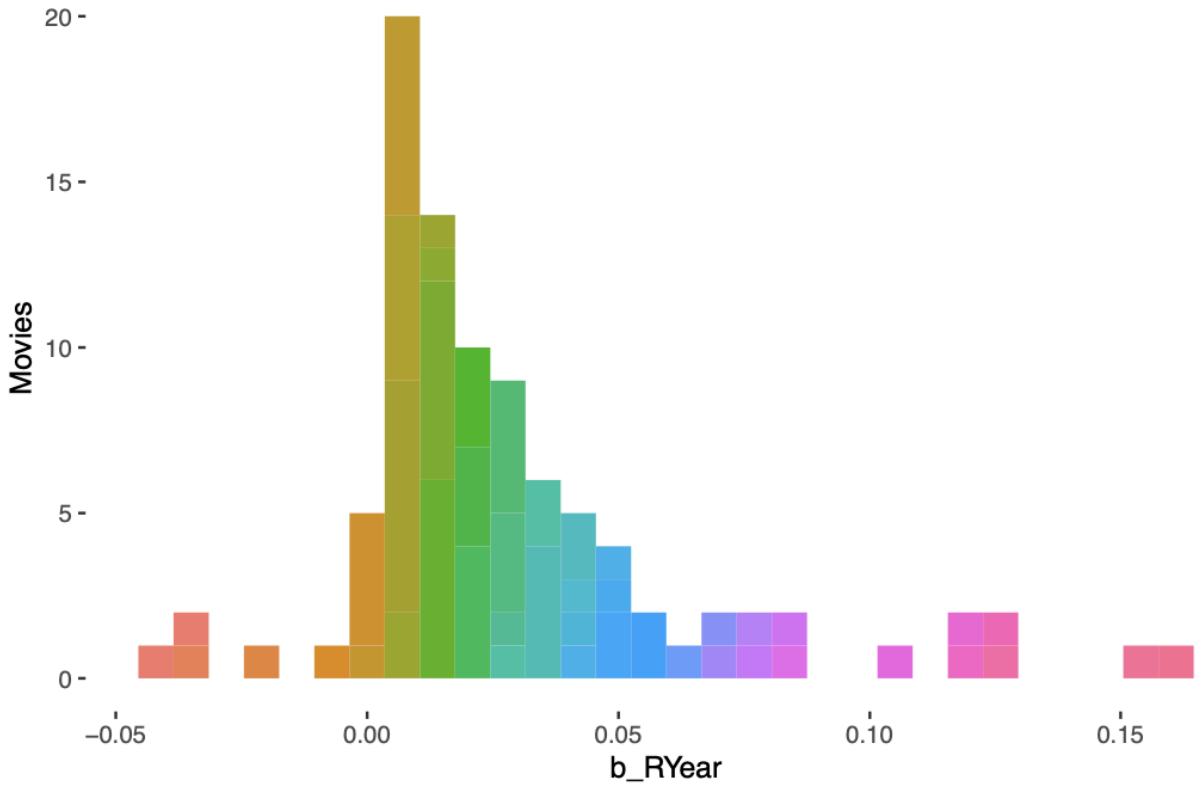
Model	RMSE	Goal
01. Average only	1.0603313	Over
02. Movie Effect	0.9423475	Over
03. +User Effect	0.8567039	Under

Then, we incorporate the release year effect, getting a RMSE = 0.8564. We can see how β RYear
values are approaching 0.

```r
# Movie, user and release year effect prediction
RYear_avgs <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(year_movie) %>%
  summarise(b_RYear = mean(rating - b_User - b_Movie - mu))

RYear_avgs %>%
  ggplot(aes(b_RYear, fill = cut(b_RYear, 100))) +
  geom_histogram(bins = 30, show.legend = F) +
  theme_update() +
  labs(x = "b_RYear", y = "Movies") +
  ggtitle("Figure 3.3: Movies per computed b_RYear") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white"))
```

