# Movie Recommendation System

Predicting Movie Ratings

Nimisha Agrawal

January 1, 2021

**Abstract**

This project seeks to implement two movie recommendation systems to predict the rating of a movie by a user. The first model uses a handcrafted hybrid method that models user-movie interactions and movies' attributes. The second model uses a collaborative filtering method via matrix factorization as done by the recosystem package. The report starts with an introduction to the basics of recommender systems. Next is a description of the data set followed by the menthodology and the procedure for data wrangling and pre-processing. Some exploratory data analysis has been done to understand the dataset and draw helpful insights. Then the evolution of our machine learning models has been described in detail. Finally, the results (including RMSE values) and the conclusion is presented, with a future outlook.

## Contents

# Introduction to Recommender Systems

Before I begin the technical description of recommender systems, consider this: in October 2006, Netflix offered **a million dollars** to anybody who could improve their recommendation system by 10%.[1] Okay, now let's delve in.

Recommender systems essentially suggest relevant items to users. This is beneficial to both the users and the businesses. With a fitting recommender system, users get a personalized experience tailored to their preferences and interactions. For businesses, such personalisation enhances user engagement and consequently revenue. Win-win!
Technically speaking, recommender systems can be divided into three categories:[2]

- Content-based Methods: Movie attributes are used as inputs and movies similar to the user's past preferences are recommended.
- Collaborative Filtering Methods: Based on past interactions between users and movies, a user is recommended movies depending on the preferences of other similar users. (In our recosystem method of matrix factorization, only user-movie interactions are used to model the system.)
- Hybrid Methods: Combining content-based and collaborative filtering methods. (In our handcrafted method, we model both movies' attributes and user-movie interactions.)

Most modern systems use a hybrid model since they incorporate more attributes and consequently account for more variability, thus reducing RMSE which is our ultimate goal. (Computational time is another story.)

[1] https://www.netflixprize.com/

[2] Suggested readings: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada and https://lionbridge.ai/articles/step-by-step-guide-to-building-a-movie-recommendation-system/

## The Data Set

In this project, the Movie Lens 10M Dataset[3] is used. Movielens is an online 'non-commercial, personalized' movie recommender service. The main Movielens data set[4] has 27 million ratings. The one used here is a randomly selected subset containing about 10 million ratings for 10,681 movies by 71,567 users of Movielens. Of the three files in the dataset, we use only *ratings.dat* and *movies.dat*. We ignore tags.dat to reduce computational complexity.

[3]Thanks to Rich Davies for generating the data set.

[4]F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI = http://dx.doi.org/10.1145/2827872

## Methodology

I use R[5] with the RStudio IDE[6] to perform data wrangling, pre-processing, exploratory analysis and machine learning. This report is generated using R Markdown with RStudio, in a Tufte Handout format.
To implement the project, the following packages are used in addition to base R:

```
#PACKAGES FOR THE REPORT:
library(rmarkdown) #converting R markdown documents into several formats
library(knitr) #a general-purpose package for dynamic report generation
library(kableExtra) #nice table generator
library(tinytex) #for compiling from .Rmd to .pdf

#PACKAGES FOR THE CODE:
library(tidyverse) #for data processing and analysis
library(caret) #for splitting data for machine learning
library(data.table) #for data wrangling
library(lubridate) #for dealing with date-time attributes
library(ggcorrplot) #for plotting the correlation matrix
library(randomcoloR) #to generate a discrete color palette
library(ggridges) #for making ridges density plots
library(recosystem) #for implementing matrix factorization
```

[5]R is free and open source. You can download it here: https://cran.r-project.org/

[6]RStudio has many useful features apart from the editor. You can download it here: https://rstudio.com/products/rstudio/download/

## Data Wrangling

Data wrangling is the process of converting raw data into tidy form. The database is downloaded into a temporary file. The delimited[7] files *ratings.dat* and *movies.dat* are read and converted into data frames.

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

#.dat files have :: as column separators
#gsub replaces all :: with a tab so that the data can be easily read into a table
#with the friendly fread
ratings <- fread(text = gsub("::", "\t",
                             readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

#reading in movies data and making a data frame
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")),"\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

#if using R 3.6 or earlier:
#movies <- as.data.frame(movies) %>%
#  mutate(
#    movieId = as.numeric(levels(movieId))[movieId],
#    title = as.character(title),
#    genres = as.character(genres))

#if using R 4.0 or later:
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(movieId),
         title = as.character(title),
         genres = as.character(genres))
```

The movie information is joined to the rating information for the corresponding movies resulting in the final *movielens* data set that we will split into test and training sets.

```
#to keep only the movies with corresponding entries in ratings and combine the
#tables' columns, ratings data is left joined to movie data, resulting into a new
#'movielens' data set
movielens <- left_join(ratings, movies, by = "movieId")
```

[7]Data in the files provided is separated using :: as delimiter.

## Pre-processing

The *movielens* data set created after wrangling is first split into test (*edx*) and training (*temp*) sets using the *createDataPartition* function of the *caret* package. The sets are then modified to ensure that the final test set (*validation*) only has users and movies which are present in the training set.

```
#SPLITTING THE DATA

#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)`

test_index <- createDataPartition(
  y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,] #training set
temp <- movielens[test_index,] #a temporary validation set

#to make sure userId and movieId in validation set are also in edx set,
#semi-join returns rows from the temporary validation set that have a match in
#the training set 'edx'
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

#adding rows removed from the temporary validation set back into the edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

Some additional columns are created in the training set which extract information from existing columns and make it more usable for machine learning. Columns thereby rendered redundant are removed. This has also been done correspondingly in the validation set to allow for smooth testing.

```
#PRE-PROCESSING THE EDX DATA SET

#extracting day, month and year of rating from timestamp,
#extracting movie release year from the title, and
#subtracting year of rating from the movie release year to get years lapsed
#also, removing title and timestamp columns since all info extracted from them
edx <- edx %>%
  mutate(month_rating = month(as_datetime(timestamp)),
         day_rating = as.numeric(factor(weekdays(as_datetime(timestamp)),
                                        levels = c("Monday", "Tuesday", "Wednesday",
                                                   "Thursday", "Friday", "Saturday",
                                                   "Sunday"))),
         year_movie = as.numeric(str_sub(title,-5,-2)),
         years_lapsed = year(as_datetime(timestamp))-year_movie) %>%
  select(-c("title", "timestamp"))

#to extract genre data, getting each genre in separate row
edx_genre_separated <- edx %>% separate_rows(genres, sep="\\|")

#getting the average rating and count stats for every genre
genres_stats <- edx_genre_separated %>%
```

```
  group_by(genres) %>%
  summarise(avg_rating=mean(rating), n=n())

#to add genre characteristic rating, adding a column with the maximum of the
#average ratings of the genres of the movie (rounded)
edx <- edx_genre_separated %>%
  group_by(userId, movieId) %>%
  left_join(genres_stats) %>%
  summarize(genres_rating=round(max(avg_rating),1)) %>%
  right_join(edx) %>%
  ungroup() %>%
  select(-genres) #genres column not needed anymore
```

```
#PRE-PROCESSING VALIDATION DATA IN ACCORDANCE WITH EDX DATA
#the columns that were added in edx are added here too.
#columns removed in edx are removed here too
validation <- validation %>%
  mutate(month_rating = month(as_datetime(timestamp)),
         day_rating = as.numeric(factor(weekdays(as_datetime(timestamp)),
                                        levels=c("Monday", "Tuesday", "Wednesday",
                                                 "Thursday", "Friday", "Saturday",
                                                 "Sunday"))),
         year_movie = as.numeric(str_sub(title,-5,-2)),
         years_lapsed = year(as_datetime(timestamp))-year_movie) %>%
  select(-c("title", "timestamp"))

#extracting the genres by separating rows and adding on the genres_rating stats
#using the avg_rating from the previously created genre_stats data set
#**The validation ratings have NOT been used in any manner here.
validation <- validation %>%
  separate_rows(genres, sep="\\|") %>%
  group_by(userId, movieId) %>%
  left_join(genres_stats) %>%
  summarize(genres_rating=round(max(avg_rating),1)) %>%
  right_join(validation) %>%
  ungroup() %>%
  select(-genres)
```

# Exploratory Data Analysis

## Basic Analysis

To begin our exploration of the *edx* training set, here's a table of what the column names mean:

| Column Name | Explanation |
|---|---|
| userId | Unique ID for the user |
| movieId | Unique ID for the movie |
| genres_rating | Maximum of the average ratings of the genres of the movie |
| rating | A rating between 0 and 5 for the movie in increments of 0.5 |
| month_rating | The month in which the rating was given where 1 is January, 2 is February and so on. |
| day_rating | The day of the week in which the rating was given where 1 is Monday, 2 is Tuesday and so on. |
| year_movie | The year in which the movie released. |
| years_lapsed | The year gap between movie released and rating given. |

Now, let's look at the basic structure of the *edx* data set:

```
str(edx)
```

```
## tibble [9,000,055 x 8] (S3: tbl_df/tbl/data.frame)
##  $ userId       : int [1:9000055] 1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId      : num [1:9000055] 122 185 292 316 329 355 356 362 364 370 ...
##  $ genres_rating: num [1:9000055] 3.6 3.7 3.7 3.5 3.7 3.5 3.8 3.6 3.7 3.4 ...
##  $ rating       : num [1:9000055] 5 5 5 5 5 5 5 5 5 5 ...
##  $ month_rating : num [1:9000055] 8 8 8 8 8 8 8 8 8 8 ...
##  $ day_rating   : num [1:9000055] 5 5 5 5 5 5 5 5 5 5 ...
##  $ year_movie   : num [1:9000055] 1992 1995 1995 1994 1994 ...
##  $ years_lapsed : num [1:9000055] 4 1 1 2 2 2 2 2 2 2 ...
```

```
summary(edx)
```

```
##      userId         movieId      genres_rating       rating
##  Min.   :    1   Min.   :    1   Min.   :3.300   Min.   :0.500
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.500   1st Qu.:3.000
##  Median :35738   Median : 1834   Median :3.700   Median :4.000
##  Mean   :35870   Mean   : 4122   Mean   :3.621   Mean   :3.512
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:3.700   3rd Qu.:4.000
##  Max.   :71567   Max.   :65133   Max.   :4.000   Max.   :5.000
##   month_rating      day_rating     year_movie    years_lapsed
##  Min.   : 1.000   Min.   :1.00   Min.   :1915   Min.   :-2.00
##  1st Qu.: 4.000   1st Qu.:2.00   1st Qu.:1987   1st Qu.: 2.00
##  Median : 7.000   Median :4.00   Median :1994   Median : 7.00
##  Mean   : 6.786   Mean   :3.86   Mean   :1990   Mean   :11.98
##  3rd Qu.:10.000   3rd Qu.:6.00   3rd Qu.:1998   3rd Qu.:16.00
##  Max.   :12.000   Max.   :7.00   Max.   :2008   Max.   :93.00
```

