Movie Recommendation System Report - MovieLens Project from Data Science: Capstone at HarvardX

Wei Guan

December 28th, 2022

1. Introduction

The aim of this project is to create a recommendation system predict movie ratings using a MovieLens data set. This report describes one possible solution to build such a recommendation system and it is divided into the following parts:

- Section Introduction describes the used data set of MovieLens, the goal of the project and the key steps that were performed.
- Section Methods & Analysis explains the process and techniques used, including data cleaning, data exploration and visualization, followed by the gained insights and the modeling approach.
- Section Results presents and discusses the final results of the recommendation model.
- Section Conclusion gives a brief summary of the outcomes and discusses its limitations and potential future work. Note: All the source code including plots and comments can be found in the corresponding R script.

1.1 Data Set

The website MovieLens provides a movie recommendation service. GroupLens Research has collected and made available rating data sets from the MovieLens web site (https://movielens.org). The data sets were collected over various periods of time, depending on the size of the set (https://grouplens.org/datasets/movielens/). In this project, the 10M version data set of MovieLens was used to build the recommendation system in order to reduce the necessary computing power. The 10M data set from MovieLens is a smaller version of the full-size data set and contains 10,000,054 ratings of the online movie recommending service MovieLens (details will be provided in Section 2.2). The following features/columns can be found in this data set:

- userId: a unique identifier for each user who made the rating with the data type *integer*.
- movieId: a unique identifier for each movie with the data type *integer*.
- rating: the score of the rating of one movie by one user on a five-star scale with the data type *numeric*.
- timestamp: the timestamp of a rating of one movie made by one user, represented in seconds with the data type *integer*.
- title: the title of the rated movie with the release year in parentheses with the data type *character*.
- genres: a sequence of genres (separated by "|") to which the rated movie belongs with the data type *character*.

1.2 Goal

The goal of this project is to build a movie recommendation system based on the provided *10M data set from MovieLens*, with which ratings of movies can be predicted for a certain user at a certain time based on the data insight in the selected data set. In order to achieve this goal, this recommendation system model was created to captures different effects or biases and identify the features that could have an impact on a user rating. Besides the rating prediction, this information could also be used to provide movie recommendations to users.

1.3 Key Steps

In this project the following key steps were performed in order to build the movie recommendation system:

- Data Initiation: The 10m data set from MovieLens" was downloaded. The rating data and the movie data was extracted, transformed and loaded into R. Then the downloaded data set was split into two data sets: edx (90%) for training and testing purpose and final_holdout_test (10%) for the final evaluation of the recommendation system model.
- Data Cleaning, Exploration and Visualization: The Data in the *edx* data set was explored and several data visualizations were made in order to get the insights from the data. Afterwards, the data was checked for NAs, filtered and enriched with additional features for the following model design. The *edx* data set was split into a training set (80%) and a test set (20%) to develop, train and test the recommendation system model.
- Model Design: According to the data exploration, the recommendation system model was built by using the training set and then it was tested with the test set. In the design phase certain effects or biases in the training data were considered. In addition, regularization was also applied to take overfitting into consideration. The Root Mean Squared Error (RMSE) was used to evaluate the model performance and to select the optimal and final model parameters.
- Model Evaluation: The recommendation system model was evaluated using the test data set and the optimized parameters finally. Predictions were calculated for the test data set and were used for the final evaluation of the recommendation system model performance.
- Results: With the optimized parameters the final results of the recommendation system model were presented and discussed by using the data set final_holdout_test.
- Conclusion: Last but not least, there was a brief summary of the report, limitations as well as potential future work.

2. Methods & Analysis

In this section the approach will be explained for initiating and analyzing the data set and also the process of designing, building and evaluation of the recommendation system model. The first step is to load the necessary libraries and to download the 10M data set from MovieLens. Then, the data will be split into two data sets: *edx* and *final_holdout_test*. With the data set *edx* the data will be explored and several data visualizations will be performed in order to get the insight more easily. After feature engineering the data set *edx* is renamed to *edxNew*, which will be split into a training set and a test set for later use. In the Model Design section the approach used to build the recommendation system model will be described. At the end the evaluation and optimization process will be presented and the performance of the recommendation system model will be discussed.

2.1 Data Initiation

As mentioned above, the 10M data set from MovieLens will be used for this project. The following code was provided by the edX course "Data Science: Capstone". In this code, the 10M data set is downloaded from MovieLens. The relevant data is extracted, transformed and loaded into R. The data is split into two data sets: *edx* (90%) and *final_holdout_test* (10%). The *edx* data set is used to analyze the data, train and optimize the recommendation system model. It is important to mention that the data set *final_holdout_test* is only be used at the end of the project with the final model, **not used** for the development and training of the model.

```
# Create edx and final_holdout_test sets

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Warning: Paket 'tidyverse' wurde unter R Version 4.1.3 erstellt
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Warning: Paket 'caret' wurde unter R Version 4.1.3 erstellt
```

```
library(tidyverse)
library(caret)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

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

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>% mutate(userId = as.integer(userId),
                              movieId = as.integer(movieId),
                              rating = as.numeric(rating),
                              timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
```

```
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>% mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

2.2 Data Cleaning, Exploration and Visualization

At first, download some more libraries that will be used later.

```
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")

library(lubridate)
library(ggplot2)
```

Now let's do some research and have a close view of the data.

There are 9.000.055 rows and 6 columns in the data set *edx*.

```
dim(edx)
```

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

There are 999.999 of rows and 6 columns in the data set *final_holdout_test*.