Figure 3.3: Movies per computed b_RYear




```
predicts_movie_user_RYear <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(RYear_avgs, by = "year_movie") %>%
  mutate(pred = mu + b_Movie + b_User + b_RYear) %>%
  pull(pred)

rmse_model3 <- RMSE(predicts_movie_user_RYear,
                    edx$rating)
rmse_results <- rmse_results %>%
  bind_rows(tibble(Model = "04. +RYear Effect",
                   RMSE = rmse_model3,
                   Goal = ifelse(rmse_model3 < 0.8649, "Under", "Over")))
```

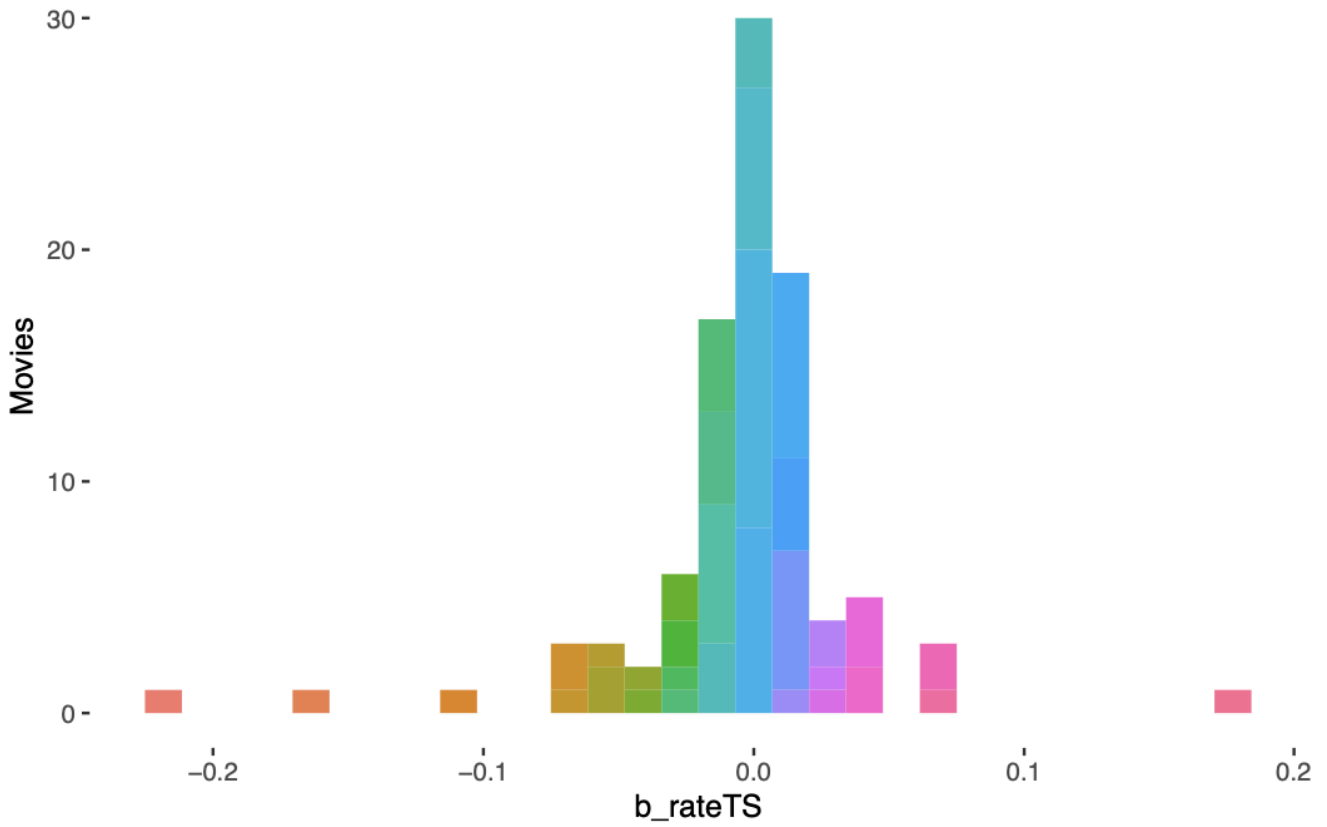
Model	RMSE	Goal
01. Average only	1.0603313	Over
02. Movie Effect	0.9423475	Over
03. +User Effect	0.8567039	Under
04. +RYear Effect	0.8563777	Under

