1 executive summary

1.1) Introduction

This project aims to predict for each movie what is the rate using RMSE as a metric for evaluating the model.

Users were selected at random for inclusion. All users selected had rated at least 20 movies. Unlike previous MovieLens data sets, no demographic information is included. Each user is represented by an id, and no other information is provided.

1.2) goal

The goal is to train a machine learning algorithm using the inputs of a provided training subset to predict movie ratings in a validation set and score RMSE less than 0.86499

1.3) Dataset

In [24]:
```
edx = readRDS("../input/movielens/edx.rds")
test = readRDS("../input/movielens/validation.rds")
```

The MovieLens dataset can be downloaded from thins link https://grouplens.org/datasets/movielens/10m/ (https://grouplens.org/datasets/movielens/10m/) After some preprocessing we have 2 data sets 1) edx which is the main train data that consists of 6 columns and +9 milion row 2) test which is the data that we will validate our model performance using it, Consists of 6 columns and 999999 row

1.4) tools used

I used kaggle kernal as my machine resources is not enought to handle a big data set like this so all model fitting and experimentation was made on it as it gives me 4 cores and 16 gega RAM, how ever it's pretty much the same as the kinted rmd file and it will serve as my pdf report.

Loading the required libraries and data

In [1]:
```
library(tidyverse)
library(caret)
library(readr)
library(dplyr)
library(tidyr)
library(stringr)
library(ggplot2)
library(gridExtra)
library(data.table)
library(parallel)
library(doParallel)
fitControl <- trainControl(allowParallel = TRUE)
```

```
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──

✓ ggplot2 3.3.0.9000     ✓ purrr   0.3.4
✓ tibble  3.0.0          ✓ dplyr   0.8.5
✓ tidyr   1.0.2          ✓ stringr 1.4.0
✓ readr   1.3.1          ✓ forcats 0.5.0

── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()

Loading required package: lattice


Attaching package: ‘caret’


The following object is masked from ‘package:purrr’:

    lift


The following object is masked from ‘package:httr’:

    progress



Attaching package: ‘gridExtra’


The following object is masked from ‘package:dplyr’:

    combine



Attaching package: ‘data.table’


The following objects are masked from ‘package:dplyr’:

    between, first, last
```

```
The following object is masked from ‘package:purrr’:

    transpose

Loading required package: foreach

Attaching package: ‘foreach’

The following objects are masked from ‘package:purrr’:

    accumulate, when

Loading required package: iterators
```

2) methods/analysis

2.1) Methods

Here is a list with the steps i performed :

1. data cleaning and exploration
2. predicting with mean rating
3. predicting with user effect on the mean rating
4. predicting with user effect plus movie effect on the mean rating
5. predicting with user effect plus movie effect on the mean rating and regularizing the model with conctant lambda
6. trying linear regression and a decision tree
7. continue with linear regression and do feature enfineering
8. feature selection
9. comparing the results
10. conclusion

2.2) Data cleaning and understanding

In [25]:
```
head(edx)
```

A data.frame: 6 × 6

	userId	movieId	rating	timestamp	title	genres
	<int>	<dbl>	<dbl>	<int>	<chr>	<chr>
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

In [26]:
```
#checking of there is any null
anyNA(edx)
```

FALSE

In [27]:
```
str(edx) # takin a look at the data
```

```
'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 838984885 838983707
838984596 ...
 $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
 $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Adventure|
Sci-Fi" ...
```