```
dim(final_holdout_test)
```

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

These are the first 6 rows of *edx* data set.

```
head(edx, 6)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2           Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5         Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7         Children|Comedy|Fantasy
```

There are 69878 different users in the *edx* data set.

```
n_distinct(edx$userId)
```

```
## [1] 69878
```

There are 10677 different movies in the *edx* data set.

```
n_distinct(edx$movieId)
```

```
## [1] 10677
```

There are 797 different genres in the *edx* data set. Please note that, movies can be in multiple genres, separated by "|".

```
n_distinct(edx$genres)
```

```
## [1] 797
```

In the *edx* data set, users have given 10 different values of ratings, ranging from 0.5 to 5.0.

```
unique(edx$rating) %>% sort()
```

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

These are the five most given ratings in order from most to least in the *edx* data set.

```
edx %>% group_by(rating) %>%
  summarize(count = n()) %>%
  top_n(5) %>% arrange(desc(count))
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

These are the first 6 movies with highest number of user ratings in the *edx* data set.

```
edx %>% group_by(movieId, title) %>%
  summarize(count = n()) %>%
  arrange(desc(count)) %>%
  head()
```

```
## # A tibble: 6 x 3
## # Groups:   movieId [6]
##   movieId title                                count
##     <int> <chr>                                <int>
## 1     296 Pulp Fiction (1994)                  31362
## 2     356 Forrest Gump (1994)                  31079
## 3     593 Silence of the Lambs, The (1991)     30382
## 4     480 Jurassic Park (1993)                 29360
## 5     318 Shawshank Redemption, The (1994)     28015
## 6     110 Braveheart (1995)                    26212
```

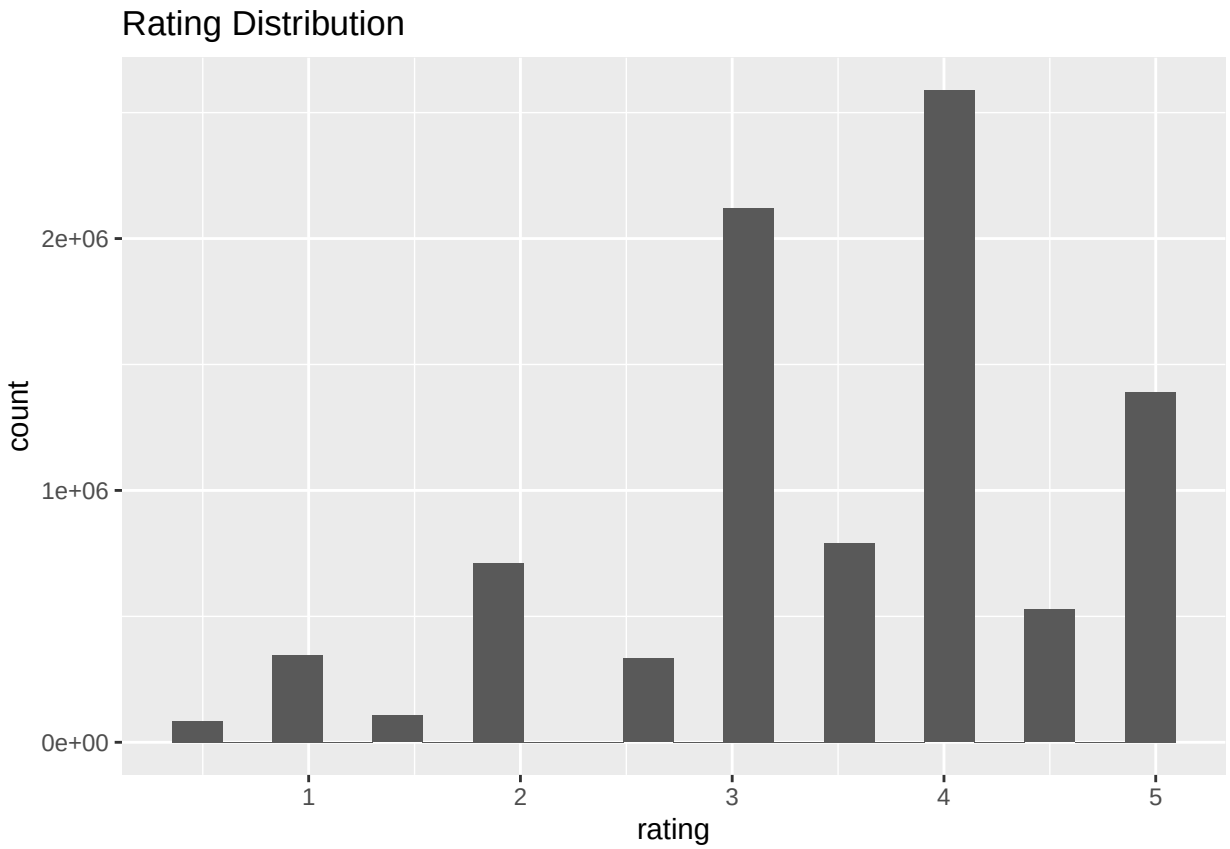
These are the first 6 movies with lowest number of user ratings in the *edx* data set. Those movies just have one rating.

```
edx %>% group_by(movieId, title) %>%
  summarize(n = n(), genres = genres) %>%
  arrange(n) %>%
  head()
```

```
## # A tibble: 6 x 4
## # Groups:   movieId, title [6]
##   movieId title                                  n genres
##     <int> <chr>                              <int> <chr>
## 1    3191 Quarry, The (1998)                     1 Drama
## 2    3226 Hellhounds on My Trail (1999)          1 Documentary
## 3    3234 Train Ride to Hollywood (1978)         1 Comedy
## 4    3356 Condo Painting (2000)                  1 Documentary
## 5    3383 Big Fella (1937)                       1 Drama|Musical
## 6    3561 Stacy's Knights (1982)                 1 Drama
```

Here is a plot the distribution of ratings without taking any features into consideration.

