MovieLens Project

Hong Wei

6/3/2020

Introduction

The objective of this project is to construct a movie recommendation system with machine learning techniques and other tools which we have learned throughout the HarvardX Data Science Professional Certificate Programme. We will be using the 10M version of MovieLens dataset. The dataset is a large data of 10 million ratings submitted by more than 70,000 users on more than 10,000 movies. Naturally not all users will rate every movie available. A movie recommendation system will help to predict the rating that a user will give to a specific movie, which is what we aim to achieve in this project.

---

1. Data Setup

Here we describe the steps taken to download the raw dataset from the source, and then to generate a training data set which we can use to perform some data explorations in the next section.

1.1 Preparing the data

We first load the packages required in RStudio working environment

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

Then we download the raw dataset from the source.

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

Note that the ratings and movie information are kept in different data files (`ratings.dat` and `movies.dat`). Key variables such as `movieId`, `userId`, `title` and `genres` are extracted from both files to form the `movielens` dataset.

```
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(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")
```

1.2 Overview of the data

Reading from the basic structure of the `movielens` dataset:

```
glimpse(movielens)
```

```
## Rows: 10,000,054
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 231, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983392, 838983421, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dumber (19...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Comedy", "Act...
```

and some sample data:

```
sample_n(movielens, 10)
```

```
##    userId movieId rating  timestamp
## 1   40453    1288    3.0  975559789
## 2   64301     321    3.0  945227223
## 3   20848   33493    4.0 1196185136
## 4   41337    1762    1.0  967053235
## 5   33414     161    5.0  836515594
## 6   25011    1009    4.0  939484000
## 7   32660    2145    3.0 1132435153
## 8   44227     316    3.0  830973714
## 9   35182    7318    3.5 1201265943
## 10  13274    2011    4.0  963745458
##                                                 title
## 1                             This Is Spinal Tap (1984)
## 2  Strawberry and Chocolate (Fresa y chocolate) (1993)
## 3  Star Wars: Episode III - Revenge of the Sith (2005)
## 4                                   Deep Rising (1998)
## 5                                  Crimson Tide (1995)
## 6                       Escape to Witch Mountain (1975)
## 7                                Pretty in Pink (1986)
## 8                                     Stargate (1994)
## 9                     Passion of the Christ, The (2004)
## 10                    Back to the Future Part II (1989)
##                            genres
```

```
## 1                    Comedy|Musical
## 2                             Drama
## 3  Action|Adventure|Fantasy|Sci-Fi
## 4             Action|Horror|Sci-Fi
## 5               Drama|Thriller|War
## 6       Adventure|Children|Fantasy
## 7             Comedy|Drama|Romance
## 8          Action|Adventure|Sci-Fi
## 9                             Drama
## 10  Action|Adventure|Comedy|Sci-Fi
```

Some basic observations can be made:

- Each observation/row represents a rating given by one user to one movie at one time.
- Each user and each movie are identified by `userId` and `movieId` respectively.
- One movie can be tagged by one or more genres.
- Timestamp refers to the time the user makes the rating.
- The release year of a movie is stated in the title. For example "Star Trek: Generations (1994)".

1.3 Creating training set

Before we perform any data exploration in detail, we split the `movielens` dataset into a training set named `edx` and a validation set named `validation`. The validation set is 10% of the `movielens` dataset.

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

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

We further split the `edx` into a training set and a testing set. This is to ensure that the **validation set is not being used to form any opinion or conclusion during the data exploration or model training process**. The validation set will only be used for the evaluation of the final algorithm.

```
#Test set will be 10% of edx set
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

#Make sure userId and movieId in validation set are also in train set
```

```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

---

2. Data Exploration

In this section, we explore into details using the `train_set` which has more than 8 million ratings. The objective is to identify the potential effects (factors) which can explain the variability of the ratings of to a movie. Then we can use these effects to construct our model in the later section.

2.1 Overview of Ratings

The `train_set` has 8100048 ratings, with an average rating of 3.513.

```
nrow(train_set) #number of ratings
```

```
## [1] 8100048
```

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

```
## [1] 3.512509
```

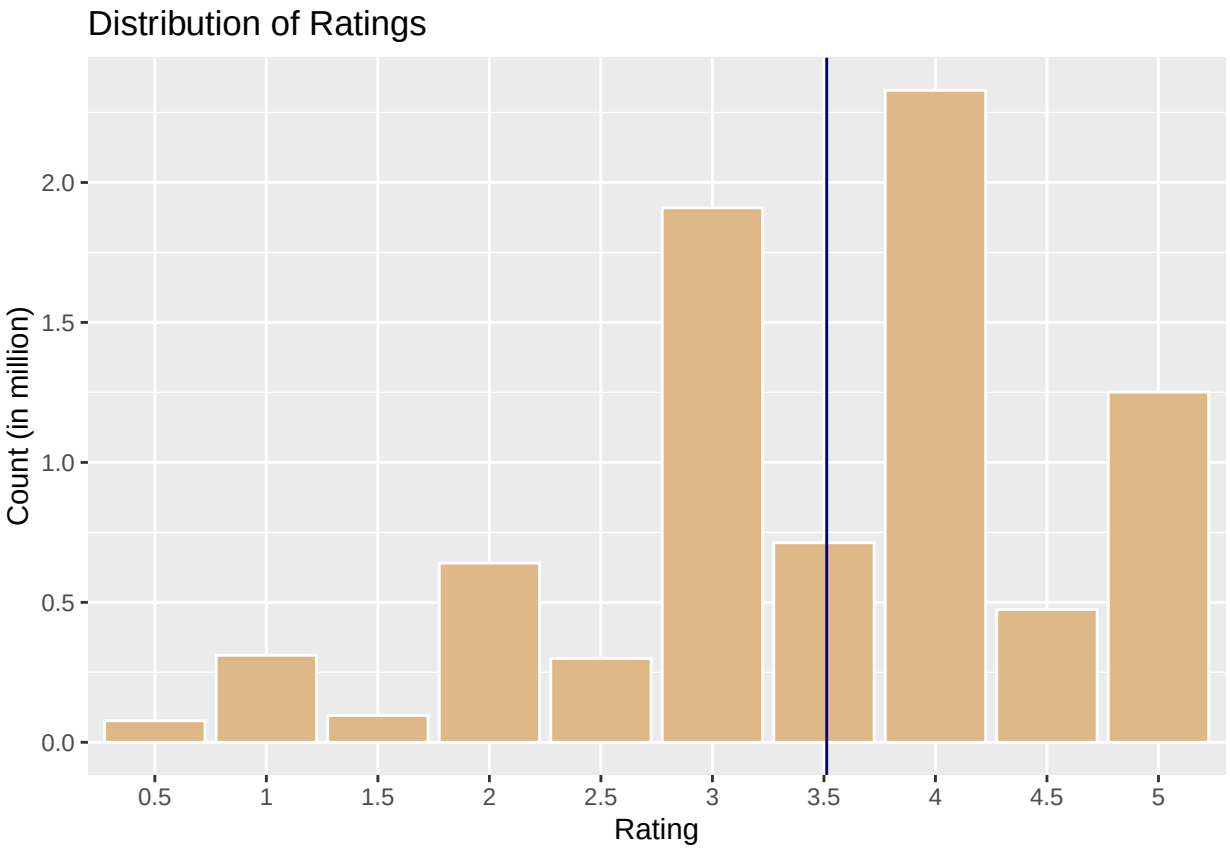
The number of unique users and unique movies are:

```
train_set %>% summarise(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10659
```

A user can give a rating between 0.5 to 5.0 to a movie. The distribution below shows that higher ratings are more frequent than others. Full star ratings (e.g. 3.0, 4.0) are also more frequent than half star ratings (e.g. 3.5, 4.5) in general.

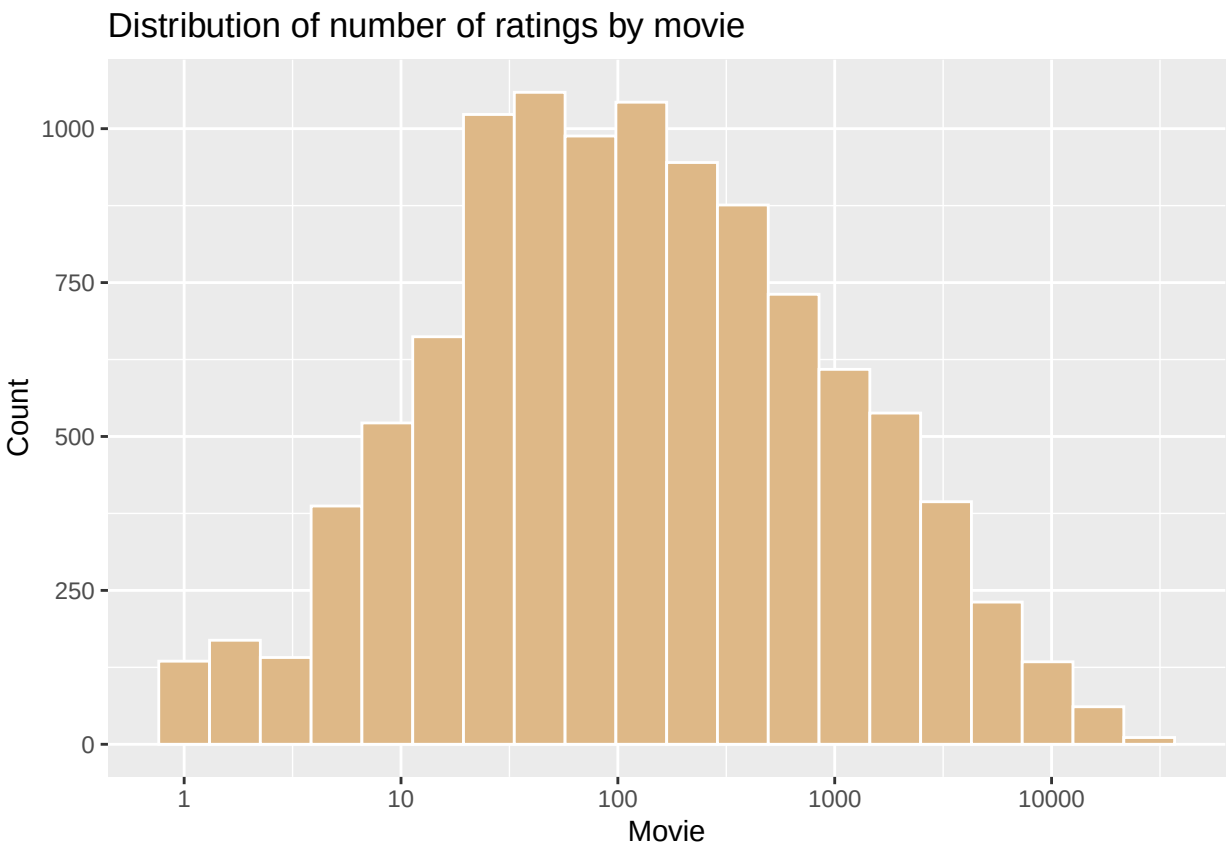
```
#Plot number of ratings by ratings
train_set %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = factor(rating), y = count/1000000)) +
  geom_bar(stat = "identity", fill = "burlywood", color = "white") +
  labs(x="Rating", y="Count (in million)") +
  geom_vline(xintercept = mu*2, color = "blue4") +
  ggtitle("Distribution of Ratings")
```



2.2 Movie Effect

Intuitively some movies such as blockbusters are watched much more times than other movies relatively, hence more ratings are given to these movies. This is shown by the distribution of number of ratings by different movies.

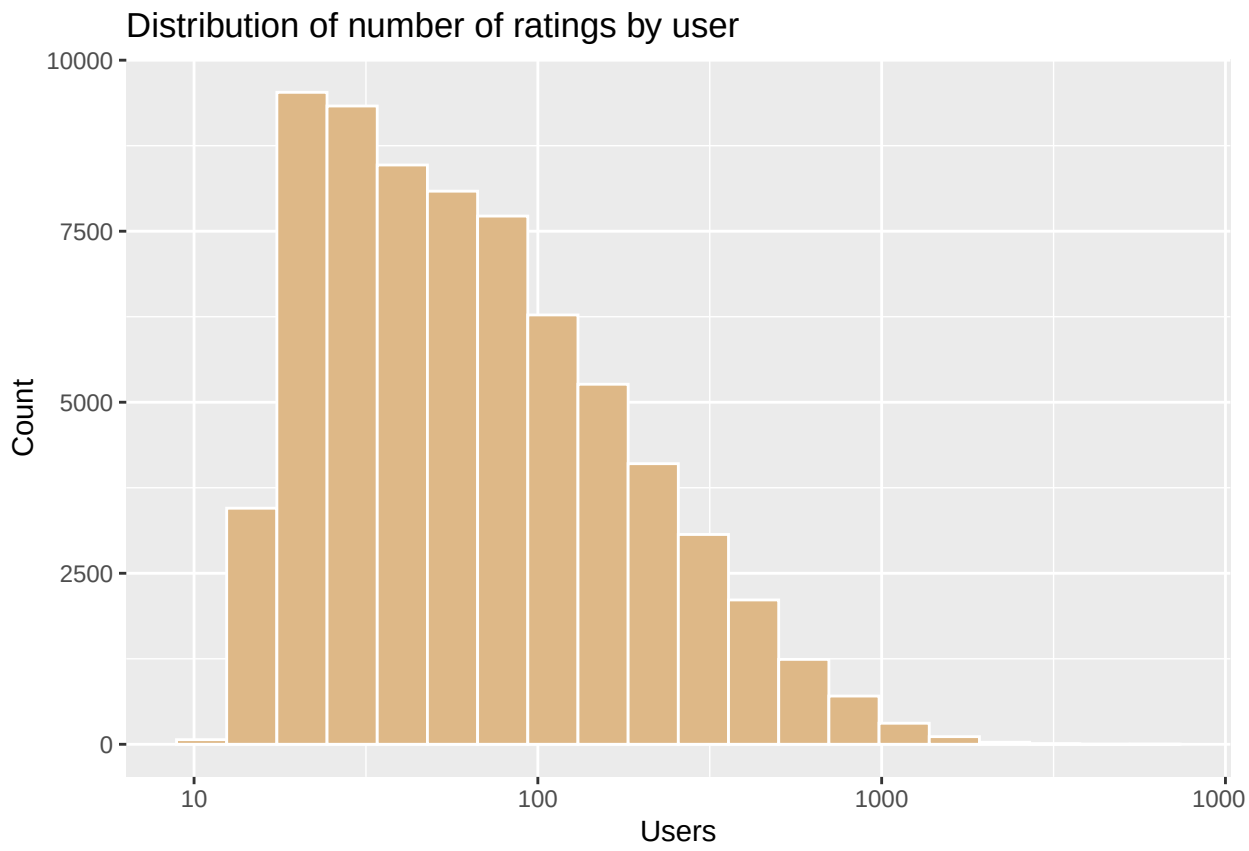
```
#Plot number of ratings by movie
train_set %>% group_by(movieId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 20, fill = "burlywood", color = "white") +
  scale_x_log10() +
  labs(x="Movie", y="Count") +
  ggtitle("Distribution of number of ratings by movie")
```



2.3 User Effect

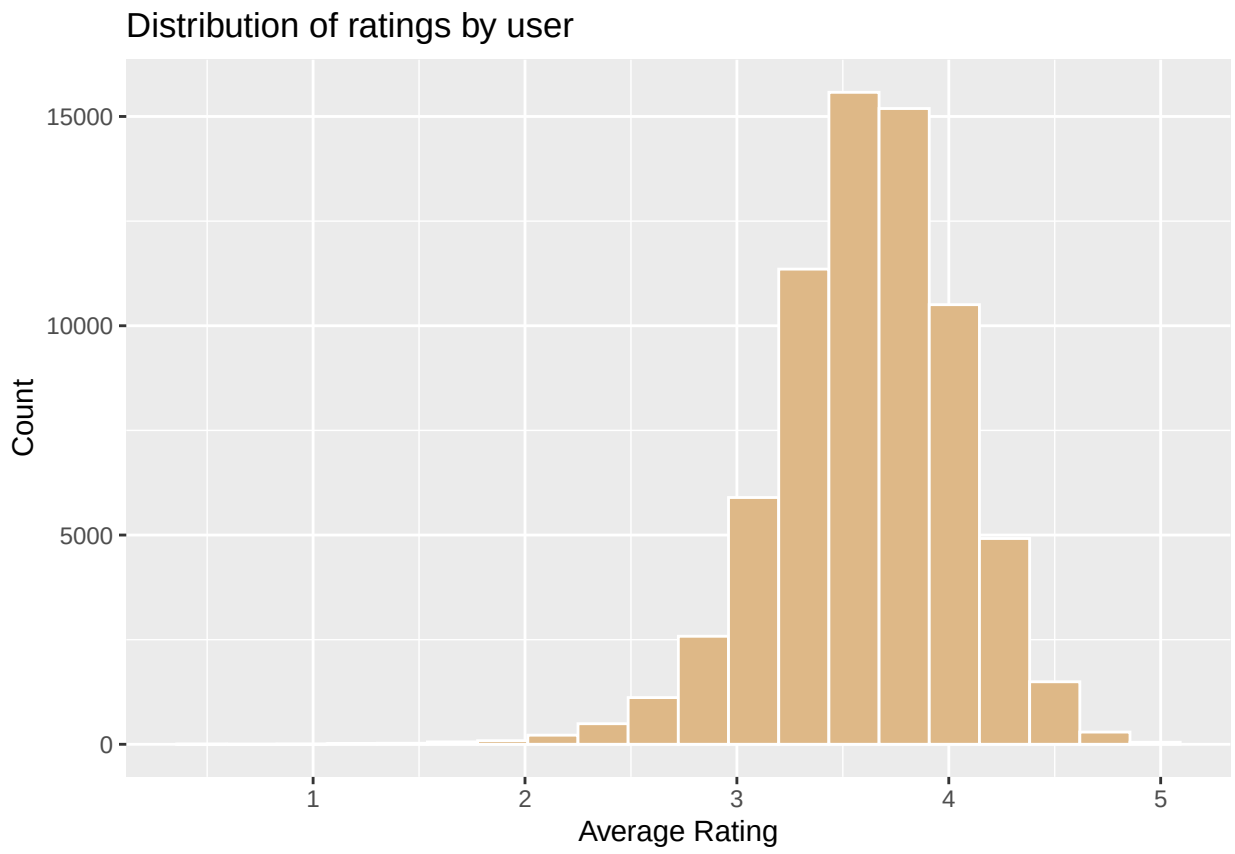
Some users are also more active than others at rating movies. This is also demonstrated by the distribution of number of ratings by different users.