so we have 6 (with 2 categorical)columns and 9+ milions row

1) UserID

In [28]:
```
n_distinct(edx$userId) # how many unique value do we have
```

69878

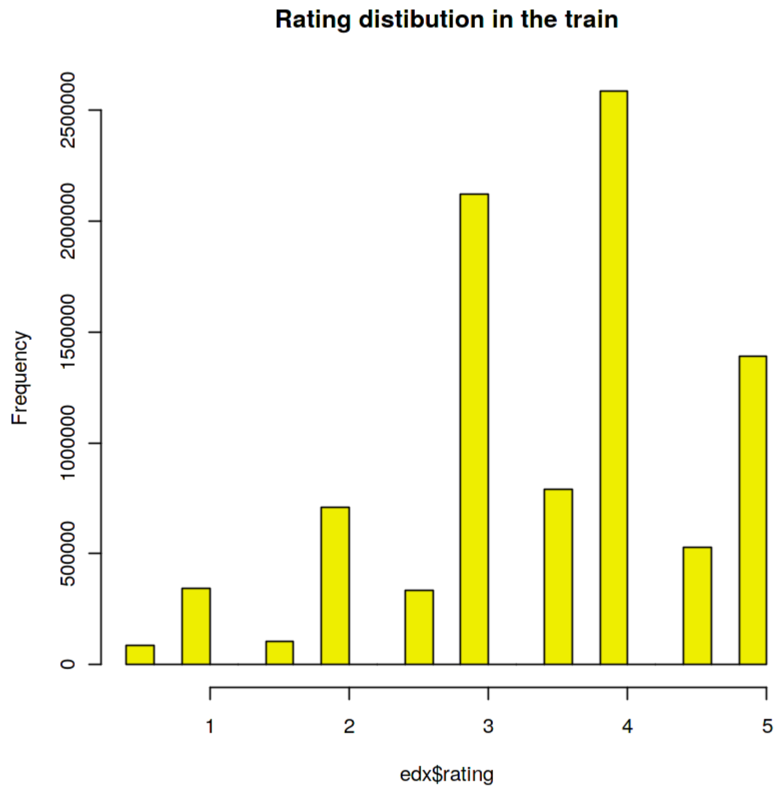
2) movieId

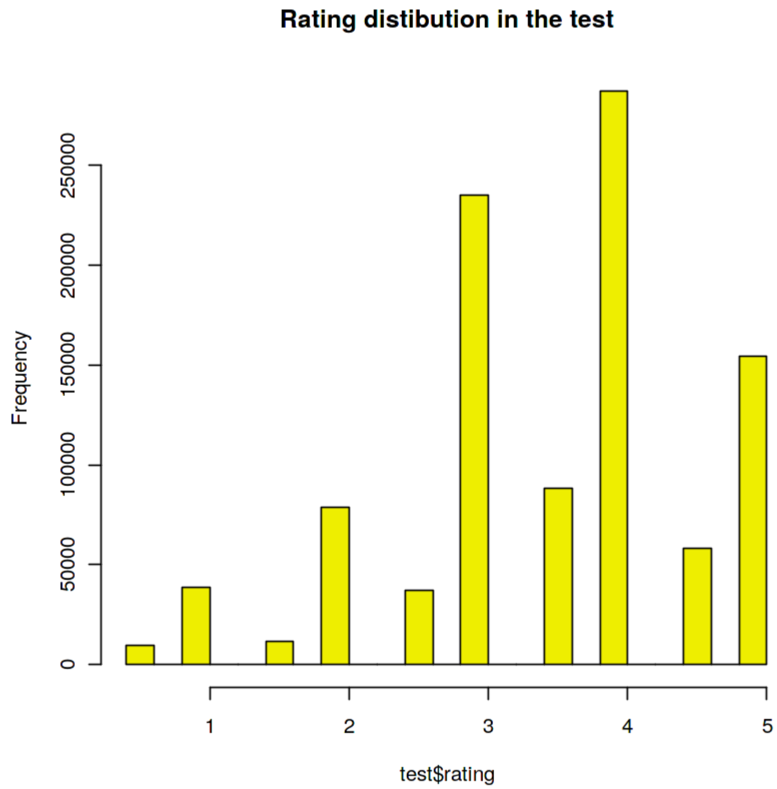
In [29]:
```
n_distinct(edx$movieId) #so how many movie do we have
```

10677

3) our target variable the rating

In [30]:
```
#Let's see the distribution of it and compare it with the test set
hist(edx$rating, col = 'yellow2',main = 'Rating distibution in the train') # plot the target in the train dat
a
hist(test$rating , col = 'yellow2',main = 'Rating distibution in the test') # plot the target in the test dat
a
```


Rating distibution in the train
Frequency
0
500000
1000000
1500000
2000000
2500000
1
2
3
4
5
edx$rating



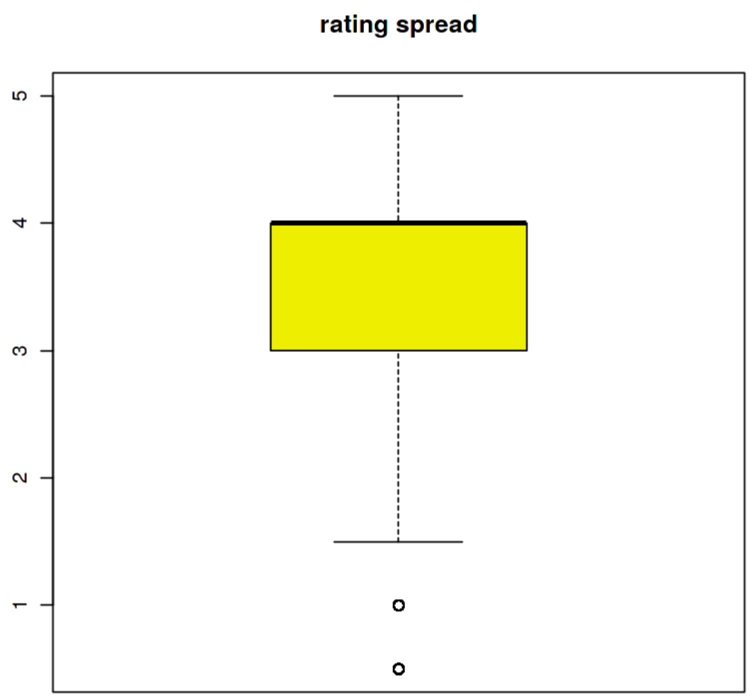
they are pretty much the same which is a good thing as the validation set that i will make will simulate the test set well

In [31]:
```
round(table(edx$rating)/nrow(edx) , 2) # what is the ratio of every ratnig group
```

```
 0.5    1  1.5    2  2.5    3  3.5    4  4.5    5 
0.01 0.04 0.01 0.08 0.04 0.24 0.09 0.29 0.06 0.15 
```

about 53% of the data for 3 and 4 ratnings which makes sens as there is not perfect or an extremly bad movie

In [32]:
```
boxplot(edx$rating ,main = 'rating spread', col = 'yellow2')
```



there are some outliers for movies rating below 1 let's how many value below 1

In [34]:
```
sum(edx$rating < 1) / nrow(edx)
```

0.00948594203035426

it's very small fraction of the data i may remove it and see as a way of eploring

4) timestamp (represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.)

i will convert it to regular date in order to extract useful information from it

In [36]:
```
as.POSIXct(min(edx$timestamp),origin = "1970-01-01",tz = "UTC")
as.POSIXct(max(edx$timestamp),origin = "1970-01-01",tz = "UTC")
```

```
[1] "1995-01-09 11:46:49 UTC"

[1] "2009-01-05 05:02:16 UTC"
```

so the first review was made in 1995 and the last one in 2009

5) title

In [37]:
```
head(edx)$title
```

'Boomerang (1992)' · 'Net, The (1995)' · 'Outbreak (1995)' · 'Stargate (1994)' · 'Star Trek: Generations (1994)' · 'Flintstones, The (1994)'

along with the movie name there is the year wbich the movie was released this can be seperated into 2 different features

