Movielens Dataset

Julieta Peisino

1- Introduction

A Recommender System refers to a system that is capable of predicting the future preference of a set of items for a user, and recommend the top items. This is beneficial for **users** because they have a better experience with the transaction, finding what they want, saving time and being able to compare different products that are similar before choosing.And for **providers** because by making the experience smoother for the user, they increase loyalty, have data about preferences of users that helps them to customize products/services to reduce costs and increase revenue.

Some examples of recommendations systems are movie recommendation in Netflix, Amazon product recommendation, Spotify music recommendation.

One example of how all this information can be used is the case of "House of Cards" in Netflix *"When the program, a remake of a BBC miniseries, was up for purchase in 2011 with David Fincher and Kevin Spacey attached, the folks at Netflix simply looked at their massive stash of data. Subscribers who watched the original series, they found, were also likely to watch movies directed by David Fincher and enjoy ones that starred Kevin Spacey. Considering the material and the players involved, the company was sure that an audience was out there."*

For this project, I am going to create a movie recommendation system using MovieLens dataset trying to predict ratings that user will give to movies based on historical information. I will analyze the impact of different variables to get better estimations and reduce RMSE (Root Mean Square Error) and select the model that estimates better the ratings. The final objetive is to get a **RMSE < 0.86490**

1.1-Data Preparation

1.1.1-Install all packages necessary for this project

First we need to get all packages nedded to run this code

```
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(lubridate)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(recosystem))
  install.packages("recosystem", repos = "http://cran.us.r-project.org")
library(recosystem)
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
library(ggplot2)
library(knitr)
library(gridExtra)
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip
```

1.2-Download database and split in edx(training set) validation (test set)

From https://grouplens.org/datasets/movielens/10m/ we will get dataset and divide it in training (edx) and test set (validation)

```
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")

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

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

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

2- Methods

2.1 - Data Exploration and cleaning

2.1.1 - Exploring databases

Training Set (edx)

```
##Analyze structure of edx
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 83898488
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Actio
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
##Analyze the range of ratings
paste0("Movie rating goes from ",round(min(edx$rating),2)," to ",round(max(edx$rating),2))
```

```
## [1] "Movie rating goes from 0.5 to 5"
```

```
##Analyze if there are NA
sum(is.na(edx$userId),is.na(edx$movieId),is.na(edx$genres),
    is.na(edx$timestamp),is.na(edx$rating),is.na(edx$title))
```

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

```
##Number of users
paste0("there are ", n_distinct(edx$userId)," diferent users")
```

```
## [1] "there are 69878 diferent users"
```

```
##Number of movies
paste0("there are ",n_distinct(edx$movieId)," diferent movies")
```

```
## [1] "there are 10677 diferent movies"
```

```
##Number of titles
paste0("there are ",n_distinct(edx$title)," diferent titless")
```

```
## [1] "there are 10676 diferent titless"
```

The **edx** training set is a data frame with 9000055 obs and 6 variables.

Each **movie** is defined by a unique number and there are 10677 different movies

Each **user** is defined by a unique number and there are 69878 different users

ratings are in a range between 0.5 (bad movie) and 5(excellent movie) there are no NA values in database. And ratings are full star or half star

timestamp is the date when the movie was rated (in next steps we will change format for better understanding)

on **title** we can see that movie title is followed by the year that the movie was released (in next steps this year is going to be saved in a new column to analyze aging effect)

There are 69878 different users and 10677 different movies (in case of War of Worlds (2005) we will assume that there are 2 movies with the same title that is the reason we have 10677 movieID and 10676 titles)

Test Set (validation)

```
##Structure of validation set
str(validation)
```

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...

##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 113357l
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (19
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Ro
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
##Range of ratings
paste0("Movie rating goes from ",round(min(validation$rating),2),
       " to ",round(max(validation$rating),2))
```

```
## [1] "Movie rating goes from 0.5 to 5"
```

```
#Check if there are NA
sum(is.na(validation$userId),is.na(validation$movieId),
    is.na(validation$genres),is.na(validation$timestamp),
    is.na(validation$rating),is.na(validation$title))
```

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

```
#number of users
paste0("there are ", n_distinct(validation$userId)," diferent users")
```

```
## [1] "there are 68534 diferent users"
```

```
#number of movies
paste0("there are ",n_distinct(validation$movieId),
       " diferent movies")
```

```
## [1] "there are 9809 diferent movies"
```

```
paste0("there are ",n_distinct(validation$title)," diferent titles")
```

```
## [1] "there are 9808 diferent titles"
```

Test set has 999999 observations, there are less movies and users than in training set (but all users and movies that are in edx are in validation)

2.1.2 - Cleaning databases

to save all cleaning we will create new data frames

```
edx_y<-edx
```

Creating new columns

year= Year movie was released

rate_date= date when user rated the movie

year_rate= year the user rated the movie

```
##generating new column for year and extracting it from title
year<-str_extract(edx$title,"\\((\\d{4})\\)")
year<-str_replace(str_replace(year,"[(]",""),"[)]", "")
year<-as.numeric(year)
##generating extra columns for year of rating, and date of rating
edx_y<-edx%>%mutate(year=year,rate_date=as_datetime(timestamp),
                    yr_rate=year(rate_date))
```

```r
##generating new column for year and extracting it from title
yearv<-str_extract(validation$title,"\\((\\d{4})\\)")
yearv<-str_replace(str_replace(yearv,"[(]",""),"[)]", "")
yearv<-as.numeric(yearv)
##generating extra columns for year of rating, and date of rating
validation<-validation%>%mutate(year=yearv,rate_date=as_datetime(timestamp),
                                yr_rate=year(rate_date))
###checking ranges
range(edx_y$year)
```

```
## [1] 1915 2008
```

```r
range(edx_y$yr_rate)
```

```
## [1] 1995 2009
```

```r
range(validation$year)
```

```
## [1] 1915 2008
```

```r
range(validation$yr_rate)
```

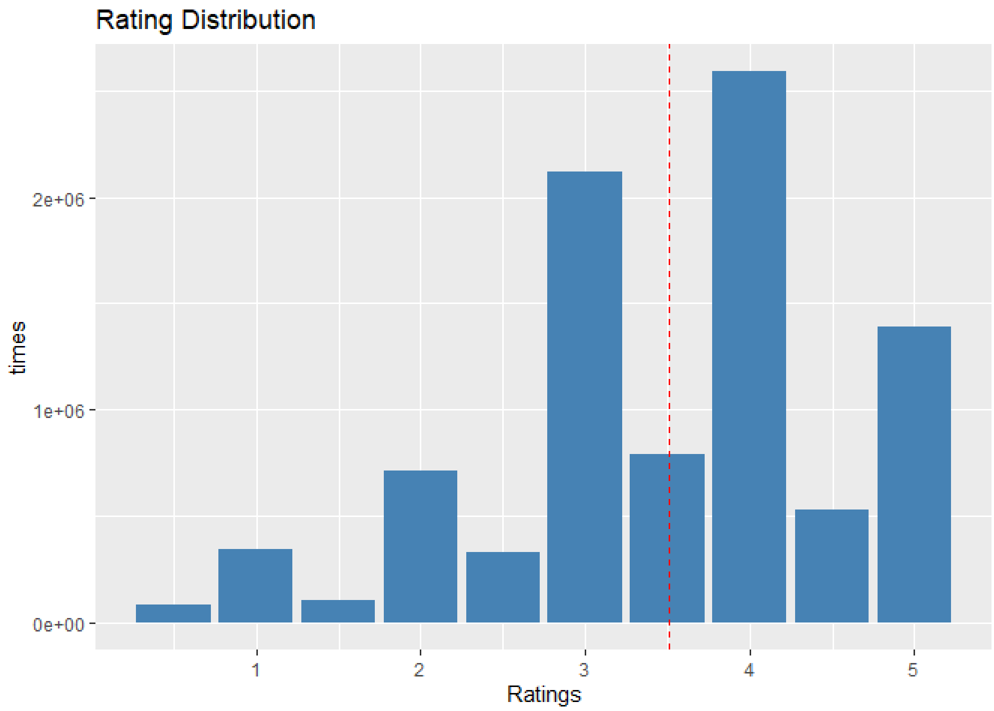
```
## [1] 1995 2009
```

Analyzing the ranges of this new values we can see that there are no strange values

2.1.3 Analyzing impact of different variables in the ratings

2.1.3.1 Rating

We can see that Average rating is 3.51 and users tend to give full star ratings instead of half star ratings. Rating distribution is skewed to the right.




```r
##Mean rating
mean_rating<-mean(edx_y$rating)
paste0("Average rating is ",round(mean(edx_y$rating),2))
```

```
## [1] "Average rating is 3.51"
```

2.1.3.2 Rating vs MovieID

```
r_mov<-edx_y%>%group_by(title)%>%mutate(n=n())
#Movie with more ratings
paste0("The movie with more ratings is: ", r_mov$title[which.max(r_mov$n)])
```

```
## [1] "The movie with more ratings is: Pulp Fiction (1994)"
```

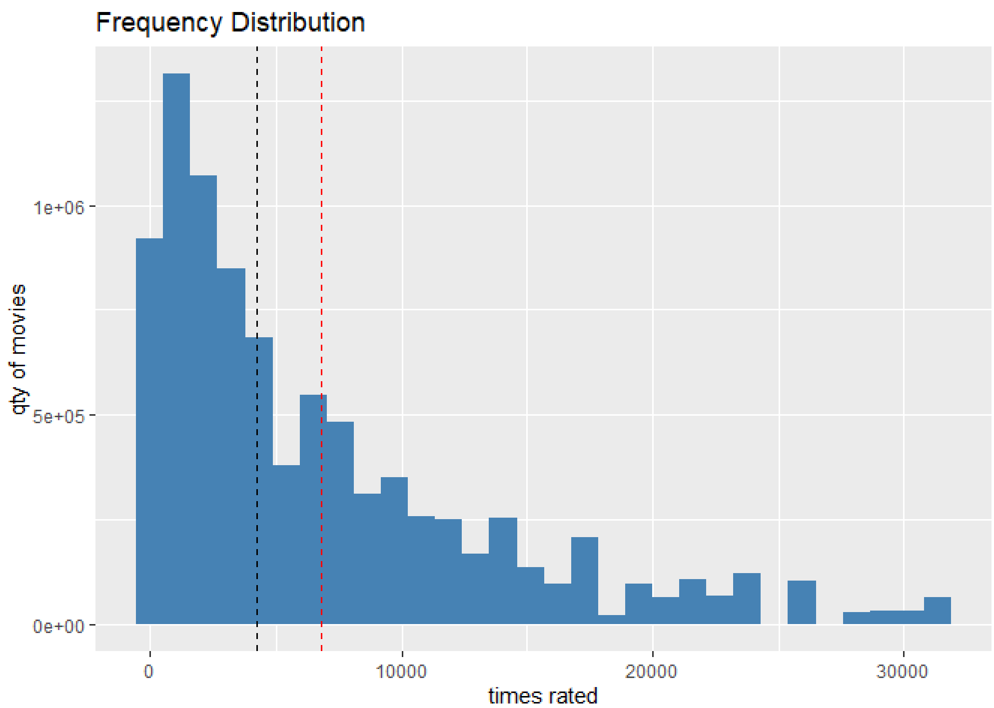
```
#Average number of ratings per movie
mean_r_movie<-mean(r_mov$n)
paste0("Average quantity of ratings per movie is: ",round(mean_r_movie,0))
```

```
## [1] "Average quantity of ratings per movie is: 6787"
```

```
#Median number of ratings per movie
paste0("Median quantity of ratings per movie is: ",round(median(r_mov$n),0))
```

```
## [1] "Median quantity of ratings per movie is: 4223"
```

