Movie Lens Project

Valerie Cheong Took

`r.Sys.Date()`

Executive Summary

This project is about creating a recommendation engine to help movie viewers make their choices in terms of movies. The engine will use all the data available to make recommendations. For example, the purchasing history of the viewer if the latter is an existing client. It will recommend movies, similar to the ones already viewed by the viewer or movies viewed by other viewers with similar tastes or demographic profile.

The database Movielens 10M is used for this purpose. The objective is to create a recommendation engine taking into account the various biases such as time, genres, movies and users. The recommendation engine will be validated by splitting the Movielens dataset into a training set and a validation set. The final version of the recommendation engine will be tested using the validation set to ensure that this is the best version.

1.Introduction

A recommendation system is an automated engine that evaluates similarities between users and/or items (in this case, movies) in order to help users to make their choices.

1A. Types of Recommendation Engines

There are two main types of recommendation engines: **Content based recommendation system** uses the *characteristics of the product* to find other similar products. In the case of movies, it would be movies of the same genres or by the same producer or featuring the same actors or launched in the same year. Another characteristics of the product can be its *popularity* based on the number of viewers. **Collaborative Recommendation System** whereby the opinions of the users matter. It can be users having the same taste or socio demographic profile or finding movies that are similar to the ones that the users have already viewed. In this project, it would be finding movies that are of the same genres as the movies that the user has watched or recommending movies that users of similar tastes have watched.

A third type of recommendation engine would be a **hybrid** of the two main types. It is important to note here that this recommendation engine is based on explicit ratings, that is, ratings effected by users that have already watched the movies. Implicit ratings are not relevant in this project.

1B. Limitations Of Recommendation Engines

A good recommendation engine minimizes its limitations. It usually has 3 main limitations:
Long Tail Problem whereby there is a large variety of products and it is not possible to display all products for customers to see and examine. This is especially true for online businesses whereby customers are unable to try on or physically examine the products. **Cold Start** which can be user cold start or product cold start. *User Cold Start* takes place when a new user comes on board and has not yet purchased anything. The recommendation engine will not have any of the user's personal buying history to provide advice. *Product cold start* is when a new product is launched and there is no existing comparable products.
Huge Data Sparsity takes place when many users have not rated most of the products or movies available. So, in this case, there is a large number of zero ratings for many movies which is a huge hurdle for the recommendation engine.

2.Data Preparation

2A. Downloading the data and Data Wrangling

The relevant data is located as detailed below:
#Movielens 10M dataset #https://grouplens.org/datasets/movielens/10m/ #https://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"

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE), stringAsFac

colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")

ratings <- ratings %>%
  mutate(userId = as.integer(userId), movieId = as.integer(movieId), rating = as.numeric(rating), times

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

colnames(movies) <- c("movieId", "title", "genres")

movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

Movielens Dataset Exploratory Data Analysis

The data frame, movielens, has 6 variables. It contains ratings of 65,133 movies by 71,567 users. This makes movielens a huge matrix if we had to view all the data (65,133 * 71, 567). The ratings ranges from a minimum of 0.5 to a maximum of 5. In terms of genres, they can be a mix of different genres such as Action, Crime and Thriller for one single movie.

```
str(movielens)
```

```
## 'data.frame':    10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

```
knitr::kable(summary(movielens), caption = "Summary of the 'movielens' dataset")
```

Table 1: Summary of the 'movielens' dataset

	userId	movieId	rating	timestamp	title	genres
	Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:10000054	Length:10000054
	1st Qu.:18123	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character
	Median :35741	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
	Mean :35870	Mean : 4120	Mean :3.512	Mean :1.033e+09	NA	NA
	3rd Qu.:53608	3rd Qu.: 3624	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
	Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

```
knitr::kable(head(movielens), caption = "First Rows of the 'movielens' dataset")
```

Table 2: First Rows of the 'movielens' dataset

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

```
knitr::kable(tail(movielens), caption = "Last Rows of the 'movielens' dataset")
```

Table 3: Last Rows of the 'movielens' dataset

	userId	movieId	rating	timestamp	title	genres
10000049	71567	2028	5	912580345	Saving Private Ryan (1998)	Action|Drama|War
10000050	71567	2107	1	912580556	Halloween H20: 20 Years Later (1998)	Horror|Thriller
10000051	71567	2126	2	912649143	Snake Eyes (1998)	Action|Crime|Mystery|Thriller
10000052	71567	2294	5	912577968	Antz (1998)	Adventure|Animation|Children|Comedy|Fantasy
10000053	71567	2338	2	912578016	I Still Know What You Did Last Summer (1998)	Horror|Mystery|Thriller
10000054	71567	2384	2	912578173	Babe: Pig in the City (1998)	Children|Comedy

2B Setting up Training and Validation Set

The data frame, movie lens, is split into two sets: edx and final_holdout_test. The validation set is the final_holdout_test and accounts for 10% of movie lens dataset. This dataset is used only for validation. The edx dataset will have to be further split into 2 sets: a training set and a test set. This will be done after all data wrangling has been completed.

