MovieLens Project

Low Yao Dong

6/13/2021

1 Introduction

This report documents the creation of a movie recommendation system by based on ratings that users have given. First, data exploration will be performed on the MovieLens data set. The insights is then used to build a model, and fine tuning it until the target root mean squared error (RMSE) is obtained.

2 Methodology

2.1 Data Pre-Processing

The first step is to download and to create test and validation sets.

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################

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

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(tidyverse)
library(caret)
library(data.table)
library(dplyr)
library(knitr)
library(stringr)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(tinytex)

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

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

```
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

set.seed(1, sample.kind="Rounding")
```

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

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

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

Next, the rating year and movie release year were extracted from the timestamp and title columns respectively. The genres and timestamp were removed to reduce data size.

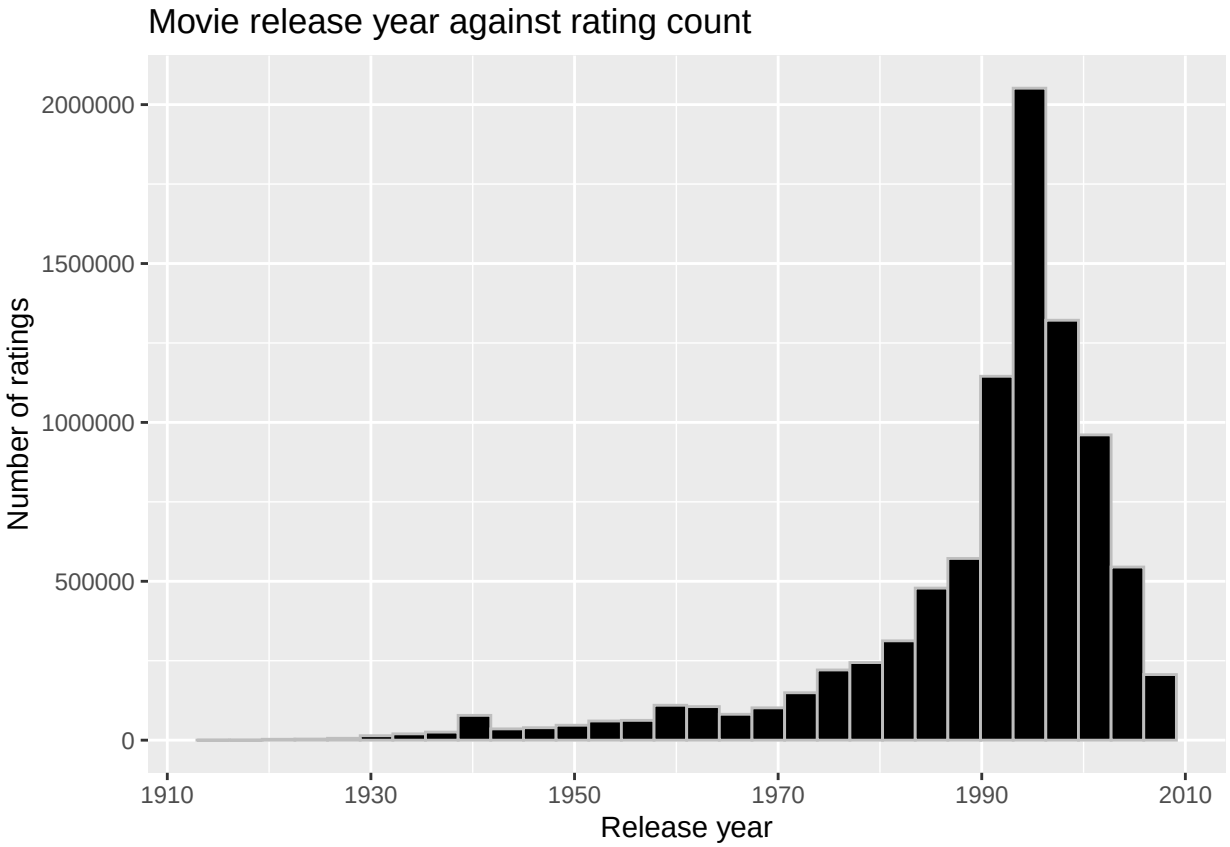
```
# Extract release year from title
edx <- edx %>%
  mutate(release_year = as.numeric(str_sub(title, -5, -2)))
validation <- validation %>%
  mutate(release_year = as.numeric(str_sub(title, -5, -2)))

# Extract rating year from timestamp then delete timestamp
edx <- edx %>% mutate(rating_year = year(as_datetime(timestamp)))
validation <- validation %>% mutate(rating_year = year(as_datetime(timestamp)))
edx <- edx %>% select (-c(timestamp))
validation <- validation %>% select (-c(timestamp))
```