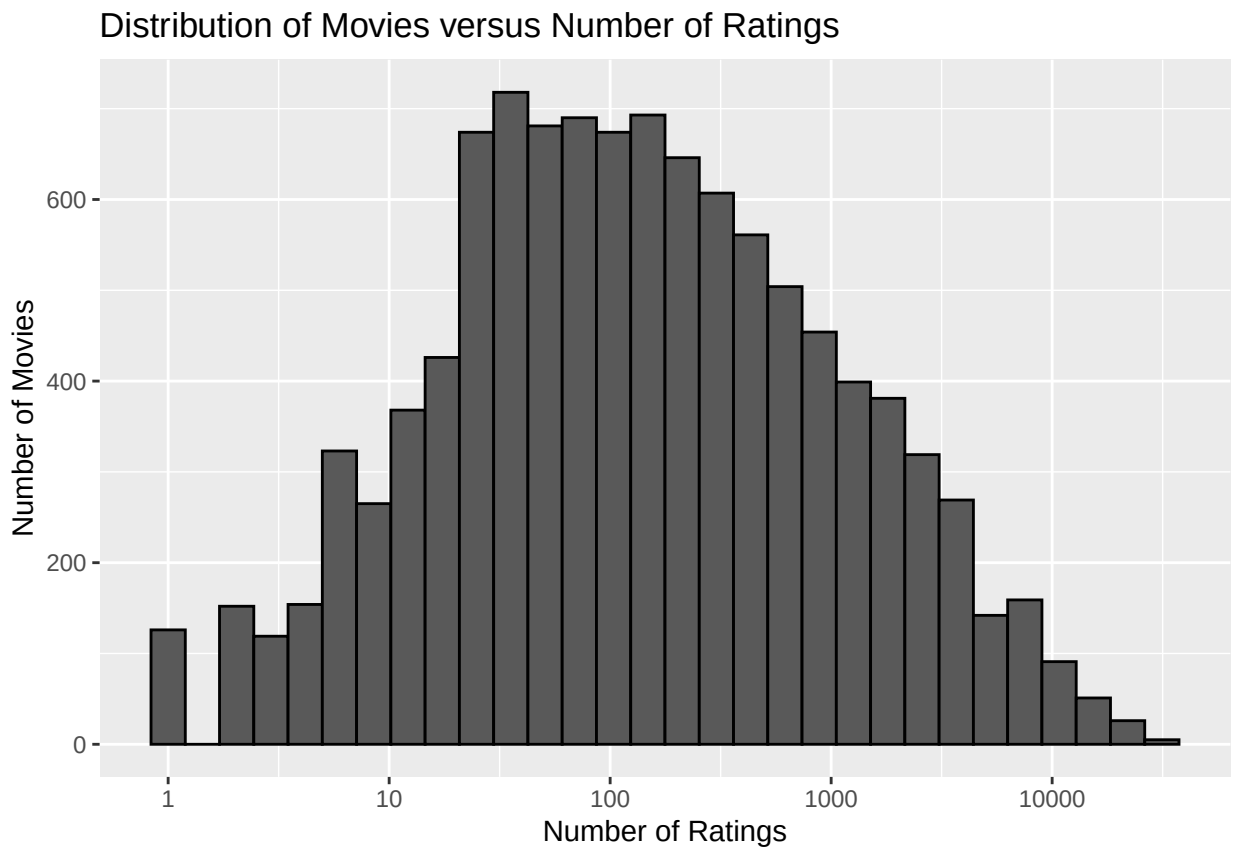
```
edx %>% ggplot(aes(rating)) +
  geom_histogram(bins = 20) +
  ggtitle("Rating Distribution")
```



From this figure we can see that half star ratings are less common than whole star ratings. There are a lot of movies that have a rating 3.0 or 4.0.

Here is a plot that shows distribution of movies versus their number of ratings.