```
#Final holdout test set will be 10% of Movielens data
set.seed(1, sample.kind = "Rounding")

test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)

edx <- movielens[-test_index,]
temp <- movielens[test_index,]

#Make sure userId and movieId in final holdout 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 holdout test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

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

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

Exploratory Data Analysis of edx

```
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
##
##
##
```

edx dataset has 9,000,055 rows and columns. Fortunately, there were no zero ratings given as the minimum rating is 0.5. Rating 3 is considered to be a neutral rating. Out of a total of 9 million observations, rating 3 accounted for 2,121,240 observations representing 24%. A neutral rating is always difficult to interpret and it is fortunate that three quarters of the observations in the edx dataset represents interpretable ratings. This is a good basis for building a recommendation model. There are 10,677 different movies and 69,878 different users in the edx dataset.

```
#Rating 3
rating_3 <- edx %>%
  filter(rating == 3)
nrow(rating_3)
```

```
## [1] 2121240
```

```
#Distinct Movies and Distinct Users
edx %>% summarise(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

```
#Number of Movie Ratings Per Genres
genres <- c("Action", "Adventure", "Animation", "Children", "Comedy", "Fantasy", "IMAX", "Sci-Fi", "Dra

genres_edx <- data.frame(Genres = genres, count = sapply(genres, function(x){
  sum(str_detect(edx$genres, x))
})
)

view(genres_edx)

order(genres_edx[,"count"])
```

```
##  [1]  7 18 19 16 17  3 15 11 10  4  6 14  8 12  2 13  1  5  9
```

From the genres_edx dataset, it is noted that movies of the genres, Drama, was the most rated (3,910,127 ratings) followed by Comedy (3,540,930 ratings) and Action.The movies of the Thriller genres received 2,325,899 ratings and the Romance genres received 1,712,100 ratings.

```
#Movie with the greatest number of ratings
edx_movies <- edx %>%
  group_by(title) %>%
  summarise(count = n()) %>%
  top_n(10, count) %>%
  arrange(desc(count))
view(edx_movies)

#Five Most Given Ratings
edx_ratings <- edx %>%
  group_by(rating) %>%
  summarise(count = n()) %>%
  arrange(desc(count))
edx_ratings
```

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

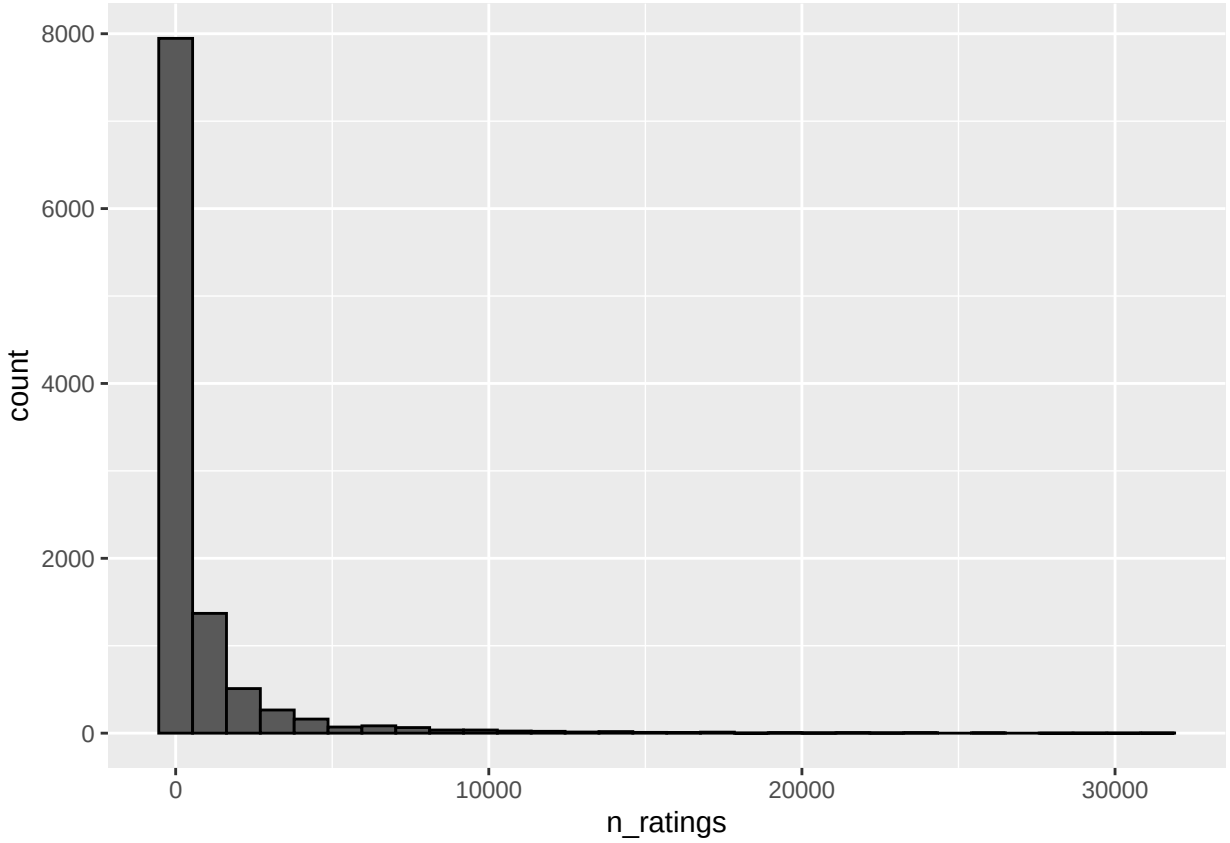
Pulp Fiction received the greatest number of ratings, 31,362 followed by Forrest Gump and Silence of the Lamb. Ratings 4,3 and 5 followed by 3.5 and 2 were the ones used most often by the users to rate movies. It is reassuring to know that in this case, users expressed their opinions wholeheartedly as ratings 4 and 5 implies that they have enjoyed watching the movies. Also, users tend to use whole star ratings more often than half star ratings.

Deep Dive Analysis

Further analysis is carried out to better understand the data structure such as number of ratings for every movie. It is obvious that a few movies are rated more than others and it is safe to assume that they are the blockbuster movies.