```
#Plot number of ratings by user
train_set %>% group_by(userId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 20, fill = "burlywood", color = "white") +
  scale_x_log10() +
  labs(x="Users", y="Count") +
  ggtitle("Distribution of number of ratings by user")
```



Looking at the plot below, we can also see that there are some group of users who are very critical and tend to give low ratings, while some users love every movie and tend to give ratings higher than average.

```
#Plot average ratings by user
train_set %>% group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n() > 100) %>% #filter by users who have rated more than 100 movies
  ggplot(aes(b_u)) +
  geom_histogram(bins = 20, fill = "burlywood", color = "white") +
  labs(x="Average Rating", y="Count") +
  ggtitle("Distribution of ratings by user")
```

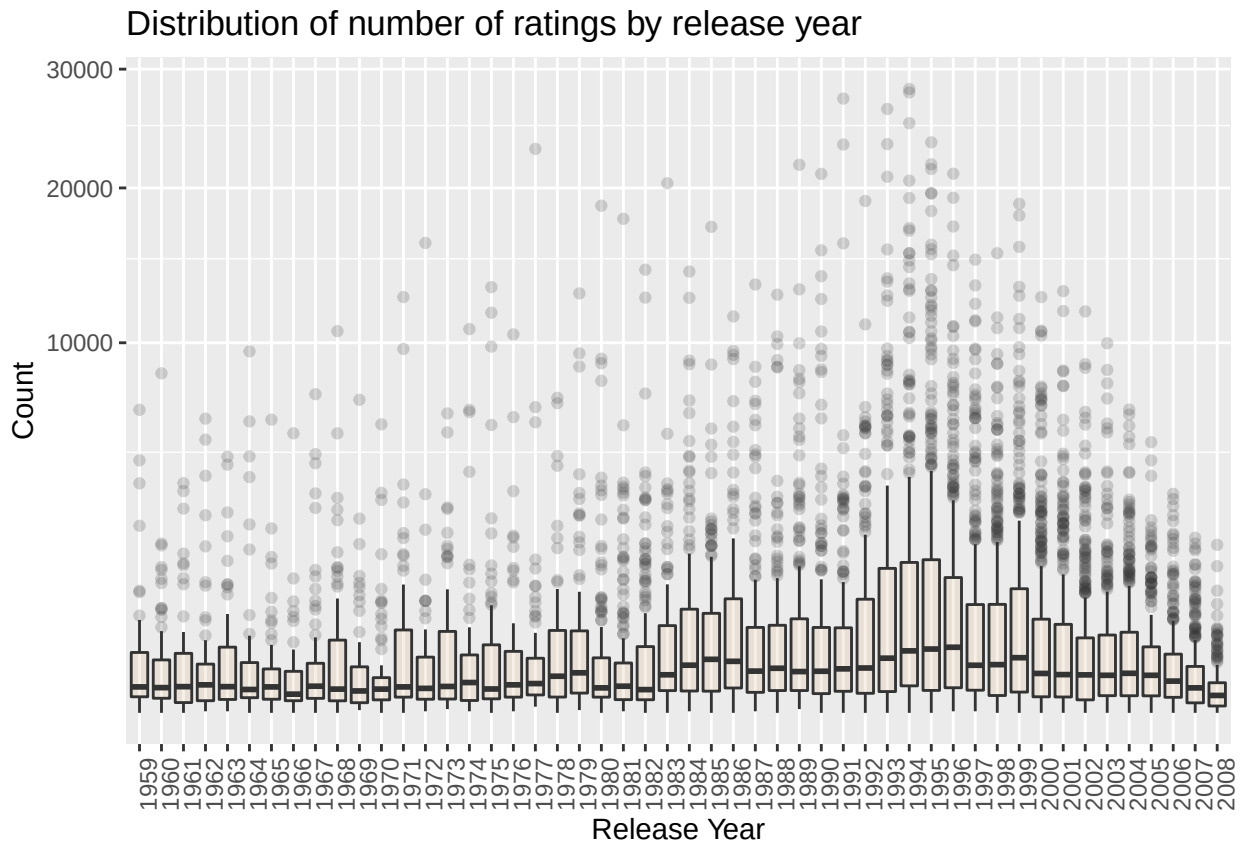


2.4 Time Effect

2.4.1 Release Year

Movies that came out earlier may have more ratings than others since more users would have watched them. In the following plot, we explore the relationship between the number of ratings for each movie, and the year the movie came out. (Note that the release year value is extracted from the movie title)