2.2 Data Exploration

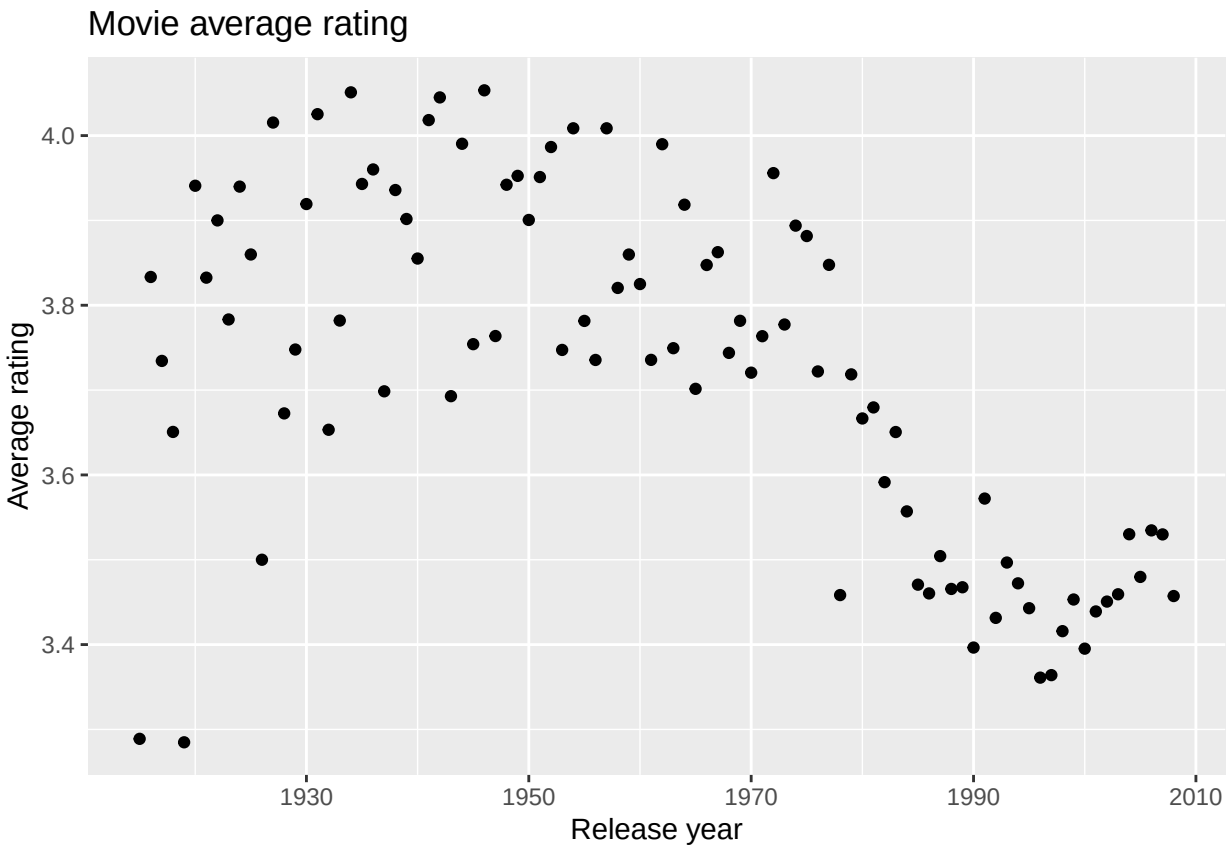
First, it was observed that there were significantly more ratings for movies released between the years 1990 and 2000. This trend seems to suggest that movies released in the earlier years were not watched as much as those released after 1990. One possible reason could be that technological advancements, for example, computer generated imagery (CGI), enabled the production of wider varieties of movies.

```
# Plot movie release year against rating count
edx %>% ggplot(aes(release_year)) +
  geom_histogram(bins=30, fill="black", col="grey") +
  labs(title = "Movie release year against rating count", x = "Release year", y = "Number of ratings")
```