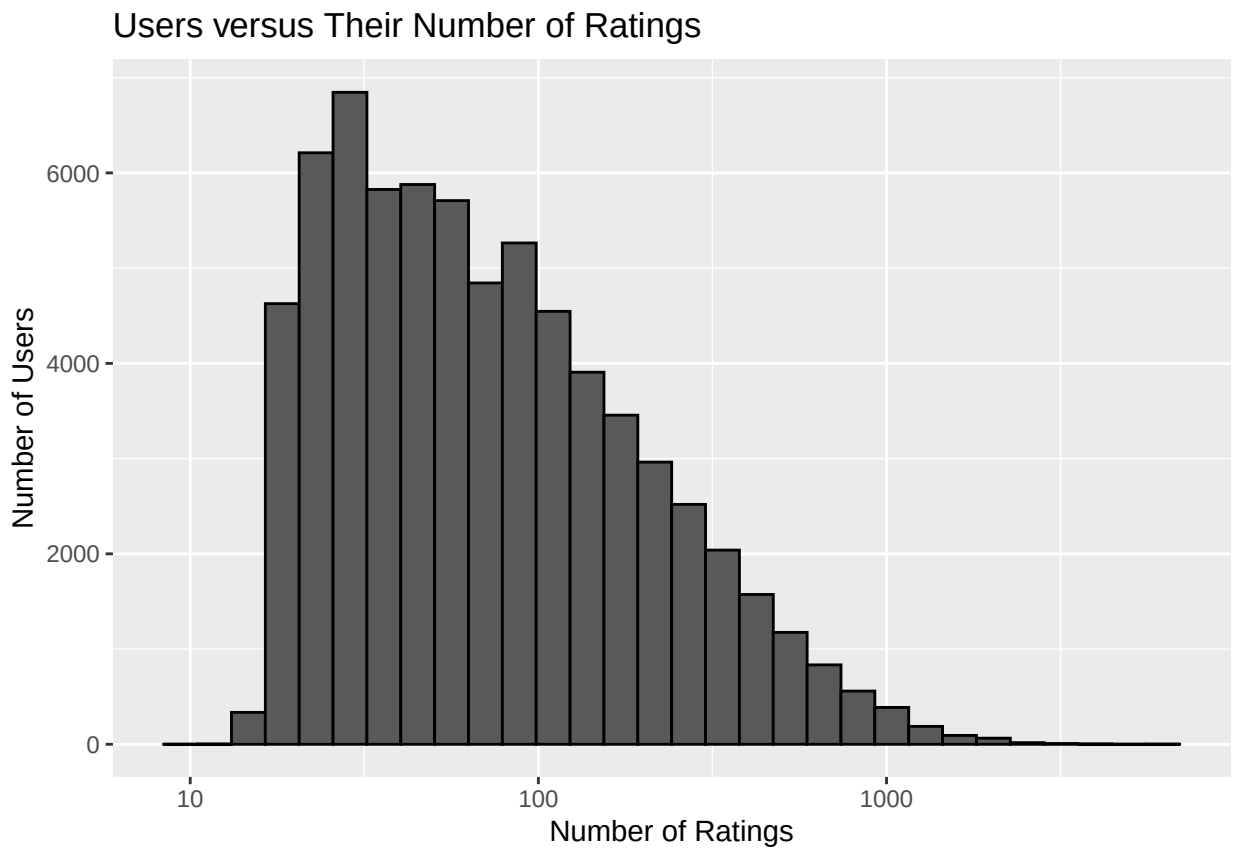
```
edx %>%
  group_by(movieId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Number of Movies") +
  ggtitle("Distribution of Movies versus Number of Ratings")
```



In this figure we can see that there are many movies with very few ratings. This indicates there is an effect of the movie feature over the rating results.

Here is another plot showing the distribution of users versus their number of ratings.

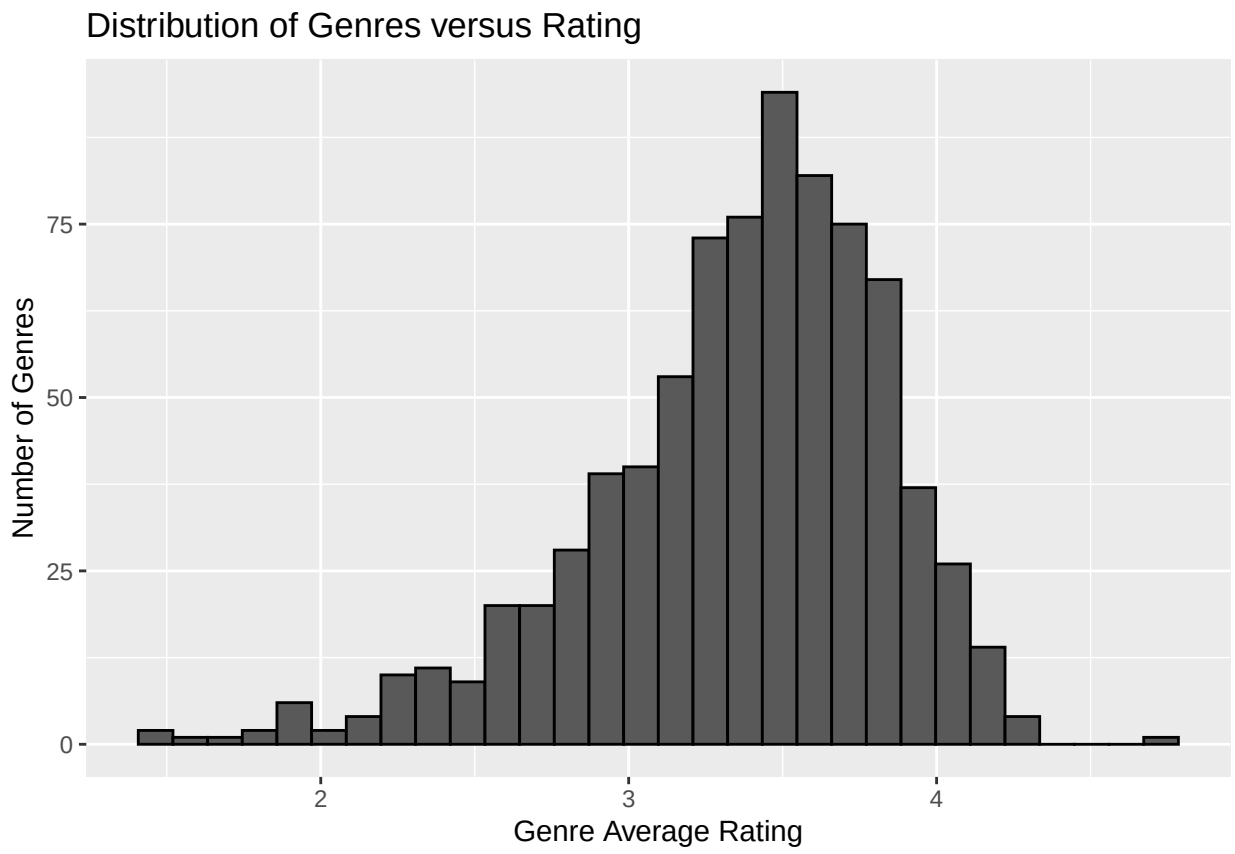
```
edx %>%
  group_by(userId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Number of Users") +
  ggtitle("Users versus Their Number of Ratings")
```



