MovieLens Report

Masayoshi Sato

2021/5/3

Introduction

Nowadays, recommendation systems have become common to us. Products and services are rated by consumers using stars and points, then the ratings accumulate into dataset. These ratings are considered to be precious; they are not only helpful for new customers to choose goods or services, but also valuable for makers and service providers in terms of predicting consumer behaviors. If you explore a user ratings in detail, you will find his/her preference, recommend things which are most likely to be bought.They enable a targeted marketing.

One of the field in which this system is used is movie industries. In this paper, I will take a movie recommendation dataset, and build a model which will predict rating as accurately as possible.

Dataset The data used in this report, MovieLens 10M Dataset, is available at the Grouplens website. The site says this data set has 10000054 ratings and 95580 tags applied to 10681 movies by 71567 users of the online movie recommender service MovieLens.

MovieLens 10M dataset https://grouplens.org/datasets/movielens/10m/

Goal The goal is to train a machine learning algorithm to predict movie ratings based on factors in the provided dataset. To achieve this, the downloaded data is split into two, edx dataset (90%) and validation dataset (10%). The validation dataset is left to final evaluation.

As GroupLens web site explains, it has several factors, such as user ID, movie title, genres, rating as well. The assumption is that they are inter-correlated to some extent and helpful to forecast ratings. The challenge of a recommendation system, however, is that the data is sparse and each factor in the dataset might affect others. Namely, some users rate movies more than others, and some movies are rated more. Genres, released year may also affect ratings.

Therefore, a linear regression will be used to find weights that minimize the residual mean squared error, the RMSE. Predictors are user ID, movie ID, genres. and released year. In addition to this, regulation is used to penalize large estimates that are produced using small sample sizes.

RMSE is denoted as follows;

Exploratory analysis

First, I would like to have a little glance of the edx dataset.

```
##    userId movieId rating timestamp                          title
## 1:      1     122      5 838985046               Boomerang (1992)
## 2:      1     185      5 838983525                Net, The (1995)
## 3:      1     292      5 838983421              Outbreak (1995)
## 4:      1     316      5 838983392              Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1:                 Comedy|Romance
## 2:          Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:        Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:        Children|Comedy|Fantasy
```