6) genres

In [38]:
```
n_distinct(edx$genres) # how many unique genres do we have
```

797

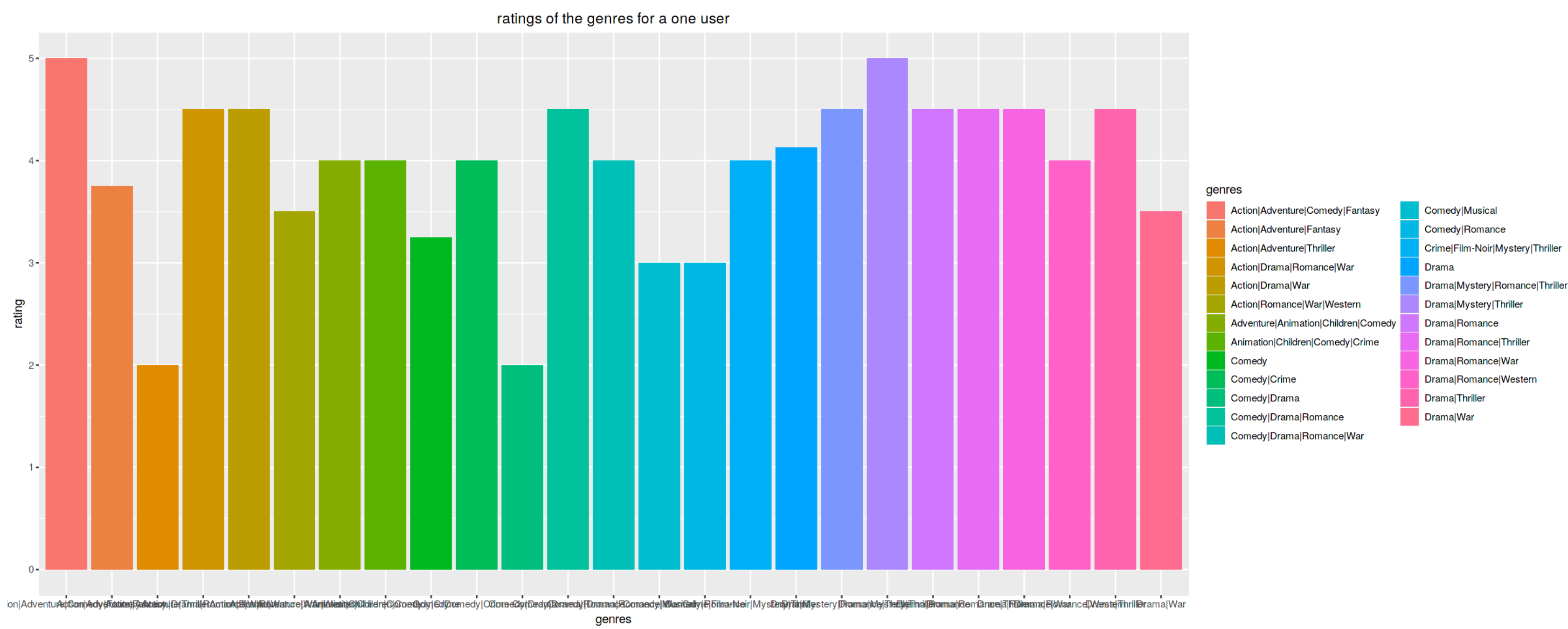
2.3) data exploration and visualization

since users are not a lot relatively with the data size the column will be important as surely users have perefrences to sepicifc geners, let's figure it out ...

In [39]:
```
user_gender_prefrences <- edx %>% filter(userId == 3) %>% group_by(userId,genres) %>% summarise(rating = mean
(rating)) %>% print ## this grouped data will be used in ggplot to represet the relation between the two vari
ables

options(repr.plot.width = 20, repr.plot.height = 8)
ggplot(data = user_gender_prefrences , aes(x = genres , y = rating , fill = genres)) +
  geom_col() +
  ggtitle("ratings of the genres for a one user") +
  theme(plot.title = element_text(hjust = 0.5))
```

```
# A tibble: 25 x 3
# Groups:   userId [1]
   userId genres                              rating
    <int> <chr>                                <dbl>
 1      3 Action|Adventure|Comedy|Fantasy       5
 2      3 Action|Adventure|Fantasy              3.75
 3      3 Action|Adventure|Thriller             2
 4      3 Action|Drama|Romance|War              4.5
 5      3 Action|Drama|War                      4.5
 6      3 Action|Romance|War|Western            3.5
 7      3 Adventure|Animation|Children|Comedy   4
 8      3 Animation|Children|Comedy|Crime       4
 9      3 Comedy                                3.25
10      3 Comedy|Crime                          4
# … with 15 more rows
```



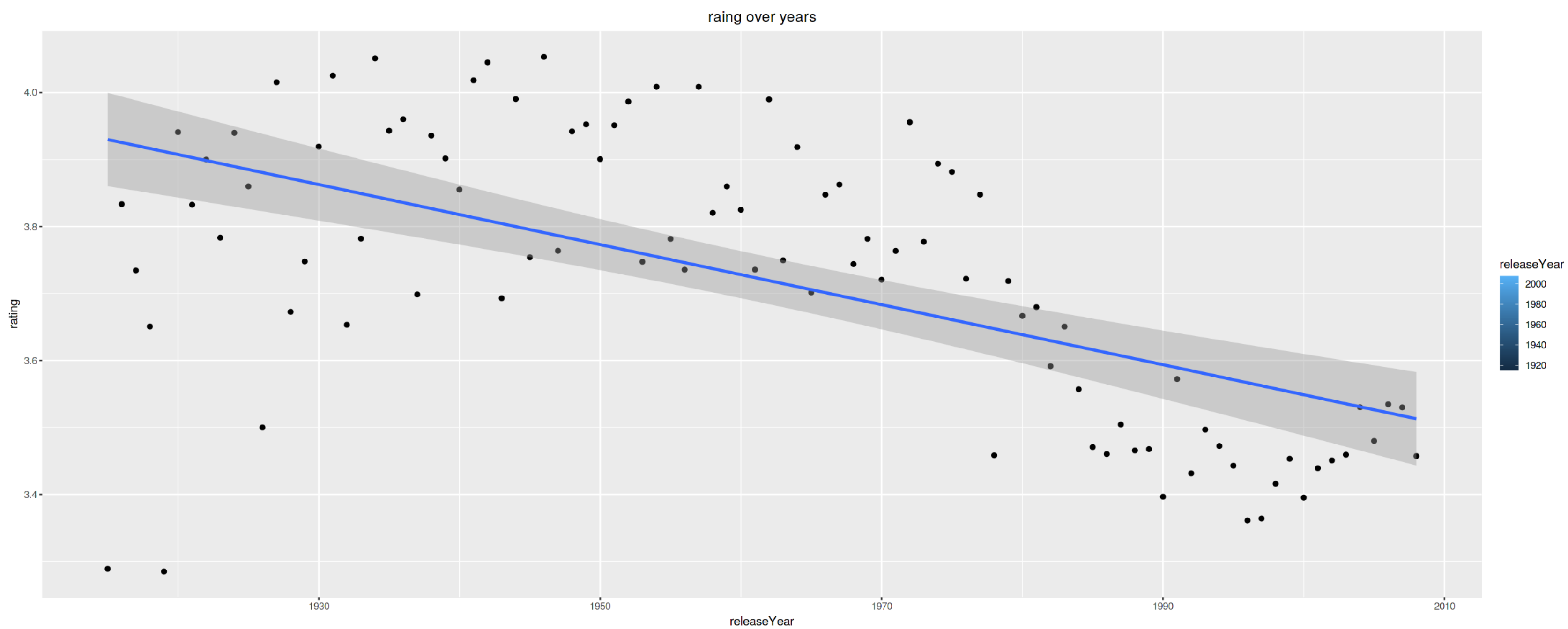
we can see the different prefrences of the user for the different genres

Is there a relationship between the release year and the movie rating ?