In this figure we can see that there are users who are quite active and there are also some users who are not very active with fewer ratings. This means there might be also a user effect on the rating results.

Here is a plot showing the distribution of genres versus their average ratings.

```
edx %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating)) %>%
  ggplot(aes(b_g)) +
  geom_histogram(bins = 30, color = "black") +
  xlab("Genre Average Rating") +
  ylab("Number of Genres") +
  ggtitle("Distribution of Genres versus Rating")
```



In this figure we can see that a lot of genres fall within the average rating interval between 3.0 and 4.0. This might indicate that there might be a genre effect on the rating results.

Let's check if there are any NAs in the *edx* data set.

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

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

The result shows that there are not any NAs.

It seems that the column *timestamp* could also be an effect. So let's create a new feature rateDate from timestamp for the data set *edx*. Note that there are 7 columns now and the data set *edx* is renamed to *edxNew*.

```
edxNew <- edx %>% mutate(rateDate = round_date(as_datetime(timestamp), unit = "day"))
```

```
head(edxNew, 6)
```

```
##   userId movieId rating timestamp                          title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
```

```
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres   rateDate
## 1                Comedy|Romance 1996-08-02
## 2          Action|Crime|Thriller 1996-08-02
## 4  Action|Drama|Sci-Fi|Thriller 1996-08-02
## 5       Action|Adventure|Sci-Fi 1996-08-02
## 6 Action|Adventure|Drama|Sci-Fi 1996-08-02
## 7       Children|Comedy|Fantasy 1996-08-02
```

```
dim(edxNew)
```

```
## [1] 9000055       7
```

Let's also create the new feature rateDate from timestamp for the data set *final_holdout_test*. Note that there are 7 columns now, too.

```
final_holdout_test <- final_holdout_test %>%
  mutate(rateDate = round_date(as_datetime(timestamp), unit = "day"))
```

```
head(final_holdout_test, 6)
```

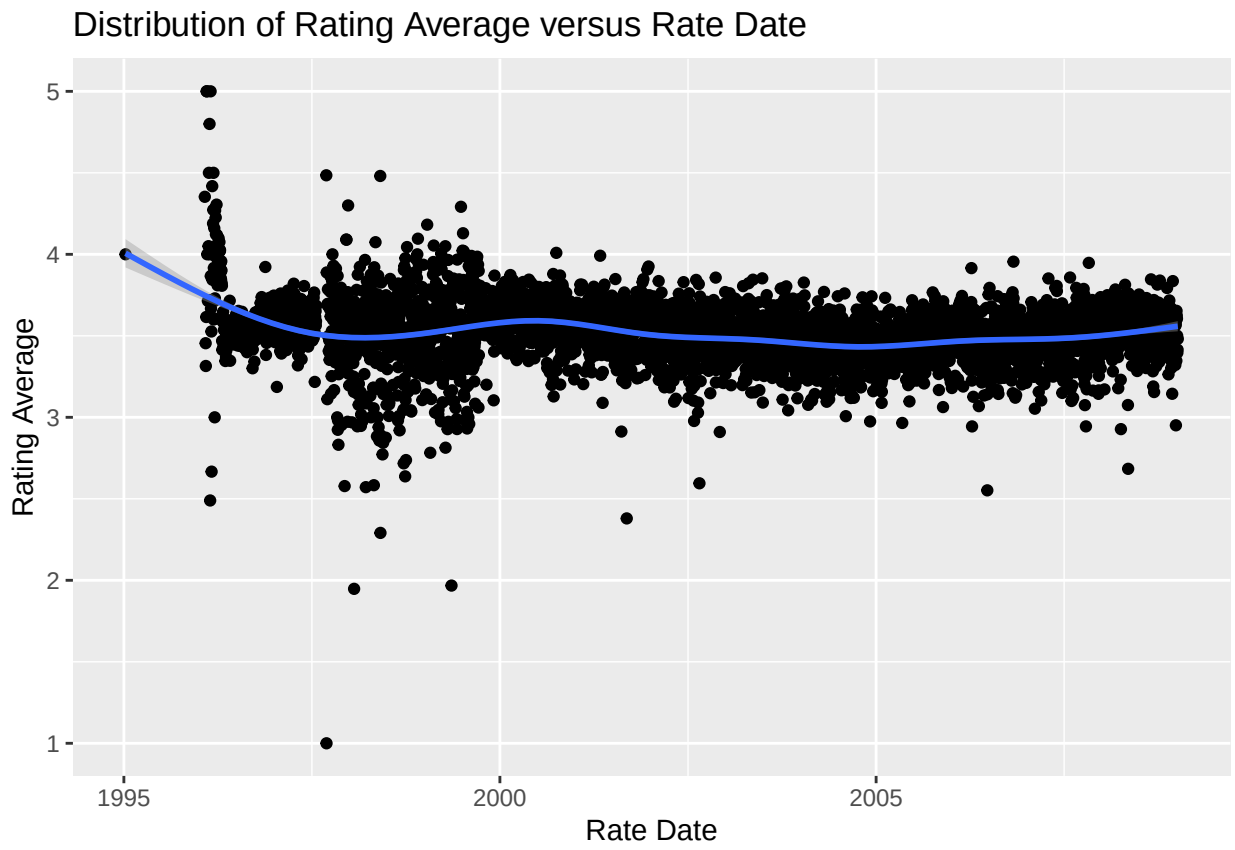
```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                  title
## 1                                  Dumb & Dumber (1994)
## 2                                  Jurassic Park (1993)
## 3                                     Home Alone (1990)
## 4                                        Rob Roy (1995)
## 5                                   Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                  genres   rateDate
## 1                                Comedy 1996-08-02
## 2        Action|Adventure|Sci-Fi|Thriller 1996-08-02
## 3                      Children|Comedy 1996-08-02
## 4               Action|Drama|Romance|War 1997-07-07
## 5                           Crime|Drama 1997-07-07
## 6 Action|Adventure|Horror|Sci-Fi|Thriller 1997-07-07
```

```
dim(final_holdout_test)
```

```
## [1] 999999      7
```