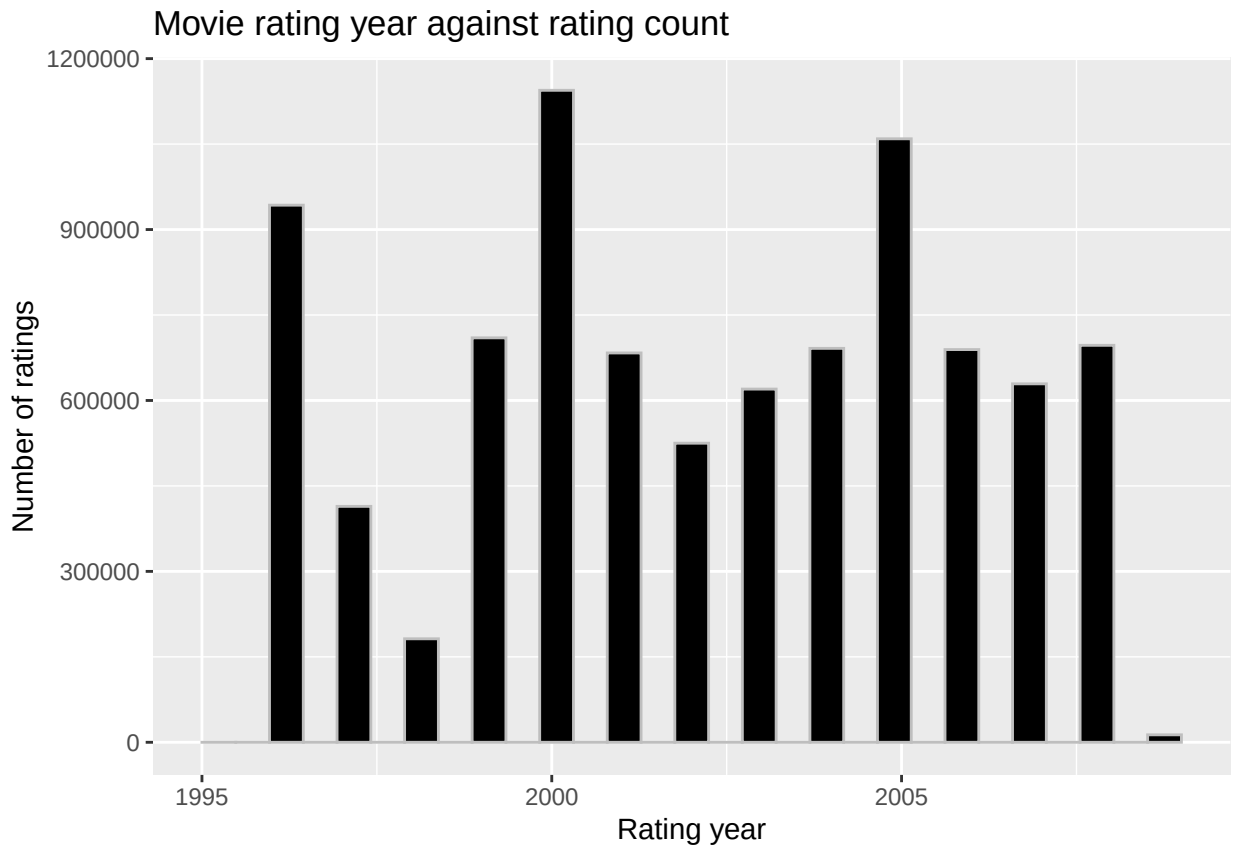
In order to see what effects release year had on the recommendation system, a plot of movie release year vs average ratings by release year was generated. It is evident that ratings of older movies were generally higher but had a greater variation.

```
# Plot movie release year vs average ratings
edx %>% group_by(release_year) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(release_year, avg_rating)) +
  geom_point() +
  labs(title = "Movie average rating", x = "Release year", y = "Average rating")
```