```
#Number of Ratings for every movie
ratings_per_movie <-edx %>%
  group_by(movieId) %>%
  summarise(n_ratings = n())

#Histogram
ggplot(ratings_per_movie, aes(x = n_ratings)) +
  geom_histogram(bins = 30, color = "black") +
  theme_gray()
```



It would be useful to know the number of ratings that each user gives. To this effect, we do the following calculation. As expected, most users have rated a few movies as most people are not generally movie buffs. The histogram is, therefore, skewed to the right.

```
#Number of Ratings For Every User
ratings_per_user <- edx %>%
  group_by(userId) %>%
  summarise(count = n())

#Histogram
ggplot(ratings_per_user, aes(x = count)) +
  geom_histogram(bins = 30, color = "blue") +
```

```
  scale_x_log10() +
  theme_linedraw()
```

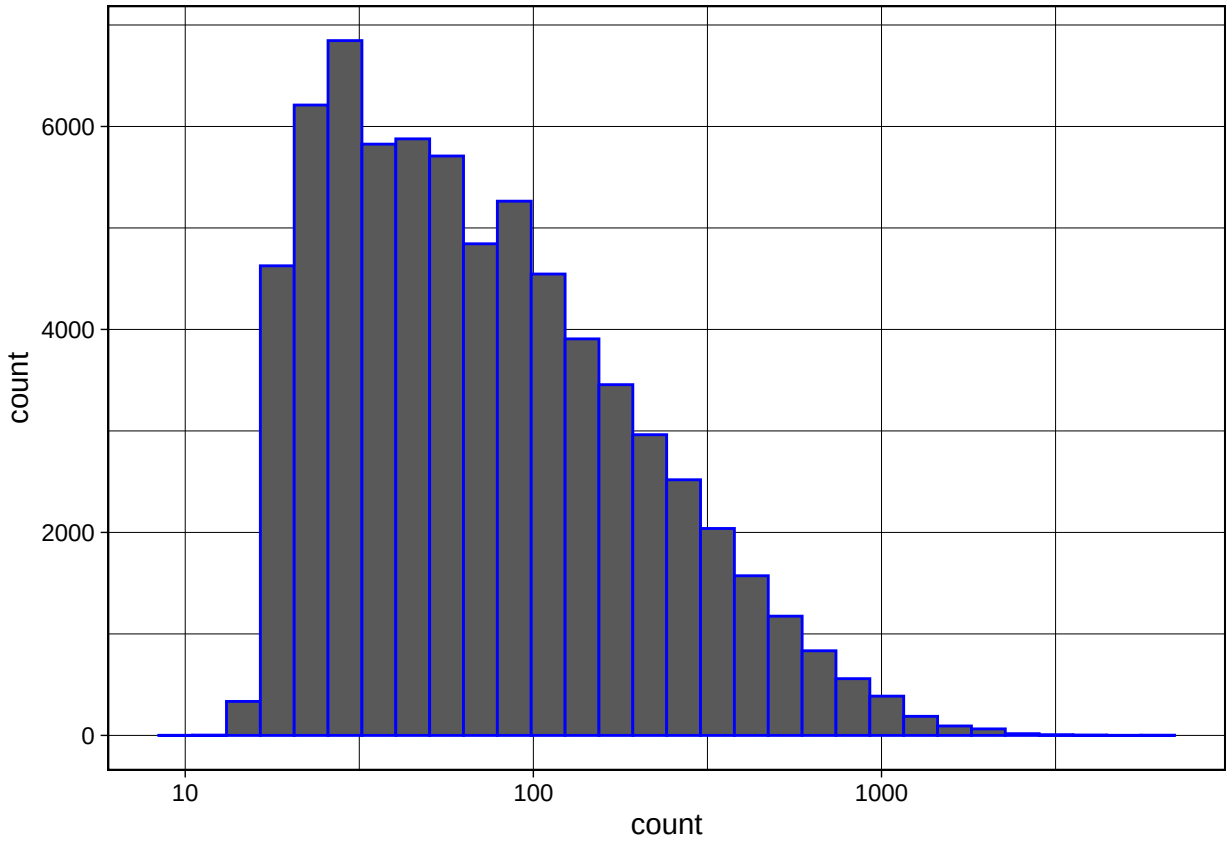


Figure 1: Number Of Ratings For Every User.

Data Wrangling of edx and final_holdout_test datasets
It is necessary to do some more data wrangling so as to prepare the data for creating the model and testing it. The variable title includes the name of the movie as well as the year of the release. It is necessary to separate the year of the release from the name of the movie.

Creating year.release column

```
#edx dataset
year.released.date <- as.numeric(str_sub(edx$title, start = -5, end = -2))

edx <- edx %>%
  mutate(edx, year.release = year.released.date)


##final_holdout_test dataset
year.released.date.validation <- as.numeric(str_sub(final_holdout_test$title, start = -5, end = -2))

final_holdout_test <- final_holdout_test %>%
  mutate(final_holdout_test, year.release = year.released.date.validation)
```

Creating Year of Review Column
The second step is to convert the timestamp into a date format so as to know the year of review. A new column, date, is added to both datasets.