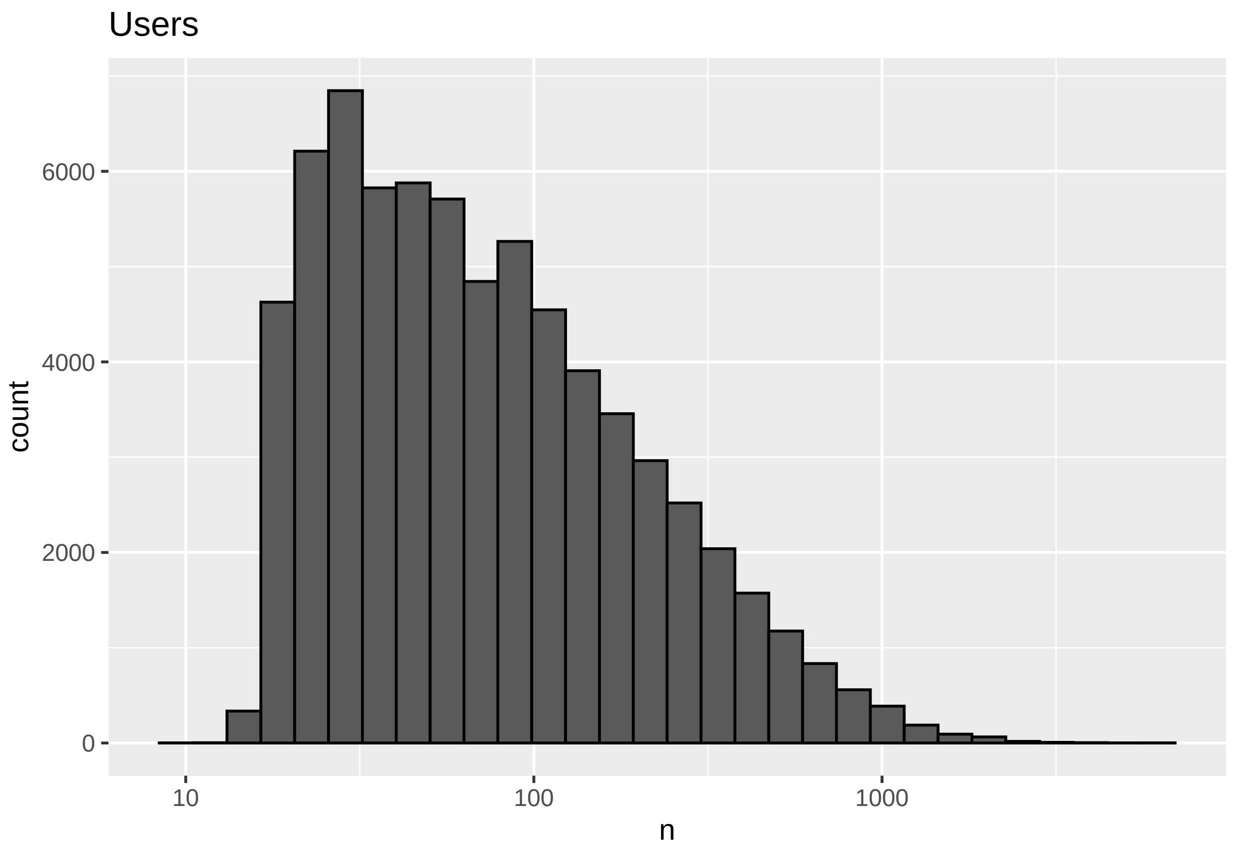
```
range(edx$rating)
```

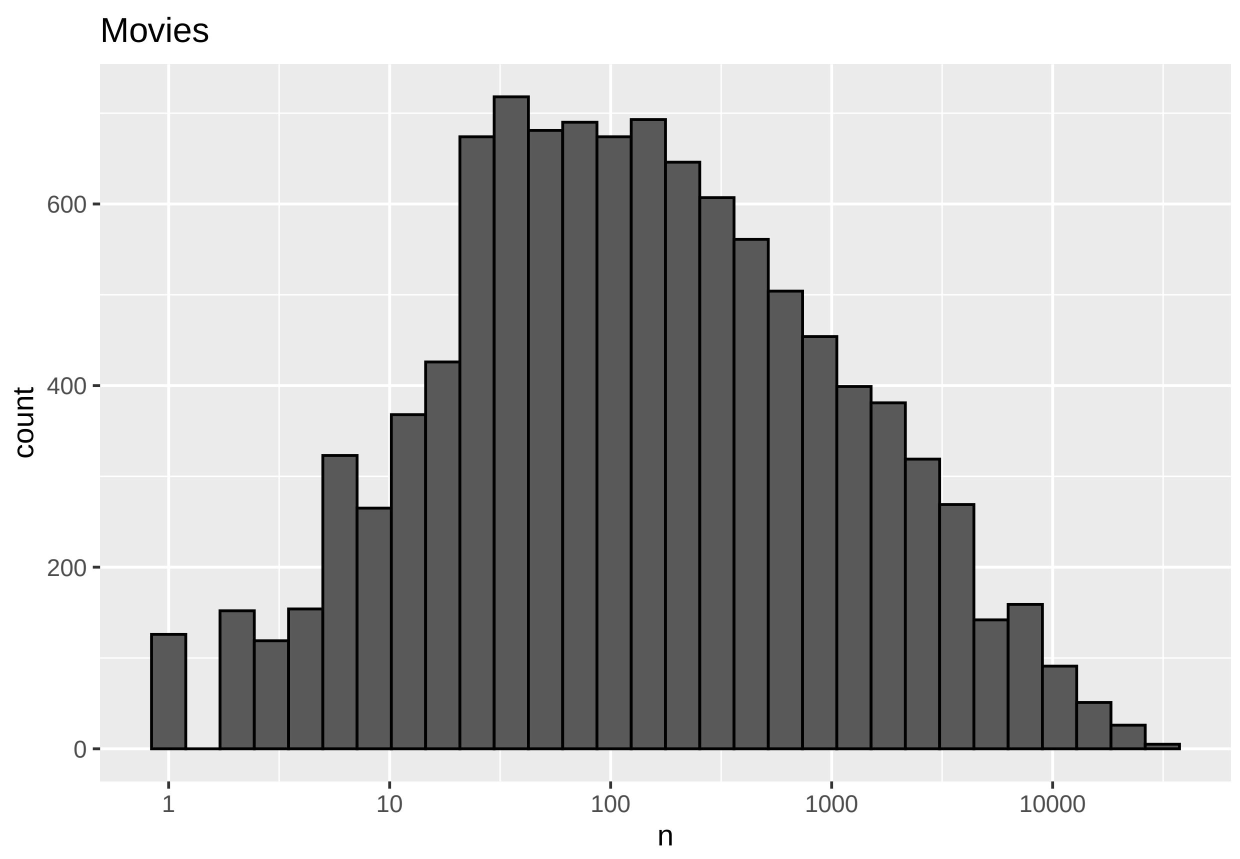
```
## [1] 0.5 5.0
```

The edx dataset has six columns, and 9000055 rows.

```
## [1] 9000055       6