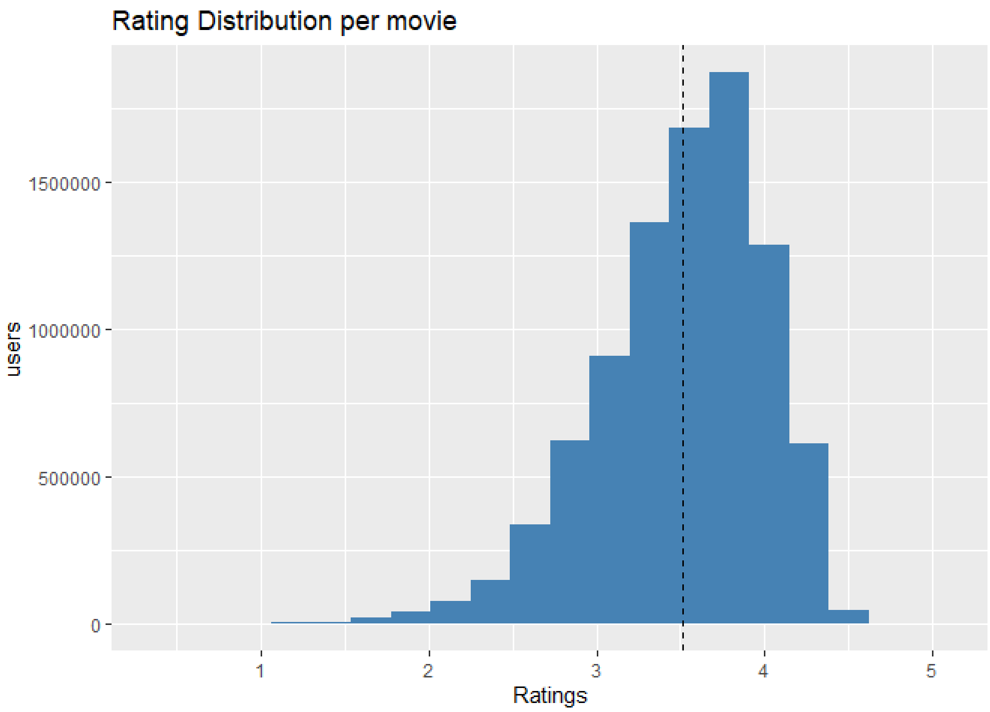
```
##Frequency of rating distribution
edx_y%>%group_by(movieId)%>%mutate(n=n())%>%ggplot(aes(n))+
  geom_histogram(bins = 30,fill="steelblue")+
  geom_vline(xintercept = mean_r_movie, lty = 2, color= "Red")+
  geom_vline(xintercept = median(r_mov$n), lty = 2, color= "Black")+
  labs(title = "Frequency Distribution",x = "times rated", y = "qty of movies")
```



We can observe that the mean quantity of rating per movie is 6787 and the median is 4223, this difference is because there are movies that were rated more than usual like Pulp fiction that was rated 31362 times.

Most of the movies were rated between 3 and 4. But is illogical to think that all the movies will receive same rating. There is a Movie effect giving some dispersion of the ratings around the mean.

```
edx_y%>%group_by(movieId)%>%mutate(avg_rat=mean(rating))%>%
  ggplot(aes(avg_rat))+
  geom_histogram(bins = 20,fill="steelblue")+
  labs(title = "Rating Distribution per movie",
       x = "Ratings", y = "users")+
  geom_vline(xintercept = mean_rating, lty = 2, color= "Black")
```



2.1.3.3 Rating vs UserID

```
ratings_per_user<-edx_y%>%group_by(userId)%>%mutate(n_rating=n())
#range of number of rating per user
paste0("max and min qty of ratings are: ",range(ratings_per_user$n_rating))
```

```
## [1] "max and min qty of ratings are: 10"
## [2] "max and min qty of ratings are: 6616"
```

```
##max number of rating per user
paste0("the user that rated more movies was: ",
       ratings_per_user$userId[which.max(ratings_per_user$n_rating)])
```

```
## [1] "the user that rated more movies was: 59269"
```

```
##Mean number ofrating per user
mean_rating_per_user<-mean(ratings_per_user$n_rating)
paste0("Average qty of ratings per user is: ",mean_rating_per_user)
```

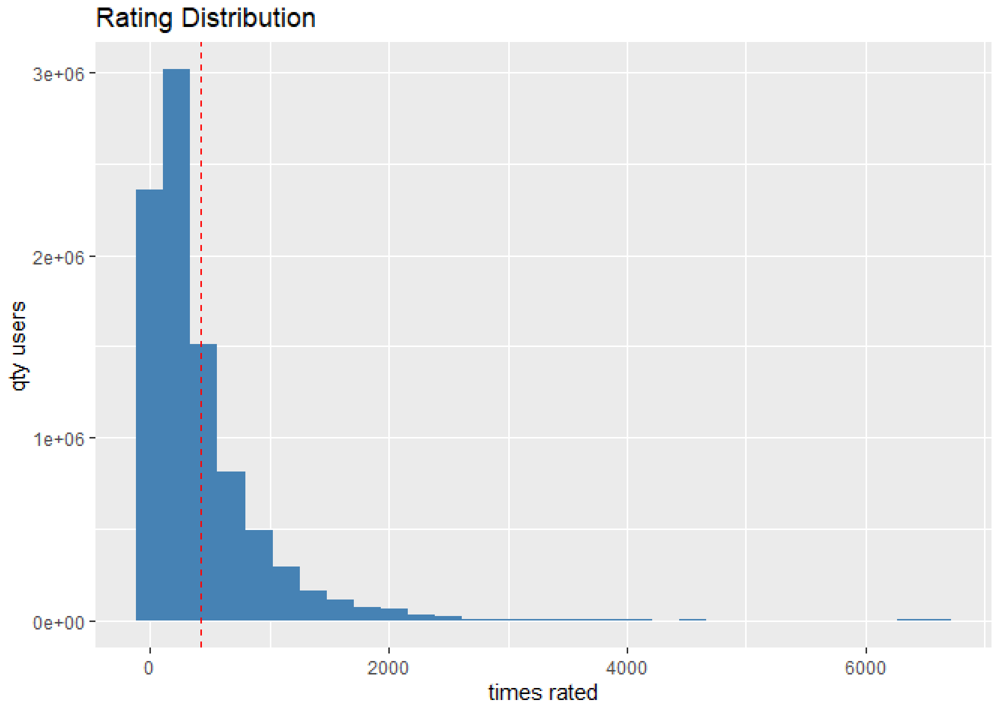
```
## [1] "Average qty of ratings per user is: 424.207504176363"
```

```
##Median of ratings per user
paste0("Median qty of ratings per user is:",
       median(ratings_per_user$n_rating))
```

```
## [1] "Median qty of ratings per user is:257"
```

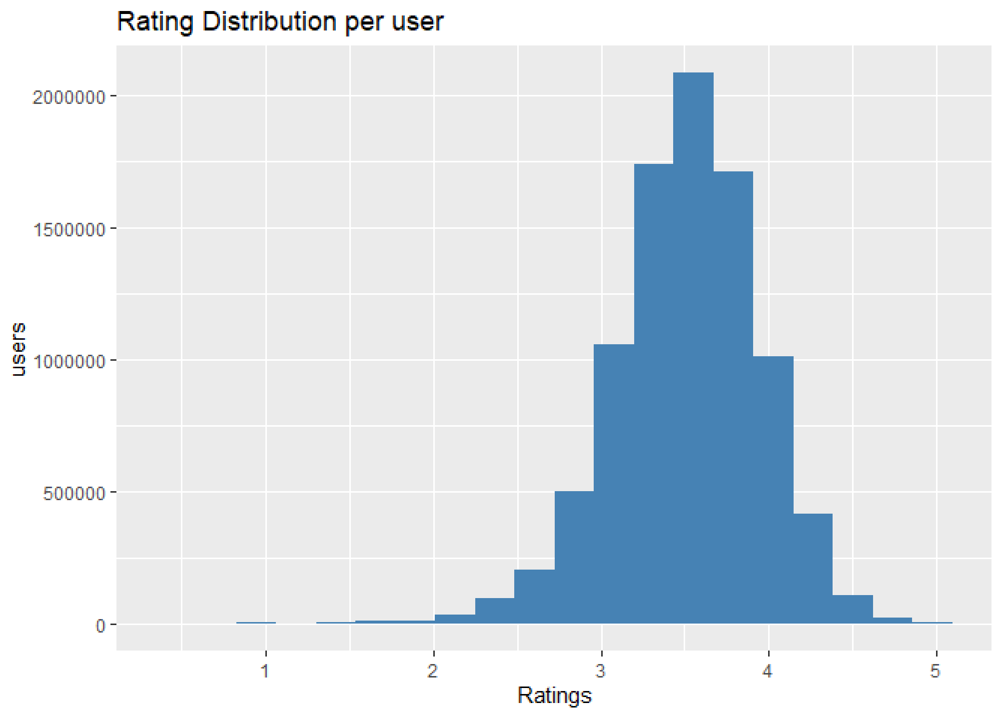
We can observe that the mean quantity of rating per user is 424 and the median is 257, this difference is because there are users that were rated more than usual like user 59269 that rated 6616 times.

```
##distribution of number of rating per user
edx_y%>%group_by(userId)%>%mutate(n=n())%>%ggplot(aes(n))+
  geom_histogram(bins = 30,fill="steelblue")+
  geom_vline(xintercept = mean_rating_per_user, lty = 2, color= "Red")+
  labs(title = "Rating Distribution",x = "times rated", y = "qty users")
```



Most of the users rated movies between 3 and 4. But there is variability on the ratings, so “who” is rating will impact, this is going to be referred as “user effect”. Means that for any given movie the rating change according to the preferences of the user that is rating

```
edx_y%>%group_by(userId)%>%mutate(avg_rat=mean(rating))%>%
  ggplot(aes(avg_rat))+
  geom_histogram(bins = 20,fill="steelblue")+
  labs(title = "Rating Distribution per user",
       x = "Ratings", y = "users")
```



2.1.3.4 Rating vs genres

We can identify 797 different genres, but if we look into the data we will see that this comes from the combination of 20

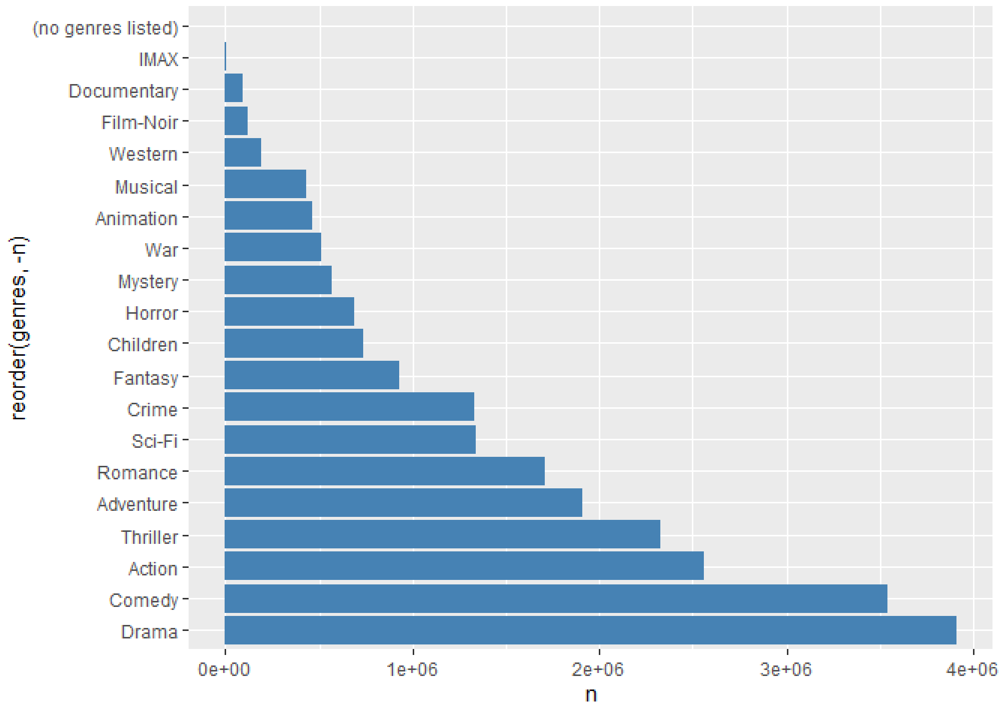
```
#numbers of genres
n_distinct(edx_y$genres)
```

```
## [1] 797
```