```
#Plot number of ratings by release year
train_set %>%
  mutate(release = str_sub(title, start = -5, end = -2)) %>%
  filter(as.numeric(release) > 1958) %>%
  group_by(movieId) %>%
  summarize(n = n(), release = first(release)) %>%
  ggplot(aes(x = release, y = n)) +
  geom_boxplot(fill = "burlywood", alpha = 0.2) +
  coord_trans(y = "sqrt") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("Release Year") + ylab("Count") +
  ggtitle("Distribution of number of ratings by release year")
```

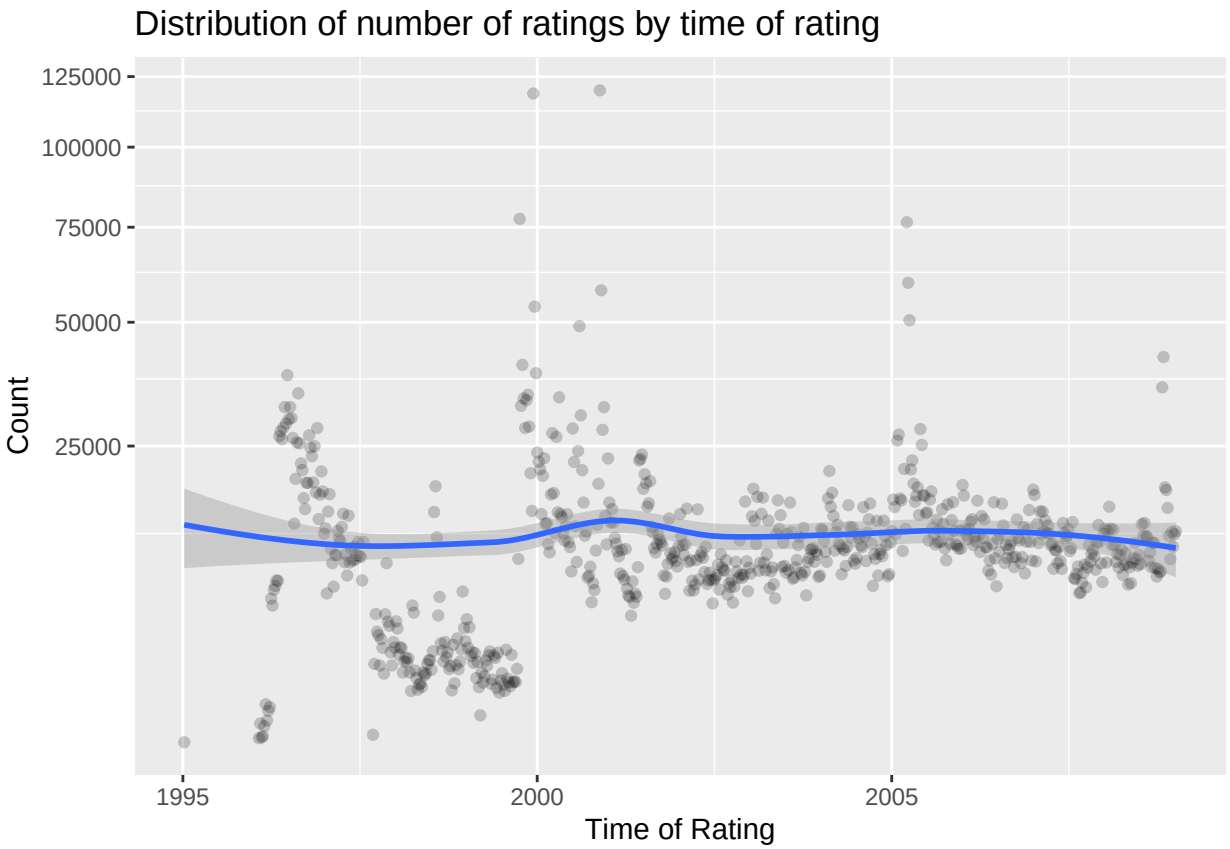


It seems that movies released in the 1990-2000 periods have more ratings on average. However, the rate of ratings tend to decrease for newer movies. This can be explained by the logic that the more recent a movie is, the less time users have time to rate it.

2.4.2 Time of Rating

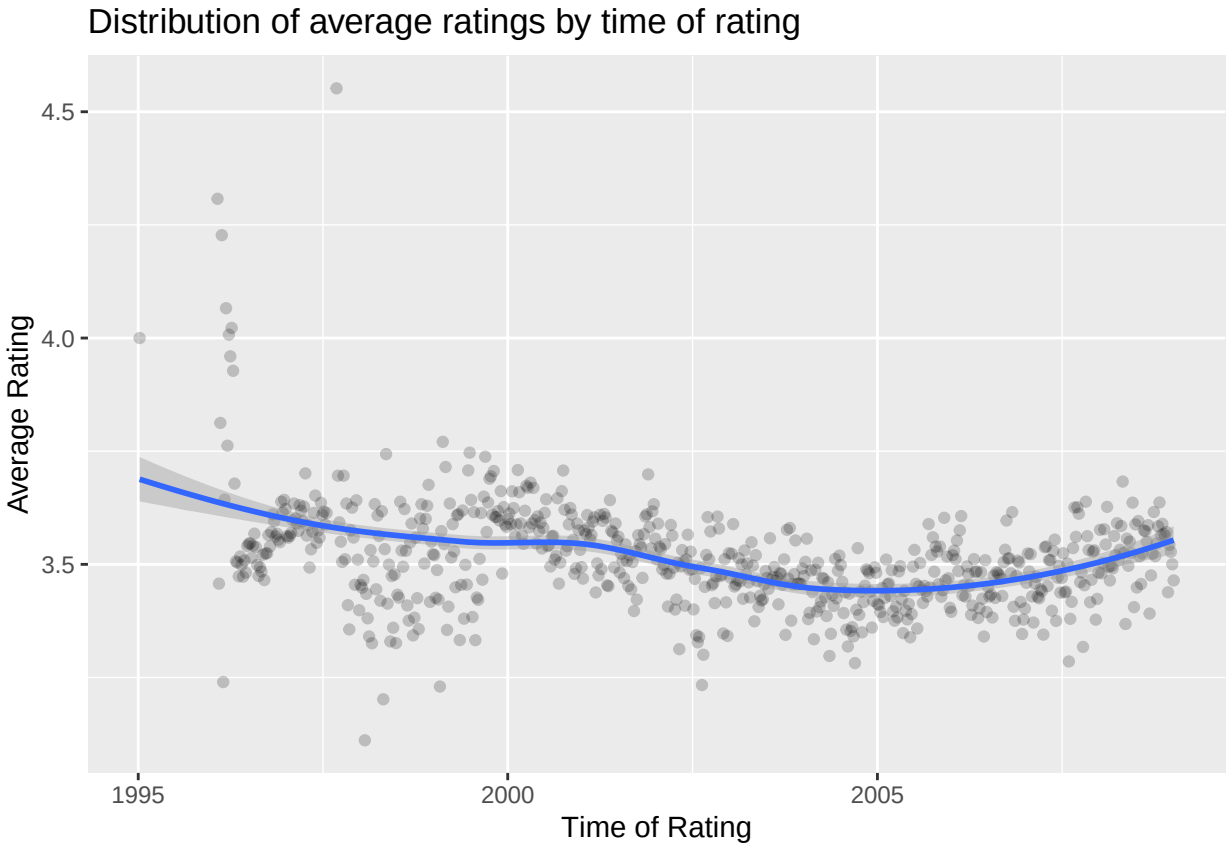
There could also be a time trend in effect where number of ratings increase over time. This could be due to more users have access to the rating platform, or more movies are easier to be accessed by more users over time. However, it is not evident from the plot below.