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```



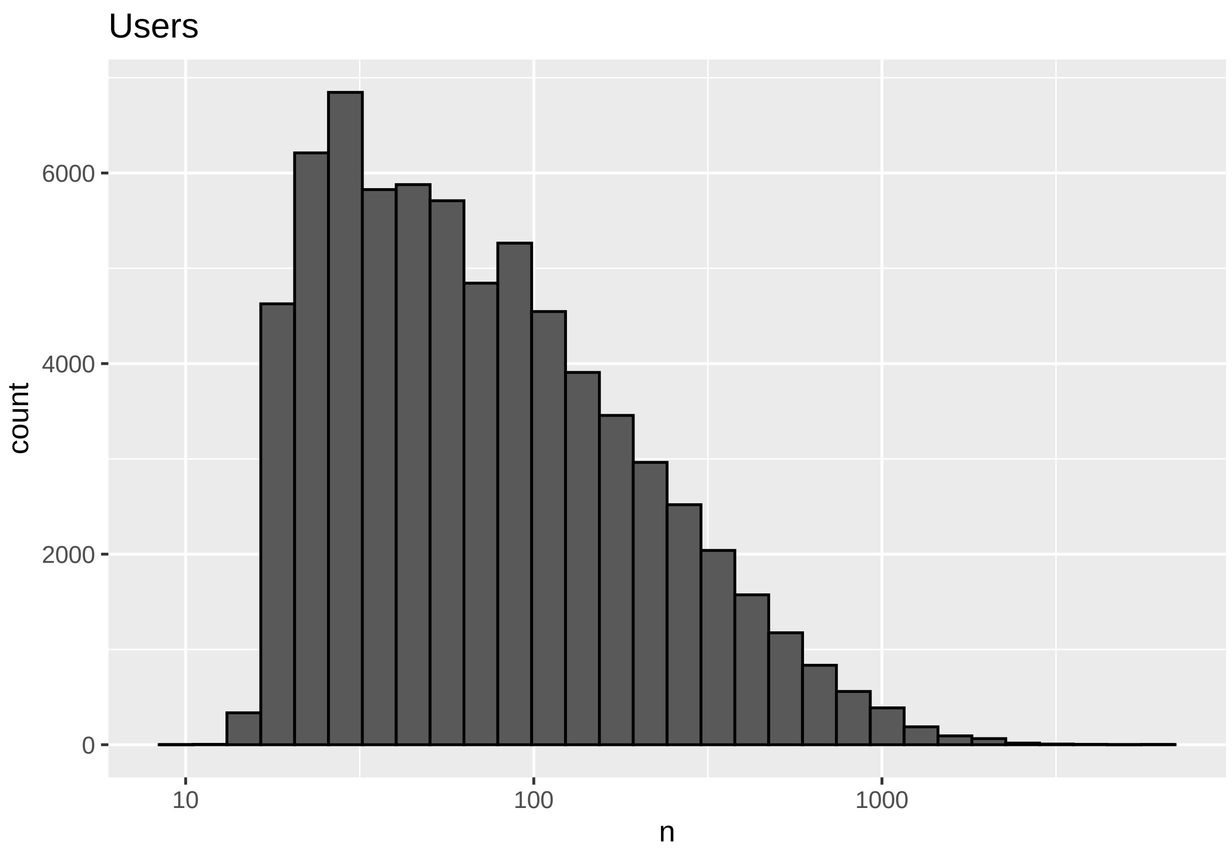


To find unique number, I use n_distinct().

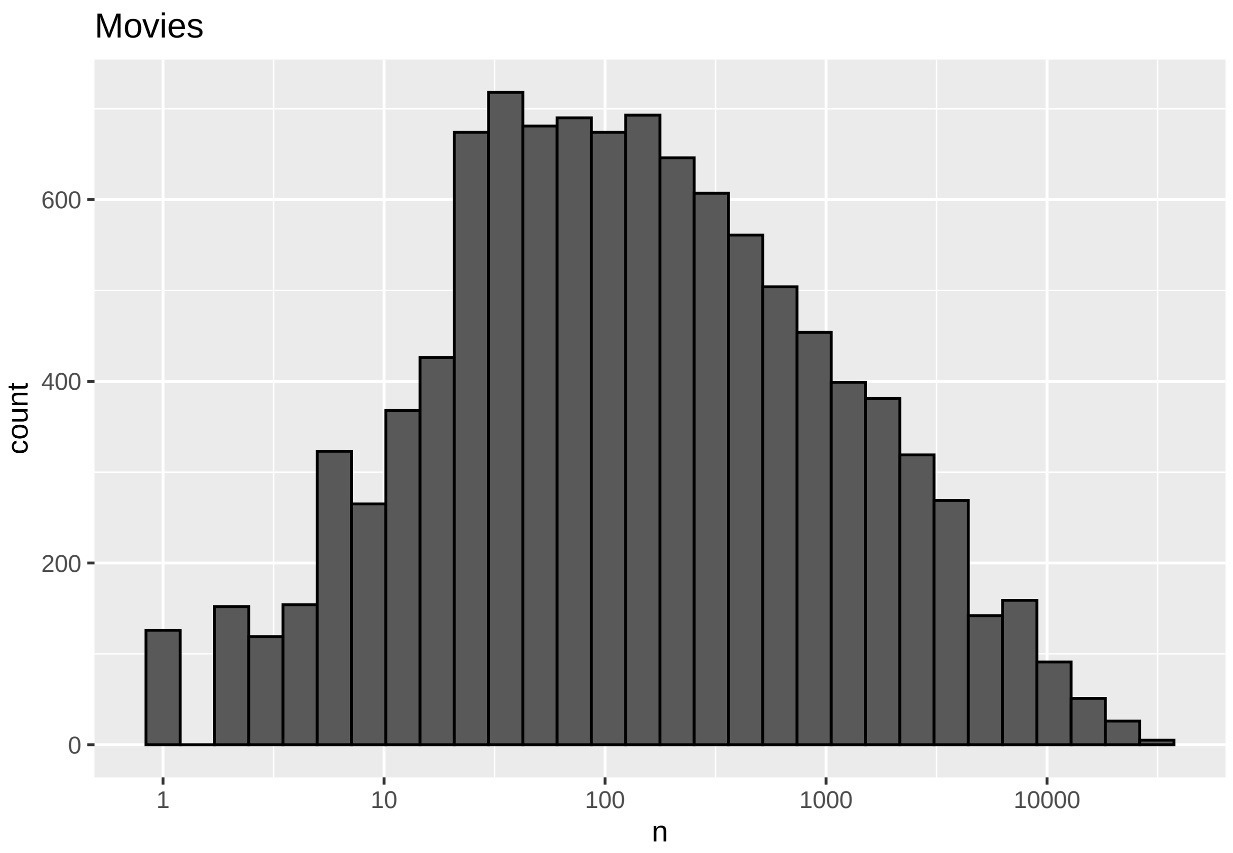
```
edx %>% summarize(n_users =n_distinct(userId), n_movies= n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

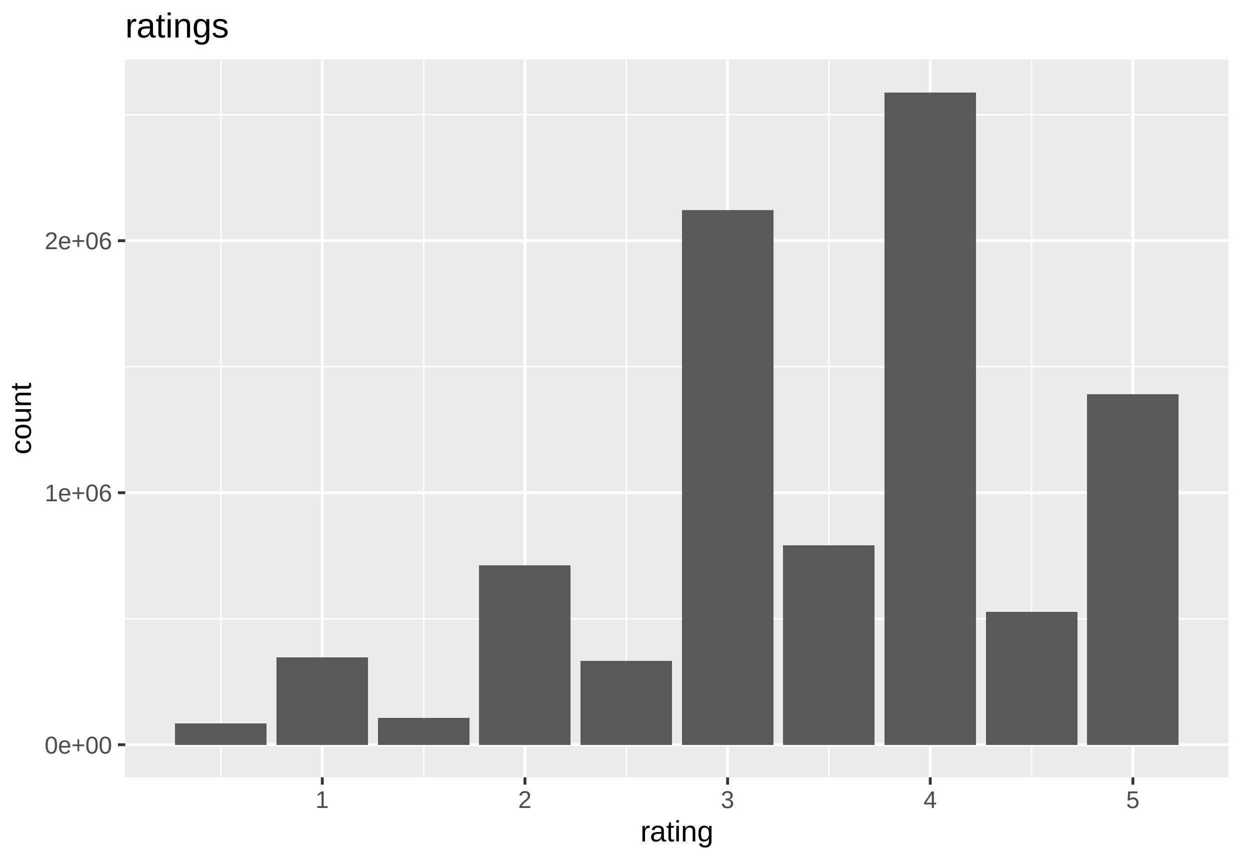
69878 user have been participated in this data, and 10677 movies have been reviewed. check each column by plotting



