HarvardX: PH125.9x Data Science - MovieLense project

Kerstin Tasotti

2022-02-23

https://github.com/KerstinTasotti/Movielense

1. INDRODUCTION

This project contains the MovieLens dataset which is related to the edX course HarvardX PH125.9x - Data Science:Capstone Course. The aim of this project is to demonstrate the acquired skills in R programming and their analysis in a real world datasets. The starting point is the MovieLens dataset which contains more than 9000000 different movie recommendations from users. The insights from this analysis are used to generate predictions of movies which are compared with the actual ratings to check the quality of the prediction algorithm. Therefor the dataset is split into a training set (edx) and a final hold-out test set (validation). The objective was for the final algorithm to predict ratings with a root mean square error (RMSE) of less than 0.86490 versus the actual rating included in the validation set. A recommendation system is a subclass of information filtering system that seeks to predict the “rating” or “preference” a user would give to an item. Recommendation systems is one of the most used machine learning algorithms and will be used in nearly all different areas of our life (Trading, Hospitality, Travelling,…). Companies like Amazon use these systems to learn more about their customer and provide them with products more effectively. In the MovieLens dataset user rate different movies with points between 0 and 5 and also, half points can be given. This dataset is prepared and different analysis were done to develop a algorithm of a machine learning which can predict movie rates.

2. DATASET

For this project we focus on the 10M version of MovieLens dataset collected by GroupLens Research and it can be found in MovieLens web site (http://movielens.org).

2.1. DATA LOAD

The data set is loaded using the code provides by the course in structure from https://learning.edx.org/course/course-v1:HarvardX+PH125.9x+2T2021/block-v1:HarvardX+PH125.9x+2T2021+type@sequential+block@e8800e37aa444297a3a2f35bf84ce452/block-v1:HarvardX+PH125.9x+2T2021+type@vertical+block@e9abcdd945b1416098a15fc95807b5db.

##

Create edx set, validation set (final hold-out test set)

##

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>")
if(!require(data.table)) install.packages("data.table", repos =
"<http://cran.us.r-project.org>")

library(tidyverse) library(caret) library(data.table)

### MovieLens 10M dataset:

### <https://grouplens.org/datasets/movielens/10m/>

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

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

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

movies <- str_split_fixed(readLines(unzip(dl,
```

```
"ml-10M100K/movies.dat")), "::", 3) colnames(movies) <- c("movieId",
"title", "genres")

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)
```

3. ANALYSIS OF THE DATA

All analysis in this section will be done with the training set (edx). The validation set will be used for the final test of the developed algorithm.

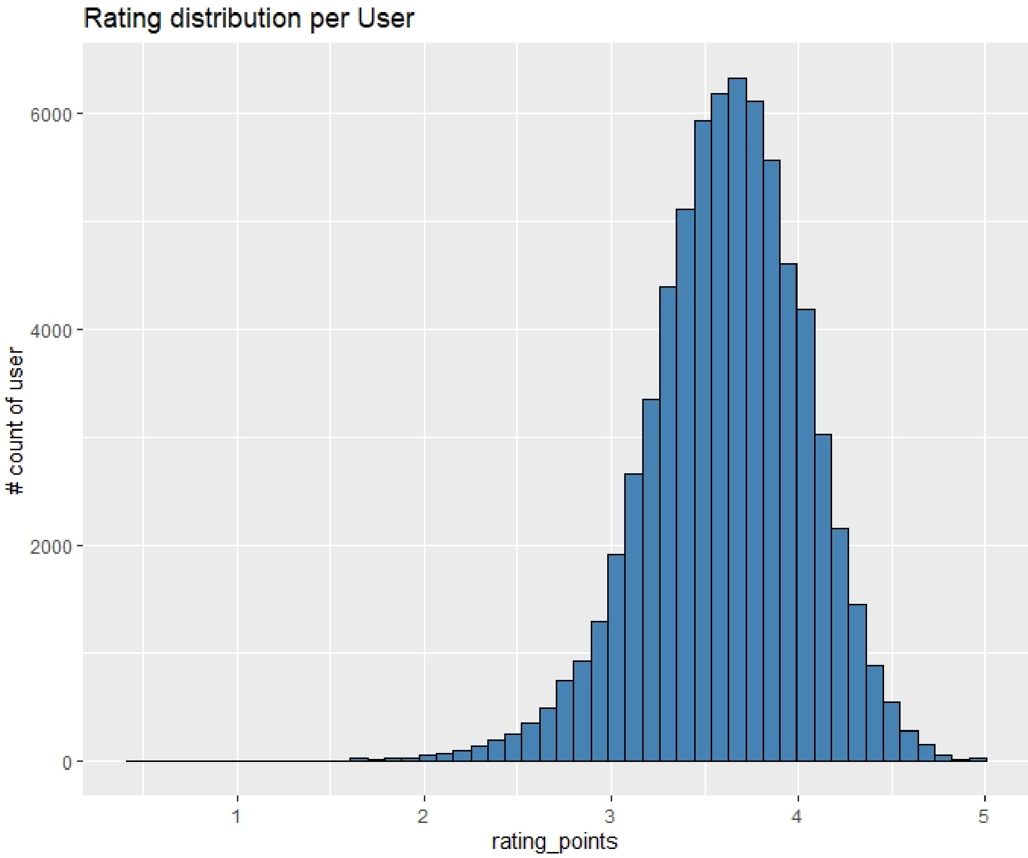
At first the structure of the dataset will be analyzed to get familiar with it. The data set contains 9000055 rows and 6 variables (userId, movieId, rating, timestamp, tile and genres). The movie title contains the year of publication.

```
str(edx)

head(edx) %>% print.data.frame
```

The rating is between 0.5 and 5 points with a mean value of 3.512 and a median of 4.0.

```
summary(edx)
```

The histogram shows the distribution of the of the average rating from the user.It shows a norm distributed curve




```
edx %>% group_by (userId) %>% summarise(rating_points=mean(rating)) %>% ggplot(aes(rating_points))+geom_histogr
am(bins=50,color="black", fill="steelblue")+ylab("# count of user")+ ggtitle("Rating distribution per User")
```

3.1. MOVIE

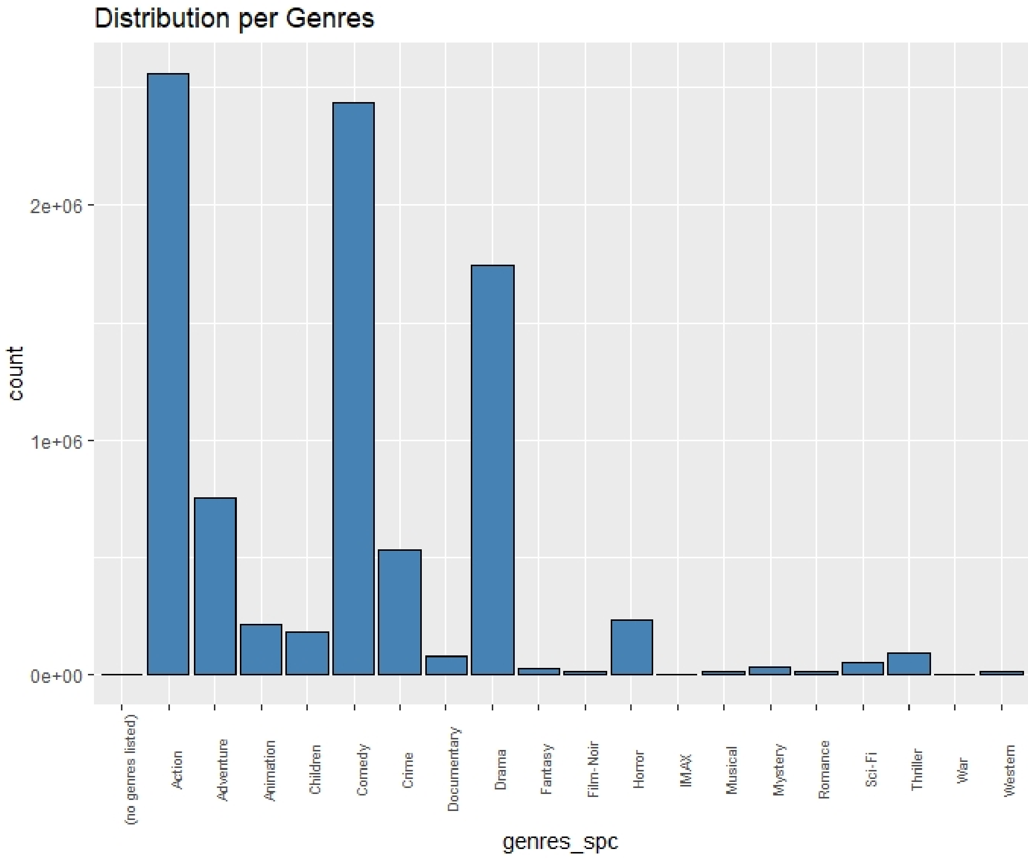
In the dataset are 10677 unique movies and 69878 different user which rated the movies in 19 specific genes (+ one empty genres= “no genres listed”). The dataset has 797 unique genres combinations, as some movies has assigned more than one specific genre.

```
edx %>% summarise(countmovies=n_distinct(movieId),countuser=n_distinct(userId),countgenres=n_distinct(genres))
```

The separation of the genres combination allows a distribution calculation of the ratings per single genres.

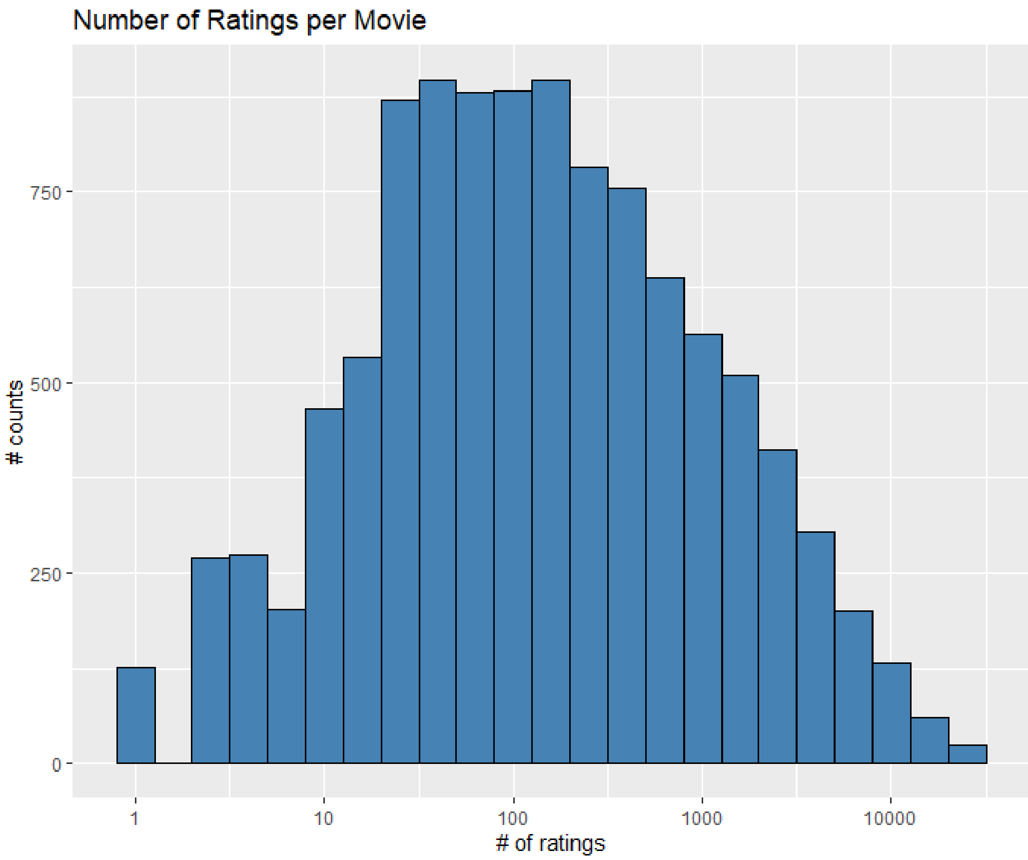
```
edx <- edx %>% mutate(genres_spc=as.character(str_replace(edx$genres,"\\|.*","")))
genres_spc <- str_replace(edx$genres,"\\|.*","")
genres_spc <- genres_spc[!duplicated(genres_spc)]
genres_spc
```

The histogram shows the distribution per genres. The rating count number in the bar chart are so high because there are many combinations of genres and therefore the rating is counted several times. Action movies get the most respectively the highest ratings.




```
edx %>% ggplot( aes(x=genres_spc,y=sum(rating))) +geom_bar(stat="identity", width=0.5, fill="steelblue")+ ggtit
le ("Distribution per Genres")+ theme(axis.text.x=element_text(angle=90,size=7))
```

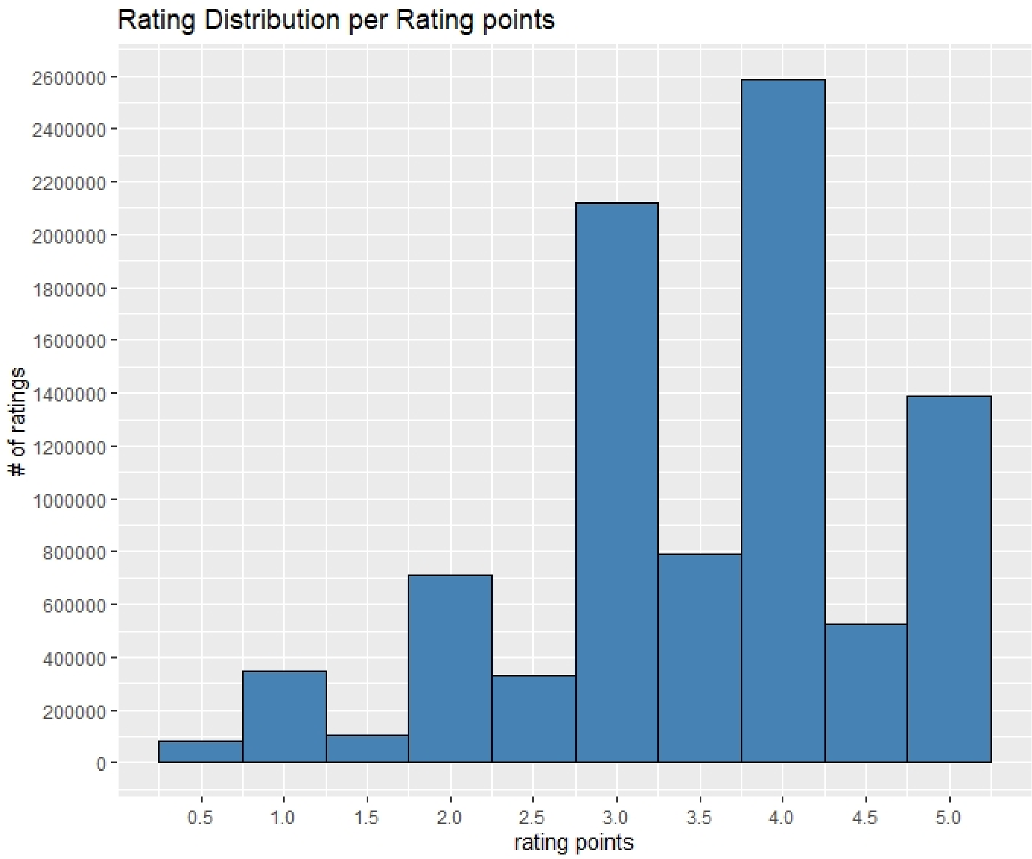
The next histogram shows the distribution of the rating points per movie. We can see that some movies get rated more and higher points than other. Blockbuster movies watched by millions and gets more ratings, independent movies watched by just a few user. About 100 movies has only one rating.




```
edx %>% count(movieId) %>%   ggplot(aes(n)) +geom_histogram(bins=30, binwidth = 0.2, color="black", fill="steel
blue")+scale_x_log10() + labs(x="# of ratings",y="# counts of movies")+ggtitle("Number of Ratings per Movie")
```

The count of movies per rating shows that the most movies have a rating of 4 and the fewest movies have a rating of 0.5. A rating of 4 is given more than 30-fold more often as the worst rated. The table and the histogram shows the count of ratings per rating points.

```
edx %>% group_by(rating)%>% summarise(count=n())%>% top_n(10) %>%
arrange(desc (count))

A tibble: 10 x 2
   rating   count
    <dbl>   <int>
 1     4   2588430
 2     3   2121240
 3     5   1390114
 4     3.5  791624
 5     2    711422
 6     4.5  526736
 7     1    345679
 8     2.5  333010
 9     1.5  106426
10     0.5   85374
```



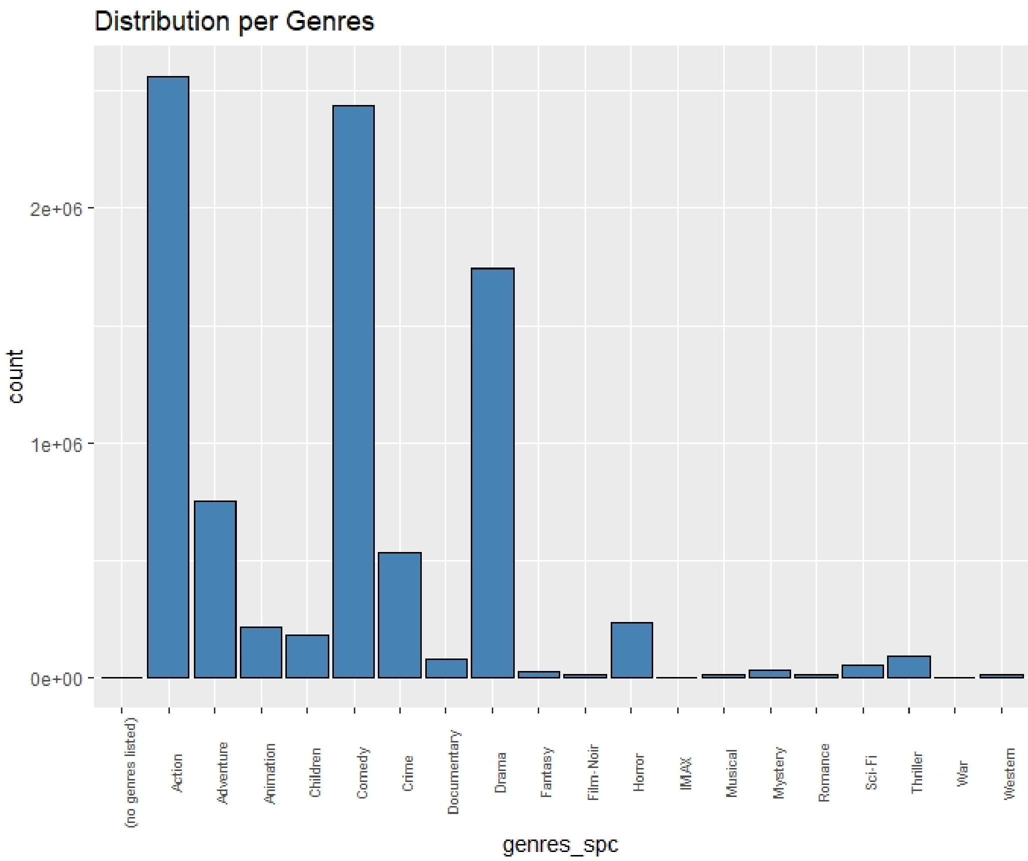

```
edx %>% ggplot(aes(rating)) +geom_histogram(binwidth=0.5, color="black", fill="steelblue")+scale_x_continuous(b
reaks = c(seq(0.5,5,0.5)))+scale_y_continuous(breaks =  c(seq(0,3000000,200000)))+ labs(x="rating points",y="#
of ratings")+ggtitle("Rating Distribution of Movies")
```

The top10 movies and the top10 genre combinations and single genres can be calculated. Some movies are more rated than other. The genres combination with the most rating are “Drama” and Comedy”. For separated genres Action and Comedy has the most rating.

```
edx %>% group_by(title)%>% summarise(count=n())%>% top_n(10) %>% arrange(desc (count))

A tibble: 10 x 2
   title                                                        count
   <chr>                                                        <int>
 1 Pulp Fiction (1994)                                          31362
 2 Forrest Gump (1994)                                          31079
 3 Silence of the Lambs, The (1991)                             30382
 4 Jurassic Park (1993)                                         29360
 5 Shawshank Redemption, The (1994)                             28015
 6 Braveheart (1995)                                            26212
 7 Fugitive, The (1993)                                         25998
```

```
 8 Terminator 2: Judgment Day (1991)                             25984
 9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  25672
10 Apollo 13 (1995)                                              24284


edx %>% group_by(genres)%>% summarise(count=n())%>% top_n(10) %>% arrange(desc (count))

A tibble: 10 x 2
   genres                     count
   <chr>                      <int>
 1 Drama                     733296
 2 Comedy                    700889
 3 Comedy|Romance            365468
 4 Comedy|Drama              323637
 5 Comedy|Drama|Romance      261425
 6 Drama|Romance             259355
 7 Action|Adventure|Sci-Fi   219938
 8 Action|Adventure|Thriller 149091
 9 Drama|Thriller            145373
10 Crime|Drama               137387


edx %>% group_by(genres_spc)%>% summarise(count=n())%>% top_n(10) %>% arrange(desc (count))

A tibble: 10 x 2
   genres_spc    count
   <chr>         <int>
 1 Action      2560545
 2 Comedy      2437260
 3 Drama       1741668
 4 Adventure    753650
 5 Crime        529521
 6 Horror       233074
 7 Animation    218123
 8 Children     181217
 9 Thriller      94718
10 Documentary   80966
```

The histogram shows the distribution of rating counts per genres



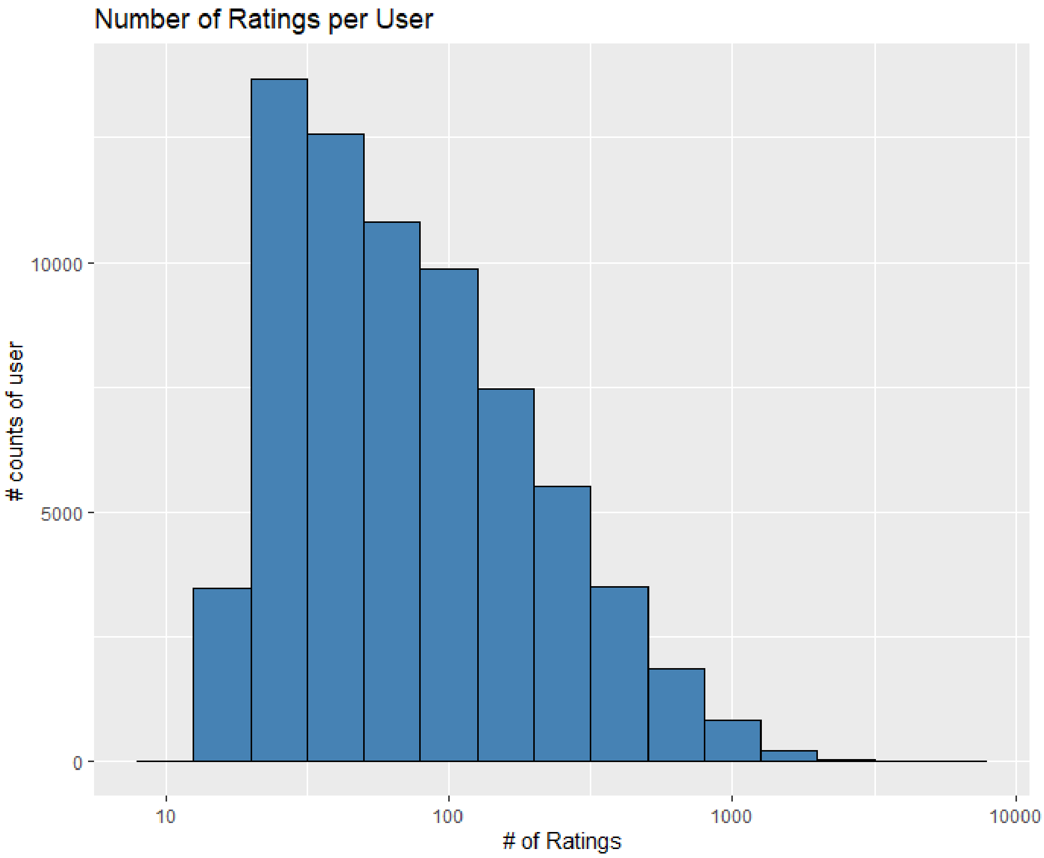
```
edx %>% group_by(rating) %>% ggplot( aes(genres_spc)) +geom_histogram( stat="count",color="black", fill="steelb
lue")+ theme(axis.text.x=element_text(angle=90,size=7))+ggtitle("Distribution per Genres")
```

The number of ratings per movie shows a high difference between the users.

```
edx %>% group_by(movieId) %>% summarise(sum(rating)) %>% arr
```

3. 2. USER

In the following histogram the rating per user is displayed. Some users are more active than others at rating movies.




```
edx %>% count(userId) %>%   ggplot(aes(n)) +geom_histogram(bins=30, binwidth = 0.2, color="black", fill="steelb
lue")+scale_x_log10() + labs(x="# of Ratings",y="# counts of user")+ggtitle("Number of Ratings per User")
```

3.3. TIMESTAMP

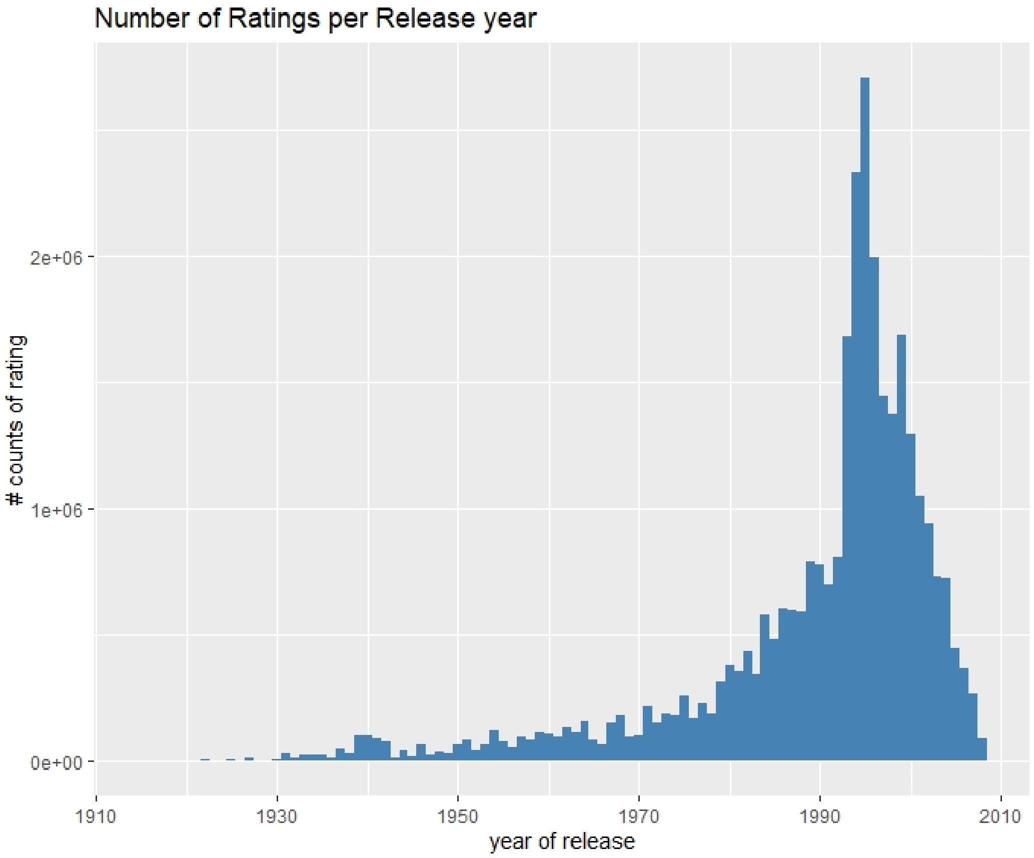
The format of the timestamp will be changed to Date format and display the rating year. In the title of the movies is in brackets also a year. This will be separated and displayed as release year. The rating date and both year columns (release and rating) are added as new columns into the dataset.

```
edx <- edx %>% mutate(rating_date=as.Date(as.POSIXct(timestamp, origin="1970-01-01")))%>% mutate(rating_year=ye
ar(rating_date))
edx <- edx %>% mutate(release_year=as.integer(substr(title,str_length(title)-4,str_length(title)-1)))
Summary(edx)

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

```
 Min.   :1995-01-09   Min.   :1995   Min.   :1915  
 1st Qu.:2000-01-01   1st Qu.:2000   1st Qu.:1987  
 Median :2002-10-24   Median :2002   Median :1994  
 Mean   :2002-09-21   Mean   :2002   Mean   :1990  
 3rd Qu.:2005-09-15   3rd Qu.:2005   3rd Qu.:1998  
 Max.   :2009-01-05   Max.   :2009   Max.   :2008  
```

The release year for the movies has a range between 1915 and 2008 and the movies are rated between 1995 and 2009. 2000 the most movies were rated followed by 2005 and 1996. In the release years 1995,1994 and 1996 are movies with the highest ratings.

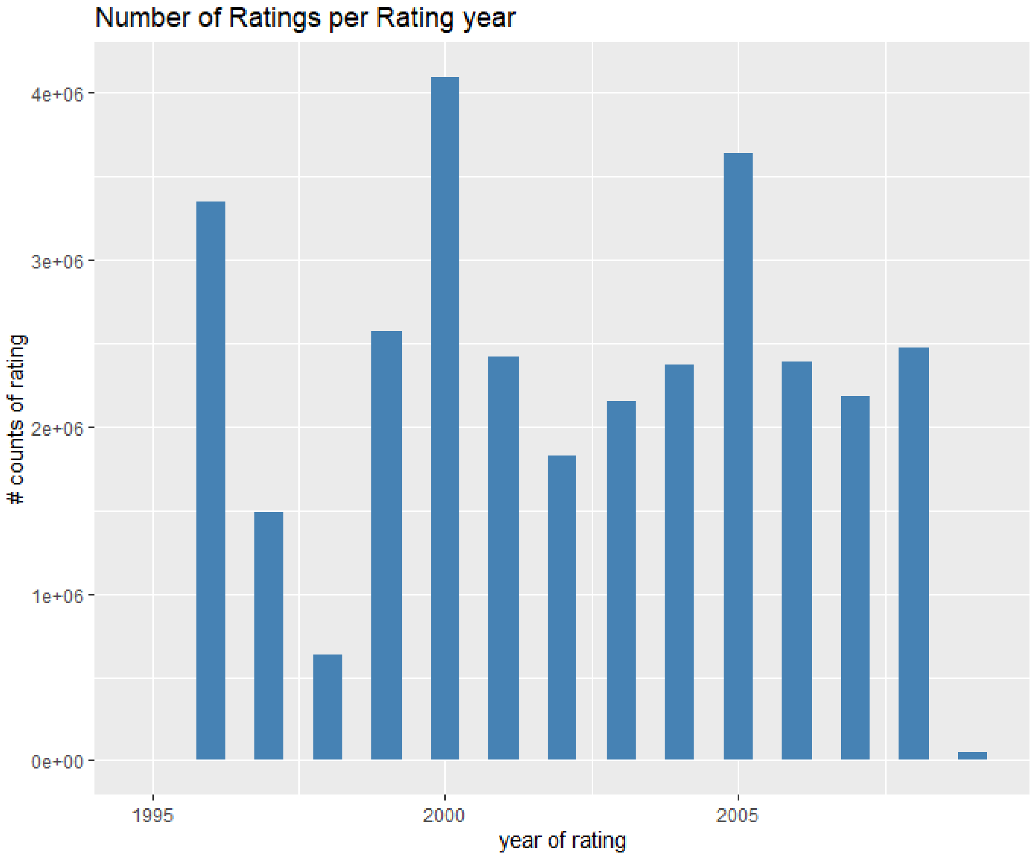
```
edx %>% group_by(edx$release_year)%>% summarise(count=n())%>% top_n(10) %>% arrange(desc (count))

 A tibble: 10 x 2
   `edx$release_year`  count
                <int>  <int>
 1               1995 786762
 2               1994 671376
 3               1996 593518
 4               1999 489537
 5               1993 481184
 6               1997 429751
 7               1998 402187
 8               2000 382763
 9               2001 305705
10               2002 272180
```




```
edx %>% group_by(release_year)%>% ggplot( aes(x=release_year, y=rating)) +geom_bar(stat="identity", width=1, co
lor="black", fill="steelblue") + labs(x="year of release",y="# counts of rating")+ggtitle("Number of Ratings pe
r Release year")
```

```
edx %>% group_by(edx$rating_year)%>% summarise(count=n())%>% top_n(10) %>% arrange(desc (count))

A tibble: 10 x 2
   `edx$rating_year`   count
               <int>   <int>
 1              2000 1144349
 2              2005 1059277
 3              1996  942772
 4              1999  709893
```

```
5              2008  696740
6              2004  691429
7              2006  689315
8              2001  683355
9              2007  629168
10             2003  619938
```




```
edx %>% group_by(rating_year) %>% ggplot( aes(x=rating_year, y=rating)) +geom_bar(stat="identity", width=0.5, f
ill="steelblue")  + labs(x="year of rating",y="# counts of rating")+ggtitle("Number of Ratings per Rating yea
r")
```

4. RESULTS - MACHINE LEARNING ALGORITHM

TO see how this type of machine learning as users look for movie recommendations. The test set for this calculation of accuracy is provided in the dataset.

LOSS FUNCTION

We define the rating for movie i by user and donate our prediction. The residual mean squared error RMSE is defined as:

\[RMSE <- function(true_ratings,predicted_ratings){sqrt(mean((true_ratings - predicted_ratings)^2))}\]

The lower the RMSE, the better it is.

4.1. FIRST MODEL

We start to build the simplest possible recommendation system. We predict the same rating for all movies regardless of user. The estimation minimizes the RMSE is the least squares estimation of the rating and is the average of all ratings.

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

[1] 3.512465
```

For the prediction of all unknown ratings with μ_hat following RMSE will be calculated:

```
naive_rmse <- RMSE(validation$rating, mu_hat)
naive_rmse
```

```
[1] 1.061202
```

The results will be saved in a table.

```
rmse_results <- data_frame (method="Just the average", RMSE=naive_rmse)
rmse_results
```

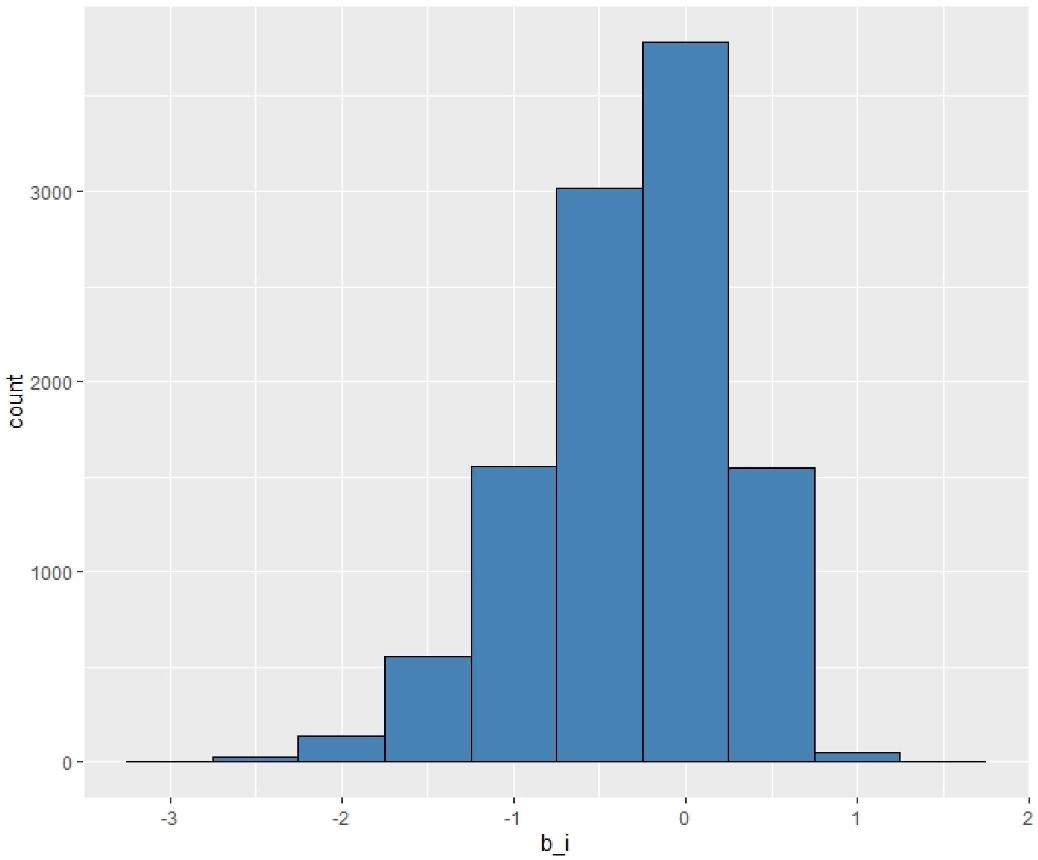
4.2. MOVIE EFFECT

Priviuous calculations show that some movies are just generally rated higher than others. This intuition that different movies are rated differently is confirmed by data. We add the term b_i to our previous model to represent the average ranking for movie i.

The lm() function will be very slow and so I compute it in the whole model using the average like in the following way:

```
mu<- mean(edx$rating)
movie_avgs <- edx %>% group_by(movieId) %>% summarise(b_i=mean(rating-mu))
```

we can see that these estimates vary substantially:




```
edx %>% group_by (movieId) %>% summarise(b_i=mean(rating-mu)) %>% ggplot(aes(b_i)) +geom_histogram(bins=10, col
or="black", fill="steelblue")
```

We know that mu= 3.5 and b_i=1.5 implies a perfect five-star rating. We can see how much our prediction improves once we use following calculation:

```
predicted_ratings <- mu + validation %>%left_join(movie_avgs, by="movieId") %>% pull (b_i)
movie_effect_rmse<- RMSE(predicted_ratings,validation$rating)
movie_effect_rmse

[1] 0.9439087
```

The results will be saved in the table:

```
rmse_results <- bind_rows(rmse_results, data_frame(method="Movie Effect Model", RMSE=movie_effect_rmse))
rmse_results
```

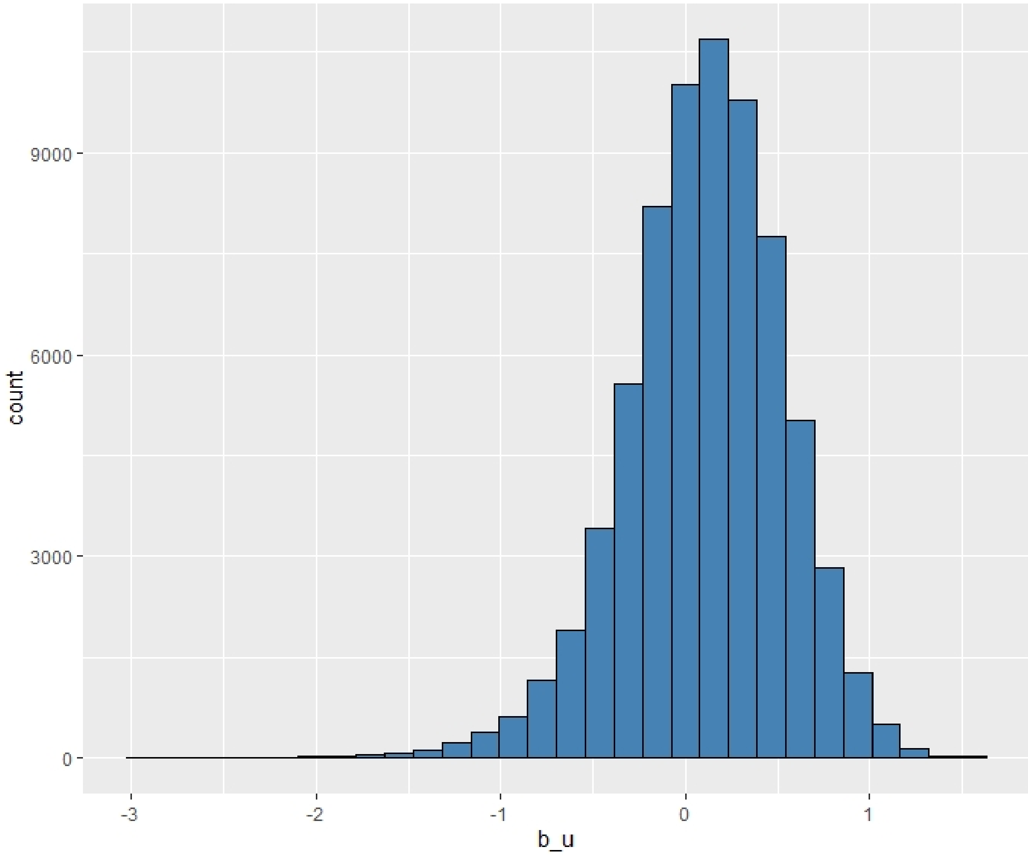
We can see an improvement.

4.3. USER AND MOVIE EFFECT

The average rating for user u computed for those that have rated over 100 movies:

The approximation is computed by the user average and we can construct predictors and see how much the RMSE improves:

```
mu<- mean(edx$rating)
user_avgs <- edx %>% left_join(movie_avgs, by="movieId") %>% group_by(userId) %>% summarise(b_u=mean(rating-mu-b_i))
```




```
edx %>% group_by (userId) %>% summarise(b_u=mean(rating-mu)) %>% filter(n()>=100)%>% ggplot(aes(b_u)) +geom_histogram(bins=30, color="black", fill="steelblue")
```

We can compute the predictors and see how the RMSE improve again:

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


user_effect_rmse <- RMSE(predicted_ratings,validation$rating)
user_effect_rmse

[1] 0.8653488
```

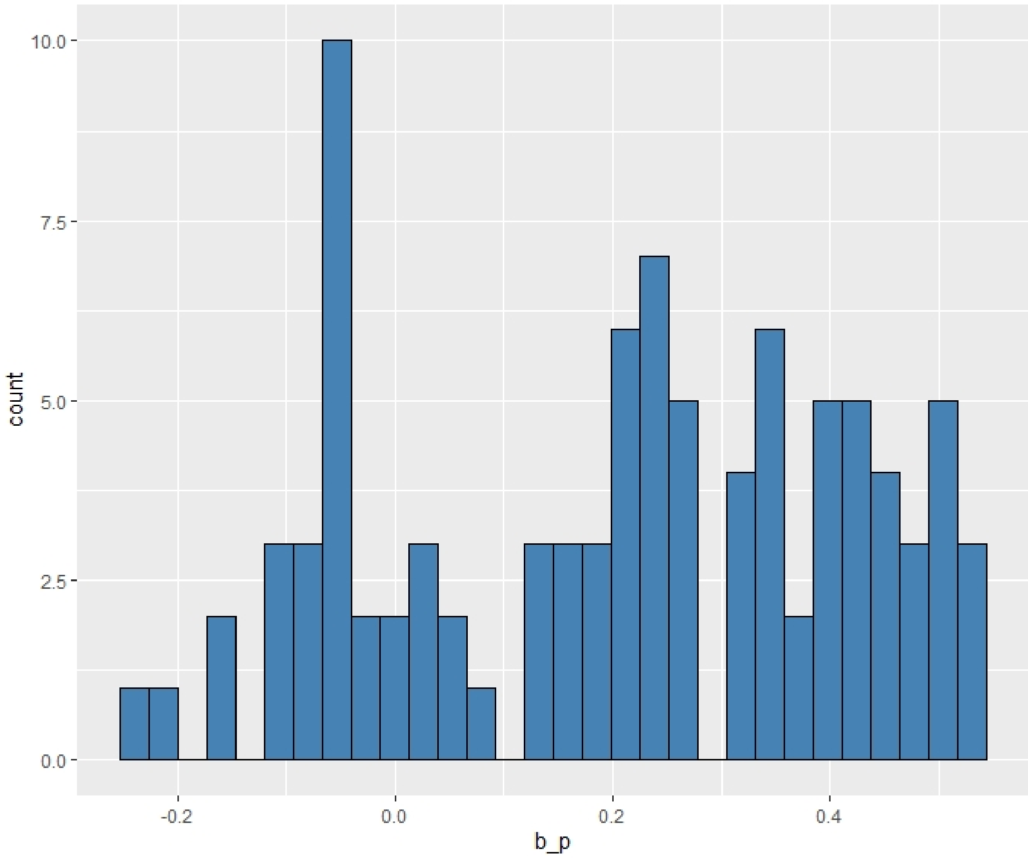
The results will be saved in a table:

```
rmse_results <- bind_rows(rmse_results, data_frame(method="User Effect Model", RMSE=user_effect_rmse))
rmse_results
```

4.4. RELEASE YEAR, USER AND MOVIE EFFECT

The release year column must be created in the validation set to compute the prediction of the release year to the algorithm.

```
validation <- validation %>% mutate(release_year=as.integer(substr(title,str_length(title)-4,str_length(title)-1)))
```




```
edx %>% group_by (release_year) %>% summarise(b_p=mean(rating-mu)) %>% ggplot(aes(b_p)) +geom_histogram(bins=30, color="black", fill="steelblue")
```

The new corresponding approximation is calculated by the release year average:

```
mu<- mean(edx$rating)
release_year_avgs <- edx %>% group_by(release_year) %>%left_join(movie_avgs, by="movieId") %>% left_join(user_avgs, by="userId") %>% summarise(b_p=mean(rating-mu-b_i-b_u))
```

The new prediction and RMSE is calculated and saved in the table:

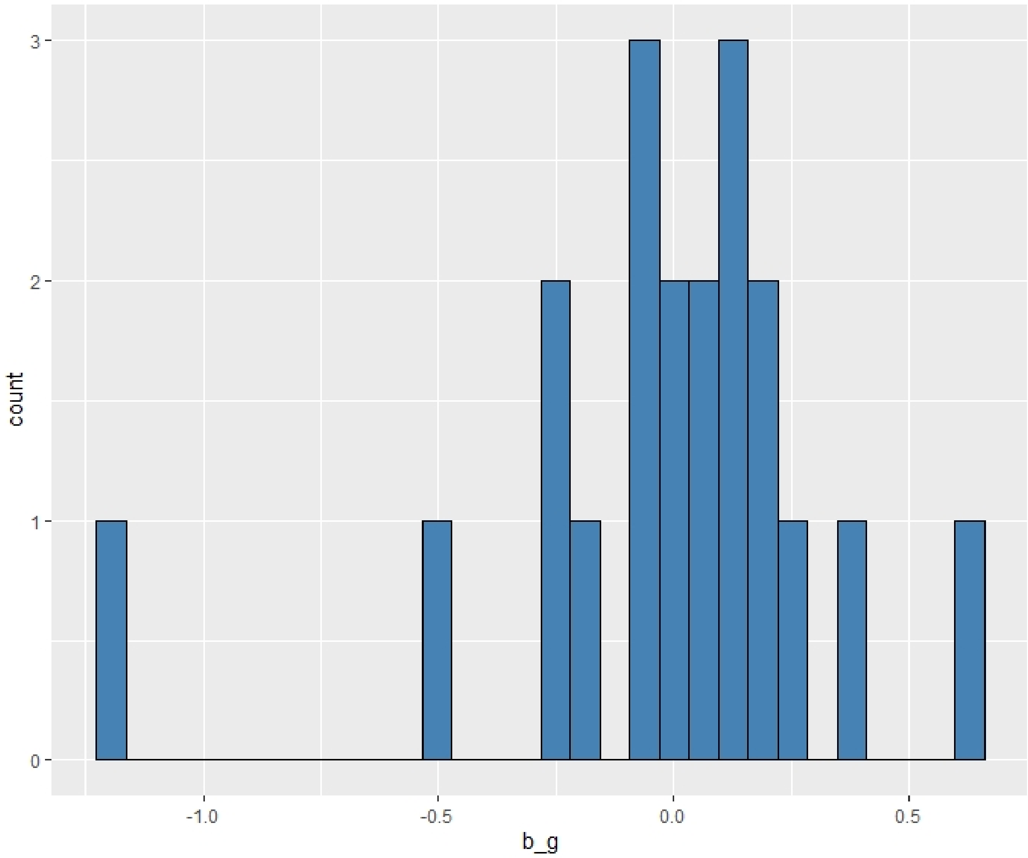
```
predicted_ratings <- validation %>%left_join(release_year_avgs, by="release_year") %>% left_join(movie_avgs, by="movieId") %>% left_join(user_avgs, by="userId") %>% mutate (pred=mu+b_i +b_u+b_p)%>% pull(pred)
release_year_effect_rmse <- RMSE(predicted_ratings,validation$rating)
release_year_effect_rmse

[1] 0.8650043

rmse_results <- bind_rows(rmse_results, data_frame(method="Release year Effect Model", RMSE=release_year_effect_rmse))
rmse_results
```

The effects on the improvement of RMSE is smaller as in the step before.

4.5. GENRE, RELEASE YEAR, USER AND MOVIE EFFECT

The genres column must be created in the validation set to compute the prediction of genres to the algorithm.

```
validation <- validation %>% mutate(genres_spc=as.character(str_replace(validation$genres,"\\|.*","")))
```




```
edx %>% group_by (genres_spc) %>% summarise(b_g=mean(rating-mu)) %>% ggplot(aes(b_g)) +geom_histogram(bins=30,
color="black", fill="steelblue")
```

The approximation is calculated by the single genres average:

```
mu<- mean(edx$rating)
genres_avgs <- edx %>% group_by(genres_spc)%>% left_join(release_year_avgs, by="release_year")%>% left_join(mov
ie_avgs, by="movieId") %>% left_join(user_avgs, by="userId") %>% summarise(b_g=mean(rating-mu-b_i-b_u-b_p))
```

The new prediction and RMSE is calculated and saved in the table:

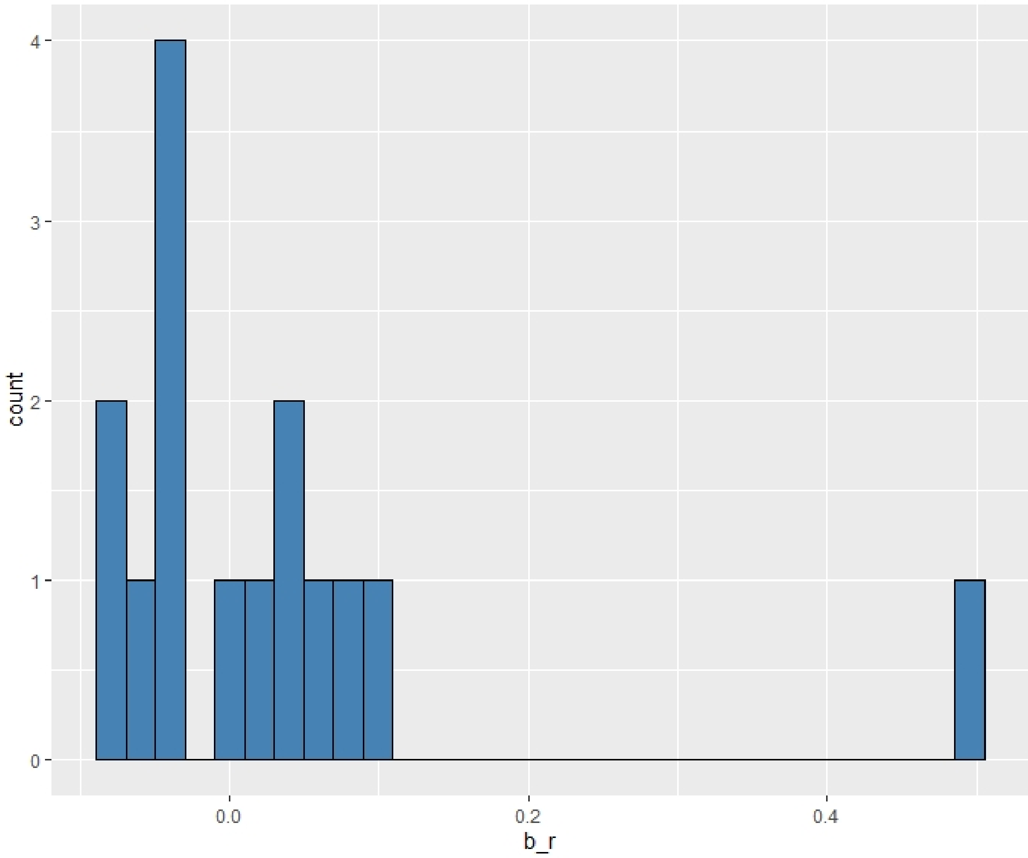
```
predicted_ratings <- validation %>%left_join(genres_avgs, by="genres_spc") %>%left_join(release_year_avgs, by
="release_year") %>% left_join(movie_avgs, by="movieId") %>% left_join(user_avgs, by="userId") %>% mutate (pred
=mu+b_i +b_u+b_g+b_p)%>% pull(pred)
genres_effect_rmse <- RMSE(predicted_ratings,validation$rating)
genres_effect_rmse

[1] 0.8649057

rmse_results <- bind_rows(rmse_results, data_frame(method="Genres Effect Model", RMSE=genres_effect_rmse))
rmse_results
```

The genres have a 3.5-fold lower effect on the improvement of RMSE than the release year.

4.6. RATING YEAR, RELEASE YEAR, USER AND MOVIE EFFECT

The rating year column must be created in the validation set to compute the prediction of genres to the algorithm.

```
validation <- validation %>% mutate(rating_date=as.Date(as.POSIXct(timestamp, origin="1970-01-01")))%>% mutate
(rating_year=year(rating_date))
```




```
edx %>% group_by (rating_year) %>% summarise(b_r=mean(rating-mu)) %>% ggplot(aes(b_r)) +geom_histogram(bins=30,
color="black", fill="steelblue")
```

The approximation is calculated by the rating year average:

```
mu<- mean(edx$rating)
rating_year_avgs <- edx %>% group_by(rating_year)%>% left_join(genres_avgs, by="genres_spc") %>% left_join(rele
ase_year_avgs, by="release_year") %>%left_join(movie_avgs, by="movieId") %>% left_join(user_avgs, by="userId")
%>% summarise(b_r=mean(rating-mu-b_i-b_u-b_p-b_g))
```

The new prediction and RMSE is calculated and saved in the table:

```
predicted_ratings <- validation %>%left_join(rating_year_avgs, by="rating_year") %>% left_join(genres_avgs, by
="genres_spc") %>% left_join(release_year_avgs, by="release_year")%>% left_join(movie_avgs, by="movieId") %>% l
eft_join(user_avgs, by="userId") %>% mutate (pred=mu+b_i+b_u+b_p+b_g+b_r)%>% pull(pred)
rating_year_effect_rmse <- RMSE(predicted_ratings,validation$rating)
rating_year_effect_rmse

[1] 0.8648283

rmse_results <- bind_rows(rmse_results, data_frame(method="Rating year Effect Model", RMSE=rating_year_effect_r
mse))
rmse_results
```

The rating year has only a very low effect on the improvement of RMSE.

4.7. REGULARIZED MOVIE AND USER EFFECT

Despite the large movie to movie variation and the different user ratings (some movies gets only few ratings and some user rates only few movies), the different ratings in the release and rating years our improvements in RMSE was good. If n is very large, the estimation will be very stable and the penalty lambda is effectively ignored since ni + lambda is about ni. When the ni is small, then the estimation bi_hat (lambda) is strunken towards 0. The larger lambda, the more we strink. We can optimize this model by using lambda. Let´s compute these regularized estimate of b_i, b_u, b_p, b_g and b_r using lambda =3:

```
lambda<- 3
mu<- mean(edx$rating)
movie_reg_avgs <- edx %>% group_by(movieId) %>% summarise(b_i_reg=sum(rating-mu)/(n()+lambda))
```

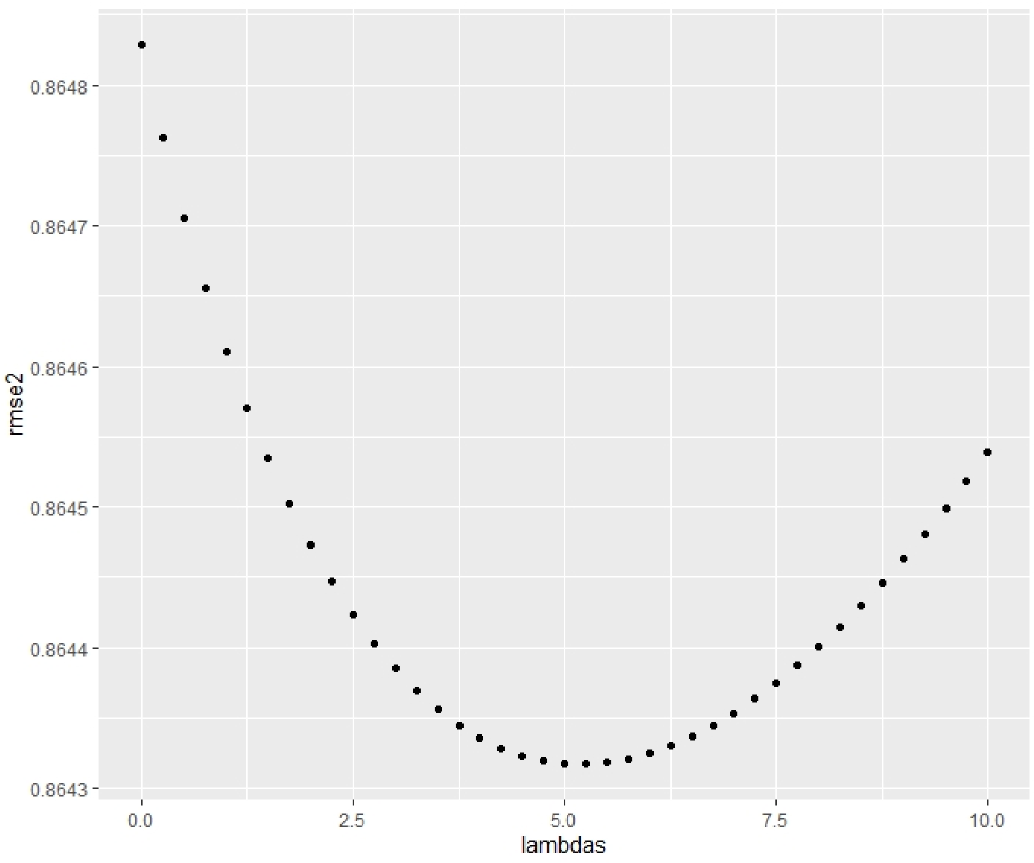
```
user_reg_avgs <- edx %>% group_by(userId) %>% left_join(movie_reg_avgs, by="movieId") %>% summarise(b_u_reg=sum
(rating-mu-b_i_reg)/(n()+lambda))

release_year_reg_avgs <- edx %>% group_by(release_year) %>% left_join(user_reg_avgs, by="userId") %>% left_join
(movie_reg_avgs, by="movieId") %>% summarise(b_p_reg=sum(rating-mu-b_i_reg-b_u_reg)/(n()+lambda))

genres_reg_avgs <- edx %>% group_by(genres_spc)%>% left_join(release_year_reg_avgs, by="release_year") %>%left_
join(movie_reg_avgs, by="movieId") %>% left_join(user_reg_avgs, by="userId") %>% summarise(b_g_reg=sum(rating-m
u-b_i_reg-b_u_reg-b_p_reg)/(n()+lambda))

rating_year_reg_avgs <- edx %>% group_by(rating_year)%>% left_join(genres_reg_avgs, by="genres_spc") %>%left_jo
in(release_year_reg_avgs, by="release_year") %>% left_join(movie_reg_avgs, by="movieId") %>% left_join(user_reg
_avgs, by="userId") %>% summarise(b_r_reg=sum(rating-mu-b_i_reg-b_u_reg-b_p_reg-b_g_reg)/(n()+lambda))

predicted_ratings <- validation %>%left_join(rating_year_reg_avgs, by="rating_year") %>%left_join(genres_reg_av
gs, by="genres_spc") %>%left_join(release_year_reg_avgs, by="release_year")%>%left_join(movie_reg_avgs, by="mov
ieId") %>% left_join(user_reg_avgs, by="userId") %>%  mutate (pred=mu+b_i_reg+b_u_reg+b_p_reg+b_g_reg+b_r_reg)%
>% pull(pred)

movie_user_year_genres_reg_effect_rmse <- RMSE(predicted_ratings,validation$rating)
movie_user_year_genres_reg_effect_rmse

[1] 0.8643849

rmse_results <- bind_rows(rmse_results, data_frame(method="Regularized Genres+Movie+User+Year Effect Model", RM
SE=movie_user_year_genres_reg_effect_rmse))
rmse_results
```

The penalized estimate provide an improvement over the least squares estimates.

4.8. CHOOSING THE PENALTY TERMS

Lambda is a turning parameter and a sequence between 0 and 10 with distances of 0.25 will be generate and compute the lowest RMSE.

```
lambdas <-seq(0,10,0.25)
mu<- mean(edx$rating)

rmse2 <- sapply(lambdas,function(l){
 mu<-mean(edx$rating)
 b_i_reg<-edx %>% group_by(movieId)%>% summarise(b_i_reg=sum(rating-mu)/(n()+l))
 b_u_reg<-edx %>% group_by(userId) %>% left_join(b_i_reg, by="movieId") %>% summarise(b_u_reg=sum(rating-mu-b_i
_reg)/(n()+l))

 b_p_reg<-edx %>% group_by(release_year) %>% left_join(b_i_reg, by="movieId") %>% left_join(b_u_reg, by="userI
d") %>% summarise(b_p_reg=sum(rating-mu-b_i_reg-b_u_reg)/(n()+l))

 b_g_reg<-edx %>% group_by(genres_spc) %>% left_join(b_p_reg, by="release_year") %>% left_join(b_i_reg, by="mov
ieId") %>% left_join(b_u_reg, by="userId") %>% summarise(b_g_reg=sum(rating-mu-b_i_reg-b_u_reg-b_p_reg)/(n()+
l))

 b_r_reg<-edx %>% group_by(rating_year) %>% left_join(b_g_reg, by="genres_spc") %>% left_join(b_p_reg, by="rele
ase_year") %>% left_join(b_i_reg, by="movieId") %>% left_join(b_u_reg, by="userId") %>% summarise(b_r_reg=sum(r
ating-mu-b_i_reg-b_u_reg-b_p_reg-b_g_reg)/(n()+l))

 predicted_ratings <-validation %>%  left_join(b_i_reg, by="movieId")%>%  left_join(b_u_reg,  by="userId")%>% l
eft_join(b_p_reg, by="release_year")%>%  left_join(b_g_reg, by="genres_spc")%>% left_join(b_r_reg, by="rating_y
ear")%>% mutate(pred=mu+b_i_reg+b_u_reg+b_p_reg+b_g_reg+b_r_reg) %>% pull(pred)
 return(RMSE(predicted_ratings, validation$rating))})

qplot(lambdas,rmse2)
```




```
lambda_low<- lambdas[which.min(rmse2)]
lambda_low
```

The optimal lambda with the lowest RMSE is 5.25. The new RMSE for the model with the lambda of 5.25 is compute as:

```
mu<- mean(edx$rating)
movie_reg_avgs <- edx %>% group_by(movieId) %>% summarise(b_i_reg=sum(rating-mu)/(n()+lambda_low))

user_reg_avgs <- edx %>% group_by(userId) %>% left_join(movie_reg_avgs, by="movieId") %>% summarise(b_u_reg=sum
(rating-mu-b_i_reg)/(n()+lambda_low))

release_year_reg_avgs <- edx %>% group_by(release_year) %>% left_join(user_reg_avgs, by="userId") %>% left_join
(movie_reg_avgs, by="movieId") %>% summarise(b_p_reg=sum(rating-mu-b_i_reg-b_u_reg)/(n()+lambda_low))

genres_reg_avgs <- edx %>% group_by(genres_spc)%>% left_join(release_year_reg_avgs, by="release_year") %>%left_
join(movie_reg_avgs, by="movieId") %>% left_join(user_reg_avgs, by="userId") %>% summarise(b_g_reg=sum(rating-m
u-b_i_reg-b_u_reg-b_p_reg)/(n()+lambda_low))

rating_year_reg_avgs <- edx %>% group_by(rating_year)%>% left_join(genres_reg_avgs, by="genres_spc") %>%left_jo
in(release_year_reg_avgs, by="release_year") %>% left_join(movie_reg_avgs, by="movieId") %>% left_join(user_reg
_avgs, by="userId") %>% summarise(b_r_reg=sum(rating-mu-b_i_reg-b_u_reg-b_p_reg-b_g_reg)/(n()+lambda_low))

predicted_ratings <- validation %>%left_join(rating_year_reg_avgs, by="rating_year") %>%left_join(genres_reg_av
gs, by="genres_spc") %>%left_join(release_year_reg_avgs, by="release_year")%>%left_join(movie_reg_avgs, by="mov
ieId") %>% left_join(user_reg_avgs, by="userId") %>%  mutate (pred=mu+b_i_reg+b_u_reg+b_p_reg+b_g_reg+b_r_reg)%
>% pull(pred)

movie_user_year_genres_reg_effect_rmse <- RMSE(predicted_ratings,validation$rating)
movie_user_year_genres_reg_effect_rmse

[1] 0.8643169

rmse_results <- bind_rows(rmse_results, data_frame(method="Regularized Model with optimal lambda", RMSE=movie_u
ser_year_genres_reg_effect_rmse))
rmse_results
```

The results from the table are:

```
A tibble: 8 x 2
  method                                              RMSE
```

```
  <chr>                                      <dbl>
1 Just the average                           1.06
2 Movie Effect Model                         0.944
3 User Effect Model                          0.865
4 Release year Effect Model                  0.865
5 Genres Effect Model                        0.865
6 Rating year Effect Model                   0.865
7 Regularized Genres+Movie+User Effect Model 0.864
8 Regularized Genres+Movie+User Effect Model 0.864
```

5. CONCLUSION

The objective of this project was to develop a recommendation system for movie rates from the dataset “MovieLens 10M” with a residual mean square error of less than 0.8649. I developed a naive model and different models with effects of movie, user, release year, genres and rating year with their regarding RMSE. This model is tested with the validation dataset. This algorithm includes a error loss, which is included in the last calculation. The model with all effects shows the lowest RMSE (0.86432).