HarvardX Data Science Professional Certificate program:capstone project

Prasad Padmanabhan

2023-11-17

1. Introduction/Overview

This project is the ninth and final course in HarvardX's multi-part Data Science Professional Certificate series. Recommendation systems rely on user-provided ratings for items to generate personalized suggestions. Due to the vastness of online information, customers often struggle to conclude their search and find contentment with a product. This can be a time-consuming and stressful endeavor for them. The inability to browse through all products available from an online business poses a challenge, as it risks losing customers. However, large companies equipped with extensive data can mitigate this issue by employing sophisticated algorithms to predict, with a high degree of certainty, the products that will attract customers' interest. Streaming

Companies such as Netflix stands as an example of such an online business, boasting a vast array of movies and a substantial customer base use their extensive data set to recommend encourage customers to rate them. These systems examine various types of data, including user ratings, reviews, and viewing histories, in order to create tailored recommendations for individuals. Netflix employs a recommendation system to anticipate the number of stars a user might assign to a particular movie. A single star implies a poor movie, while five stars indicate an outstanding one.

Netflix data is not publically available, but GroupLens Research lab (Social Computing Research at the University of Minnesota) has gathered and provided rating datasets sourced from the MovieLens website.The data consists of user information, movie titles, their user rating, various genres and the date of review (timestamp) information. The rating of a movie reflects the users interest in that movie; for example, the movie get low rating if the user did not like that kind of movie whereas high rating for the liked movie.

2. Methods/Analysis

The primary aim of this project is to develop a movie recommendation algorithm by utilizing the MovieLens dataset. Evaluating the algorithm's efficacy will involve calculating the residual mean squared error (RMSE). The ideal model will showcase the lowest RMSE, adhering to a specified threshold of below 0.86490 for this task. During the analysis phase, I will create models focusing on movie bias, user bias, and a combination of movie and user biases. As part of the process, I plan to incorporate regularization into the model to accommodate highly rated movies with significantly low viewership counts. Model optimization will be carried out using both train and test sets. Upon achieving the targeted RMSE, I will leverage the optimized model to make predictions on the final_holdout_test using the entire edx dataset.

3. Data Preparation: Load the data into R and preprocess it to make it suitable for analysis.

```
############################################################
# Create edx and final_holdout_test sets
############################################################

# Note: this process could take a couple of minutes

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")

library(tidyverse)
library(caret)

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

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
```

```
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

3.1 Split the edx data to train and test sets

The edx data will be split into train (90%) and test sets(10%).

```
set.seed(123)

test_index<-createDataPartition(edx$rating, times=1,p=0.1, list=F)
edx_train<-edx[-test_index,]
edx_temp<-edx[test_index,]
```

```
# Make sure userId(users) and movieId(movies) in edx_test set are also in
#edx_train set


edx_test<- edx_temp %>%
  semi_join(edx_train, by="movieId")%>%
  semi_join(edx_train,by="userId")



# Add rows removed from edx_test set back into edx_train set
removed <- anti_join(edx_temp, edx_test)
edx_train <- bind_rows(edx_train, removed)
```

3.2 Data cleaning

Timestamp and genre variables will not be used for the analysis, therefore, those are removed from the test and train data set. The data cleaning reduces the complexity.A good quality data reduce complexity and processing time to deliver the correct information

```
#remove timestamp and genre variables from both test and train data sets
edx_train <-edx_train %>% select(userId, movieId,title,rating)

edx_test <-edx_test %>% select(userId, movieId,title,rating)
```

4 Exploratory Data Analysis : Analyze the data to gain insights into its characteristics and identify patterns

.

```
#to obtain the structure of the data sets
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  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 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

```
str(final_holdout_test)
```

```
## 'data.frame':    999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : int  231 480 586 151 858 1544 590 4995 34 432 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
```

```
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Roman
```

The **str()** tells that the variables are of the following types

```
library(kableExtra)
variable_types<-tibble(names=names(edx),
                           data_type=c("integer","integer","double",
                                       "integer","character","character"))

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

names	data_type
userId	integer
movieId	integer
rating	double
timestamp	integer
title	character
genres	character

Among the above variables, only rating can be used for mathematical calculations.

Number of unique users and movies of relevant variables

```
library(kableExtra)

# unique users, movies and variables
unique_edx_features<-data.frame(Unique_users=n_distinct(edx$userId),
Unique_movies=n_distinct(edx$movieId),
Unique_ratings=n_distinct(edx$rating))

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

Unique_users	Unique_movies	Unique_ratings
69878	10677	10

The edx and final holdout_test datasets have 9000055 and 999999 observations respectively. Each row corresponds to a user who is rating a single movie. But there are only 69878 unique users and 10677 unique movies. This suggests that a single user rates multiple movies at different time, and similarly, same movie was rated by multiple users

4.1 Explore the rating variable of edx data set

The following code summarizes the mean and standard deviation of rating variable

```
library(kableExtra)
edx_summary_mean_and_sd <- edx %>% summarise(avg=mean(rating,na.rm=T), sds=sd(rating,na.rm=T))
edx_summary_mean_and_sd %>% knitr::kable()
```

avg	sds
3.512465	1.060331

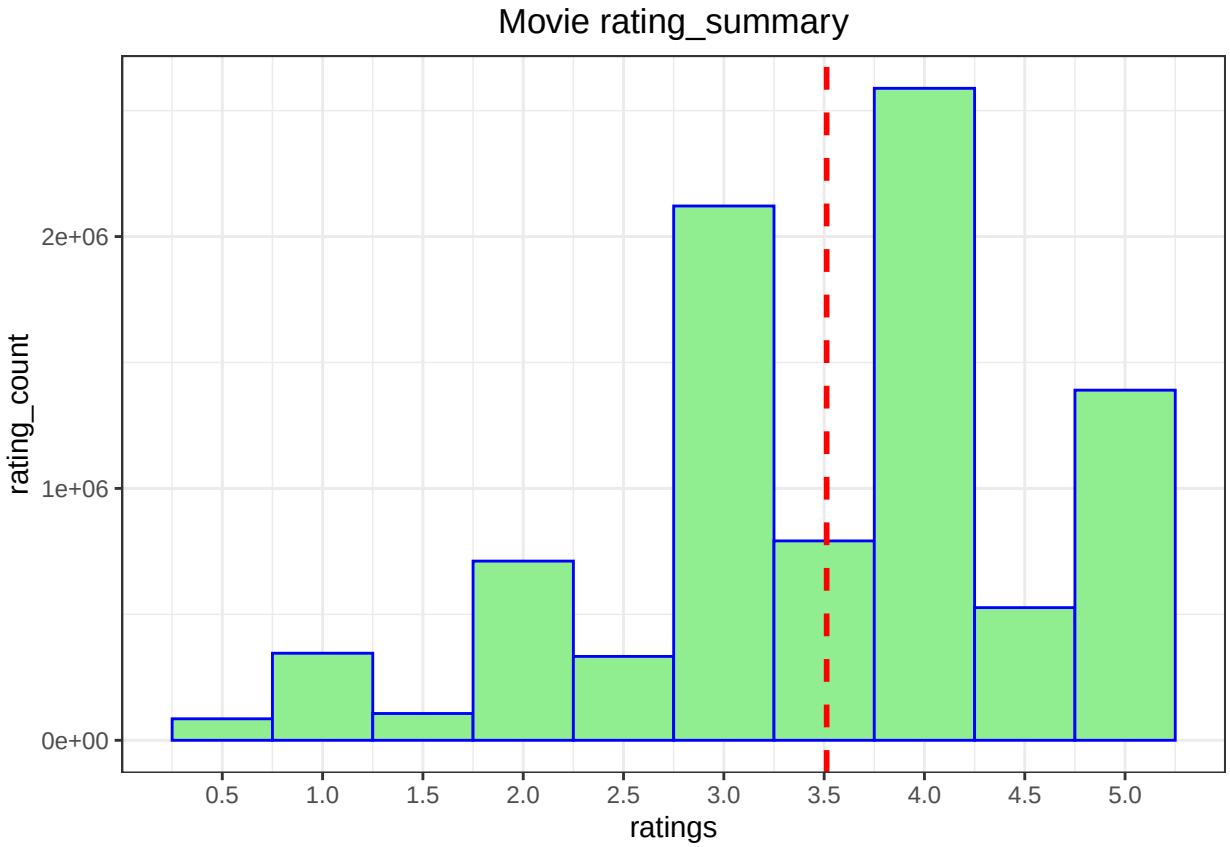
The following two figures shows, there are 10 distinct ratings, ranges from 0.5 to 5 with 0.5 interval. Among these ratings, rating 4 got maximum counts followed by 3 and 5. In general, full ratings(1,2,3,4 and 5) got

more rating than half ratings(0.5,1.5,2.5.3.5.and 4.5). The mean of rating is shown in the plot as vertical red dotted line.