```
#edx Dataset
#converting timestamp into date
edx <- edx %>%
  mutate(edx, date = as_datetime(timestamp))

#changing date format
edx <- edx %>%
  mutate(date = round_date(date, unit = "week"))


#final_holdout_test dataset
#converting timestamp into date
final_holdout_test <- final_holdout_test %>%
  mutate(final_holdout_test, date = as_datetime(timestamp))

#changing date format
final_holdout_test <- final_holdout_test %>%
  mutate(date = round_date(date, unit = "week"))
```

Adding Duration Column
All the movies in both datasets have not been released in the same year nor have been rated at the same time. It is important to know whether a movie released earlier has more ratings than a movie released at a later date. We now have to add another column in each dataset, duration which is the timespan between the date of review and this present year, 2025

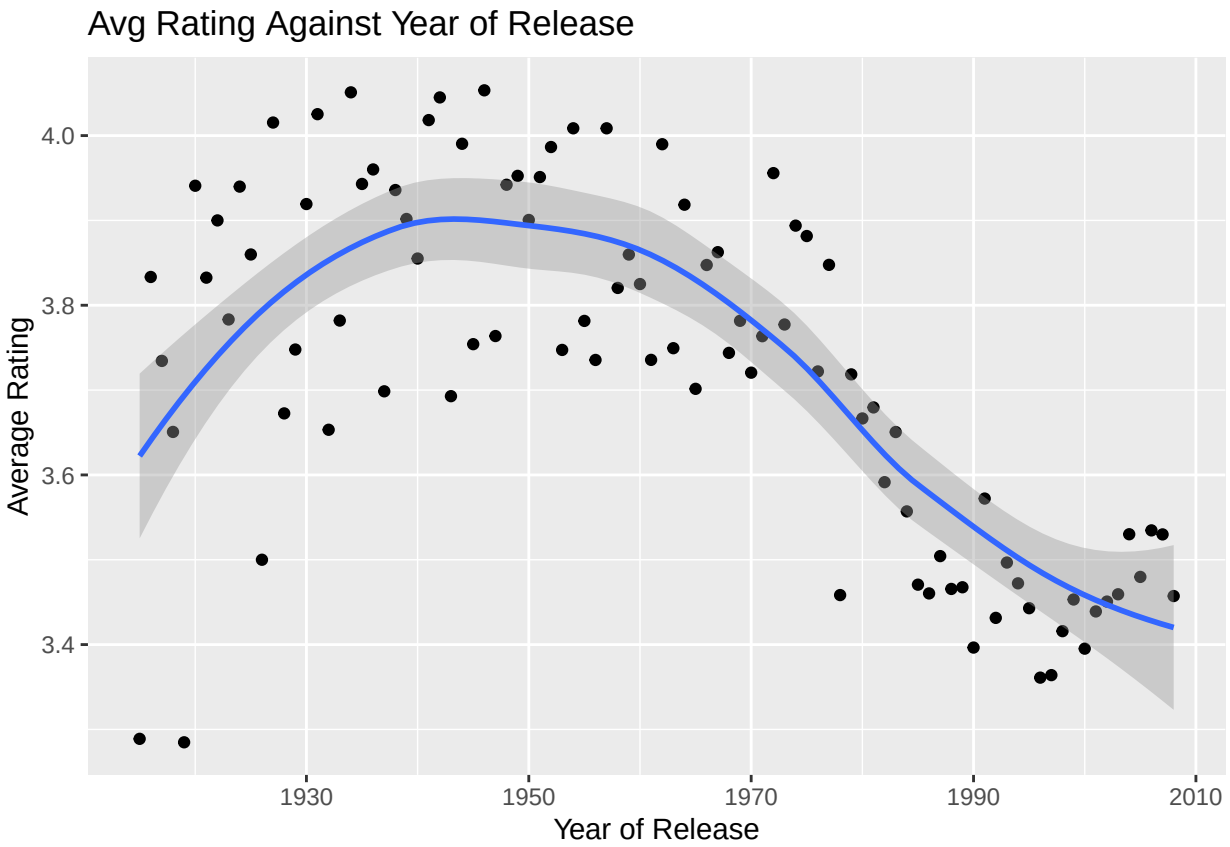
```
#edx dataset
edx <- edx %>%
  mutate(duration = 2025 - year.release)


#final_holdout_test dataset
final_holdout_test <- final_holdout_test %>%
  mutate(duration = 2025 - year.release)
```

It would be interesting to know whether a movie released earlier has a higher average rating than a movie released later. A movie released earlier would logically have been viewed by many more people, it may have been rated highly and over many years.

```
edx %>%
  group_by(year.release) %>%
  summarise(avg.rating = mean(rating)) %>%
  ggplot(aes(year.release, avg.rating)) +
  geom_point() +
  geom_smooth() +
  labs(x = "Year of Release", y = "Average Rating", title = "Avg Rating Against Year of Release")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