Here's a sneak peek of the first few rows of the *edx* data set:

Table 2: First 6 Rows of edx (train set)

| userId | movieId | genres_rating | rating | month_rating | day_rating | year_movie | years_lapsed |
|---|---|---|---|---|---|---|---|
| 1 | 122 | 3.6 | 5 | 8 | 5 | 1992 | 4 |
| 1 | 185 | 3.7 | 5 | 8 | 5 | 1995 | 1 |
| 1 | 292 | 3.7 | 5 | 8 | 5 | 1995 | 1 |
| 1 | 316 | 3.5 | 5 | 8 | 5 | 1994 | 2 |
| 1 | 329 | 3.7 | 5 | 8 | 5 | 1994 | 2 |
| 1 | 355 | 3.5 | 5 | 8 | 5 | 1994 | 2 |

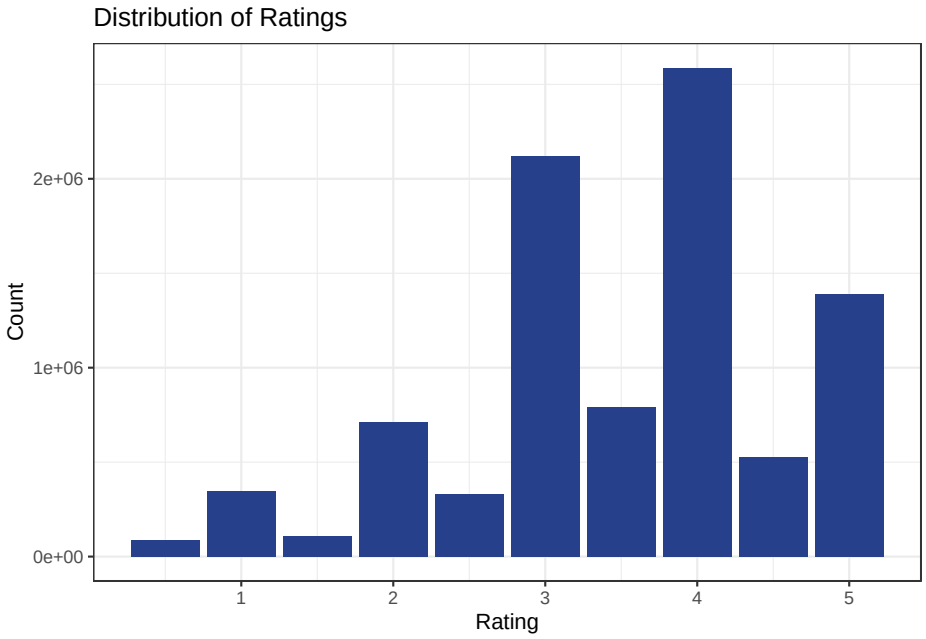
There are 10677 unique movies and 69878 unique users who have rated movies.

Now let's go through a few visuals[8] to better understand the data and maybe find something interesting?

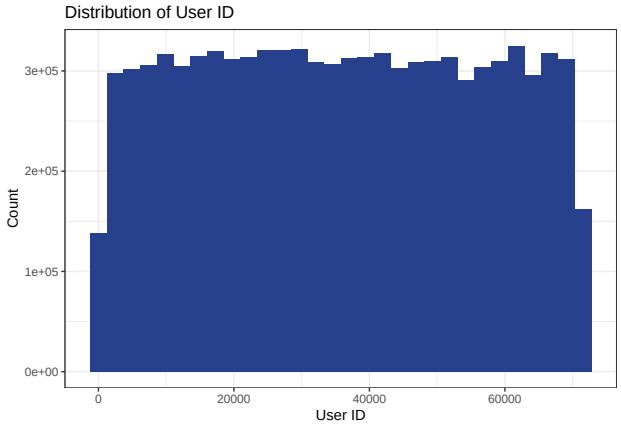
[8]The plots have been constructed using the *ggplot2* package.

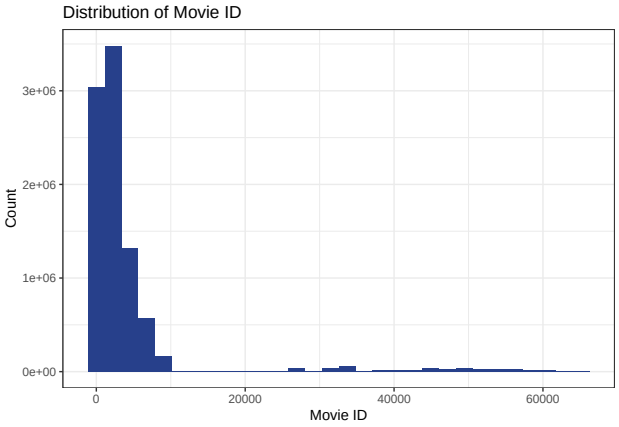
## Standalone Variable Analysis

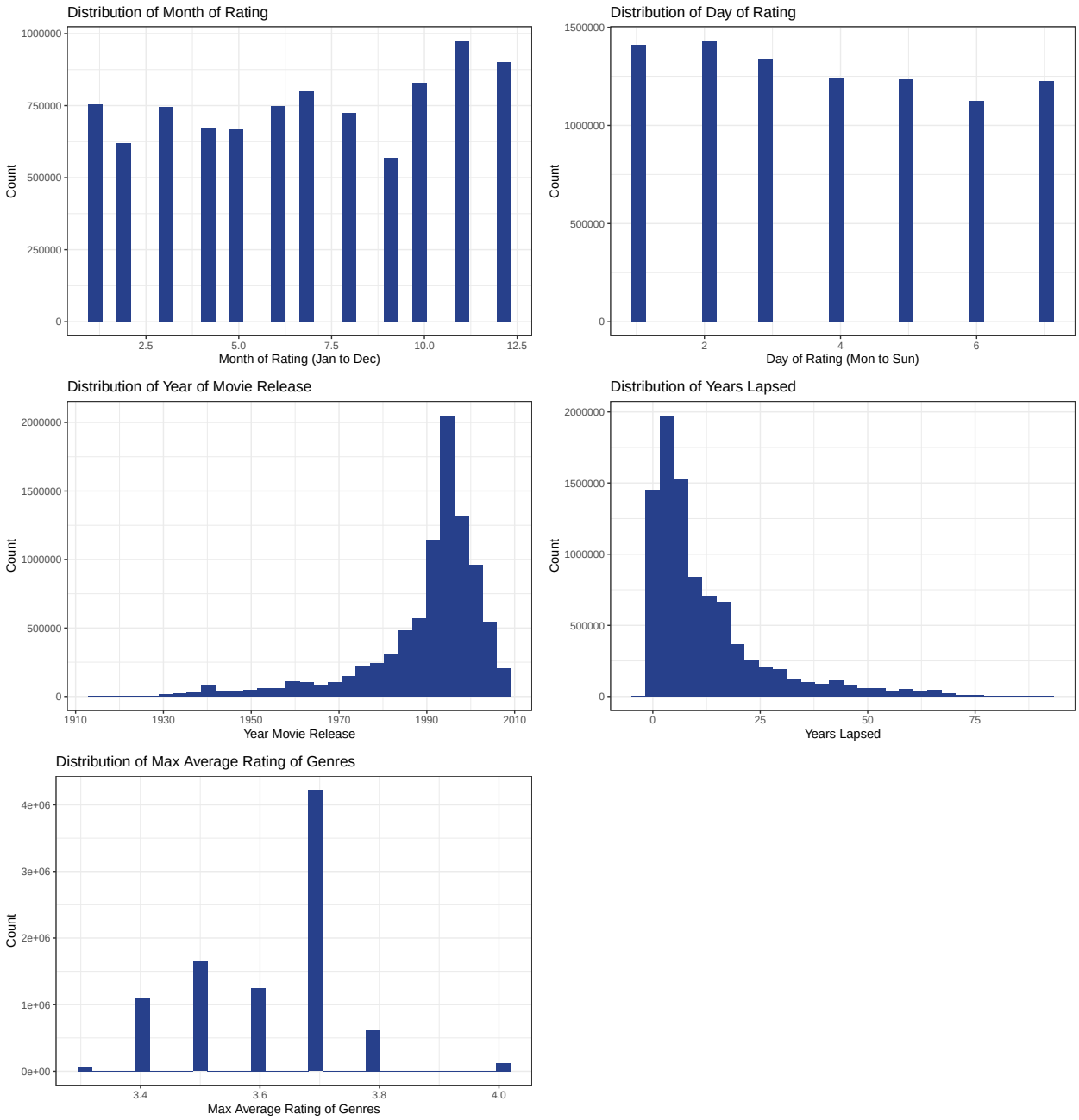
First, the different variables are analyzed standalone.




It is immediately obvious that whole number ratings are much more common than the half number ones. Because of this disparity, I will round the ratings to whole numbers in future analyses. It is also worth noting that 4 star ratings are the most common and that users rarely give the lowest rating. Guess we do love all movies then?



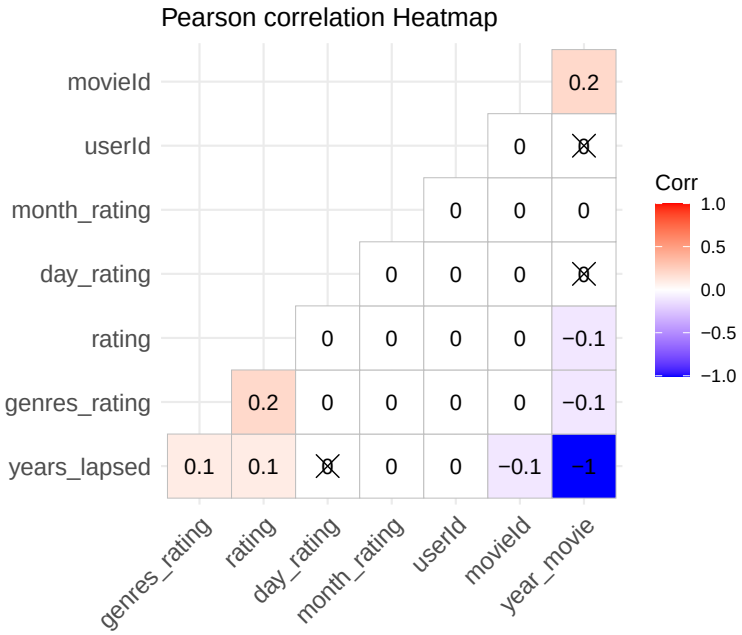






Firstly, contrary to what I think, lesser movies are rated on weekends. Second, people have rated and watched more movies in November-December (probably because a lot of time is spent indoors for the northern hemisphere people in winters?). Third, the distribution of movieId shows that majority of the people have rated a few movies (probably the blockbusters). Most other movie ratings are scarce. Fourth, all users have rated about the same number of movies. And finally, most of the movies in our dataset are relatively new and have been rated without much delay.