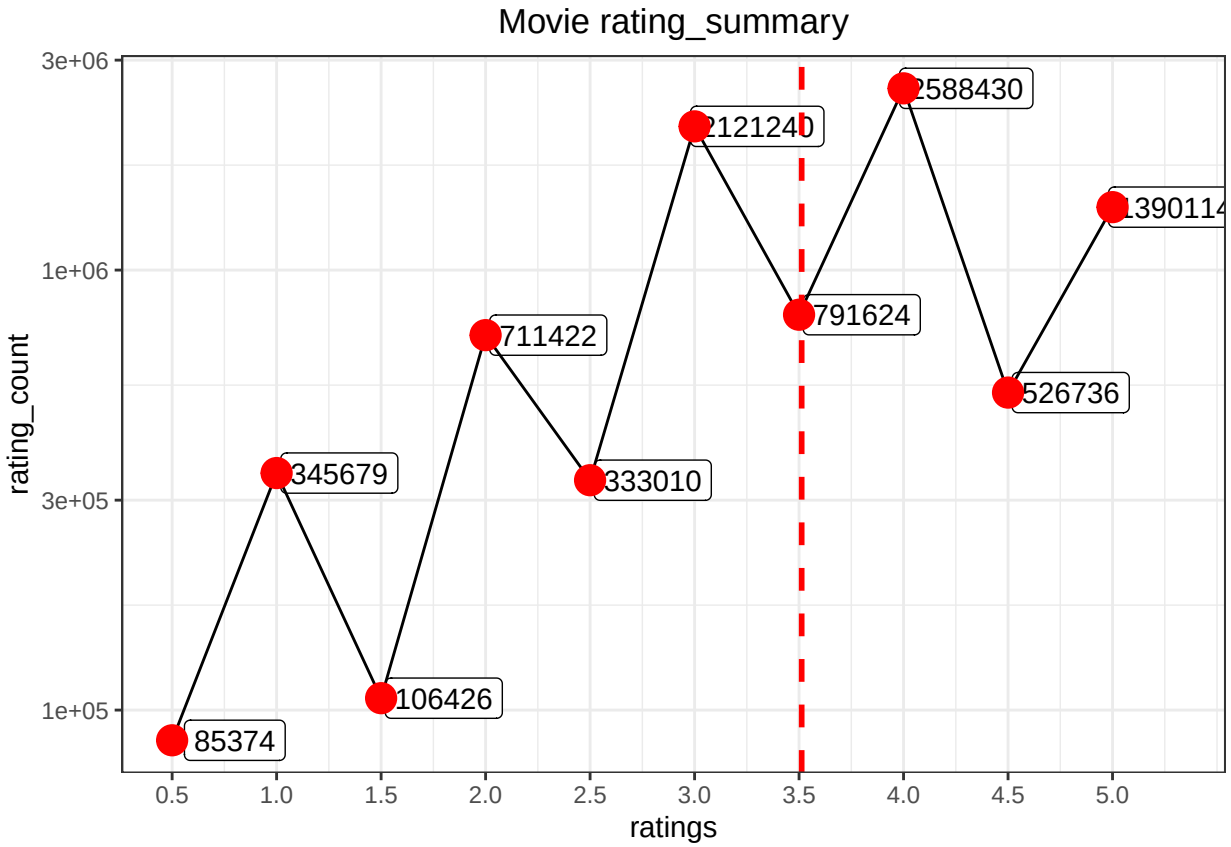
```
library(kableExtra)
rating_summary<-edx %>% group_by(rating)%>% summarise(rating_count=n()) %>% arrange(desc(rating_count))


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

rating	rating_count
4.0	2588430
3.0	2121240
5.0	1390114
3.5	791624
2.0	711422
4.5	526736
1.0	345679
2.5	333010
1.5	106426
0.5	85374

```
#barplot
edx %>% group_by(rating)%>% summarise(rating_count=n()) %>%
  ggplot(aes(rating,rating_count))+
  geom_bar(stat="identity",width =0.5,color="blue",fill="lightgreen")+
  scale_y_continuous()+
  geom_vline(aes(xintercept = mean(edx$rating)), color = "red",linetype=2,lwd=1)+
  scale_x_continuous(breaks=seq(0.5,5,0.5))+
  theme_bw()+
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle =element_text(hjust = 0.5) )+
  ggtitle("Movie rating_summary")+
  xlab("ratings")
```




```
#lineplot
edx %>% group_by(rating)%>% summarise(rating_count=n()) %>%
  ggplot(aes(x=rating,y=rating_count,label=rating_count))+
  geom_label(nudge_x = 0.3)+
  geom_line()+
  scale_y_log10()+
  scale_x_continuous(breaks = seq(0, 5, 0.5))+
  geom_point(size=5,color="red")+
  geom_vline(aes(xintercept = mean(edx$rating)), color = "red",linetype=2,lwd=1)+
  theme_bw()+
  ggtitle("Movie rating_summary")+
  theme(plot.title = element_text(hjust = 0.5), plot.subtitle =element_text(hjust = 0.5) )+
  xlab("ratings")+
  ylab("rating_count")
```



Overall, **>=3** ratings accounts for 82% of overall rating suggests, in general, users gave good ratings

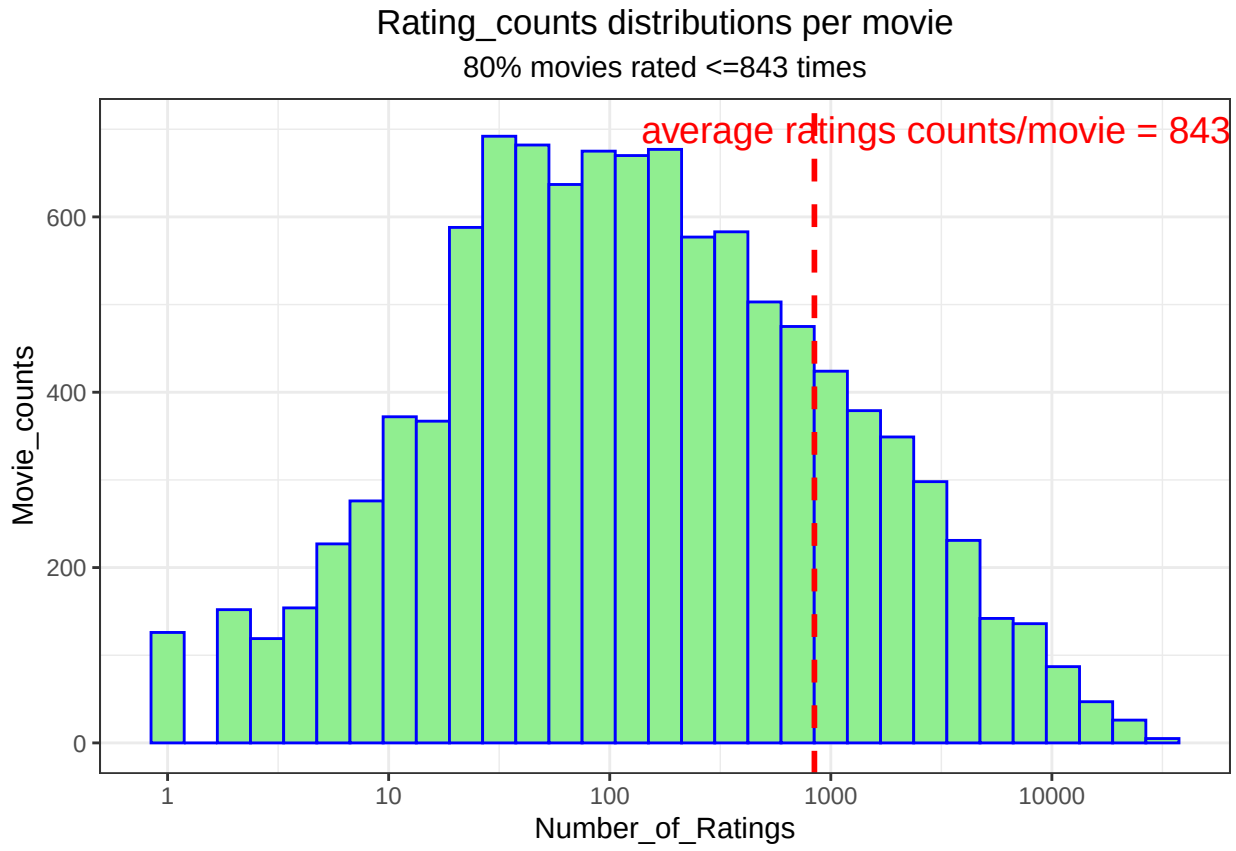
4.2 Explore the movieId variable