Our fourth and final effect is rate timestamp, giving a RMSE = 0.8561. Most values of β rateTS are closer to 0 from less than 0.1.

```
# Movie, user, release year and rate timestamp effect prediction
rateTS_avgs <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(RYear_avgs, by = "year_movie") %>%
  group_by(rate_ts) %>%
  summarise(b_rateTS = mean(rating - b_RYear - b_User - b_Movie - mu))

rateTS_avgs %>%
  ggplot(aes(b_rateTS, fill = cut(b_rateTS, 100))) +
  geom_histogram(bins = 30, show.legend = F) +
  theme_update() +
  labs(x = "b_rateTS", y = "Movies") +
  ggtitle("Figure 3.4: Movies per computed b_rateTS") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white"))
```

Figure 3.4: Movies per computed b_rateTS




```
predicts_movie_user_RYear_rateTS <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
```

```
  left_join(user_avgs, by = "userId") %>%
  left_join(RYear_avgs, by = "year_movie") %>%
  left_join(rateTS_avgs, by = "rate_ts") %>%
  mutate(pred = mu + b_rateTS + b_RYear + b_User + b_Movie) %>%
  pull(pred)

rmse_model4 <- RMSE(predicts_movie_user_RYear_rateTS,
                    edx$rating)
rmse_results <- rmse_results %>%
  bind_rows(tibble(Model = "05. +RateTS Effect",
                   RMSE = rmse_model4,
                   Goal = ifelse(rmse_model4 < 0.8649, "Under", "Over")))
```

Model	RMSE	Goal
01. Average only	1.0603313	Over
02. Movie Effect	0.9423475	Over
03. +User Effect	0.8567039	Under
04. +RYear Effect	0.8563777	Under
05. +RateTS Effect	0.8560683	Under

Now we can regularize our model. We use cross validation with a vector of tuning parameters from 0 to 5, increasing on 0.25. The result is an optimal λ= 0.5.

```
# Regularized effects
lambdas <- seq(0, 5, 0.25)

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

  b_Movie <- edx %>%
    group_by(movieId) %>%
    summarise(b_Movie = sum(rating - mu)/(n() + l))

  b_User <- edx %>%
    left_join(b_Movie, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_User = sum(rating - mu - b_Movie)/(n() + l))

  b_RYear <- edx %>%
    left_join(b_Movie, by = "movieId") %>%
    left_join(b_User, by = "userId") %>%
    group_by(year_movie) %>%
    summarise(b_RYear = sum(rating - mu - b_Movie - b_User)/(n() + l))

  b_rateTS <- edx %>%
    left_join(b_Movie, by = "movieId") %>%
```