```
#Plot number of ratings by time
train_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(n = n()) %>%
  ggplot(aes(date, n)) +
  coord_trans(y = "sqrt") +
  geom_point(alpha = 0.2) +
  geom_smooth(method = "loess", formula = "y ~ x") +
  xlab("Time of Rating") + ylab("Count") +
  ggtitle("Distribution of number of ratings by time of rating")
```



We also examine the effect of time of rating on the average rating. It does show some level of time effect on the average rating.

```
#Plot average ratings by time
train_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point(alpha = 0.2) +
  geom_smooth(method = "loess", formula = "y ~ x") +
  labs(x="Time of Rating", y="Average Rating") +
  ggtitle("Distribution of average ratings by time of rating")
```



2.5 Genre Effect

While movies can generally be categorised into different genres such as "Comedy", "Action" and "Horror", most of the movies in the dataset are categorized (or "tagged") into one or more genres. For example, from the first few entries of `train_set`, we can see that the movie "Outbreak (1995)" is tagged into four genres, namely "Action", "Drama", "Sci-Fi" and "Thriller".

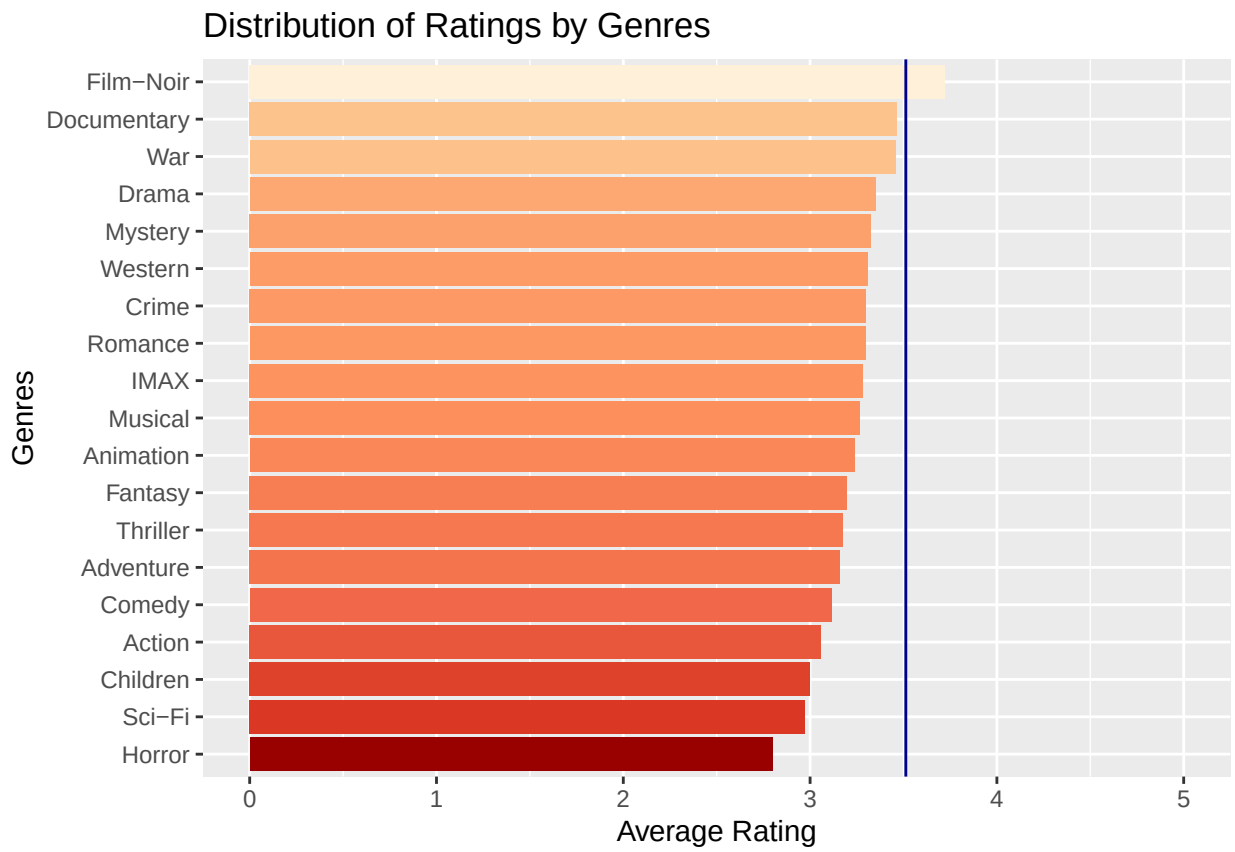
```
head(train_set, 3)
```

```
##   userId movieId rating timestamp            title                         genres
## 1      1     122      5 838985046 Boomerang (1992)                Comedy|Romance
## 2      1     185      5 838983525  Net, The (1995)         Action|Crime|Thriller
## 4      1     292      5 838983421  Outbreak (1995) Action|Drama|Sci-Fi|Thriller
```

In the following code, we extract the different genres values to examine its effect on the average rating.