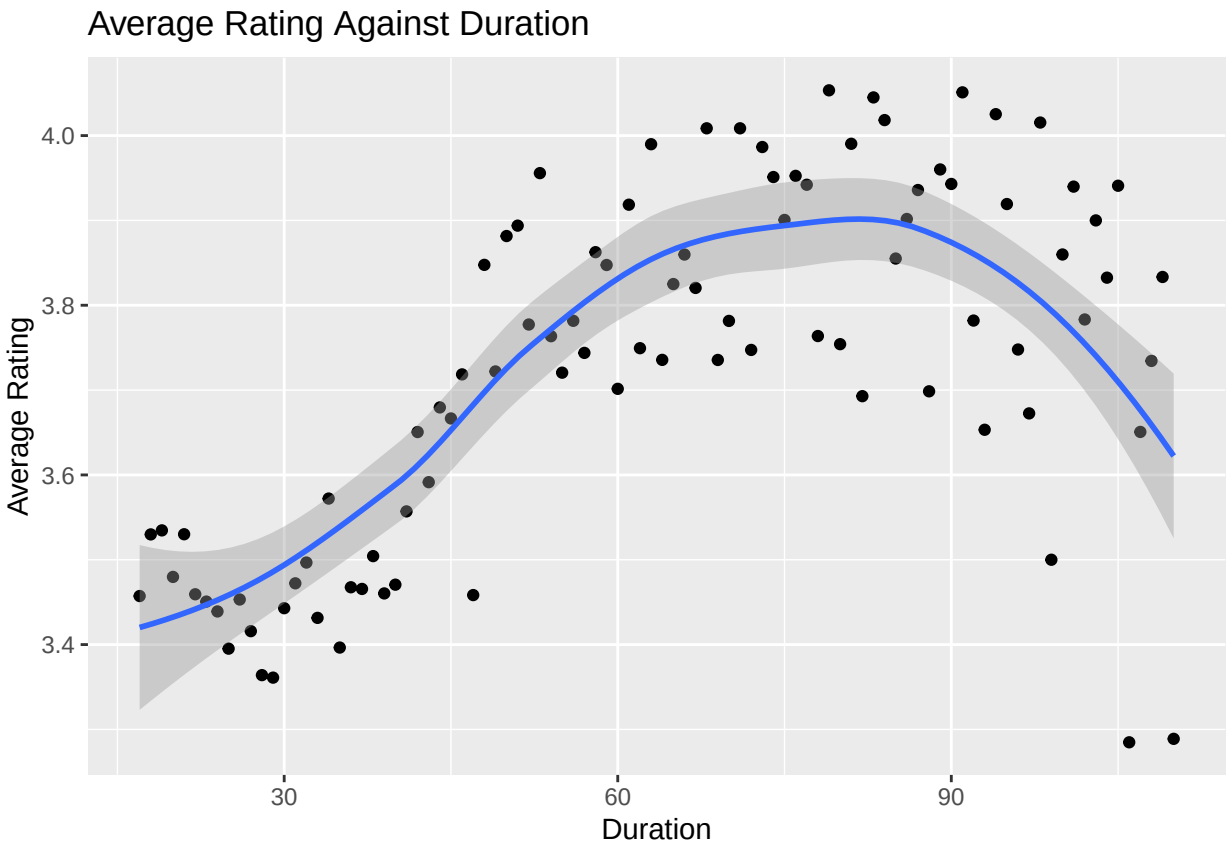
From the plot, it is interesting to note that movies released during the period 1940 - 1950 recorded a higher average rating. Movies released earlier or later recorded a lower average rating. Additional analysis is carried out to verify this finding. A closer look is taken at the relationship between duration and its average rating.

MOVIE EFFECT

Relationship Between Duration and Average Rating

```
edx %>%
  group_by(duration) %>%
  summarise(average.rating = mean(rating)) %>%
  ggplot(aes(duration, average.rating)) +
  geom_point() +
  geom_smooth() +
  labs(x = "Duration", y = "Average Rating", title = "Average Rating Against Duration")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



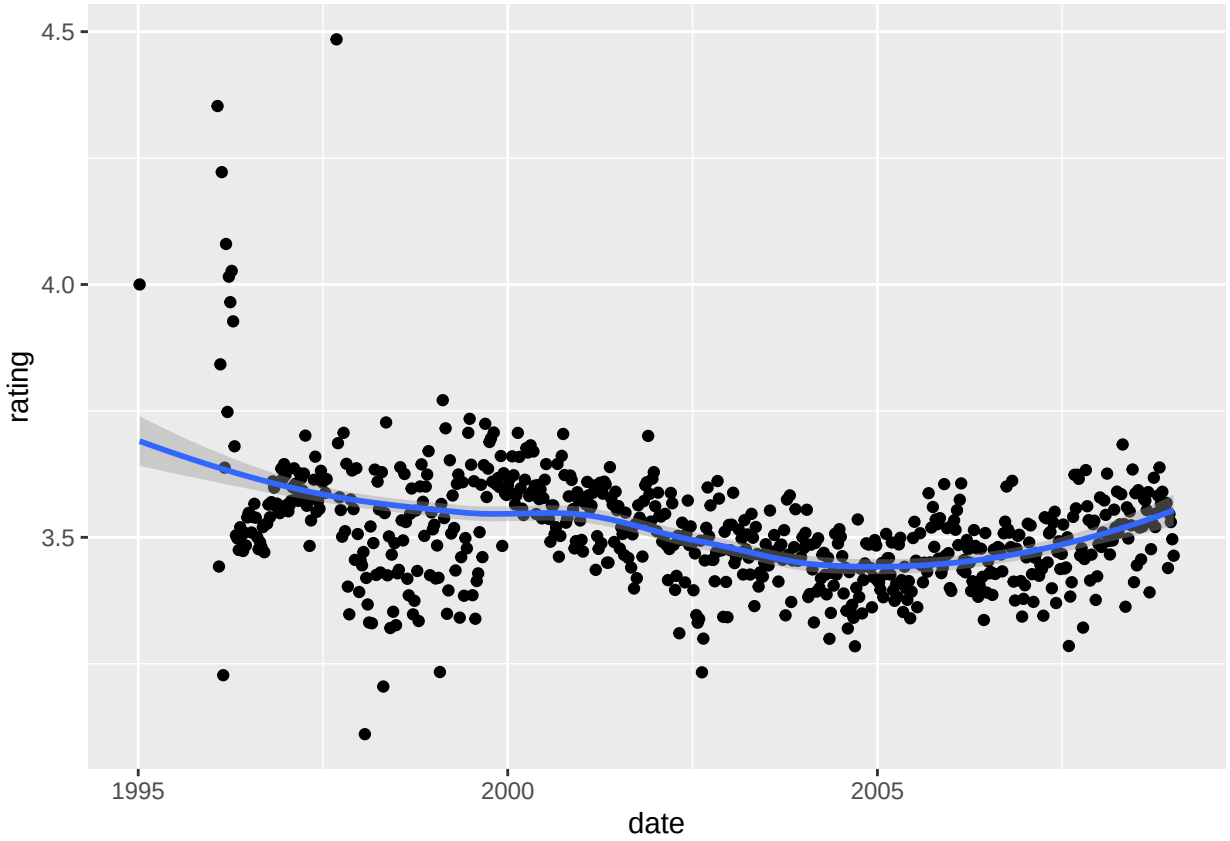
Based on the graph, it is confirmed that movies released in the 1940's and 1950's are hugely popular and are rated the most often. Thus, there is a **movie effect** that we need to take into consideration when building the recommendation model.

TIME EFFECT

Does that mean that time has no effect on average rating? The following graph will give us a clue.