```
library(kableExtra)
#The following code provides a summary of each movie's average rating and their rating counts

movie_summary<-edx %>% group_by(title)%>%
  summarise(count=n(),mean_rating=mean(rating)) %>% arrange(desc(count))

##To find the top10 rated movies
movie_summary %>% head(10) %>%knitr::kable()
```

title	count	mean_rating
Pulp Fiction (1994)	31362	4.154789
Forrest Gump (1994)	31079	4.012822
Silence of the Lambs, The (1991)	30382	4.204101
Jurassic Park (1993)	29360	3.663522
Shawshank Redemption, The (1994)	28015	4.455131
Braveheart (1995)	26212	4.081852
Fugitive, The (1993)	25998	4.009155
Terminator 2: Judgment Day (1991)	25984	3.927859
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672	4.221311
Apollo 13 (1995)	24284	3.885789

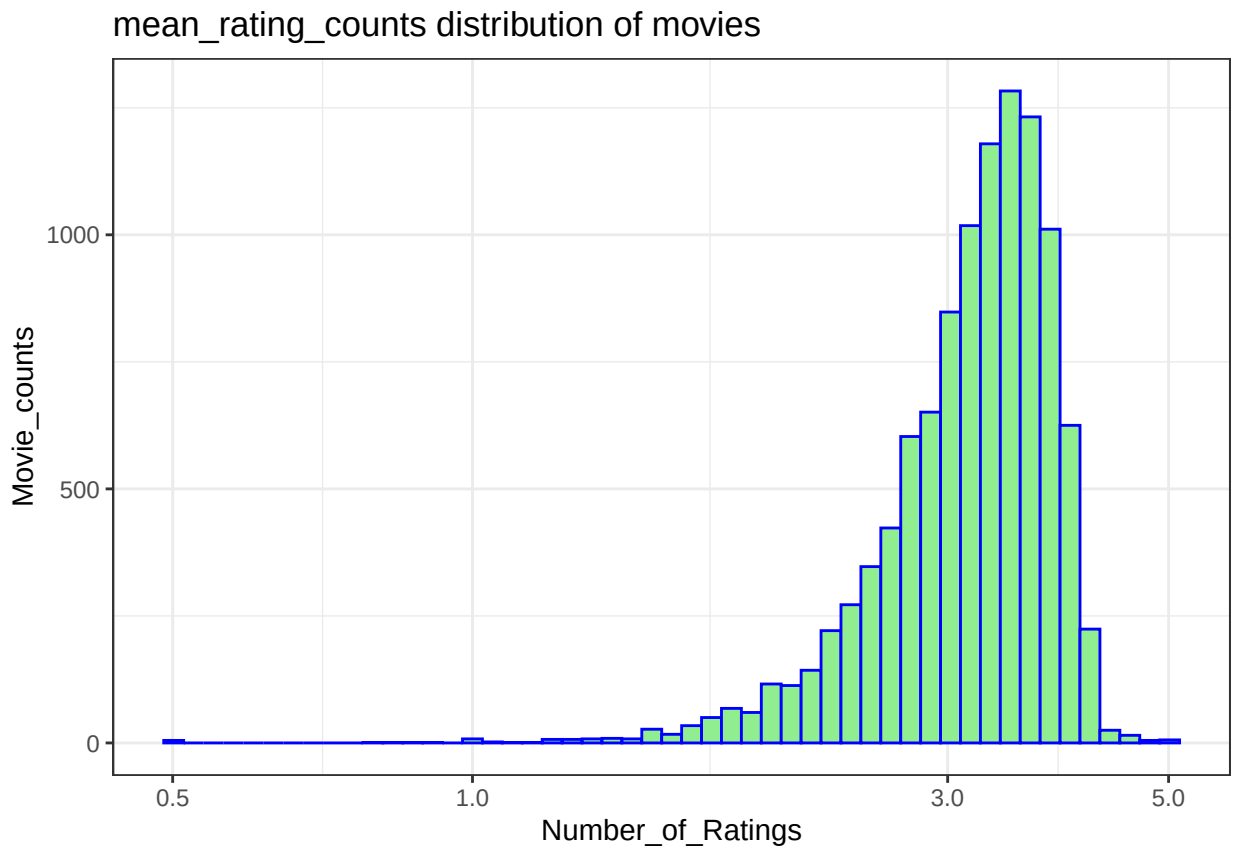
```
#movie counts distribution plot

movie_summary %>% ggplot(aes(count))+
  geom_histogram(color="blue",fill="lightgreen",binwidth = 0.15)+
  scale_x_log10()+
  theme_bw()+
  xlab("Number_of_Ratings")+
  ylab("Movie_counts")+
  ggtitle("Rating_counts distributions per movie",
          subtitle = "80% movies rated <=843 times" )+
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle =element_text(hjust = 0.5) )+
  geom_vline(aes(xintercept = mean(movie_summary$count)), color = "red",linetype=2,lwd=1)+
  annotate("text",y=700,x=3000,
           label = print(paste("average ratings counts/movie =",round(mean(movie_summary$count),0))),
           color = "red", size = 5)
```

```
## [1] "average ratings counts/movie = 843"
```




```
#movie_counts vs mean_rating of each movies: distribution plot
movie_summary %>% ggplot(aes(mean_rating))+
  geom_histogram(color="blue",fill="lightgreen",binwidth = 0.02)+
  scale_x_log10()+
  theme_bw()+
  xlab("Number_of_Ratings")+
```