In [42]:
```
edx <- edx %>% mutate(releaseYear = as.numeric(str_sub(title,-5,-2))) # the year is from the second to the fi
fth place from the end

options(repr.plot.width = 20, repr.plot.height = 8)
edx %>% group_by(releaseYear) %>%
summarize(rating = mean(rating)) %>%
ggplot(aes(releaseYear, rating, fill = releaseYear)) +
 geom_point() + geom_smooth(method = "lm" ) +
  ggtitle("raing over years") +
   theme(plot.title = element_text(hjust = 0.5))
```

```
`geom_smooth()` using formula 'y ~ x'
```



we can see the decreasing trend over years which is interesting that people tend to give rating less and less over time

2.4) Start with baseline predictors and go more complex

splitting the train data into train and validation data as trying many models on the test set will leads to overfitting and not a good practice in the real life

In [43]:
```
set.seed(1)
val_index <- createDataPartition(edx$rating, p = 0.9, list=FALSE ) # taking 10% for validation to be similar
 to the test set
temp <- edx[-val_index,]
edx <- edx[val_index,]

print(nrow(edx))

val <- 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, val)
edx <- rbind(edx, removed)

print(nrow(edx))
```

```
[1] 8100050

Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres", "releaseYear")


[1] 8100064
```

In [44]:
```
str(val) # this data set will be for experimentation
```

```
'data.frame':	899991 obs. of  7 variables:
 $ userId     : int  1 1 2 3 4 4 5 5 5 5 ...
 $ movieId    : num  420 466 1073 1288 110 ...
 $ rating     : num  5 5 3 3 5 5 3 5 3 3 ...
 $ timestamp  : int  838983834 838984679 868244562 1133571035 844416866 844416834 857912840 857912593 8579125
35 857912492 ...
 $ title      : chr  "Beverly Hills Cop III (1994)" "Hot Shots! Part Deux (1993)" "Willy Wonka & the Chocolat
e Factory (1971)" "This Is Spinal Tap (1984)" ...
 $ genres     : chr  "Action|Comedy|Crime|Thriller" "Action|Comedy|War" "Children|Comedy|Fantasy|Musical" "Co
medy|Musical" ...
 $ releaseYear: num  1994 1993 1971 1984 1995 ...
```

In [2]:
```
# making the evaluation function
RMSE <- function(actual , prediction){
sqrt(mean((actual - prediction)^2))
}
```

I will start with just predicting the global mean of all the movies

In [ ]:
```
mean_rating = mean(edx$rating)
print(c('The RMSE of global mean is : ', RMSE(val$rating, mean_rating)))
```

"The RMSE of global mean is : 1.06047991069723

the RMSE can be enhanced by adding aspects about different baises, take for example : Suppose Alice rates Inception 4 stars. We can think of this rating as composed of several parts:

1) A baseline rating (e.g., maybe the mean over all user-movie ratings is 3.1 stars). `what i have tried until now`

2) An Alice-specific effect (e.g., maybe Alice tends to rate movies lower than the average user, so her ratings are -0.5 stars lower than we normally expect).

3) An Inception-specific effect (e.g., Inception is a pretty awesome movie, so its ratings are 0.7 stars higher than we normally expect).

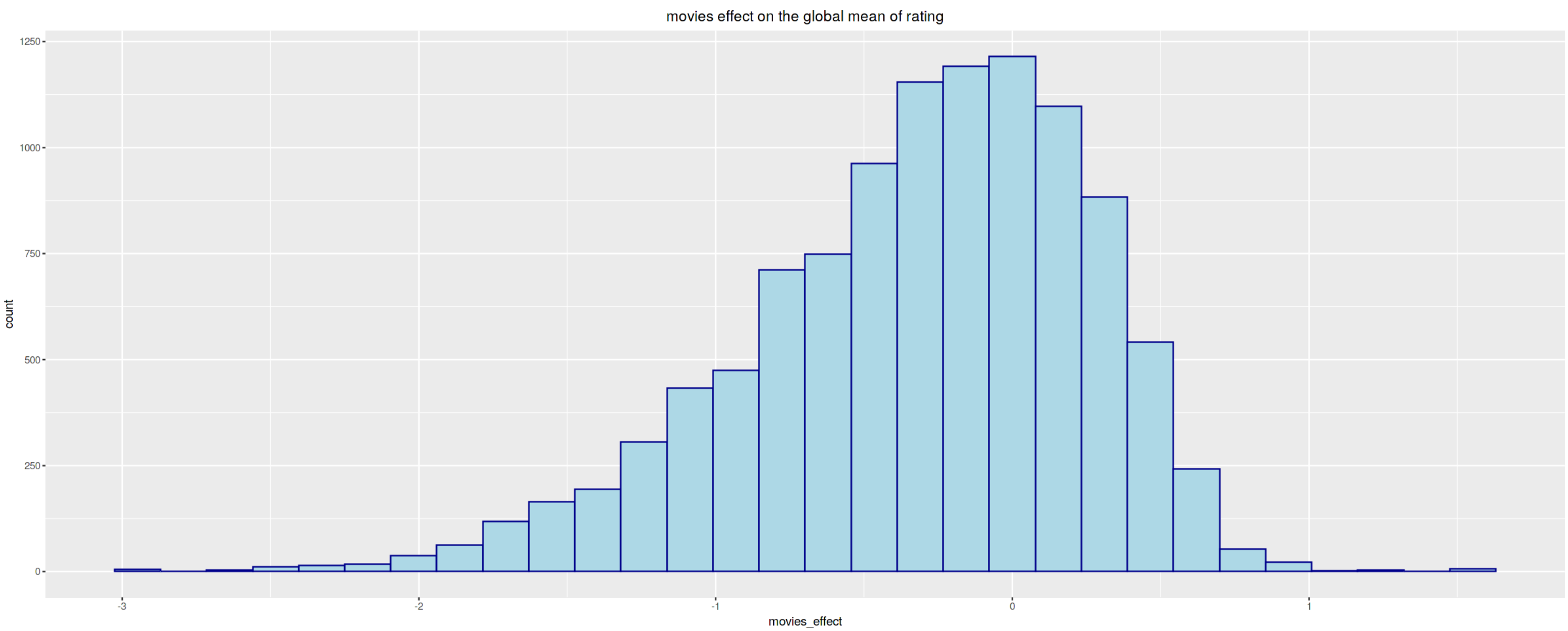
4) A less predictable effect based on the specific interaction between Alice and Inception that accounts for the remainder of the stars (e.g., Alice really liked Inception because of its particular combination of Leonardo DiCaprio and neuroscience, so this rating gets an additional 0.7 stars).

```
Let's experiment and see
```

surely there are movies that have higher rating than the average becaue thery are really better, over rated and so on. they also may be less than the average as they are not well made