Here is a plot showing the distribution of rating average versus rate date

```
edxNew %>%
  group_by(rateDate) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(rateDate, rating)) +
  geom_point() +
  geom_smooth() +
  xlab("Rate Date") +
  ylab("Rating Average") +
  ggtitle("Distribution of Rating Average versus Rate Date")
```



In this figure we can see that there is a slight trend for the rating going down when the dates are close to today. There might be a rate date effect on the rating results.

Before moving forward, let's reduce the data set by selecting users with at least 15 ratings, because it is mentioned above it seems that there might be users who are not so active (more details provided in the section *Model Design*). There are still 8.999.671 rows in the data set *edxNew*.

```
edxNew <- edxNew %>% group_by(userId) %>% filter(n() >= 15) %>% ungroup()
```

```
dim(edxNew)
```

```
## [1] 8999671       7
```

Finally, the data set *edxNew* is split into a training set (80%) and a test set (20%) to be used in the model design and parameter optimization.

```r
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
# set.seed(1) # if using R 3.5 or earlier
testIndex <- createDataPartition(y = edxNew$rating, times = 1, p = 0.2, list = FALSE)
trainSet <- edxNew[-testIndex, ]
testSet <- edxNew[testIndex, ]

testSet <- testSet %>%
  semi_join(trainSet, by = "movieId") %>%
  semi_join(trainSet, by = "userId")
```

2.3 Model Design

The recommendation system model to be built is called **Regularized Recommendation System Model with Movie + Active User + Genre + Rate Date Effects**. This model captures different biases or effects in the training data set that could influence the resulting rating. As the name of the model revealed, *movie*, *active user*, *genre* and *rate date* bias will be included. For each effect, **regularization** is used to consider **overfitting**, which occurs when a model performs well only on a specific data set, but rather poorly on an unknown data set.

In order to apply *regularization*, a penalty parameter **lambda** is introduced to each bias. *Lambda* penalizes the effect of a bias, so that the predicted rating is adjusted. It is a little bit tricky to pick out the optimal penalty parameter. Therefore, a range of lambdas will be defined and the model will be trained and evaluated against the test set repeatedly to find the optimal lambda. Due to lack of computing power, I chose the optimal lambda range to be somewhere between 4.0 to 5.0 in 0.1 steps resulting in only 10 iterations. Of course, a much broader lambda range could be chosen, but it would lead to massive computing time and even overfitting. To find out the optimal lambda, the *Root Mean Squared Error* (RMSE) will be used to evaluate the model performance for each iteration. The RMSE is always non-negative and it also penalizes large deviations disproportionately. Therefore, it is sensitive to outliers. Note: the smaller the RMSE is, the better is the performance of the model.

According to the data exploration and features in the section **2.2 Data Cleaning, Exploration and Visualization**, *Movie*, *Active User*, *Genre* and *Rate Date* are chosen as effects to build the recommendation system model. Each effect has its own penalty, depending on the order of the effects. The order of the effects also plays a role in the performance of the model, e.g. the earlier the effect appears, the more impact this effect has in the model. Due to the observation of the data insight, the following order of effects was designed, as it turns out reasonable and best results:

- **1 Movie Effect:** captures the effect that a movie has on the rating, adjusted by the overall average rating.
- **2 Active User Effect:** captures the effect that a user has on the rating, adjusted by the overall average rating and the movie effect. Note: The *edx* data set was filtered in the section *2.2 Data Cleaning, Exploration and Visualization* by users with at least 15 ratings. The assumption is that users who rate more often have more experience. Their ratings are therefore more reliable for the rating prediction.
- **3 Genre Effect:** captures the effect the genre of the movie has on the rating, adjusted by the overall average rating, the movie effect and the active user effect.
- **4 Rate Date Effect:** captures the effect of the date, on which the rating was made, adjusted by the overall average rating, the movie effect, the active user effect and the genre effect.