```
  ylab("Movie_counts")+
  ggtitle("mean_rating_counts distribution of movies")
```



The mean_rating plot is left skewed. This shows that the most movies received good ratings by users.

4.3 Explore the userId variable

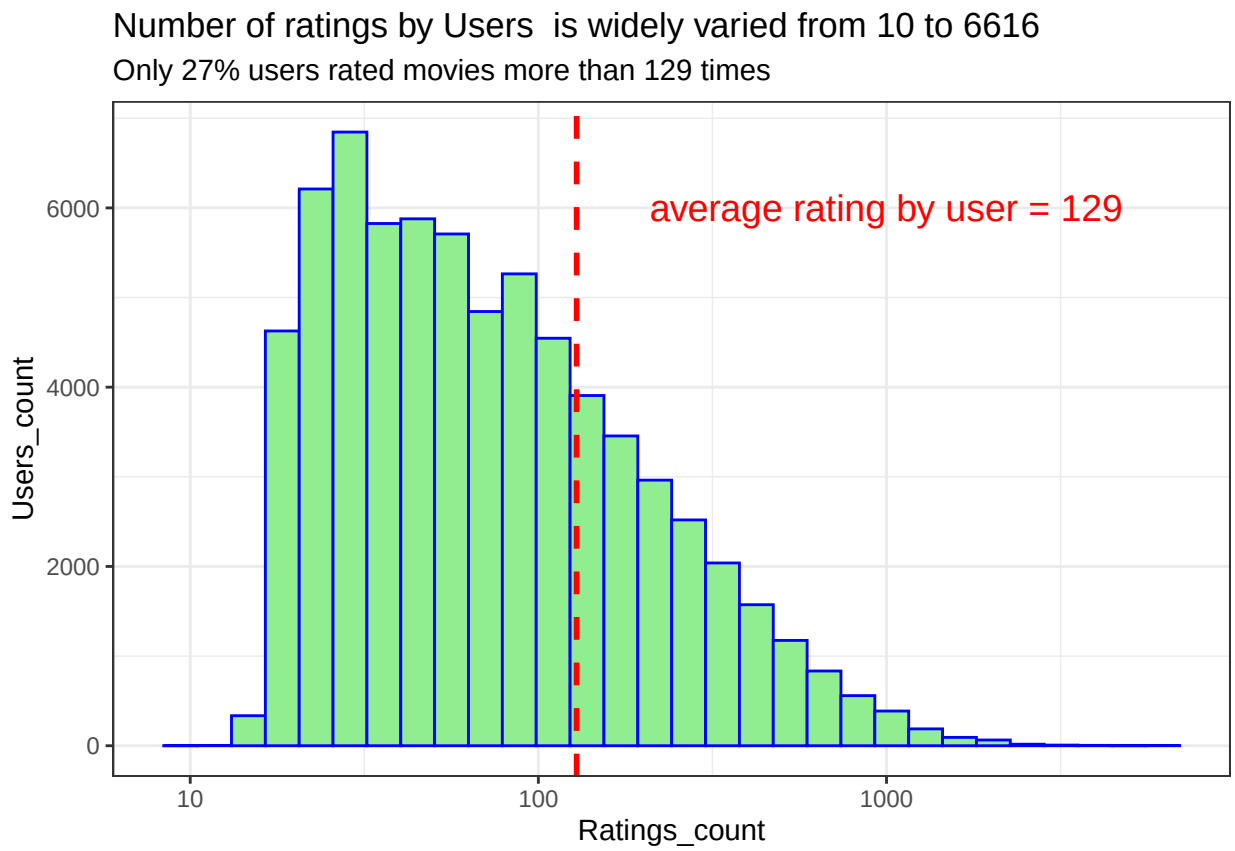
```
#userId summary
user_id_summary<-edx %>% group_by(userId) %>%
  summarise(mean_rating= mean(rating), count=n())


#An average user rates 129 times
round(mean(user_id_summary$count),0)
```

```
## [1] 129
```

```
#visualization: Distribution of users are right skewed
user_id_summary %>% ggplot(aes(count))+
  geom_histogram(fill="lightgreen",color="blue",bins = 30, color="black")+
  scale_x_log10()+
  theme_bw()+
```

```
  xlab("Ratings_count")+
  ylab("Users_count")+
  geom_vline(aes(xintercept = mean(user_id_summary$count)), color = "red",linetype=2,lwd=1)+
  annotate("text", x = 1000,y=6000,
           label = print(paste("average rating by user =",round(mean(user_id_summary$count),0))),
           color = "red", size = 5,)+
           ggtitle("Number of ratings by Users  is widely varied from 10 to 6616",
           subtitle = paste("Only 27% users rated movies more than",
                            round(mean(user_id_summary$count),0), "times"))
```

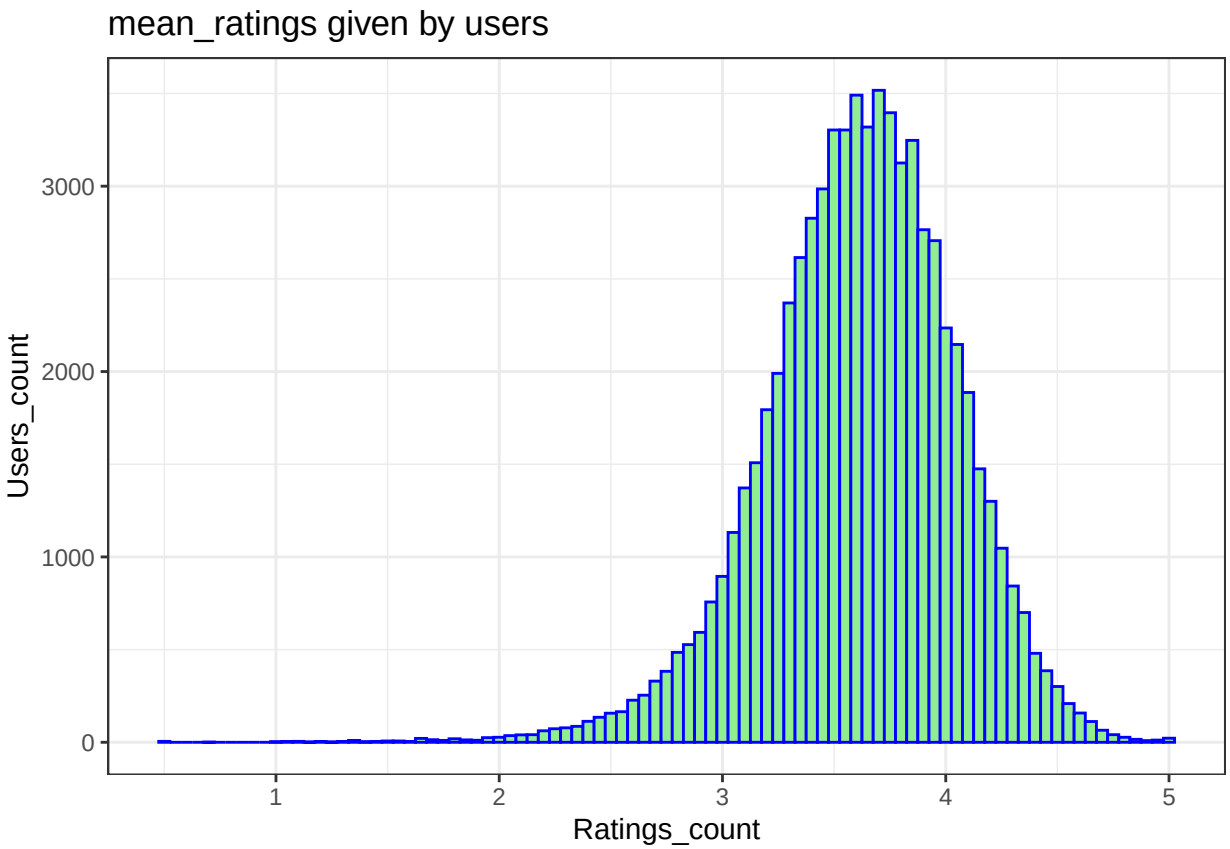
```
## [1] "average rating by user = 129"
```



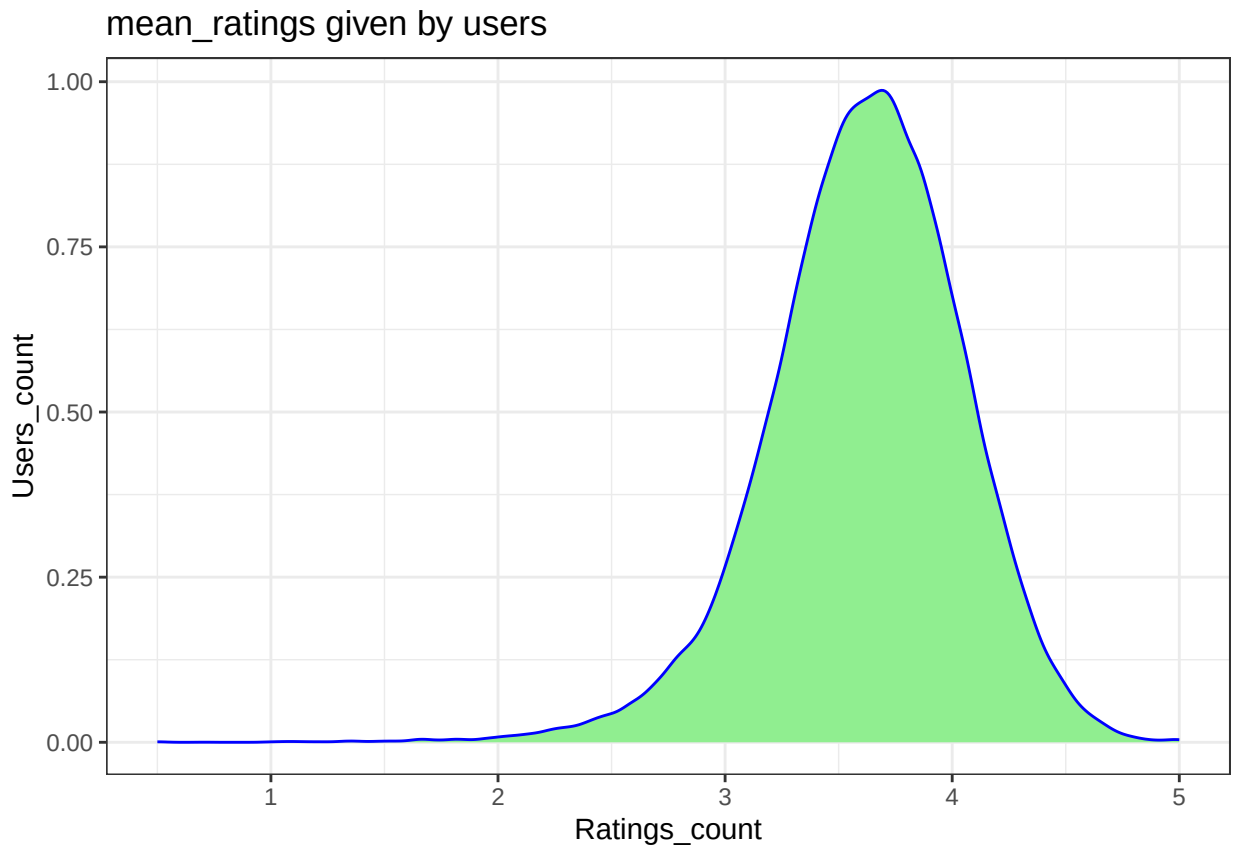