In [45]:
```
options(repr.plot.width = 20, repr.plot.height = 8)
movies_average_ratings <- edx %>%
group_by(movieId) %>%
summarize(movies_effect = mean(rating - mean_rating)) # how the movie average rating affecting the global mea
n ?
 ggplot(data = movies_average_ratings , aes(x = movies_effect)  ) + geom_histogram(color="darkblue", fill="li
ghtblue") +
  ggtitle("movies effect on the global mean of rating") +
   theme(plot.title = element_text(hjust = 0.5))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



there are some movies that adds very high effect that can deacrese the mean rating by -3 so this will catching the real life behavior

In [46]:
```
# so for each movie add the movie effect to the global mean so that we capture the movie bais
movie_effect <- val %>% left_join(movies_average_ratings, by = 'movieId') %>% mutate(prediction = mean_rating
+ movies_effect)
print(c('RMSE after adding the movie effect is: ', RMSE(val$rating , movie_effect$prediction)))
```

```
[1] "RMSE after adding the movie effect is: "
[2] "0.944219268618815"
```

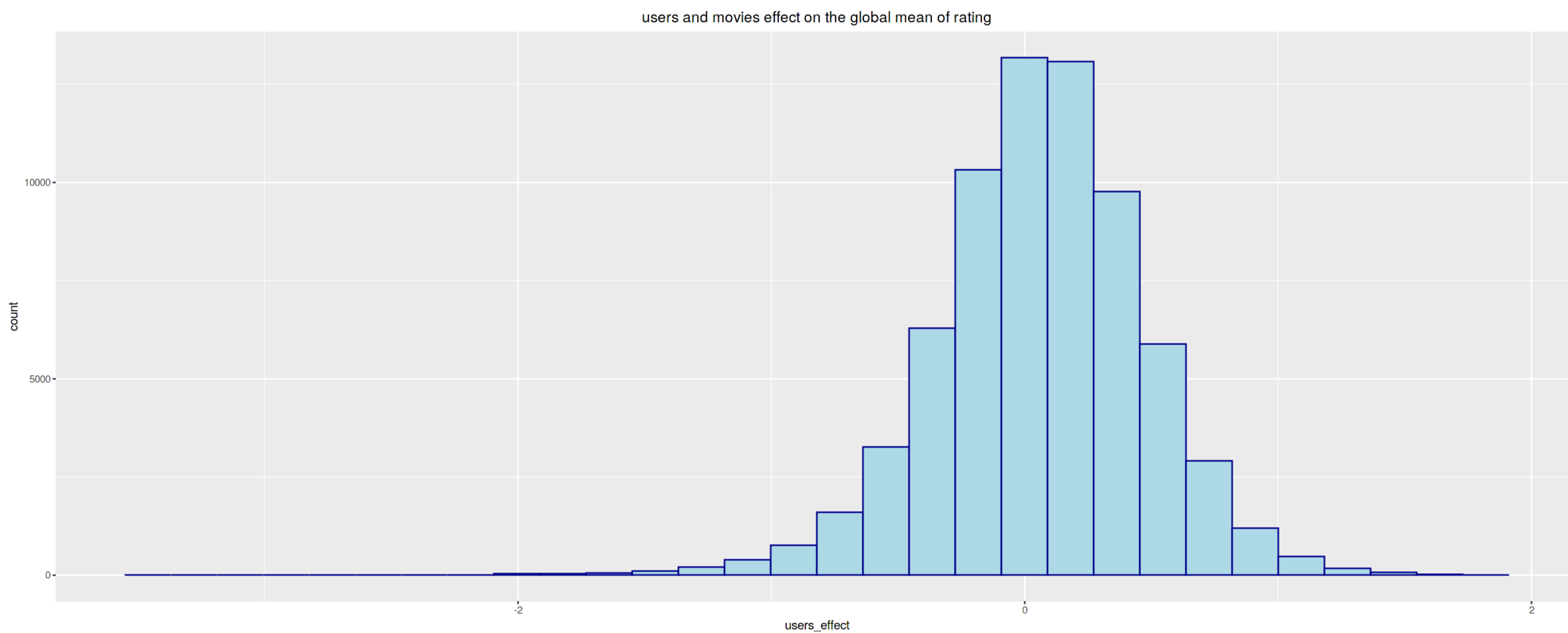
we can see the improvment `0.943782817189865`

adding the user effect for each user personal bais and prefrences

In [47]:
```
options(repr.plot.width = 20, repr.plot.height = 8)

users_average_ratings <- edx %>%
left_join(movies_average_ratings, by='movieId') %>%
group_by(userId) %>%
summarize(users_effect = mean(rating - mean_rating - movies_effect)) # so we seeing here how the effect of bo
th the user and movie on the global mean
ggplot(data = users_average_ratings , aes(x = users_effect )  ) + geom_histogram(color="darkblue", fill="ligh
tblue")+
  ggtitle("users and movies effect on the global mean of rating") +
   theme(plot.title = element_text(hjust = 0.5))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



In [48]:
```
movie_user_effect <- val %>%
left_join(movies_average_ratings, by = 'movieId') %>%
left_join(users_average_ratings, by = 'userId') %>% # at this point we have the user and movie effect columns
mutate(prediction = mean_rating + movies_effect + users_effect) # add both columns to the global mean

print(c('RMSE after adding the user-movie effect is: ', RMSE(val$rating , movie_user_effect$prediction)))
```

```
[1] "RMSE after adding the user-movie effect is: "
[2] "0.866028752146733"
```

we got better results with RMSE : `0.865845634101688`

In [49]:
```
users_reviews_count <- edx %>% group_by(userId) %>% summarise(count = n()) # how much movie reviews every use
r has made
summary(users_reviews_count$count)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    9.0    29.0    56.0   115.9   127.0  5955.0 
```

In [50]:
```
movies_reviews_count <- edx %>% group_by(movieId) %>% summarise(count = n()) # how much movie reviews every u
ser has made
summary(movies_reviews_count$count)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    1.0    27.0   110.0   758.6   508.0 28338.0 
```

we can see that there users that made little number of reviews and some made huge number and the same applies for movies there are for example a movie that is rated once so this movie effect will not be robsut, hence we need to do regularization so that the effect is relative to the number of reviews made.