1. users



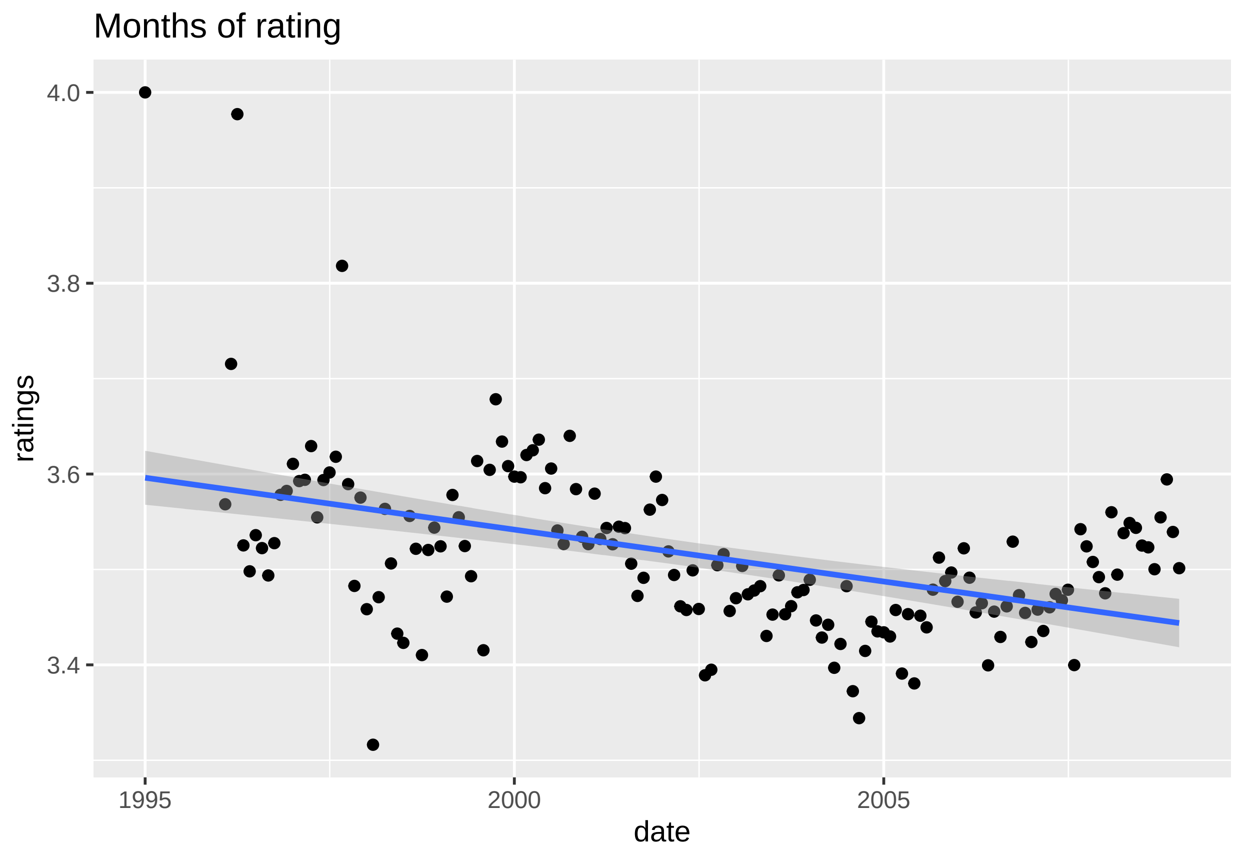
2.Movies



3.Ratings

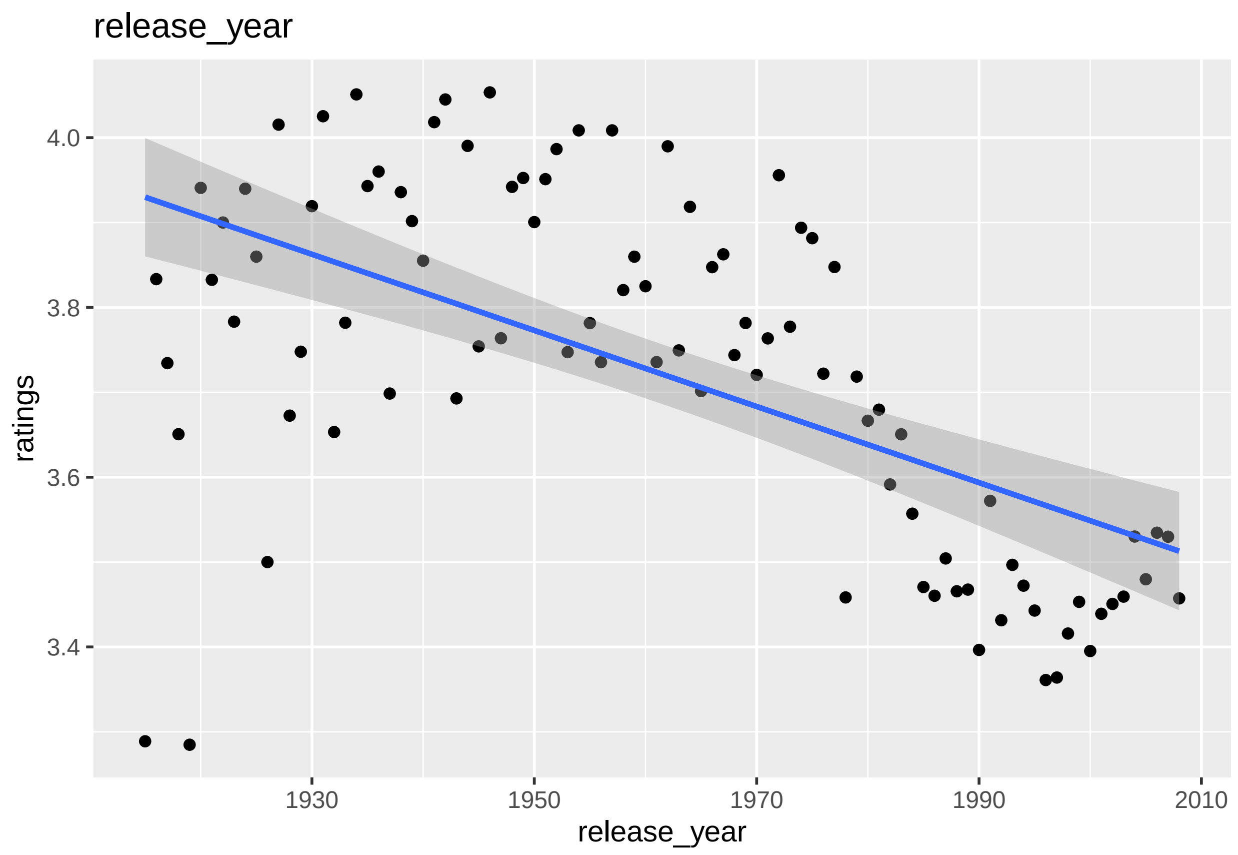
4. timestamp these figures need to be changed by lubridate and covert “timestamp” to POSIXct data, and find months of ratings.

```
## ‘geom_smooth()‘ using formula ’y ~ x’
```



5. title at a glance, they seem not to have significance, but they have release year. to pick up the release years.

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



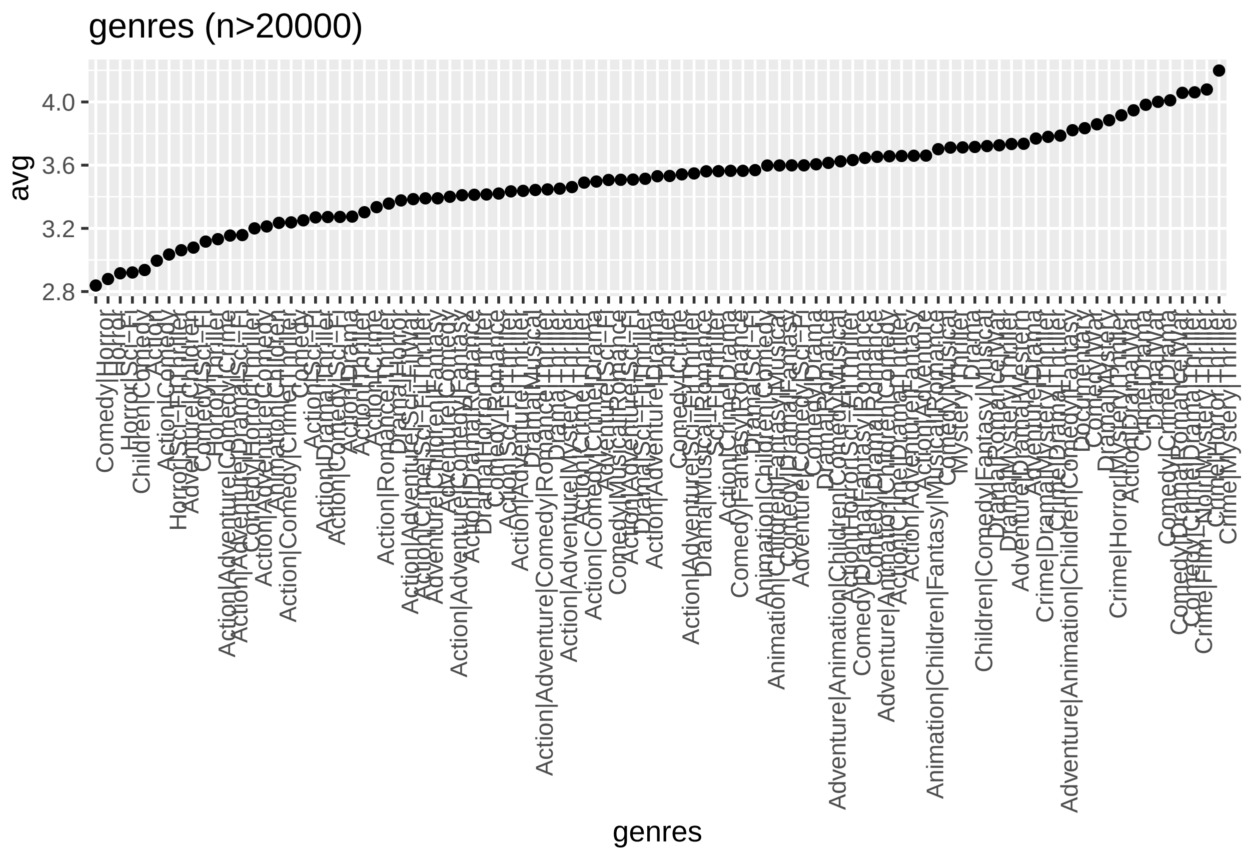
compared to months of ratings, it seems to have significance in predicting rating. to reduce data size, I remove the columns, date and timestamp.

```
edx <- edx %>% select(-timestamp, -date)
```

6. genres How many genres does the dataset have?