```
#Plot average ratings by genres
genres <- train_set %>%
  group_by(movieId) %>%
  summarize(r = mean(rating), title = title[1], genre = genres[1]) %>%
  separate_rows(genre, sep = "\\|") %>%
  group_by(genre) %>%
  summarize(r = mean(r)) %>%
  filter(!genre == "(no genres listed)")
```

```r
genres %>%
  ggplot(aes(x=r, y=reorder(genre, r), fill=r)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  coord_cartesian(xlim = c(0, 5)) +
  scale_fill_distiller(palette = "OrRd") +
  labs(x="Average Rating", y="Genres") +
  geom_vline(xintercept = mu, color = "blue4") +
  ggtitle("Distribution of Ratings by Genres")
```



From the plot above, it seems that movies tagged with the genre "Film-Noir" tend to have higher ratings, while movies in the "Horror" and "Sci-Fi" genres tend to have lower ratings. The average rating is also plotted as a blue line as a reference.

Do note that there is one movie with "no genres listed" and it was removed from the plot above.

```r
#Movie without genres
train_set %>% filter(genres == "(no genres listed)") %>% pull(title) %>% first()
```

```
## [1] "Pull My Daisy (1958)"
```

---

3. Model Strategy

In this section, we begin to construct our model to predict the rating of a movie using the effects which we have discussed in the earlier section. We will also use some advanced techniques such as regularization and matrix factorization to improve our model.

3.1 Loss Function

We will be using Residual Mean Squared Error (RMSE) on `test_set` to determine the final model. Basically this is a gauge on how well our model is able to predict the rating of a movie. A well-performing model should have a small RMSE. It is important to note that we use the `test_set`, not the `validation` dataset, throughout this section to help us on deciding the final model. The `validation` set will only be used for evaluation of the RMSE of the final model.

We define $y_{u,i}$ as a rating for a movie i by a user u, and we use $\hat{y}_{u,i}$ to denote the predictions of a rating. Then RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})}$$

with N being the number of user/movie combinations and the sum occuring over all these combinations.

We would prefer the RMSE of our final model to be as small as possible. The function below computes the RMSE of our prediction of ratings against the actual ratings in the `test_set`.

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

3.2 Naive Approach

Suppose we were to simply guess what the rating would be for any movie, our best choice would probably be the average rating of all the movies available, regardless of other factors. This is also known as the Naive Approach. The model looks like this:

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

with $\varepsilon_{u,i}$ independent errors sampled from the same distribution and μ is the true mean rating from the population. Using the least squares estimate of μ, which is also the average of all ratings, to predict all unknown ratings, we will obtain the RMSE as following:

```
#Naive Approach
mu <- mean(train_set$rating)
mu
```

```
## [1] 3.512509
```

```
rmse_naive <- RMSE(test_set$rating, mu)
rmse_naive
```

```
## [1] 1.061135
```

The RMSE is around 1. We will incorporate other effects and techniques to see if we can improve the RMSE.

3.3 Adding Movie Effect

From the previous section, we see different movies tend to get different ratings. So we can augment the model from our naive approach by adding the term b_i to represent the average ranking for each movie i.

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

The least squares estimate of b_i is the average of $Y_{u,i} - \hat{\mu}$ for each movie i. So, we can compute $\hat{b}_i$, $\hat{y}_{u,i}$ and the RMSE using this code:

```
#Movie effect b_i
movie_avgs <- train_set %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
y_hat <- mu + test_set %>% left_join(movie_avgs, by = "movieId") %>% pull(b_i)
rmse_i <- RMSE(test_set$rating, y_hat)
rmse_i
```

```
## [1] 0.9441568
```

We see that the RMSE has improved after adding the movie effect into our model.

3.4 Adding User Effect

We also notice that the user effects also contribute to the variability of ratings from the previous section. So we want to explore whether the user effect can improve our model by adding a user-specific effect term b_u:

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

We can compute the least square estimate of b_u using the average of $Y_{u,i} - \hat{\mu} - \hat{b}_i$. Then we add in the predictors $\hat{b}_i$ and $\hat{b}_u$ back to μ to compute $\hat{y}_{u,i}$. Let's see whether the RMSE improves:

```
#User effect b_u
user_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
y_hat <- test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
rmse_iu <- RMSE(test_set$rating, y_hat)
rmse_iu
```

```
## [1] 0.8659736
```

We see that the RMSE has further improved after adding the user effect into our model.

3.5 Adding Time of Rating Effect

The *Distribution of average ratings by time of rating* plot in the previous section shows that time of rating has some effect on the average rating. So let's try to incorporate time of rating into our model to see if the RMSE improves.

We will be testing the model:

$$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \varepsilon_{u,i}$$

with $d_{u,i}$ as the day of a rating by a user u to a movie i, and f a smooth function of $d_{u,i}$.

Using the similar approach, we can estimate $f(\hat{d}_{u,i})$ using this code:

```
#Time of rating effect b_t
#we round the date by week
time_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))
y_hat <- test_set %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(time_avgs, by = "date") %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  pull(pred)
rmse_iut <- RMSE(test_set$rating, y_hat)
rmse_iut
```

```
## [1] 0.8658902
```

The result shows that the RMSE barely improves.

3.6 Adding Genre Effect

Since the Time of Rating effect does not improve our model by much, let's consider replacing it with the genre effect instead.

Our model will be:

$$Y_{u,i} = \mu + b_i + b_u + \frac{1}{n}[\sum_{k=1}^{K}(x_{u,i} \times score_k)] + \varepsilon_{u,i}$$

where $x_{u,i} = 1$ if the movie i has genre k, and n is the number of genres tagged to movie i.

From the previous section, we know that there are 19 possible genres and all of the movies (except one in our `train_set`) has one or more genres.