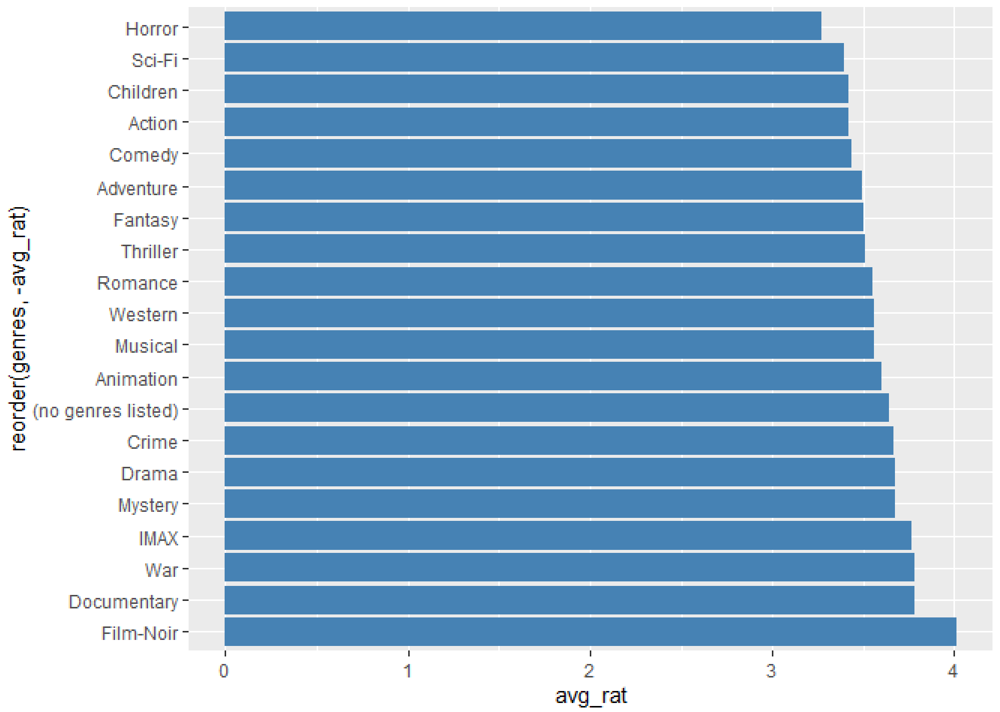
```
##Split genres
genres<-edx_y%>%
  separate_rows(genres, sep = "\\|")%>%group_by(genres)%>%
  summarize(n=n(),avg_rat=mean(rating))%>%
  arrange(desc(avg_rat))
genres%>%kable()
```

genres	n	avg_rat
Film-Noir	118541	4.011625
Documentary	93066	3.783487
War	511147	3.780813
IMAX	8181	3.767693
Mystery	568332	3.677001
Drama	3910127	3.673131
Crime	1327715	3.665925
(no genres listed)	7	3.642857
Animation	467168	3.600644
Musical	433080	3.563305
Western	189394	3.555918
Romance	1712100	3.553813
Thriller	2325899	3.507676
Fantasy	925637	3.501946
Adventure	1908892	3.493544
Comedy	3540930	3.436908
Action	2560545	3.421405
Children	737994	3.418715
Sci-Fi	1341183	3.395743
Horror	691485	3.269815

```
genres%>%ggplot(aes(reorder(genres,-n),n))+
  geom_bar(fill="steelblue",stat="identity")+
  coord_flip()
```




```
genres%>%arrange(desc(avg_rat))%>%
  ggplot(aes(reorder(genres,-avg_rat),avg_rat))+
  geom_bar(fill="steelblue",stat="identity")+
  coord_flip()
```



As conclusions in genre analysis we can see that ar genres that are better rated than others as Film_Noir, Documentary, War and IMAX are the ones with higher average rating and Horror and sci-fi the ones with worst ratings.

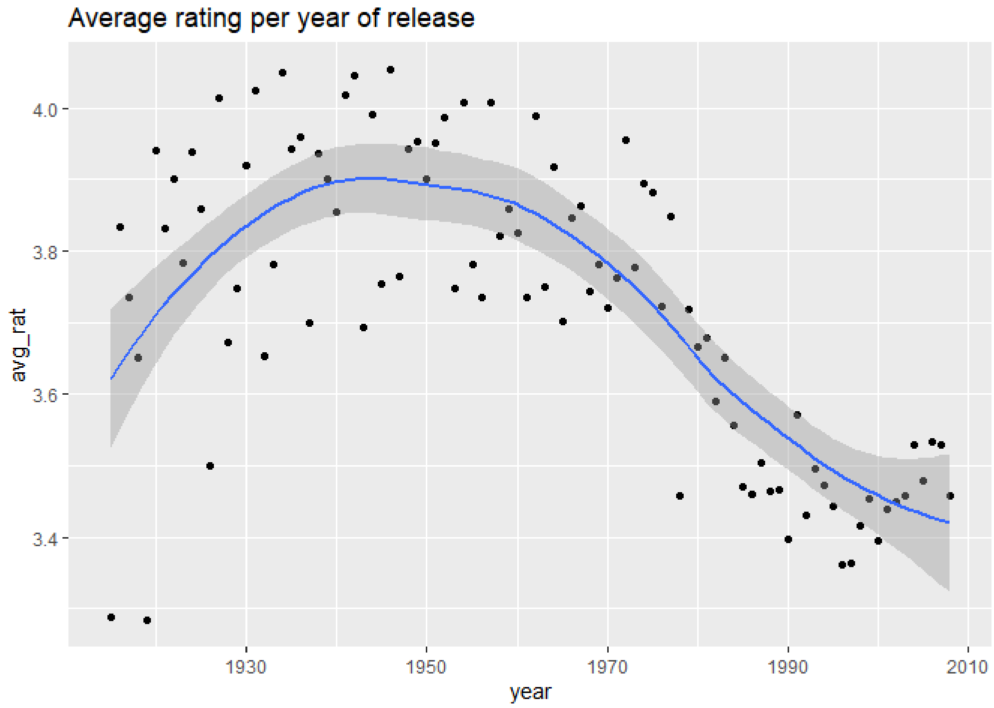
In the database we can observe more quantity of drama and comedy and less documentary , film-noir(the ones with higher ratings)

We can see clearly that genre has effect rating. we will call this genre effect

2.1.3.4 Rating vs year of release

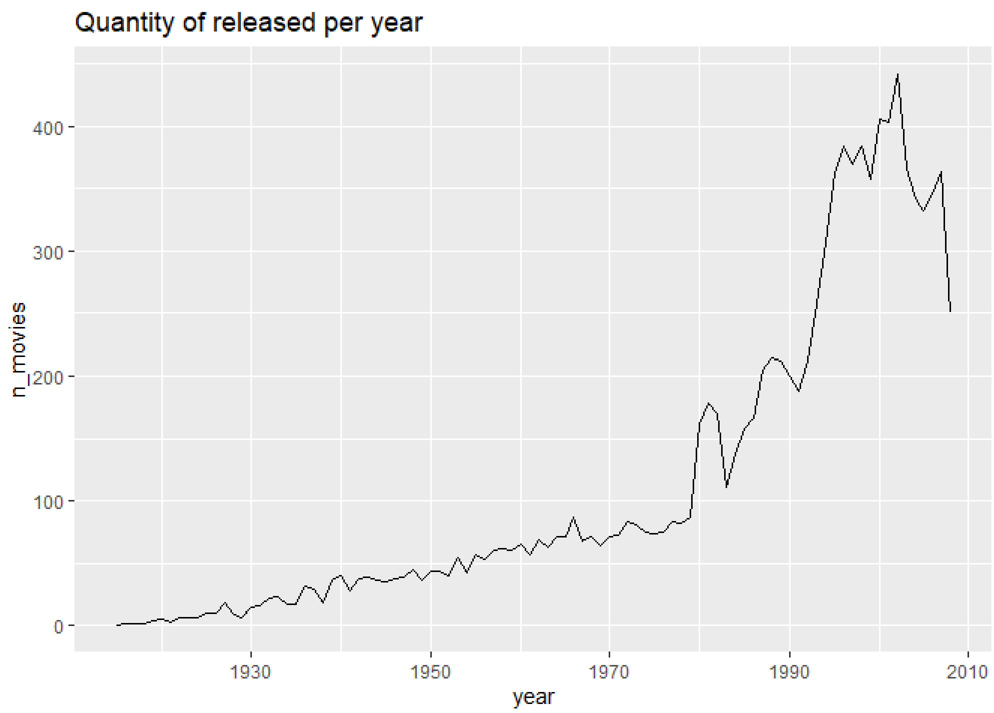
```
##Mean rating per year of release
edx_y%>%group_by(year)%>%summarize(avg_rat=mean(rating))%>%
  ggplot(aes(year,avg_rat))+geom_point()+
```

```
  labs(title = "Average rating per year of release")+
  geom_smooth()
```



We can see that average rating trend increases from 1915 until 1940 and then from 1940 to until the last year there is a decreasing trend in ratings. So we can see that older movies tend to have higher ratings than newer ones. Maybe this is related with database cleaning where some old movies were removed from system due to low ratings.

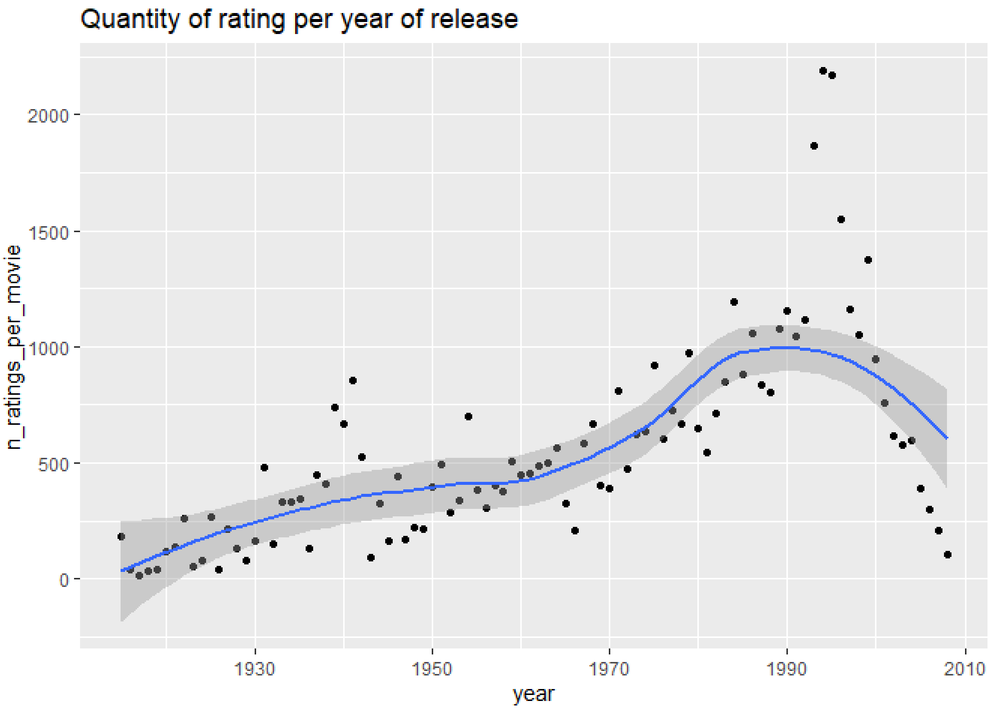
```
##Quantity of ratings per year of release
edx_y%>%group_by(year)%>%
  summarize(n_movies=n_distinct(movieId))%>%
  ggplot(aes(year,n_movies))+geom_line()+
  labs(title = "Quantity of released per year")
```



If we analyze the number of movies released per year we can see an exponential growth.

```
##Quantity of ratings per year of release
edx_y%>%group_by(year)%>%
  summarize(n=n(),n_movies=n_distinct(movieId),
```

```
           n_ratings_per_movie=n/n_movies)%>%
  ggplot(aes(year,n_ratings_per_movie))+
  geom_point()+
  labs(title = "Quantity of rating per year of release")+
  geom_smooth()
```

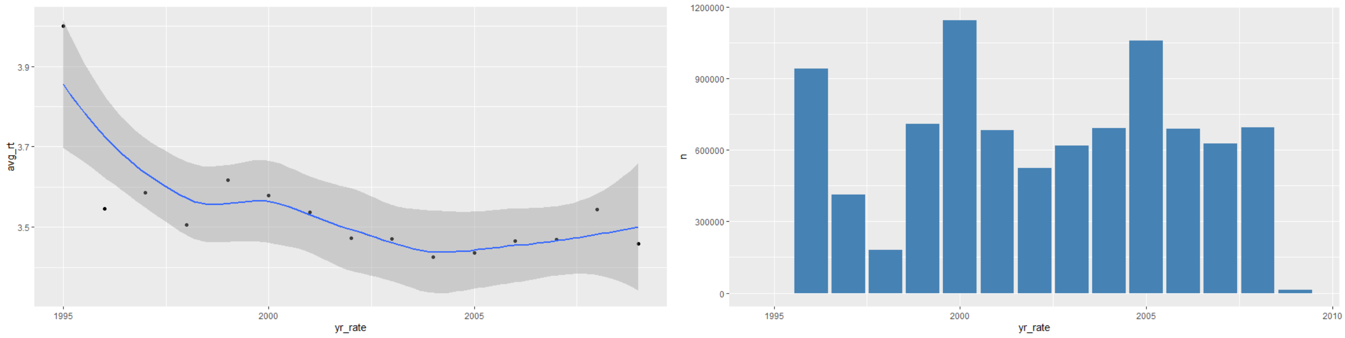


We can also see that average number of ratings per movie increases until 1990 and then starts decreasing. This can be due to more movies in the system and because newer movies had less time to be seen.