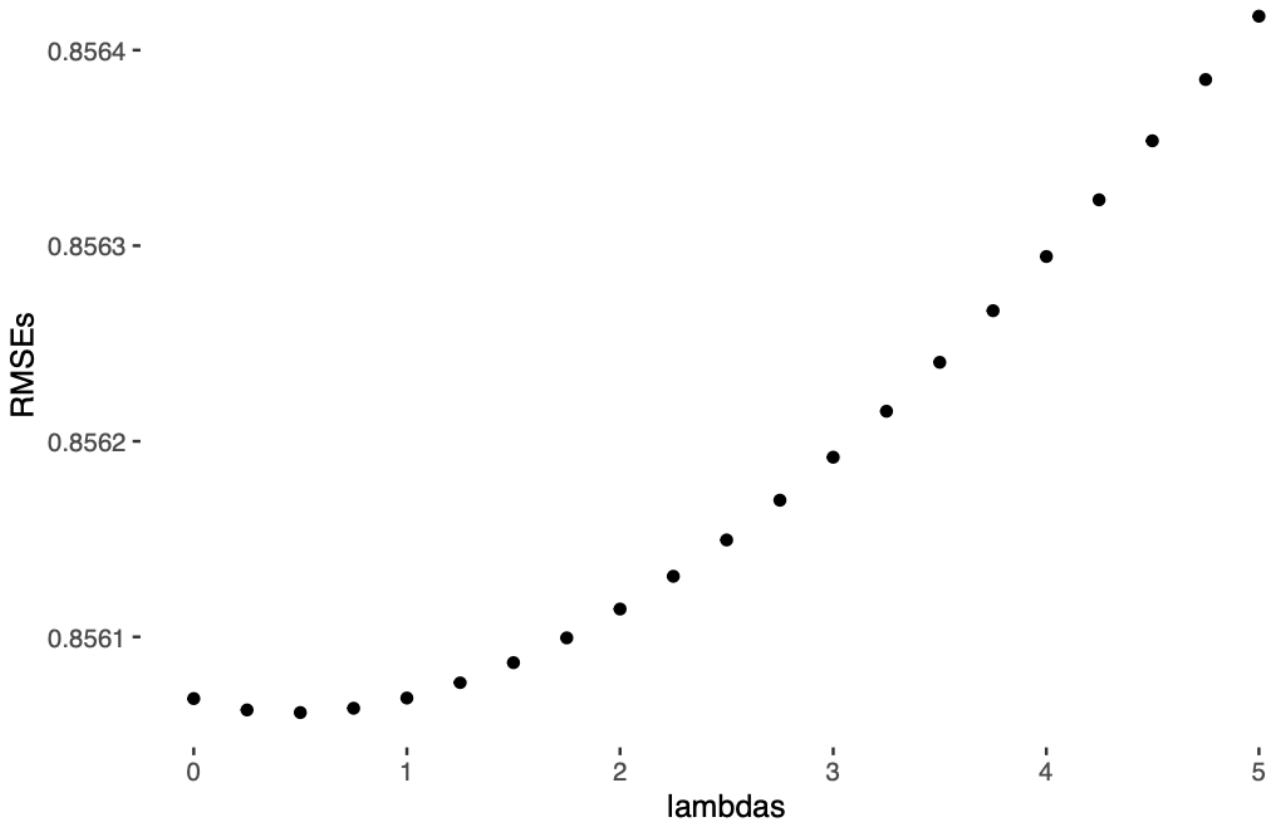
```
    left_join(b_User, by = "userId") %>%
    left_join(b_RYear, by = "year_movie") %>%
    group_by(rate_ts) %>%
    summarise(b_rateTS = sum(rating - mu - b_Movie - b_User - b_RYear)/(n() + l))

  predicts_ratings <- edx %>%
    left_join(b_Movie, by = "movieId") %>%
    left_join(b_User, by = "userId") %>%
    left_join(b_RYear, by = "year_movie") %>%
    left_join(b_rateTS, by = "rate_ts") %>%
    mutate(pred = mu + b_Movie + b_User + b_RYear + b_rateTS) %>%
    pull(pred)

  return(RMSE(predicts_ratings, edx$rating))
})

tibble(lambdas) %>%
  bind_cols(tibble(RMSEs)) %>%
  ggplot(aes(lambdas, RMSEs)) +
  geom_point() +
  theme_update() +
  labs(x = "lambdas", y = "RMSEs") +
  ggtitle("Figure 3.5: RMSE per lambda") +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_rect(fill = "white"))
```

Figure 3.5: RMSE per lambda




```
rmse_model5 <- min(RMSEs)
lambda <- lambdas[which.min(RMSEs)]
```

[1] 0.5

Implementing λ= 0.5 generates a RMSE = 0.8560611. We are now 0.0088389 under our target.

```
rmse_results <- rmse_results %>%
  bind_rows(tibble(Model = "06. Regularized",
                   RMSE = rmse_model5,
                   Goal = ifelse(rmse_model5 < 0.8649, "Under", "Over")))
```