```
#the above plot suggests mostly users rated movies very few times
```

The above plot is right skewed, This suggests that most of the movies got few numbers of ratings.The dashed red vertical line corresponds to mean user rating counts.

```
#histogram
user_id_summary %>% ggplot(aes(mean_rating))+
  geom_histogram(fill="lightgreen",color="blue",binwidth = 0.05)+
  scale_x_continuous()+
  theme_bw()+
  xlab("Ratings_count")+
  ylab("Users_count")+
  ggtitle("mean_ratings given by users")
```




```
#density plot
user_id_summary %>% ggplot(aes(mean_rating))+
  geom_density(fill="lightgreen",color="blue")+
  scale_x_continuous()+
  theme_bw()+
  xlab("Ratings_count")+
  ylab("Users_count")+
  ggtitle("mean_ratings given by users")
```



The above figures suggests, mean_ratings given by user is normally distributed.

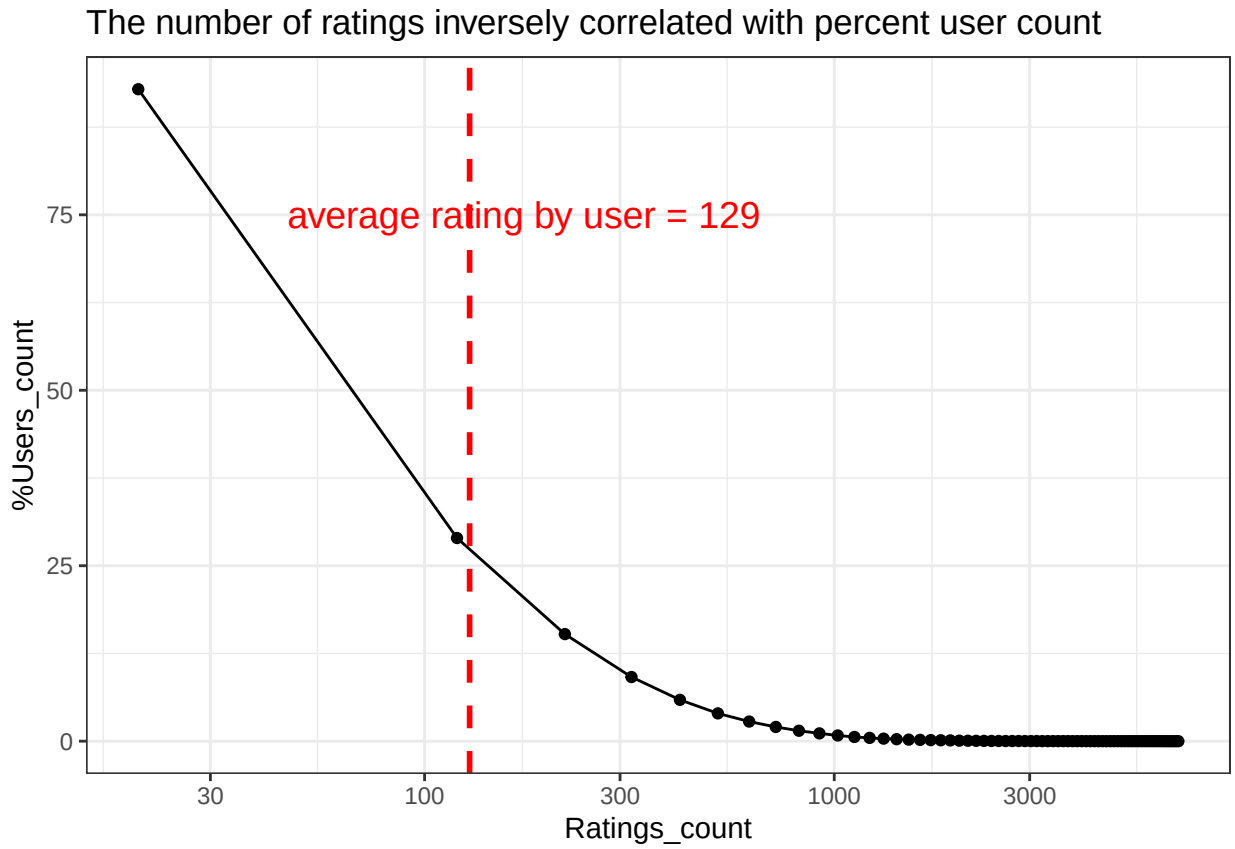
The rating_counts by users and number of users are inversely correlated. As the number of ratings increases, the user_count decreases drastically. **In other words mostly users rated few movies**.

Calculate the percentage number of users upon a given cut off value of rating counts

```
#Create an empty data frame
df<-data.frame()

#create a vector  "x" with seq(). The vector x corresponds to different values of rating counts.
#Using these values as cut offs,calculate the total number of users rated movies
x<-seq(20,7000,100)


# Generate a data frame which has three columns: cut_off, num and percent
for (i in 1:length(x)){
  cut_off<-x[i]
  num<-user_id_summary %>% filter(count>x[i])%>% count()%>% pull(n)
  percent<-user_id_summary %>% filter(count>x[i])%>% count()%>% pull(n)/69878*100
  df<-as.data.frame(rbind(df,data.frame(cut_off,num,percent)))
}

df %>% ggplot(aes(cut_off,percent))+
  geom_point()+
  geom_line()+
  theme_bw()+
  scale_x_log10()+
```

```
xlab("Ratings_count")+
  ylab("%Users_count")+
  geom_vline(aes(xintercept = mean(user_id_summary$count)), color = "red",
             linetype=2,lwd=1)+
  annotate("text", x = 175,y=75,
           label = print(paste("average rating by user =",
                               round(mean(user_id_summary$count),0))),
           color = "red", size = 5)+
  ggtitle(paste("The number of ratings inversely correlated with percent user count "))
```

```
## [1] "average rating by user = 129"
```




```
#Number of users rated more than 1000 times
above_1000<-user_id_summary %>% filter(count>=1000)%>%nrow()
#Number of users rated less than 1000 times
below_1000<-user_id_summary %>% filter(count<1000)%>%nrow()
```

The cutoff values inversely correlated with the users_count. Only 611 users rated more than 1000 times whereas there were 69267 users below the 1000 cutoff.

5 Strategy for model optimization

Here I am trying to optimize the model using edx_train and edx_test data set. The project objective is to get a RMSE value less than 0.86490.

(Step1) Use the edx_train set to predict the efficacy of model on edx_test set. After the model is optimised with edx_train set, go to the next step

(step2) use the entire edx data set to predict the final_holdout_test set.

5.1 Results

5.1.1 Required RMSE

```
library(kableExtra)
required_RMSE<- 0.86490

#make table which store the RMSE values and dataset used for optimizing the RMSE values
models_df<-tibble(method="Required_RMSE", RMSE= format(required_RMSE,digits=7), dataset=NA)




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

method	RMSE	dataset
Required_RMSE	0.8649	NA

5.1.2 First model:basic model

In the basic model, We predict the same rating for all movies and users without accounting for the user effects and movie effects.All the differences are explained by random variations. Therefore, we can denote the equation as follows.