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

```
## # A tibble: 797 x 2
##    genres                                               ratings
##  * <chr>                                                  <dbl>
##  1 (no genres listed)                                      3.64
##  2 Action                                                  2.94
##  3 Action|Adventure                                        3.66
##  4 Action|Adventure|Animation|Children|Comedy              3.96
##  5 Action|Adventure|Animation|Children|Comedy|Fantasy      2.99
##  6 Action|Adventure|Animation|Children|Comedy|IMAX         3.30
##  7 Action|Adventure|Animation|Children|Comedy|Sci-Fi       3.08
##  8 Action|Adventure|Animation|Children|Fantasy             2.70
##  9 Action|Adventure|Animation|Children|Sci-Fi              2.97
## 10 Action|Adventure|Animation|Comedy|Drama                 3.51
## # ... with 787 more rows
```



Preparing Model building and RMSE calculation

making test and training data

```r
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

To make sure we don't include users and movies in the test set that do not appear in the training set, we remove these entries using the semi_join function

```r
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

RMSE function

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

$$y_{u,i} : the\ rating\ for\ movie\ i\ by\ user\ u \hat{y}_{u,i} : the\ prediction N : the\ number of\ user/movie\ combinations$$

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Model building

```
mu <- mean(train_set$rating)
naive_rmse <- RMSE(test_set$rating, mu)
naive_rmse
```

1 Naive model assuming the same rating for all movies

```
## [1] 1.060054
```

```
#[1] 1.060054
```

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

movie_avg <- train_set %>%
  group_by(movieId)%>%
  summarize(b_i =mean(rating -mu))

movie_effect_pred <- mu + test_set %>%
  left_join(movie_avg, by="movieId") %>%
  pull(b_i)

movie_effect_rmse <- RMSE(test_set$rating, movie_effect_pred)
movie_effect_rmse
```

2 movie effects

```
## [1] 0.9429615
```

```
#[1] 0.9429615
```

```
user_avg <- train_set %>%
  left_join(movie_avg, by= "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating- mu -b_i))

movie_user_effect_pred <- test_set %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  mutate(pred = mu + b_i + b_u)%>%
  pull(pred)

movie_user_effect_rmse <- RMSE(test_set$rating, movie_user_effect_pred)
movie_user_effect_rmse
```

3 movie and user effects

```
## [1] 0.8646844
```

```
#[1] 0.8646844
```

```


g_avg <- train_set %>%
  left_join(movie_avg, by= "movieId")%>%
  left_join(user_avg, by="userId") %>%
  group_by(genres) %>%
  summarize(g = mean(rating- mu -b_i -b_u))

movie_user_genre_effect_pred <- test_set %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(g_avg, by="genres") %>%
  mutate(pred = mu + b_i + b_u + g)%>%
  pull(pred)

movie_user_genre_effect_rmse <- RMSE(test_set$rating,
                                     movie_user_genre_effect_pred)
movie_user_genre_effect_rmse
```

4 genre effects

```
## [1] 0.8643242
```

```
#[1] 0.8643242
```

```


y_avg <- train_set %>%
  left_join(movie_avg, by= "movieId")%>%
  left_join(user_avg, by="userId") %>%
  left_join(g_avg, by="genres") %>%
  group_by(release_year) %>%
  summarize(y = mean(rating- mu -b_i -b_u -g))

movie_user_genre_year_effect_pred <- test_set %>%
  left_join(movie_avg, by="movieId") %>%
  left_join(user_avg, by="userId") %>%
  left_join(g_avg, by="genres") %>%
  left_join(y_avg, by="release_year")%>%
  mutate(pred = mu + b_i + b_u + g +y)%>%
  pull(pred)

movie_user_genre_year_effect_rmse <- RMSE(test_set$rating,
                                          movie_user_genre_year_effect_pred)
movie_user_genre_year_effect_rmse
```

5 release year effects