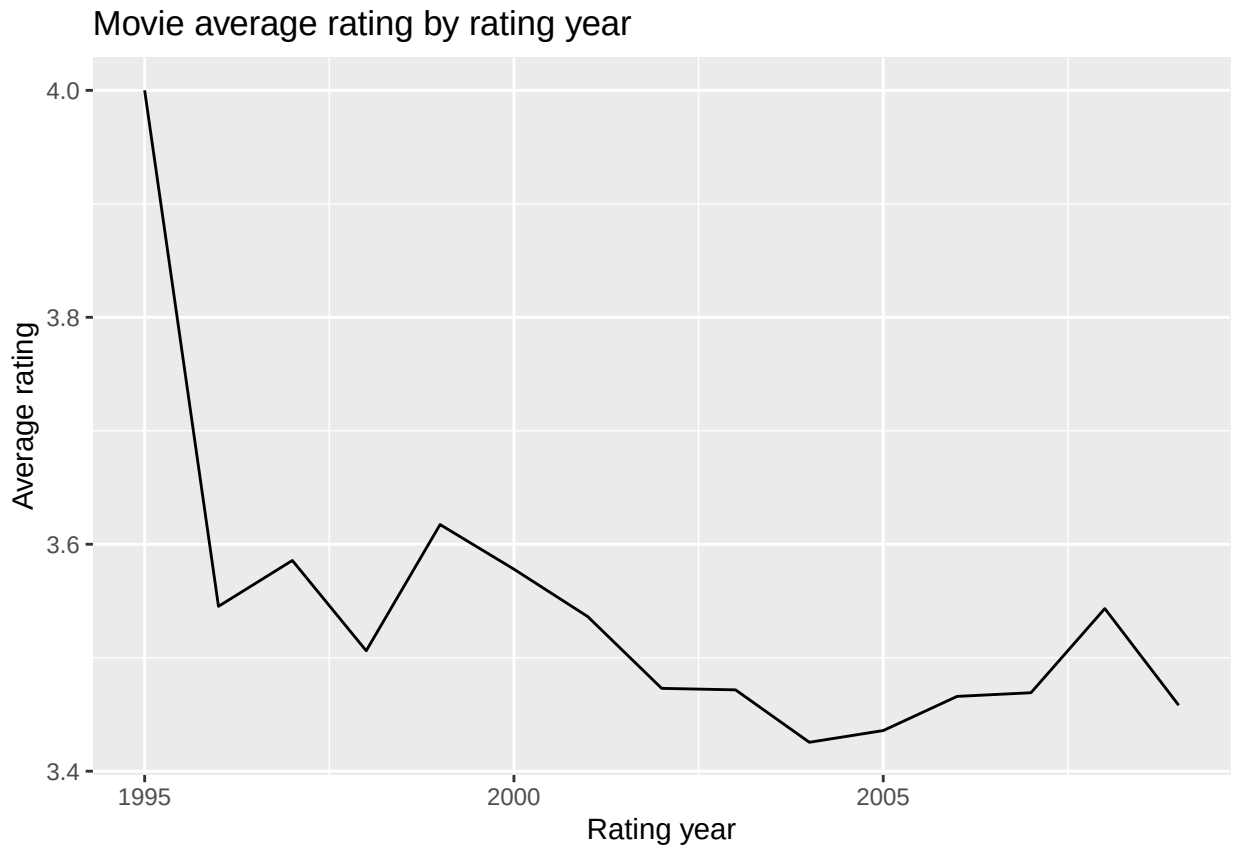


Next, the histogram below indicates that the movies were only rated from 1995 onward. There is no data available on viewer demographics, but there is a strong possibility that viewers tend to be drawn to movies from their era and therefore only rated for those. The average rating was also found to be highest in the first year, which indicates some form of bias when movie ratings were first collected.

```
# Plot movie rating year against rating count
edx %>% ggplot(aes(rating_year)) +
  geom_histogram(bins=30, fill="black", col="grey") +
  labs(title = "Movie rating year against rating count", x = "Rating year", y = "Number of ratings")
```



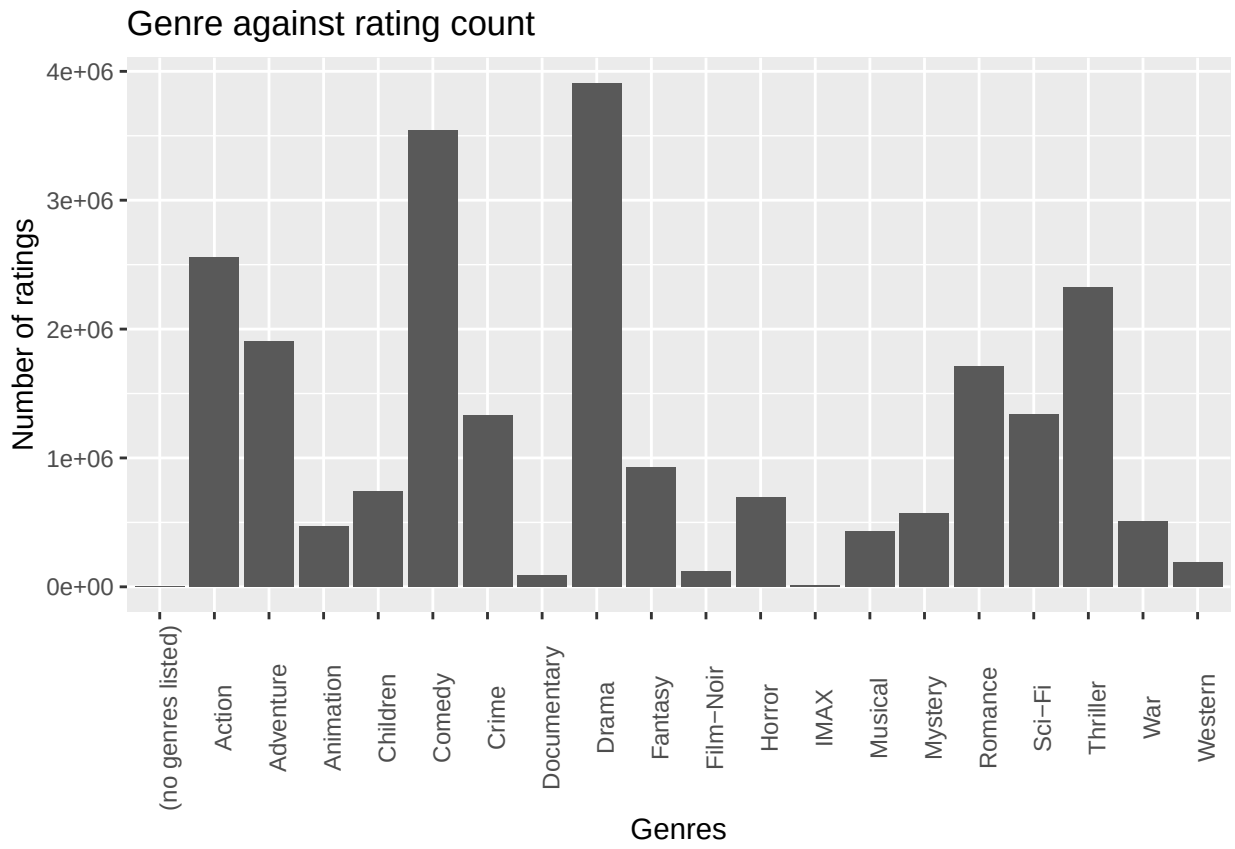

```
# Plot movie rating year vs average ratings
edx %>% group_by(rating_year) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(rating_year, avg_rating)) +
  geom_line() +
  labs(title = "Movie average rating by rating year", x = "Rating year", y = "Average rating")
```