$Y_{u,i} = \mu + \epsilon_{u,i}$. $\epsilon_{u,i}$ is the random variation of movie i and user u. $\epsilon_{u,i}$ is the independent errors sampled from the same distribution centered at 0 and μ is the "true" rating for all movies and $Y_{u,i}$ is our prediction.

```
library(kableExtra)
#Average edx_train$rating
mu_edx_train<-mean(edx_train$rating)
mu_edx_train
```

```
## [1] 3.51246
```

```
#using the mean of edx_train set rating predict the edx_test ratings

basic_RMSE<-format(RMSE(edx_test$rating, mu_edx_train), digits=7)



#Created a result table as follows. The models_df will store all the output RMSE

models_df<- bind_rows(models_df,tibble(method="Basic_model",
                                       RMSE= format(basic_RMSE,digits=7),
```

```
                                    dataset="edx_train vs edx_test"))



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

method	RMSE	dataset
Required_RMSE	0.8649	NA
Basic_model	1.06062	edx_train vs edx_test

Using average of **edx$rating** give us a RMSE value of 1.06062. We need to do better than this

5.1.3 user effect on movie ratings

Users exhibit varying rating behaviors, where some are highly active and tend to give substantially higher ratings, while others, who are less active, rate the same movie differently. Consequently, it's imperative to consider the user bias (b_u) when assessing movie ratings. To calculate the RMSE based solely on the user effect, we'll enhance the basic equation by incorporating adjustments for the user effect. This revised equation will aim to improve the user effect within the model, leading to a new formula.

$Y_{u,i} = \mu + b_u + \epsilon_{u,i}$. Therefore the new formula will looks like this $Y_u = \mu + b_u$. We can calculate b_u as $b_u = Y_u - mu$

```
library(kableExtra)

#Y_hat<- mu+b_u

#Create a new data frame with userId_summary

b_u_edx_train<- edx_train %>%  mutate(b_u= rating-mu_edx_train)%>%
  group_by(userId)%>%
  summarise(b_u=mean(b_u))

#join edx_test(test set) to userId summary to generate a table with b_u column
user_effect<- edx_test %>% left_join(b_u_edx_train, by="userId")%>%
  mutate(prediction= mu_edx_train+b_u)

#calculate RMSE
user_effect_alone<-format(RMSE(edx_test$rating,user_effect$prediction),digtis=7)


print(paste("RMSE is decreased to" , user_effect_alone,  "by user bias alone"))
```

```
## [1] "RMSE is decreased to 0.9782858 by user bias alone"
```

```
models_df <- bind_rows(models_df, tibble(method="User effect alone", RMSE=user_effect_alone, dataset="e

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

method	RMSE	dataset
Required_RMSE	0.8649	NA
Basic_model	1.06062	edx_train vs edx_test
User effect alone	0.9782858	edx_train vs edx_test

We can see that by adding user bias, we improved the RMSE.

5.1.4 movie effect on ratings

We can observe from the data that certain films have greater ratings than others. This is because different movies have varied ratings; popular movies receive high ratings and are rated in huge numbers, while low ratings are assigned to certain films. Some critical choice films receive lower ratings since they are seen by a smaller number of people than mass popular films, which have very high ratings. As a result, the movie effect, or movie bias, must be taken into consideration when calculating RMSE.

To answer the movie bias(b_i) we can use the following formula

$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$. We can calculate the movie bias b_i as $b_i = Y_{u,i} - \mu - \epsilon_{u,i}$

```
library(kableExtra)
#movie_effect
b_i_edx_train<-edx_train %>% mutate(b_i=rating - mu_edx_train) %>%
  group_by(movieId)%>%
  summarise(b_i=mean(b_i))


#The following code generate a new data frame with prediction column, which
#account for the movie bias
movie_effect<-edx_test %>%
  left_join(b_i_edx_train, by="movieId")%>%
  mutate(prediction = mu_edx_train + b_i)

#calculate RMSE
movie_bias_effect<-format(RMSE(edx_test$rating,movie_effect$prediction),7)




#add RMSE after movie bias correction to the table
models_df<-bind_rows(models_df, tibble(method="Movie_bias_alone",
                                       RMSE=movie_bias_effect,
                                       dataset="edx_train vs edx_test"))


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

method	RMSE	dataset
Required_RMSE	0.8649	NA
Basic_model	1.06062	edx_train vs edx_test
User effect alone	0.9782858	edx_train vs edx_test
Movie_bias_alone	0.9432105	edx_train vs edx_test

```
cat("RMSE is decreased to" , movie_bias_effect,  "by movie bias alone.
Interestingly accounting for movie bias reduced the RMSE \nmore than the user bias")
```

```
## RMSE is decreased to 0.9432105 by movie bias alone.
```

```
## Interestingly accounting for movie bias reduced the RMSE
## more than the user bias
```

##However, further evaluation of movie bias model(below) showed, the outputs are not reliable.The best movies as per the movie bias model are all unknown movies.

```
#create a data frame with movieIds and movie titles

movies <- edx_train %>%
  select(movieId,title) %>%
  distinct()

head(movies,10)
```

```
##    movieId                                    title
## 2      185                         Net, The (1995)
## 4      292                         Outbreak (1995)
## 5      316                         Stargate (1994)
## 6      329            Star Trek: Generations (1994)
## 7      355                  Flintstones, The (1994)
## 8      356                      Forrest Gump (1994)
## 9      362                  Jungle Book, The (1994)
## 10     364                    Lion King, The (1994)
## 11     370 Naked Gun 33 1/3: The Final Insult (1994)
## 12     377                           Speed (1994)
```

```
# 10 best movies (ranked by bi).

b_i_edx_train %>%
  inner_join(movies, by = "movieId") %>%
  arrange(-b_i) %>%
  select(title) %>%
  head(10)
```

```
## # A tibble: 10 x 1
##    title
##    <chr>
##  1 Hellhounds on My Trail (1999)
##  2 Satan's Tango (Sátántangó) (1994)
##  3 Shadows of Forgotten Ancestors (1964)
##  4 Fighting Elegy (Kenka erejii) (1966)
##  5 Sun Alley (Sonnenallee) (1999)
##  6 Blue Light, The (Das Blaue Licht) (1932)
##  7 More (1998)
##  8 Human Condition II, The (Ningen no joken II) (1959)
##  9 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1~
## 10 Human Condition III, The (Ningen no joken III) (1961)
```

```
##Top 10 worst movies

b_i_edx_train %>%
  inner_join(movies, by = "movieId") %>%
```

```
  arrange(b_i) %>%
  select(title) %>%
  head(10)
```

```
## # A tibble: 10 x 1
##    title
##    <chr>
##  1 Besotted (2001)
##  2 Hi-Line, The (1999)
##  3 Accused (Anklaget) (2005)
##  4 Confessions of a Superhero (2007)
##  5 War of the Worlds 2: The Next Wave (2008)
##  6 SuperBabies: Baby Geniuses 2 (2004)
##  7 Hip Hop Witch, Da (2000)
##  8 Disaster Movie (2008)
##  9 From Justin to Kelly (2003)
## 10 Criminals (1996)
```

```
#both the above groups were unknown movies
```

Indeed, adding movie bias alone also improve the RMSE around 11% from 1.06 (basic model) to 0.944.

What outcome might result from concurrently incorporating both movie bias and user bias to enhance the RMSE?

5.1.5 Movie and users effect on ratings

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

```
library(kableExtra)
#movies and user bias(b_i and b_u)

#Y_hat= mu+b_i+ b_u

#to get the b_u(user effect bias column in the data frame)

user_bias_df<-edx_train %>%
  left_join(b_i_edx_train, by="movieId")%>%
  group_by(userId)%>%
  summarise(b_u=mean(rating - mu_edx_train - b_i))


#to get the data frame with prediction column
movie_user_bias_pred<- edx_test %>%
  left_join(b_i_edx_train, by="movieId")%>%
  left_join(user_bias_df, by="userId")%>%
  mutate(prediction=mu_edx_train+b_i+b_u)