2.1.3.5 Rating vs date of rating

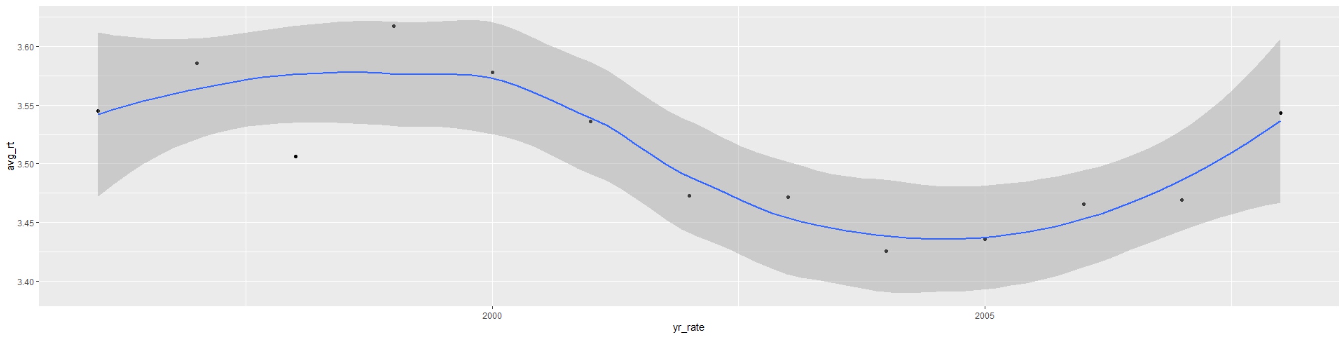
We can observe that average ratings on the first years were higher than actual ones

```
##Average ratings per year of rating
yr1<-edx_y%>%group_by(yr_rate)%>%
  summarize(avg_rt=mean(rating))%>%
  ggplot(aes(yr_rate,avg_rt))+
  geom_point()+geom_smooth()
##Quantity of ratings per year of rating
yr2<-edx_y%>%
  group_by(yr_rate)%>%summarize(n=n())%>%
  ggplot(aes(yr_rate,n))+
  geom_bar(fill="steelblue",stat="identity")

grid.arrange(yr1,yr2, ncol = 2)
```



If we exclude 1995 and 2009 because are years with almost no ratings we observe that we have an effect caused by the year when rating was done

```r
##Average ratings per year of rating
edx_y%>%filter(yr_rate>1995&yr_rate<2009)%>%
  group_by(yr_rate)%>%summarize(avg_rt=mean(rating))%>%
  ggplot(aes(yr_rate,avg_rt))+geom_point()+geom_smooth()
```



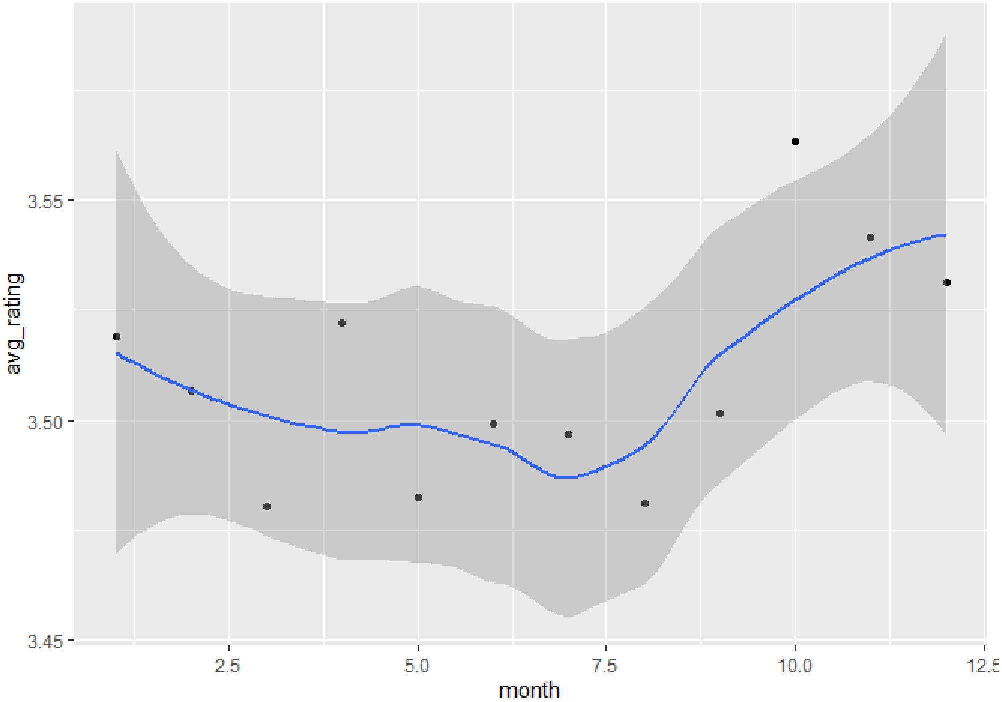
If we analyze day of the week we can observe that there is almost no difference in ratings.

```r
##Average ratings per year of rating
edx_y%>%mutate(day=weekdays(rate_date))%>%
  group_by(day)%>%summarize(avg_rating=mean(rating))%>%
  kable()
```

day	avg_rating
Friday	3.512657
Monday	3.516998
Saturday	3.528958
Sunday	3.517915
Thursday	3.500727
Tuesday	3.510664
Wednesday	3.501416

For months we can observe that average ratings tend to decrease until july and then to increase until end of year (not very big effect)

```r
##Average ratings per year of rating
edx_y%>%mutate(month=month(rate_date))%>%group_by(month)%>%
  summarize(avg_rating=mean(rating))%>%
  ggplot(aes(month,avg_rating))+
  geom_point()+geom_smooth()
```



2.2 Method

For this analysis i will follow the analysis that was done in the course and then add new approach to get better RMSE,

Model1= I will start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. And this prediction will be average rating.

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

with $\varepsilon_{i,u}$ independent errors sampled from the same distribution centered at 0 and μ the "true" rating for all movies. We know that the estimate that minimizes the

Model 2=

To add more complexity I will include the effect of movie, because there are movies that are rated better than others. This intuition, that different movies are rated differently, is confirmed by data. so we can add to the previous model the term bi to represent average ranking for movie i and we will call it "movie effect".

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

Model 3=

Then I will add user effect, this will explain the variability that we observe when different users rate the same movie, some people will love it and some will hate it so i will try to represent that by adding the term bu to the estimation. This will be called "User Effect"

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

Model 4=

Also will analyze aging effect, we saw in the data that the year the movie was released affects the rating, so I will include this effect. This will be considered as "Aging effect"

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

Model 5=

As we saw in data exploration some genres tend to be better rated than others. This will considered as "Genre Effect"

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

Model 6 to 9=

I will regularize to penalize in all of the cases the effect of small groups when using averages.

instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

1/N sum(yu,i-μ-bi)^2 + lambda sum(bi^2) **(MODEL6)**

1/N sum(yu,i-μ-bi-bu)^2 + lambda sum(bi^2 + bu^2) **(MODEL7)**

1/N sum(yu,i-μ-bi-bu-by)^2 + lambda sum(bi^2 + bu^2 +by^2) **(MODEL8)**

1/N sum(yu,i-μ-bi-bu-by-bg)^2 + lambda sum(bi^2 + bu^2 + by^2 +bg^2) **(MODEL9)**

Model 10=

I will include the effect of the date when movie was rated and will regularize at the same time this effect.

1/N sum(yu,i-μ-bi-bu-by-bg-bdr)^2 + lambda sum(bi^2 + bu^2 + by^2 +bg^{2+bdr}2)

Model 11=

To add some new contents i will do matrix factorization using the package recosystem. Matrix factorization is a class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices

3-Results

We are going to create a function to measure the RMSE

```
##Crete RMSE function to evaluate all models
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

And because Validation test can only be used for final algorithm, I am are going to divide the training set (edx_y) in train and test set

```
###divide training set in training and test to use when tuning parameters
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx_y$rating, times = 1, p = 0.1, list = FALSE)
train<-edx_y[-test_index,]
temp<-edx_y[test_index,]

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

# Add rows removed from validation set back into train set
removed <- anti_join(temp, test)
train <- rbind(train, removed)
rm(test_index, temp,removed)
```

MODEL 1: asume same rating for all: Yu,i= μ + ϵu,i

We start with a model that assumes the same rating for all movies and all users, with all the differences explained by random variation: If μ represents the true rating for all movies and users and ϵ represents independent errors sampled from the same distribution centered at zero, then:

Yu,i=μ+ϵu,i

```
##Calculate mean rating
mu_hat<-mean(train$rating)
##evaluate RMSE of the model that assumes all ratings the same
RMSE_model1<-RMSE(test$rating,mu_hat)
##Save information in a table
RMSE_1<-data.frame(Method= "Just mean", RMSE= RMSE_model1)
knitr::kable(RMSE_1)
```

Method	RMSE
Just mean	1.060054

we get 1.060054 as RMSE that is far from RMSE requested for the project

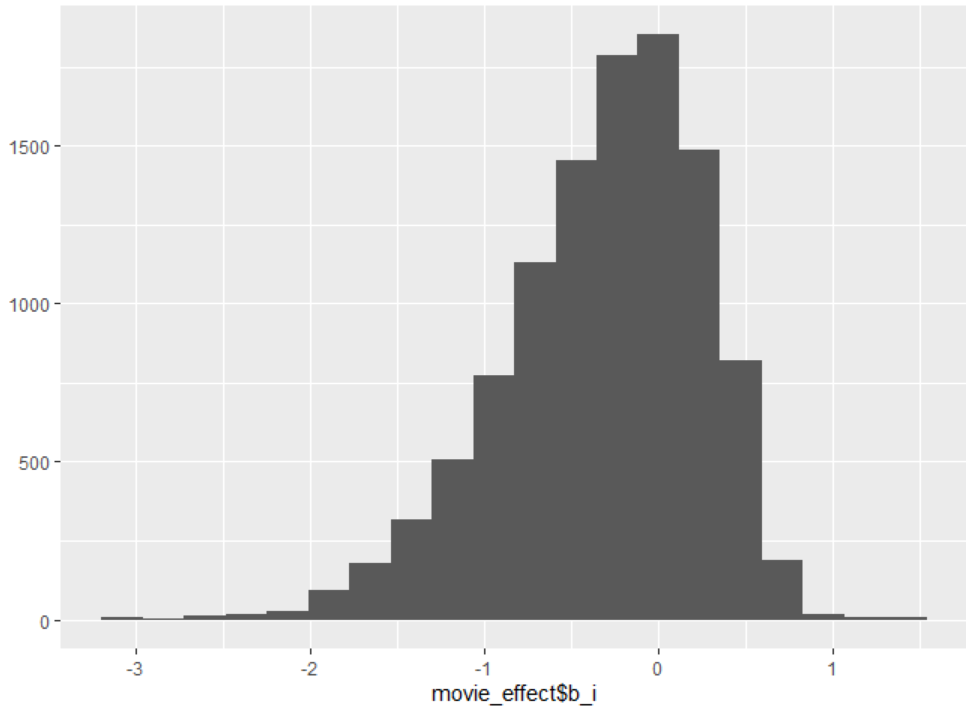
MODEL 2: considering movie effect $Y_{u,i} = \mu + b_i + \epsilon_{u,i}$

On Previous charts we saw that ratings deferred from one movie to other, some have higher ratings and some lower, so to reduce the error we are going to consider this movie effect. If we think logically is obvious that there are movies that are more liked by users