```
## [1] 0.8641262
```

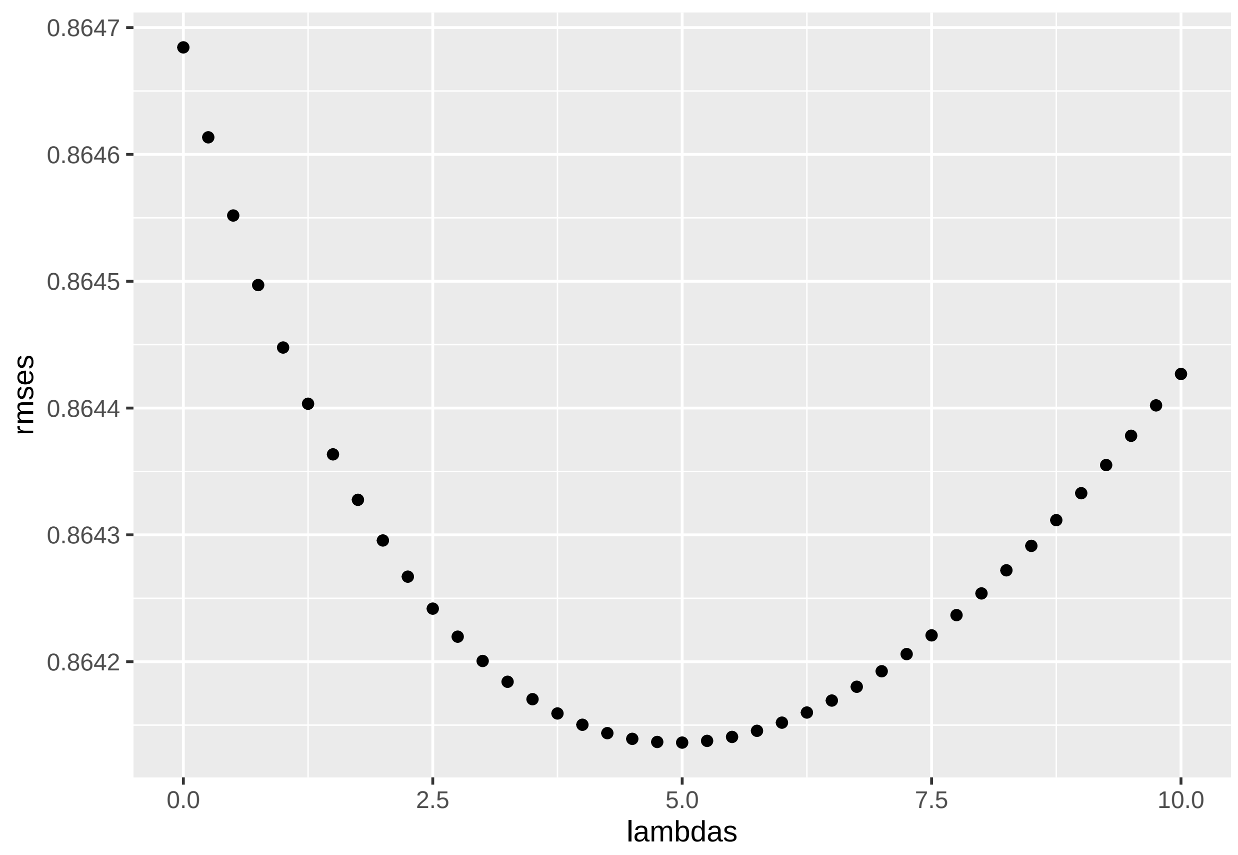
```
#[1] 0.8641262
```

6 regularization (movie, user)

introducing lambda

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

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

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

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

  return(RMSE(predicted_ratings, test_set$rating))

})
```

```
qplot(lambdas,rmses)
```



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

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

```
#[1] 5
```

```
reg_movie_user_rmse <- rmses
reg_movie_user_rmse[5]
```

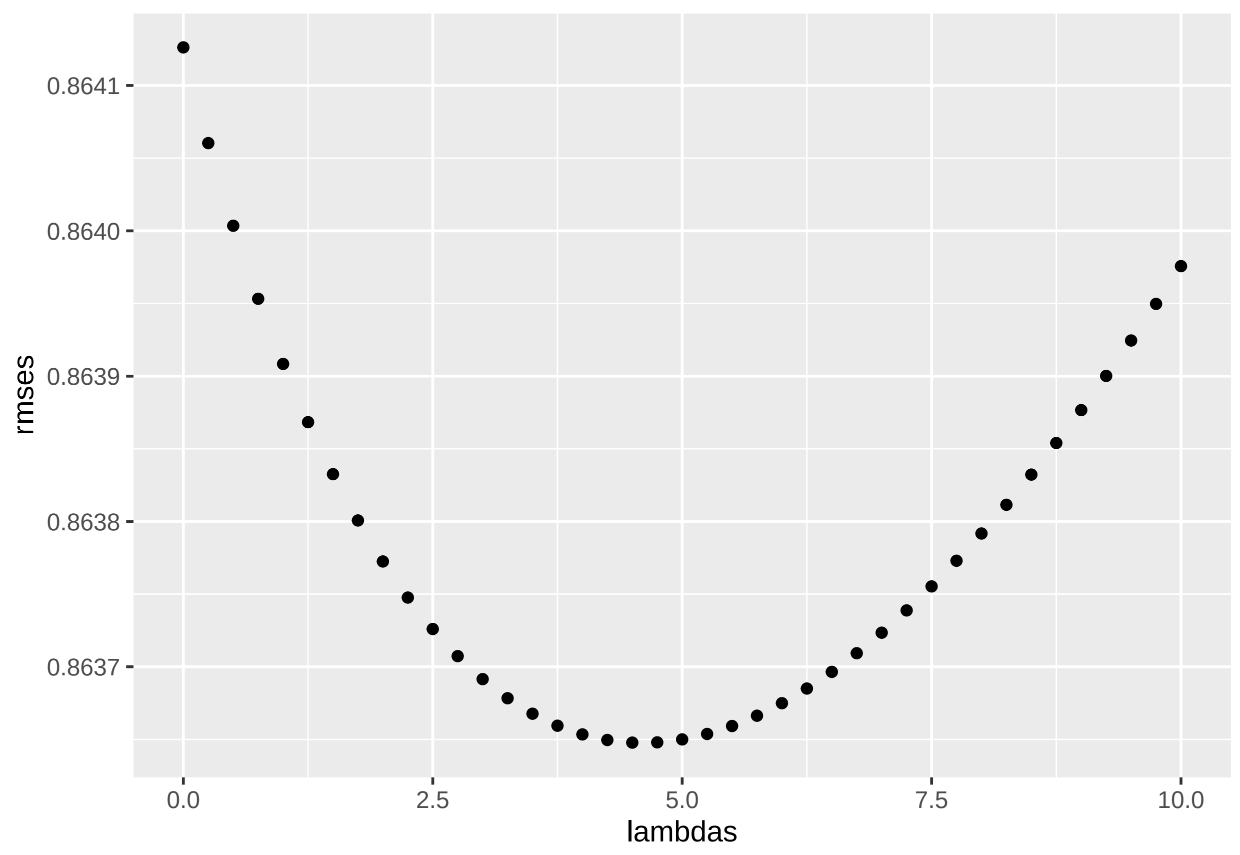
```
## [1] 0.8644477
```

```
#[1] 0.8644477
```

7 regularization (movie, user,genres, release_year) calculating lambda

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

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

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

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

    y_avg <- train_set %>%
    left_join(reg_movie_avg, by="movieId") %>%
    left_join(reg_user_avg, by="userId") %>%
    left_join(g_avg, by="genres") %>%
    group_by(release_year) %>%
    summarize(y = sum(rating - b_i -b_u -g - mu)/(n()+l))

  predicted_ratings <-test_set %>%
    left_join(reg_movie_avg, by = "movieId") %>%
    left_join(reg_user_avg, by = "userId") %>%
    left_join(g_avg, by="genres") %>%
    left_join(y_avg, by="release_year")%>%
    mutate(pred = mu + b_i + b_u + g + y) %>%
    pull(pred)