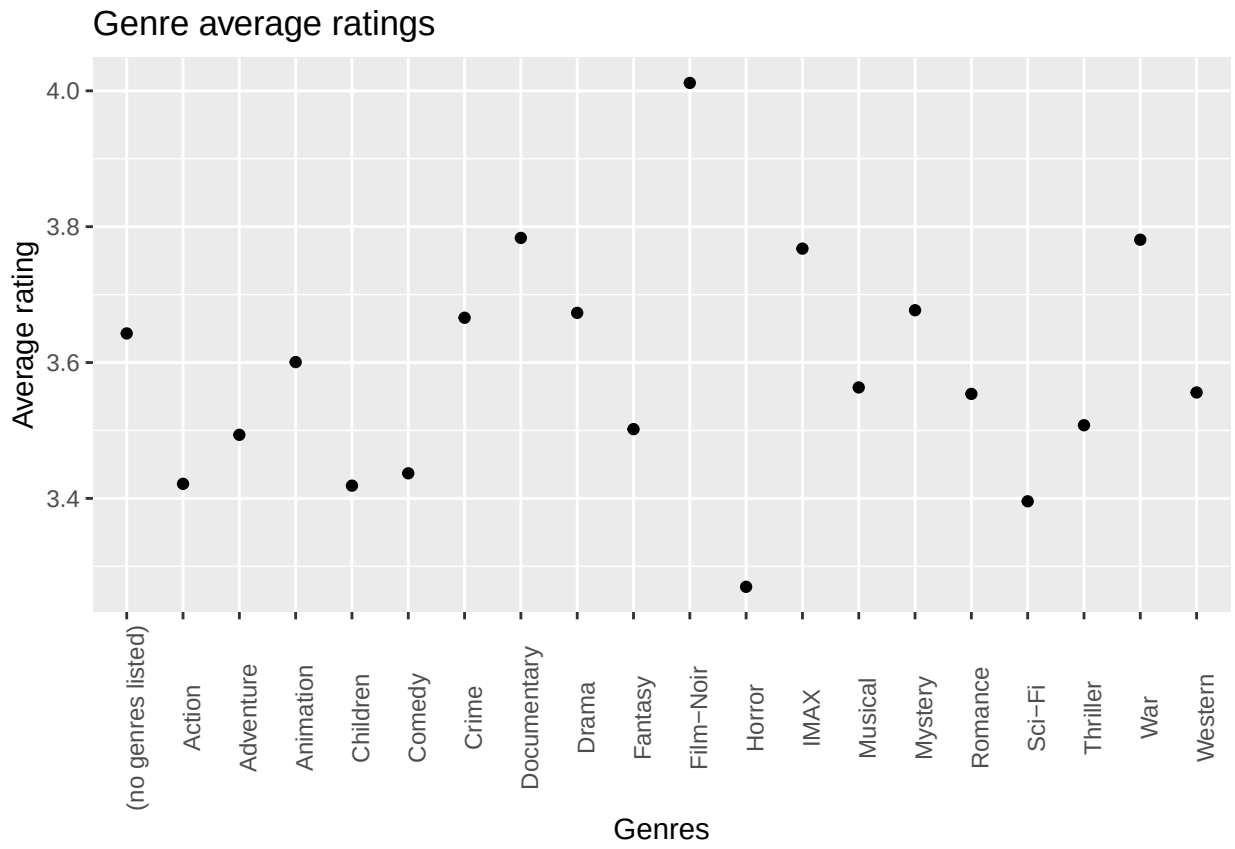
Subsequently, effect of movie genres were analyzed. It was obvious that certain genres, namely drama, comedy and action were more popular than others, and thus received more ratings. Also, certain genres, namely film-noir, had a higher mean rating than others.

```
# Unpivot data to obtain one row for each genre
edx_unpivot_genres <- edx %>%
  mutate(genre=fct_explicit_na(genres, na_level = "(None)")) %>%
  separate_rows(genre,sep = "\\|")
```

```
# Plot genre against rating count
edx_unpivot_genres %>%
  group_by(genre) %>%
  summarize(n=n()) %>%
  ggplot(aes(x=genre, y=n)) +
  geom_bar(stat='identity') +
  theme(axis.text.x = element_text(angle = 90)) +
  labs(title = "Genre against rating count", x = "Genres", y = "Number of ratings")
```

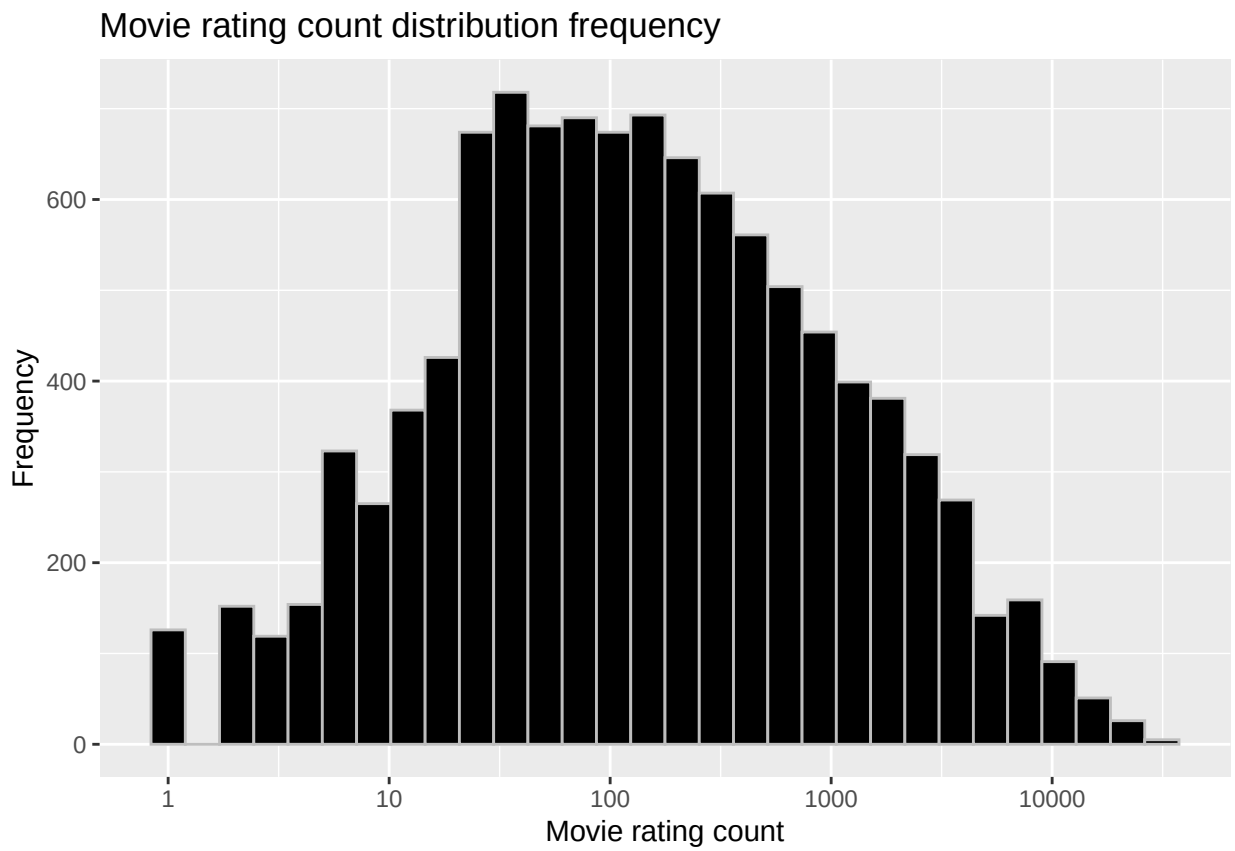



```
# Plot movie genre against average ratings
edx_unpivot_genres %>% group_by(genre) %>%
  summarize(avg_rating = mean(rating)) %>%
  ggplot(aes(genre, avg_rating)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90)) +
  labs(title = "Genre average ratings", x = "Genres", y = "Average rating")
```



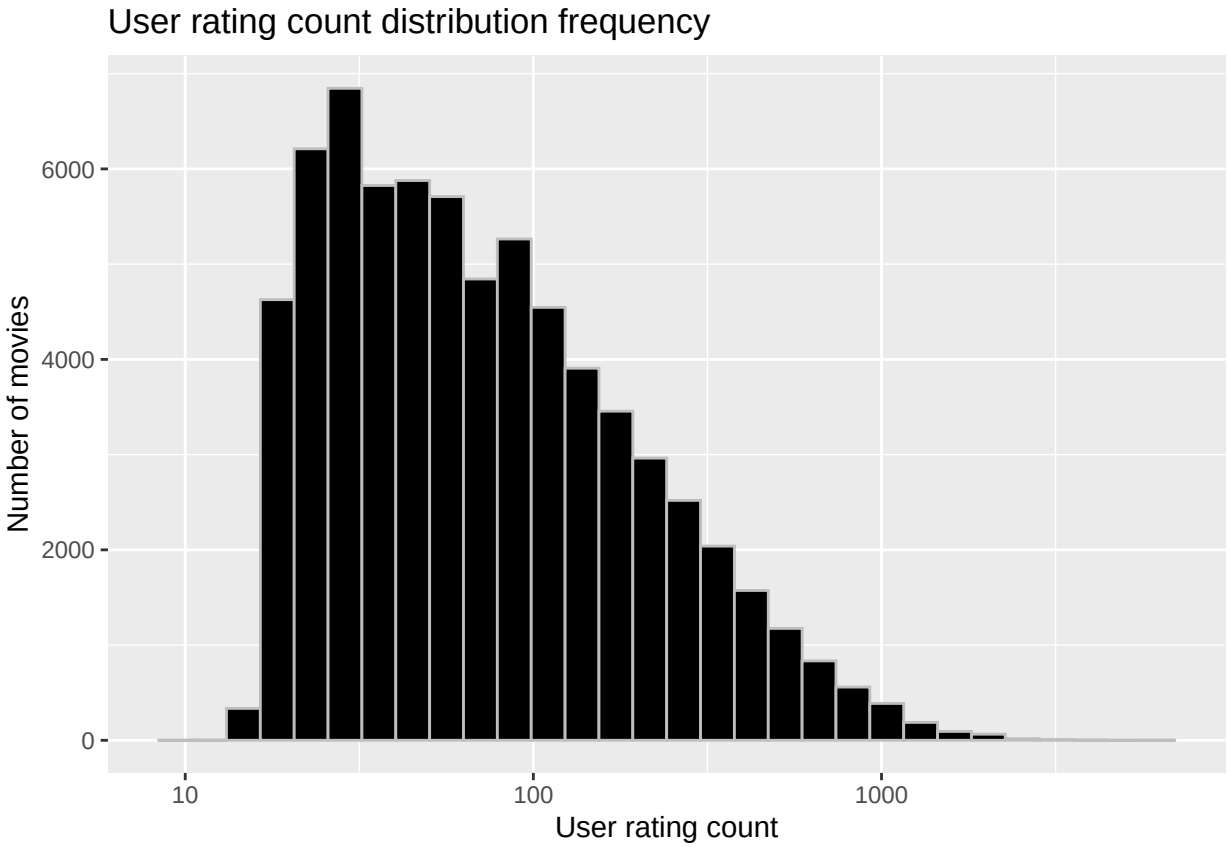
In addition, the below histogram shows that movie ratings count ranged from one to more than one hundred thousand. Based on statistics, the more a movie was rated, the closer its mean will be to the actual rating. Hence, this effect has to be accounted in the model.

```
# Plot movie rating count distribution frequency
edx %>% count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins=30, fill="black", col="grey") +
  scale_x_log10() +
  labs(title = "Movie rating count distribution frequency", x = "Movie rating count",
       y = "Frequency")
```