```
edx %>%
  mutate(date = round_date(date, unit = "week")) %>%
  group_by(date) %>%
  summarise(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()
```

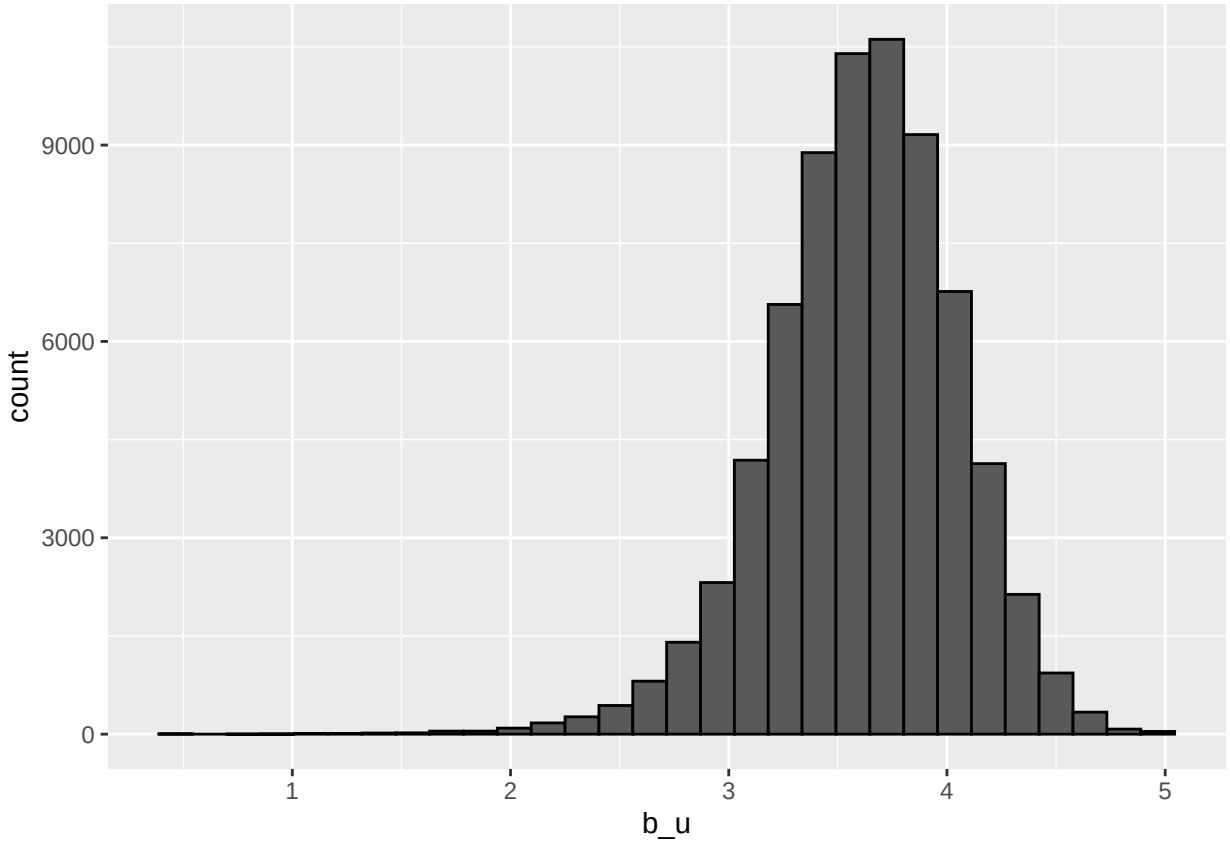
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



From the graph, it can be inferred that there is evidence of a time effect on average rating. This will need to be taken into account when building the recommendation model. Also, it is logical to infer that not all users rate the same way or have the same taste. This is verified by the following graph.

USERS EFFECT