#calculate RMSE
movie_user_bias_effect<-format(RMSE(edx_test$rating,movie_user_bias_pred$prediction),digits=7)
```

```
print(paste("Both movie and user biases together further reduced the RMSE to", movie_user_bias_effect))
```

```
## [1] "Both movie and user biases together further reduced the RMSE to 0.8650391"
```

```
#add the movie_user_effect by edx_test data to the models_df table.
models_df<-bind_rows(models_df, tibble(method="Movie_User bias effect",
                                       RMSE=movie_user_bias_effect,
                                       dataset="edx_train vs edx_test"))
```

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

method	RMSE	dataset
Required_RMSE	0.8649	NA
Basic_model	1.06062	edx_train vs edx_test
User effect alone	0.9782858	edx_train vs edx_test
Movie_bias_alone	0.9432105	edx_train vs edx_test
Movie_User bias effect	0.8650391	edx_train vs edx_test

The combined movie and user effects improved the RMSE from 1.06 to 0.865(around 19%).

5.1.6 Movie and users effect with Regularization

In machine learning, regularization serves as a technique to prevent overfitting and enhance the overall performance of models. Overfitting occurs when a model excessively learns from the training data, capturing irrelevant patterns or noise that compromises its accuracy on new, unseen data. Regularization introduces additional constraints or information during training to control the model's complexity, aiming to penalize overly intricate models. This approach reduces the model's dependence on intricate patterns within the training data that might not be applicable for predicting new data points.

Movies with high ratings but a remarkably low viewership count create uncertainty due to inadequate sample sizes. Similarly, certain users consistently rate movies from moderately high to very high, yet their total ratings amount to merely 10 instances. Consequently, these allegedly excellent movies lack substantial user feedback, resulting in uncertain evaluations. This situation increases the likelihood of larger estimates of 'b_i', whether negative or positive.

As a result, these estimations are unreliable, especially in predictive contexts, and should not be heavily relied upon. Elevated errors could contribute to an increase in our RMSE, prompting a preference for a cautious approach when encountering uncertainty. Regularization allows us to penalize substantial estimates derived from limited sample sizes.

In fact, guessing that movies with few viewers are just average movies and add a penalty (lambda) in the least square equation (rather than ignoring those movies) will be the better option for the prediction.

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

According to the above equation, n_i is the number of ratings made for movie i. When the number of ratings are large, n_i is very large. a case which will give us a stable estimate, then the penalty

λ is effectively ignored since $n_i + \lambda$ is effectively n_i. However, when the n_i is small, then the estimate $b_i(\lambda)$ is shrunken towards 0. The larger λ, the more we shrink. To select λ we can use cross validation.

These supposed good movies were rated by very few users and small sample sizes lead to uncertainty. Therefore, larger estimates of b_i, negative or positive, are more likely. Therefore,these are noisy estimates that we should not trust, especially when it comes to prediction. Large errors can increase our RMSE, so we would rather be conservative when unsure.

Movies with high ratings but a remarkably low viewership count create uncertainty due to inadequate sample sizes. Similarly, certain users consistently rate movies from moderately high to very high, yet their total ratings amount to merely 10 instances. Consequently, these allegedly excellent movies lack substantial user feedback, resulting in uncertain evaluations. This situation increases the likelihood of larger estimates of 'b_i', whether negative or positive. As a result, these estimations are unreliable, especially in predictive contexts, and should not be heavily relied upon. Elevated errors could contribute to an increase in our RMSE, prompting a preference for a cautious approach when encountering uncertainty.

Regularization allows us to penalize substantial estimates derived from limited sample sizes. In fact, guessing that movies with few viewers are just average movies and add a penalty (lambda) in the least square equation (rather than ignoring those movies) will be the better option for the prediction.

According to the above equation, n_i is the number of ratings made for movie i. When the number of ratings are large, n_i is very large. a case which will give us a stable estimate, then the penalty λ is effectively ignored since $n_i + \lambda$ is effectively n_i. However, when the n_i is small, then the estimate $b_i(\lambda)$ is shrunken towards 0. The larger λ, the more we shrink.To select λ we can use cross validation.

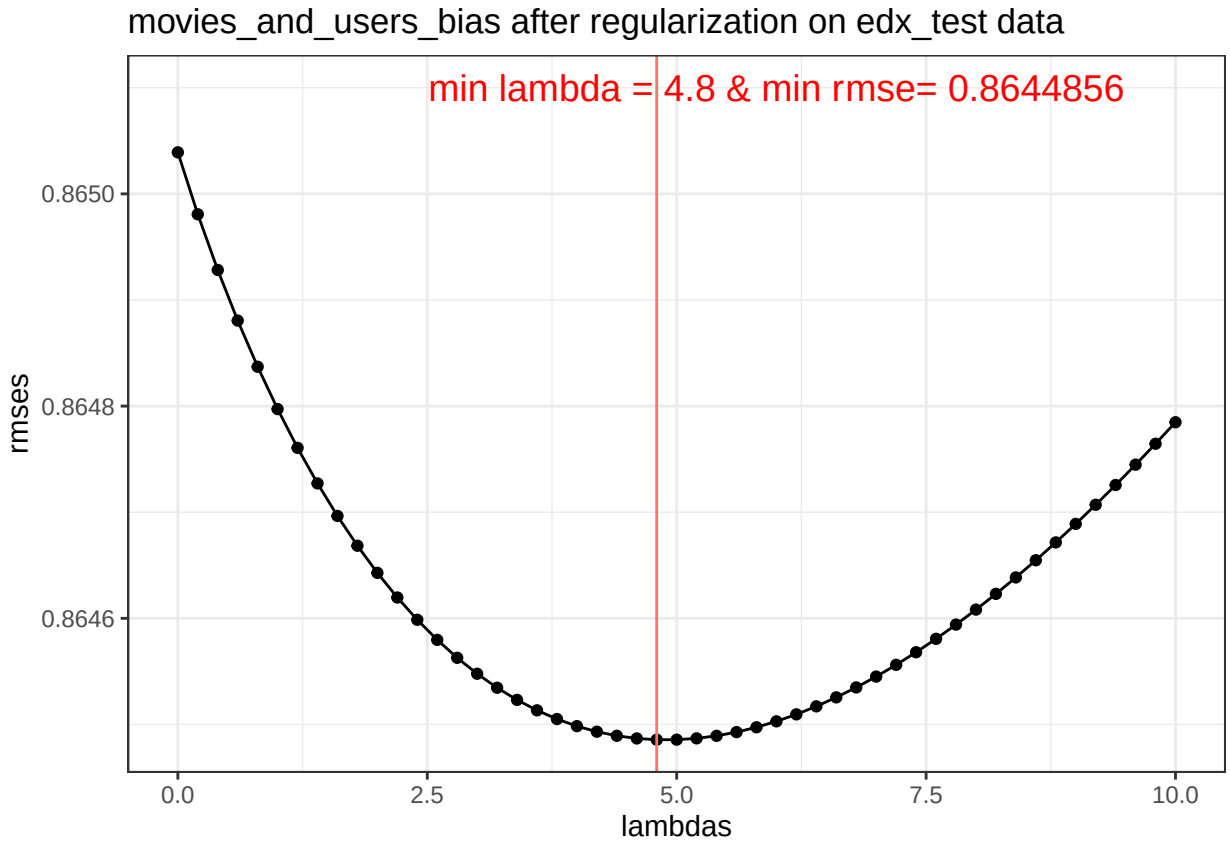
```
library(kableExtra)
#################################################################################
#movies_users_effect_with_regularization: testing on edx_test set
#Cross validation to chose a better lambda value to get the minimum RMSE