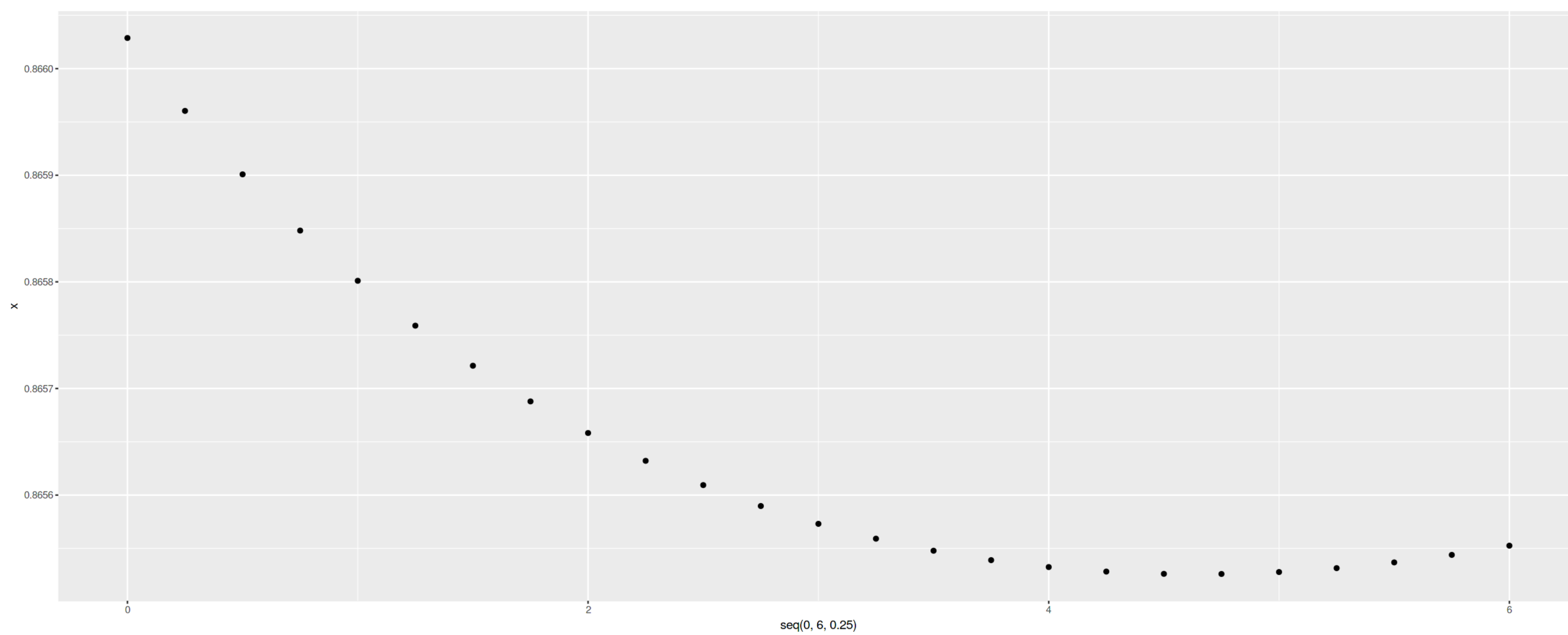
In [51]:
```
x <- c()
for (lambda in seq(0,6,0.25))  {
    movies_average_ratings_regu <- edx %>% group_by(movieId) %>% summarise(movies_effect = sum(rating - mean_rating)/(n()+lambda))
    users_average_ratings_regu <- edx %>% left_join(movies_average_ratings_regu, by='movieId') %>% group_by(userId) %>%
    summarize(users_effect = sum(rating - mean_rating - movies_effect)/(n()+lambda))

    movie_user_effect_regu <- val %>%
    left_join(movies_average_ratings_regu, by = 'movieId') %>%
    left_join(users_average_ratings_regu, by = 'userId') %>% # at this point we have the user and movie effect columns
    mutate(prediction = mean_rating + movies_effect + users_effect) # add both columns to the global mean

    x <- c(x,RMSE(val$rating , movie_user_effect_regu$prediction))

        }
```

In [52]:
```
options(repr.plot.width = 20, repr.plot.height = 8)
qplot(seq(0,6,0.25),x)
```



the RMSE improved slightly with the best RMSE `0.865525994109226` and lambda = `5.25`

so the regularized model is the best one of them i will try it on the test data and use all the train set

In [3]:
```
edx_full = readRDS("../input/movielens/edx.rds")
test = readRDS("../input/movielens/validation.rds")
```

In [ ]:
```
lambda = 5.25
movies_average_ratings_regu <- edx_full %>% group_by(movieId) %>% summarise(movies_effect = sum(rating - mean
_rating)/(n()+lambda))
users_average_ratings_regu <- edx_full %>% left_join(movies_average_ratings_regu, by='movieId') %>% group_by(
userId) %>%
summarize(users_effect = sum(rating - mean_rating - movies_effect)/(n()+lambda))

movie_user_effect_regu <- test %>%
left_join(movies_average_ratings_regu, by = 'movieId') %>%
left_join(users_average_ratings_regu, by = 'userId') %>% # at this point we have the user and movie effect co
lumns
mutate(prediction = mean_rating + movies_effect + users_effect) # add both columns to the global mean

print(c('RMSE after adding the user-movie effect is: ', RMSE(test$rating , movie_user_effect_regu$prediction
)))
```

"RMSE after adding the user-movie effect is: `0.864816994393969` which is the best until now

since the data is really big and i will run many experiments i will work on a subset of the data and fit on the full data set when i'm happy with the feature engineering and model