```
nrow(genres)
```

```
## [1] 19
```

In order to quantity these genres, first we assign a score to each possible genre based on their average ratings using this code:

```
#count how many genres are tagged to one movie
genre_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(r = mean(rating), title = title[1], genre = genres[1]) %>%
  mutate(count = str_count(genre, "\\|")+1)

#rank each genre by average rating, assign a score of 0 to 10 to each genre
k_rank <- genre_avgs %>%
       separate_rows(genre, sep = "\\|") %>%
       group_by(genre) %>%
       summarize(r = mean(r)) %>%
       filter(!genre == "(no genres listed)") %>%
       arrange(desc(r)) %>%
       mutate(score = seq(10,0,len=19)) %>%
       select(-r)
```

The score table looks like this:

Table 1: Score of each genre

genre	score
Film-Noir	10.0000000
Documentary	9.4444444
War	8.8888889
Drama	8.3333333
Mystery	7.7777778
Western	7.2222222
Crime	6.6666667
Romance	6.1111111
IMAX	5.5555556
Musical	5.0000000
Animation	4.4444444
Fantasy	3.8888889
Thriller	3.3333333
Adventure	2.7777778
Comedy	2.2222222
Action	1.6666667
Children	1.1111111
Sci-Fi	0.5555556
Horror	0.0000000

We also know that the maximum number of genre that a movie i is tagged to, is 8.

```
#max number of genre per movie is 8
max(genre_avgs$count)
```

```
## [1] 8
```

In the following code, we extract the genres information into 8 columns, then convert the genres into their respective scores. We further compute the average genre score `kscore` for each movie.

```
#calculate genre score of each movie
temp_rank <- setNames(k_rank$score, k_rank$genre)
temp <- genre_avgs %>% select(movieId, r, genre) %>%
  separate(genre, into = paste0("k",seq(1,8,1)), sep = "\\|", fill = "right")
dat <- purrr::map_df(temp[,3:10],~temp_rank[.x])
dat <- dat %>% mutate(kscore = round(rowMeans(., na.rm = TRUE),2))
genre_avgs <- genre_avgs %>% cbind(kscore = dat$kscore) %>% select(movieId, kscore)
```

We estimate the effect of this average genre score on the average rating using the same approach as before. Then we do the same for the `test_set` to calculate the RMSE result.

```
#Genre effect b_k
k_avgs <- train_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "movieId") %>%
  group_by(kscore) %>%
  summarize(b_k = mean(rating - mu - b_i - b_u))

#compute kscore for test set
temp <- test_set %>%
       group_by(movieId) %>%
       summarize(r = mean(rating), genre = genres[1]) %>%
        separate(genre, into = paste0("k",seq(1,8,1)), sep = "\\|", fill = "right")
dat <- purrr::map_df(temp[,3:10],~temp_rank[.x])
dat <- dat %>% mutate(kscore = round(rowMeans(., na.rm = TRUE),2))
temp <- temp %>% cbind(kscore = dat$kscore) %>% select(movieId, kscore)
y_hat <- test_set %>%
  left_join(temp, by = "movieId") %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(k_avgs, by = "kscore") %>%
  mutate(pred = mu + b_i + b_u + b_k) %>%
  pull(pred)
rmse_iuk <- RMSE(test_set$rating, y_hat)
rmse_iuk
```

```
## [1] 0.8657599
```

Based on the RMSE, the genre effect performs better than the time effect. However, the improvement seems to be minimal.

3.7 An interim conclusion

Before moving forward, let's take a brief look of our result thus far:

```
rmsetable <- tibble(Model = c(1:5),
                    Method = c("Naive Approach",
                               "Movie Effect",
                               "Movie + User Effects",
                               "Movie + User + Time Effects",
                               "Movie + User + Genre Effects"),
                    RMSE = c(rmse_naive, rmse_i, rmse_iu, rmse_iut, rmse_iuk))
```

Table 2: RMSE Results

Model	Method	RMSE
1	Naive Approach	1.0611350
2	Movie Effect	0.9441568
3	Movie + User Effects	0.8659736
4	Movie + User + Time Effects	0.8658902
5	Movie + User + Genre Effects	0.8657599

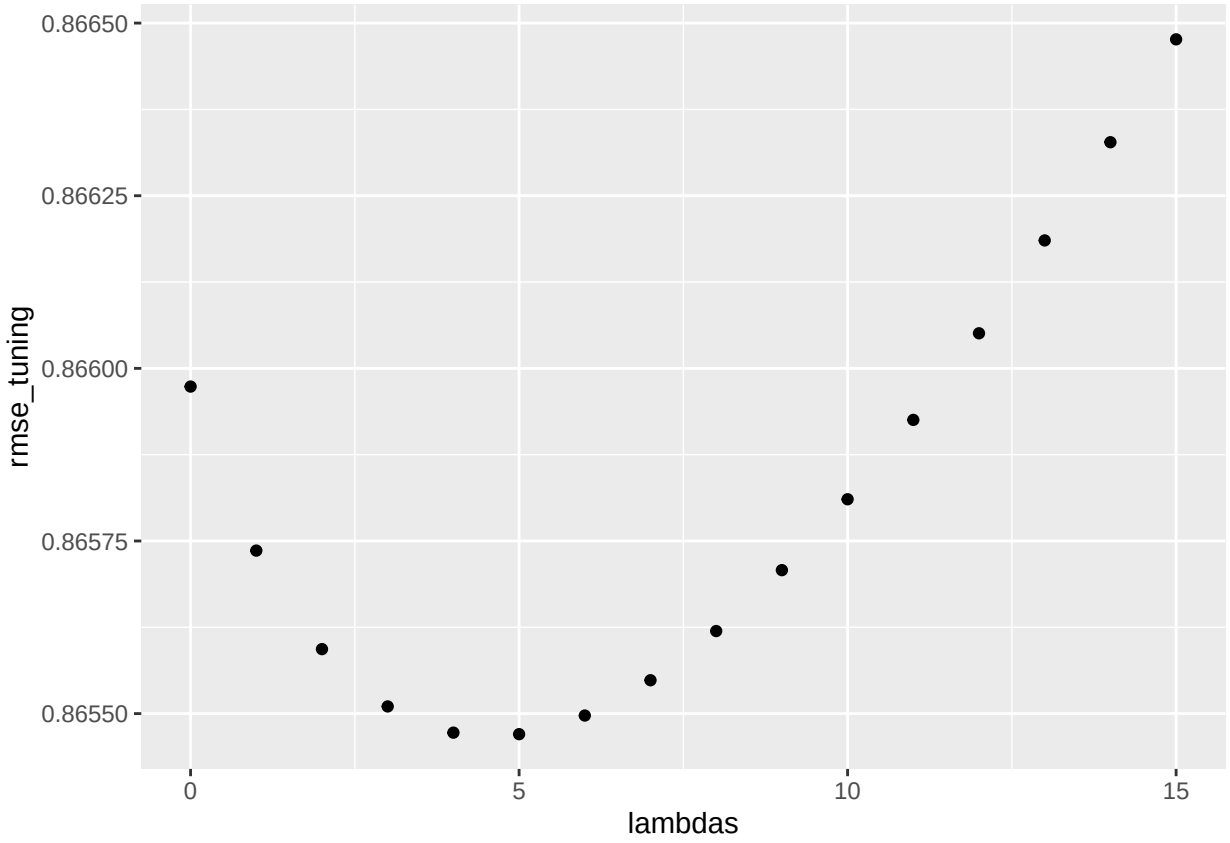
So far we have built our model by adding on effects which we thought to be able to explain the variability of the average ratings. Judging by the RMSE result from the table above and considering the computation time required, we decide to select **Model 3** to proceed into our next analysis.

3.8 Regularization

To prevent overfitting on the dataset, we apply regularization to constrain the total variability of the effect sizes. Using the penalized least square method, we add a penalty term λ to n when we estimate the least squares in our model.

To choose which λ to use, we use the `test_set` for cross validation.