```
## we group by movie and extract mean to actual rating and we get the movie effect
movie_effect<-train%>%group_by(movieId)%>%summarize(b_i=mean(rating-mu_hat))
```

It can be observed that the effects goes from almost less than -3 stars to more than one star

```
qplot(movie_effect$b_i,bins=20)
```




```
#we predict values of test set
predicted_rating_model2 <-test %>%
  left_join(movie_effect, by='movieId') %>%
  mutate(pred = mu_hat + b_i)%>%
  pull(pred)
```

Calculate RMSE for this method

```
#Calculate RMSE for this method
RMSE_model2<-RMSE(test$rating,predicted_rating_model2)
RMSE_2<-data.frame(Method= "mean + b_i", RMSE= RMSE_model2)
```

```
RMSE_results<-bind_rows(RMSE_1,RMSE_2)
knitr::kable(RMSE_results)
```

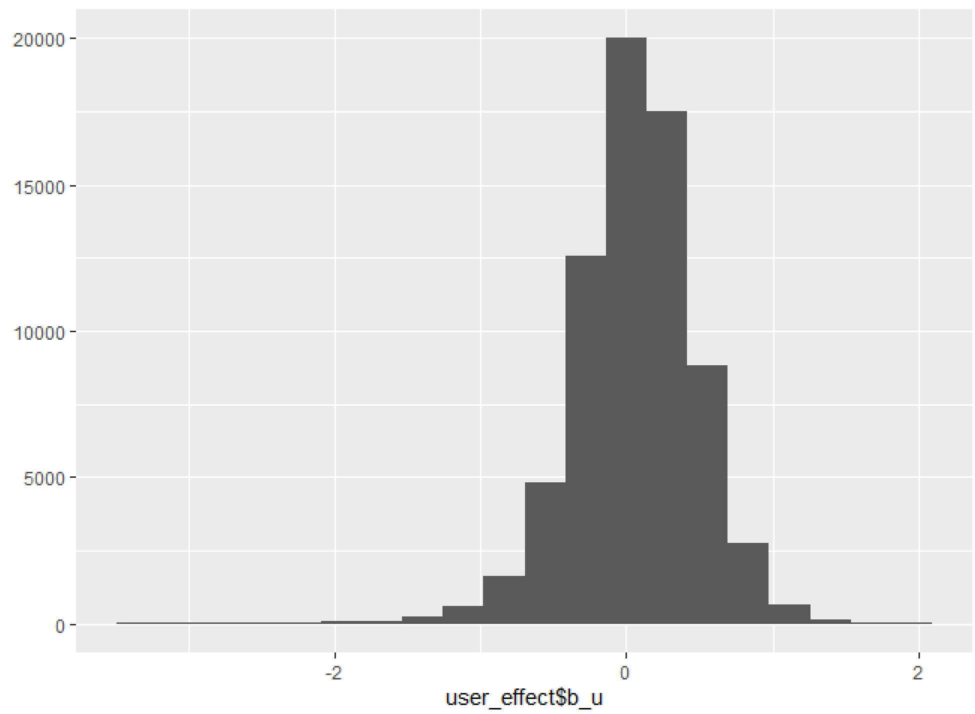
Method	RMSE
Just mean	1.0600537
mean + b_i	0.9429615

There was a significant reduction in the RMSE model but not enough for the requirement

MODEL 3: Consider user effect Yu,i=μ+ bi+ + bu + εu,i

On this case we are going to consider user preferences, there are users that tend to rate higher or less some movies. So to the movie efect we are going to add user effect

```
user_effect<-train%>%group_by(userId)%>%
  left_join(movie_effect, by='movieId') %>%
  summarize(b_u=mean(rating-mu_hat-b_i))
##Plot histogram of user effect to observe variability
qplot(user_effect$b_u,bins=20)
```




```
##Predict values
predicted_rating_model3 <-test %>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  mutate(pred = mu_hat + b_i +b_u)%>%
  pull(pred)
##evaluate RMSE
RMSE_model3<-RMSE(test$rating,predicted_rating_model3)
RMSE_3<-data.frame(Method= "mean + b_i + b_u",RMSE= RMSE_model3)
##Print result
RMSE_results<-bind_rows(RMSE_1,RMSE_2,RMSE_3)
knitr::kable(RMSE_results)
```

Method	RMSE
Just mean	1.0600537

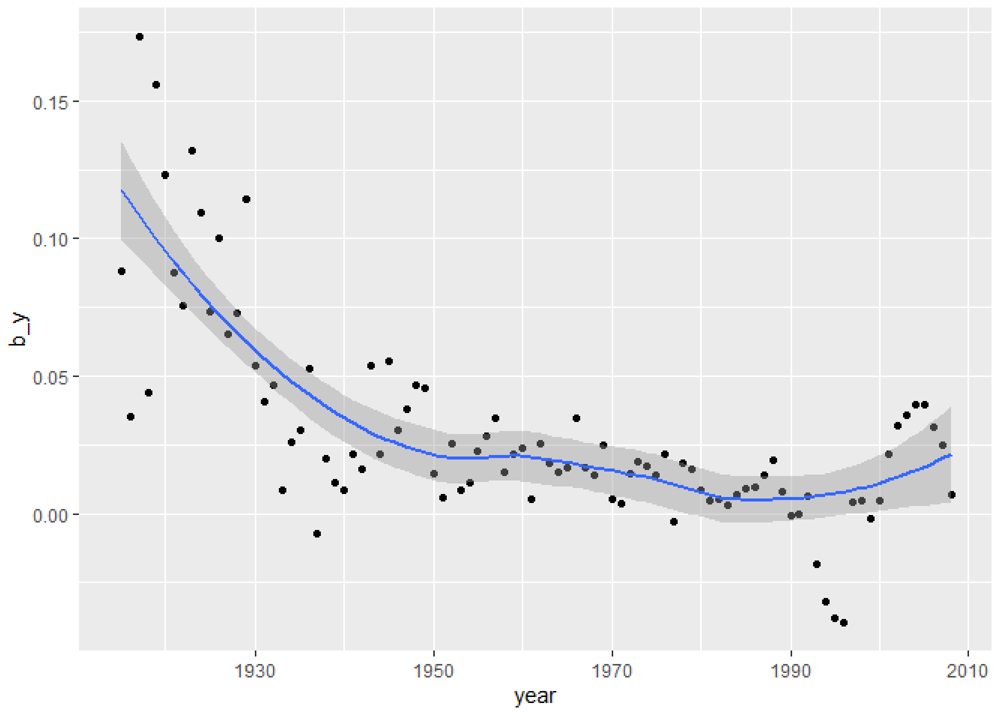
Method	RMSE
mean + b_i	0.9429615
mean + b_i + b_u	0.8646843

It seems that this is an important effect because the error decreased, but still not enough

MODEL 4 Consider aging effect Y u,i= =μ + b_u + b_y + err_u,i

There is some evidence that age of the movie impact in rating we can see in the chart that old movies tend to have higher ratings (maybe this effect is because they constantly update databases so if a movie does not succeed they delete it from the system, and in new movies there is more variability until they clean databases)

```r
##Calculate aging effect
year_effect<-train%>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  group_by(year)%>%
  summarize(b_y=mean(rating-mu_hat-b_i-b_u))

#Plot aging efect to se variability

year_effect%>%ggplot(aes(year,b_y))+geom_point()+geom_smooth()
```



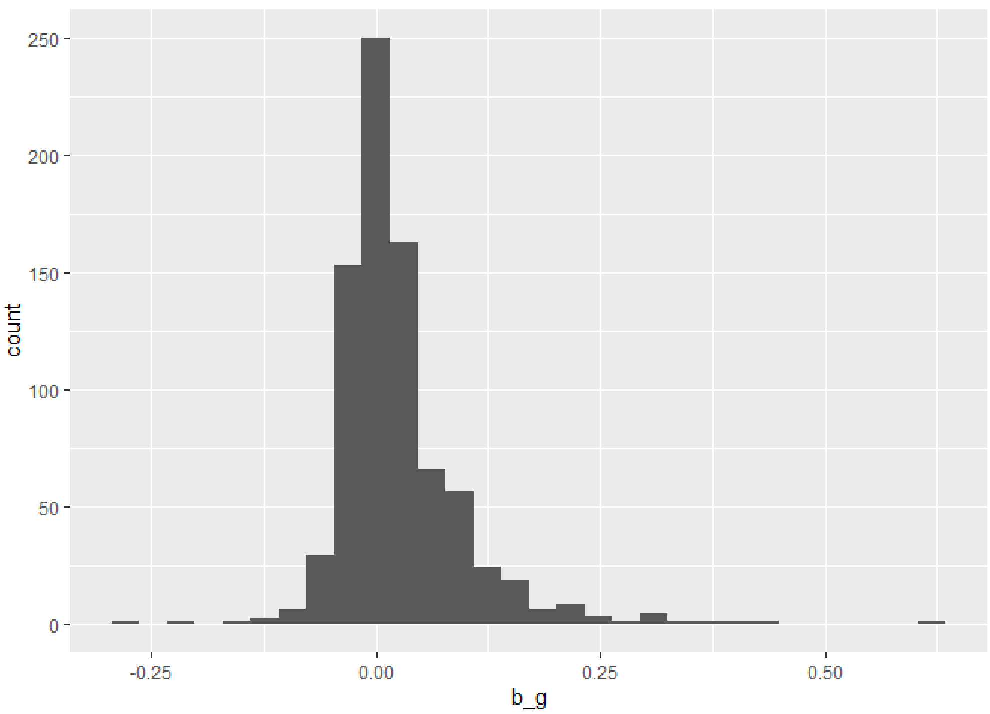

```r
##Predict ratings using all the effects
predicted_rating_model4 <- test %>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  left_join(year_effect, by='year') %>%
  mutate(pred = mu_hat + b_i +b_u+b_y)%>%
  pull(pred)
#calculate RMSE
RMSE_model4<-RMSE(test$rating,predicted_rating_model4)
RMSE_4<-data.frame(Method= "mean + b_i + b_u + b_y",
                   RMSE= RMSE_model4)
#Print result
RMSE_results<-bind_rows(RMSE_1,RMSE_2,RMSE_3,RMSE_4)
knitr::kable(RMSE_results)
```

Method	RMSE
Just mean	1.0600537
mean + b_i	0.9429615
mean + b_i + b_u	0.8646843
mean + b_i + b_u + b_y	0.8643301

MODEL 5: Consider genre Y u,i= mu+b_i + b_u +b_y + b_g + err_u,i

On this case we will add the effect of different genres (for simplicity we will consider each combination of different genres as one genre). as it can be seen in the chart there is some variation based on genre that affects the rating.

```
##Calculate genre effect
genre_effect<-train%>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  left_join(year_effect, by='year') %>%
  group_by(genres)%>%
  summarize(b_g=mean(rating-mu_hat-b_i-b_u-b_y))

##Plot to see variability
genre_effect%>%ggplot(aes(b_g))+geom_histogram()
```




```
#Predict values using all effects
predicted_rating_model5 <- test %>%
  left_join(movie_effect, by='movieId') %>%
  left_join(user_effect, by='userId') %>%
  left_join(year_effect, by='year') %>%
  left_join(genre_effect, by='genres') %>%
  mutate(pred = mu_hat + b_i +b_u+b_y+b_g)%>%
  pull(pred)
##Calculate RMSE
RMSE_model5<-RMSE(test$rating,predicted_rating_model5)
RMSE_5<-data.frame(Method= "mean + b_i + b_u + b_y + b_g",
                   RMSE= RMSE_model5)
#Print result
```