In [5]:
```
set.seed(1)
print(nrow(edx_full))
val_index <- createDataPartition(edx_full$rating, p = 0.9, list=FALSE) # taking 10% for validation to be simi
lar to the test set
temp <- edx_full[-val_index,] #take 0.1
edx <- edx_full[val_index,]# take 0.9

print(nrow(edx))

val <- 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, val)
edx <- rbind(edx, removed)

print(nrow(edx))
```

```
[1] 9000055
[1] 8100050

Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")


[1] 8100064
```

In [53]:
```
as.numeric(factor(c('a','b','a'))) # how to convert from charachter to numerical
```

1. 2. 1

In [6]:
```
# converting title and genres with label encoding
edx['title_encoded'] = as.numeric(factor(edx$title))
val['title_encoded'] = as.numeric(factor(val$title))
test['title_encoded'] = as.numeric(factor(test$title))

edx['genres_encoded'] = as.numeric(factor(edx$genres))
val['genres_encoded'] = as.numeric(factor(val$genres))
test['genres_encoded'] = as.numeric(factor(test$genres))

head(edx)
```

A data.frame: 6 × 8

	userId	movieId	rating	timestamp	title	genres	title_encoded	genres_encoded
	<int>	<dbl>	<dbl>	<int>	<chr>	<chr>	<dbl>	<dbl>
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance	1308	577
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller	6757	187
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	7179	210
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	8997	98
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	8980	71
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	3419	460

In [7]:
```
edx <- edx %>%
mutate(releaseYear = as.numeric(str_sub(title,-5,-2))) # the year is from the second to the fifth place from
 the end

val <- val %>%
mutate(releaseYear = as.numeric(str_sub(title,-5,-2))) # applying the same to the test

test <- test %>%
mutate(releaseYear = as.numeric(str_sub(title,-5,-2))) # applying the same to the test
```

In [8]:
```
edx = subset(edx, select = -c(title,genres) ) # we don't need them any more
val = subset(val, select = -c(title,genres) )
test = subset(test, select = -c(title,genres) )
```

i will start with some models and see the RMSE on the validation set and then determine which model to continue with

linear model

In [ ]:
```
set.seed(1) # fixing the seed in order to ensure reproducibility
lm_fit1 <- lm(rating ~ . , data = edx )
prediction = predict(lm_fit1, val)
print(c('The Linear model RMSE :', RMSE(prediction, val$rating)))
```

the RMS is `1.06103817963191` which is almost the same as using just the mean so we need to deacrease the RMSE

trying the decision tree

In [ ]:
```
set.seed(1)
fitControl <- trainControl(allowParallel = TRUE)
tree = train(rating ~ .,method="rpart",data= edx,trControl = fitControl )
prediction = predict(tree, test)
saveRDS(tree, "./tree.rds") # saving the model
print(c('The tree model RMSE :',' ', RMSE(prediction, test$rating)))
```

the decision tree is taking so much time and eventually the ram is full and crash so i had to continue with the linear regression

2.5) Feature engineering

In [60]:
```
as.integer(substr("1970-01-01",0,4) , 16) # how to convert the year to number
```

1970

In [9]:
```
#converting the time stamp to date and then extract the year and month from it
edx['date'] = as.POSIXct(edx$timestamp,origin = "1970-01-01",tz = "UTC")
val['date'] = as.POSIXct(val$timestamp,origin = "1970-01-01",tz = "UTC")
test['date'] = as.POSIXct(test$timestamp,origin = "1970-01-01",tz = "UTC")

#this is the review year
edx['year'] = as.integer(substr(edx$date,0,4),16)
val['year'] = as.integer(substr(val$date,0,4),16)
test['year'] = as.integer(substr(test$date,0,4),16)

#this is the review month
edx['month'] = as.integer(substr(edx$date,6,7),16)
val['month'] = as.integer(substr(val$date,6,7),16)
test['month'] = as.integer(substr(test$date,6,7),16)
```

In [61]:
```
anyNA(test)
```

FALSE

adding users and movies effect together regularized by lambda = 5.25

In [10]:
```
lambda = 5.25
movies_average_ratings_regu <- edx %>% group_by(movieId) %>% summarise(movies_effect = sum(rating - mean_rati
ng)/(n()+lambda))
users_average_ratings_regu <- edx %>% left_join(movies_average_ratings_regu, by='movieId') %>% group_by(userI
d) %>%
summarize(users_effect = sum(rating - mean_rating - movies_effect)/(n()+lambda))

edx <- edx %>%
left_join(movies_average_ratings_regu, by = 'movieId') %>%
left_join(users_average_ratings_regu, by = 'userId') %>% # at this point we have the user and movie effect co
lumns
mutate(prediction = mean_rating + movies_effect + users_effect) # add both columns to the global mean

val <- val %>%
left_join(movies_average_ratings_regu, by = 'movieId') %>%
left_join(users_average_ratings_regu, by = 'userId') %>% # at this point we have the user and movie effect co
lumns
mutate(prediction = mean_rating + movies_effect + users_effect) # add both columns to the global mean

test <- test %>%
left_join(movies_average_ratings_regu, by = 'movieId') %>%
left_join(users_average_ratings_regu, by = 'userId') %>% # at this point we have the user and movie effect co
lumns
mutate(prediction = mean_rating + movies_effect + users_effect) # add both columns to the global mean
```

notice that there is no any kind of leakage as i used the edx only to calculate the movies and users effect hence i didn't use the target variable in the test or val

let's see the effect of the added columns

In [11]:
```
set.seed(1) # fixing the seed in order to ensure reproducibility
lm_fit2 <- lm(rating ~ . , data = edx )
prediction = predict(lm_fit2, val)
print(c('The Linear model RMSE :', RMSE(prediction, val$rating)))
```

```
Warning message in predict.lm(lm_fit2, val):
“prediction from a rank-deficient fit may be misleading”
```

```
[1] "The Linear model RMSE :" "0.865458527371831"
```

The Linear model RMSE :" `0.880691567230391` , there is a good decrese in the RMSE we are on the right path

i will replace the title and genres with the mean encoding

`motivation` : the genres and title are not ordinal so label encoding them will make a fake relation and if i one hot encoded them we will fall in the curse of dimensionality hence mean encoding is needed to solve this trap, `it's basiclly for each value in the column what is the average rating that is associated with it`