Similar to movie rating count, the distribution of user rating count was observed to stretch over a wide range. Again, users who give fewer ratings will likely be inconsistent in their ratings, therefore this effect is also accounted in the model.

```
# Plot user rating count distribution frequency
edx %>% count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins=30, fill="black", col="grey") +
  scale_x_log10() +
  labs(title = "User rating count distribution frequency", x = "User rating count",
       y = "Number of movies")
```



3 Model Construction

3.1 Baseline Model

A baseline model which assumes the same rating for without adjusting for any users, movies, genres or year effect is established:

$$Y = \mu + \epsilon$$

where

$$\epsilon$$

the independent errors sampled from the same distribution centered around 0 and

$$\mu$$

the "true" rating for all movies. For the baseline model, the average of all ratings,

$$\mu$$

, is used to calculate the starting RMSE.

```
# First model
mu_hat <- mean(edx$rating)
model_1_rmse <- RMSE(validation$rating, mu_hat)
rmse_table <- tibble(Method = "Baseline average model", RMSE = model_1_rmse)
knitr::kable(head(rmse_table), "simple")
```

Method	RMSE
Baseline average model	1.061202

3.2 Modeling release year effects

Since it is believed that the movie year might have an effect on the model, a release year-specific effect b_i is added to improve the baseline model:

$$Y = \mu + b_i + \epsilon$$

```
# Modeling release year effects
mu <- mean(edx$rating)
release_year_avg <- edx %>%
  group_by(release_year) %>%
  summarize(b_i = mean(rating - mu))
predicted_ratings <- mu + validation %>%
  left_join(release_year_avg, by='release_year') %>%
  pull(b_i)
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_table <- rbind(rmse_table, tibble(Method = "Release year model",
                                       RMSE = model_2_rmse))
knitr::kable(head(rmse_table), "simple")
```

Method	RMSE
Baseline average model	1.061202
Release year model	1.050026

3.3 Modeling rating year and release year effects

Next, the effect of the rating year, b_u, was added to the model to improve it further.

$$Y = \mu + b_i + b_u + \epsilon$$

```
# Modeling rating year and release year effects
rating_year_avg <- edx %>%
  left_join(release_year_avg, by='release_year') %>%
  group_by(rating_year) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_ratings <- validation %>%
  left_join(release_year_avg, by='release_year') %>%
  left_join(rating_year_avg, by='rating_year') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_table <- rbind(rmse_table, tibble(Method = "Rating and release year model",
                                       RMSE = model_3_rmse))
knitr::kable(head(rmse_table), "simple")
```

Method	RMSE
Baseline average model	1.061202
Release year model	1.050026
Rating and release year model	1.048241

3.4 Modeling genre, rating year and release year effects

Notice that there was substantial variability across genres in terms of rating count, as well as average rating. The genre related effect, b_o, is added to lower the RMSE value.

$$Y = \mu + b_i + b_u + b_o + \epsilon$$

```
# Modeling genre, rating year and release year effects
genres_avg <- edx %>%
  left_join(release_year_avg, by='release_year') %>%
  left_join(rating_year_avg, by='rating_year') %>%
  group_by(genres) %>%
  summarize(b_o = mean(rating - mu - b_i - b_u))
predicted_ratings <- validation %>%
  left_join(release_year_avg, by='release_year') %>%
  left_join(rating_year_avg, by='rating_year') %>%
  left_join(genres_avg, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_o) %>%
  pull(pred)
model_4_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_table <- rbind(rmse_table, tibble(Method = "Genre, rating and release year model",
                                       RMSE = model_4_rmse))
knitr::kable(head(rmse_table), "simple")
```

Method	RMSE
Baseline average model	1.061202
Release year model	1.050026
Rating and release year model	1.048241
Genre, rating and release year model	1.007606

3.4 Modeling movie, user, genre, rating year and release year effects

Last but not least, the two parameters that were expected to have the biggest effect on the model, movie (b_a) and user (b_e) were both added to the model. Subsequently, the predictor is constructed to see what is the degree of improvement.

