Capstone: Movielens Project

Dhanya G

15/06/2021

Introduction

In today's world, people have unlimited access to everything and many things. So they face multiple choices every day like starting their day with news suggestions to browsing movies on internet all night long. Due to the wide range of possibilities and choices, people tends to look for suggestions to minimize their searching time. This is what Recommendation systems do. They are dominating business worlds nowadays. From Prime to Netflix, Youtube to Spotify, recommendation systems are one of the principal classifications in the field of machine learning algorithms.

In the past, people normally buy products recommended by their friends or the people who they trust. But now due to the influence of digital era, that circle has expanded to include online sites that uses different kinds of recommendation system algorithms. Recommendation systems analyse the data using different algorithms and recommends the most relevant products to users.They train the model using historical data of customers and recommend the products based on that.

For example, for a movie recommendation system, the aim is to predict how ratings differ by a specific user's interest of a movie based on their other movies rating or considering similar users categories or movie's rating according to genres etc., In this project, I use various machine learning techniques to construct Movie Recommendation System based on the Movielens dataset. I use 10M version of the MovieLens dataset generated by the GroupLens research lab as a dataset according to the instructions. In this project, I built recommendation model using Regression models, Regularization and Matrix Factorization approach to predict movie rating by training the *edx* dataset and testing by *validation* set using Root Mean Square Error(RMSE) as an evaluation metric.

Executive Summary

The initial phase of the project gives a picture about the dataset with its overview and the necessary techniques to be implemented for reorganising the data into a format that is ready to explore.

Dataset

The entire *Movielens* dataset found here https://grouplens.org/datasets/movielens/latest/, is quite large. So, 10M version of the MovieLens dataset (https://grouplens.org/datasets/movielens/10m/) is advised to use in this project generated by GroupLens. GroupLens is a research lab at the University of Minnesota specializing in recommender systems, online communities, digital libraries etc., to collect and make available several datasets for the research people.

The dataset described in this project consists of over *10 million* ratings on *10677* distinct movies by *69878* discrete users. The features available in this dataset are movieId, userId, title (movie title with its release year), genres, rating and timestamp (date and time) of the rating. In the latter project, I extracted year

from the title column for observing movie release year and to find the effect of year on the rating. As timestamp variable contains date and time of the rating given by user, I extracted for finding time elapse between movie release year and the time of rating. As genre is a multi-categorical variable, I extracted into multiple columns for calculating genre effect to predict rating.

Overview

Here, recommendation model is built motivated by the approaches of the Netfilx challenges winners.

For this project, I built recommendation system using baseline models (Regression model) with different effects(Movie Effect, User Effect, Genre effect, Time Effect, Year Effect), Regularization and Matrix Factorization approach using *Recosystem* package. This recommendation model predicts how much rating user will give for a movie. The model performance is evaluated by Root Mean Square Error (RMSE) as a loss function. RMSE compares the model evaluation by comparing actual value with the predicted value. The objective of this project is to built a recommendation system with minimum RMSE *(RMSE lower than 0.86490)*

RMSE

Root Mean Square Error (RMSE) is a frequently used loss function that measures the differences between values (sample or population values) predicted by a model and the values observed. These deviations are called residuals when the calculations are performed over the data sample that was used for estimation. RMSE is similar to standard deviation. The formula is defined by,

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

where N is the number of user - movie combinations and the sum occurring over all these combinations. $y_{u,i}$ is the rating for movie i by user u and $\overline{y}_{u,i}$ is the prediction.

Methods & Analysis

Data Preprocessing

In this project, I downloaded the required 10M movielens dataset as mentioned in the project instructions. The *movielens* dataset is divided into two parts namely, *edx* set *(90%)* which is used for training the model and *validation* set *(10%)* for evaluation purposes. As mentioned, *validation* set (final hold-out test set) should only be used at the end of the project to test the final algorithm which outputs minimum RMSE. Because of that, *edx* dataset should be further split into two parts, *training_edx* set for training and *test_edx* for testing the model until required RMSE is reached. Finally, for the recommendation model with minimum RMSE, we train the entire *edx* set with the *validation* set to acquire RMSE value.

1.Installing and loading packages

Loading required libraries and data

```
if(!require(plyr)) install.packages("plyr")
if(!require(tidyverse)) install.packages("tidyverse")
if(!require(caret)) install.packages("caret")
if(!require(data.table)) install.packages("data.table")
```

```
if(!require(lubridate)) install.packages("lubridate")
if(!require(recosystem)) install.packages("recosystem")
if(!require(RColorBrewer)) install.packages("RColorBrewer")
```

```
library(plyr)
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(recosystem)
library(RColorBrewer)
```

2.Data Extraction

Since Netflix data is not publicly available, required *movielens* dataset can be extracted from GroupLens Research lab's database (https://grouplens.org/datasets/movielens/10m/). Downloading data from "http://files.grouplens.org/datasets/movielens/ml-10m.zip".

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

3.Building the dataset

Extracted the ratings and movies information and integrated into *movielens* dataset.

```
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dll, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines(unzip(dll, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "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))
movielens <- left_join(ratings, movies, by = "movieId")
```

4.Exploring the dataset

Movielens dataset has over 10 million ratings for 10677 unique movies by 69878 discrete users. We can see this in a tidy format below:

```
# size of the entire dataset
nrow(movielens)
```

```
## [1] 10000054
```

```
# number of discrete movies in the dataset
n_distinct(movielens$movieId)
```

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

```
# number of discrete users in the dataset
n_distinct(movielens$userId)
```

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

```
# Overview of the dataset "movielens" with 6 rows
head(movielens)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     231      5 838983392          Dumb & Dumber (1994)
## 4:      1     292      5 838983421               Outbreak (1995)
## 5:      1     316      5 838983392               Stargate (1994)
## 6:      1     329      5 838983392 Star Trek: Generations (1994)
##                            genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:                        Comedy
## 4:  Action|Drama|Sci-Fi|Thriller
## 5:       Action|Adventure|Sci-Fi
## 6: Action|Adventure|Drama|Sci-Fi
```

It consists of 6 features namely *userId*, *movieId*, *rating*, *timestamp* (date and time of rating), *title* (with release year) and *genres*. Each row represents a rating given by one user to one movie.

5.Splitting the data

The *movielens* dataset is splitted into two parts namely, *edx* and *validation* sets. *Validation* 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 & movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Bring back removed rows from validation to edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
rm(dll, ratings, movies, test_index, temp, movielens, removed) #Clear out unwanted files
```

Data Exploration and Visualization

General Overview of the *edx* dataset:

```
head(edx) #initial 6 rows with 6 columns
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:        Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:        Children|Comedy|Fantasy
```

```
# Glimpse of the edx dataset
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

After implementing the given code in project instructions, *edx* dataset consists of over 9 million ratings made up of *six* features. *Validation* set consists of 999,999 observations (about *10%* of *movielens* dataset).

Recommendation model predicts the outcome (y) by the feature *rating*. The *rating* variable ranges from 0 to 5 where 1 indicates worst movie and 5 indicates good movie.

Number of unique users who rated and number of distinct movies are:

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

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

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

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

Number of observations(rows and columns) in the *edx* set are:

```
nrow(edx)
```

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

```
ncol(edx)
```

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

Ratings Distribution

Lets look at the distribution of the ratings:

```
unique(edx$rating)
```

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

There are 10 distinct ratings ranging from 0.5 to 5.0 given by user for a movie.

```
# How many zeros were given as ratings in the edx dataset?
sum(edx$rating == 0)
```

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

```
# How many threes were given as ratings in the edx dataset?
sum(edx$rating == 3)
```

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

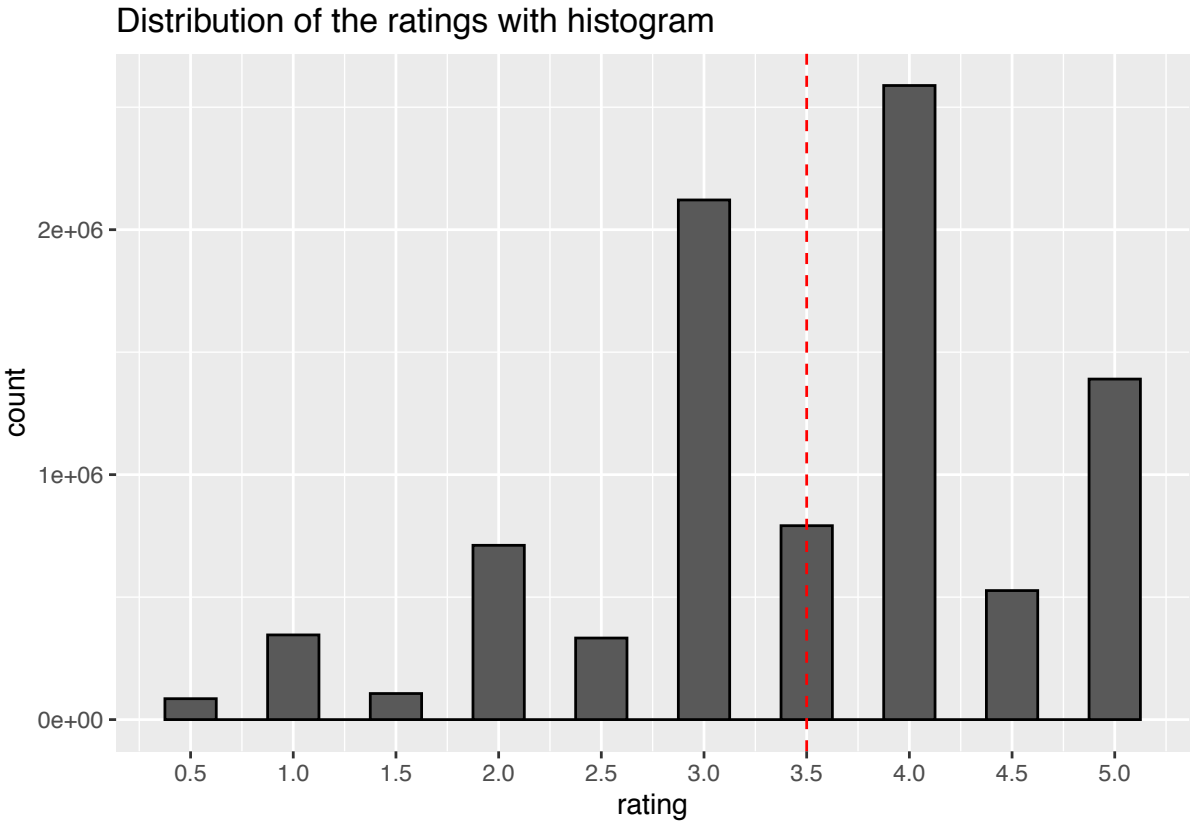
Lets look how some ratings score are higher than the others:

```
# Number of ratings for each rating score
#Most given ratings from most to least
edx %>% group_by(rating) %>% summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # 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
```

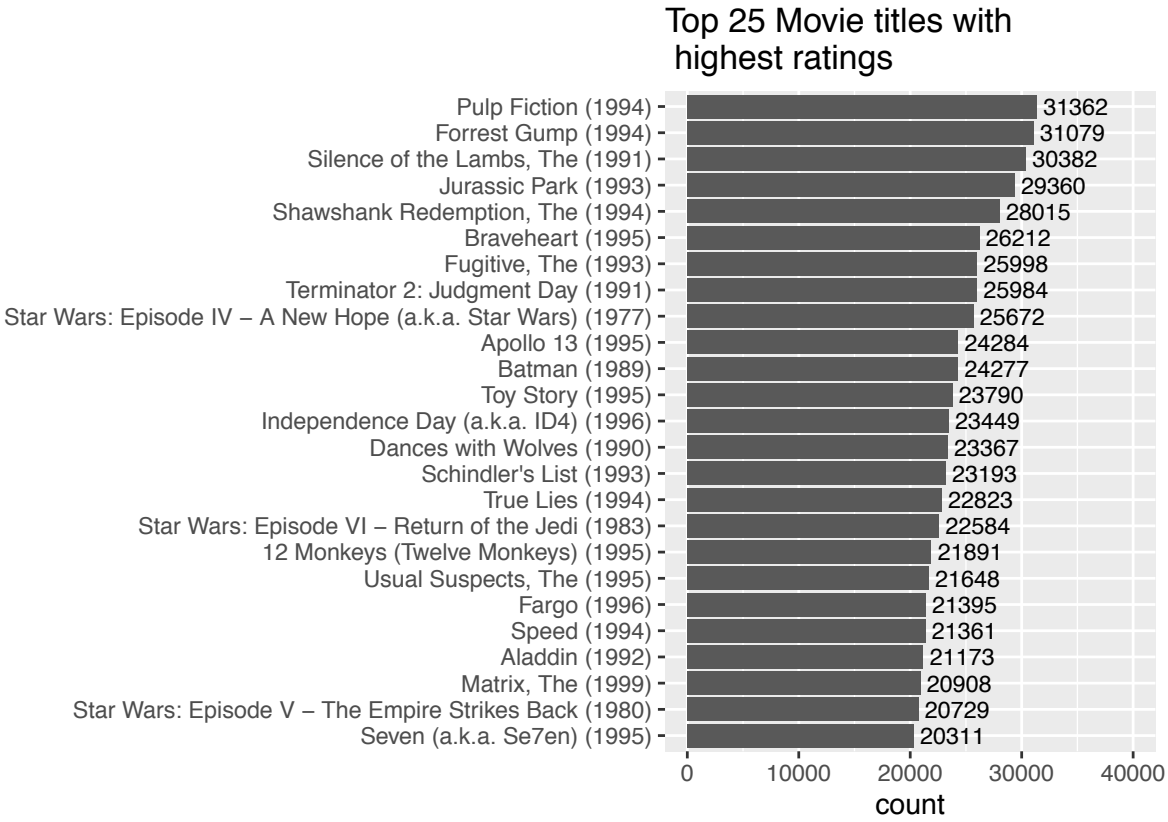
This shows that rating score 4 has been given most by the users for a movie and 0.5 as the least score for movies. The topmost five rating scores are 4,3,5,3.5 and 2.

To detect if ratings distribution has effect on recommendation model, plot histogram to understand the ratings.



This shows that half-star ratings are less common than the whole full star ratings. The plot shows the overall distribution of the ratings are skewed to the right side. It also shows no user has given 0 as a rating.

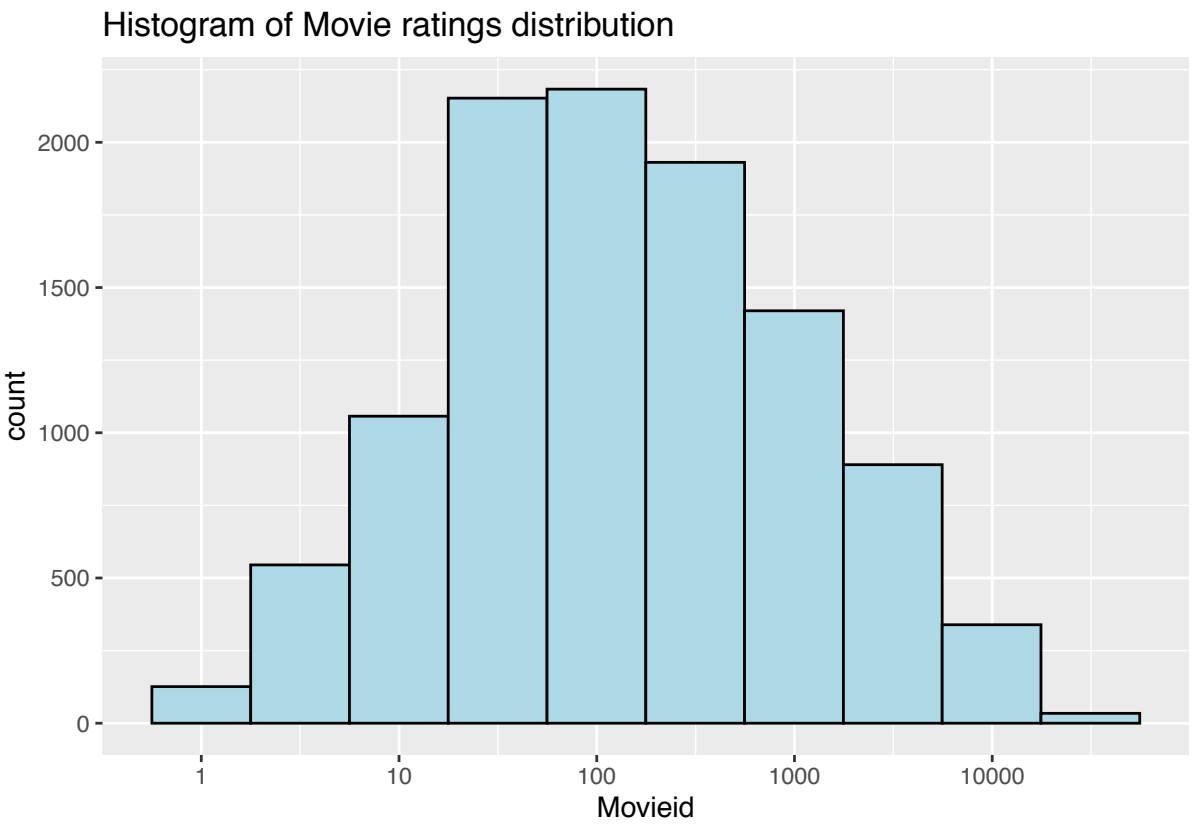
To understand that ratings are different for different movies, lets plot the topmost 25 movies which has highest number of ratings in the dataset.



Bar plot shows movie Pulp Fiction (1994) has highest movie rating of 31362 followed by Forrest Gump (1994) and Silence of the Lambs, The (1991) of ratings 31079 and 30382 respectively. Seven (a.k.a.Se7en) (1995) takes 25th position in the movie titles barplot of over 20311 number of ratings respectively.

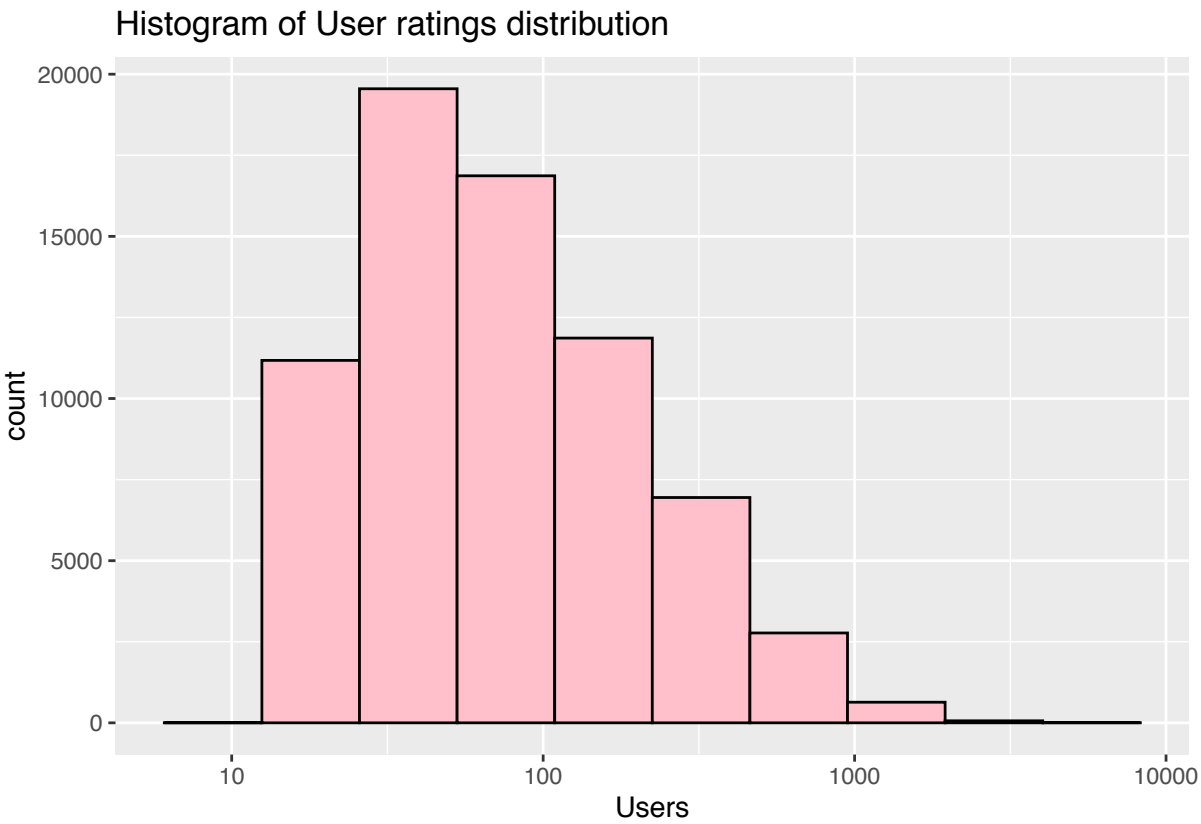
Recommendation Model is a challenging machine learning approach because each outcome has different set of predictors. To predict rating*(y)* for a movie i by user u, all ratings concerned with movie i by user u should be taken into account. However, different users rate different movies and a different number of movies. In addition to that, movie i similar to same category movies or user u who have the same preferences (genre or rating style) as other users information must be included. Let's look at some of the general properties of the data to better understand the challenges.

Distribution of the Movie ratings



This shows that some movies get rated more than others while others have been rated very few times. This distribution tells that there are popular blockbuster movies watched by many people and unpopular movies like artsy, independent movies are watched by few people. This shows there is movie effect in recommendation model.

Distribution of User ratings



This shows not every user gives rating to all movies. Likewise, this plot shows that there are some users who are more active compared to others.*i.e.,* some users give more ratings to good and average movies(*4,4.5,5*) while some are cranky to even good movies(they give rating of only *2,2.5,3*). We can find user to user differences in the rating for a movie *i*. This shows there is an user effect in the model to predict movie ratings.

Exploration of Movie Release Year Effect

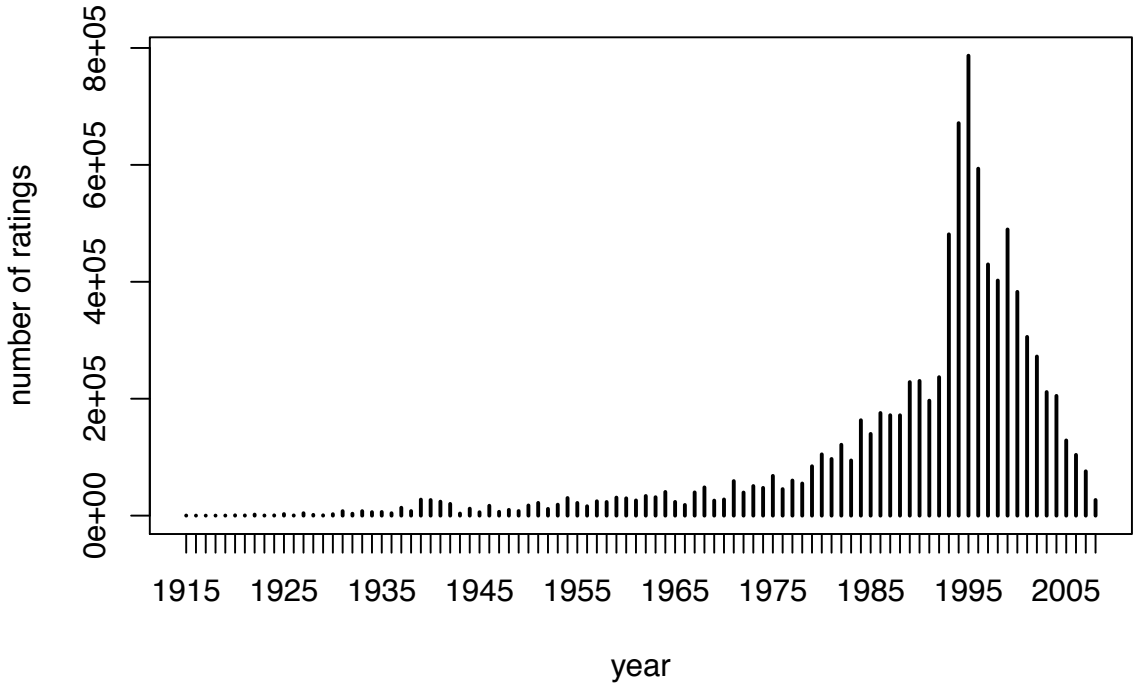
To detect if the year of the movie release have an effect in recommendation model, extract *year* variable from *title* column using str_extract function.

```
edx_year <- edx %>%
  mutate(year = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"),
                                       regex("\\d{4}"))))
head(edx_year)
```

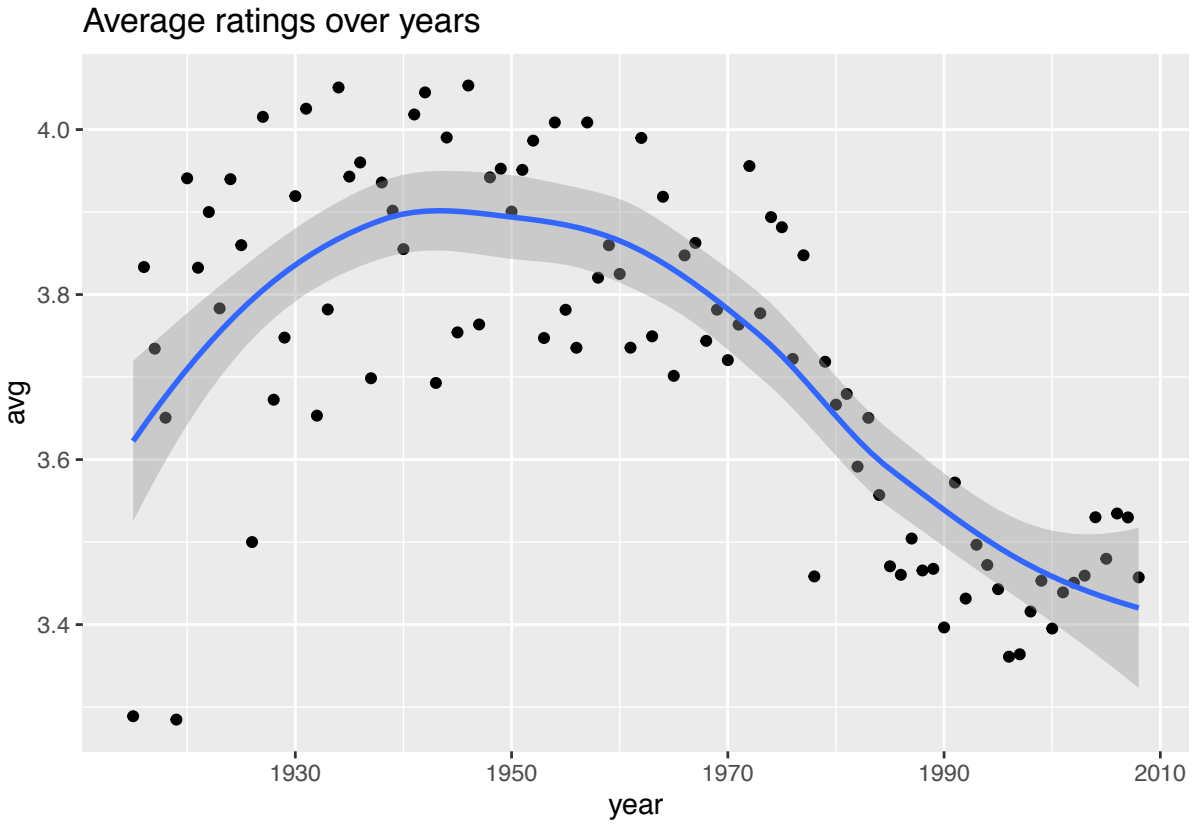
```
##    userId movieId rating timestamp                          title
## 1:      1     122      5 838985046                Boomerang (1992)
## 2:      1     185      5 838983525                 Net, The (1995)
## 3:      1     292      5 838983421                 Outbreak (1995)
## 4:      1     316      5 838983392                 Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres year
```

```
## 1:                Comedy|Romance 1992
## 2:         Action|Crime|Thriller 1995
## 3:  Action|Drama|Sci-Fi|Thriller 1995
## 4:       Action|Adventure|Sci-Fi 1994
## 5: Action|Adventure|Drama|Sci-Fi 1994
## 6:       Children|Comedy|Fantasy 1994
```

The *edx_year* dataset has 7 columns with addition of *year* column (consists of movie release year).



Plot shows that after 1990s, there are higher ratings for movies. The year 1995 has the highest number of movie ratings when compared to others. It appears that latest movies gets rated by many users than the oldest ones.



As one can see, movies that are a bit older has highest average rating compared to the newer ones. From 1930 to 1975, average rating of a movie is approximately over 3.8 when compared to 2010 which has 3.5 average rating. Plot shows newer movies have low average ratings over decades.This does not mean that older movies had good movies compared now. This is because users u have less time to rate movie i when compared to older movies who had lot of time. Perhaps, users are watching more older movies which are classic and well-acclaimed by everyone(blockbuster movies).

Exploration of Time Effect

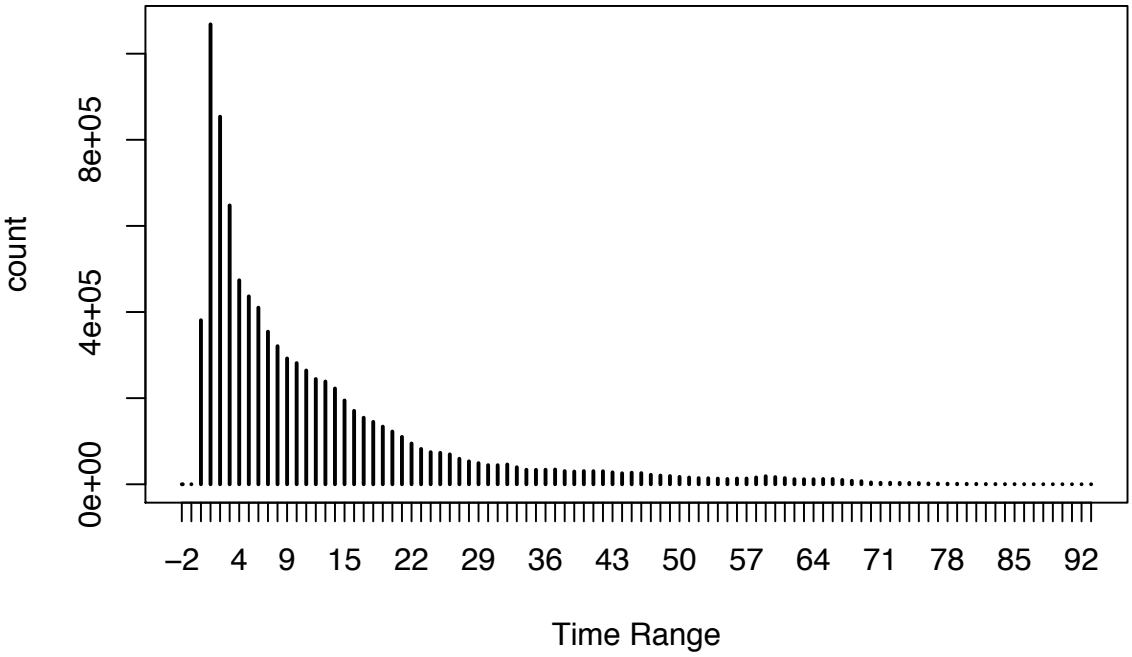
To detect if the movie rated year have an effect in recommendation model, extract the year when movie is rated by user from timestamp variable. Lets take a look at the distribution of ratings according to the year of movie rating.

```
#Calculating Time Elapse between movie release year and year rated by user
edx_year <- edx_year %>%
  mutate(year_rated = year(as_datetime(timestamp)),time_range = year_rated - year)
```

Now, *edx_year* dataset has 9 columns with addition of *year_rated*(year when user rate the movie) and *time_range*(the time(year) between movie release year and movie rated year) column.

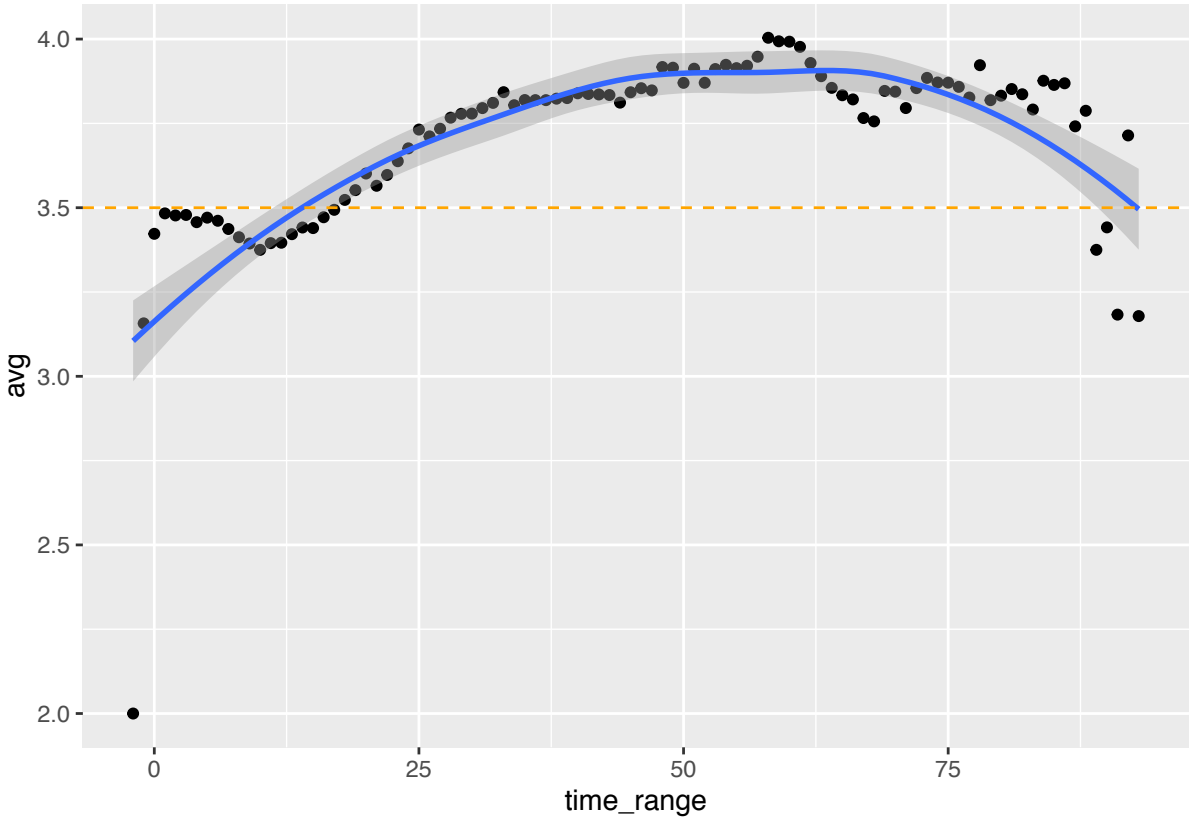
```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
```

```
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                          genres year year_rated time_range
## 1:                Comedy|Romance 1992       1996          4
## 2:          Action|Crime|Thriller 1995       1996          1
## 3:  Action|Drama|Sci-Fi|Thriller 1995       1996          1
## 4:       Action|Adventure|Sci-Fi 1994       1996          2
## 5: Action|Adventure|Drama|Sci-Fi 1994       1996          2
## 6:       Children|Comedy|Fantasy 1994       1996          2
```



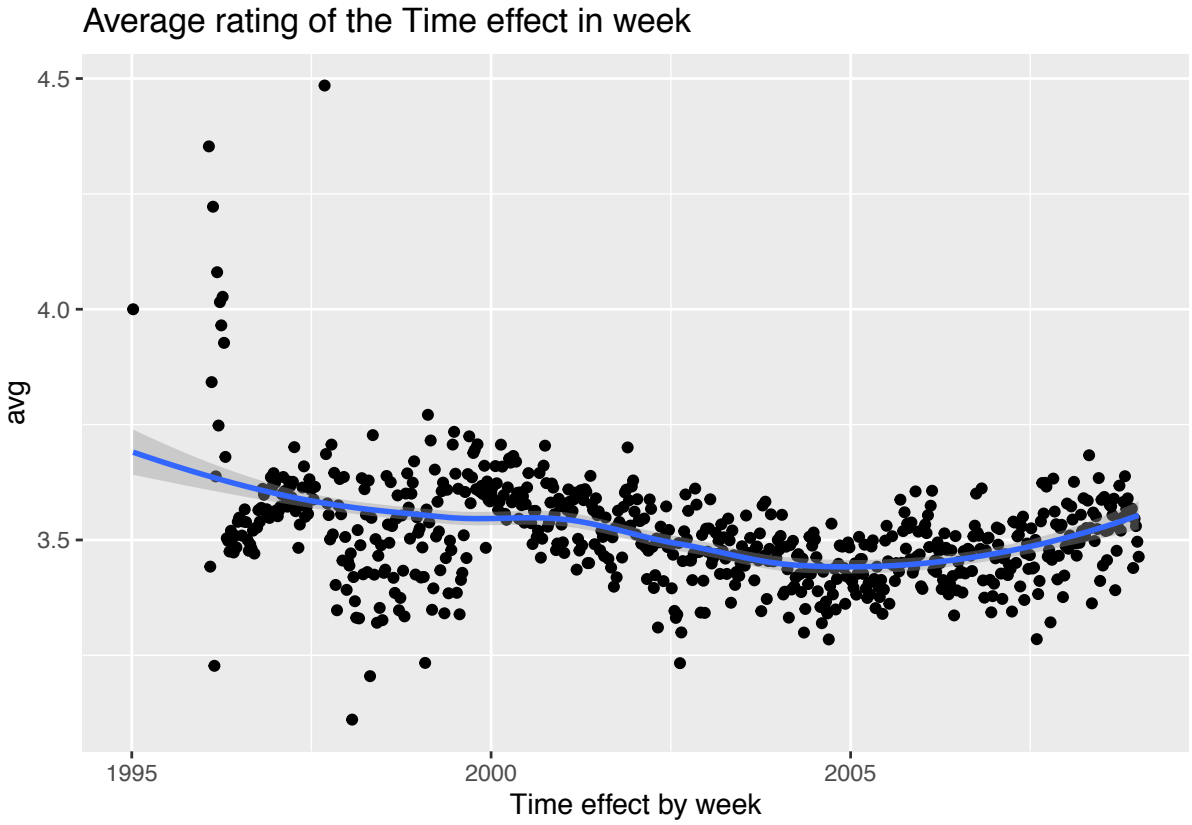
Plot shows time range of 1 is the most common among movies which has highest rating. It appears most ratings were given for movies that are within 25 years time range(between movie release year and movie rated year)

To detect if time-range of movie at the time of rating affects rating, plot average rating for movies with certain time-range at the time of rating against the time-range.



This plot shows movies with time_range below 14 have around average or slightly lower than average ratings while older movies with higher time_range have average greater than average rating with some exceptions. The orange dashed line represents overall average rating for the time_range. The blue line represents smoothened overall trend. This is because older movies with acclaimed name(popular or blockbuster) are still watched widely and highly recommended while newer movies with less time_range are not watched widely enough by users to have effect.

To detect if number of ratings for week time period affects model, lets extract date_week from timestamp variable (which has units) that can be converted into week time period by round_date function and plot it against average number of ratings.



By visualizing the trend, one can say there is some evidence of time effect in the model not a strong effect.

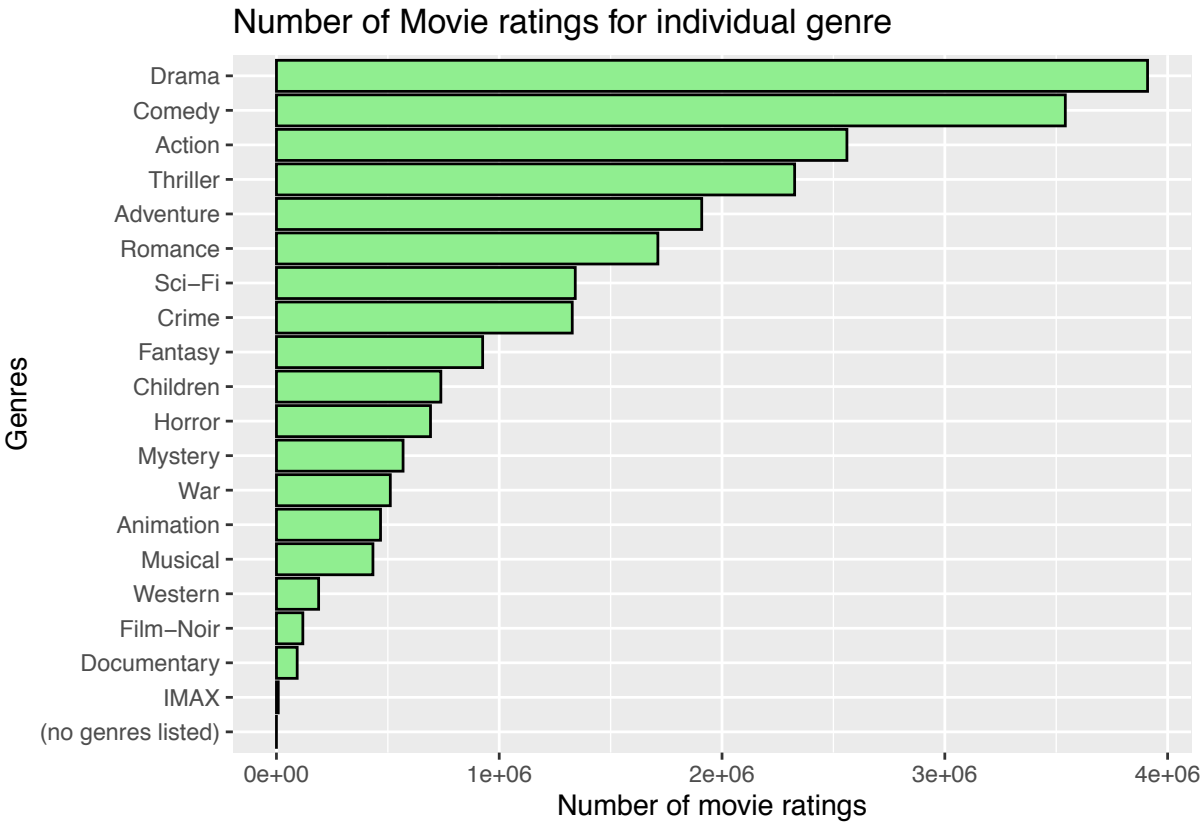
Exploration of Genre effect

To understand if genres have an effect to predict rating, lets split individual genres with their number of movie ratings.

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
```

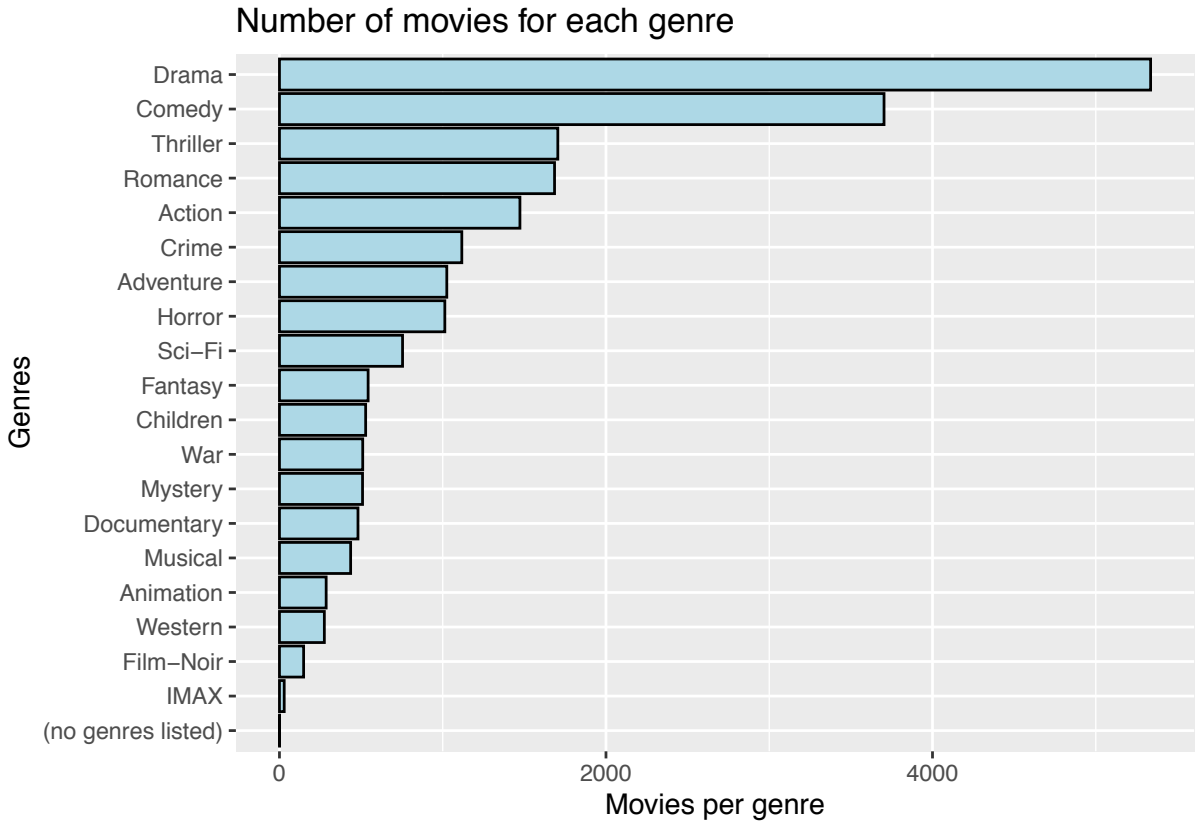
```
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

Lets plot number of movie ratings in each genres:



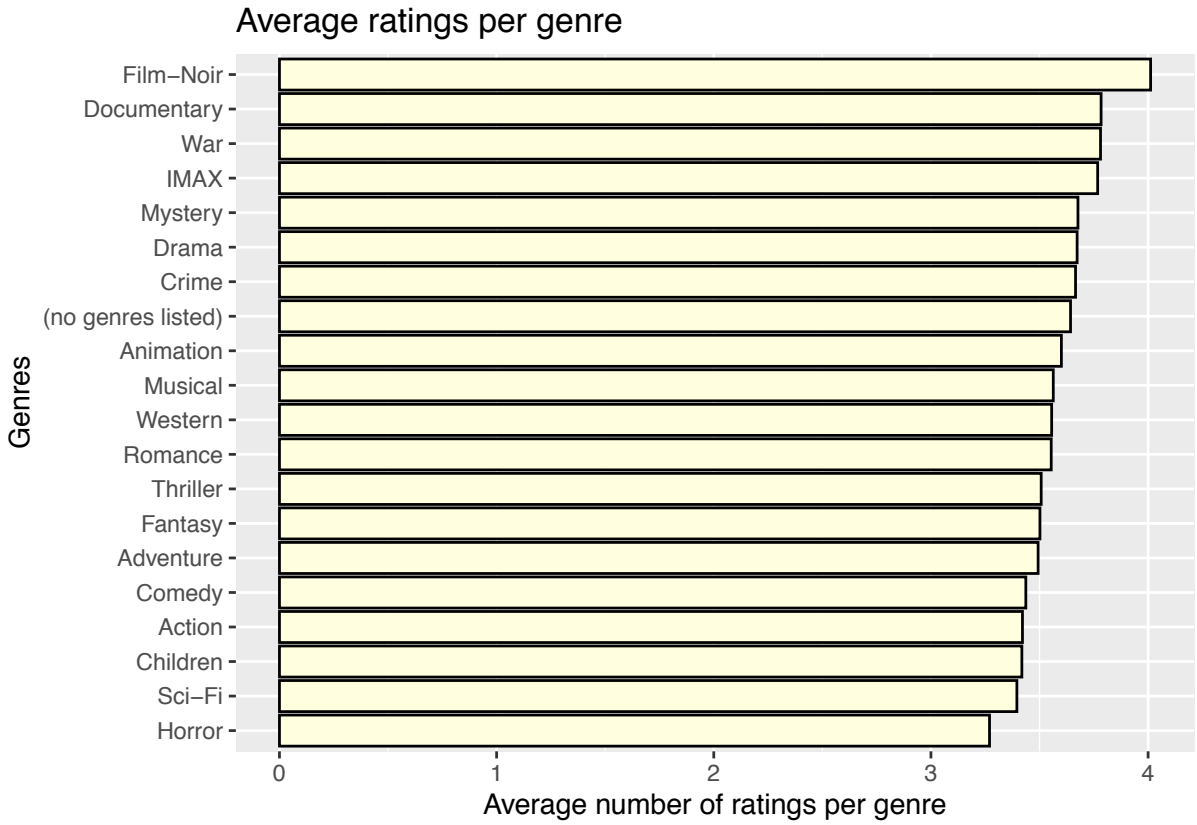
It appears that genre *Drama* has highest number of movie ratings(3910127) followed by *Comedy(3540930)* and *Action(2560545)*. The last genre in the plot is *(no genres listed)* which has 7 movie ratings. This does not mean that people prefer to watch *Drama* genre more compared to others. This is because, there may be more movies in the *Drama* genre.

Lets plot number of movies for each genre:



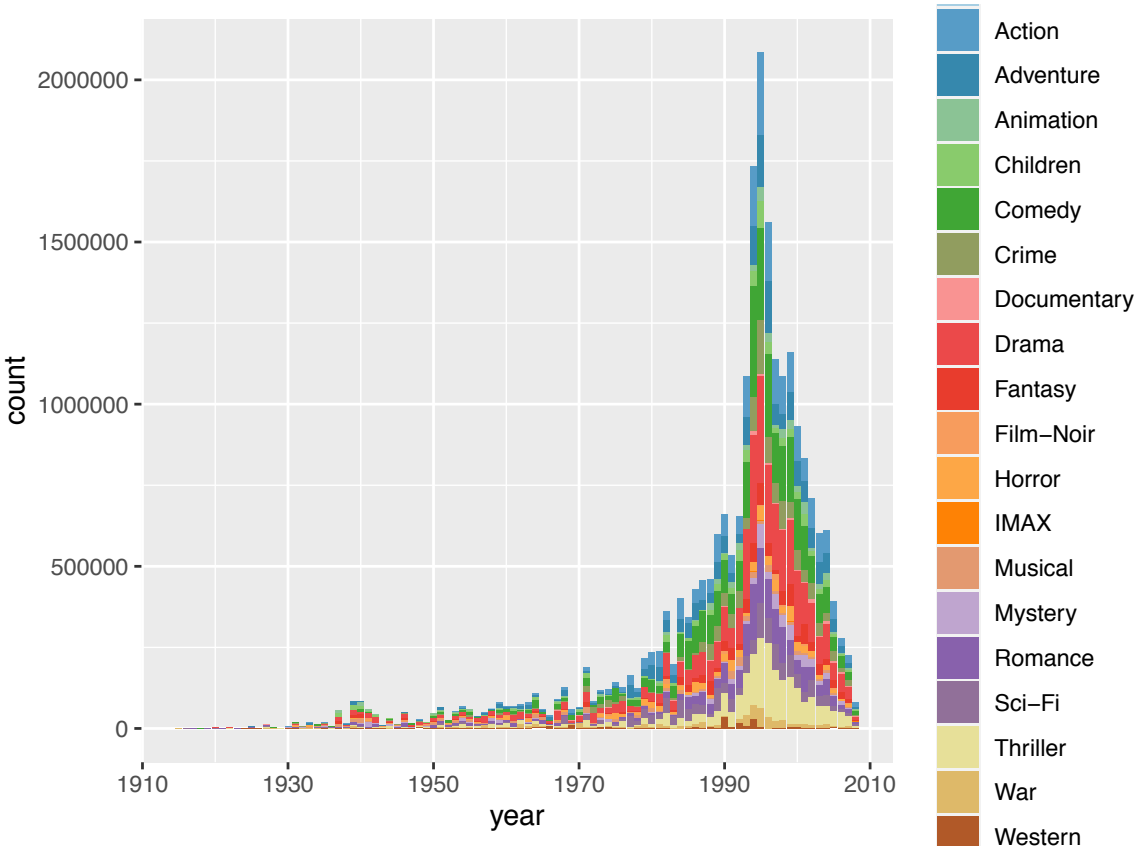
As expected, *Drama* genre has more number of movies when compared to others.

To detect how much ratings each genre accommodate, lets plot average rating per genre below:



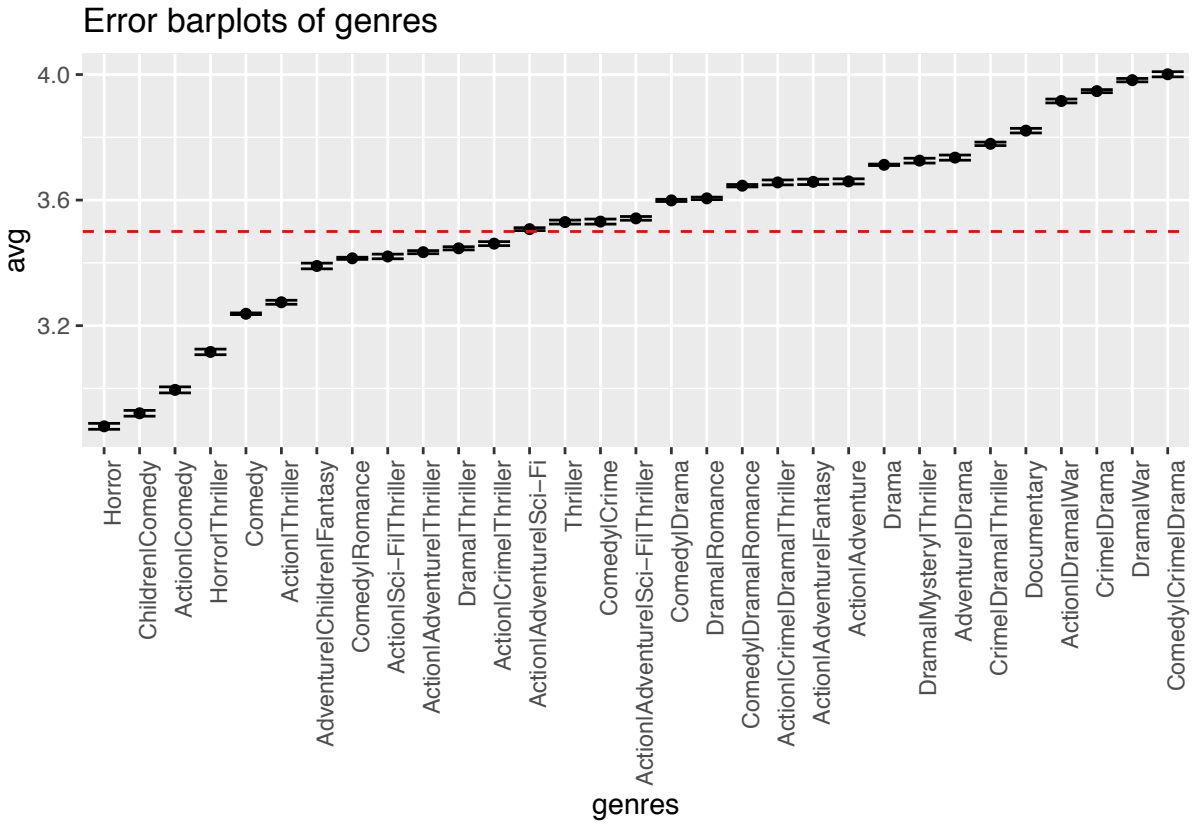
Here, one can see a different trend. Genre *Drama* does not have highest average number of ratings even though they have higher number of movies. Instead, *Film-Noir* has the highest average no. of ratings than the average followed by *Documentary* and *War* etc., This implies most of the *Film-Noir* genre movies are highly rated irrespective of their movie numbers. *Horror* genre has the least number of average ratings. It shows genre effects on recommendation model tends to be smaller.

Another way to visualize is *genres* by their *years*. In which year, which genre movies are widely watched or rated by users?



The plot shows different years have different genres as popular genre(mostly watched or rated) over time. It shows users preferences towards movies are continuously changing with respect to time. We can't pick point and tell these genre movies will be popular over decades. So, it is difficult to take this genre-year effect into consideration.

Lets analyze the *genres* without splitting the data to individual genres. Visualize the column *genres* whether it has an effect to predict rating by user u over movie i



The above plot shows genre *Horror* has the least number of ratings compared to *Horror|Thriller*. This shows horror movies are not widely rated by users. *Comedy|Crime|Drama* has average number of ratings higher than the average and also higher than individual *Drama*, *Comedy* genres. This plot shows that there is an considerable effect of genres without splitting in recommendation model.

Model Analysis

For building the models, *edx* dataset is split into two parts namely, *training_edx (90%)* and *test_edx (10%)* respectively. This is because validation set should only be used against the final model for predicting movie rating. So, validation set (final hold-out) set is not used to train the algorithm and test it with multiple models during model development.

```
#Splitting edx dataset into training and test sets
set.seed(1,sample.kind = "Rounding")
y = edx$rating
edx_test_index <- createDataPartition(y, times = 1, p = 0.1, list = FALSE)
training_edx <- edx %>% slice(-edx_test_index)
test_temp <- edx %>% slice(edx_test_index)
# Make sure userId & movieId in test_edx set are also in training_edx set
test_edx <- test_temp %>%
  semi_join(training_edx, by = "movieId") %>%
  semi_join(training_edx, by = "userId")
# Bring back removed rows from test_edx to training_edx set
removed <- anti_join(test_temp, test_edx)
training_edx <- rbind(training_edx, removed)
rm(test_temp,removed) #clear unwanted files
```

Depending on the data exploration and visualization above, one can understand there is an movie effect and user effects in the model to predict movie rating because different users rate differently and different movies have different ratings. Also, there is a slight year effect on the model not much, because every year have different popular movie choices depending on the users preferences and years trends. There is a considerable amount of genre effect on the model in movie ratings. By considering these approaches, recommendation system is built using baseline model (Regression model) with Movie and User Effect, Regularization with Movie and User effect, Year effect, Time Effect, Genre effect and Matrix factorization using recosystem. After building the model, best models are evaluated and used as final algorithm against validation set.

In this section, each algorithm model with their effects are evaluated for model performance:

1. Regression models:

1.1. Simple Average method(Baseline model)

Lets start the first recommendation model by the simplest method based on the instructions described in the textbook(https://rafalab.github.io/dsbook/recommendation-systems.html). Consider all movies have same ratings irrespective of users choices. A model which considers same rating for all the movies and users would appear like this:

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

where $Y_{u,i}$ is the outcome(rating) for movies, μ is the same rating for all movies and $\epsilon_{u,i}$ is the independent errors sampled from the same distribution centered at 0.

1.2. Movie Effect Model

We know to create a better model we can't assume that all movies have same ratings. Because in reality, some movies have more ratings than others leading to movie effect in the model. This is due to block-buster movies and reputable name movies who have a significant role in the users as first preferable choice. We can confirm this by looking our data exploration on movie ratings where different movies are rated differently. The previous model is followed by the average rating for movies, b_i for different movie ratings. Movie Effect model is represented as:

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

where b_i is movie specific effect or bias.

1.3. Movie and User Effects Model

Since some users rate movies higher than the others, there is an user effect as well as in the model. This is because some users are cranky to rate movies(for example., some give 1.5,2,2.5,3 as rating for good movie because they have more expectations or this is their way to rate movies) while others are super active in rating(like 3.5,4,4.5). This leads to the addition of user effect in the previous model:

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

where b_u is the average rating for users for user bias effect.

1.4. Movie Release Year Effect Model

Since each year have different movie release trends, there is a slight year effect in the model. Because, one can see from the year exploration plot, latest released movies have higher ratings than oldest ones and

older movies have highest average ratings when compared to newer ones because of the classics and reputed movies. Additional improvement to the model is:

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

where b_y is the year effect/bias.

1.5. Time range Model

Since there is some gap between movie release year and movie rated year by users, it affects the model slightly to predict ratings. From the exploration plot, it is clear that ratings are given mostly within 20 years of time range and movies with larger time range have average ratings higher than the others. Time range model can be expressed as

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

where b_{tr} is the time range (between released year and movie rated year) effect.

1.6. Time Effect Model

Visualizing the data exploration part, it is clear there is some evidence of time effect in the model when considering number of ratings for week time period by user u for movie i. Time Effect model is described as:

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

where b_t is the time effect/bias.

1.7. Genre Effect

Clearly, genre of movies affect the model in predicting rating because every user have certain genre preferences. This can be visualized in the genre exploration plot. Genres splitting into individual ones by rows would drastically decrease the RMSE due to model overtraining(because some movies have multiple genres combinations, when splitting, same movie record is duplicated into multiple rows containing individual genres depending on genre combinations). So, I didn't split the genres by rows and predict rating. Instead, I used two methods for calculating genre effect. The first one is taking the current genre format as it is and evaluating model performance and the second one is placing individual genres into column format and evaluating the model.

Method 1: Simply Grouping Genres

As visualized in the plot, some genres have more average number of ratings compared to the others. This can improve our model but genres without splitting(taking current format) have over 800 genres which leads to overfitting. Either say, we try this in our model:

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

where b_g is the average ranking for genres, genre bias.

Method 2: Columnizing Genres

In this method, I put each genre into separate columns to find genre effect on the model to avoid the problem of overtraining by genre-splitting into rows as described above. I opted out *"(no genres listed)"* as it is not a genre but a movie whose genre information is not provided.

```
## # A tibble: 7 x 6
##   userId movieId rating  timestamp title                genres
##    <int>   <dbl>  <dbl>      <int> <chr>                <chr>
## 1   7701    8606    5   1190806786 Pull My Daisy (1958) (no genres listed)
## 2  10680    8606    4.5 1171170472 Pull My Daisy (1958) (no genres listed)
## 3  29097    8606    2   1089648625 Pull My Daisy (1958) (no genres listed)
## 4  46142    8606    3.5 1226518191 Pull My Daisy (1958) (no genres listed)
## 5  57696    8606    4.5 1230588636 Pull My Daisy (1958) (no genres listed)
## 6  64411    8606    3.5 1096732843 Pull My Daisy (1958) (no genres listed)
## 7  67385    8606    2.5 1188277325 Pull My Daisy (1958) (no genres listed)
```

From the above, it is clear that only one movie "Pull My Daisy" (8606) released in 1958 have no genre information placed in 7 rows with different ratings. Since it have no genre information, remove *"(no genres listed)"* from the dataset. We can achieve this format by spreading individual genres into 19 genre columns and taking account of this in our model:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{k_{u,i}}\beta_k + \epsilon_{u,i}$$

with $x_{k_{u,i}} = 1$ if $g_{u,i}$ is genre k

where k = 1 to K is the number of individual genres, K = 19 (total individual genres), $x_{k_{u,i}}$ is 0 or 1 for genre k and β_k is the genre bias/effect.

1.8. Movie,User,Timerange and Genre(column format) Model.

Since Time range model and genre effect by columns shows slight effects in the model as visualised in the data exploration plot, lets combine and use it along with movie and user effects and see how prediction improves in the model. The model can be expressed as:

$$Y_{u,i} = \mu + b_i + b_u + b_{tr} + \sum_{k=1}^{K} x_{k_{u,i}}\beta_k + \epsilon_{u,i}$$

where b_i is movie specific effect, b_u is the average rating for users,b_{tr} is the time range (between released year and movie rated year) effect and $x_{k_{u,i}} = 1$ if $g_{u,i}$ is genre k and k = 1 to K is the number of individual genres, K = 19 (total individual genres) and β_k is the genre bias/effect.

2. Regularization

On predicting the model, there are some noisy estimates which leads to large errors thereby increasing the RMSE. To avoid this problem, Regularization is used. Regularization permits us to penalize large estimates that are formed using small sample sizes. Regularization is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values. It constrains the total variability of the effect sizes.

Movie Effects

The regularized movie effect controls the total variablilty of movie effects,*(b_i)*. Instead of minimizing the least squares equation, we minimize an equation that adds a penalty term to the model:

$$\sum_{u,i}(y_{u,i} - \mu - b_i)^2 + \lambda \sum_{i} b_i^2$$

where first part is sum of squares(b_i's) and second part is penalty term that gets larger for large b_i. the values of b_i that minimize this equation are:

$$\hat{b_i}(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

where n_i is the number of ratings made for movie i.

Movie and User Effects

In least squares approach, we estimate the model based on the observed deviation from average rating which is just based on one number. So, it won't give a better prediction. The general idea of penalized regression is to control the total variability of the movie*(b_i)* and user*(b_u)* effects. Instead of minimizing the least squares equation, we minimize an equation that adds a penalty term to the model:

$$\sum_{u,i} (y_{u,i} - \mu - b_i - b_u)^2 + \lambda (\sum_i b_i^2 + \sum_u b_u^2)$$

where first term is the sum of squares(b_i's and b_u's) that fit the given ratings and second term is the regularized penalty term that gets larger when many b_i and b_u are large. Here, cross-validation is used to pick the lambda, λ (which gives minimum RMSE). Using calculus, we can actually show that the values of b_u that minimize this equation are:

$$\hat{b_u}(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu} - \hat{b_i})$$

where n_i is the number of ratings made for movie i. When sample size n_i is very large, it gives stable estimate, then the penalty λ, is effectively ignored since $n_i + \lambda \approx n_i$. However, when n_i is small, then the estimates $\hat{b_i}(\lambda)$ and $\hat{b_u}(\lambda)$ are shrunken towards 0. The larger λ, the more we shrink.

3. Matrix Factorization by parallel stochastic gradient descent

Matrix factorization is one of the collaborative filtering algorithms used in recommendation systems. Collaborative filtering discovers similarities by users past preferences and predict items(movie rating) based on users who have same preferences. It identifies the relationship between users and items(movie) entities. Matrix factorization works by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. Matrix Factorization became widely popular during the announcement of Netflix prize challenge winners due to its effectiveness.The prediction results can be improved by assigning different regularization weights to the latent factors based on items(movies)' popularity and users' activeness.

In this model, matrix factorization is calculated based on the residuals of the movie and user model. I calculate matrix factorization by parallel stochastic gradient descent using *recosystem* package.

The package *recosystem* is an R wrapper of the LIBMF library, an open source library, which does matrix factorization for recommender system. LIBMF is a parallelized library, where users can take advantage of multicore CPUs to speed up the computation. It also utilizes some advanced CPU features to further improve the performance. Also, unlike most other R packages for statistical modeling which store the whole dataset and model object in memory, LIBMF (and hence recosystem) is much hard-disk-based. That is to say, recosystem will have a comparatively small memory usage.

The *recosystem* approximates the whole rating matrix $R_{m\ddot{O}n}$ by the product of two matrices of lower dimensions, $P_{k\ddot{O}m}$ and $Q_{k\ddot{O}n}$, such that

$$R \approx PQ$$

Let p_u be the u-th column of P, and q_i be the i-th column of Q, then the rating given by user u on movie i would be predicted as $p_u q_i$.

Here, P and Q is solved by optimization problem:

$$\min_{P,Q} \sum_{(u,i)\in R} [f(p_u, q_i; r_{u,i}) + \mu_P||p_u||_1 + \mu_Q||q_i||_1 + \frac{\lambda_P}{2}||p_u||_2^2 + \frac{\lambda_Q}{2}||q_i||_2^2]$$

where (u, i) are positions of observed entries in R, $r_{u,i}$ is the observed rating, f is the loss function and $\mu_P, \mu_Q, \lambda_P, \lambda_Q$ are penalty parameters to avoid overfitting.

The process of solving the matrices P and Q is model training, and the selection of penalty parameters is called parameter tuning.

The training data is a sparse matrix triplet form which consists of userId, movieId and rating. i.e., each line in the file contains three numbers. Testing data is similar to training data that serves as a validation set on which RMSE of prediction can be calculated.

Results

The above models are implemented one by one by training and testing the dataset(training_edx and test_edx). The optimal model is identified by minimum RMSE value. The final model which gives minimum RMSE is tested against validation set(final hold-out test) to predict rating for a movie i by user u.

RMSE

The model performance is evaluated using loss function,RMSE. RMSE(Root Mean Square Error) is defined as:

```
# Create a function to calculate Root Mean Square error(RMSE)
RMSE <- function(actual_value, predicted_value){
  sqrt(mean((actual_value - predicted_value)^2))
}
```

1. Simple Average method

In this model, ratings are same for all movies and users. Here, estimate that minimizes the RMSE is the least squares estimate of μ(average of all ratings):

```
#Consider all movies have same ratings regardless of users.
average <- mean(training_edx$rating)
average
```

```
## [1] 3.512456
```

```
# Calculate RMSE
simple_model_rmse <- RMSE(test_edx$rating, average)
# Lets tabulate the results
RMSE_results <- tibble(Model = "Simple Average Model", rmse = simple_model_rmse)
RMSE_results
```

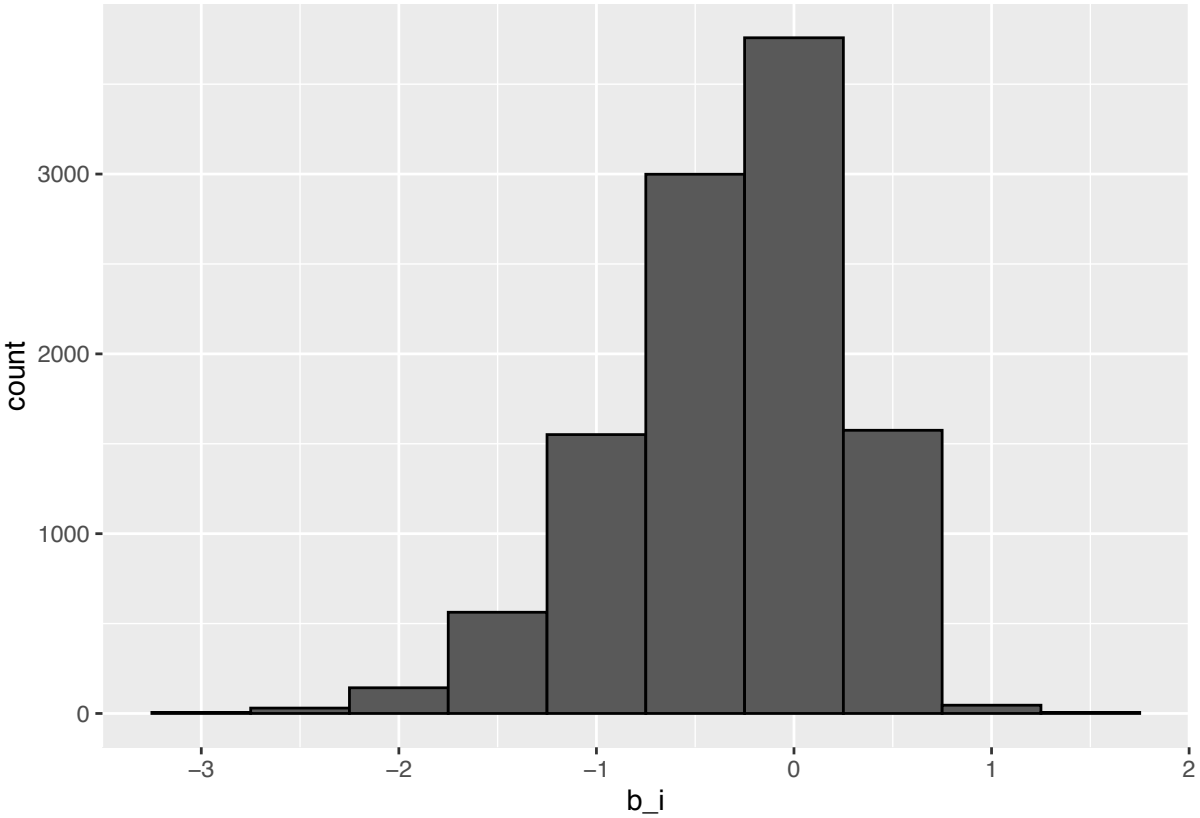
```
## # A tibble: 1 x 2
##   Model                 rmse
##   <chr>                <dbl>
## 1 Simple Average Model  1.06
```

Average rating is 3.512 and RMSE is 1.06. To get RMSE below 0.86490, we should consider other effects to improve the model.

2. Movie Effect Model

Since different movies are rated differently, movie bias term(b_i) is added to this model. For each movie, the average of the ratings on that specific movie will have a difference from the overall average rating of all movies.

```
mu <- mean(training_edx$rating)
# Calculate movie bias(b_i)
movie_average <- training_edx %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
# Plot the distribution of the Movie bias effect
movie_average %>% ggplot(aes(b_i)) + geom_histogram(bins = 10, color = "black")
```




```
# How much prediction improves by adding b_i term to the model?
predicted_value <- join(x=test_edx,y=movie_average,by = "movieId", type = "left")
pred_calc <- predicted_value %>% mutate(pred = mu + b_i) %>% pull(pred)
Movie_model <- RMSE(test_edx$rating,pred_calc)
RMSE_results <- bind_rows(RMSE_results, tibble(Model = "Movie Model", rmse = Movie_model))
RMSE_results %>% knitr::kable()
```

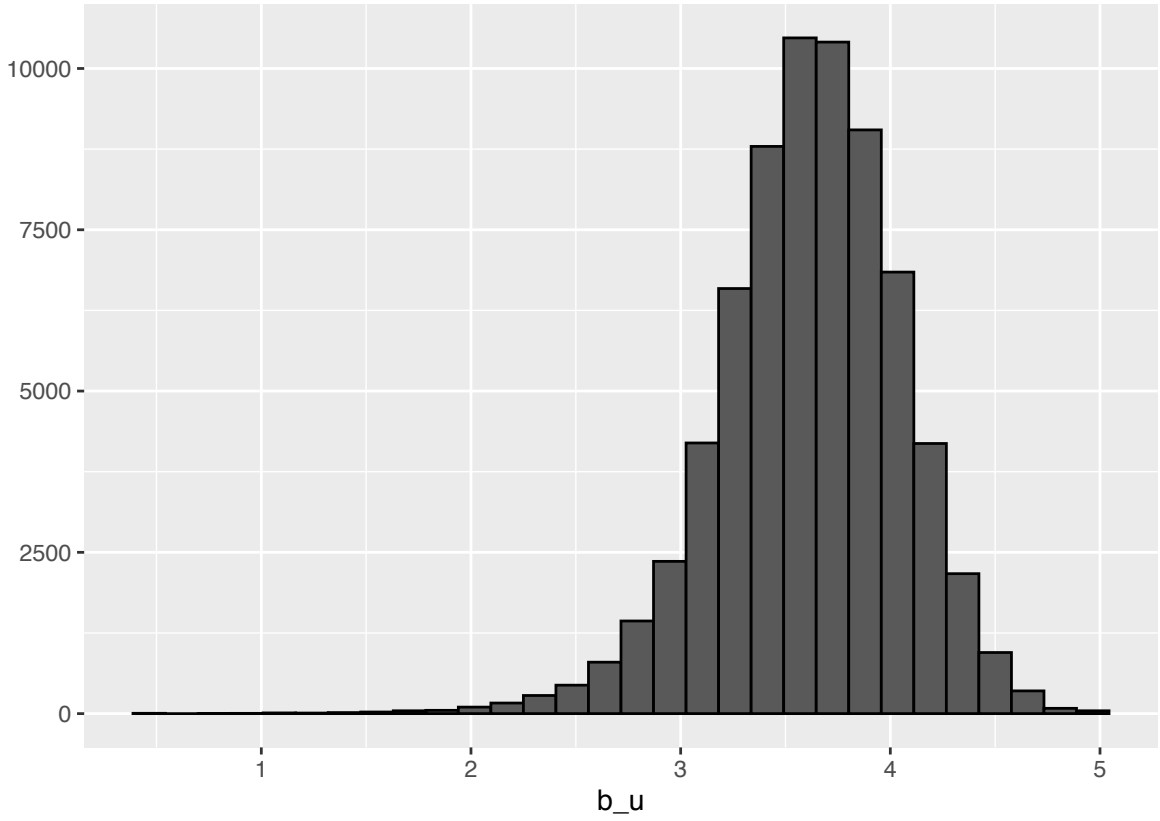
Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615

By modeling movie effects, there is an quite improvement of RMSE(about 0.94296). Still can do better!

3. Movie and User Effect model

Since some users rate every movie while others are grouchy towards rating(as seen in the plot below), user specific effect *(b_u)* is taken into consideration along with movie effect.

```
# Plot the distribution of User Effect for those who have rated 100 or more movies.
training_edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  qplot(b_u,geom = "histogram" , data = ., bins = 30 ,color = I("black"))
```




```
# Calculate b_u on the training set
user_average <- merge(x=training_edx,y=movie_average,by = "movieId",all.x = TRUE) %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
# How much prediction improves by adding b_u term along with b_i in the model?
predicted1<- merge(x=test_edx,y=movie_average,by = "movieId",all.x = TRUE)
```

```
predicted2 <- merge(x=predicted1,y=user_average,by = "userId",all.x = TRUE)
predicted_value1 <- predicted2 %>% mutate(pred = mu + b_i + b_u) %>% pull(pred)
Movie_User_Model <- RMSE(test_edx$rating,predicted_value1)
# Lets tabulate the results
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Movie & User Effect Model",
                                 rmse = Movie_User_Model))
RMSE_results %>% knitr::kable()
```

Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843

There is an quite improvement to RMSE all the way from 0.9429 to 0.8646 by modeling user specific effects along with movie effect. This explains movie ratings are dependent of users choices. Therefore, Movie and User effects should be included in the model.

4. Movie Release Year Effect Model

Since different movies are released in different years, there is variability across movies released in the years. To take that into account, year bias effect(b_y) is considered. Variable title has movie name along with release year. Extract release *year* variable from *title* column using str_extract function for both *training_edx* and *test_edx* sets.

```
# Extracting release year from training_edx and test_edx
training_edx_1<- training_edx %>%
  mutate(year = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"),
                                       regex("\\d{4}"))))
test_edx_1 <- test_edx %>%
  mutate(year = as.numeric(str_extract(str_extract(title, "[/(]\\d{4}[/)]$"),
                                       regex("\\d{4}"))))
# Calculate release year of movie bias(b_y)
year_average <- training_edx_1 %>%
  left_join(movie_average,by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  group_by(year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u))
# How much prediction improves by b_y term along with b_i and b_u?
predicted_value2 <- test_edx_1 %>% left_join(movie_average, by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  left_join(year_average,by = "year") %>%
  mutate(pred = mu + b_i + b_u + b_y) %>% pull(pred)
Movie_release_year_model <- RMSE(test_edx$rating,predicted_value2)
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Movie & User & Year Effect Model",
                                 rmse = Movie_release_year_model))
RMSE_results %>% knitr::kable()
```

Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301

Movie release Year model reduces RMSE only by 0.003%. Therefore it is not significant to consider it in main model. This implies there is not much release year effect in the model.

5. Time Range model

Time Range denotes the time between the year movie released and the year movie is rated by user. Intermediate time describes how long the user rates movie after being released. As visualized in the data exploration plot, the longer the time range interval gives higher average number of ratings in the model. *Timestamp* variable contains the year user gave rating to particular movie. Extract it using as_datetime() and year() function for both training(*training_edx*) and test(*test_edx*) datasets. Find timerange (Movie rated year by user - movie released year) for both training_edx and test_edx.

```
# Extract the year rated by user for a movie from timestamp column using as_datetime
# function and calculate intermediate time range.
training_edx_1<- training_edx_1 %>% mutate(year_rated = year(as_datetime(timestamp)),
                                           time_range = year_rated - year)
test_edx_1 <- test_edx_1 %>% mutate(year_rated = year(as_datetime(timestamp)),
                                    time_range = year_rated - year)
# Calculate b_tr on training set
time_range_average <- training_edx_1 %>%
  left_join(movie_average,by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  group_by(time_range) %>%
  summarize(b_tr = mean(rating - mu - b_i - b_u))
# How much prediction improves by b_tr term along with b_i and b_u
predicted_value3 <- test_edx_1 %>% left_join(movie_average, by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  left_join(time_range_average,by = "time_range") %>%
  mutate(pred = mu + b_i + b_u + b_tr) %>% pull(pred)
Time_Range_Model <- RMSE(test_edx$rating,predicted_value3)
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Movie & User & Time Range Effect Model",
                                 rmse = Time_Range_Model))
RMSE_results %>% knitr::kable()
```

Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301
Movie & User & Time Range Effect Model	0.8642597

Time range model has slightly improved RMSE value to 0.8642 compared to Model 3. This explains that the time period between movie release year and rated year gives time for users to rate/watch the movies

with the knowledge of classics and well-acclaimed movies all over that time. Therefore, this effect can be included in the model.

6. Time Effect Model

As explored in the plot, date of the rating of a movie has some effects in the model. timestamp variable contains date and time of rating given by user for a particular movie. Extract date of the rating from *timestamp* column using as_date_time() function and convert it into weekly format using round_date() for both *training_edx* and *test_edx*. We can explore by,

```
training_edx_1 <- mutate(training_edx_1, date = as_datetime(timestamp),
                         week =  round_date(date, unit = "week"))
test_edx_1 <- mutate(test_edx_1, date = as_datetime(timestamp),
                     week =  round_date(date, unit = "week"))
# Calculate b_t and predict the model
time_average <- training_edx_1 %>%
  left_join(movie_average,by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  group_by(week) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))
predicted_value4 <- test_edx_1 %>%
  left_join(movie_average, by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  left_join(time_average,by = "week") %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  pull(pred)
Simple_Time_model <- RMSE(test_edx$rating,predicted_value4)
# Tabulate the results
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Movie & User & Time Effect Model",
                                 rmse = Simple_Time_model))
RMSE_results %>% knitr::kable()
```

Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301
Movie & User & Time Range Effect Model	0.8642597
Movie & User & Time Effect Model	0.8645933

RMSE of Time effect model is greater than the Time range model and Movie release year model. So, having large RMSE is not worth adding to the model. This explains the date of rating of movie does not hold any effect on predicting movie ratings.

7. Genre Effect Model - Simply grouping genres

As described in the genres error bar plot, there is a moderate effect of genres without splitting in the model as some genres have more ratings than the others. Lets examine this in our model.

```
# Calculate b_g(genre-bias) by training the model
genre_average <- training_edx %>%
  left_join(movie_average,by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))
# How much prediction improves by adding b_g term along with b_i and b_u in the model?
predicted_value5 <- test_edx %>% left_join(movie_average, by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  left_join(genre_average,by = "genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
Simple_Genre_model <- RMSE(test_edx$rating,predicted_value5)
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Movie, User & Simple Genre Effect Model",
                                 rmse = Simple_Genre_model))
RMSE_results %>% knitr::kable()
```

Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301
Movie & User & Time Range Effect Model	0.8642597
Movie & User & Time Effect Model	0.8645933
Movie, User & Simple Genre Effect Model	0.8643241

Though it reduces RMSE substantially to 0.86432 compared to Time effect model, genres without splitting has approximately over 800 genre combinations which makes the model over-fitting. Therefore, it is not worthy to include this in the model.

8. Genre Effect Model - By Columnizing into wide format

To evaluate genre effect without over-fitting or over-training, lets columnize the *genres* feature into wide format. Avoid "(no genres listed)" genre because of the non-genre information as explained in the model analysis section. This leads to the addition of genre bias(b_gk) to the model where b_gc is the summation of all individual genre columns with respect to genre combination available for movie(0 or 1)

```
# Place the individual genres into column format for both training_edx and test_edx
training_edx_2 <- training_edx_1 %>% separate_rows(genres,sep = "\\|") %>% mutate(option=1)
training_edx_3 <- training_edx_2 %>% spread(genres, option, fill=0) %>%
  select(-"(no genres listed)")
head(training_edx_3)
```

```
## # A tibble: 6 x 29
##   userId movieId rating timestamp title              year year_rated time_range
##    <int>   <dbl>  <dbl>     <int> <chr>             <dbl>      <dbl>      <dbl>
## 1      1     122      5 838985046 Boomerang (1992)   1992       1996          4
## 2      1     292      5 838983421 Outbreak (1995)    1995       1996          1
## 3      1     316      5 838983392 Stargate (1994)    1994       1996          2
```

```
## 4      1     329      5 838983392 Star Trek: Genera~  1994       1996          2
## 5      1     355      5 838984474 Flintstones, The ~  1994       1996          2
## 6      1     356      5 838983653 Forrest Gump (199~  1994       1996          2
## # ... with 21 more variables: date <dttm>, week <dttm>, Action <dbl>,
## #   Adventure <dbl>, Animation <dbl>, Children <dbl>, Comedy <dbl>,
## #   Crime <dbl>, Documentary <dbl>, Drama <dbl>, Fantasy <dbl>,
## #   Film-Noir <dbl>, Horror <dbl>, IMAX <dbl>, Musical <dbl>, Mystery <dbl>,
## #   Romance <dbl>, Sci-Fi <dbl>, Thriller <dbl>, War <dbl>, Western <dbl>
```

```
test_edx_2 <- test_edx_1 %>% separate_rows(genres,sep = "\\|") %>% mutate(op=1)
test_edx_3 <- test_edx_2 %>% spread(genres, op, fill=0)
# Calculate b_gk on the training set
g_av <- training_edx_3 %>%
  left_join(movie_average,by = "movieId") %>%
  left_join(user_average,by = "userId")%>%
  group_by(Action,Adventure,Animation,Children,Comedy,Crime,Documentary,Drama,Fantasy,
           `Film-Noir`,Horror,IMAX,Musical,Mystery,Romance,`Sci-Fi`,
           Thriller,War,Western) %>%
  summarize(b_gk = mean(rating - mu - b_i - b_u))
g_av
```

```
## # A tibble: 797 x 20
## # Groups:   Action, Adventure, Animation, Children, Comedy, Crime, Documentary,
## #   Drama, Fantasy, Film-Noir, Horror, IMAX, Musical, Mystery, Romance, Sci-Fi,
## #   Thriller, War [734]
##    Action Adventure Animation Children Comedy Crime Documentary Drama Fantasy
##     <dbl>     <dbl>     <dbl>    <dbl>  <dbl> <dbl>       <dbl> <dbl>   <dbl>
##  1      0         0         0        0      0     0           0     0       0
##  2      0         0         0        0      0     0           0     0       0
##  3      0         0         0        0      0     0           0     0       0
##  4      0         0         0        0      0     0           0     0       0
##  5      0         0         0        0      0     0           0     0       0
##  6      0         0         0        0      0     0           0     0       0
##  7      0         0         0        0      0     0           0     0       0
##  8      0         0         0        0      0     0           0     0       0
##  9      0         0         0        0      0     0           0     0       0
## 10      0         0         0        0      0     0           0     0       0
## # ... with 787 more rows, and 11 more variables: Film-Noir <dbl>, Horror <dbl>,
## #   IMAX <dbl>, Musical <dbl>, Mystery <dbl>, Romance <dbl>, Sci-Fi <dbl>,
## #   Thriller <dbl>, War <dbl>, Western <dbl>, b_gk <dbl>
```

```
# How much prediction improves by b_gk term along with b_i and b_u
predicted_value6 <- test_edx_3 %>%
  left_join(movie_average, by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  left_join(g_av) %>%
  mutate(pred = mu + b_i + b_u + b_gk) %>%
  pull(pred)
Genre_model <- RMSE(test_edx$rating,predicted_value6)
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model="Movie,User & Genre Model(Column format)",
                                 rmse = Genre_model))
RMSE_results %>% knitr::kable()
```

Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301
Movie & User & Time Range Effect Model	0.8642597
Movie & User & Time Effect Model	0.8645933
Movie, User & Simple Genre Effect Model	0.8643241
Movie,User & Genre Model(Column format)	0.8643241

Though it produces same RMSE as previous model, it is devoid of over-fitting and over-training of dataset. Therefore we can include this effect in our model.

9. Movie, User, Time Range and Genre(Columnize) model

Time range model and Genre effect model by column format slightly improves RMSE as seen in Model 5 & 8. So, add these effects along with user and movie model effects and find whether it improves the model prediction further.

```
predicted_rating10 <- test_edx_3 %>%
  left_join(movie_average, by = "movieId") %>%
  left_join(user_average,by = "userId") %>%
  left_join(time_range_average,by = "time_range") %>%
  left_join(g_av,
            by =c("Action","Adventure","Animation","Children","Comedy","Crime","Documentary",
                  "Drama","Fantasy","Film-Noir","Horror","IMAX","Musical","Mystery",
                  "Romance","Sci-Fi","Thriller","War","Western")) %>%
  mutate(pred = mu + b_i + b_u + b_tr + b_gk) %>%
  pull(pred)
Movie_User_Timerange_genre_model <- RMSE(predicted_rating10, test_edx$rating)
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Movie,User,Timerange & Genre Effect Model",
                                 rmse = Movie_User_Timerange_genre_model))
RMSE_results %>% knitr::kable()
```

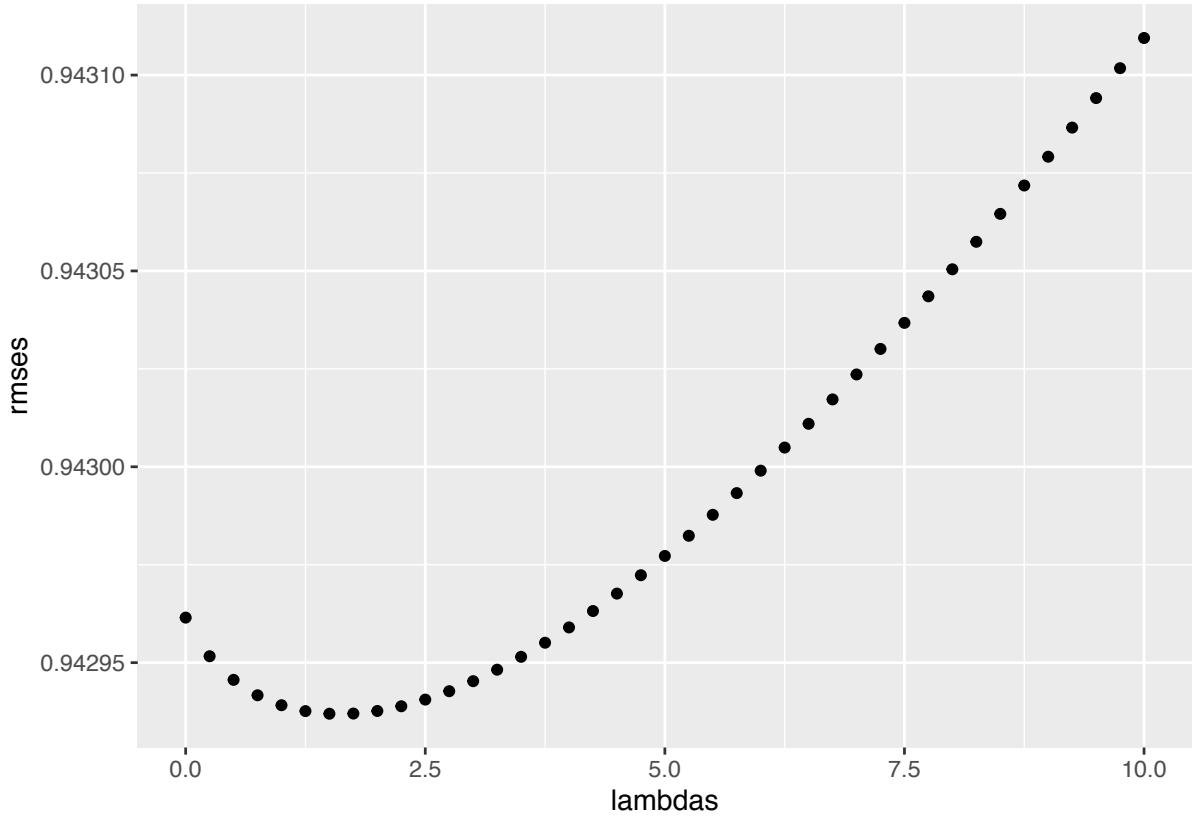
Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301
Movie & User & Time Range Effect Model	0.8642597
Movie & User & Time Effect Model	0.8645933
Movie, User & Simple Genre Effect Model	0.8643241
Movie,User & Genre Model(Column format)	0.8643241
Movie,User,Timerange & Genre Effect Model	0.8639419

On combining all the models(Movie, User, Timerange and Genre effect by column), RMSE decreased to 0.8639419 which is a good improvement in the prediction of movie ratings.

10. Regularisation - Movie Effect

Some movies (good or worst) are rated by few users leading to uncertainty. Instead of minimizing least squares equation, regularization penalizes large estimates by adding penalty term to minimize the equation. Using cross validation, choose lambda.Lets compute regularization for the estimate of movie effects:

```
lambdas <- seq(0, 10, 0.25)
mu <- mean(training_edx$rating)
# Calculate regularized movie effect,b_i and return RMSE for respective lambdas
movie_average <- training_edx %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
just_the_sum <- training_edx %>%
  group_by(movieId) %>%
  summarize(s_i = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_value7 <- join(test_edx,just_the_sum,by = "movieId",type = "left") %>%
    mutate(b_i = s_i/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_value7, test_edx$rating))
})
# Plot different values of lambda with their RMSEs.
qplot(lambdas, rmses)
```




```
# Find the optimal lambda which gives minimum RMSE value
lambda = lambdas[which.min(rmses)]
lambda
```

```
## [1] 1.5
```

```
# Find the minimum RMSE value
Regularised_Movie_model <- min(rmses)
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Regularised Movie Effect Model",
                                 rmse = Regularised_Movie_model))
RMSE_results %>% knitr::kable()
```

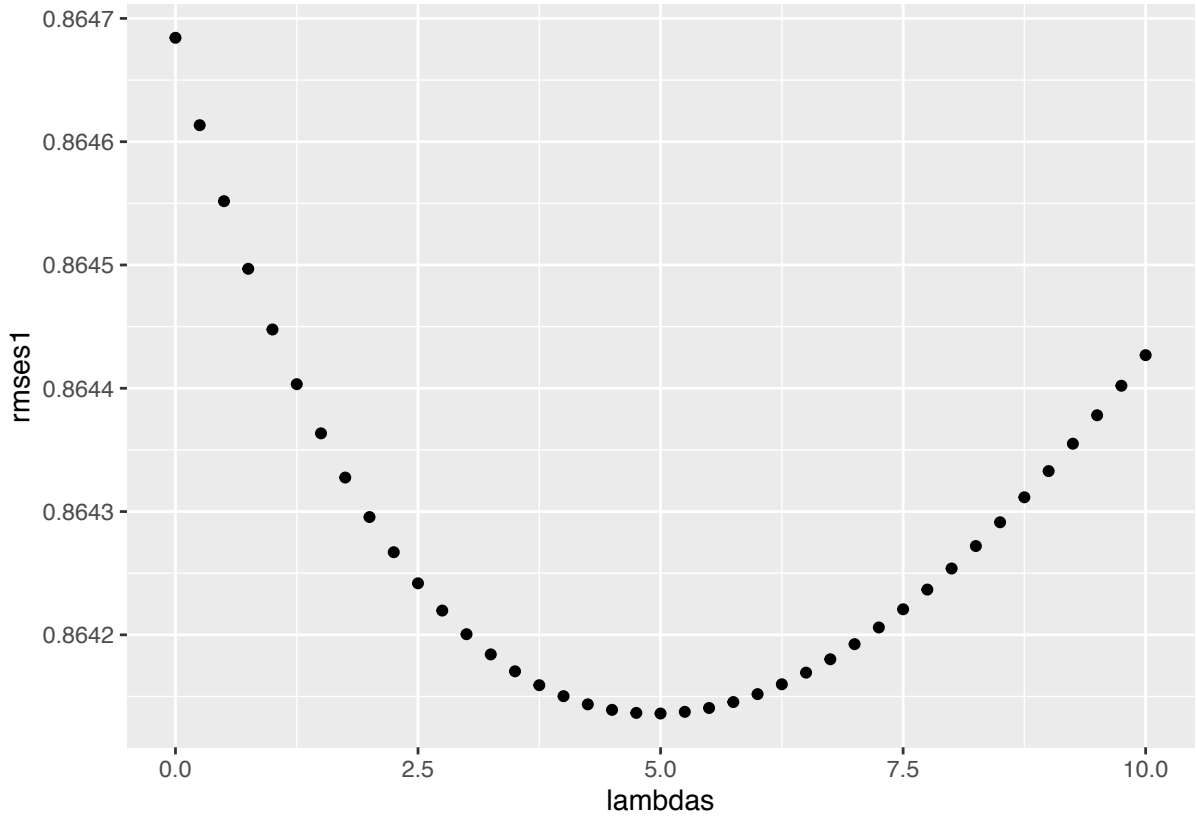
Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301
Movie & User & Time Range Effect Model	0.8642597
Movie & User & Time Effect Model	0.8645933
Movie, User & Simple Genre Effect Model	0.8643241
Movie,User & Genre Model(Column format)	0.8643241
Movie,User,Timerange & Genre Effect Model	0.8639419
Regularised Movie Effect Model	0.9429370

By penalized movie estimates, there is not much improvement in RMSE compared to least squares estimates(Model 2). Therefore, it is not necessary to include this effect in the model.

11. Regularised Movie and User Effect Model

Since some users rate differently, Regularization is used for estimating user effects along with movie effects.Consider a window of size to select optimal lambda,λ by cross validation.Here, lambda is a tuning parameter. Lets compute regularization for the estimate of movie and user effects:

```
lambdas <- seq(0, 10, 0.25)
# Calculate regularized movie and user effects,b_i and b_u and return RMSE for respective lambdas
rmses1 <- sapply(lambdas, function(l){
  mu <- mean(training_edx$rating)
  b_i <- training_edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- merge(x=training_edx,y=b_i,by = "movieId",all.x = TRUE) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted3<- merge(x=test_edx,y=b_i,by = "movieId",all.x = TRUE)
  predicted4 <- merge(x=predicted3,y=b_u,by = "userId",all.x = TRUE)
  predicted_value8 <- predicted4 %>% mutate(pred = mu + b_i + b_u) %>% pull(pred)
  return(RMSE(predicted_value8, test_edx$rating))
})
# Plot different values of lambda with their RMSEs
qplot(lambdas, rmses1)
```




```
# Find the optimal lambda which gives minimum RMSE value
lambda <- lambdas[which.min(rmses1)]
lambda
```

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

```
# Use this lambda to find the required RMSE
reg_b_i <- training_edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
reg_b_u <- merge(x=training_edx,y=reg_b_i,by = "movieId",all.x = TRUE) %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
predicted5<- merge(x=test_edx,y=reg_b_i,by = "movieId",all.x = TRUE)
predicted6 <- merge(x=predicted5,y=reg_b_u,by = "userId",all.x = TRUE)
predicted_value9 <- predicted6 %>% mutate(pred = mu + b_i + b_u) %>% pull(pred)
Regularised_Movie_User_model <- RMSE(predicted_value9, test_edx$rating)
# Tabulate the results
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Regularised Movie & User Effect Model",
                                 rmse = Regularised_Movie_User_model))
RMSE_results %>% knitr::kable()
```

Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301
Movie & User & Time Range Effect Model	0.8642597
Movie & User & Time Effect Model	0.8645933
Movie, User & Simple Genre Effect Model	0.8643241
Movie,User & Genre Model(Column format)	0.8643241
Movie,User,Timerange & Genre Effect Model	0.8639419
Regularised Movie Effect Model	0.9429370
Regularised Movie & User Effect Model	0.8641362

From the plot, we visualize the optimal lambda,λ which gives minimum RMSE value. From the cross-validation, it is clear optimal lambda is 5. By penalized movie and user estimates,RMSE has slightly decreased(from 0.864684 to 0.864136). Since it reduces RMSE drastically, it is clear to include this in our model.

12. Matrix Factorization with parallel stochastic gradient descent

Our previous models fails to describe the similarity pattern between users to users or movies to movies or users to movies. Here, Matrix factorization discovers these patterns based on the residuals of the movie and user effects and convert it into matrix form by *recosystem* package.

Training and test data is built with *training_edx* and *test_edx* datasets into a sparse matrix triplet form consisting of userId, movieId and rating. Each line in the file consists of three values(user_index, item_index and rating).

```
# Convert the train and test sets into recosystem input format
set.seed(1, sample.kind = "Rounding")
train_data <-  with(training_edx, data_memory(user_index = userId,
                                              item_index = movieId,
                                              rating     = rating))
test_data <- with(test_edx, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating     = rating))
# Create the model object
r <-  recosystem::Reco()
# Select the best tuning parameters
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l1 = 0,
                                       costq_l1 = 0,
                                       nthread  = 1, niter = 10))
```

In tune() function, the tuning parameters used are dim to specify the number of latent factors, lrate(learning rate) as step-size in gradient descent, costp_l2 as L2 regularization cost for user factors, costq_l2 as l2 regularization cost for movie factors, nthread is number of threads for parallel computing and niter is number of iterations.

```
# Train the algorithm
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9827   1.1044e+07
##    1       0.8749   8.9699e+06
##    2       0.8423   8.3361e+06
##    3       0.8208   7.9554e+06
##    4       0.8042   7.6934e+06
##    5       0.7916   7.4995e+06
##    6       0.7813   7.3527e+06
##    7       0.7728   7.2321e+06
##    8       0.7659   7.1429e+06
##    9       0.7600   7.0658e+06
##   10       0.7546   7.0013e+06
##   11       0.7499   6.9483e+06
##   12       0.7456   6.8992e+06
##   13       0.7416   6.8558e+06
##   14       0.7380   6.8180e+06
##   15       0.7346   6.7842e+06
##   16       0.7315   6.7533e+06
##   17       0.7285   6.7252e+06
##   18       0.7258   6.7006e+06
##   19       0.7233   6.6778e+06
```

```
# Predict the values
y_hat <-  r$predict(test_data, out_memory())
Matrix_Factorization_model <- RMSE(test_edx$rating,y_hat)
# Tabulate the results
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Matrix Factorization Model",
                                 rmse = Matrix_Factorization_model))
RMSE_results %>% knitr::kable()
```

Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301
Movie & User & Time Range Effect Model	0.8642597
Movie & User & Time Effect Model	0.8645933
Movie, User & Simple Genre Effect Model	0.8643241
Movie,User & Genre Model(Column format)	0.8643241
Movie,User,Timerange & Genre Effect Model	0.8639419
Regularised Movie Effect Model	0.9429370
Regularised Movie & User Effect Model	0.8641362
Matrix Factorization Model	0.7861999

The train() function trains the recommender model by using parameters and options defined. $predict() functions predict the unknown entries in the rating matrix. For final prediction, predicted residuals are added with base prediction of Model 9 and RMSE is calculated. It produces RMSE of 0.7861999. This is the lower RMSE ever achieved in this project.

13. Final Model using Matrix Factorization(against validation set)

Matrix factorization with parallel stochastic gradient descent approach gives lowest RMSE compared to other models. Hence, Matrix factoriztion is used as a final model against validation set to predict movie rating. Here, we use the entire *edx* dataset for training against validation(testing) set.

Matrix Factorization uses edx as training and validation as testing and convert it into a sparse matrix triplet form of userId, movieId and rating. The model is tuned by setting appropriate tuning parameters and trained with it.

```
# Convert the train and test sets into recosystem input format
set.seed(1, sample.kind = "Rounding")
train_data1 <-  with(edx, data_memory(user_index = userId,
                                      item_index = movieId,
                                      rating     = rating))
test_data1 <- with(validation, data_memory(user_index = userId,
                                       item_index = movieId,
                                       rating     = rating))
# Create the model object
r <-  recosystem::Reco()
# Select the best tuning parameters
opts <- r$tune(train_data1, opts = list(dim = c(10, 20, 30),
                                        lrate = c(0.1, 0.2),
                                        costp_l1 = 0,
                                        costq_l1 = 0,
                                        nthread  = 4, niter = 10))
```

```
# Train the algorithm
r$train(train_data1, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9729   1.2025e+07
##    1       0.8720   9.8755e+06
##    2       0.8385   9.1692e+06
##    3       0.8162   8.7426e+06
##    4       0.8008   8.4663e+06
##    5       0.7889   8.2687e+06
##    6       0.7790   8.1147e+06
##    7       0.7709   7.9947e+06
##    8       0.7640   7.8976e+06
##    9       0.7582   7.8163e+06
##   10       0.7532   7.7530e+06
##   11       0.7488   7.6958e+06
##   12       0.7447   7.6475e+06
##   13       0.7411   7.6047e+06
##   14       0.7378   7.5683e+06
##   15       0.7347   7.5313e+06
##   16       0.7318   7.5018e+06
##   17       0.7292   7.4743e+06
##   18       0.7268   7.4506e+06
##   19       0.7245   7.4276e+06
```

```
# Predict the values
y_hat1 <-  r$predict(test_data1, out_memory())
Final_Matrix_Factorization_model <- RMSE(validation$rating,y_hat1)
# Tabulate the results
RMSE_results <- bind_rows(RMSE_results,
                          tibble(Model = "Final model using Matrix Factorization",
                                 rmse = Final_Matrix_Factorization_model))
RMSE_results %>% knitr::kable()
```

Model	rmse
Simple Average Model	1.0600537
Movie Model	0.9429615
Movie & User Effect Model	0.8646843
Movie & User & Year Effect Model	0.8643301
Movie & User & Time Range Effect Model	0.8642597
Movie & User & Time Effect Model	0.8645933
Movie, User & Simple Genre Effect Model	0.8643241
Movie,User & Genre Model(Column format)	0.8643241
Movie,User,Timerange & Genre Effect Model	0.8639419
Regularised Movie Effect Model	0.9429370
Regularised Movie & User Effect Model	0.8641362
Matrix Factorization Model	0.7861999
Final model using Matrix Factorization	0.7825243

So, the final model using Matrix Factorization produces RMSE of about 0.7825243. Therefore, RMSE produced by final model against validation set is less than 0.86490 as similar to project instructions.

Conclusion

By summing-up the prediction of different models, it is concluded that Matrix Factorization with parallel stochastic gradient descent against validation reduces the RMSE drastically to 0.7825243. Thus, Movielens project helps us to create better recommendation system algorithms to predict rating for a movie by user. Using 10M of the movielens dataset, we split the data into edx and validation for training and testing the model by Regression model, Regularization and Matrix Factorization approach. From Simple Average method which gives RMSE of about 1, RMSE is reduced to 0.8646 by modeling Movie and User effects. Further , Regularised Movie and User Effect reduces the RMSE to 0.8641 and Matrix Factorization using recosystem reduces the RMSE all the way lower to 0.7861999. Since Matrix Factorization makes predictions based on users past preferences to movies and finding a similarity pattern between them and use of a sparse matrix for a large dataset, it appears as a strong model for predicting movie ratings thereby reducing RMSE to a lower level compared to others. For future works, one can use Restricted Boltzmann machines , which is another latent factor approach in prediction like Matrix Factorization to improve the model. Also, neighbourhood models such as item- item and user-user approach can be used in predicting movie rating. As a final choice, one can work in the future with ensemble methods by combining different algorithms thereby exploiting the individual strength of each model to make predictions.