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

```
qplot(lambdas,rmses)
```



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

```
## [1] 4.5
```

```
#[1] 4.5

min(rmses)
```

```
## [1] 0.8636478
```

```
#[1] 0.8636478
```

```


rmse_results <- bind_rows(rmse_results,
                          tibble(method=
                                "Reg Movie User Genre Release Year Effects Model",
                                MSE =rmses[4.5]))%>% as.data.frame()
```

8 model summary

method	RMSE
Rating Average	1.0600537
Movie Effects Model	0.9429615
Movie User Effects Model	0.8646844
Movie User Genre Effects Model	0.8643242
Movie User Genre Release Year Effects Model	0.8641262
Reg Movie User Effects Model	0.8644477
Reg Movie User Genre Release Year Effects Model	0.8639532

final model “Reg Movie User Genre Release Year Effects Model” is proved to extract the least RMSE Value.

Evaluation

validation calculating lambda add and cut columns in the process of training

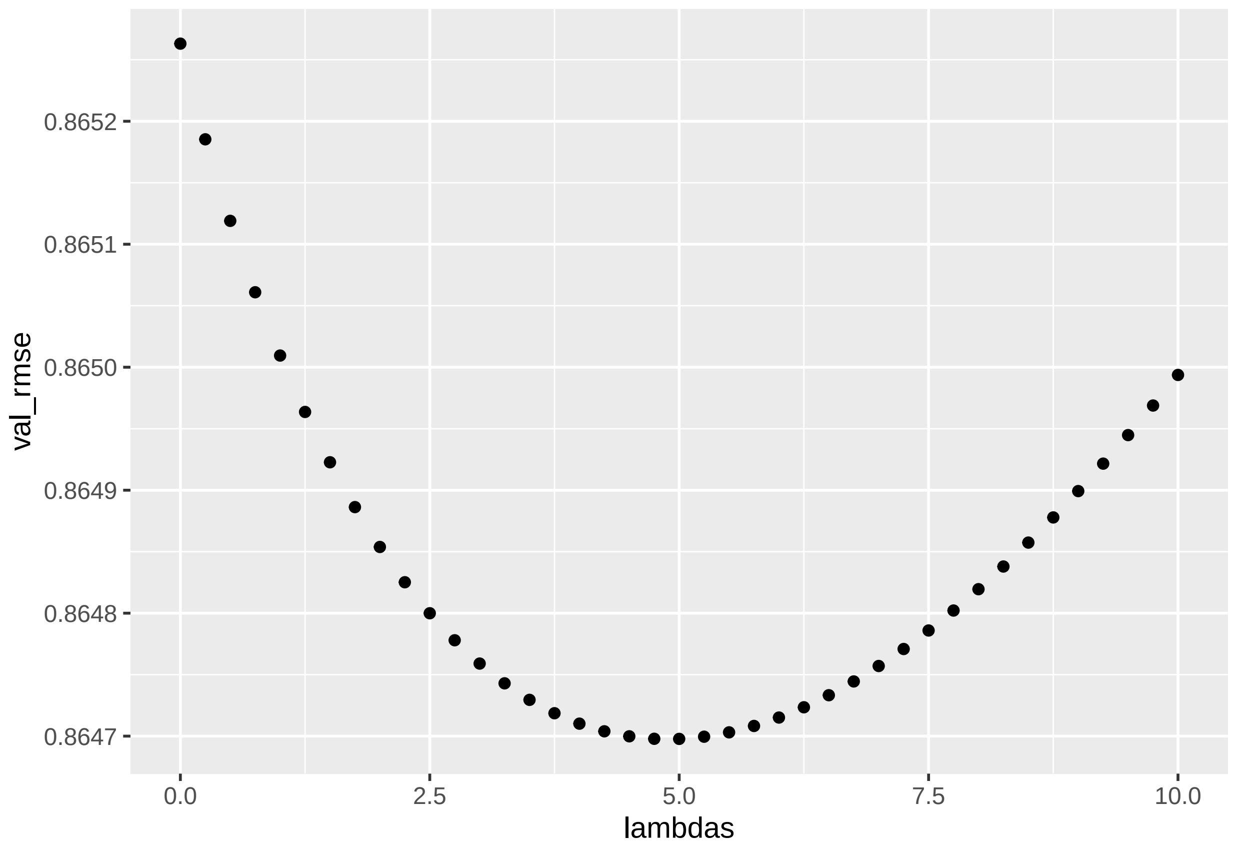
```
validation <- validation %>% mutate(release_year =
                                      as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% select(-timestamp)
```

define lambdas as a variable

```
lambdas <- seq(0, 10, 0.25)

#calculate rmse using "Reg Movie User Genre Release Year Effects Model"

val_rmse <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

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

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

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

  y_avg <- train_set %>%
    left_join(reg_movie_avg, by="movieId") %>%
    left_join(reg_user_avg, by="userId") %>%
    left_join(g_avg, by="genres") %>%
    group_by(release_year) %>%
    summarize(y = sum(rating - b_i -b_u -g - mu)/(n()+l))
  val_pred <-validation %>%
    left_join(reg_movie_avg, by = "movieId") %>%
    left_join(reg_user_avg, by = "userId") %>%
    left_join(g_avg, by="genres") %>%
    left_join(y_avg, by="release_year")%>%
    mutate(pred = mu + b_i + b_u + g + y) %>%
    pull(pred)

    #replace NA with mu(the average rating in the training)
    val_pred <-replace(val_pred, is.na(val_pred),mu)

    return(RMSE(val_pred,validation$rating))
})
```

plot lambdas and rmses

```
qplot(lambdas,val_rmse)
```



```
#find lambda and rmse which indicate the least value
lambda <- lambdas[which.min(val_rmse)]
lambda
```

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

```
#[1] 5

min(val_rmse)
```

```
## [1] 0.8646978
```

```
#[1] 0.8646954
```

Conclusion