```
RMSE_results<-bind_rows(RMSE_1,RMSE_2,RMSE_3,RMSE_4,RMSE_5)
knitr::kable(RMSE_results)
```

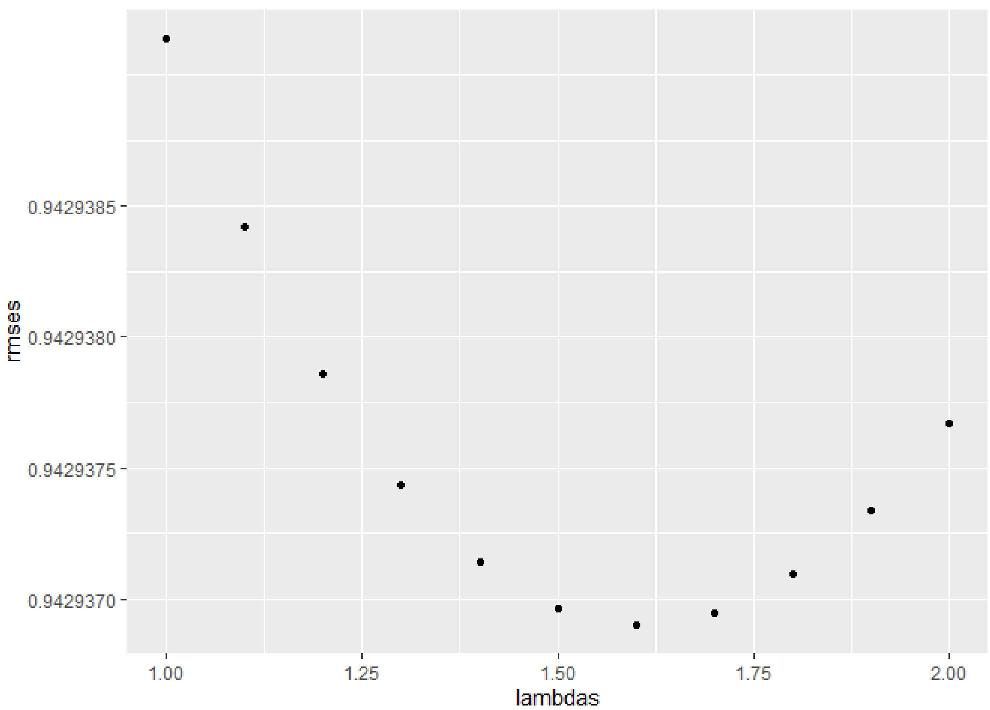
Method	RMSE
Just mean	1.0600537
mean + b_i	0.9429615
mean + b_i + b_u	0.8646843
mean + b_i + b_u + b_y	0.8643301
mean + b_i + b_u + b_y + b_g	0.8640801

REGULARIZATION

MODEL 6: REGULARIZED Consider movie effect

Regularization is an important tool to reduce the effect of small frequencies in data, for example on this cases movies that received small quantity of ratings

```
  ##First find best tuning of lambda
  lambdas<-seq(1,2,0.1)
##For all lambdas calculate rmses
  rmses <- sapply(lambdas, function(l){
   #calculate movie effect
     movie_effect_reg <- train %>%
      group_by(movieId)%>%
      summarize(b_i = sum(rating - mu_hat)/(n()+l))
    #Predict regularized movie effect
    predicted_ratings_model_6 <- test %>%
      left_join(movie_effect_reg, by = "movieId") %>%
      mutate(pred = mu_hat + b_i) %>%
      pull(pred)
    return(RMSE(predicted_ratings_model_6, test$rating))
  })
  ##Select lambda with lower RMSES
  qplot(lambdas, rmses)
```




```
  lambdas[which.min(rmses)]
```

```
## [1] 1.6
```

```
  ##Calculate model using optimum lambda
  lambda<-lambdas[which.min(rmses)]
  ##movie effect
  movie_effect_reg <- train %>%
    group_by(movieId)%>%
    summarize(b_i = sum(rating - mu_hat)/(n()+lambda))
  ##predict values
  predicted_rating_model6 <- test  %>%
    left_join(movie_effect_reg, by = "movieId")%>%
    mutate(pred = mu_hat + b_i) %>%
    pull(pred)
    #calculate RMSE
  RMSE_model6<-RMSE(test$rating,predicted_rating_model6)
  RMSE_6<-data.frame(Method= "mean + b_i REGULARIZED",
                   RMSE= RMSE_model6)
#Print values
  RMSE_results<-bind_rows(RMSE_1,RMSE_2,RMSE_6,RMSE_3,RMSE_4,RMSE_5)
knitr::kable(RMSE_results)
```

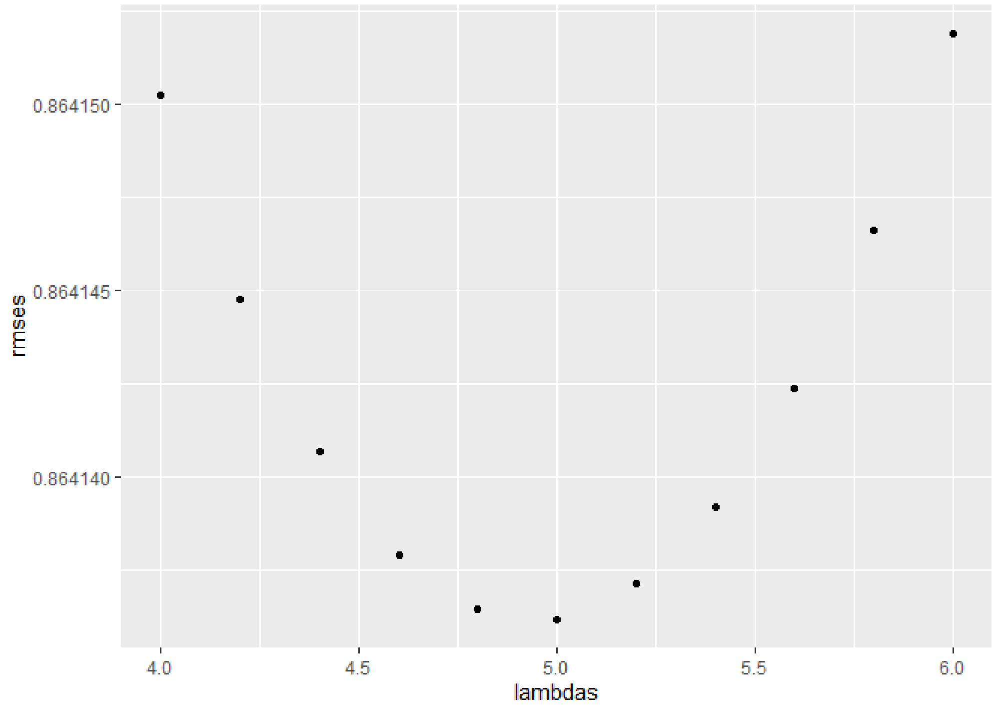
Method	RMSE
Just mean	1.0600537
mean + b_i	0.9429615
mean + b_i REGULARIZED	0.9429369
mean + b_i + b_u	0.8646843
mean + b_i + b_u + b_y	0.8643301
mean + b_i + b_u + b_y + b_g	0.8640801

MODEL 7: REGULARIZED Consider movie effect and user effect

un this case we will analyze regularization including also user effect, to weight less users with less ratings

```
 ##First find best tuning of lambda
lambdas<-seq(4,6,0.2)
  ##For all lambdas calculate rmses
  rmses <- sapply(lambdas, function(l){
    ##movie effect
    movie_effect_reg <- train %>%
      group_by(movieId)%>%
      summarize(b_i = sum(rating - mu_hat)/(n()+l))
    #user effect
    user_effect_reg <- train %>%
      left_join(movie_effect_reg, by="movieId") %>%
      group_by(userId)%>%
      summarize(b_u = sum(rating - b_i- mu_hat)/(n()+l))
    #predict calues
    predicted_ratings_model_7 <- test %>%
      left_join(movie_effect_reg, by = "movieId") %>%
      left_join(user_effect_reg, by = "userId") %>%
      mutate(pred = mu_hat + b_i + b_u) %>%
      pull(pred)
    #save all RMSES
    return(RMSE(predicted_ratings_model_7, test$rating))
  })
```

```r
  #plot lambda vs accuracy
  qplot(lambdas, rmses)
```




```r
  #select lambda that optimizes RMSE
  lambdas[which.min(rmses)]
```

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

```r
  ##calculate effects with optimum lambda
      lambda<-lambdas[which.min(rmses)]
    ##movie effect
        movie_effect_reg <- train %>%
        group_by(movieId)%>%
        summarize(b_i = sum(rating - mu_hat)/(n()+lambda))
      #user effect
      user_effect_reg <- train %>%
        left_join(movie_effect_reg, by="movieId") %>%
        group_by(userId)%>%
        summarize(b_u = sum(rating - b_i- mu_hat)/(n()+lambda))
      #prediction
      predicted_rating_model7 <- test %>%
        left_join(movie_effect_reg, by = "movieId") %>%
        left_join(user_effect_reg, by = "userId") %>%
        mutate(pred = mu_hat + b_i + b_u) %>%
        pull(pred)
      #RMSE final calculation
      RMSE_model7<- RMSE(predicted_rating_model7, test$rating)
     RMSE_model7<-RMSE(test$rating,predicted_rating_model7)
     ##Print values
  RMSE_7<-data.frame(Method= "mean + b_i + b_u REGULARIZED", RMSE= RMSE_model7)
RMSE_results<-bind_rows(RMSE_1,RMSE_2,RMSE_6,RMSE_3,RMSE_7,RMSE_4,RMSE_5)
knitr::kable(RMSE_results)
```

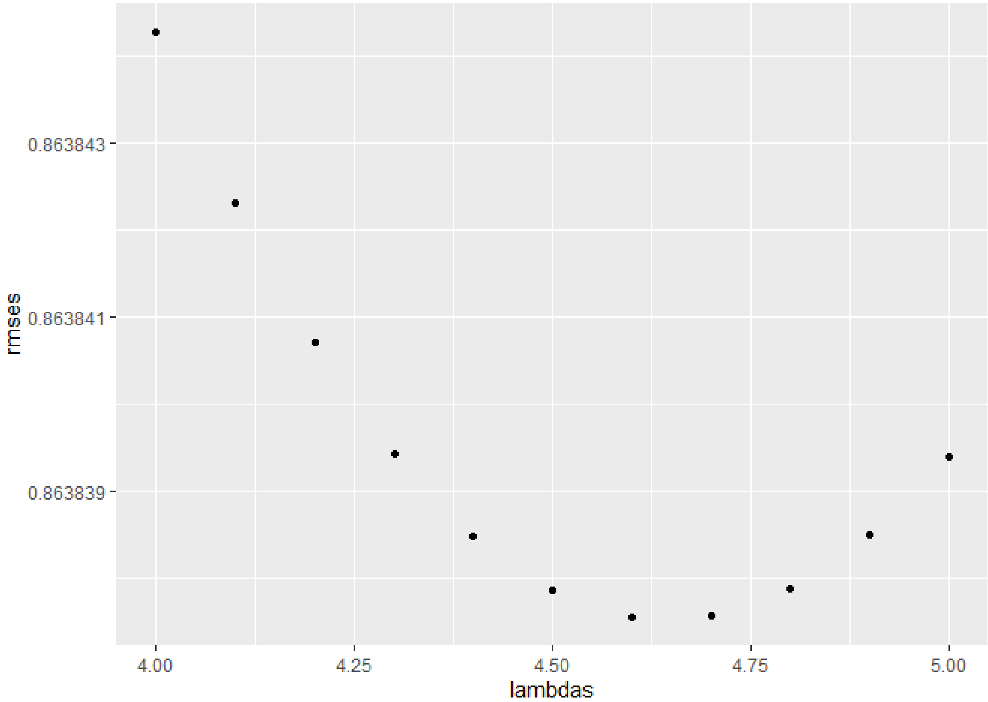
Method	RMSE
Just mean	1.0600537
mean + b_i	0.9429615

Method	RMSE
mean + b_i REGULARIZED	0.9429369
mean + b_i + b_u	0.8646843
mean + b_i + b_u REGULARIZED	0.8641362
mean + b_i + b_u + b_y	0.8643301
mean + b_i + b_u + b_y + b_g	0.8640801

MODEL 8:REGULARIZED Consider movie effect, user effect, aging effect

First we need to find the best lambda to adjust the model and then create a prediction in the test set with that value