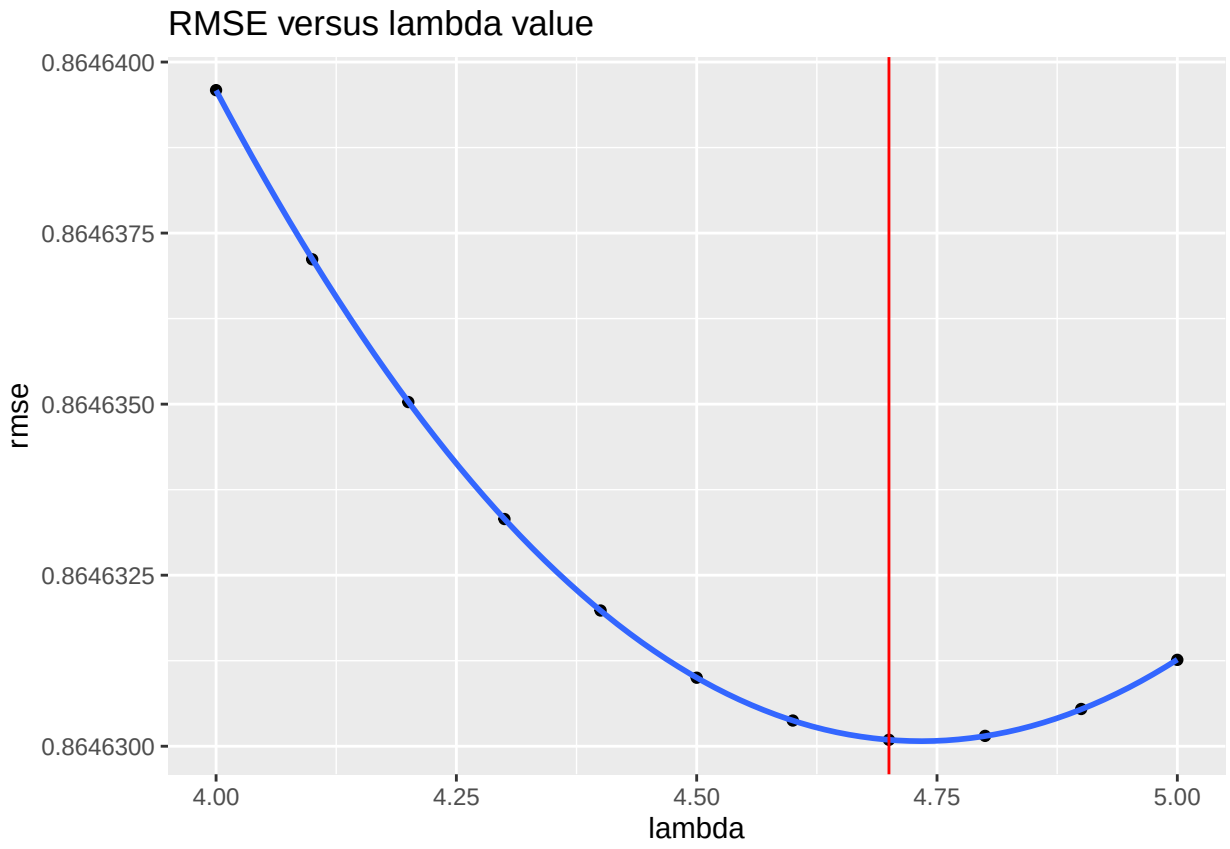
Now, the model was built with the above assumptions and ideas with the following code. The model optimal lambda was searched based on the train and test set. Due to the filter of active users with at least 15 ratings and selected

biases, there are some NAs in the prediction. Theses NAs are replace with either the movie average or with the overall average in case the movie average is not available either. In each iteration, a lambda is evaluated using the RMSE to find the optional penalty parameter.

```
lambdas <- seq(4.0, 5.0, 0.1)

RMSE <- function(true, predicted){
  sqrt(mean((true - predicted)^2, na.rm = TRUE))
}

rmses <- sapply(lambdas, function(l){
  avg <- mean(trainSet$rating)

  movie_avg <- trainSet %>%
    group_by(movieId) %>%
    summarize(movie_avg = mean(rating))

  b_m <- trainSet %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - avg)/(n()+l))

  b_u <- trainSet %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - avg - b_m)/(n()+l))

  b_g <- trainSet %>%
    left_join(b_m, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - avg - b_m - b_u)/(n()+l))

  b_rd <- trainSet %>%
    left_join(b_m, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    group_by(rateDate) %>%
    summarize(b_rd = sum(rating - avg - b_m - b_u - b_g)/(n()+l))

  predicted_ratings <- testSet %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_rd, by = "rateDate") %>%
    mutate(pred = avg + b_m + b_u + b_g + b_rd) %>%
    mutate(pred = ifelse(is.na(pred), movie_avg, pred)) %>%
    mutate(pred = ifelse(is.na(pred), avg, pred)) %>%
    mutate(pred = ifelse(pred < 0, 0.5, ifelse(pred > 5, 5, pred))) %>%
    .$pred

  return(RMSE(predicted_ratings, testSet$rating))
})
```

After running the code above for the optimal penalty lambda value, we got the lambda values and their RMSE values. Here is a figure to show the relation between them.

```
trainResults <- data.frame(lambda = lambdas, rmse = rmses)
print(trainResults %>% ggplot(aes(lambda, rmse)) +
        ggtitle("RMSE versus lambda value") +
        geom_point() +
        geom_smooth() +
        geom_vline(xintercept = lambdas[which.min(rmses)], color = "red"))
```



As mentioned above, the optimal penalty parameter lambda value has the minial RMSE value. Therefore, the optimal lambda is 4.7.