```
#Average rating for user,u
edx %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```

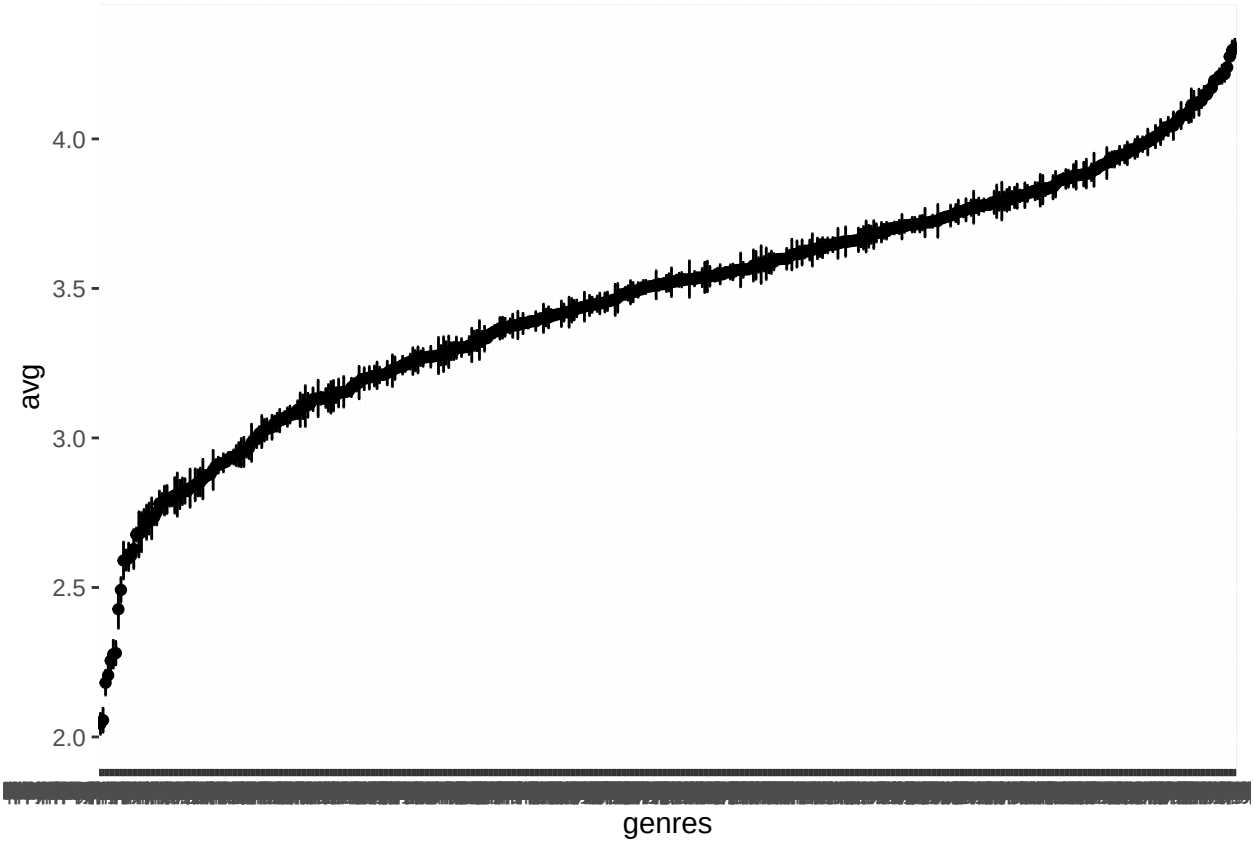


From the histogram, it can be noted that there is substantial variability across users in terms of their rating habits. This need to be kept in mind, when the recommendation engine is being built as there is a user specific effect. It is also important to note that the histogram is skewed to the left.

GENRES EFFECT

There may be also a genre effect. It is important to verify this assumption before starting the design of the recommendation model.

```
edx %>%
  group_by(genres) %>%
  summarise(n = n(), avg = mean(rating), se = sd(rating/sqrt(n()))) %>%
  filter(n >= 1000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2 * se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.x.text = element_text(angle = 90, hjust = 1))
```



The plot shows strong evidence of a genres effect which need to be accounted for in the recommendation model.

Splitting edx dataset

We now prepare the edx dataset for creating the recommendation model. We split the dataset into a training set accounting for 80% of the dataset and a test set with the remaining 20%.

```
set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_train <- edx[-test_index,]
edx_test <- edx[test_index,]

#Make sure userId and movieId in test set are also in train set
edx_test.final <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")

#Add rows removed from test set back into train set
removed <- anti_join(edx_test, edx_test.final)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres,
## year.release, date, duration)`
```

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

rm(test_index, removed, edx_test)
```

3.Modelling

The Netflix challenge used the typical error loss: they decided on a winner based on the residual mean squared error (RMSE) on a test set.

Definition of RMSE The RMSE is defined as follows:

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

The RMSE will be the measure of accuracy of the recommendation model. Its interpretation is similar to that of the standard deviation. In other words, we need to minimise its value without affecting the quality of the recommendation model.

3A.First Model

The first model is the simplest recommendation model, that is, the same rating is predicted for all movies regardless of the user. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this: $Y_(u,i)=\mu + \epsilon\_(u,i)$

The RMSE calculated from this equation is called the *naive RMSE*

```
mu_hat <- mean(edx_train$rating)

naive_RMSE <- RMSE(edx_test.final$rating, mu_hat)