```
##first select optimum lambda

      lambdas<-seq(4,5,.1)
      ##create function to calculate rmses for all lambdas
      rmses <- sapply(lambdas, function(l){
        #Movie effect
        movie_effect_reg <- train %>%
          group_by(movieId)%>%
          summarize(b_i = sum(rating - mu_hat)/(n()+l))
        #User effect
        user_effect_reg <- train %>%
          left_join(movie_effect_reg, by="movieId") %>%
          group_by(userId)%>%
          summarize(b_u = sum(rating - b_i- mu_hat)/(n()+l))
        #Aging effect
        year_effect_reg<-train%>%
          left_join(movie_effect_reg, by='movieId') %>%
          left_join(user_effect_reg, by='userId') %>%
          group_by(year)%>%
          summarize(b_y=sum(rating-mu_hat-b_i-b_u)/(n()+l))
        #predict values
        predicted_ratings_model_8 <- test %>%
          left_join(movie_effect_reg, by = "movieId") %>%
          left_join(user_effect_reg, by = "userId") %>%
          left_join(year_effect_reg, by = "year") %>%
          mutate(pred = mu_hat + b_i + b_u + b_y) %>%
          pull(pred)
        ##Save lambda
        return(RMSE(predicted_ratings_model_8, test$rating))
      })
      #plot lambda vs accuracy
      qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 4.6
```

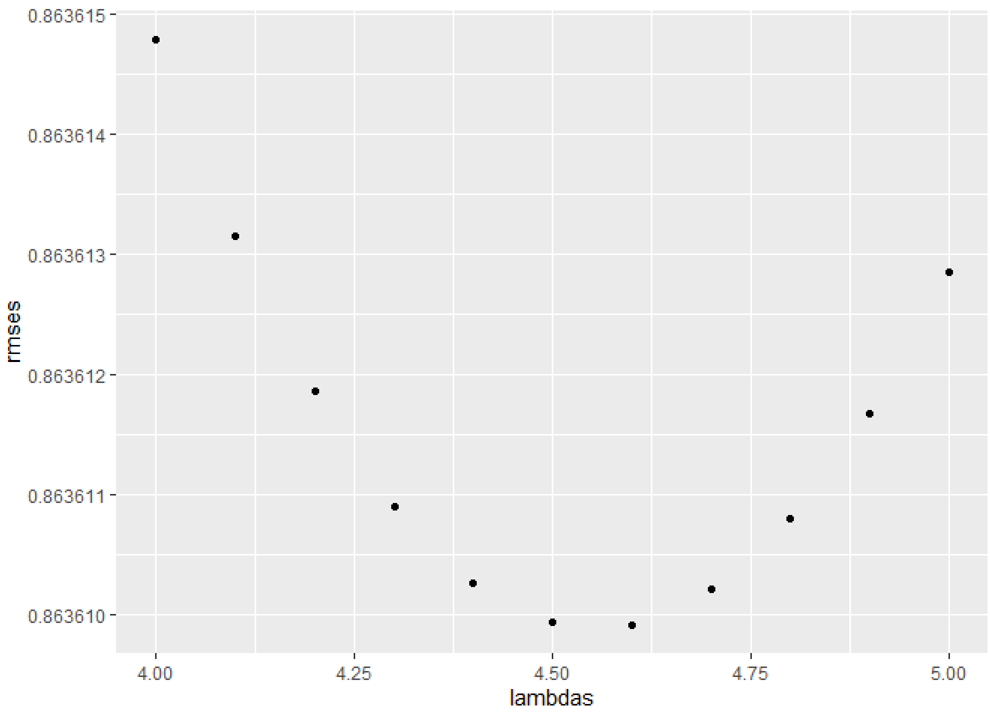
```
##Calculate final prediction with optimum lambda
lambda<-lambdas[which.min(rmses)]

##Movie effect
movie_effect_reg <- train %>%
  group_by(movieId)%>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda))
##User effect
user_effect_reg <- train %>%
  left_join(movie_effect_reg, by="movieId") %>%
  group_by(userId)%>%
  summarize(b_u = sum(rating - b_i- mu_hat)/(n()+lambda))
##Year effect
year_effect_reg<-train%>%
  left_join(movie_effect_reg, by='movieId') %>%
  left_join(user_effect_reg, by='userId') %>%
  group_by(year)%>%
  summarize(b_y=sum(rating-mu_hat-b_i-b_u)/(n()+lambda))
##Prediction
predicted_rating_model8 <- test %>%
  left_join(movie_effect_reg, by = "movieId") %>%
  left_join(user_effect_reg, by = "userId") %>%
  left_join(year_effect_reg, by = "year") %>%
  mutate(pred = mu_hat + b_i + b_u +b_y) %>%
  pull(pred)
##RMSE
RMSE_model8<- RMSE(predicted_rating_model8, test$rating)
##Save ad print value
RMSE_8<-data.frame(Method= "mean + b_i + b_u + b_y REGULARIZED",
                   RMSE= RMSE_model8)
RMSE_results<-bind_rows(RMSE_1,RMSE_2,RMSE_6,RMSE_3,RMSE_7,
                        RMSE_4,RMSE_8,RMSE_5)
knitr::kable(RMSE_results)
```

Method	RMSE

Method	RMSE
Just mean	1.0600537
mean + b_i	0.9429615
mean + b_i REGULARIZED	0.9429369
mean + b_i + b_u	0.8646843
mean + b_i + b_u REGULARIZED	0.8641362
mean + b_i + b_u + b_y	0.8643301
mean + b_i + b_u + b_y REGULARIZED	0.8638376
mean + b_i + b_u + b_y + b_g	0.8640801

MODEL 9: Consider movie effect, user effect, year effect and genre effect REGULARIZED

```
##First calculate lambda
lambdas<-seq(4,5,.1)
##Calculate rmses for all lambdas
rmses <- sapply(lambdas, function(l){
  #Movie effect
  movie_effect_reg <- train %>%
    group_by(movieId)%>%
    summarize(b_i = sum(rating - mu_hat)/(n()+l))
  ##User effect
  user_effect_reg <- train %>%
    left_join(movie_effect_reg, by="movieId") %>%
    group_by(userId)%>%
    summarize(b_u = sum(rating - b_i- mu_hat)/(n()+l))
  ###Year effect
  year_effect_reg<-train%>%
    left_join(movie_effect_reg, by='movieId') %>%
    left_join(user_effect_reg, by='userId') %>%
    group_by(year)%>%
    summarize(b_y=sum(rating-mu_hat-b_i-b_u)/(n()+l))
  ##Genre effect
  genre_effect_reg<-train%>%
    left_join(movie_effect_reg, by='movieId') %>%
    left_join(user_effect_reg, by='userId') %>%
    left_join(year_effect_reg, by='year') %>%
    group_by(genres)%>%
    summarize(b_g=sum(rating-mu_hat-b_i-b_u-b_y)/(n()+l))
  ##Predict values
  predicted_ratings_model_9 <- test %>%
    left_join(movie_effect_reg, by = "movieId") %>%
    left_join(user_effect_reg, by = "userId") %>%
    left_join(year_effect_reg, by = "year") %>%
    left_join(genre_effect_reg, by = "genres") %>%
    mutate(pred = mu_hat + b_i + b_u + b_y + b_g) %>%
    pull(pred)
  #Return RMSES
  return(RMSE(predicted_ratings_model_9, test$rating))
})
#Plot lambda vs RMSES
qplot(lambdas, rmses)
```




```r
lambdas[which.min(rmses)]
```

```
## [1] 4.6
```

```r
##Select optimum lambda
min(rmses)
```

```
## [1] 0.8636099
```

```r
##FInal calculation

l=lambdas[which.min(rmses)]
##Movie effect
movie_effect_reg <- train %>%
  group_by(movieId)%>%
  summarize(b_i = sum(rating - mu_hat)/(n()+l))
##User effect
user_effect_reg <- train %>%
  left_join(movie_effect_reg, by="movieId") %>%
  group_by(userId)%>%
  summarize(b_u = sum(rating - b_i- mu_hat)/(n()+l))
##Year effect
year_effect_reg<-train%>%
  left_join(movie_effect_reg, by='movieId') %>%
  left_join(user_effect_reg, by='userId') %>%
  group_by(year)%>%
  summarize(b_y=sum(rating-mu_hat-b_i-b_u)/(n()+l))
##Genre effect
genre_effect_reg<-train%>%
  left_join(movie_effect_reg, by='movieId') %>%
  left_join(user_effect_reg, by='userId') %>%
  left_join(year_effect_reg, by='year') %>%
  group_by(genres)%>%
  summarize(b_g=sum(rating-mu_hat-b_i-b_u-b_y)/(n()+l))
##Predictions
predicted_ratings_model_9 <- test %>%
  left_join(movie_effect_reg, by = "movieId") %>%
  left_join(user_effect_reg, by = "userId") %>%
```

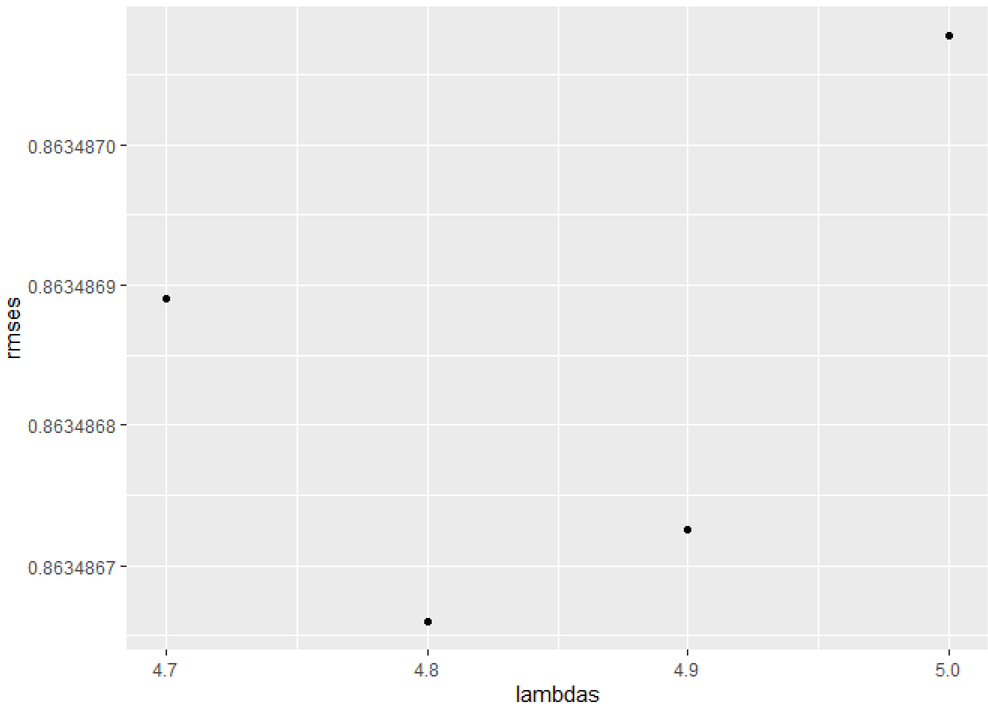
```
        left_join(year_effect_reg, by = "year") %>%
        left_join(genre_effect_reg, by = "genres") %>%
        mutate(pred = mu_hat + b_i + b_u + b_y + b_g) %>%
        pull(pred)
      ##Calculate RMSES
RMSE_model9<-RMSE(predicted_ratings_model_9, test$rating)
RMSE_9<-data.frame(
Method= "mean + b_i + b_u + b_y + b_dr REGULARIZED",
RMSE= RMSE_model9)
RMSE_results<-bind_rows(RMSE_1,RMSE_2,RMSE_6,RMSE_3,
                        RMSE_7,RMSE_4,RMSE_8,RMSE_5,RMSE_9)

knitr::kable(RMSE_results)
```

Method	RMSE
Just mean	1.0600537
mean + b_i	0.9429615
mean + b_i REGULARIZED	0.9429369
mean + b_i + b_u	0.8646843
mean + b_i + b_u REGULARIZED	0.8641362
mean + b_i + b_u + b_y	0.8643301
mean + b_i + b_u + b_y REGULARIZED	0.8638376
mean + b_i + b_u + b_y + b_g	0.8640801
mean + b_i + b_u + b_y + b_dr REGULARIZED	0.8636099

MODEL 10: Consider movie effect, user effect, year effect, rating date effect, genre effect and year of rating REGULARIZED

```
##First select optimum lambda

lambdas<-seq(4.7,5,.1)
      ##Calculate rmses for all lambdas
      rmses <- sapply(lambdas, function(l){
        ##Movie effect
        movie_effect_reg <- train %>%
          group_by(movieId)%>%
          summarize(b_i = sum(rating - mu_hat)/(n()+l))
        ##User effect
        user_effect_reg <- train %>%
          left_join(movie_effect_reg, by="movieId") %>%
          group_by(userId)%>%
          summarize(b_u = sum(rating - b_i- mu_hat)/(n()+l))
        ##Aging effect
        year_effect_reg<-train%>%
          left_join(movie_effect_reg, by='movieId') %>%
          left_join(user_effect_reg, by='userId') %>%
          group_by(year)%>%
          summarize(b_y=sum(rating-mu_hat-b_i-b_u)/(n()+l))
        ##Genre effect
        genre_effect_reg<-train%>%
          left_join(movie_effect_reg, by='movieId') %>%
          left_join(user_effect_reg, by='userId') %>%
          left_join(year_effect_reg, by='year') %>%
          group_by(genres)%>%
          summarize(b_g=sum(rating-mu_hat-b_i-b_u-b_y)/(n()+l))
```