## Correlation Analysis

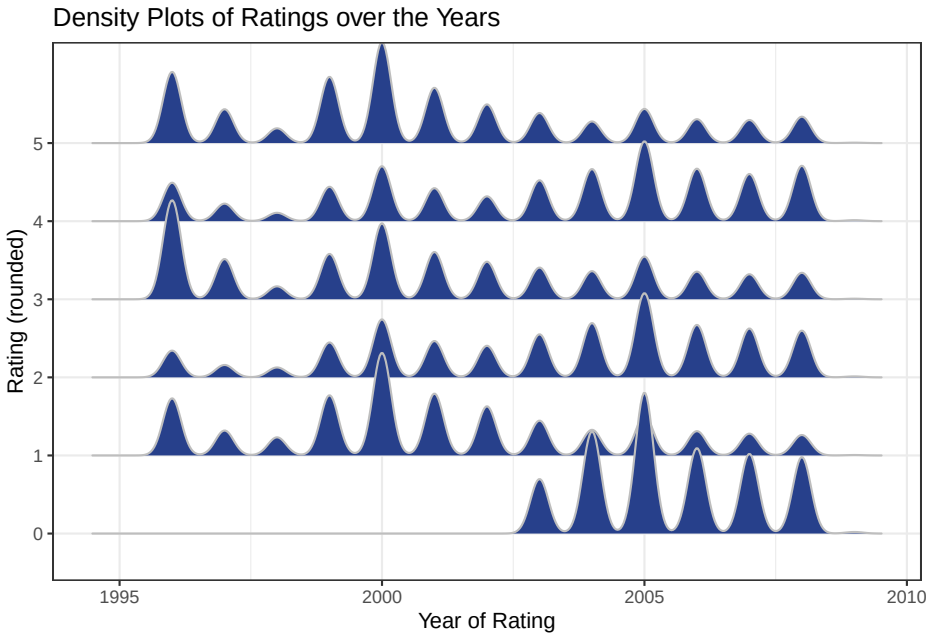
Next, the variables are checked for correlations.



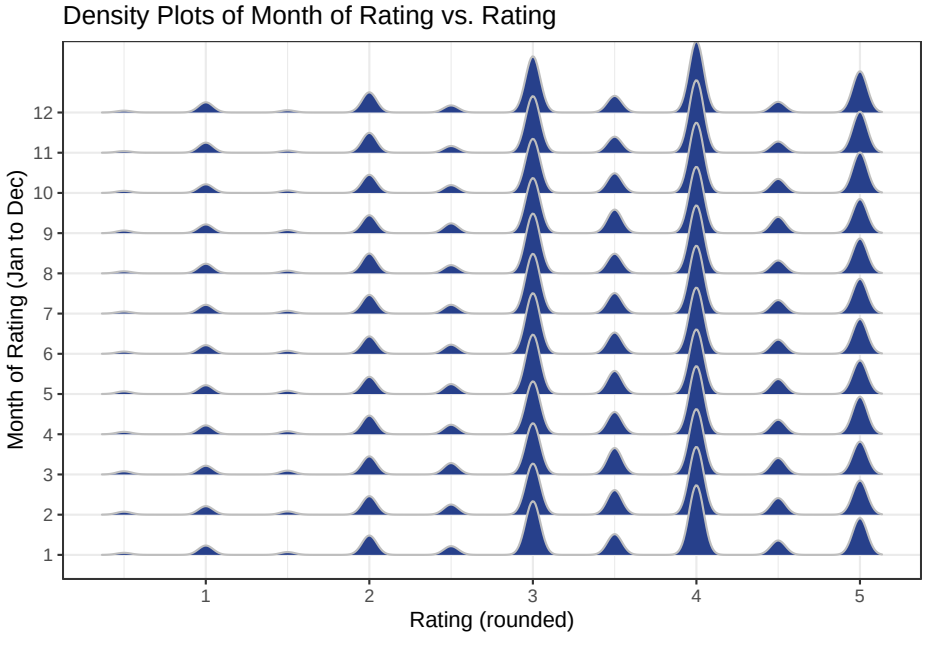

The X indicates that the p-value of the correlation is insignificant (95% C.I.). The correlations are negligible, except where a variable is explicitly used to compute another (like year_movie for years_lapsed and rating for genres_rating).

## Outcome (rating) Analysis with Features

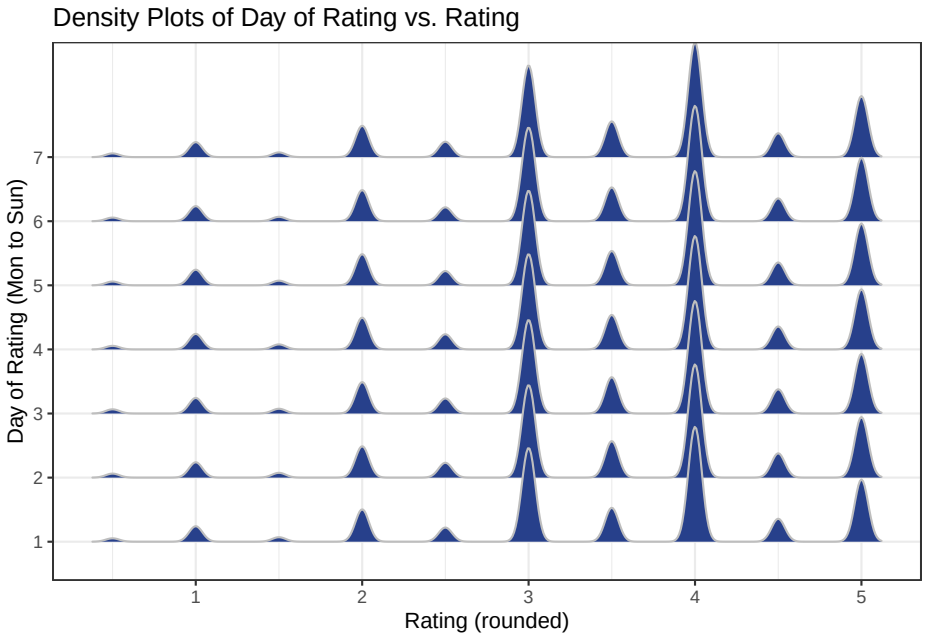
What follows is an exhaustive analysis of the outcome *rating* with the features.



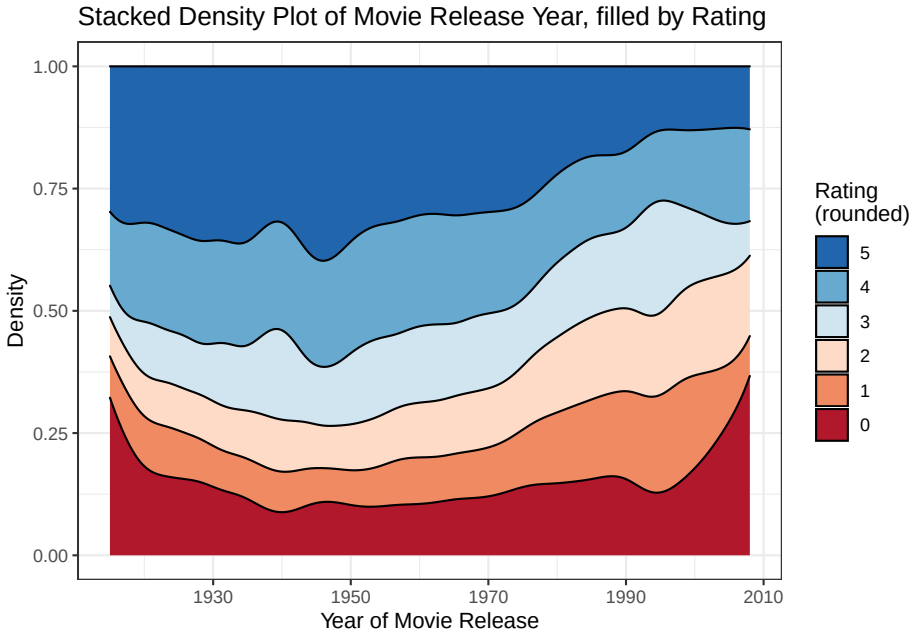

From the plot, two things are obvious. First, the lowest ratings have only come up in recent times. (So people were mostly satisfied with what they had in the old days?) Second, while the proportion of highest ratings (5) has decreased with time, that of medium-high ratings (4) has increased. (It's easier to get people to like you than to love you?)



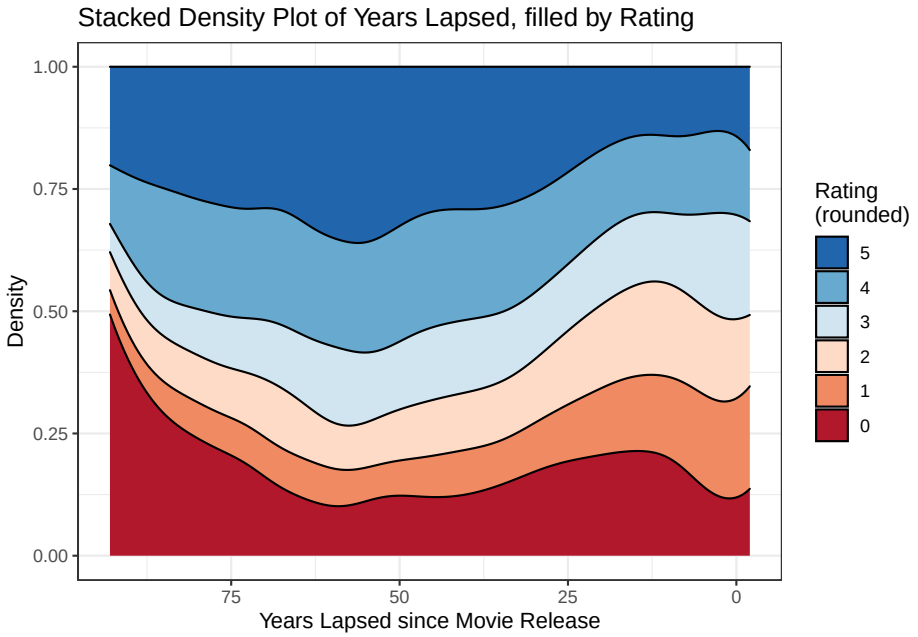

The distribution of ratings is roughly the same every month and follows the overall trend of the rating distributions.




The distribution of ratings is roughly the same every day of the week and follows the overall trend of the rating distributions.




It is seen that a higher proportion of older movies were given the highest rating as compared to newer movies. The lowest ratings witnessed a dip from the 1930s to the 2000s. (Probably because older movies are scarcely watched and that too only by the willing, and they either love it or hate it.)

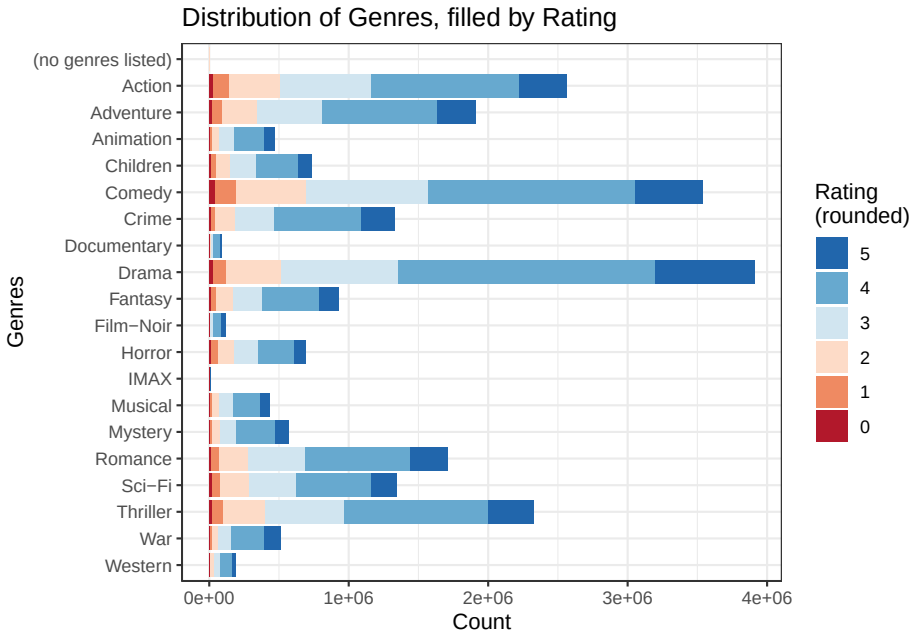



Keeping with the preceding explanation, it is seen that when people watch movies after a long time lapse, they either love it (highest rating) or hate it (lowest rating) with hardly any intermediates.

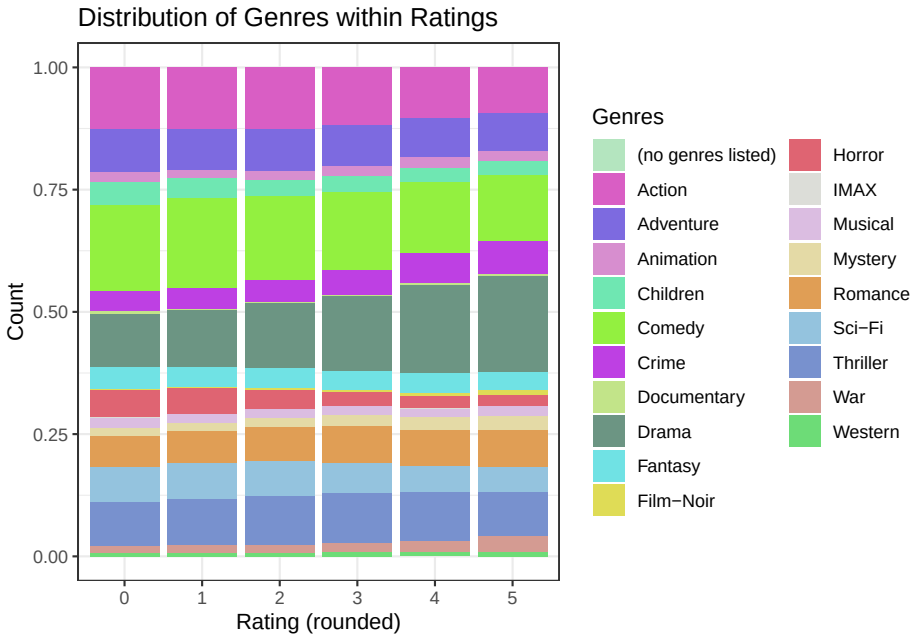
## Bonus: Genre Analysis

Presented below is a fun genre analysis purely for the purpose of exploration.

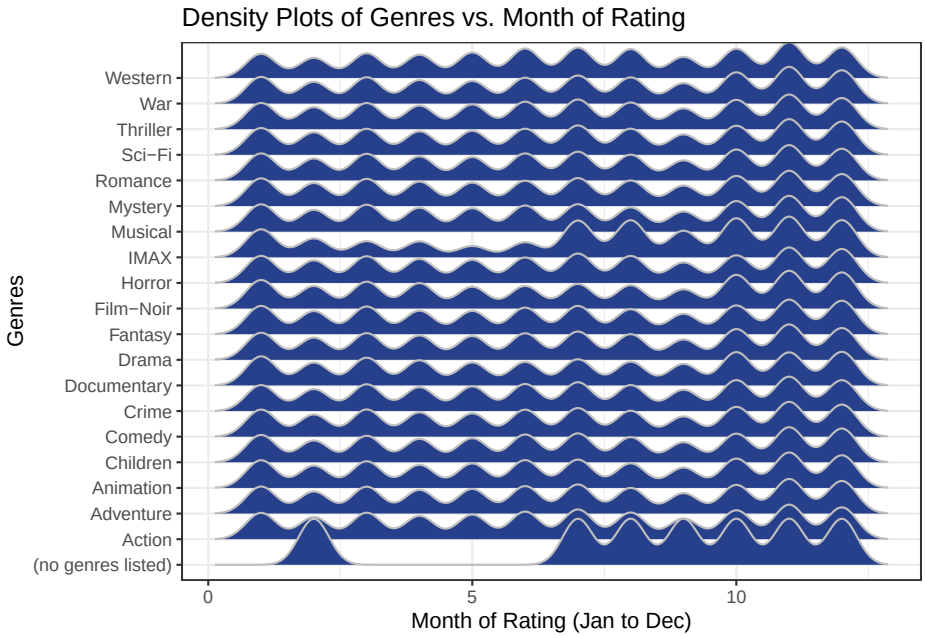
There are 19 distinct genres in the training set. There are also 7 entries for which no genre has been mentioned.



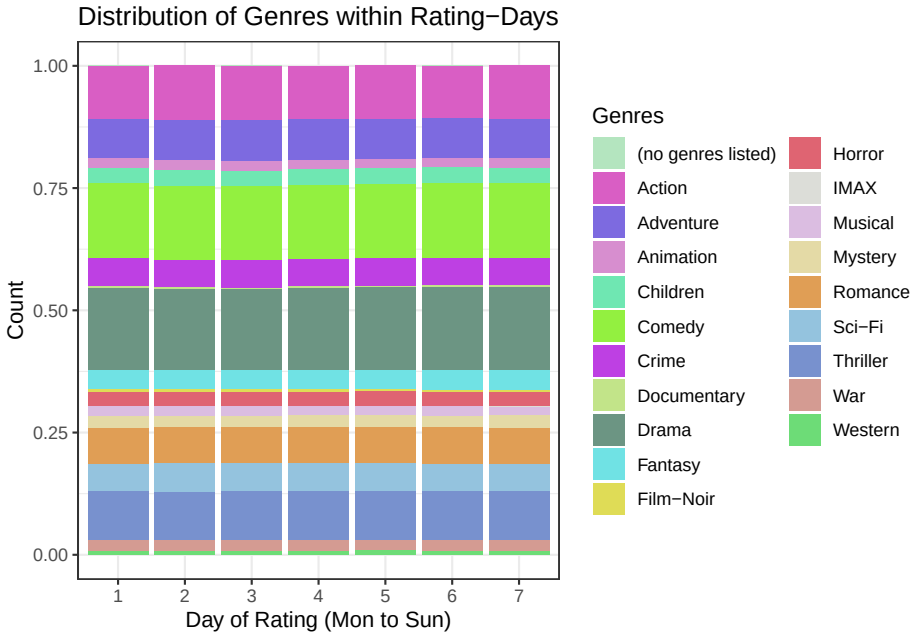

Drama is the most common genre, followed by comedy, action and thriller. The distribution of ratings within genres appears uniform across genres.



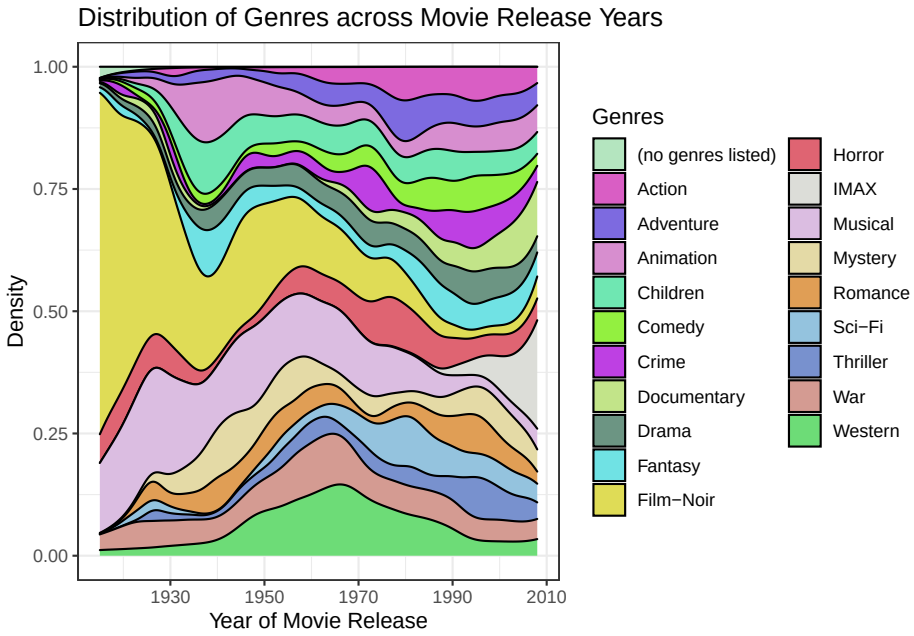

There is a higher proportion of drama movies amongst the highest rated movies. Correspondingly, they have a lower proportion of action movies. the other genres are uniformly distributed within ratings.



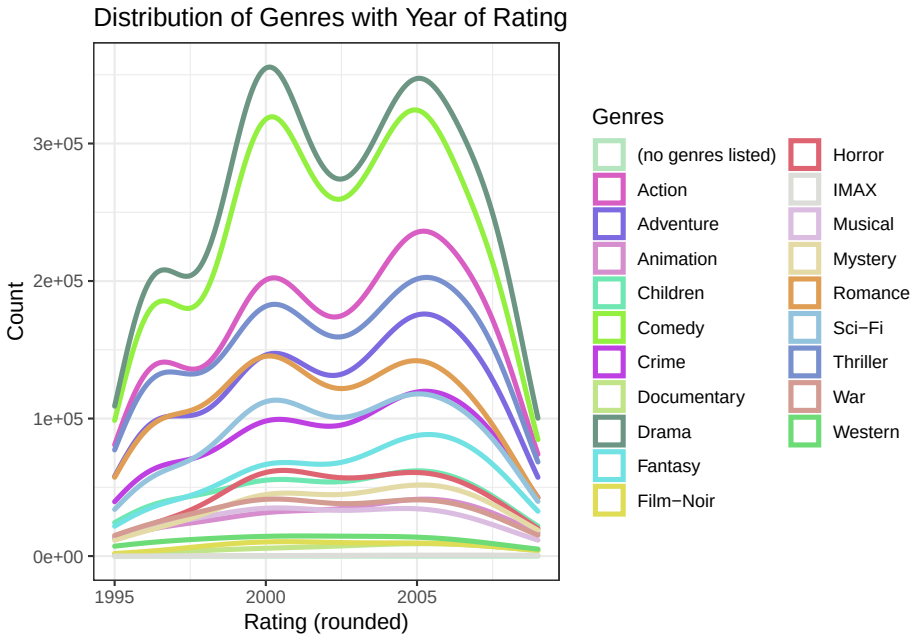

The distribution of genres across months is uniform, with the exception of IMAX ratings being low during February-June. (There are very few IMAX ratings, so that could just be random variability.)



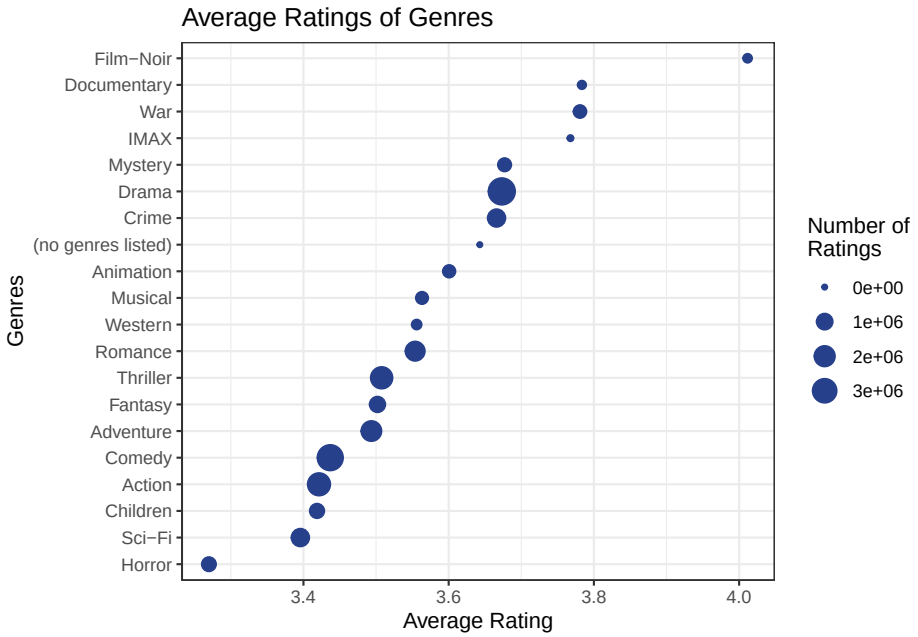

The distribution of genres within days of the week is as uniform as can be.




Film noirs have shown the most dramatic decrease, which is expected since these movies were mostly prevalent in the post-World War II era. IMAX is new technology and has thus propped up very late in the plot. Musicals have witnessed a decline in proportion, although not as dramatic as film noirs. The bright side is that all genres seem to have even distribution in the latest releases.




Drama and comedy movies witnessed two big surges in 2000 and 2005. In the same years, the other genres also saw surges, but to a lesser extent. Since this is a plot of count and each movie has multiple genres, it may be the case that the same movies are driving the surges in different genres.




Film noir has the highest average rating but the number of ratings is quite less. This is likely because users selectively watch such movies because they are classic favourites. Horror movies, unsurprisingly, have the lowest average ratings. All this proves that genre must have an effect of the movie ratings and this will likely be significant in the machine learning algorithms.

## Developing the Recommender System

In this project, I develop two main models:

1. The first one is a handcrafted hybrid method that models user-movie interactions and movies' attributes. This is done in several intermediate steps, adding one feature at a time.

2. The second model uses a collaborative filtering method via matrix factorization as done by the *recosystem* package.

All of the models are developed using only the test set (*edx*) and are tested on the *validation* set.
The Root Mean Square Error/Deviation (RMSE/RMSD)[9] is used as the sole metric used to judge these models. It is the standard deviation of the prediction errors.

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y_{u,i}} - {y_{u,i}}^2)}$$

$N$: number of user-movie combinations
$y_{u,i}$: actual rating for movie $i$ by user $u$
$\hat{y_{u,i}}$: predicted rating for movie $i$ by user $u$

The following code is used to make a function to compute RMSE for every model:

```
#Defining RMSE:
actual_rating <- validation$rating
RMSE <- function(predicted_rating){
  sqrt(mean((actual_rating - predicted_rating)^2))
}
```

[9]To know more about RMSE and why it's useful, I suggest reading https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e

# Handcrafted Hybrid Model

## Elementary Model

In this model, the average of all ratings is predicted as the rating for all movies, irrespective of the user and the movie.

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

$Y_{u,i}$: actual rating for movie $i$ by user $u$
$\mu$: average of all ratings (predicted "true" rating)
$\epsilon_{u,i}$: independent errors centered at 0

The model is implemented in R as follows:

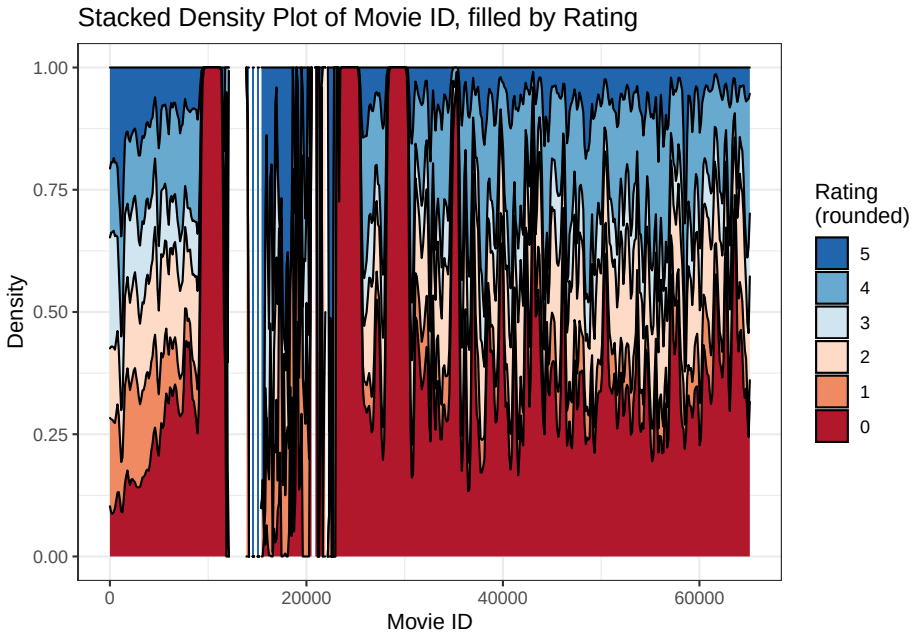
```
#1. Elementary model (using mean)
mu <- mean(edx$rating)

#although this is redundant here, I do it anyway for uniformity across models
predicted_ratings_model1 <- validation %>%
  mutate(pred = mu) %>%
  pull(pred)

#RMSE values for each model are stored for comparison
model1_rmse <- RMSE(predicted_ratings_model1)

#Printing the RMSE
model1_rmse
```

```
## [1] 1.061202
```

The RMSE is quite high, as expected. So, in subsequent steps, we add parameters to model different features so as to account for as much variability as possible.

## Modeling Movie Effects

It is known that some movies are rated higher than others. If you'd like proof anyway, the messy plot below shows the variability of ratings for different mvoieIds.




The previous model is thus augmented by adding the term $b_i$ to represent average rating for movie $i$.

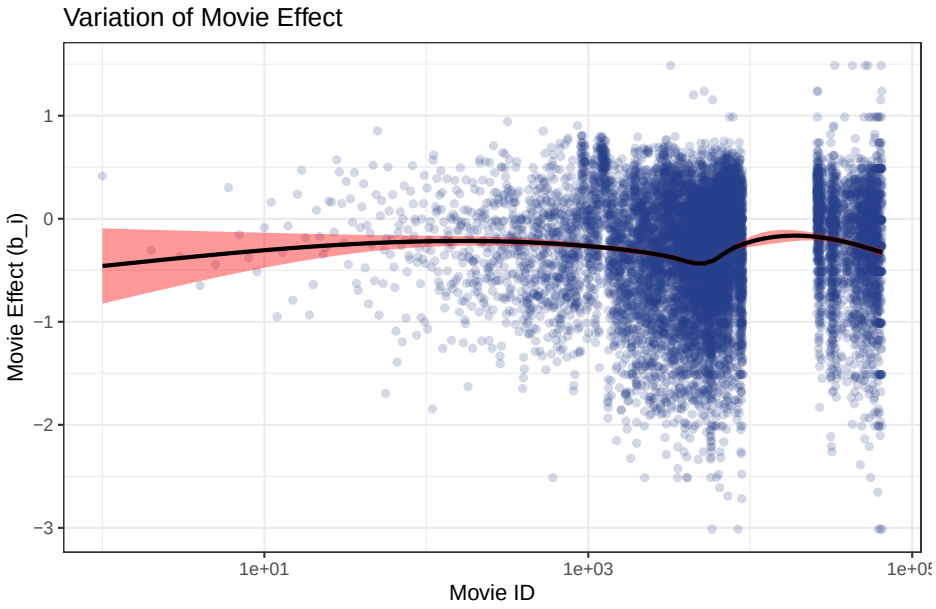
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

The model is implemented as follows:

```
#2. Modeling movie effect:
movie_effect <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating-mu))

#predicting ratings for the validation set
predicted_ratings_model2 <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

model2_rmse <- RMSE(predicted_ratings_model2)
model2_rmse
```

```
## [1] 0.9439087
```

The RMSE has improved by 11.0528597%, but there's still a long way to go.



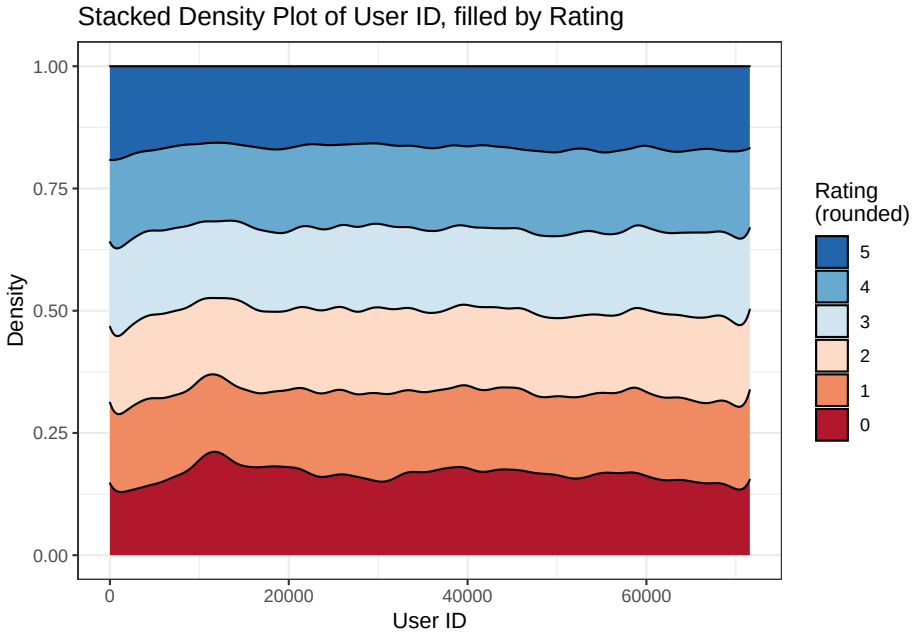

Movies affect ratings irregularly.

## Modeling User Effects

Users exhibit different rating behaviours. To account for this, the term $b_u$ is added to represent user-specific effects for user $u$.

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

The user rating behaviour doesn't seem as variable as the movie effect, but it'll definitely help improve the model.

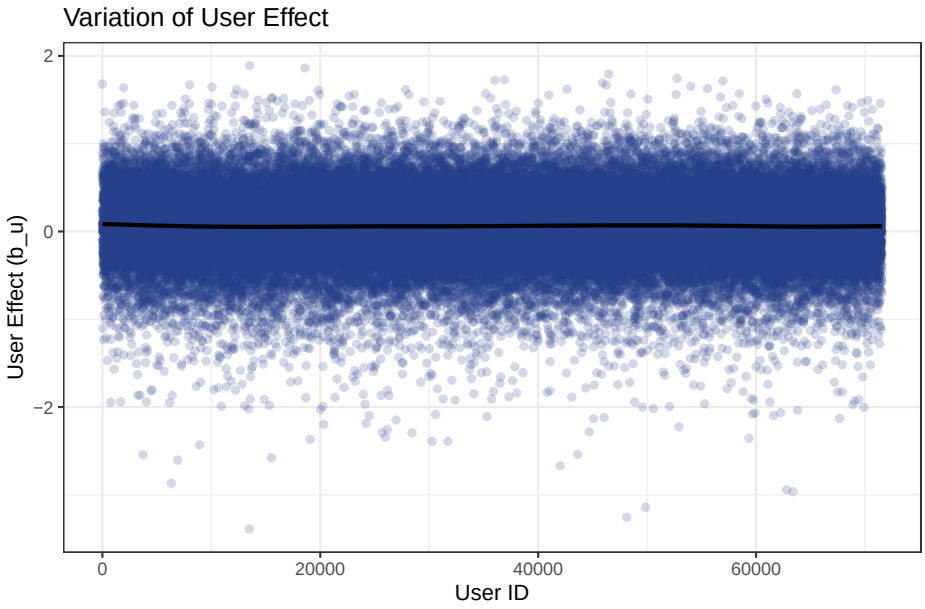



The model is implemented as follows:

```
#3. Modeling user effect:
user_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating-mu-b_i))

#predicting ratings for the validation set
predicted_ratings_model3 <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  mutate(pred = mu + b_i + b_u ) %>%
  pull(pred)

model3_rmse <- RMSE(predicted_ratings_model3)
model3_rmse
```

```
## [1] 0.8653488
```

The RMSE has improved further by 8.322822%.




Users affect ratings irregularly.

## Modeling 'Month of Rating' Effect

To account for the variation in ratings with the month of the year when rating is given, the term $b_{mr}$ is added to represent month-specific effects for the month of rating $mr$.

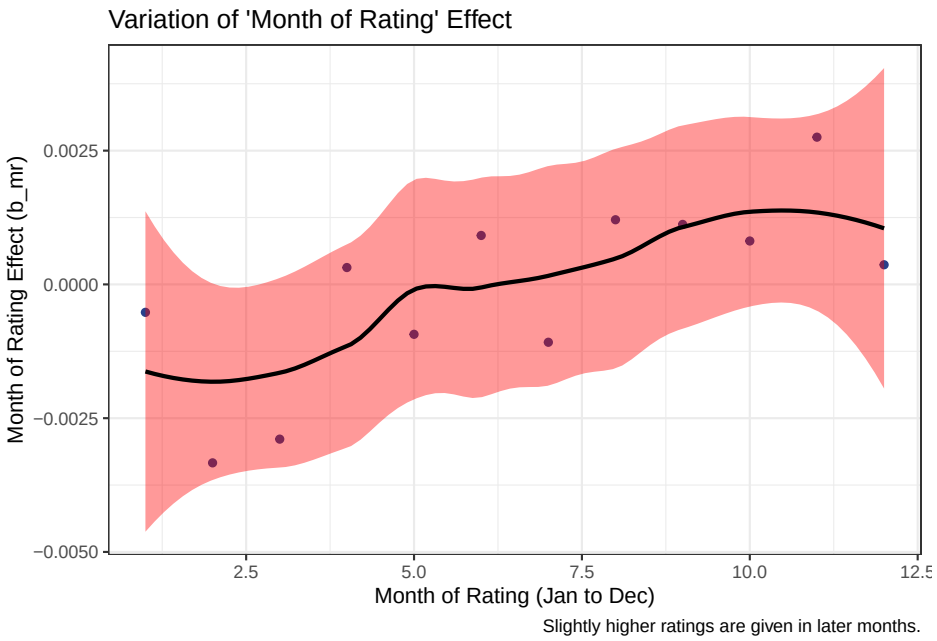
$$Y_{u,i} = \mu + b_i + b_u + b_{mr} + \epsilon_{u,i}$$

The model is implemented as follows:

```
#4. Modeling 'month of rating' effect:
month_rating_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  group_by(month_rating) %>%
  summarise(b_mr = mean(rating-mu-b_i-b_u))

#predicting ratings for the validation set
predicted_ratings_model4 <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  mutate(pred = mu + b_i + b_u + b_mr ) %>%
  pull(pred)

model4_rmse <- RMSE(predicted_ratings_model4)
model4_rmse
```

```
## [1] 0.8653459
```

The RMSE has improved further by $3.3628801 \times 10^{-4}\%$.



## Modeling 'Day of Rating' Effect

To account for the variation in ratings with the day of the week when rating is given, the term $b_{dr}$ is added to represent day-specific effects for the day of rating $dr$.

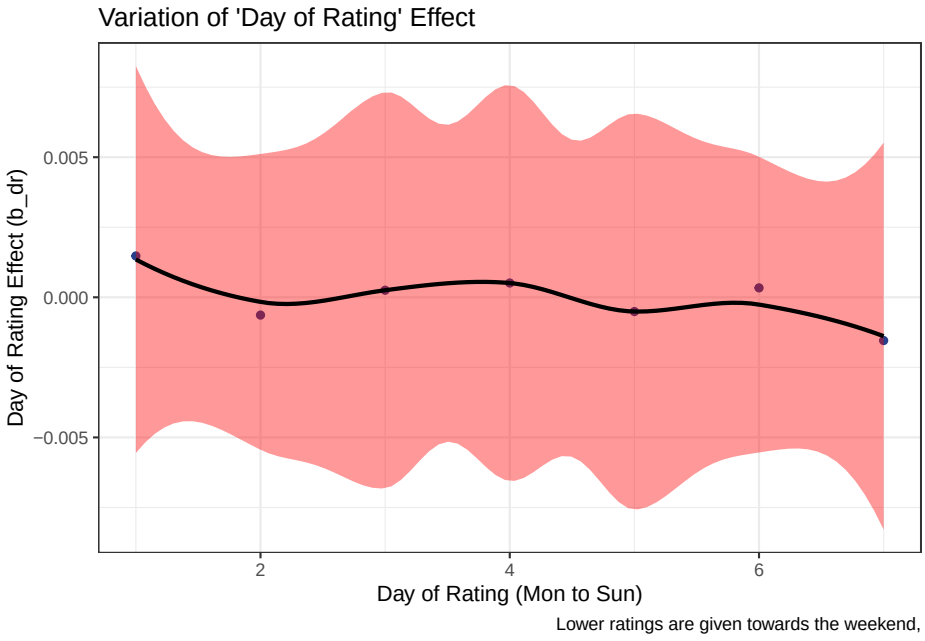
$$Y_{u,i} = \mu + b_i + b_u + b_{mr} + b_{dr} + \epsilon_{u,i}$$

The model is implemented as follows:

```
#4. Modeling 'day of rating' effect:
day_rating_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  group_by(day_rating) %>%
  summarise(b_dr = mean(rating-mu-b_i-b_u-b_mr))

#predicting ratings for the validation set
predicted_ratings_model5 <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  mutate(pred = mu + b_i + b_u + b_mr + b_dr) %>%
  pull(pred)

model5_rmse <- RMSE(predicted_ratings_model5)
model5_rmse
```

```
## [1] 0.8653447
```

The RMSE has improved further by $1.3665671 \times 10^{-4}\%$.



### Modeling 'Year of Movie Release' Effect

To account for the variation in ratings with the year in which the movie was released, the term $b_{ym}$ is added to represent year-specific effects for year of movie release $ym$.

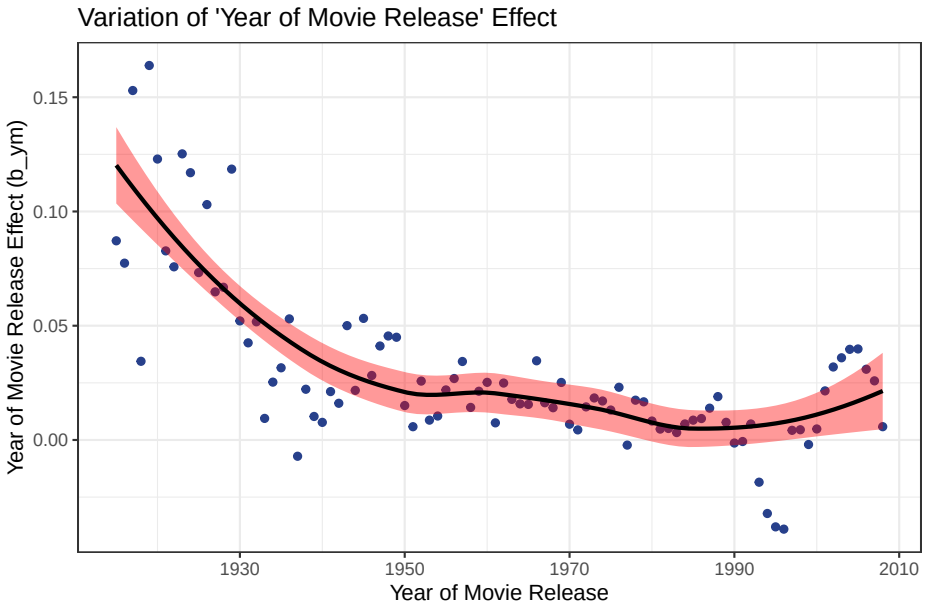
$$Y_{u,i} = \mu + b_i + b_u + b_{mr} + b_{dr} + b_{ym} + \epsilon_{u,i}$$

The model is implemented as follows:

```
#6. Modeling 'year of movie release' effect:
year_movie_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  group_by(year_movie) %>%
  summarise(b_ym = mean(rating-mu-b_i-b_u-b_mr-b_dr))

#predicting ratings for the validation set
predicted_ratings_model6 <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  left_join(year_movie_effect, by="year_movie") %>%
  mutate(pred = mu + b_i + b_u + b_mr + b_dr + b_ym) %>%
  pull(pred)

model6_rmse <- RMSE(predicted_ratings_model6)
model6_rmse
```

```
## [1] 0.8649988
```

The RMSE has improved further by 0.0399742%.


Variation of 'Year of Movie Release' Effect
Year of Movie Release Effect (b_ym)
0.15
0.10
0.05
0.00
1930
1950
1970
1990
2010
Year of Movie Release
Older movies are rated higher than newer ones.

### Modeling 'Years Lapsed' Effect

To account for the variation in ratings with the years lapsed between rating and movie release, the term $b_{yl}$ is added to represent year-gap-specific effects for years lapsed $yl$.

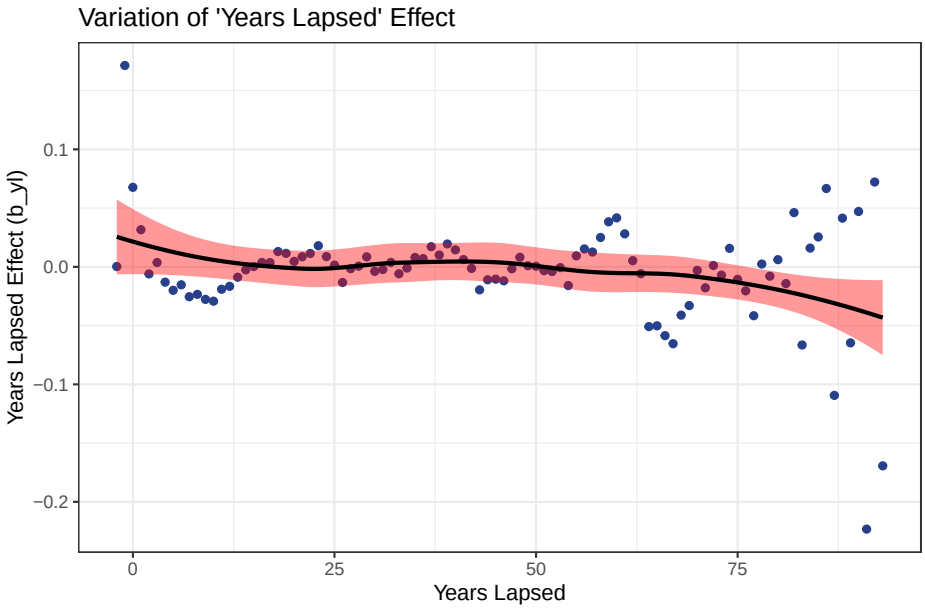
$$Y_{u,i} = \mu + b_i + b_u + b_{mr} + b_{dr} + b_{ym} + b_{yl} + \epsilon_{u,i}$$

The model is implemented as follows:

```
#7. Modeling 'years lapsed' effect:
years_lapsed_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  left_join(year_movie_effect, by="year_movie") %>%
  group_by(years_lapsed) %>%
  summarise(b_yl = mean(rating-mu-b_i-b_u-b_mr-b_dr-b_ym))

#predicting ratings for the validation set
predicted_ratings_model7 <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  left_join(year_movie_effect, by="year_movie") %>%
  left_join(years_lapsed_effect, by="years_lapsed") %>%
  mutate(pred = mu + b_i + b_u + b_mr + b_dr + b_ym + b_yl) %>%
  pull(pred)

model7_rmse <- RMSE(predicted_ratings_model7)
model7_rmse
```

```
## [1] 0.8647008
```

The RMSE has improved further by 0.0344587%.




Years lapsed affect movie ratings irregularly.

### Modeling Genre Effect

It was seen towards the end of the genre analysis that some genres receive higher average ratings than others. Earlier, a column was created with the maximum value of the average ratings of the genres of each movie. To account for the variation in ratings with the genres, the term $b_g$ is added to represent genre-specific effects for the max average genre rating $g$.

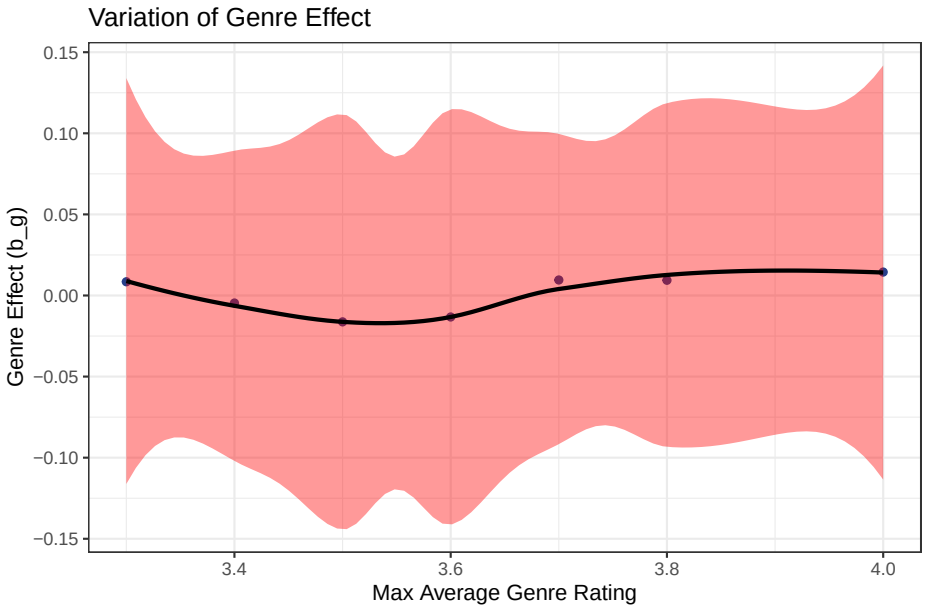
$$Y_{u,i} = \mu + b_i + b_u + b_{mr} + b_{dr} + b_{ym} + b_{yl} + b_g + \epsilon_{u,i}$$

The model is implemented as follows:

```
#8. Modeling genre effect:
genre_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  left_join(year_movie_effect, by="year_movie") %>%
  left_join(years_lapsed_effect, by="years_lapsed") %>%
  group_by(genres_rating) %>%
  summarise(b_g = mean(rating-mu-b_i-b_u-b_mr-b_dr-b_ym-b_yl))

#predicting ratings for the validation set
predicted_ratings_model8 <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  left_join(year_movie_effect, by="year_movie") %>%
  left_join(years_lapsed_effect, by="years_lapsed") %>%
  left_join(genre_effect, by="genres_rating") %>%
  mutate(pred = mu + b_i + b_u + b_mr + b_dr + b_ym + b_yl + b_g) %>%
  pull(pred)

model8_rmse <- RMSE(predicted_ratings_model8)
model8_rmse
```

```
## [1] 0.8646347
```

The RMSE has improved further by 0.007633%.




Genres affect movie ratings irregularly.

## Regularized Model

While managing to decrease the RMSE, it's still not perfect. A reason could be that there are several large effects formed by small sample sizes. The data is not uniformly distributed. Thus, regularization[10] is introduced to constrain the variability of effects and consequently penalize large effects that are formed using small sample sizes.

The new equation adds a penalty to the least squares equation that was being minimized earlier.
Least Squares Equation:

$$\sum_{u,i}(y_{u,i} - \mu - b_i - b_u - b_{mr} - b_{dr} - b_{ym} - b_{yl} - b_g)^2$$

New Equation to be minimized:

$$\sum_{u,i}(y_{u,i}-\mu-b_i-b_u-b_{mr}-b_{dr}-b_{ym}-b_{yl}-b_g)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2+\sum_{mr} b_{mr}^2+\sum_{dr} b_{dr}^2+\sum_{ym} b_{ym}^2+\sum_{yl} b_{yl}^2+\sum_g b_g^2\right)$$

The first term is the sum of the squares and the second is a penalty that gets larger when many effects are large. Calculus is used to finally show that the equation is minimized if effects are computed after dividing by sample size and penalty $\lambda$, instead of just the sample size as was done earlier. This way, when the sample size is large, the penalty is ignored and when the sample size is small, $\lambda$ shrinks the effect.

$\lambda$ needs to be tuned and cross validation will be used to do so.

The *edx* set is split for cross validation as follows:

```
#PREPARING FOR CROSS VALIDATION
set.seed(1, sample.kind="Rounding")
#if using R 3.5 or earlier, use `set.seed(1)`

#Creating the test index
edx_test_index <- createDataPartition(
  edx$rating, times=1, p=0.2, list = FALSE)

#Splitting the edx set for cross validation
edx_train <- edx[-edx_test_index,] #used for training
edx_test <- edx[edx_test_index,] #used for tuning

#To ensure only those movies and users are present in
#edx_test for which there are observations in edx_train
edx_test <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
```

The process for computing the RMSEs for a range of values of $\lambda$ is given below:

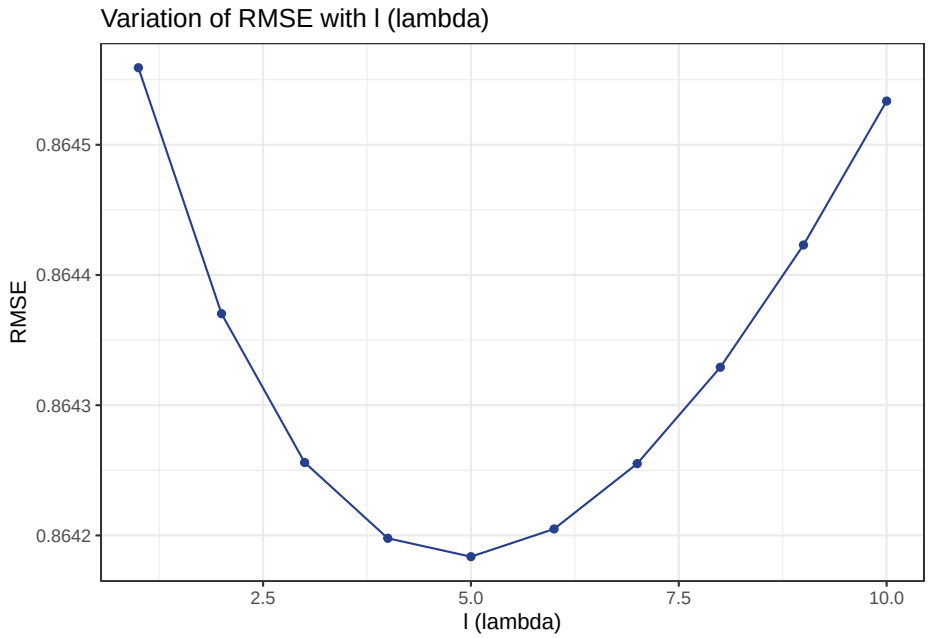
```
#Defining a sequence of values of lambda(l) to check RMSE
l <- seq(1, 10, 1)

#Applying lambda values and checking RMSE in edx_test:
rmses <- sapply(l, function(l){
```

[10]To read more about regularization, I suggest https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a

```r
#Mean Ratings:
mu <- mean(edx_train$rating)

#Movie Effect:
movie_effect <- edx_train %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating-mu)/(n()+l))

#User Effect
user_effect <- edx_train %>%
  left_join(movie_effect, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating-mu-b_i)/(n()+l))

#Month of Rating Effect:
month_rating_effect <- edx_train %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  group_by(month_rating) %>%
  summarise(b_mr = sum(rating-mu-b_i-b_u)/(n()+l))

#Day of Rating Effect:
day_rating_effect <- edx_train %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  group_by(day_rating) %>%
  summarise(b_dr = sum(rating-mu-b_i-b_u-b_mr)/(n()+l))

#Year of Movie Release Effect:
year_movie_effect <- edx_train %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  group_by(year_movie) %>%
  summarise(b_ym =sum(rating-mu-b_i-b_u-b_mr-b_dr)/(n()+l))

#Years Lapsed Effect:
years_lapsed_effect <- edx_train %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  left_join(year_movie_effect, by="year_movie") %>%
  group_by(years_lapsed) %>%
  summarise(b_yl = sum(rating-mu-b_i-b_u-b_mr-b_dr-b_ym)/(n()+l))

#Genre Effect:
genre_effect <- edx_train %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
```

```
    left_join(day_rating_effect, by="day_rating") %>%
    left_join(year_movie_effect, by="year_movie") %>%
    left_join(years_lapsed_effect, by="years_lapsed") %>%
    group_by(genres_rating) %>%
    summarise(b_g = sum(rating-mu-b_i-b_u-b_mr-b_dr-b_ym-b_yl)/(n()+l))

  #Applying the models on edx_test set:
  predicted_ratings <- edx_test %>%
    left_join(movie_effect, by="movieId") %>%
    left_join(user_effect, by="userId") %>%
    left_join(month_rating_effect, by="month_rating") %>%
    left_join(day_rating_effect, by="day_rating") %>%
    left_join(year_movie_effect, by="year_movie") %>%
    left_join(years_lapsed_effect, by="years_lapsed") %>%
    left_join(genre_effect, by="genres_rating") %>%
    mutate(pred = mu + b_i + b_u + b_mr + b_dr + b_ym + b_yl + b_g) %>%
    pull(pred)

  #Calculating the RMSE:
  sqrt(mean((edx_test$rating - predicted_ratings)^2))
})
```

The resulting RMSEs are plotted against $\lambda$ to find the value of $\lambda$ which best minimizes RMSE in the *edx_test* set.




The value of $\lambda$ that minimizes the RMSE is 5. This will be used to make our final model as follows:

```
#storing the l which minimized RMSE on cross valdiation
l <- l[which.min(rmses)]

#Mean Ratings:
```

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

#Movie Effect:
movie_effect <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating-mu)/(n()+l))

#User Effect:
user_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  group_by(userId) %>%
  summarise(b_u = sum(rating-mu-b_i)/(n()+l))

#Month of Rating Effect:
month_rating_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  group_by(month_rating) %>%
  summarise(b_mr = sum(rating-mu-b_i-b_u)/(n()+l))

#Day of Rating Effect:
day_rating_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  group_by(day_rating) %>%
  summarise(b_dr = sum(rating-mu-b_i-b_u-b_mr)/(n()+l))

#Year of Movie Release Effect:
year_movie_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  group_by(year_movie) %>%
  summarise(b_ym = sum(rating-mu-b_i-b_u-b_mr-b_dr)/(n()+l))

#Years Lapsed Effect:
years_lapsed_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  left_join(year_movie_effect, by="year_movie") %>%
  group_by(years_lapsed) %>%
  summarise(b_yl = sum(rating-mu-b_i-b_u-b_mr-b_dr-b_ym)/(n()+l))

#Genre Effect:
genre_effect <- edx %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
```

```
  left_join(year_movie_effect, by="year_movie") %>%
  left_join(years_lapsed_effect, by="years_lapsed") %>%
  group_by(genres_rating) %>%
  summarise(b_g = sum(rating-mu-b_i-b_u-b_mr-b_dr-b_ym-b_yl)/(n()+l))

#Applying the model on the validation set:
predicted_ratings_reg <- validation %>%
  left_join(movie_effect, by="movieId") %>%
  left_join(user_effect, by="userId") %>%
  left_join(month_rating_effect, by="month_rating") %>%
  left_join(day_rating_effect, by="day_rating") %>%
  left_join(year_movie_effect, by="year_movie") %>%
  left_join(years_lapsed_effect, by="years_lapsed") %>%
  left_join(genre_effect, by="genres_rating") %>%
  mutate(pred = mu + b_i + b_u + b_mr + b_dr + b_ym + b_yl + b_g) %>%
  pull(pred)
```

The final RMSE is:

```
#Calculating the RMSE:
rmse_reg <- RMSE(predicted_ratings_reg)
rmse_reg
```

```
## [1] 0.8641196
```

Thus regularization reduces the RMSE by 0.0595802%.

## Collaborative Filtering Model

A collaborative filtering model is implemented by using matrix factorization as done by the *recosystem*[11] package. In this, only user-movie interactions are used to model the system, not movie attributes.

In the handcrafted model, user-specific and movie-specific differences were accounted for, but an important source of variation was ignored. Groups of movies and groups of users have similar rating patterns.This is handled by converting the ratings data into a matrix such that each user gets a row and each movie gets a column. Matrix factorization is used to solve this complex user-movie interaction matrix by decomposing it into a product of two smaller matrices.

The code to implement matrix factorization using recosystem is as follows:

```
set.seed(1, sample.kind="Rounding")
#if using R 3.5 or earlier, use `set.seed(1)

#converting the 'edx' and 'validation' sets into recosystem-compatible forms
#data_memory function specifies a data set from R objects
edx_reco <-  with(edx, data_memory(
  user_index = userId,
  item_index = movieId,
  rating = rating))
validation_reco  <-  with(validation, data_memory(
  user_index = userId,
  item_index = movieId,
  rating = rating))

#creating the recosystem object
r <-  recosystem::Reco()

#training the model with default parameters
r$train(edx_reco)

#getting the predicted ratings
#out_memory is used to return the result as an R object
predicted_ratings_reco <- r$predict(validation_reco, out_memory())
```

The RMSE for the predictions of *recosystem* is:

```
#calculating the RMSE
rmse_reco <- RMSE(predicted_ratings_reco)
rmse_reco
```

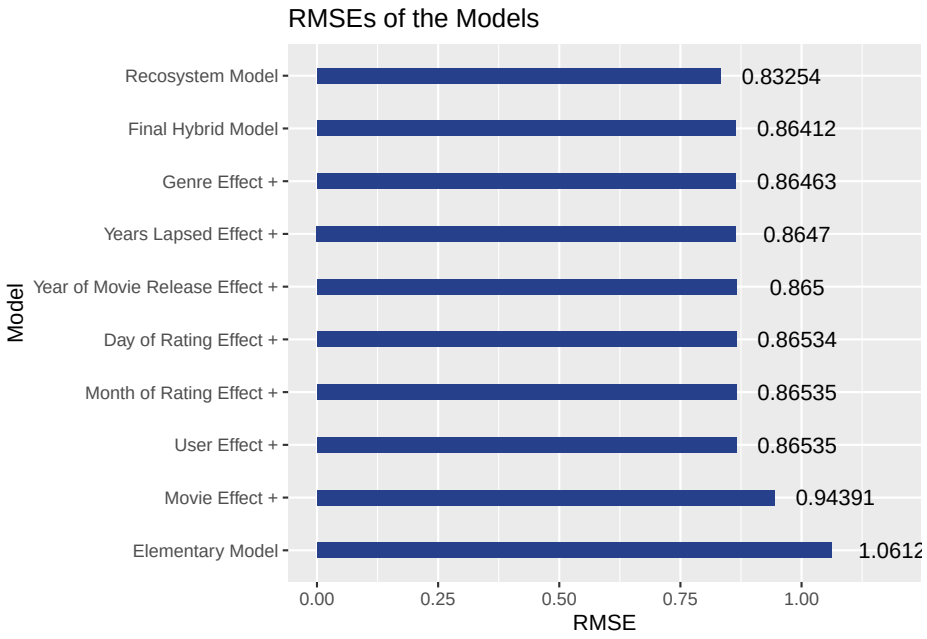
```
## [1] 0.8325396
```

This is an improvement of 3.6545831% over the final handcrafted model.

[11]The *recosystem* package does matrix factorization specifically for recommender systems, i.e. specifically for settings in which the user-movie interaction matrix has many missing values. It's written by experts in numerical matrix factorization, and features a number of useful options. Source: http://heather.cs.ucdavis.edu/~matloff/189G/Supplements/Supp02182020.pdf

# Results

The handcrafted hybrid model shows great promise and is a great way to understand underlying trends. But the genius of the *recosystem* package is unparalleled. Even without tuning the parameters and sticking to defaults, an RMSE of 0.8325396 was achieved.



## Conclusion and Future Outlook

In this project, the *movielens* data set was explored in detail, with an extra fun genre analysis. To predict movie ratings for a user and movie, two different algorithms have been constructed with the lowest reported RMSE being 0.8325396. For the first model, a step-by-step development process was shown. The second model involved an advanced method, so I stuck to the defaults. These models have aided a relatively successful movie recommendation system.

That said, with machine learning, the more the merrier. More data, more models and more tuning will unquestionably better the system. It must be noted that the computational time was quite large, and the processor RAM requirements are high. Optimization can be considered in the future.