```
#Tuning lambda
lambdas <- seq(0, 15, 1)
rmse_tuning <- sapply(lambdas, function(x){
  b_ix <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n() + x))
  b_ux <- train_set %>%
    left_join(b_ix, by = "movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n() + x))
  y_hat <- test_set %>%
    left_join(b_ix, by = "movieId") %>%
    left_join(b_ux, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>% pull(pred)
  return(RMSE(test_set$rating, y_hat))
})
qplot(lambdas, rmse_tuning)
```



Based on the cross validation, the optimal λ is:

```
lambdas[which.min(rmse_tuning)] #5
```

```
## [1] 5
```

Adding this to our result table:

```
rmsetable <- bind_rows(rmsetable, tibble(Model = 6,
                                         Method = "Regularized Movie + User Effects",
                                         RMSE = min(rmse_tuning)))
```

Table 3: RMSE Results

Model	Method	RMSE
1	Naive Approach	1.0611350
2	Movie Effect	0.9441568
3	Movie + User Effects	0.8659736
4	Movie + User + Time Effects	0.8658902
5	Movie + User + Genre Effects	0.8657599
6	Regularized Movie + User Effects	0.8654703

3.9 Matrix Factorization

Our model has attempted to account for movie-to-movie differences through b_i and user-to-user differences through b_u. However, certain groups of movies or groups of users have similar rating patterns too. In order to quantify these sources of variation, we will implement matrix factorization to our model.

We choose the `recosystem package` (more info here) to perform matrix factorization on the **residuals** of our model. We generate the matrix files of `train_set` containing `userId`, `movieId` and `residual`; and `test_set` containing `userId`, `movieId` and `rating`.

```
#Matrix factorisation
#Using lambda = 5
l = 5
b_ir <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n() + l))
b_ur <- train_set %>%
  left_join(b_ir, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i)/(n() + l))
y_hat_r <- test_set %>%
  left_join(b_ir, by = "movieId") %>%
  left_join(b_ur, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>% pull(pred)
#compute model residual
model_residual <- train_set %>%
  left_join(b_ir, by = "movieId") %>%
  left_join(b_ur, by = "userId") %>%
  mutate(residual = rating - mu - b_i - b_u) %>%
  select(userId, movieId, residual)
mf_train <- as.matrix(model_residual)
mf_test <- test_set %>% select(userId, movieId, rating)
mf_test <- as.matrix(mf_test)
```

The matrices are saved to disk for `recosystem` training. *(The computation in this code could take around more than 20 minutes to complete.)*

```
write.table(mf_train, file = "mf_train.txt", sep = " ", row.names = FALSE, col.names = FALSE)
write.table(mf_test, file = "mf_test.txt", sep = " ", row.names = FALSE, col.names = FALSE)
set.seed(2019, sample.kind = "Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
mftrain_set <- data_file("mf_train.txt")
mftest_set <- data_file("mf_test.txt")
r <- Reco()
opts <- r$tune(mftrain_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                        costp_l1 = 0, costq_l1 = 0,
                                        nthread = 1, niter = 10))
r$train(mftrain_set, opts = c(opts$min, nthread = 1, niter = 20))
predict_file <- tempfile()
r$predict(mftest_set, out_file(predict_file))
residuals_hat_mf <- scan(predict_file)
```

The predicted residuals are then added back to the base prediction to calculate the RMSE.

```r
y_hat_mf <- y_hat_r + residuals_hat_mf
rmse_mf <- RMSE(test_set$rating, y_hat_mf)
rmse_mf
```

```
## [1] 0.7906001
```

Adding the RMSE to our result table:

Table 4: RMSE Results

Model	Method	RMSE
1	Naive Approach	1.0611350
2	Movie Effect	0.9441568
3	Movie + User Effects	0.8659736
4	Movie + User + Time Effects	0.8658902
5	Movie + User + Genre Effects	0.8657599
6	Regularized Movie + User Effects	0.8654703
7	Matrix Factorization on Model 6	0.7906001

3.10 Choosing the final model

Our final model **Model 7** is essentially a model which:

- uses a baseline rating (average rating μ)
- accounts for movie effect (b_i) and user effect (b_u)
- regularized using penalized least square of $\lambda = 5$
- residuals are trained with matrix factorization

---

4. Result

We are now ready to evaluate our final model on the `validation` set to see how it performs. To be on the conservative side, we remain to use the `train_set` (which is a smaller set than `edx` set) to generate the model predictors and to train the residuals using matrix factorization.

```r
#matrix factorization on validation set
y_hat_v <- validation %>%
  left_join(b_ir, by = "movieId") %>%
  left_join(b_ur, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>% pull(pred)
mf_validation <- validation %>% select(userId, movieId, rating)
mf_validation <- as.matrix(mf_validation)
write.table(mf_validation, file = "mf_validation.txt",
            sep = " ", row.names = FALSE, col.names = FALSE)
mfvalidation_set <- data_file("mf_validation.txt")
predict_file_validation <- tempfile()
r$predict(mfvalidation_set, out_file(predict_file_validation))
residuals_hat_mf_validation <- scan(predict_file_validation)
```

```
y_hat_mf_validation <- y_hat_v + residuals_hat_mf_validation
rmse_mf_validation <- RMSE(validation$rating, y_hat_mf_validation)
```

The RMSE value of our final model using the validation set:

```
rmse_mf_validation
```

```
## [1] 0.7907142
```

---

5. Conclusion

5.1 Summary

We started the project with a large MovieLens dataset of movie ratings. We split the dataset into `edx` and `validation` set, and further split the `edx` into `train_set` and `test_set` before we begin our analysis. We explore the dataset by looking at possible effects from movies, users, timing (movie release year, and timing of rating) and genre. Then we build our model using these effects and we judge the model performance using RMSE value. We further enhance our model using regularization and matrix factorization. The final model is tested using the `validation` set and we achieve an RMSE value of 0.7907142.

5.2 Comments

One of the main challenges in creating a movie recommendation system lies with the variation and interaction between different effects. These interaction can be obscure and difficult to be quantified. For example those between movie and user, or between genre and time. This is also the reason matrix factorization has proven to be effective in improving our model. Some of the factors such as timing and genres can be further analysed to gauge their effect on the movie rating. Some other factors which are not covered may also be significant. For example the geographical location of user, budget of the film, or even the demographic of the leading actors. Advanced machine learning approaches such as regression and ensemble method could also be applied.

Nonetheless this project has helped me to put the knowledge and techniques learned throughout the course in practice. It has also inspired me of the possibility of expanding the model into other applications such as an Amazon Product Recommendation System.