```
    ##Date of rating effect
      daterating_effect_reg<-train%>%
      left_join(movie_effect_reg, by='movieId') %>%
      left_join(user_effect_reg, by='userId') %>%
      left_join(year_effect_reg, by='year') %>%
      left_join(genre_effect_reg, by = "genres") %>%
      group_by(yr_rate)%>%
      summarize(b_dr=sum(rating-mu_hat-b_i-b_u-b_y-b_g)/(n()+l))
    ##Prediction
    predicted_ratings_model_10 <- test %>%
      left_join(movie_effect_reg, by = "movieId") %>%
      left_join(user_effect_reg, by = "userId") %>%
      left_join(year_effect_reg, by = "year") %>%
      left_join(genre_effect_reg, by = "genres") %>%
      left_join(daterating_effect_reg, by = "yr_rate") %>%
      mutate(pred = mu_hat + b_i + b_u + b_y + b_g + b_dr) %>%
      pull(pred)
  return(RMSE(predicted_ratings_model_10, test$rating))
})
#Plot lambda vs value and select optimum
qplot(lambdas, rmses)
```




```
lambdas[which.min(rmses)]
```

```
## [1] 4.8
```

```
min(rmses)
```

```
## [1] 0.8634867
```

```
##Final calculation
l=lambdas[which.min(rmses)]
##Calculate Movie effect
movie_effect_reg <- train %>%
  group_by(movieId)%>%
  summarize(b_i = sum(rating - mu_hat)/(n()+l))
##Calculate user effect
user_effect_reg <- train %>%
```

```
        left_join(movie_effect_reg, by="movieId") %>%
        group_by(userId)%>%
        summarize(b_u = sum(rating - b_i- mu_hat)/(n()+l))
      #calculate year effect
      year_effect_reg<-train%>%
        left_join(movie_effect_reg, by='movieId') %>%
        left_join(user_effect_reg, by='userId') %>%
        group_by(year)%>%
        summarize(b_y=sum(rating-mu_hat-b_i-b_u)/(n()+l))
      ##calculate genre effect
      genre_effect_reg<-train%>%
        left_join(movie_effect_reg, by='movieId') %>%
        left_join(user_effect_reg, by='userId') %>%
        left_join(year_effect_reg, by='year') %>%
        group_by(genres)%>%
        summarize(b_g=sum(rating-mu_hat-b_i-b_u-b_y)/(n()+l))
      ##Calculate date of rating effect
        daterating_effect_reg<-train%>%
        left_join(movie_effect_reg, by='movieId') %>%
        left_join(user_effect_reg, by='userId') %>%
        left_join(year_effect_reg, by='year') %>%
        left_join(genre_effect_reg, by = "genres") %>%
        group_by(yr_rate)%>%
        summarize(b_dr=sum(rating-mu_hat-b_i-b_u-b_y-b_g)/(n()+l))
      ##calculate predictions
      predicted_ratings_model_10 <- test %>%
        left_join(movie_effect_reg, by = "movieId") %>%
        left_join(user_effect_reg, by = "userId") %>%
        left_join(year_effect_reg, by = "year") %>%
        left_join(genre_effect_reg, by = "genres") %>%
        left_join(daterating_effect_reg, by='yr_rate') %>%
        mutate(pred = mu_hat + b_i + b_u + b_y + b_g +b_dr) %>%
        pull(pred)
      ##Calculate RMSE and print values
      RMSE_model10<-RMSE(predicted_ratings_model_10, test$rating)
      RMSE_10<-
      data.frame(
        Method= "mean + b_i + b_u + b_y + b_g + b_dr  REGULARIZED",
        RMSE= RMSE_model10)
RMSE_results<-bind_rows(RMSE_1,RMSE_2,RMSE_6,RMSE_3,RMSE_7,
                        RMSE_4,RMSE_8,RMSE_5,RMSE_9,RMSE_10)

knitr::kable(RMSE_results)
```

Method	RMSE
Just mean	1.0600537
mean + b_i	0.9429615
mean + b_i REGULARIZED	0.9429369
mean + b_i + b_u	0.8646843
mean + b_i + b_u REGULARIZED	0.8641362
mean + b_i + b_u + b_y	0.8643301
mean + b_i + b_u + b_y REGULARIZED	0.8638376
mean + b_i + b_u + b_y + b_g	0.8640801
mean + b_i + b_u + b_y + b_dr REGULARIZED	0.8636099
mean + b_i + b_u + b_y + b_g + b_dr REGULARIZED	0.8634867

MODEL 11 Matrix factorization

```
##Set seed
set.seed(1974, sample.kind = "Rounding")
##Adjust formats of training and test sets for the model
train_data <-  with(train, data_memory(user_index = userId,
                                       item_index = movieId,
                                       rating     = rating))
test_data  <-  with(test,  data_memory(user_index = userId,
                                        item_index = movieId,
                                         rating     = rating))
# Create the model object
r <-  Reco()

# Train the algorithm  (with default parameters)
r$train(train_data)
```

```
## iter      tr_rmse          obj
##    0       0.9633   1.3302e+07
##    1       0.8817   1.2022e+07
##    2       0.8592   1.1812e+07
##    3       0.8456   1.1661e+07
##    4       0.8392   1.1603e+07
##    5       0.8343   1.1557e+07
##    6       0.8305   1.1525e+07
##    7       0.8280   1.1505e+07
##    8       0.8262   1.1492e+07
##    9       0.8248   1.1479e+07
##   10       0.8238   1.1466e+07
##   11       0.8229   1.1461e+07
##   12       0.8223   1.1457e+07
##   13       0.8218   1.1453e+07
##   14       0.8212   1.1445e+07
##   15       0.8209   1.1441e+07
##   16       0.8206   1.1441e+07
##   17       0.8203   1.1438e+07
##   18       0.8200   1.1433e+07
##   19       0.8197   1.1434e+07
```

```
##Calculate predicted data
predicted_ratings_model_11 <-  r$predict(test_data, out_memory())
##calculate RMSES for final value
RMSE_model11<-RMSE(test$rating, predicted_ratings_model_11)
RMSE_11<-data.frame(Method= "Matrix Factorization", RMSE= RMSE_model11)
RMSE_results<-bind_rows(RMSE_1,RMSE_2,RMSE_6,RMSE_3,RMSE_7,RMSE_4,
                        RMSE_8,RMSE_5,RMSE_9,RMSE_10,RMSE_11)
##Print all results to select best model
knitr::kable(RMSE_results)
```

Method	RMSE
Just mean	1.0600537
mean + b_i	0.9429615
mean + b_i REGULARIZED	0.9429369
mean + b_i + b_u	0.8646843
mean + b_i + b_u REGULARIZED	0.8641362

Method	RMSE
mean + b_i + b_u + b_y	0.8643301
mean + b_i + b_u + b_y REGULARIZED	0.8638376
mean + b_i + b_u + b_y + b_g	0.8640801
mean + b_i + b_u + b_y + b_dr REGULARIZED	0.8636099
mean + b_i + b_u + b_y + b_g + b_dr REGULARIZED	0.8634867
Matrix Factorization	0.8321857

So finally te model with lowest RMSE is:

	Method	RMSE
11	Matrix Factorization	0.8321857

Evaluating validation set:

```
###Model 11 MATRIX FACTORIZATION
set.seed(1974, sample.kind = "Rounding")
r$train(train_data)
#Convert validation dataset in matrix format
valid  <-  with(validation,  data_memory(user_index = userId,
                                        item_index = movieId,
                                         rating     = rating))
##Predict values
predicted_ratings_model_11v <-  r$predict(valid, out_memory())
##Calculate RMSES for validated models
RMSE_model11v<-RMSE(validation$rating, predicted_ratings_model_11v)
RMSE_11v<-data.frame(Method= "Matrix Factorization", RMSE= RMSE_model11v)
```

```
#Print result of MATRIX FACTORIZATION in validation set to check if RMSES<0.86490
RMSE_11v%>%kable()
```

Method	RMSE
Matrix Factorization	0.8328876

4-Conclusion

The model selected in last section is model 11 (that is the one that should be accomplishing RMSE requirement), but we are going to evaluate in all models in validation set to see impact of each model in validation set

```
knitr::kable(RMSE_resultsv)
```

Method	RMSE
validation set in mean	1.0612018
validation set in mean + movie effect	0.9439729
validation set in mean + movie effect REGULARIZED	0.9439108
validation set in mean + movie effect + user_effect	0.8658528

Method	RMSE
validation set in mean + movie effect + user_effect REGULARIZED	0.8652208
validation set in mean + movie effect + user_effect + aging_effect	0.8655043
validation set in mean + movie effect + user_effect + aging_effect REGULARIZED	0.8649275
validation set in mean + movie effect + user_effect + aging_effect + genre_effect	0.8652140
validation set in mean + movie effect + user_effect + aging_effect + genre_effect REGULARIZED	0.8646588
validation set in mean + movie effect + user_effect + aging_effect + genre_effect + date rating REGULARIZED	0.8645199
Matrix Factorization	0.8328876

For this project we were requested to find and RMSE of < 0.86490 on the validation test. Last 3 models accomplished this.

-validation set in mean + movie effect + user_effect + aging_effect + genre_effect REGULARIZED

-validation set in mean + movie effect + user_effect + aging_effect + genre_effect + date rating REGULARIZED

-Matrix Factorization

By estimating ratings with **Matrix Factorization** i was able to get 0.8328876 and if we compare this effect with the result we obtain by estimating the validation test with “just the mean”

```
paste0(round((RMSE_resultsv[1,2]-RMSE_resultsv[11,2])/
              RMSE_resultsv[1,2]*100,2),"%")
```

```
## [1] "21.51%"
```

we see an 21.51% reduction of the RMSE in the test validation

If we analyze the first 11 methods

```
reduction<-(RMSE_resultsv[1,2]-RMSE_resultsv[10,2])
RMSE_resultsv[1:10,]%>%
  arrange(desc(RMSE))%>%
  mutate(Impact =  paste0(round((RMSE - lag(RMSE, default = first(RMSE)))/
          -reduction*100, 2),"%"))%>%knitr::kable()
```

Method	RMSE	Impact
validation set in mean	1.0612018	0%
validation set in mean + movie effect	0.9439729	59.6%
validation set in mean + movie effect REGULARIZED	0.9439108	0.03%
validation set in mean + movie effect + user_effect	0.8658528	39.69%
validation set in mean + movie effect + user_effect + aging_effect	0.8655043	0.18%
validation set in mean + movie effect + user_effect REGULARIZED	0.8652208	0.14%
validation set in mean + movie effect + user_effect + aging_effect + genre_effect	0.8652140	0%
validation set in mean + movie effect + user_effect + aging_effect REGULARIZED	0.8649275	0.15%

Method	RMSE	Impact
validation set in mean + movie effect + user_effect + aging_effect + genre_effect REGULARIZED	0.8646588	0.14%
validation set in mean + movie effect + user_effect + aging_effect + genre_effect + date rating REGULARIZED	0.8645199	0.07%

We can observe that Movie effect is 59.6% of the reduction and User effect 39.69% of reduction, so 99.29% of the reduction is because of that effects. It seems that are the two most important parameters to estimate the rating, in the future it would be a good idea to get more information about that items. Like adding more information about movies like actors/actress, company (fox, miramax…), director, duration, awards or understanding better the user profile age, gender, nationality etc.

Matrix Factorization is capable to get better results just by using movieID, userId and rating, it seems that this methodology gets better estimations with same using same information as models 3 and 7.

Another conclusion i can get is that all methods improved when using regularization, showing how small groups can distort estimations if we don't weight them.

there were methods that i couldn't try due to hardware limitations and i had problems tuning parameters in matrix factorization to get a lower RMSE. There is another package for matrix factorization "recommenderlab" that can be used.

5-References

https://rafalab.github.io/dsbook/

https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

https://www.rdocumentation.org/packages/recosystem/versions/0.3

"https://www.fastcompany.com/1671893/the-secret-sauce-behind-netflixs-hit-house-of-cards-big-data

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

https://www.fastcompany.com/1671893/the-secret-sauce-behind-netflixs-hit-house-of-cards-big-data