In [ ]:
```
# mean encode the genres
edx_genres_mean_encoded = edx %>% group_by(genres_encoded) %>% summarise(genres_mean_encoded = mean(rating))

edx <- edx %>% left_join(edx_genres_mean_encoded , by = 'genres_encoded')
val <- val %>% left_join(edx_genres_mean_encoded , by = 'genres_encoded')
test <- test %>% left_join(edx_genres_mean_encoded , by = 'genres_encoded')
```

In [ ]:
```
set.seed(1) # fixing the seed in order to ensure reproducibility
lm_fit3 <- lm(rating ~ . , data = edx )
prediction = predict(lm_fit3, val)
print(c('The Linear model RMSE :', RMSE(prediction, val$rating)))
```

[1] "The Linear model RMSE :" `0.881114245860386`

the RSME increased so i will remove it

In [ ]:
```
edx$genres_mean_encoded <- NULL
val$genres_mean_encoded <- NULL
test$genres_mean_encoded <- NULL
```

In [ ]:
```
# mean encode the title
edx_title_mean_encoded = edx %>% group_by(title_encoded) %>% summarise(title_mean_encoded = mean(rating))

edx <- edx %>% left_join(edx_title_mean_encoded , by = 'title_encoded')
val <- val %>% left_join(edx_title_mean_encoded , by = 'title_encoded')
test <- test %>% left_join(edx_title_mean_encoded , by = 'title_encoded')
```

In [ ]:
```
set.seed(1) # fixing the seed in order to ensure reproducibility
lm_fit4 <- lm(rating ~ . , data = edx )
prediction = predict(lm_fit4, val)
print(c('The Linear model RMSE :', RMSE(prediction, val$rating)))
```

[1] "The Linear model RMSE :" `0.982785248589716`

RMSE increased so i will remove it

In [ ]:
```
edx$title_mean_encoded <- NULL
val$title_mean_encoded <- NULL
test$title_mean_encoded <- NULL
```

for each movie for every review i want to calculate the time since first review was made for it as we saw that with the time movies have less rate

In [12]:
```
val <- val %>% group_by(movieId) %>% mutate(first_movie_review_time =  min(date) , time_since_first_review =
as.integer(date - first_movie_review_time))

edx <- edx %>% group_by(movieId) %>% mutate(first_movie_review_time =  min(date) , time_since_first_review =
as.integer(date - first_movie_review_time))

test <- test %>% group_by(movieId) %>% mutate(first_movie_review_time =  min(date) , time_since_first_review
= as.integer(date - first_movie_review_time))
```

In [13]:
```
set.seed(1) # fixing the seed in order to ensure reproducibility
lm_fit5 <- lm(rating ~ . , data = edx )
prediction = predict(lm_fit5, val)
print(c('The Linear model RMSE :', RMSE(prediction, val$rating)))
```

```
Warning message in predict.lm(lm_fit5, val):
“prediction from a rank-deficient fit may be misleading”
```

```
[1] "The Linear model RMSE :" "0.865035565715849"
```

it worked well and now the RMSE is `0.864966593045278`

In [ ]:
```
# what is the average rating for  every movie
movies_average = edx %>% group_by(movieId) %>% summarise(movie_mean_rating = mean(rating))
edx = edx %>% left_join(movies_average , by = 'movieId')
val = val %>% left_join(movies_average , by = 'movieId')
```

In [ ]:
```
set.seed(1) # fixing the seed in order to ensure reproducibility
lm_fit6 <- lm(rating ~ . , data = edx )
prediction = predict(lm_fit6, val)
print(c('The Linear model RMSE :', RMSE(prediction, val$rating)))
```

it didn't work well

In [ ]:
```
edx$movie_mean_rating = NULL
val$movie_mean_rating = NULL
```

Finally apply this model on the best model `lm_fit5` on test set and see

In [14]:
```
prediction = predict(lm_fit5, test)
print(c('The Linear model RMSE :', RMSE(prediction, test$rating)))
```

```
Warning message in predict.lm(lm_fit5, test):
“prediction from a rank-deficient fit may be misleading”

[1] "The Linear model RMSE :" "0.864722440553649"
```

The Linear model RMSE : `0.864722440553649` the requested RMSE achived.

2.6) Feature selection

i will try to reduce the number of variables to avoid the overfitting i tried to use step() function to automate the process but the ram was crashing so i will try it manually

In [17]:
```
colnames(test)
colnames(edx)
```

'userId' · 'movieId' · 'rating' · 'genres_encoded' · 'releaseYear' · 'year' · 'month' · 'movies_effect' · 'users_effect' · 'prediction' · 'first_movie_review_time' · 'time_since_first_review'

'userId' · 'movieId' · 'rating' · 'genres_encoded' · 'releaseYear' · 'year' · 'month' · 'movies_effect' · 'users_effect' · 'prediction' · 'first_movie_review_time' · 'time_since_first_review'

In [16]:
```
# removing date, timestamp and title_encoded as they are all redundent

edx$title_encoded <- NULL
edx$date <- NULL
edx$timestamp <- NULL

test$title_encoded <- NULL
test$date <- NULL
test$timestamp <- NULL
```

In [18]:
```
set.seed(1) # fixing the seed in order to ensure reproducibility
lm_fit7 <- lm(rating ~ . , data = edx )
prediction = predict(lm_fit7, test)
print(c('The Linear model RMSE :', RMSE(prediction, test$rating)))
```

```
Warning message in predict.lm(lm_fit7, test):
“prediction from a rank-deficient fit may be misleading”

[1] "The Linear model RMSE :" "0.864721770789204"
```

"The Linear model RMSE :" `0.864721770789204` , so the perforamnce did'nt change with fewer variables and RMSE less than `0.86490` achieved

3) Results

mean rating model `1.06047991069723`

user effect model `0.865845634101688`

movie-user effect model `0.865845634101688`

regularized user-movie effect model `0.864816994393969`

linear regression `0.864721770789204`

4) Conclusion

The linear regression model is the best model but the regularized user-movie effect model achieved similar RMSE with much more effeciency. The only limitation in this project was the resources my machine didn't run any model so i used kaggle kernal which allow me to only use the simple linear regression model using lm(), i believe that if i could use more complex models like xgboost i would scored better accuracy but any way with a careful feature engineering the linear regression model achived the required RMSE.