#lambda values chose between 0 to 10 with 0.2 interval
lambdas <- seq(0, 10, 0.20)


rmses <- sapply(lambdas, function(lambda){
  mu <- mean(edx_train$rating)
  #movie bias efffect data frame
  b_i <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  #movie and user effect factor
  b_u <- edx_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  #predictiing RMSE
  predicted_ratings <- edx_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

    return( RMSE(edx_test$rating,predicted_ratings))
})
```

```
#Select the minimum RMSE after regularization
min(rmses)
```

```
## [1] 0.8644856
```

```
# visulaization movies and user bias effect after regularization
data.frame(lambdas,rmses) %>%
  ggplot(aes(lambdas,rmses))+
  geom_point()+ geom_line()+
  theme_bw()+
  geom_vline(aes(xintercept = lambdas[which.min(rmses)], color = "red",linetype="dashed"))+
  annotate("text", x = 6, y = 0.8651, label = print(paste("min lambda =", lambdas[which.min(rmses)],
                                                          "& min rmse=",round(min(rmses),7))),
           color = "red", size = 5)+ theme(legend.position = "none")+
  ggtitle("movies_and_users_bias after regularization on edx_test data")
```

```
## [1] "min lambda = 4.8 & min rmse= 0.8644856"
```




```
#Optimum lambda value which give minimum RMSE
lambda <- lambdas[which.min(rmses)]
print(lambda)
```

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

```
#Minimum RMSE
movie_user_with_regularization_bias<-format(min(rmses),digits = 7)

models_df<-bind_rows(models_df, tibble(method="Movie_user_bias_with_regularization",
                                       RMSE=movie_user_with_regularization_bias,
                                       dataset="edx_train vs edx_test"))




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

method	RMSE	dataset
Required_RMSE	0.8649	NA
Basic_model	1.06062	edx_train vs edx_test
User effect alone	0.9782858	edx_train vs edx_test
Movie_bias_alone	0.9432105	edx_train vs edx_test
Movie_User bias effect	0.8650391	edx_train vs edx_test
Movie_user_bias_with_regularization	0.8644856	edx_train vs edx_test

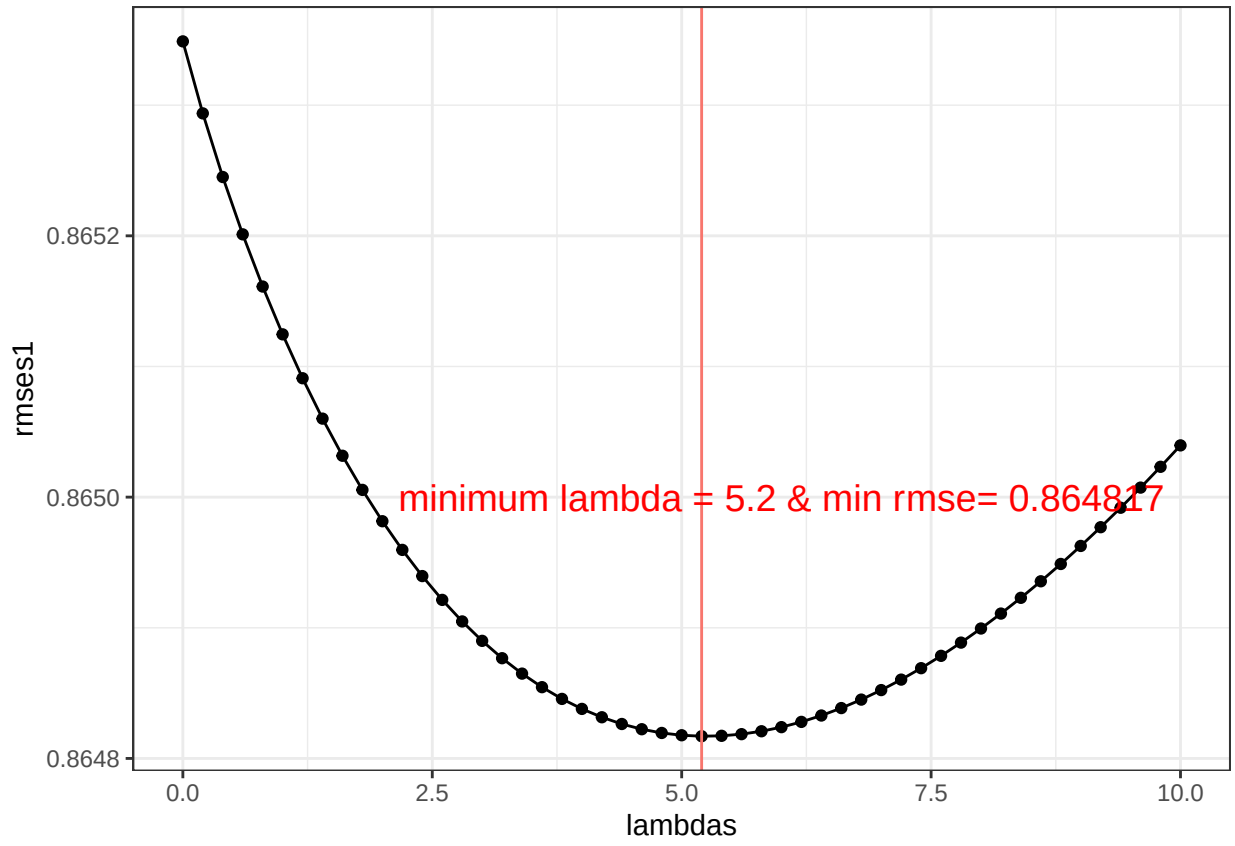
5.2 Final model: Validation using Movie and users effect with Regularization

The movies and users bias effect regularization reduced the RMSE below the reguired value. Therefore, this model was used for final_holdout_data set. The entire edx data was used as the training set

```
library(kableExtra)

lambdas <- seq(0, 10, 0.2)

rmses1 <- sapply(lambdas, function(lambda){

  mu <- mean(edx$rating)

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

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

  predicted_ratings <- final_holdout_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

    return(RMSE(final_holdout_test$rating,predicted_ratings))
})
```

```
#visualization
data.frame(lambdas,rmses1) %>%
  ggplot(aes(lambdas,rmses1))+
  geom_point()+ geom_line()+
  theme_bw()+
  geom_vline(aes(xintercept = lambdas[which.min(rmses1)], color = "red", linetype="dashed"))+
  annotate("text", x = 6, y = 0.865, label = print(paste("minimum lambda =", lambdas[which.min(rmses1)]
           color = "red", size = 5)+ theme(legend.position = "none")
```

```
## [1] "minimum lambda = 5.2 & min rmse= 0.864817"
```




```
#Identify the optimal lambda to get the minimum RMSE
lambda <- lambdas[which.min(rmses1)]
print(lambda)
```

```
## [1] 5.2
```

```
#Minimum RMSE value
Final_model_movie_user_with_regularization_bias<-format(min(rmses1),digits = 7)
```

```
#Add the RMSE value to the models_df table
models_df<-bind_rows(models_df, tibble(method="Final_model_Movie_user_bias_with_regularization",
                                       RMSE=Final_model_movie_user_with_regularization_bias,
                                       dataset="edx vs final_holdout_test"))



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

method	RMSE	dataset
Required_RMSE	0.8649	NA
Basic_model	1.06062	edx_train vs edx_test
User effect alone	0.9782858	edx_train vs edx_test
Movie_bias_alone	0.9432105	edx_train vs edx_test
Movie_User bias effect	0.8650391	edx_train vs edx_test
Movie_user_bias_with_regularization	0.8644856	edx_train vs edx_test
Final_model_Movie_user_bias_with_regularization	0.864817	edx vs final_holdout_test

6 Conclusion

The conclusive model, labeled as the **Final_model_Movie_user_bias_with_regularization**, proved to be the most effective, surpassing the project's required RMSE goal of 0.86490. This optimized model achieved an RMSE of 0.864817 due to the integration of regularization, marking an improvement of approximately 18.4% over the basic model. Interestingly, when the edx data was split into train sets of 96% (p=0.04) and 95% (0.05) respectively, there were noticeable enhancements of 18.62763% and 18.54369% in performance.

Notably, not only did the **Final_model_Movie_user_bias_with_regularization** improve the model, but the **Movie_User bias effect model** also managed to reduce the RMSE below 0.86490. Hence, it might be beneficial to fine-tune and optimize the working model by experimenting with different values of p for optimal outcomes. Additionally, exploring alternative machine learning algorithms could further enhance the model's performance.

During the Netflix challenge, the victors effectively utilized matrix factorization to enhance their model. This approach played a pivotal role in predicting user preferences based on their historical ratings, ultimately enabling Netflix to provide more accurate recommendations. However, it's worth noting that matrix factorization is computationally demanding.