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

print(lambda)
```

```
## [1] 4.7
```

2.4 Model Evaluation

The optimal penalty parameter lambda has been identified to be **4.7** in the section *2.3 Model Design*. In this section the recommendation system model will be run again with the optimal parameter lambda in order to perform the evaluation of the model with the training and test data.

```
training_Rmse <- sapply(lambda, function(l){
  avg <- mean(trainSet$rating)

  movie_avg <- trainSet %>%
    group_by(movieId) %>%
    summarize(movie_avg = mean(rating))

  b_m <- trainSet %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - avg)/(n()+l))

  b_u <- trainSet %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - avg - b_m)/(n()+l))

  b_g <- trainSet %>%
    left_join(b_m, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - avg - b_m - b_u)/(n()+l))

  b_rd <- trainSet %>%
    left_join(b_m, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    group_by(rateDate) %>%
    summarize(b_rd = sum(rating - avg - b_m - b_u - b_g)/(n()+l))

  predicted_ratings <- testSet %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_rd, by = "rateDate") %>%
    mutate(pred = avg + b_m + b_u + b_g + b_rd) %>%
    mutate(pred = ifelse(is.na(pred), movie_avg, pred)) %>%
    mutate(pred = ifelse(is.na(pred), avg, pred)) %>%
    mutate(pred = ifelse(pred < 0, 0.5, ifelse(pred > 5, 5, pred))) %>%
    .$pred

  return(RMSE(predicted_ratings, testSet$rating))
})


print(training_Rmse)
```

```
## [1] 0.8646301
```

After running on the train and test set, the recommendation system model gets a **training RMSE** of **0.8646301** with the optimal penalty parameter **4.7**. Note: This is not the final RMSE, because the final data set was not used here. Now let's do the final evaluation of the recommendation system model in the next section.

3. Results

In this section the final evaluation of the recommendation system model will be performed with the final data set **final_holdout_test**. The results will be presented and discussed. To evaluate the model results, the RMSE with the optimal penalty lambda value will be used as before. Here is the code to perform the final evaluation of the designed recommendation system model.

```
final_Rmse <- sapply(lambda, function(l){
  avg <- mean(trainSet$rating)

  movie_avg <- trainSet %>%
    group_by(movieId) %>%
    summarize(movie_avg = mean(rating))

  b_m <- trainSet %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - avg)/(n()+l))

  b_u <- trainSet %>%
    left_join(b_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - avg - b_m)/(n()+l))

  b_g <- trainSet %>%
    left_join(b_m, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - avg - b_m - b_u)/(n()+l))

  b_rd <- trainSet %>%
    left_join(b_m, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(b_g, by='genres') %>%
    group_by(rateDate) %>%
    summarize(b_rd = sum(rating - avg - b_m - b_u - b_g)/(n()+l))

  predicted_ratings <- final_holdout_test %>%
    left_join(movie_avg, by = "movieId") %>%
    left_join(b_m, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_rd, by = "rateDate") %>%
    mutate(pred = avg + b_m + b_u + b_g + b_rd) %>%
    mutate(pred = ifelse(is.na(pred), movie_avg, pred)) %>%
    mutate(pred = ifelse(is.na(pred), avg, pred)) %>%
    mutate(pred = ifelse(pred < 0, 0.5, ifelse(pred > 5, 5, pred))) %>%
    .$pred

  return(RMSE(predicted_ratings, final_holdout_test$rating))
})

print(final_Rmse)
```

```
## [1] 0.8646791
```

With the final data set **final_holdout_test** the designed recommendation system model presented a **final RMSE**: **0.8646791**. In comparison with the RMSE (*0.8646301*) when using the test data set in the section *2.4 Model Evaluation*, the RMSE when using the final data set got a little bit worse. There might be somehow overfitting. In addition, the penalty parameter lambda was picked out based on the limited test data set and this could also have a negative impact on the performance of the model on the final data set. The difference between the test data set and the final data set is quite small, so that we could say that the designed recommendation system model performs almost as expected.

4. Conclusion

In the context of the MovieLens Project, I analyzed and explored the 10M data set from MovieLens at first to get the insights of the data. Data Visualization is quite a useful method. In the phase of Model Design, the approach of building and evaluating the recommendation system model has been explained and the effects chosen for this model has been defined and elaborated according to the data insights. Based on the data insights and analysis, **Regularized Recommendation System Model with Movie + Active User + Genre + Rate Date Effects** has been designed. In the end, the optimal penalty parameter was chosen to perform on the final data set. The results have been presented and discussed. The designed recommendation system model has got a final RMSE of **0.8646791** on the final data set **final_holdout_test**.

There are also some limitations of this project. Instead of the full data set, the 10M data set from MovieLens was used to build the model. With the full data set, more data insights could be gained during the data exploration. A second limitation is the lack of the computing power of a normal notebook, despite of the subset of the full data set. During my project, R was aborted due to intensive computation, so that I had to restart R from time to time and had to give up other methods such as Boosting, Random Forests, etc.

As potential future work, the full data set of MovieLens could be used to gain more insights of the data. This could be combined by using cloud technology such as Amazon Web Services and Microsoft Azure in order to finish much more intensive computation with other modeling methods and algorithms. Furthermore, more features could be created and taken into consideration for modeling the recommendation system in order to get better predictions, e.g. release year of movies, rate year of ratings, separated movie genres, ratings of users, etc.