Model	RMSE	Goal
01. Average only	1.0603313	Over
02. Movie Effect	0.9423475	Over
03. +User Effect	0.8567039	Under
04. +RYear Effect	0.8563777	Under
05. +RateTS Effect	0.8560683	Under
06. Regularized	0.8560611	Under

3 Results

A recommendation system “is a subclass of information filtering system that seeks to predict the
‘rating’ or ‘preference’ a user would give to an item” (Shi, 2020). In order to make this prediction,
we have built an ML regularized model that considers four different effects: Movie, User, Release
Year & Rate Timestamp.
Until this moment, we have only used our edx set, obtaining an RMSE of 0.8560611. In this section,
we implement the model on the Validation Set.
We start replicating the effects while incorporating the regularized λ.

```
# Replication with optimal lambda
b_Movie <- edx %>%
  group_by(movieId) %>%
  summarise(b_Movie = sum(rating - mu)/(n() + lambda))

b_User <- edx %>%
  left_join(b_Movie, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_User = sum(rating - mu - b_Movie)/(n() + lambda))

b_RYear <- edx %>%
  left_join(b_Movie, by = "movieId") %>%
  left_join(b_User, by = "userId") %>%
  group_by(year_movie) %>%
  summarise(b_RYear = sum(rating - mu - b_Movie - b_User)/(n() + lambda))

b_RateTS <- edx %>%
  left_join(b_Movie, by = "movieId") %>%
  left_join(b_User, by = "userId") %>%
  left_join(b_RYear, by = "year_movie") %>%
  group_by(rate_ts) %>%
  summarise(b_rateTS = sum(rating - mu - b_Movie - b_User - b_RYear)/(n() + lambda))
```

Then we calculate the time variables in the validation set.

```r
# Validation data arrangement
validation <- validation %>%
  mutate(timestamp = as.POSIXct(timestamp,
                                origin = "1970-01-01",
                                tz = "GMT"),
         year_movie = as.numeric(substr(title,
                                        nchar(title)-4,
                                        nchar(title)-1)),
         year_rated = as.numeric(format(timestamp, "%Y")),
         rate_ts = year_rated - year_movie)
```

Finally we apply the model to the validation set.

```r
# Final prediction
predicts_validation <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(RYear_avgs, by = "year_movie") %>%
  left_join(rateTS_avgs, by = "rate_ts") %>%
  mutate(pred = mu + b_rateTS + b_RYear + b_User + b_Movie) %>%
  pull(pred)

rmse_model_final <- RMSE(predicts_validation,
                         validation$rating)
```

We get an RMSE = 0.8647026. Because our goal was to generate a model with RMSE < 0.8649, we
reach an RMSE 0.0001974422 under our target.

```r
rmse_results <- rmse_results %>%
  bind_rows(tibble(Model = "07. Validation set",
                   RMSE = rmse_model_final,
                   Goal = ifelse(rmse_model_final < 0.8649, "Under", "Over")))
```

Model	RMSE	Goal
01. Average only	1.0603313	Over
02. Movie Effect	0.9423475	Over
03. +User Effect	0.8567039	Under
04. +RYear Effect	0.8563777	Under
05. +RateTS Effect	0.8560683	Under
06. Regularized	0.8560611	Under
07. Validation set	0.8647026	Under

4 Conclusion

Across this project, we first described the MovieLens 10M dataset while presenting the four variables (two existing, two calculated) we focused on. Then, we conducted exploratory data analysis, gaining insights for the model. Finally, we developed a recommendation system that considers the effects of the movie, the user, the release year, and the rate timestamp. We regularized the effects with a tuning parameter of 0.5 and achieved an RMSE = 0.8647026 when applied to the Validation Set.