#Creating a results table with this naive approach
rmse_results <- tibble(method = "Just the average", RMSE = naive_RMSE)
rmse_results
```

The *naive RMSE* stands at 1.059904. However, earlier analysis has demonstrated that there are 4 other biases to be taken into account in the equation: movie effect, user effect, time effect and a genre effect. We now have to include each bias one by one.

3B. Second Model - Movie Effect

The equation will now look like this $Y_(u,i)=\mu + b_(i) + \epsilon\_(u,i)$

The new RMSE is calculated as follows:

```
mu <- mean(edx_train$rating)
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))

predicted_ratings <- mu + edx_test.final %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)

rmse_movie.effect <-RMSE(predicted_ratings, edx_test.final$rating)
rmse_movie.effect
```

The RMSE with the movie effect has declined to 0.9437. We now include a second bias, the *user effect*.

3C. Third Model - Movie Effect and User Effect

With the addition of a third bias, the new equation is as follows: $Y_(u, i)=\mu + b_(i) + b_(u) + \epsilon_(u,i)$ with $b_(u)$ representing the user effect.

Tne new RMSE integrating the user effect is now calculated.

```
user_avgs <- edx_train %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating -mu -b_i))

predicted_ratings <- edx_test.final %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

RMSE_user.movie.effect <- RMSE(predicted_ratings, edx_test.final$rating)
RMSE_user.movie.effect
```

The RMSE now stands at 0.8659 having included two predictors in its calculation.

3D. Fourth Model - Movie Effect, User Effect and Genres Effect

With the addition of the genres effect, the equation will change to: $Y_(u, i)=\mu + b_(i) + b_(u) + b_(g) + \epsilon_(u,i)$ with $b_(g)$ representing the genres effect

```
#Fourth Model adding the genres effect
genre_avgs <- edx_train %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarise(b_g = mean(rating -mu -b_i - b_u))

predicted_ratings <- edx_test.final %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

RMSE_genre.user.movie.effect <- RMSE(predicted_ratings, edx_test.final$rating)
RMSE_genre.user.movie.effect
```

Adding the genres effect, the RMSE has slightly decreased to 0.8656. We now add the last bias, the time effect

3E. Fifth Model, adding the time effect

In regards to the time effect, we have calculated duration which is the timespan from the year of release of the movie to this present year, 2025. We are now going to use duration to include the time effect in the model. Time effect is $b_(t)$ in the new equation. $Y_(u, i)=\mu + b_(i) + b_(u) + b_(g) + b_(t) +\epsilon_(u,i)$

```
#Fifth Model with the time effect added
time_avgs <- edx_train %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  group_by(duration) %>%
  summarise(b_t = mean(rating - mu - b_i - b_u - b_g))

predicted_ratings <- edx_test.final %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(time_avgs, by = "duration") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  pull(pred)

RMSE_time.genre.user.movie.effect <- RMSE(predicted_ratings, edx_test.final$rating)
RMSE_time.genre.user.movie.effect
```

RMSE is calculated to be 0.8654, having included all 4 biases.

4. Regularization

We now apply regularization to the model so as to further improve it.

```
lambdas <- seq(0,10,0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(edx_train$rating)

  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n() + 1))

  b_u <- edx_train %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarise(b_u = sum(rating - mu - b_i)/(n() + 1))

  b_g <- edx_train %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarise(b_g = sum(rating - mu - b_i - b_u)/(n() + 1))

  b_t <- edx_train %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(duration) %>%
    summarise(b_t = sum(rating - mu - b_i - b_u - b_g)/(n() + 1))

  predicted_ratings <- edx_test.final %>%
```

```
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_t, by = "duration") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
    pull(pred)

  return(RMSE(predicted_ratings, edx_test.final$rating))
})


#Identifying Optimal Tuning Parameter
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 0
```

The final value of the RMSE with regularisation is . The optimal tuning parameter is `lambda`. The value of lambda will now be used to do the final validation using the validation set as explained in the next section.

5. Validating the Final Model

We now use the optimal value of the tuning parameter to validate the final model by using the *edx dataset* as training set and the *final_hold_out dataset* as the test set.

```
mu <- mean(edx$rating)

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

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

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

b_t <- edx %>%
  left_join(b_i , by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  group_by(duration) %>%
  summarise(b_t = sum(rating - b_i - b_u - b_g)/(n() + lambda))

predicted_ratings <- final_holdout_test %>%
  left_join(b_i, by = "movieId") %>%
```

```
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_t, by = "duration") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_t) %>%
  pull(pred)

final_rmse <- RMSE(final_holdout_test$rating, predicted_ratings)
```

The final RMSE validated is `final_rmse`.

6. Concluding Remarks

With Regularization, the RMSE has definitely improved. Using `lambda`, the optimal tuning parameter, we obtain `final_rmse` as the validated and final value. The recommendation engine could be further improved if we use the Random Forest methodology which is appropriate for large datasets such as Movielens and where there is more than one level of variation such as movies with different actors, movies of different genres, movies released in different years.

7. References

1. Introduction to Data Science by Rafael A. Irizarry
2. R Graphics Cookbook, 2nd Edition by Winston Chang
3. Movie and TV Show Recommendation Engine in R by "GeeksforGeeks.org"
4. Movie Recommendation System using R by "Kaggle"
5. A Practical Guide to Data Analysis Using R: An Example Based Approach by John H Maindonald, W. John Braun, Jeffrey L Andrews
6. An Introduction to Statiscal Learnings: with Applications in R by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani