Recommendation System in R

Veneet Bhardwaj

1/28/2021

Contents

1 Introduction

Recommendation systems are part of our daily lives now. From music, books and movies streaming/content delivery to online shopping to restaurants recommendation, most of us use application which have recommendation systems at their core.

Recommendation systems apply statistical tools to the problem of product recommendations based on historical data.

These systems help in:

- improve conversion rates
- promote cross-selling
- improve customer retention

Thus leading to increased profitability for business.

This project is on making a movie recommendation system using R, with the *movielens* data.

The data set used for this project is the 10M version. The movielens data set is generated and maintained by GroupLens.

MovieLens 10M dataset:

- https://grouplens.org/datasets/movielens/10m/
- http://files.grouplens.org/datasets/movielens/ml-10m.zip

1.1 Approach

The approach used by the recommendation systems can be collaborative approach and/or content-based. A combination of both can also be used.

1.1.1 Collaborative Approach

This method is based on the user's past behavior and similar choices made by other users. These data points are then used to predict the items or ratings for items that the user may have an interest in. This method does not rely on the knowledge about the item itself.

The scope of this method is limited by:

- **ColdStart** : requires significant prior information on the user/items to make accurate recommendations.
- **Scalabilty** : The larger the data the more the computation power required to make accurate recommendations.
- **Sparsity** : Not all items will be rated and not all users will rate all items they use.

1.1.2 Content-based Approach

This method uses a set of discrete characteristics of an item in order to recommend items with similar properties. Thus this method needs little information to start with but is limited to the information on the item itself.

1.2 Objective

The Objective of this project is to use the 10M movielens data set, build a model to predict the ratings and get the loss function **RMSE** (root mean squared error) to less than **0.8649**

2 Analysis

2.1 Setting up the data

Initialization of the libraries that will be used

```
library(knitr)
library(markdown)
options(round = 5)
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
if(!require(recommenderlab)) install.packages("recommenderlab",
                                              repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate",
                                         repos = "http://cran.us.r-project.org")
if(!require(Matrix)) install.packages("Matrix",
                                      repos = "http://cran.us.r-project.org")
```

The Movielens Data set is downloaded using the following code

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)

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

# 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")
```

Create the edx and the final holdout validation set

```
# 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,]
temp <- movielens[test_index,]
```

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

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

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

The edx data table has 9000055 observations and 6 features The validation data table as 999999 observations of the same 6 features. Both these data tables have the same set of movieId and userId.

2.2 Observations and features

```
summary(edx)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

- **userId** is the unique user identification
- **movieId** is the unique movie identification

```
edx %>% summarise(unique_users = n_distinct(userId),
                  unique_movies = n_distinct(movieId))
```

```
##   unique_users unique_movies
## 1        69878         10677
```

- **rating** are the ratings given by the users on the movies and theoretically vary between 0 and 5, where 0 is lowest and 5 is the highest

```
range(edx$rating)
```

```
## [1] 0.5 5.0
```

The lowest rating that has been given by any user in our data set is 0.5 and the highest is 5.0.

- **timestamp** is the timestamp of the rating.

```
range(as_datetime(edx$timestamp))
```

```
## [1] "1995-01-09 11:46:49 UTC" "2009-01-05 05:02:16 UTC"
```

Ratings are from 1995 till 2009

- **title** is the name of the movie and also contains the year of the make of the movie.

```
head(edx$title)
```

```
## [1] "Boomerang (1992)"              "Net, The (1995)"
## [3] "Outbreak (1995)"               "Stargate (1994)"
## [5] "Star Trek: Generations (1994)" "Flintstones, The (1994)"
```

- **genres** are the genres of the movies. Some movies have been categorized into multiple genres separated by |.

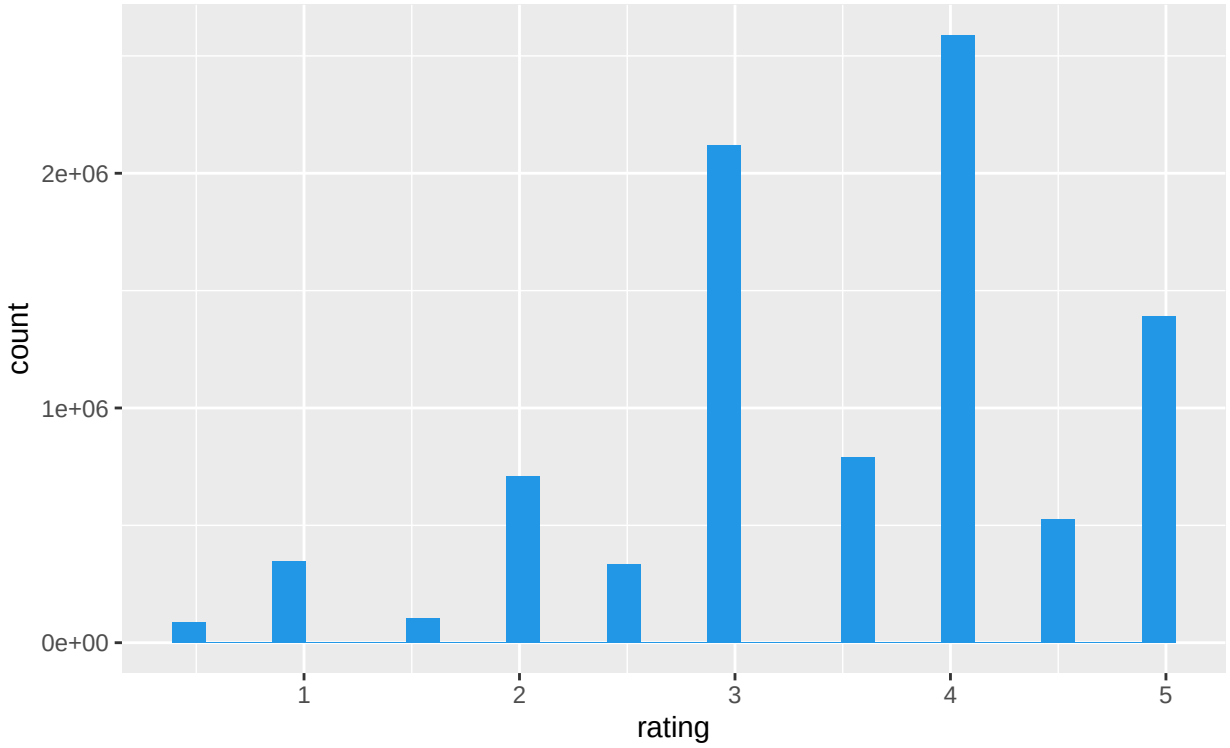
```
head(edx$genres)
```

```
## [1] "Comedy|Romance"                "Action|Crime|Thriller"
## [3] "Action|Drama|Sci-Fi|Thriller"  "Action|Adventure|Sci-Fi"
## [5] "Action|Adventure|Drama|Sci-Fi" "Children|Comedy|Fantasy"
```

2.3 Exploration of data

2.3.1 Rating distribution

```
edx %>% group_by(rating) %>%
  ggplot(aes(rating)) + geom_histogram(fill=4, bins=30)
```



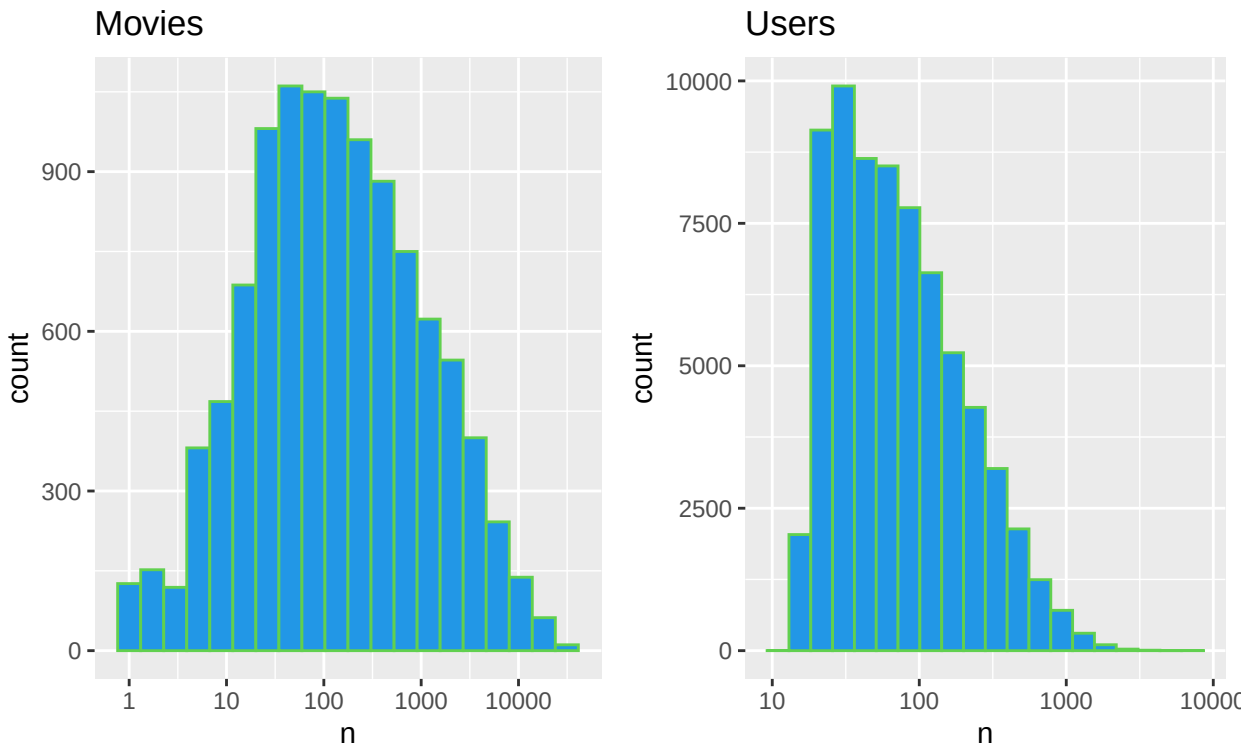
The mean of the ratings is 3.512 and the median is 4.0 (from the summary of edx)

Observations:

- No user has given a rating of 0
- The are fewer ratings of half stars than full stars
- The most common ratings are 4, 3, 5, 3.5 and 2

2.3.2 User, Movie distribution

```
p1 <- edx %>% dplyr::count(movieId) %>%
  ggplot(aes(n)) + geom_histogram(bins = 20, color=3, fill=4) +
  scale_x_log10() + ggtitle("Movies")
p2 <- edx %>% dplyr::count(userId) %>%
  ggplot(aes(n)) + geom_histogram(bins = 20, color=3, fill=4) +
  scale_x_log10() + ggtitle("Users")
gridExtra::grid.arrange(p1,p2, ncol=2)
```



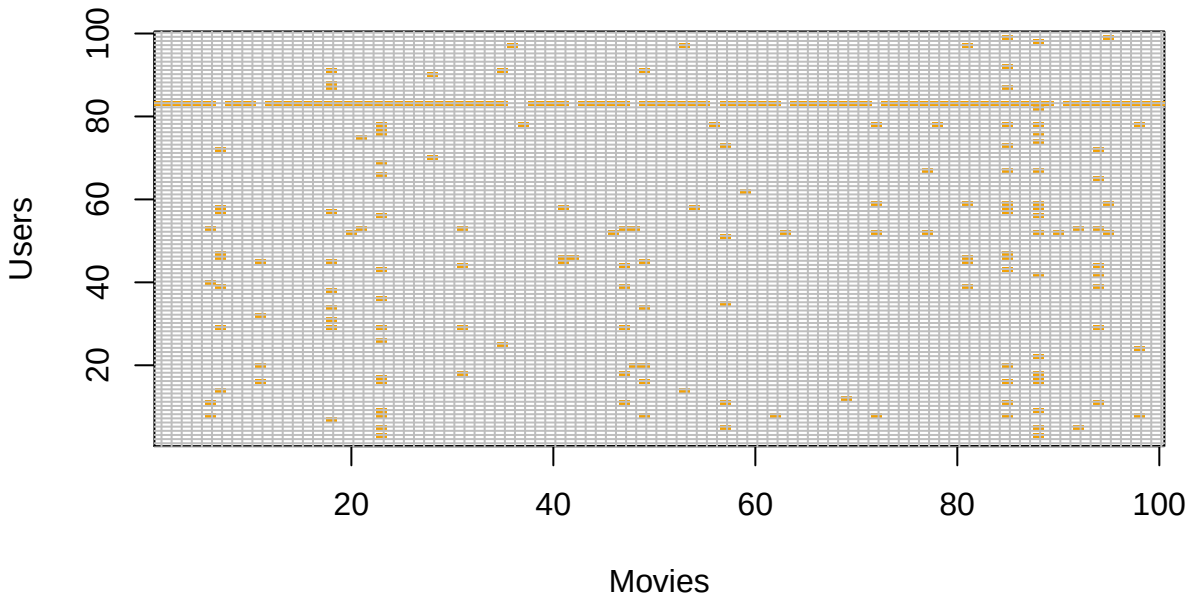
Observations:

- Some movies get rated more than the others. (The Blockbusters!)
- Some movies have not been rated at all.
- Some users are more active than others.
- All users in our data set have rated at least 10 movies (but is that enough?).

This does make it a bit difficult to recommend a movie, given that no one has rated it.

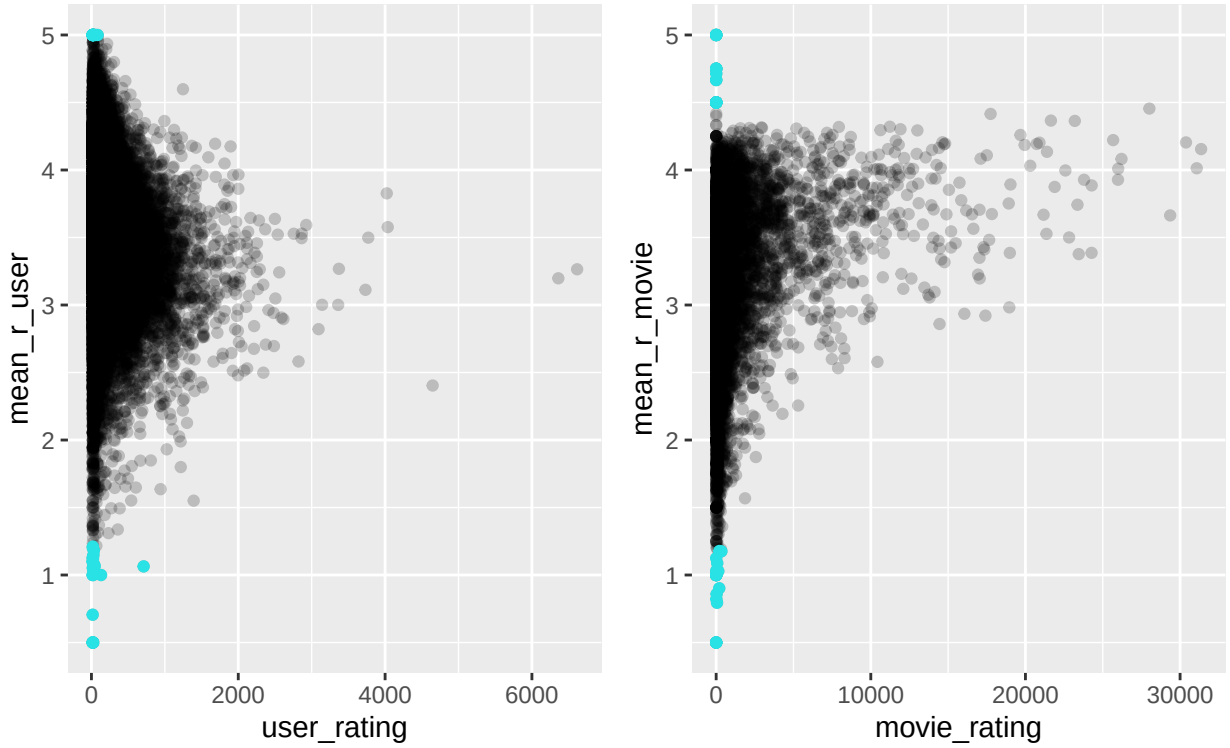
Sparsity of the data set is visualized below using a matrix of a random sample of 100 movies and 100 users with colored indicating a user/movie combination for which there is a rating.

```
users <- sample(unique(edx$userId), 100)
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users") %>%
  abline(h=0:100+0.2, v=0:100+0.2, col = "grey")
```



Ranking of the movies and users by the number of ratings, should show the disparity in the user ratings and the movie ratings.

```
temp <- edx %>% group_by(userId) %>%
  summarize(user_rating = n(), mean_r_user = mean(rating))
temp <- temp %>% mutate(rank = rank(-mean_r_user) )
p1 <- temp %>%
  ggplot(aes(user_rating, mean_r_user)) +
  geom_point(alpha=0.2) +
  geom_point(data = filter(temp, rank <= 20 | rank >= 69856), col=13)

tempm <- edx %>% group_by(movieId) %>%
  summarize(movie_rating = n(), mean_r_movie = mean(rating))
tempm <- tempm %>% mutate(rank = rank(-mean_r_movie))
p2 <- tempm %>% ggplot(aes(movie_rating, mean_r_movie)) +
  geom_point(alpha=0.2) +
  geom_point(data = filter(tempm, rank <= 20 | rank >= 10655), col=13)
gridExtra::grid.arrange(p1,p2,ncol=2)
```



Observations:

- Some users have very few mean ratings, especially very low (less than 1) or very high (a five)
- Some movies have very high mean ratings (a five) given very few ratings
- Some movie have a very low mean ratings (less than 1) given very few ratings

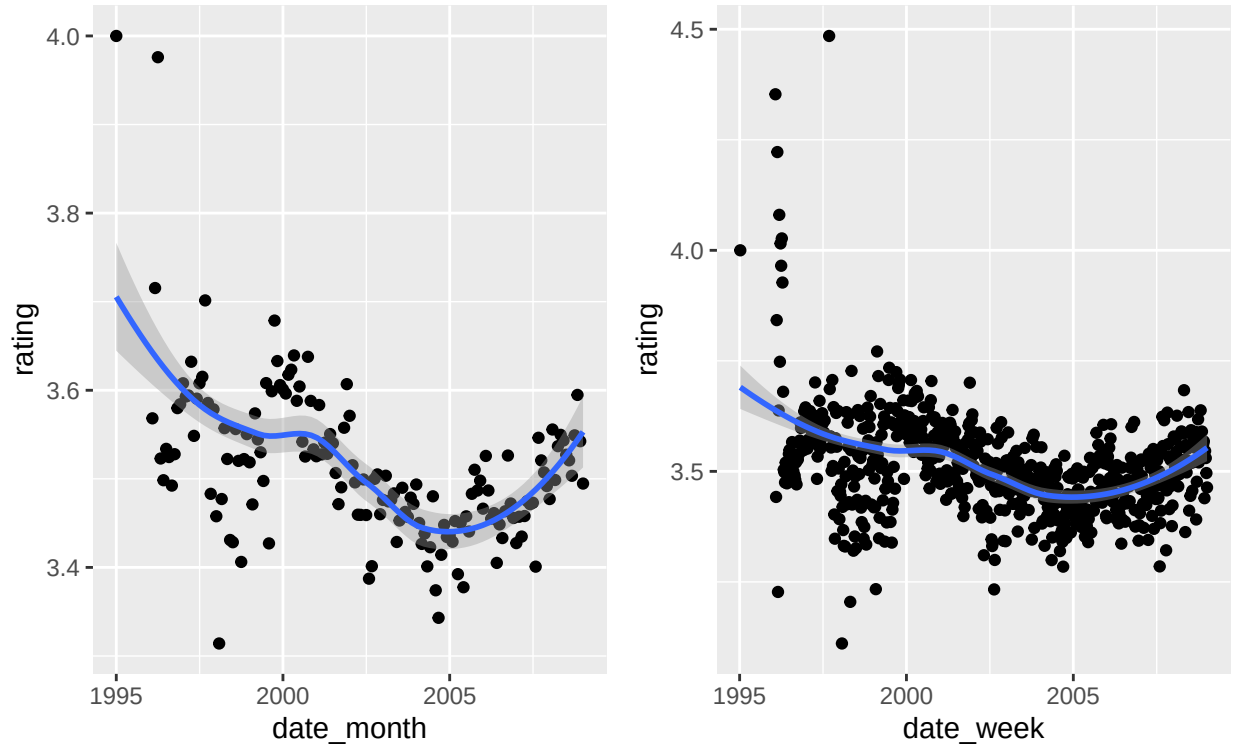
These high mean with few ratings and Low mean with few ratings might be attributed to the fact that some movies have very minimal set of viewers. These movies, given that we do not have further information on the content of the movies themselves, might create a biased overall rating based recommendation model.

2.3.3 Timestamp

The timestamp is converted to a datetime format and the month and week of the rating is used for visualization.

```
edx_m <- edx %>% mutate(date_stamp = as_datetime(timestamp))
p1 <- edx_m %>% mutate(date_month = round_date(date_stamp, unit = "month")) %>%
  group_by(date_month) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date_month, rating)) +
  geom_point() +
  geom_smooth()
p2 <- edx_m %>% mutate(date_week = round_date(date_stamp, unit = "week")) %>%
  group_by(date_week) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date_week, rating)) +
  geom_point() +
```

```
  geom_smooth()
gridExtra::grid.arrange(p1,p2, ncol=2)
```



Observation:

- There is evidence of the time effect in the plot.

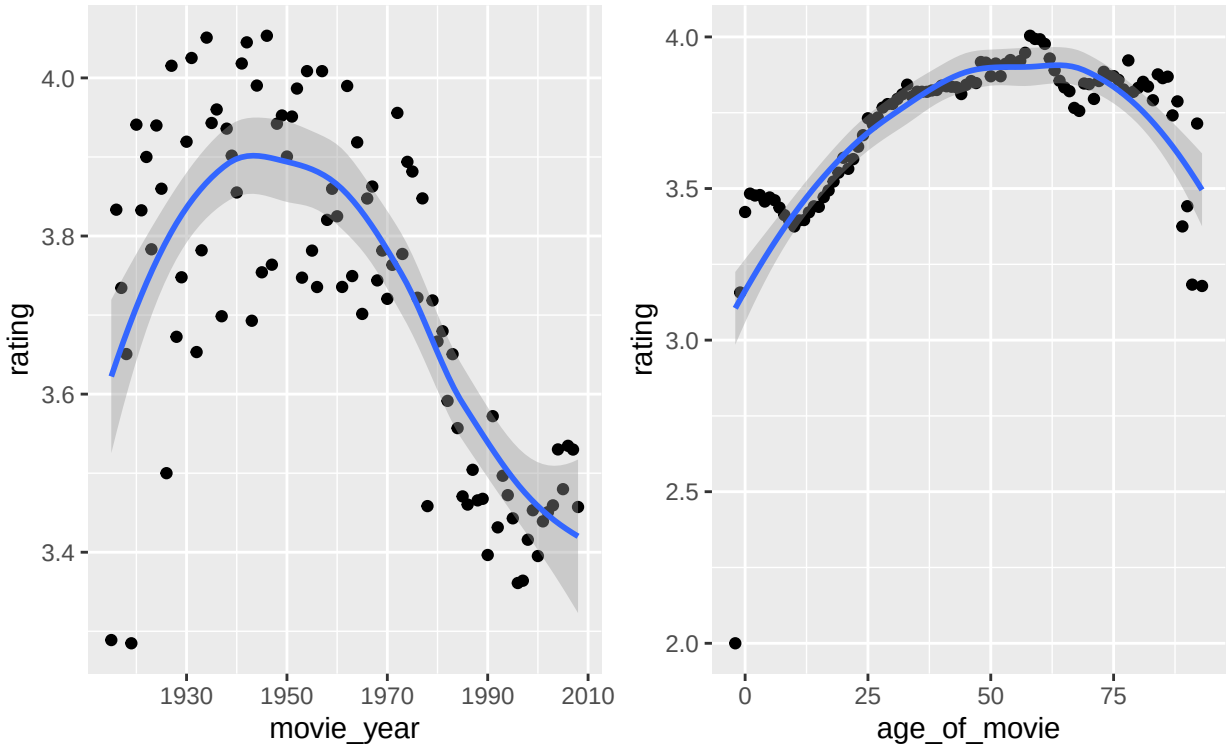
The title contains the make of the movie and from the timestamp one can get the year when the movie was watched/rated. Thus from combination of these 2 year stamps the age (how old was the movie when being watched) can be derived. The same can be defined as

$$Age_{Movie} = Year_{rated} - Year_{make}$$

```
# the str_sub function is used to get the last 6 characters from the title
# and the str_remove_all is used to remove the () from the extracted string.
edx_m <- edx %>%
  mutate( movie_year = as.numeric(str_remove_all(str_sub(title, -6),"[^[:alnum:]]")),
          date_stamp = as_datetime(timestamp),
          year_t = year(date_stamp))
edx_m <- edx_m %>% mutate(age_of_movie = as.factor(year_t - movie_year))
```

Visualizing the Age of movie effect.

```
p1 <- edx_m %>% group_by(movie_year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(movie_year, rating)) +
  geom_point() +
  geom_smooth()
p2 <- edx_m %>% mutate(age_of_movie = year_t - movie_year) %>%
  group_by(age_of_movie) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(age_of_movie, rating)) +
  geom_point() +
  geom_smooth()
gridExtra::grid.arrange(p1,p2, ncol=2)
```



Observations:

- There is some effect that the movie year has on the rating of the movie itself
- The age of movie as defined above seems to have an effect as well.

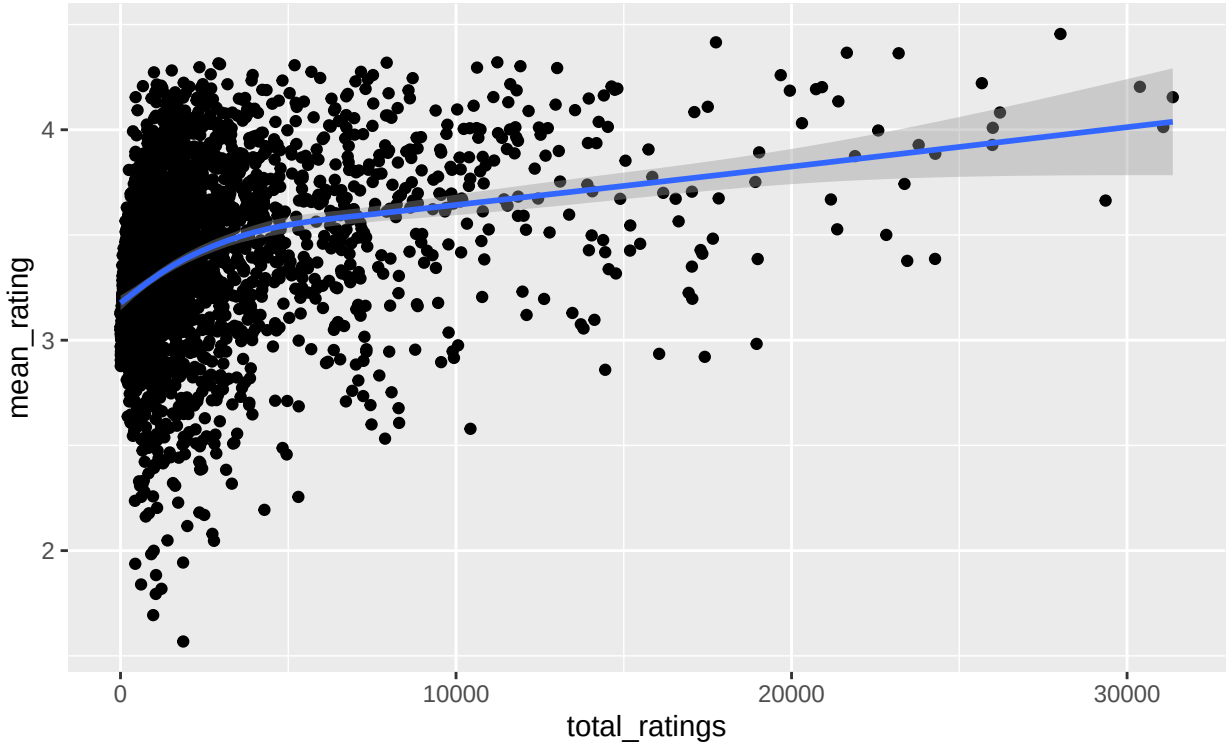
2.3.4 Number of ratings

The charts above might be representing something of the blockbuster effect of the movies itself. Highly rated movies have a higher entertainment value and thus get rated more and higher!

To check this lets define another variable, the total number of ratings

Total number of ratings = Count of the number of ratings for every movie

```
# adding the total_ratings to the data set
edx_m <- edx %>% group_by(movieId) %>% mutate(total_ratings = n())
edx_m <- data.frame(edx_m)
#visualizing
edx_m %>% group_by(movieId) %>%
  mutate(total_ratings = n()) %>%
  group_by(total_ratings) %>%
  summarize(mean_rating = mean(rating)) %>%
  ggplot(aes(total_ratings, mean_rating)) +
  geom_point() +
  geom_smooth()
```



Observation:

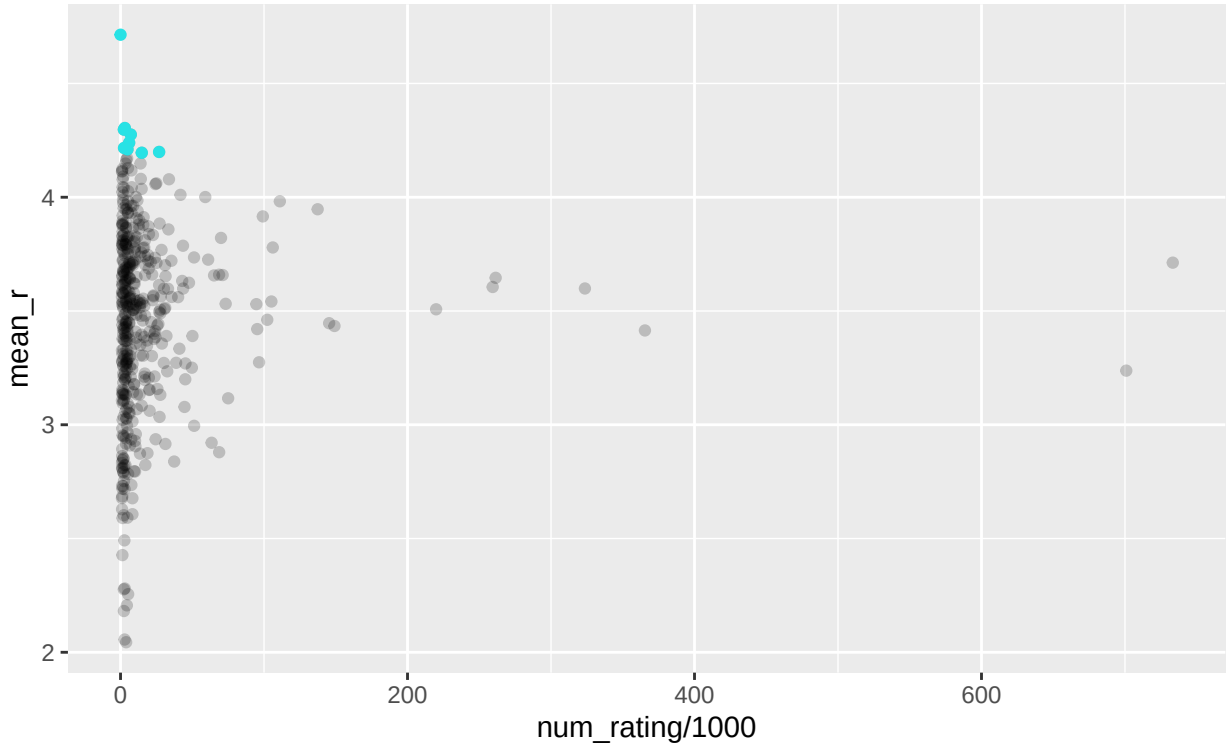
- The total number of previous ratings does have some effect on the overall rating. This might have something to do with the intrinsic value of the movie and might be something that is already accommodated in the movie-user interaction.

2.3.5 Genre distribution

Movies, in the data set, have been categorized into multiple genres separated by |, with a few exceptions of no categorization. Using rank (sorting) of the genres

```
temp <- edx %>% group_by(genres) %>%
  summarize(num_rating = n(), mean_r = mean(rating))
temp <- temp %>% mutate(rank = rank(-mean_r) )
```

```
temp %>% filter( num_rating >= 1000) %>%
  ggplot(aes(num_rating/1000, mean_r)) +
  geom_point(alpha=0.2) +
  geom_point(data = filter(temp, rank <= 10), col=13)
```




```
temp %>% arrange(desc(num_rating))
```

```
## # A tibble: 797 x 4
##    genres                    num_rating mean_r  rank
##    <chr>                          <int>  <dbl> <dbl>
##  1 Drama                         733296   3.71   188
##  2 Comedy                        700889   3.24   532
##  3 Comedy|Romance                365468   3.41   414
##  4 Comedy|Drama                  323637   3.60   269
##  5 Comedy|Drama|Romance          261425   3.65   235
##  6 Drama|Romance                 259355   3.61   265
##  7 Action|Adventure|Sci-Fi       219938   3.51   346
##  8 Action|Adventure|Thriller     149091   3.43   400
##  9 Drama|Thriller                145373   3.45   390
## 10 Crime|Drama                   137387   3.95    61
## # ... with 787 more rows
```

Observations:

- There are some genres of movies that get rated higher than the others.
- There are some genres of movies that are watched/number of ratings more than others.

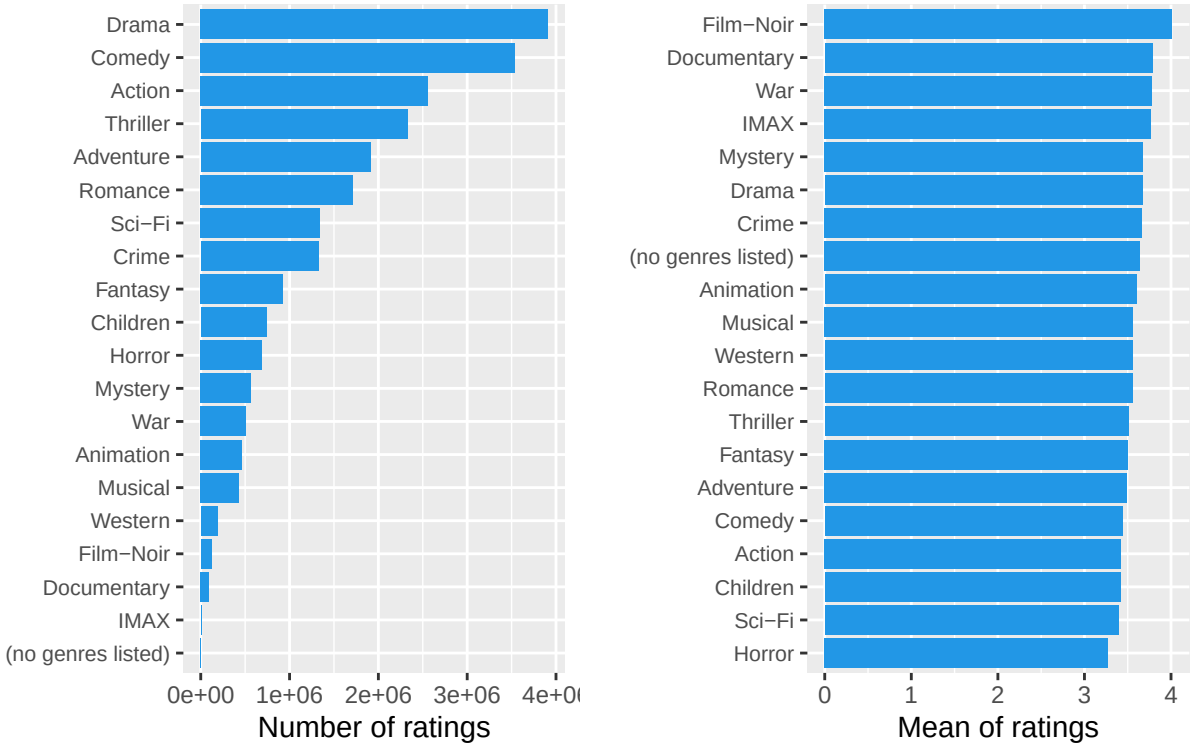
Further some movies are classified into more than one genres. To observe the impact of individual genres, splitting of the genres should help.

```
#this does take some time to complete!
#using separate_rows with | as a separator and summarising the count and the means.
genres_list <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n(), mean_g = mean(rating)) %>%
  arrange(desc(count))
#used for initializing faster
write.csv(genres_list, "genres_list.csv")
```

```
#genres_list csv uploaded along with the files
genres_list <- read.csv("genres_list.csv")
genres_list
```

```
##     X             genres   count   mean_g
## 1   1              Drama 3910127 3.673131
## 2   2             Comedy 3540930 3.436908
## 3   3             Action 2560545 3.421405
## 4   4           Thriller 2325899 3.507676
## 5   5          Adventure 1908892 3.493544
## 6   6            Romance 1712100 3.553813
## 7   7             Sci-Fi 1341183 3.395743
## 8   8              Crime 1327715 3.665925
## 9   9            Fantasy  925637 3.501946
## 10 10           Children  737994 3.418715
## 11 11             Horror  691485 3.269815
## 12 12            Mystery  568332 3.677001
## 13 13                War  511147 3.780813
## 14 14          Animation  467168 3.600644
## 15 15            Musical  433080 3.563305
## 16 16            Western  189394 3.555918
## 17 17          Film-Noir  118541 4.011625
## 18 18        Documentary   93066 3.783487
## 19 19               IMAX    8181 3.767693
## 20 20 (no genres listed)       7 3.642857
```

```
p1 <- genres_list %>%
  ggplot(aes(reorder(genres,count),count)) +
  geom_bar(stat="identity", fill= 4) +
  coord_flip() +
  theme(axis.text.y = element_text(size=8)) +
  ylab("Number of ratings") + xlab('')
p2 <- genres_list %>%
  ggplot(aes(reorder(genres,mean_g),mean_g)) +
  geom_bar(stat="identity", fill=4) +
  coord_flip() +
  theme(axis.text.y = element_text(size=8)) +
  ylab("Mean of ratings") + xlab('')
gridExtra::grid.arrange(p1,p2,ncol=2)
```



Observations:

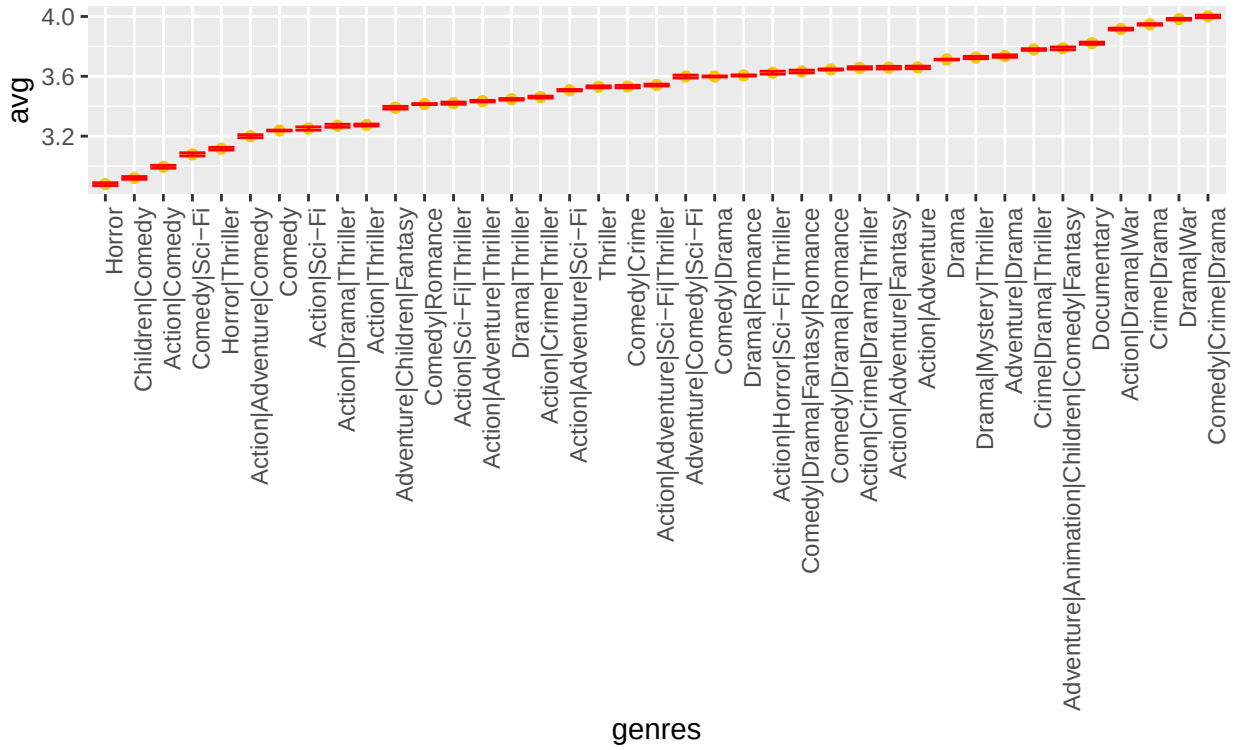
- Some movies have no genres
- The mean of the rating might be higher for genres with very few movies in them.
- The variation in the mean of the ratings is ~ 0.8

```
range(genres_list$mean_g)
```

```
## [1] 3.269815 4.011625
```

Visualization of the combined classification as is in the data set (limiting data by filtering genres with more than 42000 movies listed in them)

```
edx_m %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 42000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point(color=7) +
  geom_errorbar(color="red") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Observation:

- Using the combined genres classification as in the data set does reveal evidence of the effect of the genres on the ratings.

```
rm(edx_m, genres_list, p1, p2, temp)
```

2.3.6 Similarity of movie, user ratings

There is an indication from the analysis above that there is a structure in the movies that, though partially explained by the genres, is more complicated than a Drama or Action or Comedy or their combinations.

An approach to solving this problem is using the similarity matrix or the distance measure. This approach solves the problem of ColdStart, Scalabilty and Sparsity thus improving prediction accuracy.

Having said that, due to the size of the data and the resultant matrix and given the limited hardware capabilities we use the recommenderlab package infrastructure to create a sparse matrix from the data set.

The realRatingMatrix object stores the data in sparse format. (i.e only non-NA values are stored explicitly and the NA values are represented by a dot)

```
edx_sub <- edx %>% select(userId, movieId, rating)
edx_sub <- as.data.frame(edx_sub)
#using the recommenderlab package
#Create realRatingMatrix
rrm <- as(edx_sub, "realRatingMatrix")
#to normalize the matrix
class(rrm)
```

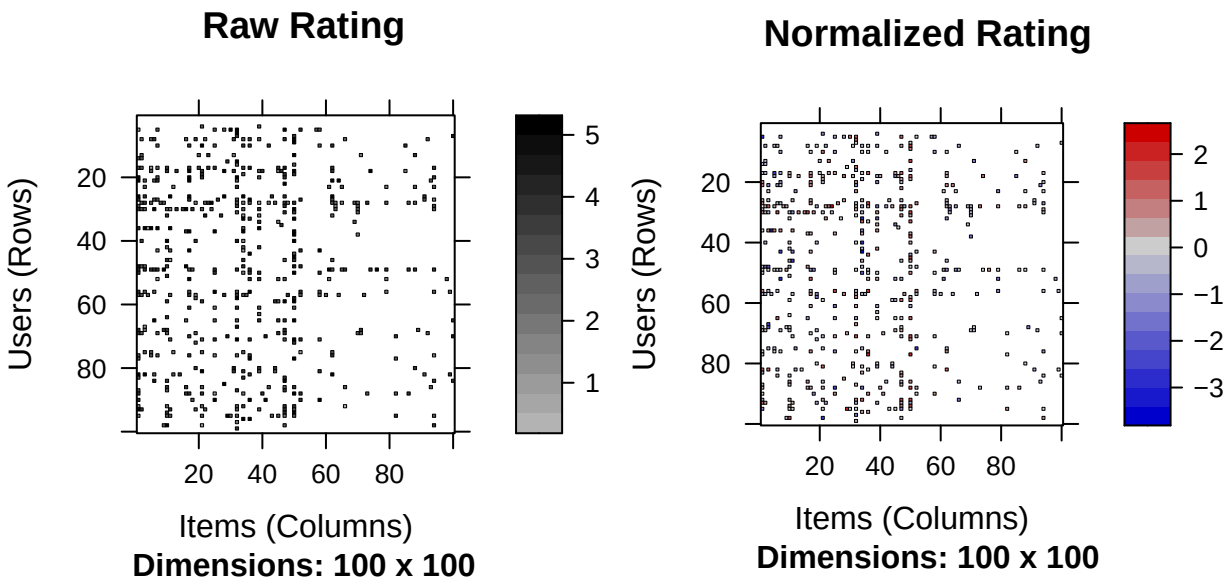
```
## [1] "realRatingMatrix"
## attr(,"package")
## [1] "recommenderlab"
```

The realRatingMatirx object can be normalized i.e. centering to remove rating bias by subtracting the row mean from all the ratings in the row.

```
#normalize the matrix
rrm_n <- normalize(rrm)
```

Image plot of the raw data and the normalised data.

```
p1 <- image(rrm[1:100,1:100], main = "Raw Rating")
p2 <- image(rrm_n[1:100,1:100], main="Normalized Rating")
gridExtra::grid.arrange(p1,p2,ncol=2)
```



For similarity calculation the Pearson Correlation is used. This is the covariance of two variables divided by their standard deviations . It is also known as the bivariate correlation. It is given by

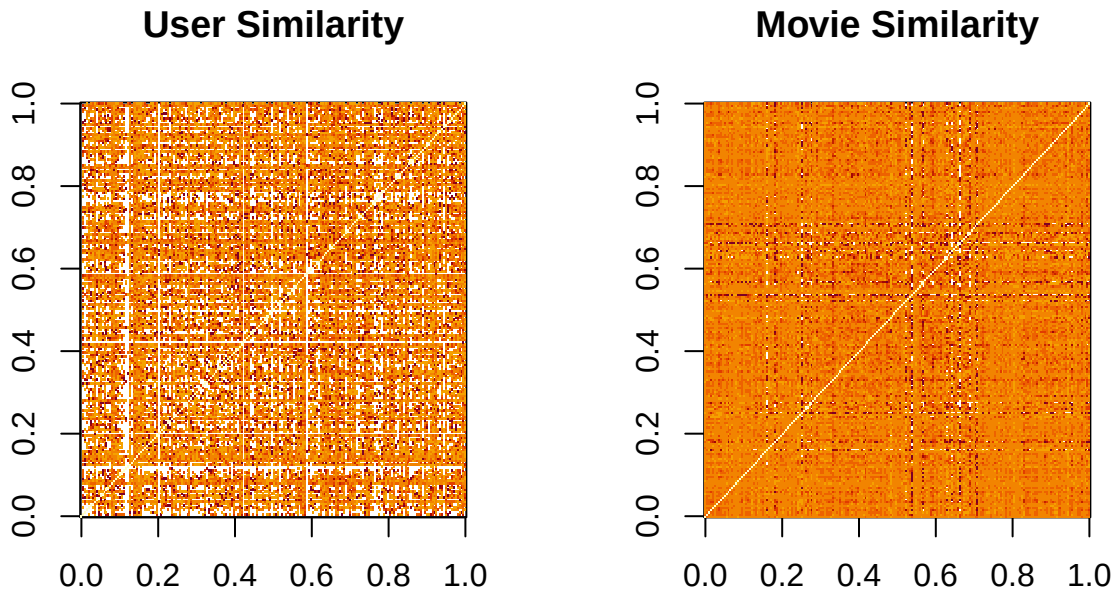
$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_x \sigma_y}$$

where

- cov is the covariance
- σ_X is the standard deviation of X
- σ_Y is the standard deviation of Y

Given the limitation of the low hardware/data ratio the similarity is calculated for 200 users and 200 movies.

```
#similarity ratings
similar_user <- similarity(rrm[1:200], method = "pearson", which = "users")
similar_movies <- similarity(rrm[,1:200], method = "pearson", which = "items")
par(mfrow=c(1,2))
image(as.matrix(similar_user), main= "User Similarity")
image(as.matrix(similar_movies), main= "Movie Similarity")
```



Observation:

- There are groups of users who give similar ratings
- There are groups of movies which get similar ratings

2.3.7 Hierarchical Clustering

To compare the interaction between users and the movie, the heatmap is generated.

Given the limitation of the low hardware/data ratio the heatmap for the top 50 movies and users with more than 25 ratings within that is used.

```
#matrix of top 50 movies and users with more than 25 ratings in that.
temp_m <- edx %>% group_by(movieId) %>% summarize(n=n(), title=first(title))%>%
  top_n(50, n) %>% pull(movieId)
x <- edx %>% filter(movieId %in% temp_m) %>% group_by(userId) %>%
  filter(n() >= 25) %>%
  ungroup() %>% select(title, userId, rating) %>%
```

```
  spread(userId,rating)
#adding the rownames and the colnames
row_names <- str_remove(x$title, ": Episode") %>% str_trunc(20)
x <- x[,-1] %>% as.matrix()
x <- sweep(x,2,colMeans(x, na.rm = TRUE))
x <- sweep(x,1,rowMeans(x, na.rm = TRUE))
rownames(x) <- row_names
```

Distance is calculated to generate the heatmap. In Hierarchical clustering it defines each observation as a separate group and then closest groups are grouped together iteratively until there is one group of all the observations.

```
#calculate the distance between each of the movies and the dendrogram
h <- dist(x) %>% hclust()
plot(h, cex=0.65, main="", xlab="")
```

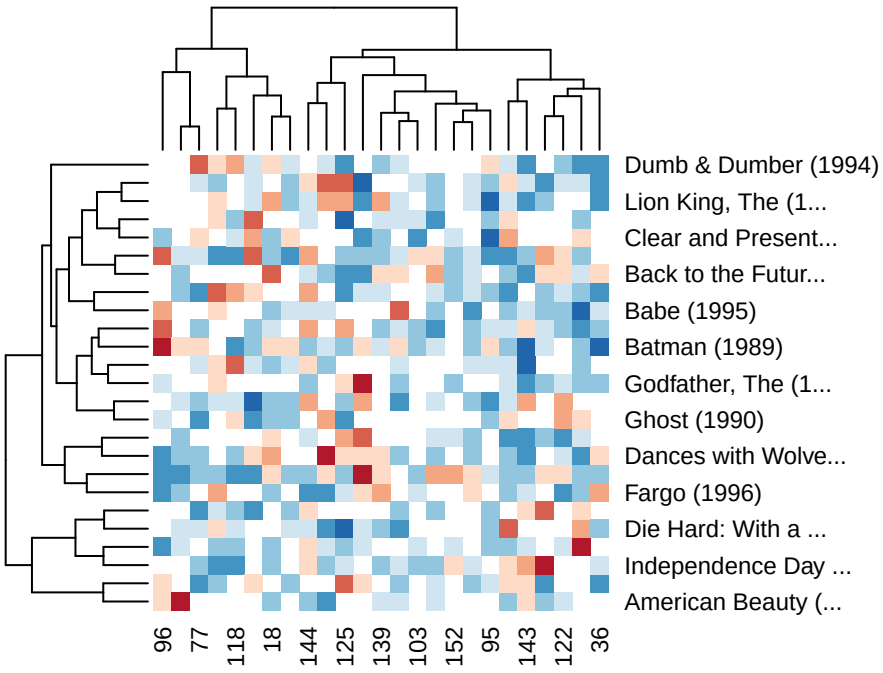
The Dendrogram shows that Star Wars movies are in one cluster and the Jim Carey movies in one cluster. This information is not there in the data set explicitly, but is intrinsically present in the rating pattern.

This is illustrated further by generating the groups and the heatmap.

```
groups <- cutree(h, k=10)
cbind(names(groups)[groups==2],names(groups)[groups==7])
```

```
##      [,1]                    [,2]
## [1,] "Ace Ventura: Pet ..." "Batman (1989)"
## [2,] "Dumb & Dumber (1994)" "Batman Forever (1..."
## [3,] "Ace Ventura: Pet ..." "Mask, The (1994)"
## [4,] "Dumb & Dumber (1994)" "Mission: Impossib..."
```

```
x_sub <- x[1:25,1:25]
heatmap(x_sub, col = RColorBrewer::brewer.pal(8,"RdBu"))
```




```
rm(edx_sub, groups, h, p1, p2, row_names, rrm, rrm_n,
   similar_movies, similar_user, temp_m, x, x_sub )
```

2.4 Prospective Model Features

Based on the exploratory analysis the edx data set is modified accordingly

- **userId** unique user identification
- **movieId** unique movie identification
- **rating** ratings given by user to the movies
- **genres** genres as classified/tagged in the data set
- **Age_of_movie** by extracting the year from the title and the timestamp this is defined as the difference between the two.

$$Age_{Movie} = Year_{rated} - Year_{make}$$

- **date_month** is the month when the movie has been rated
- **total_ratings** the count of ratings given to each movies

To fix the low number of ratings on some movies, and to avoid the ColdStart problem a filtered data set is used with movies which have had at least 25 ratings. Since no user has less than 10 ratings in the data set, there is no filter applied to the userId.

```
set.seed(1977, sample.kind = "Rounding")

#ColdStart Fix
edx_m <- edx %>% group_by(movieId) %>% filter(n() >= 25)
edx_m <- edx_m %>%
  mutate( movie_year = as.numeric(str_remove_all(str_sub(title, -6),"[^[:alnum:]]")),
          date_stamp = as_datetime(timestamp), year_t = year(date_stamp))
edx_m <- edx_m %>% mutate(age_of_movie = as.factor(year_t - movie_year))
edx_m <- edx_m %>% mutate(date_month = round_date(date_stamp, unit = "month"))
edx_m <- edx_m %>% group_by(movieId) %>%  mutate(total_ratings = n())
edx_m <- edx_m %>% select(userId, movieId, rating, genres,
                          age_of_movie, date_month, total_ratings)
edx_m <- as.data.frame(edx_m)
```

The effect of each feature can be seen by the deviations of the means of each sub group.

Using an example, the impact of the feature movies can be assessed seeing the deviations of individual means of each movie.

The overall mean of the data set edx:

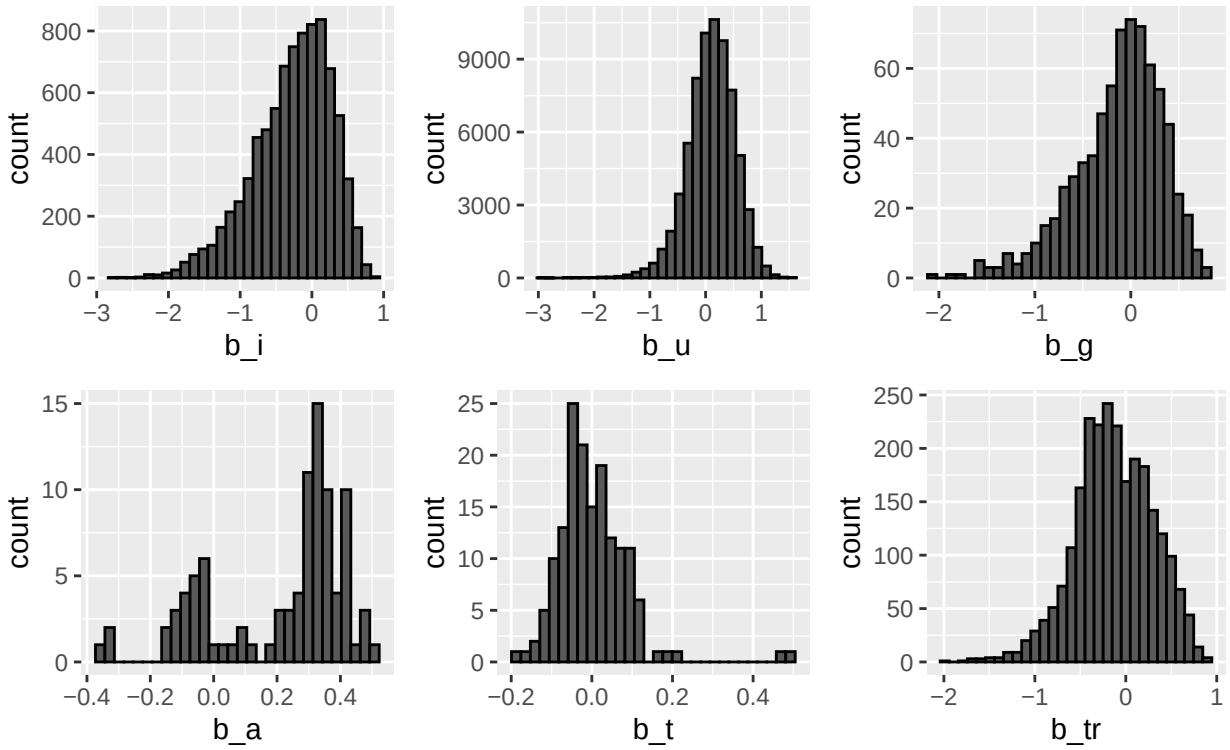
```
mu <- mean(edx_m$rating)
mu
```

```
## [1] 3.513857
```

Comparison for each feature with the overall mean.

```
#comparison of the average rating for movies
movie_avgs <- edx_m %>%
  group_by(movieId) %>%
  summarize(b_i=mean(rating-mu))
```

```
#comparison of the average rating for the user
user_avgs <- edx_m %>%
  group_by(userId) %>%
  summarize(b_u=mean(rating-mu))
#comparison of the average rating for the user
aom_avgs <- edx_m %>%
  group_by(age_of_movie) %>%
  summarize(b_a=mean(rating-mu))
#comparison of the average rating for genres
genres_avgs <- edx_m %>%
  group_by(genres) %>%
  summarize(b_g=mean(rating-mu))
#comparison of the average rating month
month_avgs <- edx_m %>%
  group_by(date_month) %>%
  summarize(b_t=mean(rating-mu))
#comparison of the average rating total movie ratings
tr_avgs <- edx_m %>%
  group_by(total_ratings) %>%
  summarize(b_tr=mean(rating-mu))

p1 <- movie_avgs %>% ggplot(aes(b_i))+ geom_histogram(bins=30, color="black")
p2 <- user_avgs %>% ggplot(aes(b_u))+ geom_histogram(bins=30, color="black")
p4 <- aom_avgs %>% ggplot(aes(b_a))+ geom_histogram(bins=30, color="black")
p3 <- genres_avgs %>% ggplot(aes(b_g))+ geom_histogram(bins=30, color="black")
p5 <- month_avgs %>% ggplot(aes(b_t))+ geom_histogram(bins=30, color="black")
p6 <- tr_avgs %>% ggplot(aes(b_tr))+ geom_histogram(bins=30, color="black")

gridExtra::grid.arrange(p1,p2,p3,p4,p5,p6, ncol=3, nrow=2)
```



Observations:

- Means of the movies and the users have a clear impact
- Means of genres also shows variations from the mean
- Means of the AgeofMovies also shows some impact, though much lesser
- Means of the month effect on the ratings shows little impact, and will not be used.
- The total ratings, though showing variation independently might have already been explained by the other features and will not be used.

3 Model Evaluation

3.1 Data Preprocessing

- Only the features that will be used, based on the exploratory analysis are being kept.
- The function data_process mutates and selects the features that will be used.

```
data_process <- function(preData){
  processData <- preData %>%
  mutate( movie_year = as.numeric(str_remove_all(str_sub(title, -6),"[^[:alnum:]]")),
          date_stamp = as_datetime(timestamp), year_t = year(date_stamp))
  processData <- processData %>% mutate(age_of_movie = as.factor(year_t - movie_year))
  processData <- processData %>% select(userId, movieId, rating, age_of_movie)
  return (as.data.frame(processData))
}
```

Splitting, using slice, the data set into training and test sets.

- The training set will be 90% of the total set with 10% kept for testing the models.
- The test set and the training set will contain the same movies and users.

```
edx_m <- data_process(edx)

test_index <- createDataPartition(y = edx_m$rating, times=1, p =0.1, list=FALSE)
train_set <- edx_m %>% slice(-test_index)
test_set <- edx_m %>% slice(test_index)

#remove users and movies not it the training set but in the test set
test_set <- test_set %>%
  semi_join(train_set, by="movieId") %>%
  semi_join(train_set, by="userId")
```

3.2 RMSE Function

RMSE is defined as

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

where $y_{u,i}$ as the actual rating for movie i by user u and $\hat{y}_{u,i}$ is the predicted rating for movie i by user u with N being the number of user/movie combinations.

One can interpret the RMSE similarly to a standard deviation. If this number is larger than 1, it means our typical error is larger than one star.

```
RMSE <- function(TrueRatings, PredictedRatings){
  sqrt(mean((TrueRatings - PredictedRatings)^2))
}
```

3.3 The Baseline Model

The Objective of any model is to make a prediction of the ratings. To measure the performance of the model it needs to perform better than the base model, which simply assumes a static rating and assumes that all the variations in the ratings are random.

Thus we take the mean ratings of all the movies in our dataset as the static rating and assume that all the users will rate the movie the same way as the mean.

The Baseline model:

The mean

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

where $\varepsilon_{i,u}$ independent errors sampled from the same distribution centered at 0. μ the "true" rating for all movies. The estimate that minimizes the RMSE is the least squares estimate of μ.

```
# Model based on the mean. the baseline
mu <- mean(train_set$rating)
naive_rmse <- RMSE(test_set$rating, mu)
rmse_results <- tibble(method="Naive_baseline", RMSE= naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive_baseline	1.059698

This is the baseline RMSE and any model of significance needs to perform better than the baseline RMSE.

3.4 Regression Model

3.4.1 Movie, User and AgeofMovie effects Model

As confirmed in the exploratory analysis the Baseline model is augmented with the terms :

- b_i for the movie effect
- b_u for the user effect
- b_a for the Age of movie effect

$$b_a = t_i - t_{u,i}$$

where t_i is the year(of item) from the title and $t_{u,i}$ the year(of user,item interaction) from the timestamp of the rating.

The Model is represented as:

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

```
#movie effects
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i=mean(rating-mu))
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effects Model",
                                     RMSE = model_1_rmse ))
#user effects
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred=mu + b_i + b_u) %>%
  pull(pred)

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
#Age of movies
age_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(age_of_movie) %>%
  summarize(b_a = mean(rating - mu - b_i - b_u))
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(age_avgs, by='age_of_movie') %>%
```

```
  mutate(pred=mu + b_i + b_u + b_a) %>%
  pull(pred)

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User + AgeofMovie Effects Model",
                                     RMSE = model_3_rmse ))
```

```
rmse_results %>% knitr::kable()
```

method	RMSE
Naive_baseline	1.0596984
Movie Effects Model	0.9432589
Movie + User Effects Model	0.8645401
Movie + User + AgeofMovie Effects Model	0.8641147

This model does show a significant improvement in the RMSE scores as compared to the baseline model. The movie and the user effects have an ~0.11 and ~0.9 decrease in the RMSE. The AgeofMovie effect reduces the RMSE slightly further.

3.4.2 Regularization

The underlying cause for regularization is due to extreme (higher or lower) means of items with very few observations in the sample which end up causing increased variance in the means of the sample and thus incorrect estimates of the deviations from the means. Thus this results in biased estimates of the sample.

For example in case of movies with very few observations but extreme (0.5 or 5) ratings

Regularization for the estimate of the movie effects.

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

The first term is just the sum of squares and the second is a penalty that gets larger when many b_i are large.

Using calculus we can actually show that 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.

- When our sample size n_i is very large, a case which will give us a stable estimate, then the penalty λ is effectively ignored since $n_i + \lambda \approx n_i$.
- When the n_i is small, then the estimate $\hat{b}_i(\lambda)$ is shrunken towards 0. The larger λ, the more we shrink.

Regularization for the estimate user effects.

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

Regularization for the estimate Age of Movie effects.

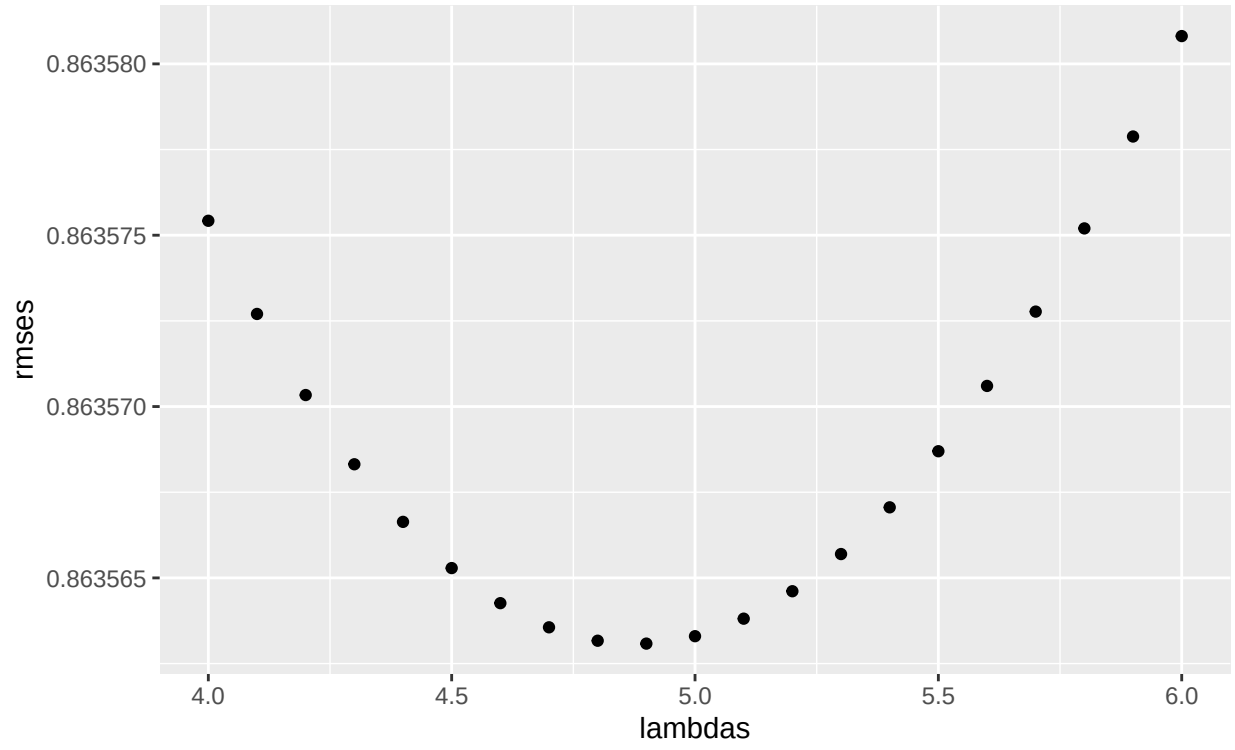
$$\sum_{u,i} (y_{u,i} - \mu - b_i - b_u - b_a)^2 + \lambda \left(\sum_i b_i^2 + \sum_u b_u^2 + \sum_u b_a^2 \right)$$

To decide on an effective or optimal λ, the tuning parameter, cross validation is used.

```
lambdas <- seq(4,6,0.1) # seq(0,10,0.5)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating-mu)/(n()+l))

  b_u <- train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i)/(n()+l))
```

```
  b_a <- train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(age_of_movie) %>%
    summarize(b_a = sum(rating - mu - b_i - b_u)/(n()+l))

  predict_ratings <- test_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_a, by="age_of_movie") %>%
    mutate(pred = mu + b_i + b_u + b_a) %>%
    pull(pred)

  return(RMSE(predict_ratings, test_set$rating))
})
```

```
lambda <- lambdas[which.min(rmses)]
qplot(lambdas, rmses)
```



The optimal λ is

```
lambda
```

```
## [1] 4.9
```

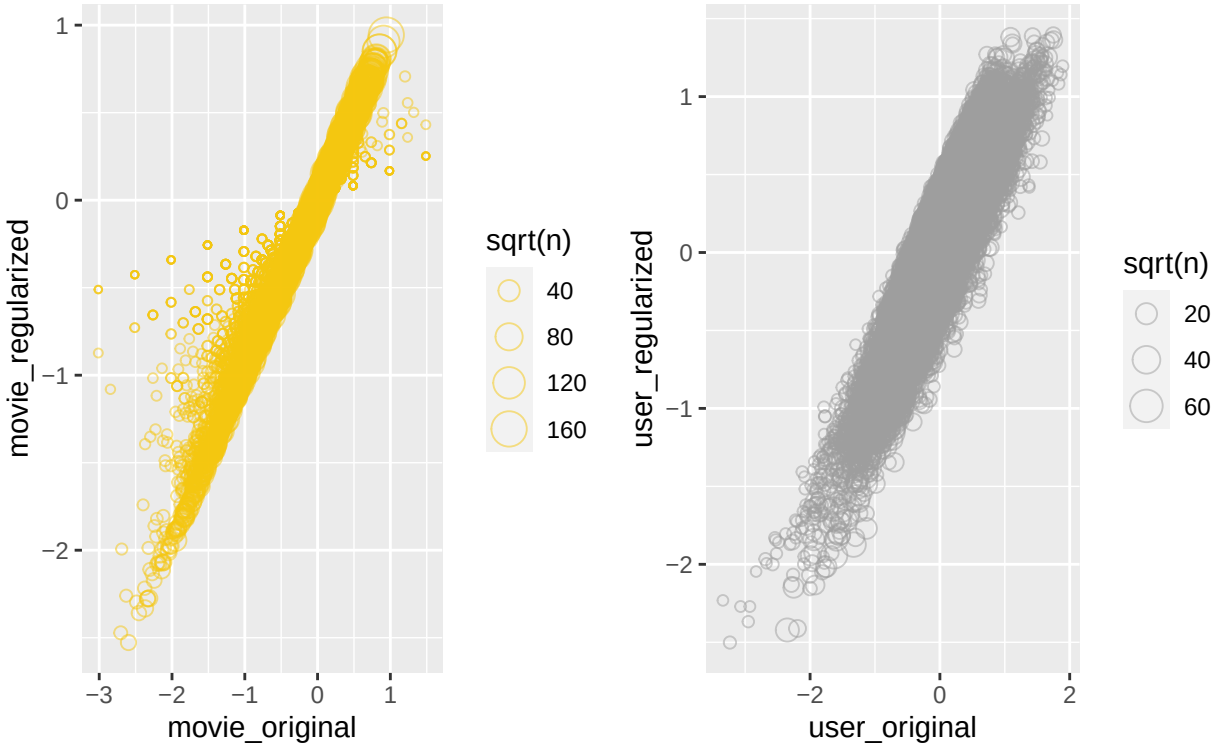
```
model_4_rmse <- rmses[lambda]
rmse_results <- bind_rows(
  rmse_results,data_frame(method="Movie + User + AgeofMovie Regularized Model",
                          RMSE = model_4_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive_baseline	1.0596984
Movie Effects Model	0.9432589
Movie + User Effects Model	0.8645401
Movie + User + AgeofMovie Effects Model	0.8641147
Movie + User + AgeofMovie Regularized Model	0.8635683

Regularization reduces the RMSE of the combined effects model by ~0.001

Visualizing the estimate shrinkage using regularization

```
#impact of regularization
movie_reg_m <- train_set %>%
  group_by(movieId) %>% summarize(b_i=sum(rating-mu)/(n()+lambda), n_i =n() )
movie_reg_u <- train_set %>%
  group_by(userId) %>% summarize(b_u=sum(rating-mu)/(n()+lambda), n_i =n() )

p1 <- tibble(movie_original = movie_avgs$b_i, movie_regularized = movie_reg_m$b_i,
             n = movie_reg_m$n_i) %>%
  ggplot(aes(movie_original, movie_regularized, size=sqrt(n))) +
  geom_point(shape=1, alpha = 0.5, color=7)

p2 <- tibble(user_original = user_avgs$b_u, user_regularized = movie_reg_u$b_u,
             n = movie_reg_u$n_i) %>%
  ggplot(aes(user_original, user_regularized, size=sqrt(n))) +
  geom_point(shape=1, alpha = 0.5, color=8)
gridExtra::grid.arrange(p1,p2, ncol=2)
```


movie_regularized
movie_original
sqrt(n)
40
80
120
160
user_regularized
user_original
sqrt(n)
20
40
60

3.5 Matrix Factorization

Matrix factorization technique approximates the rating matrix $R_{m\times n}$ by the product of two lower dimension matrices $P_{k\times m}$ and $Q_{K\times n}$ such that $R \approx \tilde{P}Q$

If p_u is the u-th column of P and q_v is the v-th column of Q, then the rating given by the user u on item v is predicted as $\tilde{p}_u q_v$

3.5.1 Recommenderlab

The recommenderlab provides an infrastructure to test and develop recommender algorithms.

The recommender is created using the Recommender() function. The available recommendation methods are stored in the registry which can be queried.

A list of methods for the realRating data can be queried with the command:

recommenderRegistry$get_entries(dataType="realRatingMatrix")

In this project the following algorithms are used:

- UBCF which is the User-based Collaborative Filtering.
- POPULAR which is based on Item (in this case movies) popularity.
- SVD which is based on SVD approximation with column-mean imputation.

The Rating matrix can be created as below:

```
##recommenderlab
temp <- edx %>% select(userId, movieId, rating)
temp <- as.data.frame(temp)
rrm <- as(temp, "realRatingMatrix")
class(rrm)
```

```
## [1] "realRatingMatrix"
## attr(,"package")
## [1] "recommenderlab"
```

Function to get the title of the movies from a list of movieId, which is the output of a recomender function.

```
movie_titles <- edx %>% distinct(movieId, title)
get_movie <- function(i){
  movie_titles %>% filter(movieId %in% i) %>% select(title)
}
```

The Recommender is run on a subset of the first 2000 observations, and makes a prediction for a user.

```
#recommender function
recom <- Recommender(rrm[1:2001], method="UBCF")
#predict
recom_p <- predict(recom, rrm[2020], n=5)
#results
print("UBCF Results")
```

```
## [1] "UBCF Results"
```

```
sapply(as(recom_p,"list") , function(i){
  movie_titles %>% filter(movieId %in% i) %>% select(title)
})
```

```
## $`2115.title`
## [1] "Saving Private Ryan (1998)" "Ronin (1998)"
## [3] "Breakdown (1997)"           "Out of Sight (1998)"
## [5] "Lady and the Tramp (1955)"
```

```
#recommender function
recom <- Recommender(rrm[1:2001], method="POPULAR")
#predict
recom_p <- predict(recom, rrm[2020], n=5)
#results
print("POPULAR Results")
```

```
## [1] "POPULAR Results"
```

```
sapply(as(recom_p,"list") , function(i){
  movie_titles %>% filter(movieId %in% i) %>% select(title)
})
```

```
## $`2115.title`
## [1] "Braveheart (1995)"
## [2] "Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)"
## [3] "Schindler's List (1993)"
## [4] "Matrix, The (1999)"
## [5] "Shawshank Redemption, The (1994)"
```

```
#recommender function
recom <- Recommender(rrm[1:2001], method="SVD")
#predict
recom_p <- predict(recom, rrm[2020], n=5)
#results
print("SVD Results")
```

```
## [1] "SVD Results"
```

```
sapply(as(recom_p,"list") , function(i){
  movie_titles %>% filter(movieId %in% i) %>% select(title)
})
```

```
## $`2115.title`
## [1] "Dumb & Dumber (1994)"              "Ace Ventura: Pet Detective (1994)"
## [3] "Fargo (1996)"                      "Jerry Maguire (1996)"
## [5] "American Beauty (1999)"
```

The different movies predicted for the same user can be attributed to the limited amount of data that we are providing for learning and the algorithm used for learning.

Given the limitation of the low hardware/data ratio running of the recommender using the entire data set is not possible.

Thus, for the purpose of illustration, the upper and lower 90% quantiles of the data are being ignored. This will result in most of the information being there in the matrix that is used for learning and testing.

```
rrm_m <- rrm[rowCounts(rrm)>quantile(rowCounts(rrm),0.9),
             colCounts(rrm)>quantile(colCounts(rrm),0.9)]
```

The usage of the recommender consists of the following steps:

$$evaluationScheme() \rightarrow Recommender() \rightarrow predict() \rightarrow calcPredictionAccuracy()$$

Recommenderlab evaluation scheme starts with determining the split for the training and the test set. In this case it is 0.8. The given=-5 implies that 5 ratings for the 20% users are excluded for testing. The goodRating = 4 implies that ratings above 4 are considered as a preferable rating for the user. Method can be split, bootstrap, k-fold cross validation.

```
#evaluation
e <- evaluationScheme(rrm_m, method="split", train=0.8, given=-5, goodRating=4)

r1 <- Recommender(getData(e,"train"),"UBCF",
                  param=list(normalize="center", method="pearson"))
p1 <- predict(r1, getData(e,"known"), type="ratings")

r2 <- Recommender(getData(e,"train"),"POPULAR")
p2 <- predict(r2, getData(e,"known"), type="ratings")

r3 <- Recommender(getData(e,"train"),"SVD")
p3 <- predict(r3, getData(e,"known"), type="ratings")
```

The RMSE for the tests are collated below:

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="RecommenderLab UBCF",
                          RMSE = calcPredictionAccuracy(
                            p1, getData(e,"unknown"))['RMSE']))
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="RecommenderLab POPULAR",
                          RMSE = calcPredictionAccuracy(
                            p2, getData(e,"unknown"))['RMSE']))
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="RecommenderLab SVD",
                          RMSE = calcPredictionAccuracy(
                            p3, getData(e,"unknown"))['RMSE']))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive_baseline	1.0596984
Movie Effects Model	0.9432589
Movie + User Effects Model	0.8645401
Movie + User + AgeofMovie Effects Model	0.8641147

method	RMSE
Movie + User + AgeofMovie Regularized Model	0.8635683
RecommenderLab UBCF	0.8714847
RecommenderLab POPULAR	0.8486814
RecommenderLab SVD	0.8463411

3.5.2 Recosystem

Recosystem package, used due to the limitation of the low hardware/data ratio, provides a recommendation system using matrix factorization. The Recosystem is a wrapper of the LIMBF,a C++ library, which is used for large scale matrix factorization.

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

The usage of the Recosystem consists of the following steps:

$$Reco() \rightarrow tune() \rightarrow train() \rightarrow output() \rightarrow predict()$$

Reco() is to call the model, tune() to tune the parameters, train() to train the model, output() to export the model and predict() to compute the predicted values.

```
require(recosystem)
set.seed(1977, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times=1, p =0.2, list=FALSE)
train_set <- edx %>% slice(-test_index)
test_set <- edx %>% slice(test_index)

#remove users and movies not it the training set but in the test set
test_set <- test_set %>%
  semi_join(train_set, by="movieId") %>%
  semi_join(train_set, by="userId")

#specify the train and the test data
train_data <- with(train_set, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating = rating))
test_data <- with(test_set, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating = rating))
#create the Model object
r <- recosystem::Reco()
#tuning the parameters; nthread is the number of threads to use
#and niter is the number of iterations.

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= 4, niter=10, verbose=FALSE))
```

```
#Train the model using the optimized parameters
r$train(train_data, opts = c(opts$min, nthread=4, niter=42, verbose=FALSE))
#predicts the results
y_hat <- r$predict(test_data, out_memory())
```

```
rmse_results <- bind_rows(
  rmse_results,data_frame(method="RecoSystem",
                          RMSE = RMSE(test_set$rating, y_hat)))
```

3.6 Collated RMSE results

```
rmse_results %>% knitr::kable()
```

method	RMSE
Naive_baseline	1.0596984
Movie Effects Model	0.9432589
Movie + User Effects Model	0.8645401
Movie + User + AgeofMovie Effects Model	0.8641147
Movie + User + AgeofMovie Regularized Model	0.8635683
RecommenderLab UBCF	0.8714847
RecommenderLab POPULAR	0.8486814
RecommenderLab SVD	0.8463411
RecoSystem	0.7904280

The results show that the Recosystem method using matrix factorization provides the least RMSE, followed by the recommenderlab POPULAR method. The Regularized linear model also performs well on our training set.

4 Final Validation

Based on the Model evaluation the three features that had an impact were the userId, movieId and the age_of_movie.

- **userId** unique user identification
- **movieId** unique movie identification
- **rating** ratings given by user to the movies
- **Age_of_movie** by extracting the year from the title and the timestamp this is defined as the difference between the two.

The optimal λ based on regularization of the training set will be used

```
edx_f <- data_process(edx)
validation_f <- data_process(validation)
```

Thus the validation_f and the edx_f are the data sets for the final validation of the model.

4.1 Regularized Linear Model

In the testing process, the linear model with regularization on the users, the movies effects and the AgeofMovies achieved the required project RMSE objective. The same model is now run on the complete edx set with the validation set being used for predict.

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

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

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

b_a <- edx_f %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  group_by(age_of_movie) %>%
  summarize(b_a = sum(rating - mu - b_i - b_u)/(n()+lambda))

predict_ratings <- validation_f %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_a, by="age_of_movie") %>%
  mutate(pred = mu + b_i + b_u + b_a) %>%
  pull(pred)
Final_rmse <- RMSE(predict_ratings, validation_f$rating)
```

The RMSE using Regularized linear model with the movie, user and AgeofMovie effects is

```
Final_rmse
```

```
## [1] 0.864347
```

```
rmse_results <- bind_rows(rmse_results,data_frame(
  method="Final: Movie + User + AgeofMovie Regularized Model (validation)",
                           RMSE = Final_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive_baseline	1.0596984
Movie Effects Model	0.9432589
Movie + User Effects Model	0.8645401
Movie + User + AgeofMovie Effects Model	0.8641147
Movie + User + AgeofMovie Regularized Model	0.8635683
RecommenderLab UBCF	0.8714847
RecommenderLab POPULAR	0.8486814
RecommenderLab SVD	0.8463411
RecoSystem	0.7904280
Final: Movie + User + AgeofMovie Regularized Model (validation)	0.8643470

The top 5 best movies

```
validation %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  arrange(-pred) %>% group_by(title) %>% select(title) %>%
  head(5)
```

```
## # A tibble: 5 x 1
## # Groups:   title [3]
##   title
##   <chr>
## 1 Usual Suspects, The (1995)
## 2 Shawshank Redemption, The (1994)
## 3 Shawshank Redemption, The (1994)
## 4 Shawshank Redemption, The (1994)
## 5 Eternal Sunshine of the Spotless Mind (2004)
```

The top 5 worst movies

```
validation %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  arrange(pred) %>% group_by(title) %>% select(title) %>%
  head(5)
```

```
## # A tibble: 5 x 1
## # Groups:   title [5]
##   title
##   <chr>
## 1 Battlefield Earth (2000)
## 2 Police Academy 4: Citizens on Patrol (1987)
## 3 Karate Kid Part III, The (1989)
## 4 PokÃ©mon Heroes (2003)
## 5 Turbo: A Power Rangers Movie (1997)
```

4.2 Matrix Factorization

Matrix factorization is implemented with the Recosystem package.

```
require(recosystem)
set.seed(1977, sample.kind = "Rounding")

#specify the train and the test data
train_data <- with(edx_f, data_memory(user_index = userId,
                                      item_index = movieId,
                                      rating = rating))
test_data <- with(validation_f, data_memory(user_index = userId,
                                     item_index = movieId,
                                     rating = rating))
#create the Model object
r <- recosystem::Reco()
#tuning the parameters; nthread is the number of threads to use
#and niter is the number of iterations.

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= 4, niter=10, verbose=FALSE))
```

```
#train the model
r$train(train_data, opts = c(opts$min, nthread=4, niter=42, verbose = FALSE))
#predicts the results
y_hat <- r$predict(test_data, out_memory())
```

The RMSE for Matrix Factorization using recosystem library is:

```
RMSE(validation_f$rating, y_hat)
```

```
## [1] 0.7805731
```

```
rmse_results <- bind_rows(
  rmse_results,data_frame(method="RecoSystem (validation)",
                          RMSE = RMSE(validation_f$rating, y_hat)))
```

```
rmse_results %>% knitr::kable()
```

method	RMSE
Naive_baseline	1.0596984
Movie Effects Model	0.9432589
Movie + User Effects Model	0.8645401
Movie + User + AgeofMovie Effects Model	0.8641147
Movie + User + AgeofMovie Regularized Model	0.8635683
RecommenderLab UBCF	0.8714847
RecommenderLab POPULAR	0.8486814
RecommenderLab SVD	0.8463411
RecoSystem	0.7904280
Final: Movie + User + AgeofMovie Regularized Model (validation)	0.8643470
RecoSystem (validation)	0.7805731

Using the recommenderlab library for the movielens data set (most relevant subset), as downloaded and preprocessed prior to the edx and validation set creation. (As an illustration)

```
temp <- movielens %>% select(userId, movieId, rating)
temp <- as.data.table(temp)
rrm <- as(temp, "realRatingMatrix")
#Using only most relevant users and movies due to low data/hardware ratio
rrm_m <- rrm[rowCounts(rrm)>quantile(rowCounts(rrm),0.9),
             colCounts(rrm)>quantile(colCounts(rrm),0.9)]
#evaluation
e <- evaluationScheme(rrm_m, method="split", train= 0.9, given=-5, goodRating=4)

r1 <- Recommender(getData(e,"train"),"POPULAR")
p1 <- predict(r1, getData(e,"known"), type="ratings")

r2 <- Recommender(getData(e,"train"),"UBCF",
                  param=list(normalize="center", method="pearson"))
p2 <- predict(r2, getData(e,"known"), type="ratings")

r3 <- Recommender(getData(e,"train"),"SVD")
p3 <- predict(r3, getData(e,"known"), type="ratings")
```

```
rmse_results <- bind_rows(
  rmse_results,data_frame(method="Movielens RecommenderLab POPULAR (movielens)",
                          RMSE = calcPredictionAccuracy(
                            p1, getData(e,"unknown"))['RMSE']))
rmse_results <- bind_rows(
  rmse_results,data_frame(method="Movielens RecommenderLab UBCF (movielens)",
                          RMSE = calcPredictionAccuracy(
                            p2, getData(e,"unknown"))['RMSE']))
rmse_results <- bind_rows(
  rmse_results,data_frame(method="Movielens RecommenderLab SVD (movielens)",
                          RMSE = calcPredictionAccuracy(
                            p3, getData(e,"unknown"))['RMSE']))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive_baseline	1.0596984
Movie Effects Model	0.9432589
Movie + User Effects Model	0.8645401
Movie + User + AgeofMovie Effects Model	0.8641147
Movie + User + AgeofMovie Regularized Model	0.8635683
RecommenderLab UBCF	0.8714847
RecommenderLab POPULAR	0.8486814
RecommenderLab SVD	0.8463411
RecoSystem	0.7904280
Final: Movie + User + AgeofMovie Regularized Model (validation)	0.8643470
RecoSystem (validation)	0.7805731
Movielens RecommenderLab POPULAR (movielens)	0.8338237
Movielens RecommenderLab UBCF (movielens)	0.8482094
Movielens RecommenderLab SVD (movielens)	0.8304354

The recommenderlab SVD performs the best, followed by the POPULAR and the UBCF. This result however is on a subset, which though explains most of the variance in the ratings, will have limitations when implementing.

5 Conclusion

The project went through the exploratory stage, where the data and all the features and some combination were assessed using various techniques.

In the model evaluation, linear models using regularization and matrix factorization methods were explored.

The Final validation was done with a Regularized linear model. Matrix factorization was implemented using the recosystem library.

As an illustration, the recommenderlab library on the movielens data set was also implemented.

5.1 Results Summary

Summarizing the results of this project, on building a recommendation system for the movielens data set (with the 10M version), the objective of **RMSE less than 0.8649** was achieved.

- Regularized linear model with the movie, user and AgeofMovie effects achieved the **RMSE of 0.86435**
- Matrix factorization using the recosystem library achieved the **RMSE of 0.78**

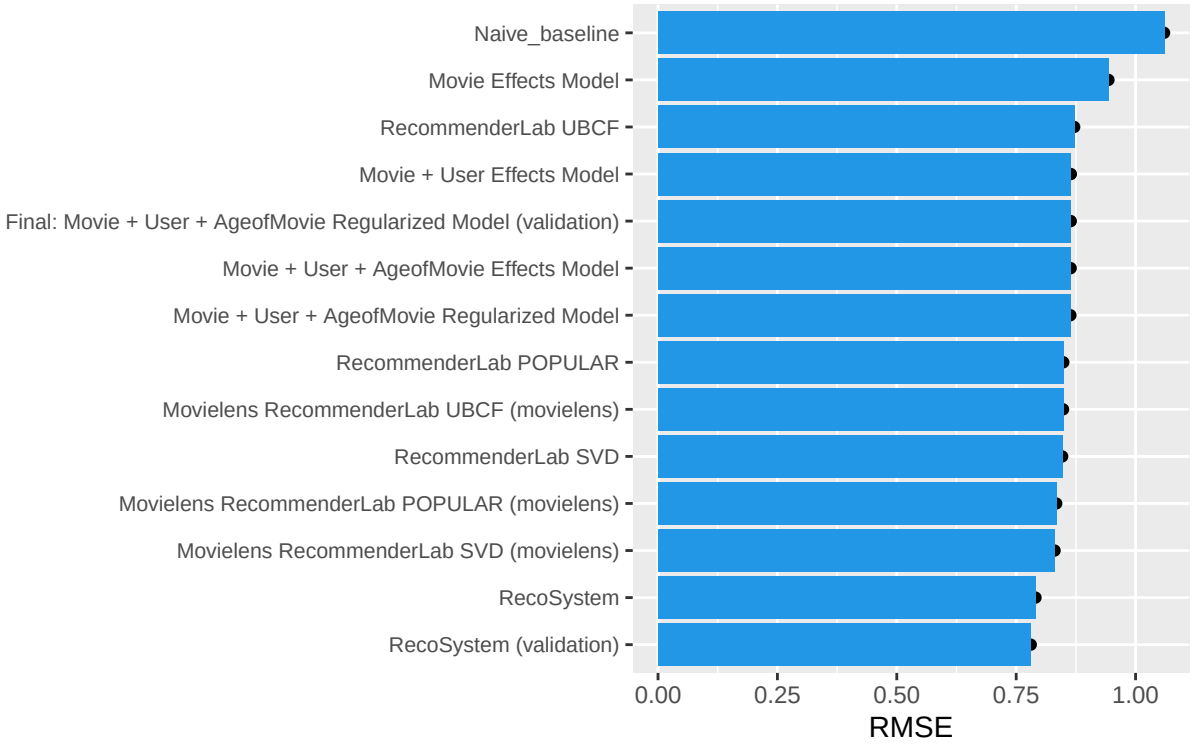
A table of all the models that were run in this project.

```
rmse_results %>% knitr::kable()
```

method	RMSE
Naive_baseline	1.0596984
Movie Effects Model	0.9432589
Movie + User Effects Model	0.8645401
Movie + User + AgeofMovie Effects Model	0.8641147
Movie + User + AgeofMovie Regularized Model	0.8635683
RecommenderLab UBCF	0.8714847
RecommenderLab POPULAR	0.8486814
RecommenderLab SVD	0.8463411
RecoSystem	0.7904280
Final: Movie + User + AgeofMovie Regularized Model (validation)	0.8643470
RecoSystem (validation)	0.7805731
Movielens RecommenderLab POPULAR (movielens)	0.8338237
Movielens RecommenderLab UBCF (movielens)	0.8482094
Movielens RecommenderLab SVD (movielens)	0.8304354

Visualizing the Results

```
rmse_results %>%
  ggplot(aes(reorder(method,RMSE),RMSE)) + geom_point() +
  geom_bar(stat="identity", fill= 4) +
  coord_flip() +
  theme(axis.text.y = element_text(size=8)) +
  ylab("RMSE") + xlab('')
```



All the models performed better than the baseline model of the mean of the ratings.

The Recosystem model, which implements matrix factorization achieved the best results.

Though the recommender models did perform better than the Regression models using regularization, the dataset used was a subset of the original. This subset was used to overcome the errors caused due to the memory restrictions. These have thus been used for illustration only.

All of these models can be implemented for existing users and movies and will need to be retrained regularly to incorporate changes and additions in the ratings of the movies.

Users and movies with no ratings available do not have any initial recommendation. This limitation can be overcome using the genres, and other user specific information. The same has not been implemented here.

5.2 References

- **Rafael A. Irizarry Introduction to Data Science**: https://rafalab.github.io/dsbook/
- Netflix prize : http://www.netflixprize.com/
- GroupLens : http://grouplens.org/datasets/movielens/
- Recosystem : https://cran.r-project.org/web/packages/recosystem/vignettes/
- RecommenderLab : https://cran.r-project.org/web/packages/recommenderlab/index.html

veneet.bhardwaj@gmail.com