Even though the model met its goal with a 1.9744224×10^{-4} margin, there are several opportunity areas for improvement in future work:

1. Many top places are occupied by movies with fewer than five ratings, leading the model to consider less-ranked movie characteristics.
2. The `genres` variable was not used in this model. Breaking this column into distinct variables could provide additional insights for better predictions.
3. Another option is to focus on data enrichment by developing APIs to collect more detailed characteristics about the movies.
4. GroupLens is continually improving its software, so downloading larger and more recent datasets could provide more data for accurate predictions.
5. Advanced ML techniques, as highlighted by Meng (2020) and other bloggers, could further refine the analysis.

References

- Harper, F.M. & Konstan, J.A. (2015). *"The MovieLens Datasets: History and Context"*. ACM Transactions on Interactive Intelligent Systems: University of Minnesota. Available at: https://dl.acm.org/doi/10.1145/2827872
- Irizarry, R. A. (2020). *"Introduction to Data Science: Data Analysis and Prediction Algorithms with R"*. Available at: https://rafalab.github.io/dsbook/
- Meng, M. (2020). *"Movie Recommendation System based on MovieLens: Leveraged Natural Language Processing (NLP) and Computer Vision (CV)"*. Towards Data Science. Available at: https://towardsdatascience.com/movie-recommendation-system-based-on-movielens-ef0df580cd0e
- Shi, W. (2020). *"Recommendation Systems: A Review"*. Towards Data Science. Available at: https://towardsdatascience.com/recommendation-systems-a-review-d4592b6caf4b

sessionInfo()

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19041)
##
## Matrix products: default
##
## Random number generation:
## RNG: Mersenne-Twister
## Normal: Inversion
## Sample: Rounding
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats graphics grDevices utils datasets methods base
##
## other attached packages:
## [1] knitr_1.30 data.table_1.13.2 caret_6.0-86 lattice_0.20-41
## [5] forcats_0.5.0 stringr_1.4.0 dplyr_1.0.2 purrr_0.3.4
## [9] readr_1.4.0 tidyr_1.1.2 tibble_3.0.4 ggplot2_3.3.2
## [13] tidyverse_1.3.0
##
## loaded via a namespace (and not attached):
## [1] httr_1.4.2 jsonlite_1.7.1 splines_4.0.3
## [4] foreach_1.5.1 prodlim_2019.11.13 modelr_0.1.8
## [7] assertthat_0.2.1 highr_0.8 stats4_4.0.3
## [10] cellranger_1.1.0 yaml_2.2.1 ipred_0.9-9
## [13] pillar_1.4.7 backports_1.2.0 glue_1.4.2
## [16] pROC_1.16.2 digest_0.6.27 rvest_0.3.6
## [19] colorspace_2.0-0 recipes_0.1.15 htmltools_0.5.0
## [22] Matrix_1.2-18 plyr_1.8.6 timeDate_3043.102
## [25] pkgconfig_2.0.3 broom_0.7.2 haven_2.3.1
```

[28] scales_1.1.1 gower_0.2.2 lava_1.6.8.1
[31] farver_2.0.3 generics_0.1.0 ellipsis_0.3.1
[34] withr_2.3.0 nnet_7.3-14 cli_2.2.0
[37] survival_3.2-7 magrittr_2.0.1 crayon_1.3.4
[40] readxl_1.3.1 evaluate_0.14 fs_1.5.0
[43] fansi_0.4.1 nlme_3.1-150 MASS_7.3-53
[46] xml2_1.3.2 class_7.3-17 tools_4.0.3
[49] hms_0.5.3 lifecycle_0.2.0 munsell_0.5.0
[52] reprex_0.3.0 compiler_4.0.3 rlang_0.4.9
[55] grid_4.0.3 iterators_1.0.13 rstudioapi_0.13
[58] labeling_0.4.2 rmarkdown_2.5 gtable_0.3.0
[61] ModelMetrics_1.2.2.2 codetools_0.2-18 DBI_1.1.0
[64] reshape2_1.4.4 R6_2.5.0 lubridate_1.7.9.2
[67] utf8_1.1.4 stringi_1.5.3 Rcpp_1.0.5
[70] vctrs_0.3.5 rpart_4.1-15 dbplyr_2.0.0
[73] tidyselect_1.1.0 xfun_0.19