$$Y = \mu + b_i + b_u + b_o + b_a + b_e + \epsilon$$

```
# Modeling movie, genre, rating year and release year effects
movie_rating_count_avg <- edx %>%
  left_join(release_year_avg, by='release_year') %>%
  left_join(rating_year_avg, by='rating_year') %>%
  left_join(genres_avg, by='genres') %>%
```

```
  group_by(movieId) %>%
  summarize(b_a = mean(rating - mu - b_i - b_u - b_o))
predicted_ratings <- validation %>%
  left_join(release_year_avg, by='release_year') %>%
  left_join(rating_year_avg, by='rating_year') %>%
  left_join(genres_avg, by='genres') %>%
  left_join(movie_rating_count_avg, by='movieId') %>%
  mutate(pred = mu + b_i + b_u + b_o + b_a) %>%
  pull(pred)
model_5_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_table <- rbind(rmse_table,
                    tibble(Method = "Movie, genre, rating and release year model",
                                        RMSE = model_5_rmse))
knitr::kable(head(rmse_table), "simple")
```

Method	RMSE
Baseline average model	1.0612018
Release year model	1.0500259
Rating and release year model	1.0482410
Genre, rating and release year model	1.0076056
Movie, genre, rating and release year model	0.9420006

```
# Modeling movie, user, genre, rating year and release year effects
user_rating_count_avg <- edx %>%
  left_join(release_year_avg, by='release_year') %>%
  left_join(rating_year_avg, by='rating_year') %>%
  left_join(genres_avg, by='genres') %>%
  left_join(movie_rating_count_avg, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_e = mean(rating - mu - b_i - b_u - b_o - b_a))
predicted_ratings <- validation %>%
  left_join(release_year_avg, by='release_year') %>%
  left_join(rating_year_avg, by='rating_year') %>%
  left_join(genres_avg, by='genres') %>%
  left_join(movie_rating_count_avg, by='movieId') %>%
  left_join(user_rating_count_avg, by='userId') %>%
  mutate(pred = mu + b_i + b_u + b_o + b_a + b_e) %>%
  pull(pred)
model_6_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_table <- rbind(rmse_table,
                    tibble(Method = "User, movie, genre, rating and release year model",
                                        RMSE = model_6_rmse))
knitr::kable(head(rmse_table), "simple")
```

Method	RMSE
Baseline average model	1.0612018
Release year model	1.0500259
Rating and release year model	1.0482410
Genre, rating and release year model	1.0076056
Movie, genre, rating and release year model	0.9420006

Method	RMSE
User, movie, genre, rating and release year model	0.8648722

3.5 Regularization of final model

As a final step, regularization will constrain the total variability of the effect sizes. The training set is used for tuning, and the tuning parameter,

$$\lambda$$

is selected using the cross-validation method. The formula for regularization of user, movie, genre, rating and release year effects is as follows.

$$\sum(y - \mu - b_i - b_u - b_o - b_a - b_e)^2 + \lambda(\sum b_i^2 + \sum b_u^2 + \sum b_o^2 + \sum b_a^2 + \sum b_e^2)$$

```
# Regularization of final model
lambda <- seq(0, 10, 0.25)

rmses <- sapply(lambda, function(l){

  mu <- mean(edx$rating)

  release_year_avg <- edx %>%
    group_by(release_year) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  rating_year_avg <- edx %>%
    left_join(release_year_avg, by='release_year') %>%
    group_by(rating_year) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))

  genres_avg <- edx %>%
    left_join(release_year_avg, by='release_year') %>%
    left_join(rating_year_avg, by='rating_year') %>%
    group_by(genres) %>%
    summarize(b_o = sum(rating - mu - b_i - b_u)/(n()+l))

  movie_rating_count_avg <- edx %>%
    left_join(release_year_avg, by='release_year') %>%
    left_join(rating_year_avg, by='rating_year') %>%
    left_join(genres_avg, by='genres') %>%
    group_by(movieId) %>%
    summarize(b_a = sum(rating - mu - b_i - b_u - b_o)/(n()+l))

  user_rating_count_avg <- edx %>%
    left_join(release_year_avg, by='release_year') %>%
    left_join(rating_year_avg, by='rating_year') %>%
    left_join(genres_avg, by='genres') %>%
    left_join(movie_rating_count_avg, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_e = sum(rating - mu - b_i - b_u - b_o - b_a)/(n()+l))

  predicted_ratings <- validation %>%
```

```
    left_join(release_year_avg, by='release_year') %>%
    left_join(rating_year_avg, by='rating_year') %>%
    left_join(genres_avg, by='genres') %>%
    left_join(movie_rating_count_avg, by='movieId') %>%
    left_join(user_rating_count_avg, by='userId') %>%
    mutate(pred = mu + b_i + b_u + b_o + b_a + b_e) %>%
    pull(pred)

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

rmse_table <- rbind(rmse_table,
                    tibble(Method = "Regularized final model",
                           RMSE = min(rmses)))

knitr::kable((rmse_table), "simple")
```

Method	RMSE
Baseline average model	1.0612018
Release year model	1.0500259
Rating and release year model	1.0482410
Genre, rating and release year model	1.0076056
Movie, genre, rating and release year model	0.9420006
User, movie, genre, rating and release year model	0.8648722
Regularized final model	0.8643132

Conclusion

In conclusion, user and movie effects had the biggest impact on the model as expected. In comparison, the rating and release year had a limited impact. Regardless, if the model was built on all five parameters and regularized, the RMSE achieved was able to beat the target.